User's Manual

RX ファミリ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ

コンパイラパッケージ V.1.01　　ユーザーズマニュアル

誤記に関するお詫び：
本資料の187 ページに誤記があり、訂正いたしました。

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2011.02

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

はじめに

本マニュアルは、**「RX ファミリ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ」**の使用方法を述べたものです。

本製品は C 言語、C++言語およびアセンブリ言語で記述したソースプログラムを、RX ファミリ用オブジェクトプログラムおよびロードモジュールに変換するソフトウェアシステムです。

ご使用になる前に、本マニュアルを良く読んで理解してください。

■表記上の注意事項

本マニュアルの説明の中で用いられる記号は、次の意味を示しています。

〈 〉	この記号で囲まれた内容を指定することを示します。
［ ］	省略してもよい項目を示します。
． ． ．	直前の項目を 1 回以上指定することを示します。
△	1 個以上の空白を示します。
｜	|で区切られた項目を選択できることを示します。

本マニュアルは IBM PC*1 互換機およびその互換機上で動作する Microsoft® Windows® XP、 Windows® Vista または Windows 7®*2 に対応するように書かれています。

【注】 *1 IBM PC **は、米国** International Business Machines Corporation **の登録商標です。**

*2 Microsoft®, Windows®**は、米国** Microsoft Corporation **の米国及びその他の国における登録商標または商標です。**

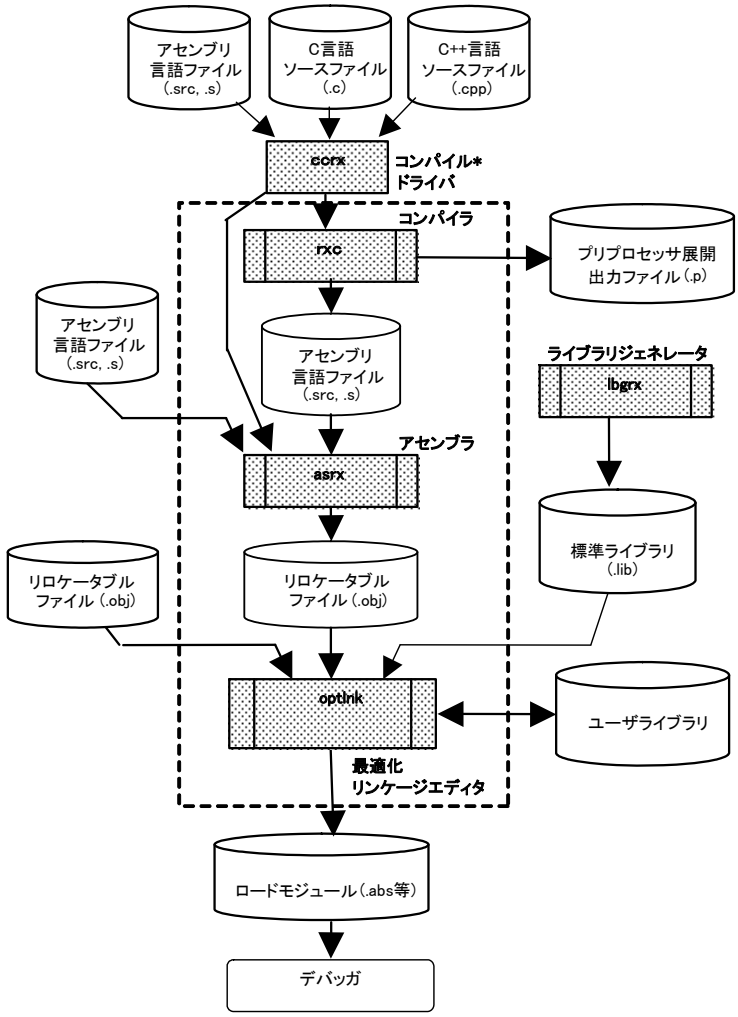
※ **その他、本マニュアルの文中に使われている会社名および製品名、システム名などは各社の登録商標または商標です。**

目次

1. 概要

1.1 コンパイラの構成

RX ファミリ用 C/C++コンパイラの構成を示します。



【注】 * コンパイルドライバは、入力ファイルがアセンブリ言語ファイル(.src または .s)の場合、
アセンブラを呼び出し、それ以外のファイルの場合コンパイラを呼び出します。

図 1.1 コンパイラの構成

1.1.1 コンパイルドライバへの入力

ASCII 文字と、シフト JIS 文字（オプションにより、EUC、Latin1 または UTF-8 に変更可能）からなる、ANSI 準拠 C 言語(C89/C99(可変長配列は除く))、ANSI 準拠 C++言語、EC++言語で記述されたソースファイル(.c, .cpp)、およびアセンブリ言語ファイル(.src, .s)です。

1.1.2 コンパイルドライバの出力

プリプロセッサ展開出力ファイル(.p)、アセンブリ言語ファイル(.src, .s)、リロケータブルファイル、ロードモジュールを出力します。

1.1.3 ccrx

ccrx は、コンパイルドライバの実行ファイルです。

ccrx は、オプションの指定によりコンパイルからリンクまでの処理を一括して行うことができます。また、ccrx の起動オプション"-asmcmd "、" -lnkcmd "、" -asmopt "、" -lnkopt "に続けてアセンブラ asrx、最適化リンケージエディタ optlnk のオプションを指定することができます。

1.1.4 asrx

asrx は、アセンブラの実行ファイルです。

アセンブリ言語ファイル(.src, .s)を、リロケータブルファイルに変換します。

1.1.5 optlnk

optlnk は、最適化リンケージエディタの実行ファイルです。

複数のリロケータブルファイル(.obj)およびライブラリファイル(.lib)を、ロードモジュールファイル(.abs 等)またはライブラリファイル(.lib)に変換します。

1.1.6 lbgrx

lbgrx は、ライブラリジェネレータの実行ファイルです。

ユーザが指定したオプションに応じた標準ライブラリファイル(.lib)を生成します。

1.2 オプション指定規則

以下に本コンパイラパッケージで利用できる起動コマンドを説明します。

なお、これらのコマンドを利用する前に、6 章「環境変数」を参照のうえ、必要な環境変数が設定されているかを確認してください。

1.2.1 コンパイラ(ccrx)

ccrx はコンパイルドライバの起動コマンドです。

本コマンド起動により、コンパイル、アセンブル、リンクを行うことができます。

入力ファイルの拡張子が「.s」「.src」「.S」「.SRC」のいずれかである場合、コンパイラはそのファイルをアセンブリ言語ファイル(.src, .s)と解釈して、アセンブラを起動します。

これら以外の拡張子のファイルは、C/C++言語ソースファイル(.c, .cpp)としてコンパイルします。

【コマンド記述形式】

ccrx [△<**オプション**> …][△<**ファイル名**>[△<**オプション**> …] …]
　<**オプション**>：－<**オプション**>[=<**サブオプション**>[=<**サブオプション**>]][,…]

1.2.2 アセンブラ(asrx)

asrx は、アセンブラの起動コマンドです。

【コマンド記述形式】

asrx [△<**オプション**> …][△<**ファイル名**>[△<**オプション**> …] …]
　<**オプション**>：－<**オプション**>[=<**サブオプション**>][,…]

1.2.3 最適化リンケージエディタ(optlnk)

optlnk は、最適化リンケージエディタの起動コマンドです。

リンク処理だけではなく、以下に挙げる機能も含んでいます。

- リロケータブルファイル結合時の最適化
- ライブラリファイルの作成や編集
- モトローラS形式ファイル、インテルHEX形式ファイル、およびバイナリファイルへのコンバート

【コマンド記述形式】

optlnk [△<**オプション**> …][△<**ファイル名**>[△<**オプション**> …] …]
　<**オプション**>：－<**オプション**>[=<**サブオプション**>][,…]

1.2.4 ライブラリジェネレータ(lbgrx)

lbgrx は、ライブラリジェネレータの起動コマンドです。

【コマンド記述形式】

```
lbgrx [△<オプション> … ]
  <オプション>：－<オプション>[=<サブオプション>][,…]
```

1.3 コマンドの記述例

1.3.1 コンパイル、アセンブル、リンクを 1 コマンドで実施する場合

以下の手順全てを 1 コマンドで実施します。

- C/C++言語ソースファイル(tp1.cとtp2.c)をccrxでコンパイルする
- コンパイル後、asrxでアセンブルする
- アセンブル後、optlnkでリンクして、アブソリュートファイル(tp.abs)を作成する

【コマンド記述】

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.c tp2.c
```

【備考】

- outputオプションの出力形式指定を"-output=sty"に変えると、リンク後のファイルをモトローラS形式ファイルとして生成します。
- アブソリュートファイル生成過程で生じる中間ファイル(アセンブリ言語ファイルや、リロケータブルファイル)は残りません。生成されるファイルは、outputオプションで指示した形式のファイルのみです。
- ccrxに対して、アセンブラ、最適化リンケージエディタにのみ有効なアセンブルオプションやリンクオプションを指示したい場合には、-asmcmdオプション、-lnkcmdオプション、-asmoptオプション、-lnkoptオプションを使用して指示してください。
- リンク対象のオブジェクトは、0番地から配置します。セクションの並び順は保証されません。配置アドレスやセクションの配置順序を指示したい場合には、-lnkcmdオプション、-lnkoptオプションを使用して最適化リンケージエディタへオプション指示してください。

1.3.2 コンパイルとアセンブルを 1 コマンドで実施する場合

以下の手順を 1 コマンドで実施し、別コマンドでリンカを起動して、tp.abs を作成します。

- C/C++言語ソースファイル(tp1.cとtp2.c)をccrxでコンパイルする
- コンパイル後、asrxでアセンブルして、リロケータブルファイル(tp1.obj, tp2.obj)を作成する

【コマンド記述】

```
ccrx -cpu=rx600 -output=obj tp1.c tp2.c
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【備考】

- ccrxに対して"-output=obj"オプションを指示すると、ccrxはリロケータブルファイルを生成します。
- リロケータブルファイル名を変更する場合は、ccrxへC/C++言語ソースファイルをひとつずつ入力する必要があります。
- optlnkのformオプションを、"-form=sty"に変えると、リンク後のファイルをモトローラS形式ファイルとして生成します。

1.3.3 コンパイル、アセンブル、リンクを各々別コマンドで実施する場合

以下の個々の手順を、それぞれ 1 コマンドで実施します。

- C/C++言語ソースファイル(tp1.cとtp2.c)をccrxでコンパイルして、アセンブリ言語ファイル (tp1.src, tp2.src)を作成する
- アセンブリ言語ファイル(tp1.src, tp2.src)をasrxでアセンブルして、リロケータブルファイル(tp1.obj, tp2.obj)を生成する
- リロケータブルファイル(tp1.obj, tp2.obj)をoptlnkでリンクして、アブソリュートファイル(tp.abs)を作成する

【コマンド記述】

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【備考】

- ccrxに対して"-output=src"オプションを指示すると、ccrxはアセンブリ言語ファイルを生成します。

1.3.4 アセンブルとリンクを 1 コマンドで実施する場合

以下の手順全てを 1 コマンドで実施します。

- アセンブリ言語ファイル(tp1.src, tp2.src)をasrxでアセンブルする
- アセンブル後、optlnkでリンクして、アブソリュートファイル(tp.abs)を作成する

【コマンド記述】

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.src tp2.src
```

【備考】

- リンク対象のオブジェクトは、0番地から配置します。セクションの並び順は保証されません。配置アドレスやセクションの配置順序を指示したい場合には、-lnkcmdオプション、-lnkoptオプションを使用して最適化リンケージエディタへオプション指示してください。

1.3.5 アセンブルとリンクを別コマンドで実施する場合

以下の個々の手順を、それぞれ 1 コマンドで実施します。

- アセンブリ言語ファイル(tp1.src, tp2.src)をasrxでアセンブルして、リロケータブルファイル(tp1.obj, tp2.obj)を生成する
- リロケータブルファイル(tp1.obj, tp2.obj)をoptlnkでリンクして、アブソリュートファイル(tp.abs)を作成する

【コマンド記述 1】

```
ccrx -cpu=rx600 -output=obj tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【コマンド記述 2】

```
asrx -cpu=rx600 tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

2. C/C++コンパイラオプション

2.1 ソースオプション

表 2.1 ソースオプション一覧

No.	オプション	ダイアログメニュー	内容
1	lang = { c | cpp | ecpp | c99 }	コンパイラ <ソース> [オプション項目:] [ソースファイル] [言語 :] [C :] [C(C89)] [C99] [C++ :] [C++] [EC++]	C(C89)言語ソースファイルとしてコンパイル C++言語ソースファイルとしてコンパイル EC++言語ソースファイルとしてコンパイル C(C99)言語ソースファイルとしてコンパイル
2	include = <パス名>[,…]	コンパイラ <ソース> [オプション項目 :] [インクルードファイル ディレクトリ]	インクルードファイルの取り込み先パス名を指定
3	preinclude = <ファイル名>[,…]	コンパイラ <ソース> [オプション項目 :] [デフォルトインクルードファイル]	指定したファイルをコンパイル単位の先頭にインクルード
4	define = <sub>[,…] <sub>:<マクロ名>[=<文字列>]	コンパイラ <ソース> [オプション項目 :] [マクロ定義]	<文字列>を<マクロ名>として定義
5	undefine = <sub>[,…] <sub>:<マクロ名>	コンパイラ <ソース> [オプション項目 :] [マクロ定義の無効化]	<マクロ名>のプリデファインドマクロを無効化
6	message nomessage[=<エラー番号> [-<エラー番号>][,…]]	コンパイラ <ソース> [オプション項目 :] [インフォメーションメッセージ] [インフォメーションレベルメッセージ抑止]	インフォメーションメッセージ出力有効 インフォメーションメッセージ出力無効

No.	オプション	ダイアログメニュー	内容
7	change_message =<sub>[,…] <sub>:<level> [=<n>[-m],[…]] <level>:{Information | warning | error }	コンパイラ <その他> [ユーザ指定オプション :]	コンパイラ出力メッセージのレベル変更
8	file_inline_path=<パス名>[,…]	コンパイラ <ソース> [オプション項目 :] [ファイル間インライン展開ディレクトリ]	ファイル間インライン展開ファイル取り込み先パス名を指定
9	comment = { nest | nonest }	コンパイラ <ソース> [オプション項目:] [ソースファイル] [コメントの(/* */)のネストを許す]	コメント(/* */)のネストを許す コメント(/* */)のネストを許さない
10	check={ nc | ch38 | shc }	コンパイラ <ソース> [オプション項目:] [ソースファイル] [互換性チェック :] [なし] [NC コンパイラ] [H8 コンパイラ] [SH コンパイラ]	既存プログラムとの互換性をチェックする

lang

書　式　lang= { c | cpp | ecpp | c99 }

説　明　ソースファイルの言語を指定します。

lang=c オプション指定時は、C (C89)言語ソースファイルとしてコンパイルします。

lang=cpp オプション指定時は、C++言語ソースファイルとしてコンパイルします。

lang=ecpp オプション指定時は、Embedded C++言語ソースファイルとしてコンパイルします。

lang=c99 オプション指定時は、C (C99)言語ソースファイルとしてコンパイルします。

本オプションを省略した場合は、拡張子が cpp、cc、cp のときには C++言語ソースファイルとしてコンパイルし、それ以外の場合は C(C89)言語ソースファイルとしてコンパイルします。ただし、拡張子が src、s の場合は本オプション指定に関わらずアセンブリ言語ファイルとして扱います。

備　考　Embedded C++言語仕様では、catch、const_cast、dynamic_cast、explicit、mutable、namespace、reinterpret_cast、static_cast、template、throw、try、typeid、typename、using、多重継承、仮想基底クラスをサポートしていません。これらを記述した場合、エラーメッセージを出力します。EC++ライブラリを使用する場合は、必ず lang=ecpp オプションを指定してください。

include

書　式　include = <パス名>[,…]

説　明　インクルードファイルの存在するパス名を指定します。

パス名が複数ある場合にはカンマ(,)で区切って指定することができます。

システムインクルードファイルの検索は、include オプション指定フォルダ、環境変数 INC_RX 指定フォルダ、環境変数 BIN_RX 指定フォルダの順序で行います。

ユーザインクルードファイルの検索は、コンパイル対象ソースファイルのあるフォルダ、include オプション指定フォルダ、環境変数 INC_RX 指定フォルダ、環境変数 BIN_RX 指定フォルダの順序で行います。

備　考　本オプションを複数回指定した場合、指定した全てのパス名が有効になります。

preinclude

書　式　　preinclude = <ファイル名>[,…]

説　明　　指定したファイルの内容をコンパイル単位の先頭に取り込みます。ファイル名が複数ある場合にはカンマ(,)で区切って指定することができます。
include オプション指定フォルダが複数ある場合、左に指定したものから順に検索を行います。

備　考　　本オプションを複数回指定した場合、指定した全てのファイルが取り込み対象となります。

define

書　式　　define = <sub>[,…]

<sub>：<マクロ名> [= <文字列>]

説　明　　ソースファイル内で記述する#define と同等の効果を得ます。

<マクロ名>=<文字列>と記述することで<文字列>をマクロ名として定義できます。

サブオプションに<マクロ名>を単独で指定した場合は、そのマクロ名が定義されたものと仮定します。<文字列>には、名前または整定数を記述することができます。

備　考　　本オプションで指定したマクロ名がソース中で#define により既に定義されている場合、#define を優先します。

本オプションを複数回指定した場合、指定した全てのマクロ名が有効となります。

undefine

書　式　　undefine = <sub >[,…]

<sub>：<マクロ名>

説　明　　＜マクロ名＞のプリデファインドマクロを無効化します。
マクロ名が複数ある場合にはカンマ（,）で区切って指定することができます。

備　考　　指定可能なプリデファインドマクロについては、**「6.2　プリデファインドマクロ」**を参照ください。

本オプションを複数回指定した場合、指定した全てのマクロ名が未定義となります。

message, nomessage

書　式　　message

nomessage [= <エラー番号> [- <エラー番号>][,…]

説　明　　インフォメーションレベルメッセージの出力有無を指定します。

message オプションを指定した場合、インフォメーションレベルメッセージを出力します。

nomessage オプションを指定した場合、インフォメーションレベルメッセージの出力を抑止します。また、サブオプションでエラー番号を指定すると、指定したインフォメーションレベルメッセージの出力だけを抑止します。エラー番号が複数ある場合にはカンマ (,) で区切って指定することができます。
<エラー番号>-<エラー番号>のようにハイフン(-)で抑止するエラー番号の範囲を指定することもできます。

本オプションの省略時解釈は、nomessage です。

備　考　　本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。最適化リンケージエディタのメッセージについては、lnkcmd オプションにより、最適化リンケージエディタの message オプションおよび nomessage オプションを指定することで出力制御が可能です。

nomessage オプションを複数回指定した場合、指定した全てのエラー番号について抑止します。

change_message

書　式　　change_message = <sub>[,…]

<sub> : <エラーレベル>[=<エラー番号>[- <エラー番号>][,…]]

<エラーレベル> : { information | warning | error }

説　明　　インフォメーション、ウォーニングのメッセージレベルを変更します。

エラー番号が複数ある場合にはカンマ（,）で区切って指定することができます。

例　　change_message=information=エラー番号

ウォーニングレベルの指定エラー番号のみインフォメーションレベルに変更します。

change_message=warning=エラー番号

インフォメーションレベルの指定エラー番号のみウォーニングレベルに変更します。

change_message=error=エラー番号

インフォメーション、ウォーニングレベルの指定エラー番号のみエラーレベルに変更します。

change_message=information

全てのウォーニングメッセージをインフォメーションレベルに変更します。

change_message=warning

全てのインフォメーションメッセージをウォーニングレベルに変更します。

change_message=error

全てのインフォメーション、ウォーニングメッセージをエラーレベルに変更します。

備　考　　インフォメーションレベルに変更したメッセージについては、nomessage オプション指定により出力を抑止できます。

本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。最適化リンケージエディタのメッセージについては、lnkcmd オプションにより、最適化リンケージエディタの message オプションおよび nomessage オプションを指定することで出力制御が可能です。

本オプションを複数回指定した場合、指定した全てのエラー番号について有効になります。

エラーメッセージは本オプションのレベル制御対象外です。

file_inline_path

書　式　　file_inline_path= <パス名>[,…]

説　明　　ファイル間インライン展開対象となるファイルの存在するパス名を指定します。

パス名が複数ある場合にはカンマ(,)で区切って指定することができます。ファイル間インライン展開対象ファイルの検索は、file_inline_path オプション指定フォルダ、カレントフォルダの順序で行います。

備　考　　本オプションを複数回指定した場合、指定した全てのパス名が有効となります。

comment

書　式　　comment = { nest | nonest }

説　明　　comment=nest を指定した場合、ネストしたコメントの記述を可能にします。

comment=nonest を指定した場合、ネストしたコメントを記述するとエラーになります。

本オプションの省略時解釈は、comment=nonest です。

例　　/* This is an example of /* nested */ comment */
↑
[1]

comment=nest を指定した場合は全てコメントと解釈しますが、comment=nonest を指定した場合は[1]でコメントが終わっていると解釈します。

check

書　式　　check = { nc | ch38 | shc}

説　明　　R8C, M16C ファミリ用 C コンパイラ、H8, H8S, H8SX ファミリ用 C/C++コンパイラおよび SuperH ファミリ用 C/C++コンパイラ向けにコーディングした C/C++言語ソースファイルを本コンパイラへ流用する際、互換性に影響するオプション指定、ソース記述をチェックすることができます。

check=nc では、R8C,M16C ファミリ用 C コンパイラとの互換性をチェックします。
チェックされるオプションや型には、次のようなものがあります。

- オプション：signed_char, signed_bitfield, bit_order=left, endian=big, dbl_size=4
- inline、enum型、#pragma BITADDRESS、#pragma ROM、
  #pragma PARAMETER、__asm()
- -int_to_shortの指定がないときに、signed short範囲外の定数をint,signed int型へ代入、あるいはunsigned short範囲外の定数をint型またはunsigned int型へ代入
- signed short, unsigned short共範囲外の定数をlong, long long型へ代入
- signed short範囲外の定数とint, short, char型(char型は符号付き除く)との比較式

check=ch38 では、H8,H8S,H8S ファミリ用 C/C++コンパイラとの互換性をチェックします。
チェックされるオプションや型には、次のようなものがあります。

- オプション：unsigned_char, unsigned_bitfield, bit_order=right, endian=little,
  dbl_size=4
- __asm、#pragma unpack
- signed longの最大値より大きな定数との比較式
- -int_to_shortの指定がないときに、signed short範囲外の定数をint,signed int型へ代入、あるいはunsigned short範囲外の定数をint型またはunsigned int型へ代入
- signed short, unsigned short共範囲外の定数をlong, long long型へ代入
- signed short範囲外の定数とint, short, char型(char型は符号付き除く)との比較式

check=shc では、SuperH ファミリ用 C/C++コンパイラとの互換性をチェックします。
チェックされるオプションや型には、次のようなものがあります。

- オプション：unsigned_char, unsigned_bitfield, bit_order=right, endian=little,
  dbl_size=4 , round=nearest
- #pragma unpack
- volatile修飾した変数

表示された項目により、それぞれ次の項目を確認ください。

オプション：言語仕様で規定されていない実装依存の内容がコンパイラ間で異なっています。メッセージで出力されたオプションの選択を確認してください。

拡張仕様：プログラムの動作に影響を及ぼす可能性がある拡張仕様です。メッセージで出力された拡張仕様の記述を確認してください。

備　考　dbl_size=4 が有効な時に、R8C, M16C ファミリ用 C コンパイラ、H8, H8S, H8SX ファミリ用 C/C++コンパイラおよび SuperH ファミリ用 C/C++コンパイラと浮動小数点関連の変換/ライブラリの計算結果が異なる場合があります。dbl_size=4 は、本コンパイラでは、double 型および long double 型を 32 ビットにしますが、各種 R8C, M16C ファミリ用 C コンパイラ(fdouble_32)、H8, H8S, H8SX ファミリ用 C/C++コンパイラ(double=float) および SuperH ファミリ用 C/C++コンパイラ(double=float)では、double 型のみ 32 ビットにします。

unsigned int 型と long 型をオペランドとする二項演算(加減乗除や比較など)に対する結果が、SuperH ファミリ用 C/C++コンパイラと異なる場合があります。本コンパイラではオペランドを unsigned long 型に変換してから演算しますが、SuperH ファミリ用 C/C++コンパイラ(ただし、strict_ansi を指定しないとき)では、signed long long 型に変換してから演算します。

volatile 修飾した変数に対し、読み出しや書き込みのサイズが、SuperH ファミリ用 C/C++コンパイラと異なる場合があります。volatile 修飾したビットフィールドは、本コンパイラでは宣言型より小さなサイズでアクセスすることがありますが、SuperH ファミリ用 C/C++コンパイラでは宣言型のサイズ通りにアクセスします。

構造体およびビットフィールドメンバの割り付けについては、本オプションでメッセージを出力しません。割り付けを意識した宣言をしている場合には、**「9.1.2　データの内部表現」**を参照してください。

R8C, M16C ファミリ用 C コンパイラ(fextend_to_int を指定しない)では、条件式で汎整数拡張を行わずに評価したコードを生成するので、本コンパイラの生成コードと動作が異なる場合があります。

2.2 オブジェクトオプション

表 2.2 オブジェクトオプション一覧

No.	オプション	ダイアログメニュー	内容
1	output = {prep | src | obj | abs | hex | sty} [= ファイル名]	コンパイラ <オブジェクト> [出力ファイル形式 :] [機械語プログラム] [アセンブリプログラム] [プリプロセッサ展開プログラム]	出力ファイル形式を指定 プリプロセッサ展開後のソースファイルを出力 アセンブリ言語ファイルを出力 リロケータブルファイルを出力 アブソリュートファイルを出力 インテル HEX 形式ファイルを出力 モトローラ S 形式ファイルを出力
2	noline	コンパイラ <オブジェクト> [出力ファイル形式 :] [プリプロセッサ展開プログラム (#line 出力抑止)]	プリプロセッサ展開時に#line の出力を抑止
3	debug nodebug	コンパイラ <オブジェクト> [デバッグ情報出力]	デバッグ情報出力あり デバッグ情報出力なし
4	section = <sub>[,…] <sub>: {P = <セクション名> | C = <セクション名> | D = <セクション名> | B = <セクション名> | L = <セクション名> | W = <セクション名>}	コンパイラ <オブジェクト> [詳細…] [セクション :] [プログラム領域 (P)] [定数領域 (C)] [初期化データ領域(D)] [未初期化データ領域(B)] [リテラル領域(L)] [switch 文分岐テーブル領域(W)]	セクション名変更 プログラム領域のセクション名 定数領域のセクション名 初期化データ領域のセクション名 未初期化データ領域のセクション名 リテラル領域のセクション名 switch 文分岐テーブル領域のセクション名

No.	オプション	ダイアログメニュー	内容
5	stuff	コンパイラ <オブジェクト> [詳細…] [変数の配置 :]	変数のアライメントに応じたセクションに配置
	nostuff[= { B	[未初期化データ領域(B)]	初期値なし変数をアライメント 4 のセクションに配置
	| D	[初期化データ領域(D)]	初期値あり変数をアライメント 4 のセクションに配置
	| C	[定数領域(C)]	const 修飾変数をアライメント 4 のセクションに配置
	| W } [,…]]	[switch 文分岐テーブル領域(W)]	switch 文分岐テーブルをアライメント 4 のセクションに配置
6	-instalign4[=<sub>] -instalign8[=<sub>]	コンパイラ [分岐先の命令実行向け整合:]	
	-noinstalign	[4 バイト整合]	分岐先を 4 バイトで命令実行向け整合する
		[8 バイト整合]	分岐先を 8 バイトで命令実行向け整合する
		[なし]	分岐先を命令実行向け整合をしない
	<sub>:	[指定なし]	
	{ loop |	[loop]	ループの先頭
	inmostloop }	[inmostloop]	最内周ループの先頭

output

書　式　output = <sub> [=<ファイル名>]

<sub> : { prep | src | obj | abs | hex | sty }

説　明　出力ファイルの形式を指定します。

サブオプションと出力ファイルの一覧を以下に示します。

<ファイル名>を指定しない場合は、先頭に入力したソースファイル名に以下表の拡張子をつけたファイルを作成します。

本オプションの省略時解釈は、output=obj です。

表 2.3　サブオプション出力形式

サブオプション	出力形式	ファイル名指定省略時の拡張子
prep	**プリプロセッサ展開後のソースファイル**	C(C89, C99)**言語ソースファイル：**p C++**言語ソースファイル：**pp
src	**アセンブリ言語ファイル**	src
obj	**リロケータブルファイル**	obj
abs	**アブソリュートファイル**	abs
hex	**インテル** HEX **形式ファイル**	hex
sty	**モトローラ** S **形式ファイル**	mot

【注】　リロケータブルファイルは、アセンブラの出力ファイルです。
アブソリュートファイル、インテル HEX **形式ファイル、およびモトローラ** S **形式ファイルは、最適化リンケージエディタの出力ファイルです。**

備　考　指定した形式のファイルを作成するための中間ファイルは、フォルダ指定があればそのフォルダに、フォルダ指定がなければカレントフォルダに作成します。

noline

書　式　noline

説　明　プリプロセッサ展開時に#line 出力を抑止します。

備　考　本オプションは output=prep オプションの指定が無い場合は無効となります。

debug, nodebug

書　式　　debug

nodebug

説　明　　debug オプションを指定した場合、C ソースレベルデバッグに必要なデバッグ情報を出力します。debug オプションは、最適化オプションを指定した場合も有効となります。

nodebug オプションを指定した場合、デバッグ情報を出力しません。

本オプションの省略時解釈は、nodebug です。

section

書　式　　section = <sub>[,…]

<sub>: { P = <セクション名> |

C = <セクション名> |

D = <セクション名> |

B = <セクション名> |

L = <セクション名> |

W = <セクション名> }

説　明　　セクション名を指定します。

section=P =<セクション名>は、プログラム領域のセクション名を指定します。

section=C =<セクション名>は、定数領域のセクション名を指定します。

section=D =<セクション名>は、初期化データ領域のセクション名を指定します。

section=B =<セクション名>は、未初期化データ領域のセクション名を指定します。

section=L =<セクション名>は、リテラル領域のセクション名を指定します。

section=W=<セクション名>は、switch 文分岐テーブル領域のセクション名を指定します。

<セクション名>は、英字、数字、下線(_)、または$の列で、先頭が数字以外のものです。

本オプションの省略時解釈は、section=P=P,C=C,D=D,B=B,L=L,W=W です。

備　考　　プログラムとセクション名の対応についての詳細は、**「**8.1.2　C/C++**プログラムのセクション」**を参照してください。

V.1.00 と同様に L セクションを出力せず、リテラル領域を C セクションに出力したい場合は、section=L=C を選択してください。

L セクションを C セクションと同じ名前のセクション名に変更する場合を除き、領域が異なるセクションに同じセクション名を指定できません。

セクション名長の翻訳限界については、**「第 14 章　翻訳限界」**を参照してください。

stuff, nostuff

書　式　　stuff

nostuff [= <セクション種別>[,…]]

<セクション種別>：{ B | D | C | W}

説　明　　stuff オプションを指定した場合、全ての変数をアライメント数に応じてアライメント数が 4 のセクション、2 のセクション、1 のセクションに配置します(**表** 2.4)。

表 2.4　stuff **オプション指定時の、各変数と出力先セクションの関係**

変数の種類	変数のアライメント数	変数が所属するセクション
const **修飾変数**	4	C
	2	C_2
	1	C_1
初期値あり変数	4	D
	2	D_2
	1	D_1
初期値なし変数	4	B
	2	B_2
	1	B_1
switch **文分岐テーブル**	4	W
	2	W_2
	1	W_1

nostuff オプションを指定した場合、指定した<セクション種別>に属する変数をアライメント数が 4 のセクションに配置します。<セクション種別>を省略した場合は、すべてのセクション種別の変数が対象になります。

C、D、B は section オプションまたは#pragma section で指定したセクション名になります。W は section オプションで指定したセクション名になります。各セクション内のデータは常に定義順に出力されます。

本オプションの省略時解釈は、stuff です。

例

```
int a;
char b=0;
const short c=0;
struct {
    char x;
    char y;
} ST;
```

＜stuff オプション指定時＞

```
        .SECTION    C_2,ROMDATA,ALIGN=2
        .glb        _c
_c:
        .word       0000H
        .SECTION    D_1,ROMDATA
        .glb        _b
_b:
        .byte       00H
        .SECTION    B,DATA,ALIGN=4
        .glb        _a
_a:
        .blkl       1
        .SECTION    B_1,DATA,ALIGN=2
        .glb        _ST
_ST
        .blkb       2
```

＜nostuff オプション指定時＞

```
        .SECTION    C,ROMDATA,ALIGN=4
        .glb        _c
_c:
        .word       0000H
        .SECTION    D,ROMDATA,ALIGN=4
        .glb        _b
_b:
        .byte       00H
        .SECTION    B,DATA,ALIGN=4
        .glb        _a
_a:
        .blkl       1

        .glb        _ST
_ST
        .blkb       2
```

備　考　　sutff オプションは B,D,C および W 以外のセクションに対しては無効です。
nostuff オプションに B,D,C および W 以外のセクションを指定することはできません。

instalign4, instalign8, noinstalign

書　式　　instalign4[={loop|inmostloop}]

instalign8[={loop|inmostloop}]

noinstalign

説　明　　分岐先の命令実行向け整合を行います。

instalign4 を指定した場合は 4 バイト、-instalign8 を指定した場合は 8 バイトでそれぞれ配置アドレスを命令実行向け整合します

本オプションの省略時解釈は、noinstalign です。

命令実行向け整合とは、指定された場所の命令が、アライメント数(4 または 8)の倍数であるアドレスをまたいでいる場合(*1)にだけ整合を取るものです。

分岐先の種類は、-instalign4 と-instalign8 のサブオプションの指定により次の 3 種類から選択します(*2)。

指定なし:　関数先頭、switch 文の case および default ラベル

inmostloop:　各最内周ループの先頭、関数先頭、switch 文の case および default ラベル

loop:　各ループの先頭、関数先頭、switch 文の case および default ラベル

本オプションを選択すると、プログラムセクションのアライメント数が 1 から 4(instalign4 の場合)または 8(instalign8 の場合)に変わります。

本オプションは、分岐先命令のアドレス整合することで RX CPU の命令キューを効率よく動作させ、プログラムの実行速度向上を図るものです。
それぞれのオプションは、次の用途を想定した仕様となっております。

instalign8 … 命令キューが 64 ビットの CPU(主に RX600 シリーズ)で速度向上を図る場合

instalign4 … 命令キューが 32 ビットの CPU(主に RX200 シリーズ)で速度向上を図る場合

noinstalign … 本機能の効果を期待しない、もしくはコードサイズを重視したい場合

[注]

*1) 命令サイズがアライメント数以下の場合です。命令サイズがアライメント数よりも大きい場合は、またいでいる箇所が 2 以上の場合にだけ整合を取ります。

*2) ここに挙げたもの以外の分岐先は、命令実行向け整合の対象ではありません。たとえば、loop を選択した場合はループの先頭は対象ですが、ループ内にあるループを構成しない if 文などの分岐先は対象ではありません。

例 ＜C ソース＞

```
long a;
int f1(int num)
{
    return (num+1);
}
void f2(void)
{
    a = 0;
}
void f3(void)
{
}
```

＜出力コード＞

[-instalign8 を指定してコンパイルした場合]

下記は、各関数の先頭を、8 バイトで命令実行向け整合させた場合の例です。

8 バイトの命令実行向け整合では、対象命令が 8 バイトの境界をまたがない限り、配置アドレスを変更しないため、実際に整合がかかるのは関数 f2 のアドレスだけです。

```
        .SECTION P,CODE,ALIGN=8
        .INSTALIGN 8
_f1:                            ; 関数 f1。配置アドレス=0000H
        ADD     #01H,R1           ; 2 バイト
        RTS                       ; 1 バイト
        .INSTALIGN 8
_f2:                            ; 関数 f2。配置アドレス=0008H ※整合有り
                                  ; 0003H に 6 バイト命令が来ると、8 バイト境界を
                                  ; またぐため、整合が取られる。
        MOV.L   #_a,R4            ; 6 バイト
        MOV.L   #0,[R4]           ; 3 バイト
        RTS                       ; 1 バイト
        .INSTALIGN 8
_f3:                            ; 関数 f3。配置アドレス=0012H
        ADD     #01H,R1
        RTS
        .END
```

2.3 リストオプション

表 2.5 リストオプション一覧

No.	オプション	ダイアログメニュー	内容
1	listfile[=<ファイル名>] nolistfile	コンパイラ <リスト> [リスト出力]	ソースリストファイル出力あり ソースリストファイル出力なし
2	show = <sub>[,…] <sub>: {source | conditionals | definitions | expansions }	コンパイラ <リスト> [リスト内容 :]	ソースリストの内容の設定 C/C++ソースの出力 条件アセンブルで偽の行の出力 .DEFINE 置換前の情報 アセンブラマクロ記述展開行の出力

listfile, nolistfile

書　式　　listfile[=<ファイル名>]

nolistfile

説　明　　ソースリストファイルの出力有無を指定します。

listfile オプションを指定した場合、ソースリストファイルを出力します。＜ファイル名＞を指定することもできます。

nolistfile オプションを指定した場合、ソースリストファイルは出力しません。

＜ファイル名＞を指定しない場合は、ソースファイルと同じファイル名で、拡張子が lst のソースリストファイルを作成します。

本オプションの省略時解釈は、nolistfile です。

備　考　　本オプションを指定してリンケージリストを出力することはできません。リンケージリストを出力するには、lnkcmd オプションにより最適化リンケージエディタの list オプションを指定してください。

コンパイラが出力する情報は、ソースリストに書き込まれます。ソースリストファイルのフォーマットについては、**「7.2 ソースリストの参照方法」**を参照ください。

show

書　式　　show=<sub>[,…]

<sub> : { source | conditionals | definitions | expansions }

説　明　　ソースリストファイルの内容の設定を行います。

サブオプションと指定内容の一覧を以下に示します。

表 2.6　サブオプション指定一覧

サブオプション	内容
source	C/C++ソースを出力
conditionals	条件アセンブルで条件が偽となる行も含めて出力
definitions	.DEFINE を置き換える以前の情報を出力
expansions	アセンブラマクロ記述展開行を出力

備　考　　本オプションは listfile オプション指定時のみ有効です。

コンパイラが出力する情報は、ソースリストに書き込まれます。ソースリストファイルのフォーマットについては、**「7.2　ソースリストの参照方法」**を参照ください。

2.4 最適化オプション

最適化に関わるオプションは、条件により、適用されない場合もあります。当該最適化が適用されたかどうかは出力コードで確認してください。

表 2.7 最適化オプション一覧

No.	オプション	ダイアログメニュー	内容
1	optimize = { 0 | 1 | 2 | max }	コンパイラ <最適化> [最適化レベル :]	最適化レベルの指定
2	goptimize	コンパイラ <最適化> [モジュール間最適化]	モジュール間最適化用付加情報出力
3	speed size	コンパイラ <最適化> [最適化方法 :] [スピード優先 :] [サイズ優先 :]	最適化選択 実行性能重視の最適化を実施 コードサイズ重視の最適化を実施
4	loop[=<数値>]	コンパイラ <最適化> [詳細…] [その他] [ループ展開 :]	最大展開数=<数値> のループ展開を行う
5	inline[=<整数>] noinline	コンパイラ <最適化> [詳細…] [インライン展開] [自動インライン展開 :]	自動インライン展開を行う 自動インライン展開を行わない
6	file_inline = <ファイル名>[,…]	コンパイラ <最適化> [詳細…] [インライン展開] [インライン展開ファイル]	ファイル間インライン展開を行う
7	case = { ifthen | table | auto }	コンパイラ <最適化> [詳細…] [その他] [switch 文展開 :]	if_then 方式で展開 テーブルジャンプ方式で展開 展開方式をコンパイラが選択
8	volatile novolatile	コンパイラ <最適化> [詳細…] [外部変数] [外部変数の volatile 化]	外部変数を volatile 化する 外部変数を volatile 化しない
9	const_copy noconst_copy	コンパイラ <最適化> [詳細…] [外部変数] [外部変数の定数伝播 :]	const 宣言された外部変数の定数伝播を実施 const 宣言された外部変数の定数伝播を抑止

No.	オプション	ダイアログメニュー	内容
10	const_div noconst_div	コンパイラ <最適化> [詳細…] [その他] [定数除算/剰余算 :]	定数除算(剰余算)を乗算を用いた命令列で行う 定数除算(剰余算)を除算を用いた命令列で行う
11	library = { function | intrinsic }	コンパイラ <最適化> [詳細…] [その他] [ライブラリ関数 :]	ライブラリ関数を関数呼び出し 一部のライブラリ関数を命令展開
12	scope noscope	コンパイラ <最適化> [詳細…] [その他] [最適化範囲の分割 :]	最適化範囲を分割する 最適化範囲を分割しない
13	schedule noschedule	コンパイラ <最適化> [詳細…] [その他] [命令並べ替え :]	命令並べ替えを行う 命令並べ替えを行わない
14	map=<ファイル名> smap nomap	コンパイラ <最適化> [外部変数アクセス最適化 :] [モジュール間] [モジュール内] [なし]	外部変数アクセス最適化を行う コンパイル対象ファイル内で定義された外部変数に対し、外部変数アクセス最適化を行う 外部変数アクセス最適化を抑止
15	approxdiv	コンパイラ <最適化> [詳細…] [その他] [浮動小数点定数除算の乗算化]	浮動小数点定数除算の乗算化を行う
16	enable_register	コンパイラ <最適化> [詳細…] [その他] [register 指定変数の 優先レジスタ割り付け]	register 記憶クラスを指定した変数を優先的にレジスタ割り付け
17	simple_float_conv	コンパイラ <最適化> [詳細…] [その他] [浮動小数点数 – 整数変換時の範囲チェック省略]	浮動小数点型の型変換処理の一部を省略

No.	オプション	ダイアログメニュー	内容
18	fpu nofpu	コンパイラ <最適化> [詳細…] [その他] [FPU 命令の使用 :]	FPU 命令を使用したオブジェクトを出力 FPU 命令を使用しないオブジェクトを出力
19	alias={ noansi | ansi }	コンパイラ <最適化> [詳細…] [その他] [ポインタ指示先の型を考慮した最適化]	ポインタ指示先の型を考慮した最適化をしない ポインタ指示先の型を考慮した最適化をする
20	float_order	コンパイラ <最適化> [詳細…] [その他] [浮動小数点式の演算順序変更]	浮動小数点式の演算順序変更の最適化を行う

optimize

書　式　　optimize = { 0 | 1 | 2 | max }

説　明　　最適化レベルを指定します。

optimize=0 を指定した場合、最適化を実施しません。これにより、デバッグ情報を高い精度で出力でき、ソースレベルデバッグがしやすくなります。

optimize=1 を指定した場合、自動変数のレジスタ割付、関数出口ブロックの統合、統合可能な複数命令の統合など、一部最適化を実施します。これにより、optimize=0 指定時よりもコードサイズを削減できます。

optimize=2 を指定した場合、全般的に最適化を実施します。ただし、実施する最適化の内容は、size/speed オプションの選択によって若干異なります。

optimize=max を指定した場合、実施可能な最適化を最大限に行います。たとえば、最適化の適用範囲を最大限に拡大したり、speed オプション指定時には、大規模なループ展開を可能にします。最適化の効果が期待できる反面、コンパイル時間の増大や、speed オプション指定時のコードサイズ大幅な増加など、副作用を伴う場合があります。

本オプションの省略時解釈は、optimize=2 です。

備　考　　各種最適化オプションの説明で、デフォルトが記述されていないものは、optimize オプションと speed, size オプションの指定値によりデフォルトが変化することを意味します。デフォルトについての詳細は、speed, size オプションを参照ください。

goptimize

書　式　　goptimize

説　明　　モジュール間最適化時に使用する付加情報を、出力ファイル内部に生成します。
本オプションを指定したファイルは、リンク時にモジュール間最適化の対象になります。

speed, size

書　式　　speed
size

説　明　　speed オプションを指定した場合、実行性能重視の最適化を実施します。
size オプションを指定した場合、コードサイズ重視の最適化を実施します。

備　考　　speed オプション、size オプションを指定した場合、optimize オプションの指定により、以下オプションが指定されているとみなします。ただし、以下オプションを明示的に指定した場合はそちらが有効になります。

表 2.8　指定オプション

＜optimize=max 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=8	inline=250	const_div	schedule	const_copy	noscope	map* nomap*	alias=ansi
size	loop=1	inline=0	noconst_div	schedule	const_copy	noscope	map* nomap*	alias=ansi

【注】　*　入力が C/C++ソースで、かつ出力の指定が output=abs か mot の場合は map がデフォルトに、それ以外では nomap がデフォルトとなります。

＜optimize=2 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=2	inline=100	const_div	schedule	const_copy	scope	nomap	alias=noansi
size	loop=1	noinline	noconst_div	schedule	const_copy	scope	nomap	alias=noansi

＜optimize=0 または optimize=1 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=1	noinline	const_div	noschedule	noconst_copy	scope	nomap	alias=noansi
size	loop=1	noinline	noconst_div	noschedule	noconst_copy	scope	nomap	alias=noansi

loop

書　式　　loop[=<数値>]

説　明　　ループ展開の最適化を行うかどうかを指定します。

loop オプションを指定した場合、ループ文(for, while, do-while)を展開します。

<数値>で、最大で何倍の展開を行うかを指定することができます。<数値>は 1～32 の整数を指定することができます。<数値>を指定しなかった場合は 2 とします。

本オプションの省略時解釈は、optimize オプションと speed, size オプションの指定に従います。詳細は、speed, size オプションを参照してください。

inline, noinline

書　式　inline[=<数値>]

noinline

説　明　関数の自動インライン展開を行うかどうかを指定します。

<数値>として使える値の範囲は 0～65535 です。

inline オプションを指定した場合、自動インライン展開を行います。ただし、#pragma noinline を指定した関数はインライン展開を行いません。<数値>で、関数サイズが何%増加するまでインライン展開を行うかを指定できます。例えば、inline=100 を指定した場合、関数サイズが 100%増加するまで(2 倍まで)インライン展開を行います。

inline オプションを数値を省略して指定した場合の解釈は、inline=100 と同じです。

noinline オプションを指定した場合、自動インライン展開を行いません。

本オプションの省略時解釈は、optimize オプションと speed, size オプションの指定に従います。詳細は、speed, size オプションを参照してください。

備　考　#pragma inline を指定した関数、および inline 指定子付きの関数は、オプションの指定に関わらず、展開を試みます。なお、確実に関数をインライン展開したい場合は、#pragma inline を関数に指定してください。本オプションの選択もしくは inline 指定子を関数に指定しても、コンパイラで効率が悪くなると判断したときは、インライン展開を行わないことがあります。

file_inline

書　式　　file_inline=<ファイル名>[,…]

説　明　　<ファイル名>で指定されたファイルについて、ファイル間にまたがった関数インライン展開を行います。

ファイルが複数ある場合にはカンマ（,）で区切って指定することができます。

例

```
<a.c>
func(){
   g();
}
<b.c>
g(){
  h();
}
```

ccrx. -inline -file_inline=b.c a.c と指定してコンパイルすることにより a.c の中の関数 g()の呼び出しが展開され以下のようになります。

```
func(){
  h();
}
```

備　考　　file_inline オプションは、inline オプションまたは#pragma inline を指定した場合のみ有効となります。

file_inline オプションで指定された複数のファイルで同じ名前の extern 関数が定義されていた場合、動作は保証しません(任意に選んだ 1 つの関数定義を用いてインライン展開します)。

<ファイル名>で指定するファイル名の拡張子を省略することはできません。

コンパイル対象のファイルを file_inline オプションで指定することはできません。

<ファイル名>にワイルドカード(*, ?)を指定することはできません。

本オプションを複数回指定した場合、指定した全てのファイルが展開対象となります。

case

書　式　　case= { ifthen | table | auto }

説　明　　switch 文のコード展開方式を指定します。

case=ifthen を指定した場合、switch 文を if_then 方式で展開します。if_then 方式は、switch 文の評価式の値と case ラベルの値を比較し、一致すれば case ラベルの文へ飛ぶ処理を case ラベルの回数繰り返す展開方式です。この方式は、switch 文に含まれる case ラベルの数に比例してオブジェクトコードのサイズが増大します。

case=table を指定した場合、switch 文をテーブル方式で展開します。テーブル方式は、case ラベルの飛び先を分岐テーブルに確保し、1 回の分岐テーブルの参照で switch 文の評価式と一致する case ラベルの文へ飛ぶ展開方式です。この方式は、switch 文に含まれる case ラベルの数に比例して分岐テーブルのサイズが増えますが、実行速度は常に一定です。分岐テーブルは、switch 文分岐テーブル領域のセクションに出力されます。

case=auto を指定した場合、if_then 方式、テーブル方式いずれかをコンパイラが自動的に選択します。

本オプションの省略時解釈は、case=auto です。

備　考　　case=table 指定時に作成される分岐テーブルは、nostuff オプション指定時は W セクションに出力されますが、nostuff オプション指定がない場合は、switch 文の規模により W、W_2 または W_1 セクションのいずれか振り分けて出力されます。

volatile, novolatile

書　式　　volatile

novolatile

説　明　　volatile を指定した場合、すべての外部変数を volatile 宣言したものとして扱います。したがって、外部変数のアクセス回数、アクセス順序は C/C++言語ソースファイルで記述した通りになります。

novolatile を指定した場合、volatile 修飾のない外部変数に対して最適化を行います。したがって、外部変数のアクセス回数、アクセス順序が C/C++言語ソースファイルで記述した場合と異なることがあります。

本オプションの省略時解釈は、novolatile です。

const_copy, noconst_copy

書　式　const_copy

noconst_copy

説　明　const_copy を指定した場合、const 修飾型外部変数についても定数伝播を行います。

noconst_copy を指定した場合、const 修飾型外部変数の定数伝播を抑止します。

本オプションの省略時解釈は、optimize=2 または optimize=max オプションを指定した場合は const_copy、それ以外の場合は noconst_copy です。

備　考　C++言語ソースファイルの const 修飾型変数については、本オプションで制御することはできません(常に定数伝播されます)。

const_div, noconst_div

書　式　const_div

noconst_div

説　明　const_div を指定した場合、ソースファイル中の整数型定数による除算および剰余算を、乗算を用いた命令列に変換します。

noconst_div を指定した場合、ソースファイル中の整数型定数による除算および剰余算を、除算を用いた命令列に変換します。

本オプションの省略時解釈は、speed オプションを指定した場合は const_div、size オプションを指定した場合は noconst_div です。

備　考　シフト演算で行える定数乗算、およびビット論理積で行える剰余算は、const_div オプションおよび noconst_div オプションの制御対象外となります。

library

書　式　library = { function | intrinsic }

説　明　library=function を指定した場合、ライブラリ関数を全て関数呼び出しします。

library=intrinsic を指定した場合、abs()、fabsf()およびストリング操作命令が使用できるライブラリ関数を命令展開します。

本オプションの省略時解釈は、library=intrinsic です。

scope, noscope

書　式　scope

noscope

説　明　scope を指定した場合、サイズの大きい関数について、最適化範囲を複数に分割してコンパイルします。

noscope を指定した場合、最適化範囲を分割せずにコンパイルします。最適化範囲が広がることによりコンパイル速度は遅くなりますが、一般的にはオブジェクト性能が向上します。ただし、レジスタ数が不足するとオブジェクト性能が低下する場合があります。本オプションは、プログラムによって実行性能に影響しますので、性能チューニング時に試してください。

本オプションの省略時解釈は、optimize=max オプションを指定した場合は noscope、それ以外の場合は scope です。

schedule, noschedule

書　式　schedule

noschedule

説　明　schedule を指定した場合、パイプライン処理を考慮した命令並べ替えを行います。

noschedule を指定した場合、命令並べ替えを行いません。基本的に C/C++言語ソースファイルで記述した順番で処理を行います。

本オプションの省略時解釈は、optimize=2 または optimize=max オプションを指定した場合は schedule、それ以外の場合は noschedule です。

map, smap, nomap

書　式　　map[= <ファイル名>]

smap

nomap

説　明　　外部変数アクセス最適化を行います。

map オプションを指定した場合、最適化リンケージエディタが生成する外部シンボル割り付け情報を元にベースアドレスを設定し、外部変数もしくは静的変数のアクセスをベースアドレス相対で行うコードを生成します。

smap オプションを指定した場合、コンパイル対象ファイル内で定義された外部変数もしくは静的変数についてベースアドレスを設定し、アクセスをベースアドレス相対で行うコードを生成します。

map オプションによる外部変数アクセス最適化を使用する場合は、output オプションの指定により使い方が異なります。

[output=abs または mot の場合]
map のみ指定してください(optimize=max 指定時は不要)。自動的にコンパイル・リンクを 2 回行い、外部シンボル割り付け情報を元にベースアドレスを設定したコード生成を行います

[output=obj の場合]
ソースファイルを本オプションを指定しないで一度コンパイルし、最適化リンケージエディタでのリンク時に map=<ファイル名>を指定して外部シンボル割り付け情報ファイルを作成し、再度 ccrx に map=<ファイル名>を指定してコンパイルしてください。

nomap オプションを指定した場合、外部変数アクセス最適化を行いません。

本オプションの省略時解釈は、optimize=max オプションを指定した場合は map、それ以外の場合は nomap です。

例　　＜C ソース＞

```
long A,B,C;
void func()
{
    A = 1;
    B = 2;
    C = 3;
}
```

＜出力コード＞

(1) 最適化非実施の場合

```
_func:
        MOV.L       #_A,R4
        MOV.L       #1,[R4]
        MOV.L       #_B,R4
        MOV.L       #2,[R4]
        MOV.L       #_C,R4
        MOV.L       #3,[R4]
```

(2) 最適化実施の場合

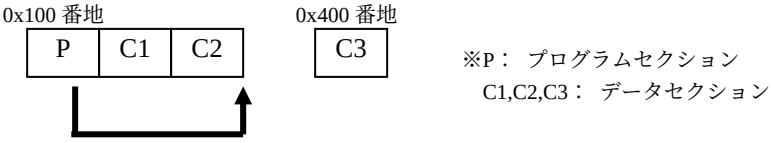
```
_func:
        MOV.L       #_A,R4      ; A のアドレスをベースアドレスに設定
        MOV.L       #1,[R4]
        MOV.L       #2,4[R4]    ; A のアドレスをベースとして、B にアクセス
        MOV.L       #3,8[R4]    ; A のアドレスをベースとして、C にアクセス
```

備　考　外部変数もしくは静的変数の定義順を変更した場合は、外部シンボルアドレス情報ファイルを生成し直す必要があります。map オプション以外で 1 回目のコンパイル時に指定したオプションと異なるオプションを指定した場合と、関数内の処理を追加した場合は、動作は保証しません。これらの場合は必ず外部シンボルアドレス情報ファイルを生成し直してください。

C/C++ソースをコンパイルする場合のみ適用されます。コンパイル時に output=src を用いて作成、またはアセンブリ言語で記述されたプログラムには適用されません

map オプションと smap オプションを同時に指定した場合は、map オプションが有効となります。

プログラムセクションの次に連続してデータセクションを配置すると、外部変数アクセス最適化が無効になる、あるいは、最適化が十分に機能しない場合があります。
最適化を最大限に機能させるためには、連続して複数のセクションを配置させる場合、プログラムセクションを末尾に配置してください。
以下に具体例を示します。



P を 0x100 番地から、C1,C2 を P の直後、C3 を 0x400 番地から配置したとします。
この場合、P セクションと連続して C1,C2 セクションが配置されているため、C2 の後方に配置してください。C3 セクションは配置関係が連続していないため無関係です。

approxdiv

書　式　approxdiv

説　明　浮動小数点定数除算を、定数の逆数の乗算に変換します。

すなわち、変数÷除数 という式があるとき、除数が定数の場合は、変数×除数の逆数 に変換したコードを生成します。

備　考　本オプションを指定した場合、浮動小数点定数除算の実行速度は向上しますが、演算の精度が変わる場合がありますので注意が必要です。

enable_register

書　式　enable_register

説　明　register 記憶クラスを指定した変数を、優先的にレジスタに割り付けます。

備　考　message オプション指定時、レジスタに割り付かなかった場合は、インフォメーションメッセージ

C0102 (I) Register is not allocated to "変数名" in "関数名"

を出力します。ただし、引数がレジスタに割り付かなかった場合は、本メッセージは出力しません。

simple_float_conv

書　式　　simple_float_conv

説　明　　浮動小数点型の型変換処理の一部を省略します。
本オプション選択時は、次の浮動小数点の型変換を行う生成コードが変化します。
(1) 32bit 浮動小数点型から符号無し整数型への変換
(2) 符号無し整数型から 32bit 浮動小数点型への変換
(3) 32bit 浮動小数点型を経由した、整数型から 64bit 浮動小数点型への変換

備　考　　本オプション指定時、該当する型変換の処理に対するコード性能は向上しますが、変換結果が C,C++言語規格と異なる場合がありますので、ご注意ください。

例　1　　＜32bit 浮動小数点型から符号無し整数型への変換＞

```
unsigned long func(float f)
{
        return ((unsigned long)f);
}
```

オプション非指定時：

```
    _func1:
        MOV.L   R1,R5
        FCMP    #4F000000H,R1
        BLT     L12
        FADD    #0CF800000H,R5
    L12:
        FTOI    R5,R1
        RTS
```

オプション指定時：

```
        FTOI    R1,R1
        RTS
```

例　2　　＜符号無し整数型から 32bit 浮動小数点型への変換＞

```
float func2(unsigned long u)
{
    return ((float)u);
}
```

オプション非指定時：

```
_func2:
        BTST    #31,R1
        BEQ     L15
        SHLR    #1,R1,R4
        AND     #1,R1
        OR      R4,R1
        ITOF    R1,R4
        FMUL    #40000000H,R4
        BRA     L16
L15:
        ITOF    R1,R4
L16:
        MOV.L   R4,R1
        RTS
```

オプション指定時：

```
        ITOF    R1,R1
        RTS
```

例 3 ＜32bit 浮動小数点型を経由した、整数型から 64bit 浮動小数点型への変換＞

【注意】-dbl_size=8 の指定が有効でない場合は該当しません。

```
double func3(long l)
{
    return (double)(float)(double)l;
}
```

オプション非指定時：

```
_func3:
        BSR     __COM_CONV32sd
        BSR     __COM_CONVdf
        BRA     __COM_CONVfd
```

オプション指定時：

```
        BRA     __COM_CONV32sd
```

fpu, nofpu

書　式　fpu

nofpu

説　明　浮動小数点演算に対するコード生成について、FPU 命令の使用有無を選択します。

fpu を指定した場合、FPU 命令を使用したコード生成を行います。

nofpu を指定した場合、FPU 命令を使用せず、実行時ルーチン呼び出しによるコード生成を行います。

本オプションの省略時解釈は、CPU として RX600 が選択(*1)された場合は fpu、RX200 が選択(*1)された場合は nofpu です。

*1) cpu オプションまたは環境変数 CPU_RX による選択を指します。

備　考　FPU 命令の具体的な内容については、**「RX ファミリ ソフトウェアマニュアル」**を参照してください。

CPU として RX200 が選択されている場合、fpu を指定するとエラーになります。

-fpu 有効時、非正規化数に前回と結果が同じになる演算(かける 1 など)をした場合、演算結果が 0 に変化することがあります。

例　次のソースコードを -fpu -denormalize=on が有効な場合、実行の結果は、f=0.0 または f=1E-40 のいずれかになります。

```
float func(void)
{
    float f = 1E-40;
    f*=1;
    return f;
}
```

alias

書　式　　alias = { noansi | ansi }

説　明　　ポインタ指示先の型を考慮した最適化を実施するかどうかを選択します。

alias=ansi を指定した場合、ANSI 規格に基づき、ポインタ指示先の型を考慮した最適化を行います。一般には、alias=noansi を指定した場合よりもオブジェクト性能が向上しますが、alias=ansi と alias=noansi とで実行結果が異なる場合があります。

alias=noansi を指定した場合は V.1.00 と同様で、ANSI 規格に基づくポインタ指示先の型を考慮した最適化を行いません。

本オプションの省略時解釈は、alias=noansi です。

例

```
long x;
long n;
void func(short * ps)
{
    n = 1;
    *ps = 2;
    x = n;
}

[alias=noansi 指定時]
; ※ *ps = 2; によって、n の値が書き換わる可能性があるとみなし
;     (A)で n の値を再ロードします。
    _func:
        MOV.L       #_n,R4
        MOV.L       #1,[R4]     ;   n = 1;
        MOV.W       #2,[R1]     ;   *ps = 2;
        MOV.L       [R4],R5     ; (A) n を再ロードする
        MOV.L       #_x,R4
        MOV.L       R5,[R4]
        RTS

[alias=ansi 指定時]
; ※ *ps と n は型が異なるため、*ps = 2; では n の値は変化しないと判断し、
;     (B) で、n = 1 で代入に使用した値を再利用します。
;   (もし*ps = 2; によって n の値が書き換わる場合、結果は変わります。)
    _func:
```

```
MOV.L     #_n,R4
MOV.L     #1,[R4]     ;   n = 1;
MOV.W      #2,[R1]     ;   *ps = 2;
MOV.L     #_x,R4
MOV.L     #1,[R4]     ; (B) n = 1  で代入に使用した値を再利用する
RTS
```

備　考　optimize=0 または optimize=1 が有効な場合に alias オプションを選択すると、alias=ansi の選択は無視され、常に alias=noansi が選択されたものとしてコード生成します。

float_order

書　式　float_order

説　明　浮動小数点演算式の演算順序変更の最適化を行ないます。

一般には、float_order を指定しない場合よりもオブジェクト性能が向上しますが、演算の精度が float_order を指定しなかった場合と異なることがあります。

例

```
float a,b,c;
f()
{
    a = b * 100.0f + c * 100.0f;
}
```

例で示した浮動小数点演算式は、float_order を有効にすると a = (b + c) * 100.0f; と等価な演算式に置き換わります。

備　考　本オプションは、optimize=2 または optimize=max を指定した場合のみ有効です。

optimize=0 または optimize=1 を指定した場合は、本オプションは無視されます。その場合、警告 C1301(W)を表示します。

2.5 マイコンオプション

表 2.9 マイコンオプション一覧

No.	オプション	ダイアログメニュー	内容
1	cpu = { rx600 | rx200 }	CPU [CPU 種別 :]	RX600 向けの命令コードを生成 RX200 向けの命令コードを生成
2	endian = { big | little }	CPU [エンディアン :]	データのエンディアン指定 Big Endian Little Endian
3	round = { zero | nearest }	CPU [詳細…] [詳細] [丸めモード :]	round to zero で丸める round to nearest で丸める
4	denormalize = { off | on }	CPU [詳細…] [詳細] [非正規化数を 非正規化数として扱う]	非正規化数を 0 として扱う 非正規化数を非正規化数として扱う
5	dbl_size = { 4 | 8 }	CPU [詳細…] [詳細] [double 型の精度 :] [単精度] [倍精度]	double 型、long double 型を単精度として扱う double 型、long double 型を倍精度として扱う
6	int_to_short	CPU [詳細…] [詳細] [int 型を short 型に置換]	int 型を short 型に、unsigned int 型を unsigned short 型に置換
7	signed_char unsigned_char	CPU [詳細…] [詳細] [char 型の符号 :]	char 型を signed char 型として扱う char 型を unsigned char 型として扱う
8	signed_bitfield unsigned_bitfield	CPU [詳細…] [詳細] [ビットフィールドの符号 :]	ビットフィールドの符号を signed で解釈 ビットフィールドの符号を unsigned で解釈

No.	オプション	ダイアログメニュー	内容
9	auto_enum	CPU [詳細…] [詳細] [列挙型サイズの自動選択]	列挙型サイズの自動選択
10	bit_order = { left | right }	CPU [詳細…] [詳細] [ビットフィールドの並び順 :] [上位ビット] [下位ビット]	ビットフィールドメンバを左から順に詰め込む ビットフィールドメンバを右から順に詰め込む
11	pack unpack	CPU [詳細…] [詳細] [構造体メンバの境界調整数を１とする]	構造体メンバのアライメントを 1 とする データのアライメントに従う
12	exception noexception	CPU [詳細…] [詳細] [C++の try、throw、catch を有効にする]	例外処理機能を有効にする 例外処理機能を無効にする
13	rtti= { on | off}	CPU [詳細…] [詳細] [C++の dynamic_cast、typeid を有効にする]	dynamic_cast、typeid を有効にする dynamic_cast、typeid を無効にする
14	fint_register = { 0 | 1 | 2 | 3 | 4 }	CPU [高速割り込みレジスタ :] [なし] [R13] [R12,R13] [R11,R12,R13] [R10,R11,R12,R13]	高速割り込み関数でのみ使用する汎用レジスタを指定 高速割り込み専用のレジスタはなし R13 を高速割り込み専用で使用 R13～R12 を高速割り込み専用で使用 R13～R11 を高速割り込み専用で使用 R13～R10 を高速割り込み専用で使用
15	branch = { 16 | 24 | 32 }	CPU [詳細…] [詳細] [関数の分岐幅 :]	分岐幅のサイズが 16bit 以内であることを保証 分岐幅のサイズが 24bit 以内であることを保証 分岐幅のサイズを限定しない

No.	オプション	ダイアログメニュー	内容
16	base = { rom = <レジスタ> | ram= <レジスタ> | <アドレス値> = <レジスタ> }	CPU [ベースレジスタ :]	ROM 用ベースレジスタ指定 RAM 用ベースレジスタ指定 アドレス値を設定するベースレジスタを指定
17	patch = { rx610 }	CPU [CPU タイプ特有の問題を回避 :]	CPU の品種ごとに特有の問題を回避する MVTIPL 命令を使用しない (RX610 グループ向け)
18	pic	CPU [詳細…] [PIC/PID] [コードセクションを位置独立コードとして生成]	PIC 機能を有効にする
19	pid = { 16 | 32 }	CPU [詳細…] [PIC/PID] [コードセクションを位置独立コードとして生成] [オフセットの最大幅:] [16 ビット] [32 ビット]	PID 機能を有効にする 16 ビット(64KB～256KB)のアドレッシング対応 32 ビット(4GB)のアドレッシング対応
20	nouse_pid_register	CPU [詳細…] [PIC/PID] [PID 用レジスタを使用しない]	PID レジスタをコード生成に使用しない
21	save_acc	CPU [詳細…] [詳細] [アキュムレータの退避/回復コードを生成:]	割り込み関数で ACC を退避・回復する

cpu

書　式　cpu={ rx600 | rx200 }

説　明　生成する命令コードのマイコン種別を指定します。
cpu=rx600 を指定した場合、RX600 向けの命令コードを生成します。
cpu=rx200 を指定した場合、RX200 向けの命令コードを生成します。

備　考　cpu=rx200 を選択すると、nofpu が自動的に選択されます。
cpu=rx200 と fpu は同時に指定することはできません。
nofpu と fpu のいずれの選択もない場合に cpu=rx600 を指定すると、fpu が自動的に選択されます。

endian

書　式　endian={ big　|　little }

説　明　endian=big を指定した場合、データのバイト並びが big endian になります。
endian=little を指定した場合、データのバイト並びが little endian になります。
#pragma endian 拡張子でも指定できます。オプションと#pragma 拡張子の両方が指定された場合には、#pragma 拡張子の指定を優先します。
本オプションの省略時解釈は、endian=little です。

round

書　式　round={ zero | nearest }

説　明　浮動小数点定数演算の丸め方式を選択します。
round=zero を指定した場合、round to zero で丸めます。
round=nearest を指定した場合、round to nearest で丸めます。
本オプションの省略時解釈は、round=nearest です。

備　考　本オプションでは、実行時の浮動小数点演算における丸め方式を変更することはできません。
本オプションのデフォルトの選択は、fpu, nofpu オプション選択の影響を受けません。

denormalize

書　式　denormalize={ off | on }

説　明　浮動小数点定数に非正規化数を記述した場合の扱いを指定します。
denormalize=off を指定した場合、非正規化数を 0 として扱います。
denormalize=on を指定した場合、非正規化数を非正規化数として扱います。
本オプションの省略時解釈は、denormalize=off です。

備　考　本オプションでは、実行時の浮動小数点演算における非正規化数の扱いを変更することはできません。
本オプションは、fpu, nofpu オプション選択で自動的に有効になることはありません。

dbl_size

書　式　dbl_size={4 | 8}

説　明　double 型、および long double 型の精度を指定します。

dbl_size=4 を指定した場合、単精度浮動小数点型(4 バイト)として扱います。

dbl_size=8 を指定した場合、倍精度浮動小数点型(8 バイト)として扱います。

本オプションの省略時解釈は、dbl_size=4 です。

備　考　dbl_size=4 を選択した場合、標準関数のうち math.f.h と math.h とで同じ仕様の関数(例: sqrtf と sqrt など)を一体化して標準ライブラリが構築されます。このため dbl_size=4 選択時は、例えば mathf.h ヘッダの関数である sqrtf を呼び出すところを、RX のシミュレータやエミュレータでトレース(ステップ実行)すると、sqrtf ではなく同じ仕様を持つ math.h ヘッダの関数 sqrt が呼び出されたように見えることがあります。

int_to_short

書　式　int_to_short

説　明　ソースファイル内の int を short に、unsigned int を unsigned short に置換してコンパイルを行います。

備　考　limits.h の INT_MAX、INT_MIN、および UINT_MAX は本オプションの変換対象外となります。

C++および EC++コンパイル時は、本オプションは無効になります。C++、EC++プログラム内から C プログラムを参照する可能性がある外部名に対して C1804(W)を出力します。

C 標準ヘッダをインクルードしたファイルを int_to_short オプションを指定して C++または EC++コンパイルした場合も、C1804(W)が出力されることがあります。この場合は動作には問題ありませんので無視してください。

C と C++(EC++)との間で共通にアクセスするデータは、int 型ではなく long 型または short 型で宣言してください。

signed_char, unsigned_char

書　式　signed_char

unsigned_char

説　明　符号指定のない char 型の符号を指定します。

signed_char を指定した場合、signed char 型として扱います。

unsigned_char を指定した場合、unsigned char 型として扱います。

本オプションの省略時解釈は、unsigned_char です。

備　考　char 型のビットフィールドメンバは、本オプションの制御対象外です。signed_bitfield および unsigned_bitfield オプションで制御してください。

signed_bitfield, unsigned_bitfield

書　式　signed_bitfield

unsigned_bitfield

説　明　符号指定のないビットフィールド型の符号を指定します。

signed_bitfield を指定した場合、符号付き型として扱います。

unsigned_bitfield を指定した場合、符号なし型として扱います。

本オプションの省略時解釈は、unsigned_bitfield です。

auto_enum

書　式　　auto_enum

説　明　　enum 宣言した列挙型のデータを、列挙値が収まる最小型として処理します。
本オプションの省略時解釈は、列挙型サイズを signed long 型として処理します。
列挙型のとりうる値と型の関係を以下に示します。

表 2.10　列挙型のとりうる値と型の関係

列挙子		選択される型
最小値	最大値	
-128	127	signed char
0	255	unsigned char
-32768	32767	signed short
0	65535	unsigned short
上記以外		signed long

bit_order

書　式　　bit_order = { left | right }

説　明　　ビットフィールドのメンバの並び順を指定します。

bit_order=left を指定した場合は上位ビットからメンバを割り付けます。

bit_order=right を指定した場合は下位ビットからメンバを割り付けます。

#pragma bit_order 拡張子でも指定できます。オプションと#pragma の両方が指定された場合には、拡張子の指定を優先します。

本オプションの省略時解釈は、bit_order=right です。

pack, unpack

書　式　　pack

unpack

説　明　　構造体メンバ、クラスメンバのアライメント数を指定します。

構造体メンバのアライメント数は、#pragma pack 拡張子でも指定できます。オプションと#pragma の両方が指定された場合には、#pragma 拡張子の指定を優先します。構造体、クラスのアライメント数は、メンバの最大のアライメント数と同じになります。

本オプションの省略時解釈は、unpack です。

備　考　　本オプション指定時の構造体メンバのアライメント数を以下に示します。

表 2.11　pack オプション指定時の構造体メンバ、クラスメンバのアライメント数

メンバの型	pack	unpack	指定なし
(signed) char	1	1	1
(unsigned) short	1	2	2
(unsigned) int *, (unsigned) long, (unsigned) long long, **浮動小数点型**, **ポインタ型**	1	4	4

【注】　*　int_to_short **オプションを指定した場合は、**short **と同じになります。**

exception, noexception

書　式　exception

noexception

説　明　exception オプションを指定した場合、C++例外処理機能(try, catch, throw)を有効にします。

noexception オプションを指定した場合、C++例外処理機能(try, catch, throw)を無効にします。

exception オプションを指定した場合、コード性能が低下する可能性があります。

本オプションの省略時解釈は、noexception です。

備　考　ファイル間で例外処理機能を有効にするには以下を行ってください。

(1) rtti=on を指定する。

(2) 最適化リンケージエディタで noprelink オプションを指定しない。

exception オプションは C++コンパイル時にのみ指定できます。lang=cpp の指定がなく、かつ入力ファイルの拡張子が.c または .p の場合、exception オプションは指定できません。指定するとエラーとなります。

rtti

書　式　rtti={ on | off }

説　明　実行時型情報の有効/無効を指定します。

rtti=on を指定した場合、dynamic_cast、typeid を有効にします。

rtti=off を指定した場合、dynamic_cast、typeid を無効にします。

本オプションの省略時解釈は、rtti=off です。

備　考　本オプションを指定して作成したリロケータブルファイル(.obj)をライブラリに登録したり、最適化リンケージエディタでリロケータブル形式(.rel)で出力しないでください。シンボルの二重定義エラーや未定義エラーになることがあります。

rtti=on は、C++コンパイル時にのみ指定できます。lang=cpp の指定がなく、かつ入力ファイルの拡張子が.c または .p の場合、rtti=on は指定できません。指定するとエラーとなります。

fint_register

書　式　　fint_register = {0 | 1 | 2 | 3 | 4 }

説　明　　高速割り込み関数（#pragma interrupt で割り込み仕様に高速割り込み指定(fint)のある関数）でのみ使用する汎用レジスタを指定します。高速割り込み関数以外では、指定されたレジスタは使用しません。本オプションで指定した汎用レジスタは、高速割り込み関数内では退避回復なしで使用できるため、高速割り込み関数の高速化が見込めます。反面、他の関数で使用可能な汎用レジスタが減るため、プログラム全体のレジスタ割付効率は低下します。

オプションとレジスタの関係を以下に示します。

表 2.12　オプションとレジスタの関係

オプション	高速割り込み専用レジスタ
fint_register=0	**なし**
fint_register=1	R13
fint_register=2	R12, R13
fint_register=3	R11, R12, R13
fint_register=4	R10, R11, R12, R13

本オプションの省略時解釈は、fint_register=0 です。

備　考　　高速割り込み関数以外で、本オプションで指定したレジスタを使用した場合の動作は保証しません。本オプションの指定の対象となるレジスタが、base オプションで指定されていた場合、エラーとなります。

branch

書　式　　branch = { 16 | 24 | 32 }

説　明　　分岐幅を指定します。

branch =16 を指定した場合、分岐幅が 16bit 以内であるとしてコンパイルします。

branch =24 を指定した場合、分岐幅が 24bit 以内であるとしてコンパイルします。

branch =32 を指定した場合、分岐幅を指定しません。

本オプションの省略時解釈は、branch=24 です。

base

書　式　　base = {rom=<レジスタ>
| ram=<レジスタ>
| <アドレス値> = <レジスタ>}
<レジスタ>:= {R8 ～ R13}

説　明　　プログラム全体で、ベースアドレスとして固定で使用する汎用レジスタを指定します。

base=rom=<レジスタ A>を指定した場合は、const 変数のアクセスはすべて指定したレジスタ A 相対で行います。ただし、定数領域セクション全体の大きさが 64KB～256KB*[1] 以内でなければなりません。

base=ram=<レジスタ B>を指定した場合は、初期化変数および未初期化変数のアクセスはすべて指定したレジスタ B 相対で行います。ただし、RAM データ全体の大きさが 64KB～256KB*[1] 以内でなければなりません。

<アドレス値>=<レジスタ C>を指定した場合は、アドレス値から 64KB～256KB*[1] 以内の領域のアクセスは、指定したレジスタ C 相対で行います。

【注】　*1　この値は、アクセスする変数のサイズにより、64KB から 256KB の間で変化します。

備　考　　異なる領域に対して同じレジスタを指定することはできません。

レジスタはそれぞれの領域で 1 個だけ指定可能です。fint_register オプションで指定したレジスタを本オプションで指定した場合、エラーとなります。

pid オプション選択時は、base=rom=<レジスタ> を選択できません。選択すると、警告として C1801(W)メッセージを表示して base=rom=<レジスタ>の選択を無効とします。

patch

書　式　　patch ={rx610 }

説　明　　CPU の品種ごとに特有の問題を回避します。

-patch=rx610 を指定すると、RX610 グループで問題となる、MVTIPL 命令を生成コードに使用しません。
-patch=rx610 を指定しなければ、組み込み関数 set_ipl の呼び出しに対する生成コードは、MVTIPL 命令を含んだものとなります。

pic

書　式　　pic

説　明　　プログラムセクションを PIC(位置独立コード)としてコード生成します。

PIC においては、関数呼び出しは全て BSR または BRA 命令を用いて行い、また関数のアドレスを取得するときは PC からの相対アドレスを用いるようにします。これによって、PIC はリンク後に任意のアドレスに配置することができます。

例

例 1　関数呼び出し (ただし、branch=32 の場合)

```
void func()
{
        sub();
}
```

[-pic なし]

```
_func:
        MOV.L    #_sub,R14
        JMP      R14
```

[-pic あり]

```
_func:
        MOV.L    #_sub-L11,R14
L11:
        BRA      R14
```

例 2　関数アドレスの取得

```
void func1(void);
void (*f_ptr)(void);
void func2(void)
{
        f_ptr = func1;
}
```

[-pic なし]

```
_func2:
        MOV.L    #_f_ptr,R4
        MOV.L    #_func1,[R4]
        RTS
```

[-pic あり]

```
_func2:
        MOV.L    #_f_ptr,R4
L11:
        MVFC     PC,R14
        ADD      #_func1-L11,R14
        MOV.L    R14,[R4]
        RTS
```

備　考　C++または EC++コンパイル時は、pic オプションを選択できません。選択すると、警告として C1801(W)メッセージを表示して pic の選択を無効とします。

PIC である関数のアドレスを、静的初期化の初期化式に使用しないでください。エラー C6698(E) となります。

PIC アドレスを静的初期化に用いる例:

```
void pic_func1(void), pic_func2(int), pic_func3(int);   /* PIC になる */
void (*fptr1_for_pic) = pic_func1;        /* PIC アドレスを静的初期化で使用: エラー */
struct PIC_funcs{ int code; void (*fptr)(int); };
struct PIC_funcs pic_funcs[] = {
    { 2, pic_func2 },        /* PIC アドレスを静的初期化で使用: エラー */
    { 3, pic_func3 },        /* PIC アドレスを静的初期化で使用: エラー */
};
```

PIC 機能を利用するアプリケーションプログラムのスタートアップを作成する際は、「8.3 スタートアップ」ではなく、「8.4.7 アプリケーションのスタートアップ」を参照ください。

PIC 機能については、「8.4 PIC/PID 機能の利用」の項目も参照してください。

pid

書　式　pid[={ 16|32 }]

説　明　定数領域セクション C, C_2 および C_1、リテラルセクション L と switch 文分岐テーブルセクション W, W_2 および W_1 を PID(位置独立データ)とします。

PID のアクセスは、全て PID レジスタからの相対アドレスで行います。これにより、PID はリンク後に任意のアドレスに配置することができます。

PID 機能を実現するためには、汎用レジスタを 1 本消費します。

＜PID レジスタ＞

下記の表の規則に基づき、fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお、fint_register の指定がない場合は R13 を選択します。

表 2.13　fint_register オプションと PID レジスタの関係

fint_register オプション	PID レジスタ
fint_register 指定なし	R13
fint_register=0	
fint_register=1	R12
fint_register=2	R11
fint_register=3	R10
fint_register=4	R9

PID レジスタは、PID のアクセスに使用する用途以外には使用されません。

＜パラメータ＞

パラメータは、PID レジスタから定数領域セクションをアクセスする際の、オフセットの最大幅のビット数を 16 または 32 で選択します。

オフセット幅を省略して pid オプションを選択した場合の解釈は、pid=16 です。pid=16 では、PID レジスタがアクセスできる定数領域セクションのサイズが 64KB～256KB(アクセス幅により変動する)に制限されます。pid=32 では、PID レジスタがアクセスできる定数領域セクションのサイズに制限はありませんが、PID をアクセスするコードのサイズが増大します。

なお、pid=32 と 外部シンボル割り付け情報が有効な map オプションを同時指定した場合は、割り付け情報により、PID レジスタによるアクセスが可能な場合は pid=16 と同じコードを生成します。

例

例 1　const 修飾した外部参照シンボルをアクセス

```
extern const int pid;
int work;
void func1()
{
        work = pid;
}
[-pid なし]
_func1:
        MOV.L       #_pid,R4
        MOV.L       [R4],R5
        MOV.L       #_work,R4
        MOV.L       R5,[R4]
        RTS
[-pid=16 あり] (※ただし、PID レジスタが R13 の場合)
_func1:
        MOV.L       _pid-__PID_TOP:16[R13],R5
        MOV.L       #_work,R4
        MOV.L       R5,[R4]
        RTS
        .glb        __PID_TOP

[-pid=32 あり] (※ただし、PID レジスタが R13 の場合)
_func1:
        ADD         #(_pid-__PID_TOP),R13,R6
        MOV.L       [R6],R5
        MOV.L       #_work,R4
        MOV.L       R5,[R4]
        RTS
        .glb        __PID_TOP
```

例 2　const 修飾した外部定義シンボルのアドレスを取得

```
extern const int pid = 1000;
const int *ptr;
void func2()
{
        ptr = &pid;
}
```

```
[-pid なし]
_func2:
        MOV.L   #_ptr,R4
        MOV.L   #_pid,[R4]
        RTS
[-pid あり] (※ただし、PID レジスタが R13 の場合)
_func2:
        ADD     #(_pid-__PID_TOP),R13,R5
        MOV.L   #_ptr,R4
        MOV.L   R5,[R4]
        RTS
        .glb    __PID_TOP
```

備　考　PID である領域のアドレスを、静的初期化の初期化式に使用しないでください。エラー C6699(E) となります。

PID アドレスを静的初期化に用いる例:

```
extern const int pid_data1;     /* PID になる */
const int *ptr1_for_pid = &pid_data1;   /* PID のアドレスで静的初期化: エラー */
const int pid_data4[] = {1,2,3,4};   /* PID になる */
const int *ptr2_for_pid = pid_data4;   /* PID のアドレスで静的初期化: エラー */
```

PID 機能を利用するアプリケーションプログラムのスタートアップを作成する際は、「8.3 スタートアップ」ではなく、「8.4.7 アプリケーションのスタートアップ」を参照ください。

pid オプション選択時は、同一の外部変数は、必ずファイル間で const 修飾を統一してください。これは、pid オプションは const 修飾のある変数を PID にするためです。もし const 修飾の統一が不明な外部変数がある場合は、pid オプション(PID 機能)は使用しないでください。

pid オプション選択時に、ファイル指定のある map オプションを有効にした場合、ファイル間で同じ外部変数に対して const 修飾が統一されていない外部参照変数に対して警告 C1805(W) または C1806(W)を表示することがあります。C1805(W)は const 修飾した外部参照変数が定数領域でなかった場合、C1806(W)は const 修飾されていない外部参照変数が定数領域だった場合です。いずれの場合も表示された変数は PID として扱います。

C++または EC++コンパイル時は、pid オプションを選択できません。選択すると、警告 C1801(W)を表示して pid の選択を無効とします。

pid オプション選択時は、base=rom=<レジスタ> を選択できません。選択すると、警告 C1801(W)を表示して base=rom=<レジスタ>の選択を無効とします。

pid オプションで選択された PID レジスタが、base オプションでも選択された場合はエラー C2028(E)になります。
pid オプションと nouse_pid_register オプションを同時に選択するとエラーC3305(F)になります。

アプリケーションおよび PID 機能の内容については、「8.4 PIC/PID 機能の利用」を参照してください。

nouse_pid_register

書　式　nouse_pid_register

説　明　PID レジスタを使用せずにコード生成を行います。
PID レジスタについては、pid オプションの項目を参照してください。

PID 機能が有効なアプリケーションプログラムから呼び出されるマスタプログラムは、本オプションでコンパイルする必要があります。このとき、アプリケーションに fint_register オプションの選択がある場合は、マスタプログラムにも同じパラメータの fint_register を選択してください。

なお、本オプションを選択しても PID 機能は有効になりません。

備　考　nouse_pid_register オプションと pid オプションを同時に選択するとエラーC3305(F)になります。

nouse_pid_register オプションで選択されたレジスタが、base オプションでも選択された場合はエラーC2028(E)になります。

マスタ、アプリケーションおよび PID 機能の内容については、「8.4 PIC/PID 機能の利用」を参照してください。

save_acc

書　式　save_acc

説　明　割り込み関数に対して、アキュムレータ(ACC)の退避・回復コードを生成します。

備　考　生成される退避・回復コードは、#pragma interrupt に acc を選択したときに生成されるコードと同じものです。実際の退避・回復コードは、「9.2.1 #pragma キーワード」の#pragma interrupt (acc、noacc)の項目を参照ください。

2.6 アセンブル、リンクオプション

表 2.14 アセンブル、リンクオプション一覧

No.	オプション	ダイアログメニュー	内容
1	asmcmd=<ファイル名>	−	asrx のオプションをサブコマンドファイルで指定する
2	lnkcmd=<ファイル名>	−	optlnk のオプションをサブコマンドファイルで指定する
3	asmopt=['']<アセンブラオプション>['']	−	asrx のオプションを指定する
4	lnkopt=['']<リンクオプション>['']	−	optlnk のオプションを指定する

asmcmd

書　式　　asmcmd=<ファイル名>

説　明　　asrx に渡すアセンブラオプションをサブコマンドファイルにより指定します。

例　　ccrx –cpu=rx600 –asmcmd=file.sub sample.c
と記述した場合、以下の 2 行のコマンド記述と同じ意味になります。
ccrx –cpu=rx600 –output=src sample.c
asrx –cpu=rx600 –subcommand=file.sub sample.src

備　考　　本オプションを複数回指定した場合、指定した全てのサブコマンドファイルが有効となります。

lnkcmd

書　式　　lnkcmd=<ファイル名>

説　明　　optlnk に渡すリンクオプションをサブコマンドファイルにより指定します。

例　　ccrx -cpu=rx600 -output=abs=tp.abs –lnkcmd=file.sub tp1.c tp2.c
と記述した場合、以下の 3 行のコマンド記述と同じ意味になります。
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx -cpu=rx600 tp1.src tp2.src
optlnk –subcommand=file.sub -form=abs -output=tp tp1.obj tp2.obj

備　考　　本オプションを複数回指定した場合、指定した全てのサブコマンドファイルが有効となります。

asmopt

書　式　　asmopt=["]＜アセンブラオプション＞["]

説　明　　asrx に渡すアセンブラオプションを文字列により指定します。
空白をパラメータに含むオプションを指定する場合は、ダブルクォーテーション(")で囲んで指定します。

例　　ccrx –cpu=rx600 –asmopt="–chkpm" sample.c
と記述した場合、以下の 2 行のコマンド記述と同じ意味になります。
ccrx –cpu=rx600 –output=src sample.c
asrx –cpu=rx600 –chkpm sample.src

備　考　　本オプションを複数回指定した場合、指定した全てのアセンブラオプションが有効となります。

lnkopt

書　式　lnkopt=["]＜リンクオプション＞["]

説　明　optlnk に渡すリンクオプションを文字列により指定します。
空白をパラメータに含むオプションを指定する場合は、ダブルクォーテーション(")で囲んで指定します。

例　ccrx -cpu=rx600 -output=abs=tp.abs -lnkopt="-start=P,C,D/100,B/8000" tp1.c tp2.c
と記述した場合、以下の 3 行のコマンド記述と同じ意味になります。

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx -cpu=rx600 tp1.src tp2.src
optlnk -start=P,C,D/100,B/8000 -form=abs -output=tp tp1.obj tp2.obj
```

備　考　本オプションを複数回指定した場合、指定した全てのリンクオプションが有効となります。

2.7 その他のオプション

表 2.15 その他オプション一覧

No.	オプション	ダイアログメニュー	内容
1	logo nologo	－ (常に nologo が有効)	コピーライトを出力 コピーライトの出力を抑止
2	 euc sjis latin1 utf8	コンパイラ <ソース> [オプション項目 :] [ソースファイル] [入力文字コード :]	入力プログラムの文字コードを指定 EUC コード SJIS コード ISO-Latin1 コード UTF-8 コード
3	outcode = { euc | sjis | utf8 }	コンパイラ <オブジェクト> [出力文字コード :]	出力アセンブリ言語ファイルの文字コードを指定 EUC コード SJIS コード UTF-8 コード
4	subcommand = <ファイル名>	－	<ファイル名>で指定したファイルからコマンドオプションを取り込む

logo, nologo

書　式　　logo
nologo

説　明　　コピーライトの出力を抑止します。
logo オプション指定時は、コピーライト表示が出力されます。
nologo オプション指定時は、コピーライトの表示の出力が抑止されます。
本オプションの省略時解釈は、logo です。

euc, sjis, latin1, utf8

書　式　　euc
sjis
latin1
utf8

説　明　　文字列、文字定数およびコメント内の文字を指定した文字コードで扱います。
オプションと文字コードの関係を以下に示します。
本オプションの省略時解釈は、sjis です。

表 2.16　オプションと文字コードの関係（euc, sjis, latin1, utf8）

オプション	文字コード
euc	EUC コード
sjis	SJIS コード
latin1	ISO-Latin1 コード
utf8	UTF-8 コード

備　考　　utf8 オプションは、lang=c99 オプション指定時のみ有効です。

outcode

書　式　　outcode = { euc | sjis | utf8 }

説　明　　文字列、文字定数内の文字を指定した文字コードで出力します。
オプションと文字コードの関係を以下に示します。
本オプションの省略時解釈は、outcode=sjis です。

表 2.17　オプションと文字コードの関係（outcode）

オプション	文字コード
euc	EUC コード
sjis	SJIS コード
utf8	UTF-8 コード

備　考　　utf8 オプションは、lang=c99 オプション指定時のみ有効です。

subcommand

書　式　　subcommand=<サブコマンドファイル名>

説　明　　subcommand オプション指定時は、コンパイラ起動時のコンパイラオプションをサブコマンドファイルで指定します。サブコマンドファイル中の書式は、コマンドラインの書式と同一です。

備　考　　本オプションを複数回指定した場合、指定した全てのサブコマンドファイルが有効となります。

3. ライブラリジェネレータオプション

3.1 ライブラリオプション

表 3.1 ライブラリジェネレータオプション一覧

No.	オプション	ダイアログメニュー	内容
1	head=<sub>[,…] <sub>:{ all | runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios | new | complex | cppstring | c99_complex | fenv | inttypes | wchar | wctype }	標準ライブラリ <標準ライブラリ> [カテゴリ :]	構築対象のライブラリを指定 すべてのライブラリ関数とランタイムライブラリ ランタイムライブラリ ctype.h(C89/C99)とランタイムライブラリ math.h(C89/C99)とランタイムライブラリ mathf.h(C89/C99)とランタイムライブラリ stdarg.h(C89/C99)とランタイムライブラリ stdio.h(C89/C99)とランタイムライブラリ stdlib.h(C89/C99)とランタイムライブラリ string.h(C89/C99)とランタイムライブラリ ios(EC++)とランタイムライブラリ new(EC++)とランタイムライブラリ complex(EC++)とランタイムライブラリ string(EC++)とランタイムライブラリ complex.h(C99)とランタイムライブラリ fenv.h(C99)とランタイムライブラリ inttypes.h(C99)とランタイムライブラリ wchar.h(C99)とランタイムライブラリ wctype.h(C99)とランタイムライブラリ
2	output=<ファイル名>	標準ライブラリ <オブジェクト> [出力ファイル :]	出力ライブラリファイル名を指定
3	nofloat	標準ライブラリ <オブジェクト> [簡易入出力関数の生成:] [機能縮小版 1]	簡易入出力関数の生成

No.	オプション	ダイアログメニュー	内容
4	reent	標準ライブラリ <オブジェクト> [リエントラントライブラリを生成]	リエントラントライブラリを生成
5	lang = { c | c99 }	標準ライブラリ <標準ライブラリ> [ライブラリ構成 :] [C(C89)] [C99]	使用可能な C 言語標準ライブラリ関数構成の選択
6	simple_stdio	標準ライブラリ <オブジェクト> [簡易入出力関数の生成:] [機能縮小版 2]	機能縮小版入出力関数の生成
7	logo nologo	－ (常に nologo が有効)	コピーライトを出力 コピーライトの出力を抑止

head

書　式　　head=<sub>[,…]

<sub>:{ all

| runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios | new

| complex | cppstring | c99_complex | fenv | inttypes | wchar | wctype

}

説　明　　構築対象をヘッダファイル名で指定します。

head=all を指定した場合、全てのヘッダファイル名が構築対象として指定されます。

ランタイムライブラリは常に構築対象になります。

本オプションの省略時解釈は、head=all です。

output

書　式　output=<ファイル名>

説　明　出力ファイル名を指定します。
本オプションの省略時解釈は、output=stdlib.lib です。

nofloat

書　式　nofloat

説　明　浮動小数点変換(%f、%e、%E、%g、%G)をサポートしない、簡易入出力関数を生成します。
浮動小数点変換を必要としないファイル入出力を行う場合、ROM サイズを削減することができます。

対象関数　fprintf、fscanf、printf、scanf、sprintf、sscanf、vfprintf、vprintf、vsprintf

備　考　本オプションを指定して作成したライブラリでは、対象関数で浮動小数点数の入出力をした場合の動作は保証しません。

reent

書　式　　reent

説　明　　リエントラントライブラリを生成します。ただし、rand、srand 関数はリエントラントではありません。

備　考　　リエントラントライブラリをリンクする場合は、プログラム内で標準インクルードファイルをインクルードする前に_REENTRANT というマクロ名を#define で定義するか、コンパイル時に define オプションで_REENTRANT を定義してください。

lang

書　式　　lang = { c | c99 }

説　明　　使用可能な C 言語標準ライブラリ関数の構成を選択します。
lang=c を選択すると、C 言語の標準関数を、C89 規格準拠のものだけで構成し、C99 規格で拡張された関数を含めません。lang=c99 を選択すると、C 言語の標準関数を、C89 規格および C99 規格準拠の内容で構成します。
本オプションの省略時解釈は、lang=c です。

備　考　　C++,EC++ライブラリの標準関数の構成は変化しません。
lang=c99 を指定すると、C99 規格を含めた全ての関数が使用できますが、lang=c 指定時に比べて関数の数が多いため、ライブラリ生成に多くの時間が必要になる場合があります。

simple_stdio

書　式　simple_stdio

説　明　機能縮小版の入出力関数を生成します。

機能縮小版では、浮動小数点の変換(nofloat オプションでサポートされない機能と同じ)、long long 型の変換、2 バイトコードの変換が含まれません。これらの機能を必要としないファイル入出力を行う場合、ROM サイズを削減することができます。

対象関数　fprintf、fscanf、printf、scanf、sprintf、sscanf、vfprintf、vprintf、vsprintf

備　考　対象関数で縮小された機能を使用した場合、本オプションを指定して作成したライブラリをリンクした時の動作は保証しません。

本機能は、C++または EC++コンパイル時は無効です。

logo, nologo

書　式　logo

nologo

説　明　コピーライトを出力または抑止します。

logo オプション指定時は、コピーライト表示が出力されます。

nologo オプション指定時は、コピーライトの表示の出力が抑止されます。

本オプションの省略時解釈は、logo です。

3.2 無効となるコンパイラオプション

ライブラリジェネレータでは、「3.1 ライブラリオプション」の他に C/C++コンパイラオプションを指定し、ライブラリをコンパイルするときに用いるオプションとして選択することができます。ただし、以下に示すオプションは無効となり、ライブラリのコンパイルでは選択されません。

表 3.2 無効オプション一覧

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
1	lang	常に無効	なし
2	include	常に無効	なし
3	define	常に無効	なし
4	undefined	常に無効	なし
5	message nomessage	常に無効	nomessage
6	change_message	常に無効	なし
7	file_inline_path	常に無効	なし
8	comment	常に無効	なし
9	check	常に無効	なし
10	output	常に無効	output=obj
11	noline	常に無効	なし
12	debug nodebug	常に無効	nodebug
13	object noobject	常に無効	なし
14	listfile nolistfile show	常に無効	nolistfile
15	file_inline	常に無効	なし
16	asmcmd	常に無効	なし
17	lnkcmd	常に無効	なし
18	asmopt	常に無効	なし
19	lnkopt	常に無効	なし
20	logo nologo	常に無効	nologo
21	euc sjis latin1 utf8	常に無効	なし
22	outcode	常に無効	なし

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
23	subcommand	常に無効	なし
24	alias	常に無効	alias=noansi
25	pic pid	lang=cpp または C++ ソースコンパイル時*	なし

【注】 * 警告 C1801(W)が表示されます。

4. アセンブラオプション

4.1 ソースオプション

表 4.1 ソースオプション一覧

No.	オプション	ダイアログメニュー	内容
1	include = <パス名>[,...]	アセンブラ<ソース> [オプション項目 :] [インクルードファイルディレクトリ]	インクルードファイルの取り込み先パス名を指定
2	define = <sub>[,...] <sub>:<マクロ名>=<文字列>	アセンブラ<ソース> [オプション項目 :] [シンボル定義]	<文字列>を<マクロ名>として定義
3	chkpm	アセンブラ<その他> [その他のオプション :]	特権命令のチェック
4	chkfpu	アセンブラ<その他> [その他のオプション :]	浮動小数点演算命令のチェック
5	chkdsp	アセンブラ<その他> [その他のオプション :]	DSP 機能命令のチェック

include

書 式 include=<パス名>[,…]

説 明 インクルードファイルの存在するパス名を指定します。
パス名が複数ある場合にはカンマ(,)で区切って指定することができます。
インクルードファイルの検索は、カレントフォルダ、include オプション指定フォルダ、環境変数 INC_RXA 指定フォルダの順序で行います。

例 asrx -include=c:¥usr¥inc,c:¥usr¥rxc test.src
フォルダ c:¥usr¥inc と c:¥usr¥rxc をインクルードファイルパスとして検索します。

define

書　式　define=<sub>[,…]

<sub> : <マクロ名> = <文字列>

説　明　マクロ名を対応する文字列に置き換えます。

(ソースファイル先頭で.DEFINE 指示命令を記述するのと同じです)

備　考　define オプションと.DEFINE が同時指定された場合、.DEFINE を優先します。

chkpm

書　式　chkpm

説　明　本オプションを指定した場合、特権命令を記述するとウォーニング A1011 を通知します。

備　考　特権命令の詳細な説明については、**「RX ファミリ ソフトウェアマニュアル」**を参照してください。

chkfpu

書　式　chkfpu

説　明　本オプションを指定した場合、浮動小数点演算命令を記述するとウォーニング A1012 を通知します。

備　考　浮動小数点演算命令の詳細な説明については、**「RX ファミリ ソフトウェアマニュアル」**を参照してください。

chkdsp

書　式　　chkdsp

説　明　　本オプションを指定した場合、DSP 機能命令を記述するとウォーニング A1013 を通知します。

備　考　　DSP 機能命令の詳細な説明については、**「RX ファミリ ソフトウェアマニュアル」**を参照してください。

4.2 オブジェクトオプション

表 4.2 オブジェクトオプション一覧

No.	オプション	ダイアログメニュー	内容
1	output=<出力ファイル名>	アセンブラ<オブジェクト> [出力ディレクトリ :]	リロケータブルファイルの指定
2	debug nodebug	アセンブラ<オブジェクト> [デバッグ情報出力]	デバッグ情報あり デバッグ情報なし
3	goptimize	アセンブラ <オブジェクト> [モジュール間最適化]	モジュール間最適化用付加情報出力

output

書　式　　output=<出力ファイル名>

説　明　　出力するリロケータブルファイル名を指定します。

出力するリロケータブルファイル名は、出力ファイル名に拡張子がない場合、出力ファイル名に拡張子「.obj」を付加した文字列となり、拡張子がある場合、出力ファイル名の拡張子を「.obj」で置き換えた文字列となります。

本オプションを指定しない場合には、ソースファイルと同じファイル名で拡張子が「obj」のリロケータブルファイルを出力します。

debug, nodebug

書　式　　debug

nodebug

説　明　　debug オプションを指定した場合、リロケータブルファイル内にデバッグ情報を出力します。

nodebug オプションを指定した場合、リロケータブルファイル内にデバッグ情報を出力しません。

本オプションの省略時解釈は、nodebug です。

goptimize

書　式　　goptimize

説　明　　モジュール間最適化用付加情報を出力します。
本オプションを指定したファイルは、リンク時にモジュール間最適化の対象になります。

4.3 リストオプション

表 4.3 リストオプション一覧

No.	オプション	ダイアログメニュー	内容
1	listfile[=<ファイル名>] nolistfile	アセンブラ<リスト> [アセンブルリスト出力]	アセンブルリストファイル出力あり アセンブルリストファイル出力なし
2	show = <sub>[,...] <sub>:{ conditionals | definitions | expansions }	アセンブラ<リスト> [アセンブルリスト出力] [ソースプログラム :]	アセンブルリスト出力内容の設定 条件アセンブルで偽となった行も出力 .DEFINE で置き換える以前の情報で出力 マクロ記述の展開行を出力

listfile, nolistfile

書　式　　listfile[=<ファイル名>]

nolistfile

説　明　　アセンブルリストファイルの出力有無を指定します。

listfile オプションを指定した場合、アセンブルリストファイルを出力します。＜ファイル名＞を指定することもできます。

nolistfile オプションを指定した場合、アセンブルリストファイルは出力しません。

＜ファイル名＞は、**「7.1 ファイル名の付け方」**に従って指定できます。

listfile オプションで＜ファイル名＞を指定しない場合には、ソースファイルと同じファイル名で、拡張子が「lst」のアセンブルリストファイルが作成されます。

本オプションの省略時解釈は、nolistfile です。

show

書　式　　show=<sub>[,…]

<sub> : { conditionals

| definitions

| expansions }

説　明　　アセンブラが出力するリスト内容の設定を行います。各指定をした場合に出力される内容は以下の通りです。

表 4.4　show オプション指定一覧

出力種別	内容
conditionals	条件アセンブルで条件が偽となった行もアセンブルリストファイルに出力します。
definitions	.DEFINE で置き換える以前の情報でアセンブルリストファイルに出力します。
expansions	マクロ記述展開行をアセンブルリストファイルに出力します。

4.4 マイコンオプション

表 4.5 マイコンオプション一覧

No.	オプション	ダイアログメニュー	内容
1	cpu = { rx600 | rx200 }	CPU [CPU 種別 :]	RX600 向けのリロケータブルファイルを生成 RX200 向けのリロケータブルファイルを生成
2	endian = { big | little }	CPU [エンディアン :]	Big Endian Little Endian
3	fint_register = { 0 | 1 | 2 | 3 | 4 }	CPU [高速割り込みレジスタ :]	高速割り込みでのみ使用する汎用レジスタを指定 高速割り込み専用のレジスタはなし R13 を高速割り込み専用で使用 R13～R12 を高速割り込み専用で使用 R13～R11 を高速割り込み専用で使用 R13～R10 を高速割り込み専用で使用
4	base = { rom = <レジスタ> | ram= <レジスタ> | <アドレス値> = <レジスタ> }	CPU [ベースレジスタ :]	ROM 用ベースレジスタ指定 RAM 用ベースレジスタ指定 SFR 用ベースレジスタ指定
5	patch = { rx610 }	CPU [CPU タイプ特有の問題を回避 :]	CPU の品種ごとに特有の問題を回避する MVTIPL 命令を使用しない (RX610 グループ向け)
6	pic	CPU [詳細…] [PIC/PID] [コードセクションを位置独立コードとして生成]	PIC 機能が有効なオブジェクトを生成する

No.	オプション	ダイアログメニュー	内容
7	pid = { 16 | 32 }	CPU [詳細…] [PIC/PID] [コードセクションを位置独立コードとして生成] [オフセットの最大幅:] [16 ビット] [32 ビット]	PID 機能が有効なオブジェクトの生成と、オフセット幅を選択する 16 ビット(64KB～256KB)のアドレッシング対応 32 ビット(4GB)のアドレッシング対応
8	nouse_pid_register	CPU [詳細…] [PIC/PID] [PID 用レジスタを使用しない]	PID レジスタをコード生成に使用しない

cpu

書　式　cpu={ rx600 | rx200 }

説　明　生成する命令コードのマイコン種別を指定します。
cpu=rx600 を指定した場合、RX600 向けのリロケータブルファイルを生成します。
cpu=rx200 を指定した場合、RX200 向けのリロケータブルファイルを生成します。

備　考　サブオプションは今後のマイコン製品展開に応じて追加されます。
rx200 を指定した場合、RX200 でサポートされていない浮動小数点演算命令(FADD, FCMP, FDIV, FMUL, FSUB, FTOI, ITOF, ROUND)の記述および MVTC, MVFC, PUSHC または POPC 命令の制御レジスタにおける FPSW の記述をエラーとします。

endian

書　式　endian={ big　|　little }

説　明　endian=big を指定した場合、データのバイト並びが BigEndian になります。
endian=little を指定した場合、データのバイト並びが LittleEndian になります。
本オプションの省略時解釈は、endian=little です。

fint_register

書　式　fint_register = {0 | 1 | 2 | 3 | 4 }

説　明　コンパイラの同名のオプションで指定された、高速割り込み専用で使用する汎用レジスタの情報を、リロケータブルファイルに出力します。

備　考　本オプションは、プロジェクト全体で指定を統一してください。指定が異なる場合の動作は保証しません。
高速割り込み専用に指定した汎用レジスタを、アセンブリ言語ファイルにおいて、高速割り込み以外の用途で使用しないでください。使用した場合の動作は保証しません。
本オプションの指定の対象となるレジスタが、base オプションで指定されていた場合、エラーとなります。

base

書　式　　base = | rom=<レジスタ>
| ram=<レジスタ>
| <アドレス値> = <レジスタ>}
<レジスタ>:= {R8 ～ R13}

説　明　　コンパイラの同名のオプションで指定された、ベースアドレスとして固定で使用する汎用レジスタの情報を、リロケータブルファイルに出力します。

備　考　　本オプションは、プロジェクト全体で指定を統一してください。指定が異なる場合の動作は保証しません。
本オプションで指定した汎用レジスタをベースレジスタ以外の用途に使用しないでください。使用した場合の動作は保証しません。
異なる領域に対して同じ汎用レジスタを指定した場合はエラーとなります。
fint_register オプションで指定した汎用レジスタを本オプションで指定した場合はエラーとなります。

patch

書　式　　patch = { rx610 }

説　明　　CPU の品種ごとに特有の問題を回避します。

-patch=rx610 を指定した場合、RX610 グループで問題となる MVTIPL 命令を未定義命令として扱います。MVTIPL は命令と認識されずにエラーメッセージ A2113 (E)が出力されます。

pic

書　式　　pic

説　明　　PIC 機能が有効な状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。

備　考　　本オプションに矛盾するコードをアセンブリコードに記述しても、チェックされません。

PIC 機能が有効なリロケータブルオブジェクトは、PIC 機能が有効でないリロケータブルオブジェクトとはリンクできません。

PIC 機能については、「8.4 PIC/PID 機能の利用」の項目も参照してください。

pid

書　式　　pid[={16|32}]

説　明　　PID 機能が有効な状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。

＜PID レジスタ＞

下記の表の規則に基づき、fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお、fint_register の指定がない場合は R13 を選択します。

表 4.6　fint_register オプションと PID レジスタの関係

fint_register オプション	PID レジスタ
fint_register 指定なし	R13
fint_register=0	
fint_register=1	R12
fint_register=2	R11
fint_register=3	R10
fint_register=4	R9

PID レジスタは、PID のアクセスに使用する用途以外には使用されません。

＜パラメータ＞

パラメータの意味は、コンパイラの同名オプションと同じです。

備　考　本オプションに矛盾するコードをアセンブリコードに記述しても、チェックされません。

PID 機能が有効なリロケータブルオブジェクトは、PID 機能が有効でないリロケータブルオブジェクトとはリンクできません。

pid オプションで選択された PID レジスタが、base オプションでも選択された場合はエラーA3111(F)になります。

pid オプションと nouse_pid_register オプションを同時に選択するとエラーA3103(F)になります。

PID 機能については、「8.4 PIC/PID 機能の利用」の項目も参照してください。

nouse_pid_register

書　式　nouse_pid_register

説　明　PID レジスタを使用しないでコード生成されたことを表すリロケータブルオブジェクトを生成します。

アセンブリソースファイル中で PID レジスタを使用している場合に、エラーとして A2058 (E) メッセージを出力します。

PID 機能が有効なアプリケーションプログラムから呼び出されるマスタプログラムは、本オプションでアセンブルする必要があります。このとき、アプリケーションに fint_register オプションの選択がある場合は、マスタプログラムにも同じパラメータの fint_register を選択してください。

備　考　nouse_pid_register オプションと pid オプションを同時に選択するとエラーA3103(F)になります。

nouse_pid_register オプションで選択されたレジスタが、base オプションでも選択された場合はエラーA3112(F)になります。

PID 機能の詳細については、「8.4 PIC/PID 機能の利用」の項目を参照してください。

4.5 その他のオプション

表 4.7 その他オプション一覧

No.	オプション	ダイアログメニュー	内容
1	logo nologo	－ (常に nologo が有効)	コピーライトを出力 コピーライトの出力を抑止
2	subcommand = <ファイル名>	－	コマンドラインをファイルから入力
3	euc sjis latin1	－	EUC コードを選択 SJIS コードを選択 ISO-Latin1 コードを選択

logo, nologo

書　式　　logo
nologo

説　明　　コピーライトの出力を抑止します。
logo オプション指定時は、コピーライト表示が出力されます。
nologo オプション指定時は、コピーライトの表示の出力が抑止されます。
本オプション省略時解釈は、logo です。

subcommand

書　式　　subcommand=<ファイル名>

説　明　　subcommand オプション指定時は、アセンブラ起動時のアセンブラオプションをサブコマンドファイルで指定します。サブコマンドファイル中の書式は、コマンドラインの書式と同一です。

例　　＜サブコマンドファイル opt.sub の内容＞

-listfile

-debug

＜コマンドライン指定＞

(1)のコマンドライン指定を行うと、アセンブラで(2)のように解釈されます。

(1) asrx –endian=big –subcommand=opt.sub test.src

(2) asrx -endian=big -listfile –debug test.src

euc, sjis, latin1

書　式　　euc

sjis

latin1

説　明　　文字列、文字定数およびコメント内の文字を指定した文字コードで扱います。

オプションと文字コードの関係を以下に示します。

表 4.8　オプションと文字コードの関係（euc, sjis, latin1）

オプション	文字コード
euc	EUC コード
sjis	SJIS コード
latin1	ISO-Latin1 コード

5. 最適化リンケージエディタ操作方法

5.1 オプション指定規則

5.1.1 コマンドラインの形式

コマンドラインの形式は以下のとおりです。

```
optlnk [ {△<ファイル名> | △<オプション列>}...]
  <オプション列>：-<オプション>[= <サブオプション>[,...]]
```

5.1.2 サブコマンドファイルの形式

サブコマンドファイルの形式は以下のとおりです。

```
<オプション> {= | △}[<サブオプション>[,...]] [△&] [;<コメント>]
  &：継続行指定
```

サブコマンドファイル形式の詳細は、**「5.2.8 サブコマンドファイルオプション」**を参照してください。

5.2 オプション解説

オプション、サブオプションの英大文字は短縮形指定時の文字を、下線は省略時解釈を示します。

また、統合開発環境の対応するダイアログメニューを、タブ名<カテゴリ名>[項目]…で示します。オプションの順序は、統合開発環境のタブとその中のカテゴリに対応しています。

ファイル名、パス名には、括弧記号"("および")"を含まないようにしてください。

5.2.1 入力オプション

表 5.1　入力カテゴリオプション一覧

	項目	オプション	ダイアログメニュー	指定内容
1	入力ファイル	Input = <sub> [{,|△}...] <sub>: <ファイル名> [(<モジュール名>[,...])]	リンカ<入力> [オプション項目 :] [リロケータブルファイル/オブジェクトファイル]	入力ファイルを指定 (コマンドラインでは input なしで指定します)
2	ライブラリファイル	LIBrary = <ファイル名>[,…]	リンカ<入力> [オプション項目 :] [ライブラリファイル]	入力ライブラリファイルを指定
3	バイナリファイル	Binary = <sub>[,...] <sub> : <ファイル名> (<セクション名> [:<アライメント数>] [/<セクション属性>] [,<シンボル名>])	リンカ<入力> [オプション項目 :] [バイナリファイル]	入力バイナリファイルを指定
4	シンボル定義	DEFine = <sub>[,...] <sub>: <シンボル名> = {<シンボル名> | <数値> }	リンカ<入力> [オプション項目 :] [シンボル定義]	未定義シンボルの強制定義 シンボル名と同値として定義 数値で定義
5	実行開始アドレス	ENTry = { <シンボル名> | <アドレス> }	リンカ<入力> [エントリポイント :]	エントリシンボルを指定 エントリアドレスを指定
6	プレリンカ	NOPRElink	リンカ<入力> [プレリンカ制御 :]	プレリンカの起動を抑止

入力ファイル

Input

リンカ<入力>[オプション項目 :][リロケータブルファイル/オブジェクトファイル]

書　式　　Input = <サブオプション>[{,|△}…]
<サブオプション> : <ファイル名>[(<モジュール名>[,…])]

説　明　　入力ファイルを指定します。複数ある場合にはカンマ(,)または空白文字で区切って指定します。
ワイルドカード(*,?)も指定できます。ワイルドカードで指定した文字列はアルファベット順に展開します。数字と英文字は数字が先、英大文字と英小文字は英大文字が先になります。
入力ファイルとして指定できるのは、コンパイラ、アセンブラ出力オブジェクトファイル、最適化リンケージエディタ出力のリロケータブルファイルおよびアブソリュートファイルです。またライブラリ名(<モジュール名>)の形式で、ライブラリ内モジュールを入力ファイルとして指定することもできます。モジュール名は拡張子なしで指定します。
入力ファイル名に拡張子の指定がない場合、モジュール名がない時は「obj」、モジュール名がある時は「lib」を仮定します。

例　　input=a.obj lib1(e)　　; a.obj と lib1.lib 内のモジュール e を入力します
input=c*.obj　　; c で始まる拡張子 obj のファイルを全て入力します

備　考　　form=object および extract 指定時、本オプションは無効です。
コマンドライン上で入力ファイルを指定する場合は、input 無しで指定します。

ライブラリファイル

LIBrary

リンカ<入力>[オプション項目 :][ライブラリファイル]

書　式　　LIBrary = <ファイル名>[,…]

説　明　　ライブラリファイルを指定します。複数ある場合にはカンマ(,)で区切って指定します。
ワイルドカード(*,?)も指定できます。ワイルドカードで指定した文字列はアルファベット順に展開します。数字と英文字は数字が先、英大文字と英小文字は英大文字が先になります。
入力ファイル名に拡張子の指定がない場合は、「lib」を仮定します。
form=library オプションまたは extract オプション指定時は、ライブラリファイルを編集対象ライブラリとして入力します。
それ以外の場合は、入力ファイルとして指定されたファイル間でのリンケージ処理後に、未定義シンボルをライブラリファイルから検索します。
ライブラリファイル内シンボルの検索は、ライブラリオプション指定ユーザライブラリファイル(指定順)、ライブラリオプション指定システムライブラリファイル(指定順)、デフォルトライブラリ(環境変数 HLNK_LIBRARY1,2,3)の順序で行います。

例　　library=a.lib,b　　; a.lib と b.lib を入力します。
library=c*.lib　　; c で始まる拡張子 lib のファイルを全て入力します。

バイナリファイル

Binary

リンカ<入力>[オプション項目 :][バイナリファイル]

書　式　Binary = <サブオプション>[,…]
<サブオプション> : <ファイル名>(<セクション名>
[:<アライメント数>][/<セクション属性>][,<シンボル名>])
<セクション属性> : CODE | DATA
<アライメント数> : 1 | 2 | 4 | 8 | 16 | 32 (デフォルトは 1)

説　明　入力バイナリファイルを指定します。複数ある場合にはカンマ(,)で区切って指定します。
ファイル名に拡張子の指定がない場合は、「bin」を仮定します。
入力したバイナリデータは、指定したセクションのデータとして配置します。セクションのアドレスは start オプションで指定します。セクションは省略できません。
またシンボルを指定することにより、定義シンボルとしてリンクすることもできます。C/C++プログラムで参照している変数名の場合、プログラム中での参照名先頭に_を付加します。
本オプションで指定したセクションには、セクション属性、アライメント数の指定が可能です。
セクション属性は、CODE または DATA を指定できます。
セクション属性の指定が無い場合、デフォルトとして書き込み、読み取り、実行全ての属性が有効になります。
アライメント数に指定可能な値は 2 の累乗です。それ以外の値を指定することはできません。
アライメント数の指定がない場合、デフォルト値として 1 が有効になります。

例
```
input=a.obj
start=P,D*/200
binary=b.bin(D1bin),c.bin(D2bin:4,_datab)
form=absolute
```

b.bin を D1bin セクションとして、0x200 番地から配置します。
c.bin を D2bin セクション(アライメント数 4)として、D1bin の後に配置します。
c.bin データを定義シンボル_datab としてリンクします。

備　考　form={object | library}または strip 指定時、本オプションは無効です。
また入力オブジェクトファイル指定がない場合、本オプションは指定できません。

シンボル定義

DEFine

リンカ<入力>[オプション項目 :][シンボル定義]

書　式　DEFine = <サブオプション>[,…]
<サブオプション> : <シンボル名> = {<シンボル名> | <数値>}

説　明　未定義シンボルを外部定義シンボルまたは数値で強制定義します。
数値は 16 進数で指定します。先頭が A～F の場合は先にシンボルを検索し、該当するシンボルがなければ数値として解釈します。先頭に 0 を付加した場合は常に数値と解釈します。シンボル名が C/C++変数名の場合、プログラム中での定義名先頭に_を付加します。C++関数名の場合は(main 関数は除く)引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし引数が void の場合は、"関数名()"で指定します。

例　define=_sym1=data　　;_sym1 を外部定義シンボル data と同値として定義します。
　　define=_sym2=4000　　;_sym2 を 0x4000 として定義します。

備　考　form={object | relocate | library}指定時、本オプションは無効です。

実行開始アドレス

ENTry

リンカ<入力>[エントリポイント :]

書　式　ENTry = {<シンボル名> | <アドレス>}

説　明　実行開始アドレスを外部定義シンボルまたはアドレスで指定します。
アドレスは 16 進数で指定します。先頭が A～F の場合は先に定義シンボルを検索し、該当するシンボルがなければアドレスと判断します。先頭に 0 を付加した場合は常にアドレスと解釈します。
シンボル名は、C 関数名の場合プログラム中での定義名先頭に_を付加します。C++関数名の場合は(main 関数は除く)引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし引数が void の場合は、"関数名()"で指定します。
コンパイル、アセンブル時に entry シンボルを指定している場合、本オプション指定を優先します。

例　entry=_main　　;C/C++の main 関数を実行開始アドレスとして設定します。
　　entry="init()"　　;C++の init 関数を実行開始アドレスとして設定します。
　　entry=100　　;0x100 を実行開始アドレスとして設定します。

備　考　form={object | relocate | library}または strip 指定時、本オプションは無効です。
未参照シンボル削除最適化(optimize=symbol_delete)指定時には、実行開始アドレスは必ず必要です。指定がない場合は、未参照シンボル削除最適化指定は無効です。本オプションでアドレスを指定している場合は、未参照シンボル削除最適化を無効にします。

プレリンカ

NOPRElink

リンカ<入力>[プレリンカ制御 :]

書　式　NOPRElink

説　明　プレリンカの起動を抑止します。
プレリンカは、C++テンプレートインスタンスの自動生成機能および実行時型検査機能をサポートします。C++テンプレート機能および実行時型検査機能を使用していない場合は、noprelink オプションを指定してください。リンク時間が短くなります。

備　考　extract または strip 指定時、本オプションは無効です。
C++テンプレート機能および実行時型検査機能を使用し、form=lib または、form=rel を指定する場合には、noprelink を指定しないでください。

5.2.2 出力オプション

表 5.2 出力カテゴリオプション一覧

	項目	オプション	ダイアログメニュー	指定内容
1	出力形式	FOrm = { Absolute | Relocate | Object | Library [= {S | U}] | Hexadecimal | Stype | Binary }	リンカ<出力> [出力形式 :]	アブソリュート形式 リロケータブル形式 オブジェクト形式 ライブラリ形式 インテル HEX 形式 モトローラ S 形式 バイナリ形式
2	デバッグ情報	DEBug SDebug NODEBug	リンカ<出力> [デバッグ情報 :]	出力あり(出力ファイル内) デバッグ情報ファイル出力 出力なし
3	レコードサイズ統一	REcord = { H16 | H20 | H32 | S1 | S2 | S3 }	リンカ<出力> [レコードサイズ統一 :]	インテル HEX レコード インテル拡張 HEX レコード インテル 32bitHEX レコード S1 レコード S2 レコード S3 レコード
4	ROM 化支援	ROm = <sub>[,...] <sub> : <ROM セクション名> =<RAM セクション名>	リンカ<出力> [オプション項目 :] [ROM から RAM へマップするセクション]	RAM に領域を確保し、シンボルを RAM 上のアドレスでリロケーション解決
5	出力ファイル	OUtput = <sub>[,…] <sub> : <ファイル名> [=<出力範囲>] <出力範囲>: { <先頭アドレス> - <終了アドレス> | <セクション名>[:...] }	リンカ<出力> [オプション項目 :] [出力ファイル/インフォメーション抑止] [出力ファイルの分割]	出力ファイルを指定 (範囲指定、分割出力可能)
6	外部シンボル割り付け情報ファイル	MAp [= <ファイル名>]	リンカ<出力> [外部シンボル割り付け情報ファイル出力]	外部シンボル割り付け情報ファイル出力を指定(SuperH ファミリ, RX ファミリ向け)
7	空きエリア出力指定	SPace [= {<数値> | Random}]	リンカ<出力> [オプション項目 :] [空きエリア出力指定] [空きエリア出力]	空きエリアへの出力値の指定
8	インフォメーションメッセージ	Message_ NOMessage [= <sub>[,...]] <sub> : <エラー番号> [- <エラー番号>]	リンカ<出力> [オプション項目 :] [出力ファイル/インフォメーション抑止] [インフォメーションレベルメッセージ抑止]	出力あり 出力なし (エラー番号、範囲指定可能)

	項目	オプション	ダイアログメニュー	指定内容
9	参照されない定義シンボルの通知	MSg_unused	リンカ<出力> [オプション項目 :] [メッセージ出力指定] [参照されない定義シンボルの通知]	1 回も参照されない定義シンボルをメッセージ出力により通知
10	セクション内データの詰め込み配置	DAta_stuff	リンカ<出力> [オプション項目 :] [セクション内データの詰め込み配置]	コンパイル単位間の空き領域を詰めてデータを配置(SuperH ファミリ, H8,H8S,H8SX ファミリ向け)
11	データレコードのバイト数指定	BYte_count=<数値>	リンカ<出力> [データレコード長 :]	データレコードの最大バイト数を指定
12	CRC 演算	CRc = <サブオプション> <サブオプション> : <出力位置>=<計算範囲>[/<多項式>][:<エンディアン>] <出力位置> : <アドレス> <計算範囲> : <先頭アドレス>-<終了アドレス>[,...] <多項式> : { CCITT | 16 } <エンディアン>： {BIG|LITTLE}	リンカ<出力> [オプション項目 :] [CRC コード]	リンク時に計算範囲の CRC(Cyclic Redundancy Check)演算を行い、計算結果を出力位置に埋め込む
13	セクション終端にパディング	PADDING	リンカ<出力> [パディング]	アライメントにあわせてセクション終端にパディングを出力
14	特定ベクタ番号のアドレス設定	VECTN=<サブオプション>[,...] <サブオプション>: <ベクタ番号>={<シンボル>|<アドレス>}	リンカ<出力> [オプション項目 :] [ベクタ] [特定ベクタ :]	可変ベクタの特定ベクタ番号へのアドレスを設定(RX ファミリ、M16C ファミリ向け)
15	可変ベクタの空き領域のアドレス設定	VECT={<シンボル>|<アドレス>}	リンカ<出力> [オプション項目 :] [ベクタ] [空きベクタ :]	可変ベクタの空き領域へのアドレスを設定(RX ファミリ、M16C ファミリ向け)
16	utl30 情報出力	UTL	リンカ<出力> [UTL 情報]	UTL30 向け情報を出力(M16C ファミリ向け)
17	ジャンプテーブル出力	JUMP_ENTRIES_FOR_PIC=<セクション名>[,...]	リンカ<出力> [ジャンプテーブル出力]	ジャンプテーブルを出力(RX ファミリの PIC 機能向け)

出力形式

FOrm

リンカ<出力>[出力形式 :]

書　式　　FOrm = {Absolute | Relocate | Object | Library[={S|U}]
| Hexadecimal | Stype | Binary}

説　明　　出力形式を指定します。
本オプションの省略時解釈は、form=absoluteです。サブオプションの一覧を**表** 5.3に示します。

表 5.3　form **オプションのサブオプション一覧**

	サブオプション名	内　容
1	absolute	**アブソリュートファイルを出力します。**
2	relocate	**リロケータブルファイルを出力します。**
3	object	**オブジェクトファイルを出力します。**extract **オプションでライブラリから** 1 **個のモジュールをオブジェクトファイルとして取り出すときに使用します。**
4	library	**ライブラリファイルを出力します。** library=s **指定時、出力ライブラリファイルをシステムライブラリとします。** library=u **指定時、出力ライブラリファイルをユーザライブラリとします。** **省略時解釈は、**library=u **です。**
5	hexadecimal	**インテル** HEX **形式ファイルを出力します。インテル** HEX **フォーマットは「**16.1.2　**インテル** HEX **形式ファイル」を参照してください。**
6	stype	**モトローラ** S **形式ファイルを出力します。モトローラ** S **フォーマットは「**16.1.1 **モトローラ** S **形式ファイル」を参照してください。**
7	binary	**バイナリファイルを出力します。**

備　考　　出力形式と入力ファイル、他オプションとの関係を**表** 5.4に示します。

表 5.4　出力形式と入力ファイル、他オプションとの関係

	出力形式	指定オプション	入力可能なファイル形式	指定可能なオプション*1
1	Absolute	strip あり	アブソリュートファイル	input, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, binary, debug/nodebug, sdebug, cpu, ps_check, start, rom, entry, output, map, hide, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, section_forbid, absolute_forbid, profile, cachesize, sbr, compress, rename, delete, define, fsymbol, stack, noprelink, memory, msg_unused, data_stuff, show=symbol, reference, xreference, jump_entries_for_pic, aligned_section
2	Relocate	extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, debug/nodebug, output, hide, rename, delete, noprelink, msg_unused, data_stuff, show=symbol, xreference
3	Object	extract あり	ライブラリファイル	library, output
4	Hexadecimal Stype Binary		オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, binary, cpu, ps_check, start, rom, entry, output, map, space, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, section_forbid, absolute_forbid, profile, cachesize, sbr, rename, delete, define, fsymbol, stack, noprelink, record, s9*2, byte_count*3, memory, msg_unused, data_stuff, show=symbol, reference, xreference, jump_entries_for_pic, aligned_section
			アブソリュートファイル	input, output, record, s9*2, byte_count*3, show=symbol, reference, xreference
5	Library	strip あり	ライブラリファイル	library, output, memory*4, show=symbol, section
		extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル	input, library, output, hide, rename, delete, replace, noprelink, memory*4, show=symbol, section

【注】 *1　message/nomessage, change_message, logo/nologo, form, list, subcommand は常に指定できます。
*2　s9 は出力形式が form=stype のときだけ指定できます。

*3 byte_count は出力形式が form= hexadecimal のときだけ指定できます。

*4 hide 指定する場合は使用できません。

デバッグ情報

DEBug
SDebug
NODEBug

リンカ<出力>[デバッグ情報 :]

書 式 DEBug
SDebug
NODEBug

説 明 debug 情報の出力有無を指定します。
debug オプションは、出力ファイル中にデバッグ情報を出力します。
sdebug オプションは、<出力ファイル名>.dbg ファイルにデバッグ情報を出力します。
nodebug オプションは、デバッグ情報を出力しません。
form=relocate 指定時に sdebug オプションを指定したときは、debug オプションと解釈します。
output オプションで複数ファイル出力を指定時に debug オプションを指定したときは、sdebug オプションと解釈して、<先頭出力ファイル名>.dbg に出力します。
本オプション省略時解釈は、debug です。

備 考 form={object | library | hexadecimal | stype | binary}、strip、または、extract 指定時、本オプションは無効です。

レコードサイズ統一

REcord

リンカ<出力>[レコードサイズ統一 :]

書 式 REcord = {H16 | H20 | H32 | S1 | S2 | S3}

説 明 アドレス範囲に関係なく、一定のデータレコードで出力します。
指定したデータレコードより大きいアドレスが存在した場合、アドレスに合わせてデータレコードを選択します。
本オプション省略時は、それぞれのアドレスに合わせて混在したデータレコードを出力します。

備 考 form=hexadecimal または stype 指定がないとき、本オプションは無効です。

ROM 化支援

ROm

リンカ<出力>[オプション項目 :][ROM から RAM へマップするセクション]

書　式　ROm = <サブオプション>[,…]
<サブオプション> : <ROM セクション名>=<RAM セクション名>

説　明　初期化データ領域の ROM 用、RAM 用領域を確保し、ROM セクション内定義シンボルを RAM セクション内アドレスになるようリロケーションします。
ROM セクションには初期値のあるリロケータブルセクションを指定します。
RAM セクションには存在しないセクションまたはサイズ 0 のリロケータブルセクションを指定します。

例　rom=D=R
start=D/100,R/8000
D セクションと同サイズの R セクションを確保し、D セクション内定義シンボルを R セクション上のアドレスでリロケーションします。

備　考　form={object | relocate | library}または strip 指定時、本オプションは無効です。

出力ファイル

OUtput

リンカ<出力>[オプション項目 :][出力ファイル/インフォメーション抑止][出力ファイルの分割]

書　式　OUtput ＝ <サブオプション>[,…]
<サブオプション> : <ファイル名>[=<出力範囲>]
<出力範囲>: {<先頭アドレス>-<終了アドレス> | <セクション名>[:…]}

説　明　出力ファイル名を指定します。form={absolute | hexadecimal | stype | binary}のときは、複数ファイルを指定できます。アドレスは 16 進数で指定します。先頭が A～F の場合は先にセクションを検索し、該当するセクションがなければアドレスと判断します。先頭に 0 を付加した場合は常にアドレスと解釈します。
本オプションの省略時解釈は、<先頭入力ファイル名>.<デフォルト拡張子>です。
デフォルト拡張子は、次のようになります。
form=absolute：「abs」、form=relocate　：「rel」、form=object：「obj」
form=library ：「lib」、form=hexadecimal：「hex」、form=stype ：「mot」
form=binaruy ：「bin」

例　output=file1.abs=0-ffff,file2.abs=10000-1ffff
0～0xffff 間を file1.abs に、0x10000～0x1ffff 間を file2.abs に出力します。

output=file1.abs=sec1:sec2,file2.abs=sec3
sec1,sec2 セクションを file1.abs に、sec3 セクションを file2.abs に出力します。

備　考　マイコン種別が RX ファミリでビッグエンディアンのときに、セクション単位で出力する場合は、セクションサイズを 4 の倍数にしてください。

外部シンボル割り付け情報ファイル出力

MAp

リンカ<出力>[外部シンボル割り付け情報ファイル出力]

書　式　　MAp [= <ファイル名>]

説　明　　コンパイラが外部変数アクセス最適化で使用する外部変数割り付け情報ファイルを出力します。
<ファイル名>を指定しなかった場合は、output オプションで指定したファイル名、もしくは先頭入力ファイル名で、拡張子が bls のファイルを出力します。
外部変数割り付け情報ファイル作成時の変数宣言順と、再コンパイル後のオブジェクトを読み込んだ時の変数宣言順が変わっている場合はエラーを出力します。

備　考　　form={absolute | hexadecimal | stype | binary}を指定した場合のみ、本オプションは有効です。
マイコン種別が SuperH ファミリおよび RX ファミリで有効です。

空きエリア出力指定

SPace

リンカ<出力>[オプション項目 :][空きエリア出力指定][空きエリア出力]

書　式　　SPace [＝ {<数値> | Random}]

説　明　　出力範囲のメモリの空き領域を、ユーザが指定するデータで充填します。
充填するデータとしては、乱数、もしくは 16 進数の数値を指定することができます。
空きエリアを埋める方法は、output オプション指定時の出力範囲指定方法によって下記のように異なります。
・出力範囲:セクション指定
　指定されたセクション間に空きが存在した場合に指定データを出力
・出力範囲:アドレス範囲指定
　指定された範囲内に空きが存在した場合に指定データを出力
出力データサイズは、1,2,4 バイト単位で有効となります。出力データサイズは space オプションに指定する 16 進数の数値で決まります。3 バイトデータを指定した場合、上位桁を 0 拡張し 4 バイトのデータとして扱われます。また、奇数桁データを指定した場合も、上位桁に 0 拡張して偶数桁入力として扱われます。
空きエリアのサイズが出力データサイズの倍数でない場合、出力できるだけ出力し、メッセージによる警告を行います。

備　考　　本オプションにてサブオプションの指定がされなかった場合は、空きエリアへの出力は行いません。
本オプションは form={ binary |stype | hexadecimal}オプションを指定した場合にのみ有効となります。
output オプションによる出力範囲指定がされなかった場合は、本オプション指定は無効となります。

Message
NOMessage

リンカ<出力>[オプション項目 :][出力ファイル/インフォメーション抑止][インフォメーションレベルメッセージ抑止]

書　式　　Message
NOMessage [= <サブオプション>[,…]]
<サブオプション> : <エラー番号>[-<エラー番号>]

説　明　　インフォメーションレベルメッセージの出力有無を指定します。
message オプション指定時は、インフォメーションレベルメッセージを出力します。
nomessage オプション指定時は、インフォメーションレベルメッセージの出力を抑止します。またエラー番号を指定すると、指定したエラー番号のメッセージ出力を抑止できます。ハイフン(-)を使用して抑止するエラー番号の範囲を指定することもできます。エラー番号としてウォーニング、エラーレベルメッセージ番号を指定した場合、change_message でインフォメーションレベルに変更したと仮定し、メッセージ出力を抑止します。
本オプションの省略時解釈は nomessage です。

例　　nomessage=4,200-203,1300
L0004 および L0200～L0203 および L1300 のメッセージ出力を抑止します。

MSg_unused

リンカ<出力>[オプション項目 :][メッセージ出力指定][参照されない定義シンボルの通知]

書　式　MSg_unused

説　明　本オプションを指定した場合、リンク処理の中で一度も参照されることのなかった外部定義シンボルを、メッセージ出力によってユーザに知らせます。

例　optlnk -msg_unused a.obj

備　考　入力ファイルが absolute 形式の場合、本オプション指定は無効です。
メッセージ出力させるためには、同時に message オプションの指定が必要です。
コンパイル時にインライン展開された関数に対してメッセージ出力する場合があります。その場合、関数定義に static 宣言することで、メッセージ出力を抑えることができます。
以下のいずれかに該当する場合、参照関係の解析が正しく行うことができず、メッセージ出力により通知される情報が不正確となります。

- アセンブル時に goptimize オプションが指定されておらず、同一ファイル内、かつ同一セクションへの分岐がある場合 (マイコンが H8,H8S,H8SX ファミリの場合のみ)
- 同一ファイル内の定数シンボルへの参照
- コンパイル時に最適化が有効で、直下の関数を呼び出す場合
- コンパイル時に外部変数アクセス最適化が有効な場合(マイコンが SuperH ファミリの場合のみ)
- ソースファイル上で#pragma tbr を記述した際にオフセット値を直接指定している場合(マイコンが SH-2A/SH2A-FPU の場合のみ)
- リンク時の最適化によって、定数やリテラルの統合が生じる場合

DAta_stuff

リンカ<出力>[オプション項目 :][セクション内データの詰め込み配置]

書　式　DAta_stuff

説　明　リンク時に、セクション内のデータを詰め込んで配置します。本オプション機能の対象となるセクションは、定数領域、初期化データ領域、未初期化データ領域です。
本オプションを指定した場合、コンパイル単位のセクションのアライメントにより生じる空き領域を詰めてリンクを行います。
ただし、データの配置順は変更しません。
本オプションを指定しない場合、コンパイル単位のセクションのアライメントに従いリンクを行います。本オプションの指定により、アライメントで生じる冗長な空き領域を詰めることができ、データセクション全体のサイズ低減が期待できます。

例

```
<tp1.c>           <tp2.c>
----------------  -------------------
long a;           char  d;
char b,c;         long  e;
                  char  f;
```

<コンパイル後のデータセクションサイズ (SuperH ファミリ用コンパイラの出力例)>
tp1.obj : 4+1+1 = 6 バイト
tp2.obj : 1+3[*]+4+1 = 9 バイト

<tp1.obj と tp2.obj、リンク後のデータセクションサイズ>
1) data_stuff 指定なし
オブジェクトファイルを各セクションのアライメントに従ってリンクします(従来処理)。
6 バイト[tp1] + 2 バイト[*] + 9 バイト[tp2] = 17 バイト

2) data_stuff 指定あり
セクション内のデータを詰めて配置させ、アライメントによる 冗長な空き領域を埋めてリンクします。
(4+1+1)バイト + 1 バイト + 1 バイト[*] + 4 バイト + 1 バイト　=　13 バイト
【注 1】 * : アライメントのために生じる空き領域
【注 2】　コンパイル後のデータセクションサイズは、コンパイル時のオプション指定などによって変化するので、上記例のようにならない場合があります。

備　考　SuperH ファミリ用コンパイラの smap オプションを指定したオブジェクトファイルをリンクする際に本オプションを指定した場合、動作は保証しません。
アセンブラ出力のオブジェクトファイルに対しては、本オプション機能は適用されません。
下記のいずれかの条件の場合、本オプション指定は無効です。
・form=library,object,relocate 指定時
・アブソリュートファイル入力時
・memory=low 指定時
・nooptimize 指定がない場合
本オプションを指定して生成したリロケータブルファイルに対してはリンク時の最適化が適用されません。
マイコン種別が RX ファミリ、M16C シリーズ、R8C ファミリの場合は、本機能を使用できません。

データレコードのバイト数指定

BYte_count

リンカ<出力>[データレコード長 :]

書　式　　BYte_count=<数値>

説　明　　Intel-Hex 形式ファイルを生成する際に、データレコードのバイト数最大値を指定するためのオプションです。バイト数としては、1byte の 16 進数(01～FF)を指定することができます。本オプションを記述しない場合、バイト数最大値は FF として Intel-Hex ファイルを生成します。

例　　byte_count=10

備　考　　生成するファイル形式が Intel-Hex 形式(form=hex)ではない場合、本オプションは無効です。

CRC 演算

CRc

リンカ<出力>[オプション項目 :][CRC コード]

書　式

```
CRc = <サブオプション>
   <サブオプション>: <出力位置>=<計算範囲>[/<多項式>][:<エンディアン>]
     <出力位置>： <アドレス>
     <計算範囲>： <先頭アドレス>-<終了アドレス>[,...]
     <多項式>  ： { CCITT | 16 }
     <エンディアン>: {BIG|LITTLE}
```

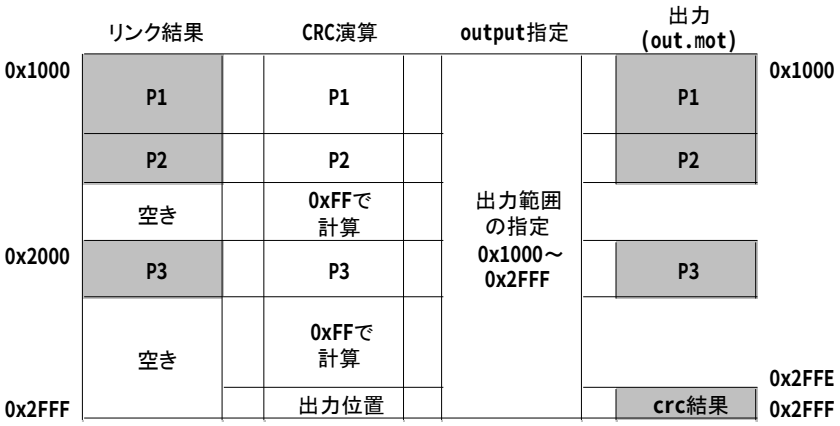
説　明　　計算範囲で指定された内容を下位アドレスから上位アドレスの順で CRC(Cyclic Redundancy Check)演算を行い、計算結果を出力位置のアドレスに出力します。
エンディアンは、RX ファミリの場合に指定可能なオプションです。エンディアンを指定した場合は、エンディアンにしたがって、計算結果を出力位置のアドレスに出力します。指定しない場合は、アブソリュートファイルのエンディアンで計算結果を出力位置のアドレスに出力します。
多項式は CRC-CCITT または CRC-16 を選択できます。(デフォルトは CRC-CCITT)

多項式

CRC-CCITT
X^16+X^12+X^5+1
ビット表現(10001000000100001)

CRC-16
X^16+X^15+X^2+1
ビット表現(11000000000000101)

例１　optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
　　-crc=2FFE=1000-2FFD -output=out.mot=1000-2FFF

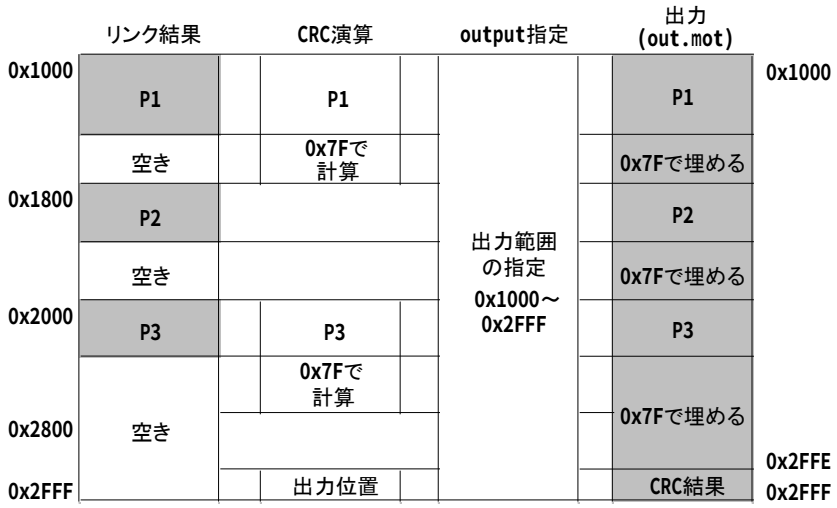


crc オプション：-crc=2FFE=1000-2FFD
　　0x1000～0x2FFD の領域に対して CRC 演算を行い、その結果を 0x2FFE 番地に出力します。計算範囲にある空き領域は space オプションが指定されていない場合、space=0xFF が指定されていると仮定して、CRC 演算を行います。

output オプション：-output=out.mot=1000-2FFF
　　space オプションが指定されていないため、空きの領域は「out.mot」ファイルに出力されません。CRC 演算は、空き領域では 0xFF で計算を行いますが、0xFF を埋めることはありません。

【注】 1. CRC **出力位置は、計算範囲に含むことは出来ません。**
2. CRC **出力位置は** output **オプションの出力範囲に含まれている必要があります。**

例 2　　optlnk *.obj -form=stype -start=P1/1000,P2/1800,P3/2000
　　-space=7F -crc=2FFE=1000-17FF,2000-27FF
　　-output=out.mot=1000-2FFF



crc オプション：-crc=2FFE=1000-2FFD,2000-27FF
　　0x1000～0x17FF と 0x2000～0x27FF の 2 つの領域に対して CRC 演算を行い、その結果を 0x2FFE 番地に出力します。
　　CRC 演算は計算対象として、連続していない複数の計算範囲を指定できます。

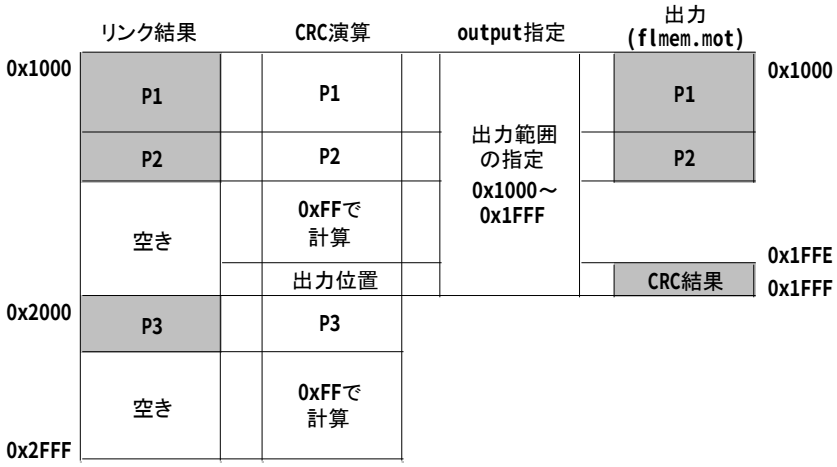
space オプション：-space=7F
　　指定された計算範囲の空き領域は space オプションの値(0x7F)で計算されます。

output オプション：-output=out.mot=1000-2FFF
　　space オプションが指定されているため、空き領域は「out.mot」ファイルに出力されます。
　　空き領域は 0x7F で充填されます。

【注】 1. CRC **演算の計算順は計算範囲の指定順ではありません。下位アドレスから上位アドレスの順に計算されます。**

2. crc **オプションと** space **オプションを同時に指定する場合、**space **オプションに** random **または 2 バイト以上の値を指定することは出来ません。1 バイトのデータを指定してください。**

例 3　　optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
　　-crc=1FFE=1000-1FFD,2000-2FFF
　　-output=flmem.mot=1000-1FFF



crc オプション：-crc=1FFE=1000-1FFD,2000-2FFF

0x1000～0x1FFD と 0x2000～0x2FFF の領域に対して CRC 演算を行い、その結果を 0x1FFE 番地に出力します。

計算範囲にある空き領域は space オプションが指定されていない場合、space=0xFF が指定されていると仮定して、CRC 演算を行います。

output オプション：-output=flmem.mot=1000-1FFF

space オプションが指定されていないため、空きの領域は「flmem.mot」ファイルに出力されません。

CRC 演算は、空き領域では 0xFF で計算を行いますが、0xFF を埋めることはありません。

備　考　　複数のアブソリュートファイル入力時は、本オプションは無効です。

出力形式が form={hexadecimal | stype}の場合に有効です。

space オプションが指定されていない場合で、計算範囲に出力されない空き領域があるとき、空き領域には 0xFF が設定されているものとして CRC の計算が行われます。

CRC 演算の計算範囲にオーバーレイ指定されている領域が含まれる場合はエラーになります。

サンプルコード

crc オプションで計算された CRC 演算結果を比較するためのサンプルコードです。
サンプルコードのプログラムは、optlnk の CRC 演算結果と一致します。

多項式 CRC-CCITT の場合

```
typedef  unsigned  char    uint8_t;
typedef  unsigned  short   uint16_t;
typedef  unsigned  long    uint32_t;

uint16_t CRC_CCITT(uint8_t *pData, uint32_t iSize)
{
  uint32_t ui32_i;
  uint8_t      *pui8_Data;
  uint16_t ui16_CRC = 0xFFFFu;

  pui8_Data = (uint8_t *)pData;

  for(ui32_i = 0; ui32_i < iSize; ui32_i++)
  {
    ui16_CRC  = (uint16_t)((ui16_CRC >> 8u) |
                  ((uint16_t)((uint32_t)ui16_CRC << 8u)));
    ui16_CRC ^= pui8_Data[ui32_i];
    ui16_CRC ^= (uint16_t)((ui16_CRC & 0xFFu) >> 4u);
    ui16_CRC ^= (uint16_t) ((ui16_CRC << 8u) << 4u);
    ui16_CRC ^= (uint16_t)(((ui16_CRC & 0xFFu) << 4u) << 1u);
  }
  ui16_CRC = (uint16_t)( 0x0000FFFFul &
               ((uint32_t)~(uint32_t)ui16_CRC) );
  return ui16_CRC;
}
```

多項式 CRC-16 の場合

```
#define POLYNOMIAL  0xa001      // 生成多項式 CRC-16

typedef  unsigned  char    uint8_t;
typedef  unsigned  short   uint16_t;
typedef  unsigned  long    uint32_t;

uint16_t CRC16(uint8_t *pData, uint32_t iSize)
{
  uint16_t crcdData = (uint16_t)0;
  uint32_t data = 0;
  uint32_t i,cycLoop;

  for(i=0;i<iSize;i++){
    data = (uint32_t)pData[i];
    crcdData = crcdData ^ data;
    for (cycLoop = 0; cycLoop < 8; cycLoop++) {
        if (crcdData & 1) {
            crcdData = (crcdData >> 1) ^ POLYNOMIAL;
        } else {
            crcdData = crcdData >> 1;
        }
    }
  }
  return crcdData;
}
```

セクション終端にパディング埋め込み

PADDING

リンカ<出力>[パディング :]

書　式　　PADDING

説　明　　セクションサイズが、セクションのアライメントの倍数となるように、セクション終端にデータを埋め込みます。

例　　-start=P,C/0 -padding
P セクションのアライメント:4 バイト
P セクションのサイズ:0x06 バイト
C セクションのアライメント:1 バイト
C セクションのサイズ:0x03 バイト
の場合、
P セクションに 2 バイトのパディングデータを埋め込んで、サイズを 0x08 バイトにしてリンクする。
-start=P/0,C/7 -padding
P セクションのアライメント:4 バイト
P セクションのサイズ:0x06 バイト
C セクションのアライメント:1 バイト
C セクションのサイズ:0x03 バイト
の場合、
P セクションに 2 バイトのパディングデータを埋め込んで、サイズを 0x08 バイトにしてリンクすると、C セクションと重複してしまうため、L2321 エラーを出力する。

備　考　　生成するパディングデータの値は 0x00 です。
絶対アドレスセクションには、パディングを行いませんので、絶対アドレスセクションはユーザにてサイズを調整してください。
マイコン種別が SuperH ファミリおよび RX ファミリのときに有効です。

特定ベクタ番号のアドレス設定

VECTN

リンカ<出力>[オプション項目 :] [特定ベクタ]

書　式　　VECTN = <サブオプション>[,...]
<サブオプション> : <ベクタ番号> = {<シンボル> | <アドレス>}

説　明　　可変ベクタテーブルセクションの特定ベクタ番号に対して、オプションで指定されたアドレスを設定します。
本オプションを使用した場合、ソース上に割り込み関数記述がなくても、可変ベクタテーブルセクションを作成し、テーブルへアドレスを設定します。

<ベクタ番号>は、10 進数で 0～255 の範囲で指定してください。
<シンボル>は、対象関数の外部名で指定してください。
<アドレス>は、指定アドレスを 16 進数で指定してください。

例　　-vectn=30=_f1,31=0000F100 ;ベクタ番号 30 番に_f1 のアドレスを、
;ベクタ番号 31 番に 0x0f100 を設定します

備　考　　マイコン種別が RX ファミリ、M16C シリーズ、R8C ファミリの場合に有効です。
ユーザが可変ベクタテーブルセクションをソースプログラムで作成している場合､可変ベクタテーブルの自動生成は行なわないため、本オプションは無効になります。

空きベクタ領域のアドレス設定

VECT

リンカ<出力> [オプション項目:] [空きベクタ]

書　式　　VECT={<シンボル>|<アドレス>}

説　明　　可変ベクタテーブルセクションで、アドレス未設定のベクタ番号に対してオプション指定のアドレスを設定します。
本オプションを使用した場合、ソース上の割り込み関数記述がなくても、可変ベクタテーブルセクションをリンカが作成し、テーブルへアドレスを設定します。
<シンボル>は、対象関数の外部名を記述してください。
<アドレス>は、設定するアドレスを 16 進数表記で記述してください。

備　考　　マイコン種別が RX ファミリ、M16C シリーズ、R8C ファミリの場合に有効です。
ユーザが可変ベクタテーブルセクションをソースプログラムで作成している場合､可変ベクタテーブルの自動生成は行なわないため、本オプションは無効になります。
{<シンボル>|<アドレス>}の記述で、先頭を 0 と記述したものは全てアドレスとして判断します。

UTL

リンカ<その他>[その他のオプション]utl ファイル出力

書　式　　UTL

説　明　　コンパイラパッケージ付属のツール(utl30)に入力するための外部ファイル(utl ファイル)を生成します。
生成するファイルの名称は「<出力ファイル名>.utl」となります。

例　　
```
tp.obj
utl
output=test.abs
```
tp.obj 内のインスペクタ情報を test.utl に出力します。

備　考　　本オプションは、M16C マイコン向けのコンパイラを使用した場合のみ有効です。
本オプションは、abs ファイル入力時の処理には使用できません。
form={object | library} 指定時、本オプションは無効です。

JUMP_ENTRIES_FOR_PIC

リンカ<出力>[ジャンプテーブル:]

書　式　　JUMP_ENTRIES_FOR_PIC = <セクション名>[,...]

説　明　　指定セクション内の外部定義シンボルへ分岐するジャンプテーブルのアセンブラソースを出力します。
RX ファミリ用コンパイラの PIC 機能向けに用意されたオプションです。
ファイル名は、<出力ファイル>.jmp です。

例　　jump_entries_for_pic=sct2,sct3
output=test.abs
セクション sct2,sct3 の外部定義シンボルへ分岐するジャンプテーブルを test.jmp に出力します。

[test.jmp の出力例]

```
;OPTIMIZING LINKAGE EDITOR GENERATED FILE 2009.07.19
          .glb _func01
          .glb _func02
          .SECTION    P,CODE
_func01:
          MOV.L      #1000H,R14
          JMP        R14
_func02:
          MOV.L      #2000H,R14
          JMP        R14
          .END
```

備　考　　form={object | relocate | library}または strip 指定時、本オプションは無効です。
マイコン種別が RX 系以外の場合は、本オプションは無効です。
生成するジャンプテーブルは、P セクションへ出力します。
セクション名に指定できるセクション種別は、プログラムセクションのみです。

5.2.3 リストオプション

表 5.5 リストカテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	リスト ファイル	LISt [= <ファイル名>]	リンカ <リスト> [リンケージリスト出力]	リストファイル出力を指定
2	リスト 内容	SHow [= <sub>[,...]] <sub> : {SYmbol| Reference| SEction| Xreference| Total_size| VECTOR| ALL }	リンカ <リスト> [リスト内容 :]	 シンボル情報 参照回数 セクション情報 クロスリファレンス情報 合計セクションサイズ ベクタ情報出力 全情報出力

リストファイル

LISt

リンカ <リスト>[リンケージリスト出力]

書　式　　LISt [= <ファイル名>]

説　明　　リストファイル出力およびリストファイル名を指定します。
リストファイル名を指定しない場合には、出力(または先頭出力)ファイルと同じファイル名で、拡張子が form=library または extract 指定時「lbp」、それ以外のとき「map」のリストファイルが作成されます。

SHow

リンカ <リスト>[リスト内容 :]

書　式　　SHow[= <sub>[,…]]
　　<sub> : { SYmbol | Reference | SEction | Xreference | Total_size
　　|VECTOR | ALL }

説　明　　リストの出力内容を指定します。
サブオプションの一覧を**表** 5.6 に示します。
各リストの具体例については「7.3　**リンケージリスト**」、「7.4　**ライブラリリスト**」を参照してください。

表 5.6　show **オプションのサブオプション一覧**

	出力形式	サブオプション名	意味
1	form=library **または** extract **指定時**	symbol	**モジュール内シンボル名一覧を出力します。**(extract **指定時**)
		reference	**指定できません。**
		section	**モジュール内セクション一覧を出力します。**(extract **指定時**)
		xreference	**指定できません。**
		total_size	**指定できません。**
		vector	**指定できません。**
		all	**指定できません**(extract **指定時**)。 **モジュール内シンボル名、セクション一覧を出力します**(form=library **指定時**)。
2	form=library **以外かつ** extract **指定なし時**	symbol	**シンボルアドレス、サイズ、種別、最適化内容を出力します。**
		reference	**シンボルの参照回数を出力します。**
		section	**指定できません。**
		xreference	**クロスリファレンス情報を出力します。**
		total_size	ROM **配置対象、**RAM **配置対象ごとに、セクションの合計サイズを表示します。**
		vector	**ベクタ情報を出力します。**
		all	show=symbol,xreferrence,total_size **指定時と同内容を出力します。**(form=rel) show=symbol,total_sizel **指定時と同内容を出力します。**(form=rel,data_stuff) show=symbol,reference,xreference ,total_size **指定時と同内容を出力します。**(form=abs) show=symbol,reference,xreference,total_size **指定時と同内容を出力します。**(form=hex/stype/bin) form=obj **のときは指定できません。**

備　考　オプション form とオプション show および show=all で有効/無効になる組み合わせは以下のようになります。

		Symbol	Reference	Section	Xreference	Vector	Total_size
form=abs	show **のみ**	**有効**	**有効**	**無効**	**無効**	**無効**	**無効**
	show=all	**有効**	**有効**	**無効**	**有効**	**有効**	**有効**
form=lib	show **のみ**	**有効**	**無効**	**有効**	**無効**	**無効**	**無効**
	show=all	**有効**	**無効**	**有効**	**無効**	**無効**	**無効**
form=rel	show **のみ**	**有効**	**無効**	**無効**	**無効**	**無効**	**無効**
	show=all	**有効**	**無効**	**無効**	**有効***1	**無効**	**有効**
form=obj	show **のみ**	**有効**	**有効**	**無効**	**無効**	**無効**	**無効**
	show=all	**無効**	**無効**	**無効**	**無効**	**無効**	**無効**
form=hex/ bin/sty	show **のみ**	**有効**	**有効**	**無効**	**無効**	**無効**	**無効**
	show=all	**有効**	**有効**	**無効**	**有効**	**有効***1	**有効***1

*1　入力ファイルが absolute 形式の場合は無効です。

クロスリファレンス情報の出力に関しては、下記制限があります。
- 出力ファイルが relocatable 形式で、かつ data_stuff オプションを使用している場合、クロスリファレンス情報は出力できません。
- 入力ファイルが absolute 形式の場合、参照側アドレスの情報は出力されません。
- アセンブル時に goptimize オプションが指定されていない場合、同一ファイル内への分岐に関する情報は出力されません。(マイコンが H8,H8S,H8SX ファミリの場合のみ)
- 同一ファイル内の、定数シンボルへの参照に関する情報は出力されません。
- コンパイル時に最適化が有効で、直下の関数を呼び出す場合についての情報は出力されません。
- 外部変数アクセス最適化が有効な場合、ベースとなるシンボルを除いて、変数の参照情報は出力されません。(マイコンが SuperH ファミリおよび RX ファミリの場合のみ)
- ソースファイル上で#pragma tbr を記述した際にオフセット値を直接指定している場合、当該関数についての情報は出力されません。(マイコンが SH-2A/SH2A-FPU の場合のみ)
- リンク時の最適化を指定した場合、定数やリテラルの統合が生じると、その定数やリテラルに関する参照情報は出力されません。
- show=total_size で表示する情報は、別オプション total_size での表示内容と同じです。
- show=vector は、マイコン種別が RX ファミリ、M16C シリーズ、R8C ファミリのとき使用できます。
- show=reference が有効な場合に、#pragma address で指定された変数の参照回数が 0 として出力されます。(マイコンが SuperH ファミリおよび RX ファミリの場合のみ)

5.2.4 最適化オプション

表 5.7 最適化カテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	最適化	OPtimize [= <sub>[,...]]	リンカ<最適化>	最適化あり
		<sub> : { STring_unify	[最適化方法 :]	定数/文字列の統合
		| SYmbol_delete	[最適化設定]	未参照シンボルの削除
		| Variable_access	[設定 :]	短絶対アドレッシングモード活用
		| Register		レジスタ退避/回復の最適化
		| SAMe_code		共通コードの統合
		| SHort_format		アドレッシングモードの短縮
		| Function_call		間接アドレッシングモード活用
		| Branch		分岐命令の最適化
		| SPeed		実行速度優先の最適化
		| SAFe }		安全な最適化
		NOOPtimize		最適化なし
2	共通コードサイズ	SAMESize = <サイズ>	リンカ<最適化>	共通コード統合の対象となる最小サイズの指定
		(省略時：sames=1e)	[統合サイズ :]	
3	プロファイル情報	PROfile = <ファイル名>	リンカ<最適化>	プロファイル情報ファイルの指定
			[プロファイル情報 :]	(動的最適化を行います)
4	キャッシュサイズ	CAchesize = <sub>	リンカ<最適化>	
		<sub>: Size = <サイズ>|	[キャッシュサイズ :]	キャッシュサイズの指定
		Align = <ラインサイズ>		キャッシュラインサイズの指定
		(省略時：ca=s=8,a=20)		(SuperH ファミリ向け)
5	最適化部分抑止	SYmbol_forbid =	リンカ<最適化>	未参照シンボル削除抑止シンボル
		<シンボル名>[,...]	[最適化方法 :]	
		SAMECode_forbid =	[最適化部分抑止]	共通コード統合抑止シンボル
		<関数名>[,...]		
		Variable_forbid =		短絶対アドレッシングモード活用抑止シンボル
		<シンボル名>[,...]		
		FUnction_forbid =		間接アドレッシングモード活用抑止シンボル
		<関数名>[,...]		
		SEction_forbid = <sub>[,...]		最適化抑止セクション
		<sub> : [<ファイル名>|		
		<モジュール名>]		
		(<セクション名>[,...])		
		Absolute_forbid =		
		<アドレス> [+ <サイズ>] [,...]		最適化抑止アドレス範囲

最適化

OPtimize
NOOPtimize

リンカ<最適化>[最適化方法 :][最適化設定][設定 :]

書　式　　OPtimize[= <サブオプション>[,…]]
NOOPtimize
<サブオプション> : {STring_unify | SYmbol_delete | Variable_access
| Register | SAMe_code | SHort_format
| Function_call | Branch | SPeed | SAFe}

説　明　　モジュール間最適化実行有無を指定します。
optimize オプション指定時は、コンパイル、アセンブル時に goptimize オプションを指定したファイルに対して最適化を行います。
nooptimize オプション指定時は、モジュールの最適化を行いません。
本オプションの省略時解釈は、optimize です。サブオプションの一覧を**表** 5.8 に示します。

表 5.8　optimize **オプションのサブオプション一覧**

サブオプション	意　味	最適化対象プログラム*1							
		SHC	SHA	H8C	H8A	RXC	RXA	NCC	NCA
パラメータなし	**全ての最適化を実行します。**	○	×	○	○	○	×	○	×
string_unify	const **属性を持つ定数に対し、同一値定数を統合します。**const **属性を持つ定数には次のものが含まれます。** **・**C/C++**プログラム中の** const **修飾型変数** **・文字列データの初期値/リテラル定数**	○	×	○	×	×	×	×	×
symbol_delete	1 **度も参照のない変数/関数を削除します。必ずコンパイル 時に**#pragma entry **を指定するか、**optlnk **で** entry **オプションを指定してください。**	○	×	○	×	○	×	○	×
variable_access	8/16 **ビット絶対アドレッシングモードでアクセス可能な領域にアクセス回数の多い変数を割り当てます。必ずコンパイル、アセンブル時に** cpu **オプションを指定してください。**	×	×	○	○	×	×	×	×
register	**関数の呼び出し関係を解析し、レジスタの再割付および冗長なレジスタ退避/回復コードを削除します。必ずコンパイル 時に**#pragma entry **を指定するか、**optlnk **で** entry **オプションを指定してください。**	○	×	○	×	×	×	×	×

サブオプション	意　味	最適化対象プログラム*1							
		SHC	SHA	H8C	H8A	RXC	RXA	NCC	NCA
same_code	複数の同一命令列をサブルーチン化します。	○	×	○	×	○	×	×	×
short_format	ディスプレースメント/イミディエートのコードサイズを短縮可能な場合、コードサイズがより小さくなる命令に置き換えます。	×	×	○	○	○	×	×	×
function_call	0〜0xFF の範囲に空きがあれば、アクセス回数の多い関数のアドレスを割り当てます。また、マイコン種別が H8SX ファミリの場合には、下記領域も使用されます。 H8SXN : 0x100〜0x1FF H8SXM,H8SXA,H8SXX : 0x200〜0x3FF 必ずコンパイル、アセンブル時に cpu オプションを指定してください。	×	×	○	○	×	×	×	×
branch	プログラムの配置情報に基づいて、分岐命令サイズを最適化します。他の最適化項目を実行すると、指定の有無に関わらず必ず実行します。	○	×	○	○	○	×	○	×
speed	オブジェクトスピード低下を招く可能性のある最適化以外を実行します。 optimize=string_unify, symbol_delete, varaible_access, register, short_format, branch と同じです。	○	×	○	○	○ *2	×	○ *2	×
safe	変数や関数の属性によって制限される可能性のある最適化以外を実行します。 optimize=string_unify, register, short_format, branch と同じです。	○	×	○	○	○ *4	×	○ *3	×

【注】 *1 SHC: SuperH ファミリ用 C/C++プログラム、SHA: SuperH ファミリ用アセンブリプログラム
H8C: H8,H8S,H8SX ファミリ用 C/C++プログラム、H8A: H8,H8S,H8SX ファミリ用アセンブリプログラム
RXC: RX ファミリ用 C/C++プログラム、RXA: RX ファミリ用アセンブリプログラム
NCC: M16C シリーズ、R8C ファミリ用 C/C++プログラム、NCA: M16C シリーズ、R8C ファミリ用アセンブリプログラム

*2 speed で有効になる最適化のうち symbol_delete, branch, short_format が有効になります。

*3 safe で有効になる最適化のうち branch が有効になります。

*4 safe で有効になる最適化のうち short_format, branch が有効になります。

備　考　form=｛object|relocate|library｝または strip 指定時、本オプションは無効です。
コンパイル時に外部変数アクセス最適化を指定した場合、定数/リテラル統合最適化(optimize=string_unify)は無効になります。
optimize=short_format 指定は、マイコン種別が H8SX ファミリ、RX ファミリの場合にのみ有効です。
マイコン種別が SH-2A/SH2A-FPU の場合、optimize=register の機能によってコードサイズが増加する場合があります。
プログラム内に#pragma entry で実行開始関数を指定、または実行開始アドレス(entry)を指定していない場合、optimize=symbol_delete は無効になります。

共通コードサイズ

SAMESize

リンカ<最適化>[統合サイズ :]

書　式　SAMESize = <サイズ>

説　明　共通コード統合最適化(optimize=same_code)で、最適化対象となる最小コードサイズを指定します。8～7FFF までの 16 進数で指定してください。
本オプションの省略時解釈は、samesize=1E です。

備　考　optimize=same_code の指定がないとき、本オプションは無効です。

PROfile

リンカ<最適化>[プロファイル情報 :]

書　式　　PROfile = <ファイル名>

説　明　　プロファイル情報ファイルを指定します。
プロファイル情報ファイルとして指定できるのは、ルネサス統合開発環境 Ver. 2.0 以降が出力するプロファイル情報ファイルだけです。
プロファイル情報ファイルを指定すると、モジュール間最適化で動的情報に基づいた最適化を実行できます。
プロファイル情報入力により影響がある最適化を**表**5.9 に示します。

表 5.9　**プロファイル情報と最適化の関係**

サブオプション	意　味	最適化対象プログラム*1			
		SHC	SHA	H8C	H8A
variable_access	動的アクセス回数の多い変数を優先的に割り当てます。	×	×	○	○
function_call	動的アクセス回数の多い関数の最適化優先順位を下げます。	×	×	○	○
branch	動的に呼び出し回数の多い関数を呼び出し元の関数の近くに配置します。 SuperH ファミリ用プログラムの場合は、cachesize オプションで指定するキャッシュサイズを意識した配置最適化を行います。	○	△ *2	○	△

【注】　*1　SHC: SuperH **ファミリ用** C/C++**プログラム、**SHA: SuperH **ファミリ用アセンブリプログラム、**
H8C: H8,H8S,H8SX **ファミリ用** C/C++**プログラム、**H8A: H8,H8S,H8SX **ファミリ用アセンブリプログラム**

*2　**関数単位の移動は行いませんが、入力ファイル単位の移動は実行します。**

備　考　　optimize 指定がないとき、本オプションは無効です。

キャッシュサイズ

CAchesize

リンカ<最適化>[キャッシュサイズ :]

書　式　　CAchesize = <sub>
<sub>:Size = <サイズ> | Align = <ラインサイズ>

説　明　　キャッシュサイズおよびキャッシュラインサイズを指定します。
profile オプション指定時、分岐命令最適化(optimize=branch)で使用します。
サイズはキロバイト単位、ラインサイズはバイト単位の 16 進数で指定してください。
本オプションの省略時解釈は、cachesize=size=8,align=20 です。

備　考　　profile 指定がないとき、本オプションは無効です。

最適化部分抑止

SYmbol_forbid
SAMECode_forbid
Variable_forbid
FUnction_forbid
SEction_forbid
Absolute_forbid

リンカ<最適化>[最適化方法 :][最適化部分抑止]

書　式
```
SYmbol_forbid = <シンボル名>[,…]
SAMECode_forbid = <関数名>[,…]
Variable_forbid = <シンボル名>[,…]
FUnction_forbid = <関数名>[,…]
SEction_forbid  = <sub>[,…]
   <sub>: [<ファイル名>|<モジュール名>](<セクション名>[,…])
Absolute_forbid = <アドレス>[+ <サイズ>][,…]
```

説　明　　特定のシンボル、セクション、アドレス範囲の最適化を抑止します。アドレス、サイズは 16 進数で指定してください。C/C++変数名、C 関数名はプログラム中での定義名先頭に_を付加します。C++関数の場合は、引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。但し引数が void の場合は、"関数名()"で指定します。各オプションの意味を**表** 5.10 に示します。

表 5.10　最適化部分抑止オプション一覧

オプション	パラメータ	意味
symbol_forbid	関数名|変数名	未参照シンボル削除最適化を抑止します。
samecode_forbid	関数名	共通コード統合最適化を抑止します。
variable_forbid	変数名	短絶対アドレッシングモード活用最適化を抑止します。
function_forbid	関数名	間接アドレッシングモード活用最適化を抑止します。
section_forbid	セクション名 ファイル名 モジュール名	特定セクションの最適化を抑止します。入力ファイル名、もしくはライブラリモジュール名を同時に指定することで、最適化抑止対象をセクション全体だけでなく、特定ファイルに限定することも可能です。
absolute_forbid	アドレス[+サイズ]	アドレス＋サイズの範囲の最適化を抑止します。

例　　`symbol_forbid="f(int)"  ; C++関数 f(int) は参照回数 0 でも削除しません。`
`section_forbid=(P1)    ; P1 セクションへの全最適化を抑止します。`
`section_forbid=a.obj(P1,P2)`
`; a.obj 内の P1,P2 セクションへの全最適化を抑止します。`

備　考　　最適化を使用しないリンク処理では、本オプションは無効です。
パスを記述した入力ファイルを最適化抑止する場合、`section_forbid` オプションではファイル名にパスを記述してください。

5.2.5 セクションオプション

表 5.11 セクションカテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	セクション アドレス	STARt= <sub>[,...] <sub> : [(]<セクション名> [{: | ,} <セクション名>[,...]] [)][,...] [/<アドレス>]	リンカ <セクション> [設定項目 :] [セクション]	セクションの開始アドレス指定
2	シンボル アドレス ファイル	FSymbol = <セクション名>[,...]	リンカ <セクション> [設定項目 :] [シンボルアドレスファイル]	外部定義シンボルアドレスの定義ファイル出力
3	セクション アライメント指定	ALIGNED_SECTION= <セクション名>[,...]	リンカ <セクション> [設定項目 :] [セクションアライメント指定]	セクションアライメントを 16 バイトに変更

セクションアドレス

STARt

リンカ <セクション>[設定項目 :][セクション]

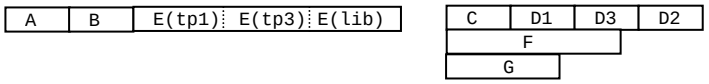
書　式　　STARt = <sub> [,…]
<sub>: [(]<セクション名>[{:|,} <セクション名>[,…]][)][,…][/<アドレス>]

説　明　　セクションの開始アドレスを指定します。アドレスは 16 進数で指定してください。
セクション名はワイルドカード"*"も指定できます。ワイルドカードで指定したセクションは入力順に展開します。
セクションをコロン":"で区切ることにより、複数のセクションを同一アドレスに割り付ける(セクションオーバーレイ配置)ことが可能です。
同一アドレスに割り付け指定したセクション間は、指定順に割り付けます。
また、丸括弧"()"で囲むことにより、オーバーレイ配置する対象セクションを変更できます。
同一セクション内オブジェクトは、入力ファイルの指定順、入力ライブラリの指定順に割り付けます。
アドレスの指定がない場合は、0 番地から割り付けます。
start オプションで指定していないセクションは、最終割り付けアドレスに続いて割り付けます。

例　　下記順番でオブジェクトを入力する場合のセクション配置を例に示します。
(括弧内は各オブジェクトが持つセクション)

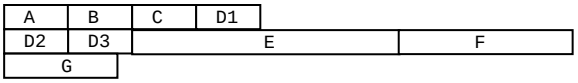
```
tp1.obj(A,D1,E) -> tp2.obj(B,D3,F) -> tp3.obj(C,D2,E,G) -> lib.lib(E)
```

```
(1) -start=A,B,E/400,C,D*:F:G/8000
  0x400                              0x8000
```



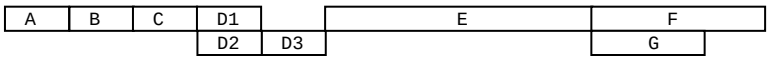
- ":"で区切った C,F,G セクションは、同一アドレスに割りつきます。
- ワイルドカードで記述したセクション(ここでは D で始まる名前のセクション)は、入力した順番で割りつきます。
- 同名セクション内(ここでは E セクション)は、入力したオブジェクトから順番に割りつきます。
- ライブラリ入力による同名セクション(ここでは E セクション)は、入力オブジェクトの次に割りつきます。

```
(2) -start=A,B,C,D1:D2,D3,E,F:G/400
  0x400
```



- ":"で区切った直後のセクション(この例の場合は A,D2,G)を先頭として、それぞれ先頭が同一アドレスに割りつきます。

```
(3) -start=A,B,C,(D1:D2,D3),E,(F:G)/400
  0x400
```



- "()"で同一アドレス配置を括った場合、"()"の直前のセクション(この例の場合は C,E)の直後を先頭として、"()"内の同一アドレス配置が行われます。
- "()"の直後のセクション(この場合 E)は、"()"内の最後尾のセクションの直後に続けて配置されます。

備　考　　form={object | relocate | library}または strip 指定時、本オプションは無効です。
括弧"()"は、ネストして記述することはできません。
括弧"()"内では、少なくともひとつはコロン":"の記述が必要です。コロン":"を記述しない場合には、括弧"()"は記述できません。
括弧"()"を記述した場合、"()"外にコロン":"を記述することはできません。
括弧"()"を使用して本オプションを記述した場合、リンカの最適化機能は無効になります。

シンボルアドレスファイル

FSymbol

リンカ <セクション>[設定項目 :][シンボルアドレスファイル]

書　式　FSymbol = <セクション名>[,…]

説　明　指定したセクション内外部定義シンボルをアセンブラ制御命令形式でファイルに出力します。ファイル名は、<出力ファイル>.fsy です。

例

```
fsymbol=sct2,sct3
output=test.abs
```

セクション sct2,sct3 の外部定義シンボルを test.fsy に出力します。

[test.fsy の出力例]

```
;OPTIMIZING LINKAGE EDITOR GENERATED FILE 1999.11.26
;fsymbol = sct2, sct3

;SECTION NAME = sct2
 .export _f
_f: .equ h'00000000
 .export _g
_g: .equ h'00000016
;SECTION NAME = sct3
 .export _main
_main: .equ h'00000020
 .end
```

備　考　form={object | relocate | library}または strip 指定時、本オプションは無効です。マイコン種別が H8,H8S,H8SX ファミリ, SuperH ファミリ, RX ファミリのときに使用できます。

セクションアライメント数を 16 バイトに変更

ALIGNED_SECTION

リンカ<セクション>[設定項目:][セクション]

書　式　ALIGNED_SECTION = <セクション名>[,…]

説　明　指定セクションのアライメント数を 16 byte に変更します。

備　考　form={object | relocate | library}および extract、strip 指定時、本オプションは無効です。

5.2.6 ベリファイオプション

表 5.12 ベリファイカテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	アドレス整合性のチェック	CPu = { <cpu 情報ファイル名> | <メモリ種別> = <アドレス範囲>[,...] | STRIDE } <メモリ種別> = { ROm | RAm | XROm | XRAm | YROm | YRAm | FIX} <アドレス範囲>: <先頭アドレス> - <終了アドレス>	リンカ <ベリファイ> [アドレス整合チェック :]	セクションアドレスの割り付け可能範囲を指定 セクション名をセクション分割の対象に指定
2	物理空間上の重複チェック	PS_check=<sub>[:<sub>...] <sub>: <LS>,<LS>[,...] <LS>: <開始アドレス> -<終端アドレス>	リンカ <ベリファイ> [物理空間上の重複チェック :]	物理空間上で重なり合うアドレス範囲を指定
3	セクション分割対象外指定	CONTIGUOUS_SECTION = <セクション名>[,…]	リンカ <ベリファイ> [分割対象がセクション :]	セクション名をセクション分割の対象外セクションに指定

CPu

リンカ <ベリファイ>[アドレス整合チェック :]

書　式　CPu = { <cpu 情報ファイル名>
　　| <メモリ種別> = <アドレス範囲>[,…]
　　| STRIDE}
＜メモリ種別＞ = { ROm | RAm | XROm | XRAm | YROm | YRAm | FIX }
＜アドレス範囲＞ : <先頭アドレス> - <終了アドレス>

説　明　cpu=stride 未指定時は、セクションの割り付けアドレスに対して、アドレス範囲に入らない場合は、エラーを出力します。
cpu=stride 指定時は、セクションの割り付けアドレスに対して、アドレス範囲に入らない場合は、次の同メモリ種別に配置、または、分割して配置します。

[例]サブオプション stride を指定しない場合
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF
　D1 が 100-1FF、D2 が 200-2FF の範囲に収まるとき、正常終了します。収まらないときエラーを出力します。
[例]サブオプション stride を指定した場合
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF,ROM=300-3FF
cpu=stride
　D1,D2 が ROM 属性の領域に(セクションを分割して/分割しないで)収まるとき、正常終了します。セクションを分割しても収まらないときリンクエラーになります。

xrom/xram は DSP の X メモリ領域、yrom/yram は DSP の Y メモリ領域を指定します。
セクション割り付けが可能なアドレス範囲を 16 進数で指定してください。ROM/RAM の属性は、モジュール間最適化で使用します。
メモリ種別 ”FIX ” には、アドレス固定の領域(I/O エリア等)を指定します。
メモリ種別 ”FIX ”と、それ以外のメモリ種別のアドレス範囲が重複した場合は、メモリ種別 ”FIX ” を有効とします。

サブオプション stride は、メモリ種別が、ROM または RAM で、アドレス範囲にセクションが収まらなかった場合に、セクションを分割して同じメモリ種別の領域に割り付けます。
サブオプション stride で、セクションを分割する単位は、モジュール単位になります。
[例]
cpu=ROM=0-FFFF,RAM=10000-1FFFF
セクションアドレスが、0-FFFF または 10000-1FFFF の間に入っているかチェックします。
モジュール間最適化では、異なる属性間でのオブジェクトの移動は行いません。

cpu=ROM=100-1FF,ROM=400-4FF,RAM=500-5FF
cpu=stride
セクションアドレスが、100-1FF の間に収まらなかった場合に、セクションをモジュール単位で分割して 400-4FF に割り付けます

備　考　form={object | relocate | library}または strip 指定時、本オプションは無効です。
cpu=stride および memory=low 指定時、無効になります。
マイコン種別が SH2DSP,SH3DSP,SH4ALDSP 以外の場合は、メモリ種別が
xrom,xram,yrom,yram の指定は無効となります。
cpu=stride および optimize=register が有効な場合、L2320 エラーが出力されることがあります。その場合には、optimize=register を無効にしてください。
cpu=stride を指定し、B セクションが分割された場合、0 初期化するための情報として 8 バイト×分割数分だけ C$BSEC セクションのサイズが増加します。

物理空間上の重複チェック

PS_check

リンカ <ベリファイ>[物理空間上の重複チェック :]

書　式　PS_check=<sub>[:<sub>...]
<sub>: <LS>,<LS>[,...]
<LS>: <開始アドレス>-<終端アドレス>

説　明　アドレス値では重なっていないが、実際にメモリ上に配置すると重なってしまうオブジェクトを検出するためのオプションです。
本オプションを使用することにより、SH3 や SH4 などで、論理アドレス上では重ならないが実メモリ上に配置する際に重なってしまうオブジェクトを検出することが可能です。
本オプションによって重複を検出した場合、エラーとしてリンク処理を終了します。
メモリ上で重なり合うアドレス範囲(書式の中の<LS>)をオプションに記述してください。
複数の物理メモリに対してチェックしたい場合には、':'で区切って記述する
ことでチェック可能です。

例　SH4 は、MMU が無効状態の場合、4G バイトのアドレス空間は、512M バイト(29bit)の外部メモリ空間へマッピングします(4G バイトアドレスの上位 3bit を無視してマッピングします)。
たとえば、ユーザモードで使用可能な U0 領域(00000000～0x7fffffff)に対して、外部メモリ(512M)にマッピングする場合のオブジェクトの重なりは、下記記述で検出可能です。

```
-PS_check=00000000-1fffffff,20000000-3fffffff,40000000-5fffffff,60000000-7fffffff
```

本オプション記述により、00000000,20000000,40000000,60000000 番地はすべて、実メモリ上では同じ場所に配置されることを表します。

備　考　本オプションは、SuperH ファミリのマイコンに対してのみ有効です。
出力形式(form オプション)が object,relocate,library の場合、本オプションは無効です。
absolute ファイルを入力する場合の処理は、本オプションは無効です。
マイコンのアドレス空間の仕様については、各マイコンのハードウェアマニュアルを参照してください。

セクション分割対象外指定

CONTIGUOUS_SECTION

リンカ<ベリファイ>[分割対象外セクション :]

書　式　　CONTIGUOUS_SECTION=<セクション名>[,...]

説　明　　cpu=stride が有効なときに、セクションを分割せずに同じメモリ種別の割り付け可能なアドレス領域に割り付けるセクションを指定します。

```
[例]
start=P,PA,PB/100
cpu=ROM=100-1FF,ROM=300-3FF,ROM=500-5FF
cpu=stride
contiguous_section=PA
```

セクション P を 100 番地に割り付けます。
contiguous_section 指定したセクション PA が、1FF 番地までに割り付けることができない場合、セクション PA を分割せずに、300 番地から割り付けます。
contiguous_section 指定してないセクション PB が、3FF 番地までに割り付けることができない場合、セクション PB を分割して、500 番地から割り付けます。

備　考　　cpu オプションのサブオプションの stride が無効なとき、本オプションは無効です。

5.2.7 その他オプション

表 5.13 その他カテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	終端コード	S9	リンカ <その他> [その他のオプション :] [S9 レコードを終端に出力]	S9 レコードを常に出力
2	スタック情報ファイル	STACk	リンカ <その他> [その他のオプション :] [スタック情報ファイル(sni)出力]	スタック使用量情報ファイル出力
3	デバッグ情報圧縮	COmpress NOCOmpress	リンカ <その他> [その他のオプション :] [デバッグ情報圧縮]	デバッグ情報を圧縮する デバッグ情報を圧縮しない
4	メモリ使用量削減指定	MEMory = [High｜Low]	リンカ <その他> [その他のオプション :] [入力ファイルロード時のメモリ使用量削減]	入力ファイルをロードする際のメモリ使用量指定
5	シンボル名変更	RENname = <sub>[,...] <sub> : { [<ファイル名>] (<名前>=<名前>[,...]) | [<モジュール名>] (<名前>=<名前>[,...]) }	リンカ <その他> [ユーザ指定オプション :]	シンボル名、セクション名の変更
6	シンボル名削除	DELete = <sub>[,...] <sub> : { <モジュール名> | [<ファイル名>] (<名前>[,...]) }	リンカ <その他> [ユーザ指定オプション :]	シンボル名、モジュール名の削除
7	モジュールの置き換え	REPlace = <sub>[,...] <sub> : <ファイル> [(<モジュール>[,...])]	リンカ <その他> [ユーザ指定オプション :]	ライブラリファイル内同名モジュールの置き換え
8	モジュールの抽出	EXTract = <モジュール>[,...]	リンカ <その他> [ユーザ指定オプション :]	ライブラリファイル内指定モジュールの抽出
9	デバッグ情報削除	STRip	リンカ <その他> [ユーザ指定オプション :]	アブソリュートファイル、ライブラリファイルのデバッグ情報削除

	項目	コマンドライン形式	ダイアログメニュー	指定内容
10	メッセージレベル	CHange_message = <sub>[,...] <sub>: {Information | Warning | Error} [=<エラー番号> [-<エラー番号>] [,...]]	リンカ <その他> [ユーザ指定オプション :]	メッセージレベルの変更
11	ローカルシンボル名秘匿指定	Hide	リンカ <その他> [ユーザ指定オプション :]	ローカルシンボル名情報を削除
12	合計セクションサイズの表示	Total_size	リンカ <その他> [ユーザ指定オプション :]	標準出力へ、リンク後の合計セクションサイズを表示できます。
13	エミュレータ向けの情報ファイル	RTs_file	リンカ<その他> [その他のオプション :] [関数出口情報ファイル(rts)出力]	エミュレータ向けの情報ファイルを出力します。 (SuperH ファミリ向け)

終端コード

S9

リンカ <その他>[その他のオプション :][S9 レコードを終端に出力]

書　式　S9

説　明　エントリアドレスが 0x10000 を超える場合でも、S9 レコードを終端に出力します。

備　考　form=stype 指定がないとき、本オプションは無効です。

スタック情報ファイル

STACk

リンカ <その他>[その他のオプション :][スタック情報ファイル(sni)出力]

書　式　STACk

説　明　スタック使用量情報ファイルを出力します。
ファイル名は、<出力ファイル名>.sni になります。

備　考　form={object | relocate | library}および strip 指定時、本オプションは無効です。

デバッグ情報圧縮

COmpress
NOCOmpress

リンカ <その他>[その他のオプション :][デバッグ情報圧縮]

書　式　　COmpress
NOCOmpress

説　明　　デバッグ情報の圧縮有無を指定します。
compress オプションを指定した場合、デバッグ情報を圧縮します。
nocompress オプションを指定した場合、デバッグ情報を圧縮しません。
デバッグ情報を圧縮すると、デバッガのロード速度が速くなります。また、nocompress オプションを指定すると、リンク時間が短くなります。
本オプションの省略時解釈は、nocompress です。

備　考　　form={object | relocate | library | hexadecimal | stype | binary}
または strip オプションを指定した場合、compress オプションは無効です。

メモリ使用量削減指定

MEMory

リンカ <その他>[その他のオプション :][入力ファイルロード時のメモリ使用量削減]

書　式　　MEMory ＝ [High | Low]

説　明　　リンク時に使用するメモリ量を指定します。
memory=high オプションを指定した場合、従来通りの処理を行います。
memory=low オプションを指定した場合、リンク時に必要な情報のロードを細かく行うことにより、使用するメモリ量の削減を行います。ファイルアクセスの頻度が増えるため、メモリ使用量が実装メモリを超えない状況では memory=high オプション指定より処理が遅くなります。

大規模なプロジェクトをリンクした際、最適化リンケージエディタのメモリ使用量が稼動マシンの実装メモリ量を越えてしまい、動作が遅くなっているような場合には memory=low オプション指定をお試しください。

備　考　　下記オプションを指定した場合、memory=low オプション指定は無効となります。
form=absolute,hexadecimal,stype,binary 指定時
　compress,delete,rename,map,stack,cpu=stride
　list と show[={reference | xreference}]を同時指定
form=library 指定時
　delete,rename,extract,hide,replace
form=object,relocate 指定時
　extract
　マイコン種別が NC ファミリ以外で optimize を指定時
また、入力ファイルや出力ファイル形式によっても無効となる組み合わせがあります。詳細は、**「**5.2.2　**出力オプション」**の**表** 5.4を参照してください。

シンボル名変更

REName

リンカ <その他>[ユーザ指定オプション :]

書　式　　REName = <サブオプション>[,…]
<サブオプション> :　{ [<ファイル>](<名前> = <名前>[,…])
| [<モジュール>](<名前> = <名前>[,…])}

説　明　　外部シンボル名、セクション名を変更します。
特定のファイルまたは特定のライブラリ内モジュールに含まれるシンボル名、セクション名を変更することもできます。
C/C++変数名の場合、プログラム中での定義名先頭に_を付加します。
関数名を変更した場合の動作は保証できません。
指定した名前がセクション、シンボルの両方に存在した場合、シンボル名を優先します。
同一ファイル名、モジュール名が複数存在する場合は、先に入力した方を優先します。

例　　rename=(_sym1=data)　　;_sym1 を data に変更します。
rename=lib1(P=P1)　　;ライブラリモジュール lib1 内の P セクションを
;P1 セクションに変更します。

備　考　　extract または strip 指定時、本オプションは無効です。
form=absolute 指定時、入力されたライブラリのセクション名を変更することができません。

シンボル名削除

DELete

リンカ <その他>[ユーザ指定オプション :]

書　式　　DELete = <サブオプション>[,…]
<サブオプション> :　{ [<ファイル>](<名前>[,…])
| <モジュール> }

説　明　　外部シンボル名またはライブラリモジュールを削除します。
特定のファイルに含まれるシンボル名、モジュールを削除することもできます。
C/C++変数名、C 関数名はプログラム中での定義名先頭に_を付加します。C++関数の場合は、引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。但し引数が void の場合は、"関数名()"で指定します。同一ファイル名が複数存在する場合は、先に入力した方を優先します。
本オプションで、シンボル名削除を指定した場合、オブジェクトは削除されず、属性が内部シンボルに変更されます。

例　　delete=(_sym1)　　; 全ファイル中のシンボル名_sym1 を削除します。
delete=file1.obj(_sym2) ;file1.obj 内のシンボル名_sym2 を削除します。

備　考　　extract または strip 指定時、本オプションは無効です。
form=library のときに、モジュールを削除できます。
form={absolute|relocate|hexadecimal|stype|binary}のときに、外部シンボルを削除できます。

モジュールの置き換え

REPlace

リンカ <その他>[ユーザ指定オプション :]

書　式　　REPlace = <サブオプション>[,…]
<サブオプション> :　<ファイル名>[(<モジュール名>[,…])]

説　明　　ライブラリモジュールを置換します。
指定したファイルまたはライブラリモジュールと library オプションで指定したライブラリ内同名モジュールを置換します。

例　　replace=file1.obj　　;モジュール file1 と file1.obj を置換します。
replace=lib1.lib(mdl1)　;モジュール mdl1 とライブラリファイル lib1.lib 内
;モジュール mdl1 を置換します。

備　考　　form={object | relocate | absolute | hexadecimal | stype | binary}
および extract、strip 指定時、本オプションは無効です。

モジュールの抽出

EXTract

リンカ <その他>[ユーザ指定オプション :]

書　式　　EXTract = <モジュール名>[,…]

説　明　　ライブラリモジュールを抽出します。
指定したライブラリモジュールを library オプションで指定したライブラリファイルから抽出します。

例　　extract=file1　　;モジュール file1 を抽出します。

備　考　　form={absolute | hexadecimal | stype | binary}および strip 指定時、本オプションは無効です。
form=library 指定時、モジュールを削除できます。
form={absolute|relocate|hexadecimal|stype|binary}指定時、外部シンボルを削除できます。

デバッグ情報削除

STRip

リンカ <その他>[ユーザ指定オプション :]

書　式　　STRip

説　明　　アブソリュートファイル、ライブラリファイルのデバッグ情報を削除します。
strip オプション指定時は、入力ファイルと出力ファイルは 1 対 1 対応になります。

例　　input=file1.abs file2.abs file3.abs
strip
file1.abs, file2.abs, file3.abs のデバッグ情報を削除し、それぞれ file1.abs, file2.abs, file3.abs に出力します。デバッグ情報削除前のファイルは、file1.abk, file2.abk, file3.abk にバックアップします。

備　考　　form={object | relocate | hexadecimal | stype | binary}指定時、本オプションは無効です。

メッセージレベル

CHange_message

リンカ <その他>[ユーザ指定オプション :]

書　式　　CHange_message = <サブオプション>[,…]
<サブオプション> : <エラーレベル>[=<エラー番号>[-<エラー番号>][,…]]
<エラーレベル>　: {Information | Warning | Error}

説　明　　インフォメーション、ウォーニング、エラーレベルのメッセージレベルを変更します。
メッセージ出力時の実行継続/中断を変更できます。

例　　change_message=warning=2310
L2310 をウォーニングレベルに変更し、L2310 出力時も処理を継続します。

change_message=error
全てのインフォメーション、ウォーニングメッセージをエラーレベルに変更します。
メッセージを一つでも出力すると、処理を中断します。

ローカルシンボル名秘匿指定

Hide

リンカ <その他>[ユーザ指定オプション :]

書　式　Hide

説　明　本オプションを指定した場合、出力ファイル内のローカルシンボル名情報を消去します。
ローカルシンボルに関する名前の情報が消去されますので、バイナリエディタなどでファイルを開いてもローカルシンボル名は確認できなくなります。生成されるファイルの動作への影響は一切ありません。
ローカルシンボル名を機密扱いにしたい場合などに本オプションを指定してください。

秘匿対象となるシンボルの種類を以下に挙げます。
・ソースファイル：static 型修飾子を指定した変数名、関数名など
・ソースファイル：goto 文のラベル名
・アセンブリソース：外部定義(参照)シンボル宣言していないシンボル名
※ エントリ関数名は秘匿対象になりません。

例　ソースファイルで本オプションの機能が有効となる記述の例を以下に示します。

```
int g1;
int g2=1;
const int g3=3;
static int s1;          //<--- static 変数名は秘匿対象
static int s2=1;        //<--- static 変数名は秘匿対象
static const int s3=2;  //<--- static 変数名は秘匿対象

static int sub1()       //<--- static 関数名は秘匿対象
{
   static int s1;       //<--- static 変数名は秘匿対象
   int l1;

   s1 = l1; l1 = s1;
   return(l1);
}

int main()
{
   sub1();
   if (g1==1)
      goto L1;
   g2=2;
L1:                     //<--- goto 文のラベル名は秘匿対象
   return(0);
}
```

備　考　本オプションは出力ファイル形式が absolute, relocate, library の場合のみ有効です。
コンパイル、アセンブル時に goptimize オプションを指定したファイルを入力する場合、出力ファイル形式が relocate, library の場合は本オプションを指定できません。
外部変数アクセス最適化を行う状況で本オプションを指定する場合は、一度目のリンク時には指定せず、二度目のリンク時にのみ本オプションを指定してください。
デバッグ情報内のシンボル名は、本オプションを指定しても削除されません。

合計セクションサイズの表示

Total_size

リンカ<その他>[その他のオプション :] [合計セクションサイズ画面表示]

書　式　Total_size

説　明　リンク後のセクションの合計サイズを、標準出力に表示するためのオプションです。
下記の 3 種類のセクションに分けて、合計サイズを表示します。
・実行可能なプログラムセクション
・プログラムセクション以外の ROM 領域配置セクション
・RAM 領域配置セクション

本オプションを使用することにより、ROM,RAM に配置する合計のセクションサイズを容易に認識することができます。

備　考　リンケージリストへ合計サイズを表示するには、別途 show=total_size オプションを使用する必要があります。
ROM 化支援機能(rom オプション)対象のセクションの場合、転送元(ROM)と転送先(RAM)の両方で領域を使用するため、双方の合計サイズに対してセクションサイズを加算します。

エミュレータ向けの情報ファイル

RTs_file

リンカ<その他>[その他のオプション :][関数出口情報ファイル(rts)出力]

書　式　-RTs_file

説　明　エミュレータで使用するための情報、関数出口情報ファイル(.rts ファイル)を生成するオプションです。
お使いのエミュレータのマニュアルに従って、本オプションを使用してください。エミュレータの機種によって使用できない場合があります。
関数出口情報ファイルは、「<出力するロードモジュール名>.rts」というファイル名で生成されます。例えば、output オプションで指定する出力ファイル名を「test.abs」とした場合、関数出口情報ファイルは「test.rts」というファイル名で生成されます。
関数出口情報ファイルはロードモジュールと同じディレクトリに作成されます。

備　考　form={object | relocate | library}指定時、本オプションは無効です。
アブソリュートファイルを入力する場合、本オプションは無効です。
エミュレータのマニュアルに従って本オプションを使用してください。エミュレータの機種によって使用できない場合があります。
本オプションは、マイコン種別が SuperH ファミリのとき使用できます。

5.2.8 サブコマンドファイルオプション

表 5.14 サブコマンドファイルカテゴリオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	サブコマンドファイル	SUbcommand = <ファイル名>	リンカ <サブコマンドファイル> [サブコマンドファイルを指定]	サブコマンドファイルによるオプション指定

サブコマンドファイル

SUbcommand

リンカ<サブコマンドファイル> [サブコマンドファイルを指定]

書　式　　SUbcommand = <ファイル名>

説　明　　オプションをサブコマンドファイルで指定します。
サブコマンドファイルの書式は以下の通りです。

<オプション> {= | △}［<サブオプション>［,…］][△&] [;<コメント>]

オプションとサブオプションの区切りは、=の代わりに空白も指定できます。
input オプションの場合は、サブオプション区切りに空白を指定できます。
サブコマンドファイル内では 1 オプション/行で指定します。
サブオプションを 1 行に記述できない場合は、&を用いて継続指定できます。
サブコマンドファイル中に subcommand オプションは指定できません。

例　　コマンドライン指定：optlnk file1.obj -sub=test.sub file4.obj
サブコマンド指定　：input　file2.obj file3.obj ;ここはコメントです。
library lib1.lib, &　;継続行を指定します。
lib2.lib
サブコマンドファイルで指定したオプション内容を、コマンドライン上のサブコマンド指定位置に展開し、実行します。
ファイルの入力順序は、file1.obj, file2.obj, file3.obj, file4.obj になります。

5.2.9 マイコンオプション

表 5.15 CPU タブオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	SBR アドレス指定	SBr = { <SBR アドレス> | User}	CPU [SBR 値 :]	8bit 絶対領域の開始アドレスを指定(H8SX ファミリ向け)

8bit 絶対領域アドレス値指定

SBr

CPU [SBR 値 :]

書　式　　SBr = { <アドレス> | User }

説　明　　SBR のアドレス値を指定します。
本オプションでアドレス値を指定することにより、abs8 領域を用いた最適化が可能になります。本オプションで user を指定した場合は、abs8 領域への最適化は抑止されます。

備　考　　本オプションはマイコン種別が H8SX ファミリの場合にのみ有効です。
ソース内、あるいはツールのオプション指定などで、複数の SBR アドレスが指定された場合には、本オプションは指定の如何に関わらず user が指定されたものとして扱われます。

5.2.10 残りのオプション

表 5.16 残りのオプション一覧

	項目	コマンドライン形式	ダイアログメニュー	指定内容
1	コピーライト	LOgo	-	出力あり
		NOLOgo	(常に NOLOgo が有効)	出力なし
2	継続指定	END	-	既入力オプション列を実行し、処理終了後は以降のオプション列を入力し、処理を継続
3	終了指定	EXit	-	オプション入力の終了を指定

コピーライト

LOgo
NOLOgo

なし(常に nologo が有効)

書　式　LOgo
NOLOgo

説　明　コピーライトの出力有無を指定します。
logo オプション指定時はコピーライト表示を出力します。
nologo オプション指定時はコピーライト表示出力を抑止します。
本オプションの省略時解釈は、logo です。

継続処理

END

なし

書　式　END

説　明　END より前に指定したオプション列を実行します。リンケージ処理終了後、END 以降に指定したオプション列の入力、リンケージ処理を継続します。
本オプションは、コマンドライン上では指定できません。

例
```
input=a.obj,b.obj              ; 処理(1)
start=P,C,D/100,B/8000         ; 処理(2)
output=a.abs                   ; 処理(3)
end
input=a.abs                    ; 処理(4)
form=stype                     ; 処理(5)
output=a.mot                   ; 処理(6)
```

(1)～(3)の処理を実行し、a.abs を出力します。
その後、(4)～(6)の処理を実行し、a.mot を出力します。

終了処理

EXIt

なし

書　式　　EXIt

説　明　　オプション指定の終了を指定します。
本オプションは、コマンドライン上では指定できません。

例　　コマンドライン指定： optlnk -sub=test.sub -nodebug

```
test.sub:          input=a.obj,b.obj ; 処理(1)
                   start=P,C,D/100,B/8000       ; 処理(2)
                   output=a.abs                 ; 処理(3)
                   exit
```

(1)～(3)の処理を実行し、a.abs を出力します。
Exit 実行後のコマンドライン指定の nodebug オプションは無効になります。

6. 環境変数

6.1 環境変数一覧

環境変数の一覧を**表** 6.1 に示します。

表6.1 **環境変数**

	環境変数	説明	設定省略時の解釈
1	path	実行ファイルの格納ディレクトリを指定します。	省略不可
2	BIN_RX	ccrx を格納したディレクトリを指定します。	＜ccrx 格納ディレクトリ＞ lbgrx コマンド利用時は、省略不可
3	CPU_RX	CPU 種別を指定します。 <CPU 種別> RX600 RX200	省略時、値は設定されません
4	INC_RX	コンパイラのインクルードファイル格納ディレクトリを指定します。	＜ccrx 格納ディレクトリ＞¥..¥include
5	INC_RXA	アセンブラのインクルードファイル格納ディレクトリを指定します。	省略時、値は設定されません
6	TMP_RX	テンポラリファイルを作成するディレクトリを指定します。	ccrx コマンド利用時は、%TEMP%
7	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	最適化リンケージエディタが使用するデフォルトライブラリ名を指定します。library オプションで指定したライブラリを優先してリンクします。その後未解決のシンボルがある場合、1,2,3 の順にデフォルトライブラリを検索します。	省略時、値は設定されません
8	HLNK_TMP	最適化リンケージエディタがテンポラリファイルを作成するフォルダを指定します。この環境変数の指定がない場合は、カレントフォルダにテンポラリファイルを作成します。	省略時、値は設定されません
9	HLNK_DIR	最適化リンケージエディタの入力ファイル格納フォルダを指定します。 input オプション、library オプションで指定したファイルの検索順序は、カレントフォルダ、HLNK_DIR 指定フォルダになります。 ただし、ワイルドカードで指定したファイルは、カレントフォルダ内だけ検索します。	省略時、値は設定されません

環境変数 CPU_RX の設定は、cpu オプションによる選択がない場合は必須となります。CPU 種別として RX600 または RX200 以外を指定した場合、エラーとなります。

CPU_RX と cpu オプションの関係は「2.5 マイコンオプション」の cpu オプションの項目を参照してください。

INC_RX、INC_RXA、HLNK_LIBRARY1、HLNK_LIBRARY2、HLNK_LIBRARY3、および HLNK_DIR で複数ディレクトリを指定する場合は、";"(セミコロン)で区切ってください。

ccrx コマンド実行時は、BIN_RX、INC_RX、TMP_RX については、設定済みの環境変数の値がある場合はその値を使用し、ない場合は設定省略時の解釈の値を使用します。

これらの環境変数は、インストール時に作成されるバッチファイル setccrx.bat を実行することで簡単に設定できます。setccrx.bat は＜High-performance Embedded Workshop 格納ディレクトリ＞¥Tools¥Renesas¥RX¥1_0_0 または ＜ccrx 格納ディレクトリ＞¥..に格納されています。

6.2 プリデファインドマクロ

オプション指定やバージョンに合わせて、以下のようなプリデファインドマクロが定義されます。

表6.2 コンパイラのプリデファインドマクロ

	オプション	プリデファインドマクロ	
1	cpu=rx600	#define __RX600	1
	cpu=rx200	#define __RX200	1
2	endian=big	#define __BIG	1
	endian=little	#define __LIT	1
3	dbl_size=4	#define __DBL4	1
	dbl_size=8	#define __DBL8	1
4	int_to_short	#define __ INT_SHORT	1
5	signed_char	#define __SCHAR	1
	unsigned_char	#define __UCHAR	1
6	signed_bitfield	#define __SBIT	1
	unsigned_bitfield	#define __UBIT	1
7	round=zero	#define __ROZ	1
	round=nearest	#define __RON	1
8	denormalize=off	#define __DOFF	1
	denormalize=on	#define __DON	1
9	bit_order=left	#define __BITLEFT	1
	bit_order=right	#define __BITRIGHT	1
10	auto_enum	#define __AUTO_ENUM	1
11	library=function	#define __FUNCTION_LIB	1
	library=intrinsic	#define __INTRINSIC_LIB	1
12	fpu	#define __FPU	1
13	−	#define __RENESAS__ *[1]	1
14	−	#define __RENESAS_VERSION__ *[1]	0xAABBCC00 *[2]
15	−	#define __RX*[1]	1
16	pic	#define __PIC	1
17	pid	#define __PID	1

【注】 *1 オプションに関わらず常に定義されます。

*2 バージョンが V.AA.BB.CC の場合、__RENESAS_VERSION__ の値は 0xAABBCC00 となります。
例) V.1.01.00 の場合、#define __RENESAS_VERSION__ 0x01010000

表6.3 アセンブラのプリデファインドマクロ

	オプション	プリデファインドマクロ	
1	cpu=rx600	__RX600	.DEFINE 1
	cpu=rx200	__RX200	.DEFINE 1
2	endian=big	__BIG	.DEFINE 1
	endian=little	__LITTLE	.DEFINE 1
3	−	__RENESAS_VERSION__ *1	.DEFINE AABBCC00H *2
4	−	__RX *1	.DEFINE 1

【注】 *1 オプションに関わらず常に定義されます。

*2 バージョンが V.AA.BB.CC の場合、__RENESAS_VERSION__ の値は 0xAABBCC00 となります。
例) V.1.01.00 の場合、__RENESAS_VERSION__ .DEFINE 01010000H

7. ファイル仕様

7.1 ファイル名の付け方

ファイル名指定時に拡張子を省略した場合、標準のファイル拡張子を付加したファイル名を使用します。統合開発環境で使用する標準のファイル拡張子を**表** 7.1に示します。

表 7.1 **統合開発環境で使用する標準のファイル拡張子**

No.	拡張子	意味
1	c	C ソースプログラムファイル
2	cpp,cc,cp	C++ソースプログラムファイル
3	h	インクルードファイル
4	p	C プログラム用プリプロセッサ展開ファイル
5	pp	C++プログラム用プリプロセッサ展開ファイル
6	src	アセンブリソースプログラムファイル
7	lst	アセンブリプログラム用リストファイル
8	obj	リロケータブルオブジェクトプログラムファイル
9	abs	アブソリュートロードモジュールファイル
10	map	リンケージリストファイル
11	lib	ライブラリファイル
12	lbp	ライブラリリストファイル
13	mot	モトローラ S フォーマット
14	hex	インテル(拡張)HEX フォーマット
15	bin	バイナリファイル
16	sni	スタック情報ファイル
17	pro	プロファイル情報ファイル
18	dbg	デバッグ情報ファイル
19	rti	拡張子 td のファイルで指定された定義を含むオブジェクトファイル
20	cal	呼び出し情報ファイル
21	bls	外部シンボル割り付け情報ファイル
22	jmp	ジャンプテーブルファイル (アセンブリ言語)
23	fsy	シンボルアドレスファイル (アセンブリ言語)

【注】 ファイル名について
コンパイラで入力可能なファイル名は次の条件を満たす必要があります。
- OS **で記述可能なファイル名であること**
- **ハイフン**('-')**で始まらないこと**

rti_ではじまるファイル名はシステム予約名ですので使用しないでください。

tpldirのフォルダの下に一時的に出力される、各ファイルの拡張子を**表** 7.2に示します。

表 7.2 tpldir フォルダ出力ファイル

No.	拡張子	意味
1	td	tentative 定義の変数情報
2	ti	テンプレート情報ファイル
3	pi	パラメータ情報ファイル
4	ii	インスタンス情報ファイル

7.2 ソースリストの参照方法

7.2.1 ソースリストの構成

ソースリストファイルには、コンパイル結果およびアセンブル結果の情報を表示します。

ソースリストの構成と内容を**表** 7.3に示します。

表 7.3 ソースリストの構成と内容

No.	リストファイルへ表示する情報	内容	サブオプション*	show オプション省略時
1	ソース情報	アセンブリソースに対応して、C/C++言語ソースを表示	show=source	出力しない
2	オブジェクト情報	オブジェクトプログラムの機械語、アセンブリソースコード	なし	出力する
3	統計情報	エラーの総数、ソースプログラムの行数、セクションサイズ	なし	出力する
4	コマンド指定情報	コマンドで指定されたファイル名とオプションを表示	なし	出力する

【注】 * listfile オプションを指定した場合に有効です。

7.2.2 ソース情報

ソース情報は、show=source オプションを指定することでオブジェクト情報に埋め込まれる形で出力されます。出力例は、「7.2.3 **オブジェクト情報**」を参照ください。

7.2.3 オブジェクト情報

オブジェクト情報の出力例を**図** 7.1 に示します。

```
* RX FAMILY ASSEMBLER V.1.00.00 *  SOURCE LIST  Sat May 16 11:56:15 2009

LOC.     OBJ.              0XMDA SOURCE STATEMENT
(1)      (2)                (3)      (4)

                                   ;C LABEL    INSTRUCTION OPERAND    COMMENT
                                      (5)          (6)                 (7)
                                   ;LineNo. C-SOURCE STATEMENT
                                      (8)          (9)

                                               .SECTION    P,CODE
                                   ;       1 #include      "include.h"
                                   ;       1 extern int    x;
                                   ;       2 extern int    y = 1;
                                   ;       2 int func01(int);
                                   ;       3 int func03(int);
                                   ;       4
                                   ;       5 int func02(int z)
                                               .glb        _func02
00000000                           _func02:                            ; function: func02
                                               .STACK      _func02=8
00000000 7EA6                                  PUSH.L      R6
00000002                           L10:
                                               .LINE       "D:\RXC\work\list\now\sample.c",7
                                   ;       6 {
                                   ;       7       x = func01(z);
00000002 EF16                                  MOV.L       R1,R6
00000004 05rrrrrr             A                BSR         _func01
00000008 FB42rrrrrrrr                          MOV.L       #_x,R4
0000000E E341                                  MOV.L       R1,[R4]
                                               .LINE       "D:\RXC\work\list\now\sample.c",8
                                   ;       8       if (z == 2) {
00000010 6126                                  CMP         #02H,R6
00000012 18                    S               BNE         L12
00000013                           L11:
                                               .LINE       "D:\RXC\work\list\now\sample.c",9
                                   ;       9               x++;
00000013 711501                                ADD         #01H,R1,R5
00000016 E345                                  MOV.L       R5,[R4]
00000018 2E11                  B               BRA         L13
0000001A                           L12:
                                               .LINE       "D:\RXC\work\list\now\sample.c",11
                                   ;      10       } else {
                                   ;      11               x = func03(x + 2);
0000001A 6221                                  ADD         #02H,R1
0000001C 391200                W               BSR         _func03
0000001F FB42rrrrrrrr                          MOV.L       #_x,R4
00000025 E341                                  MOV.L       R1,[R4]
00000027 EF15                                  MOV.L       R1,R5
00000029                           L13:
                                               .LINE       "D:\RXC\work\list\now\sample.c",13
                                   ;      12       }
                                   ;      13       return x;
00000029 EF51                                  MOV.L       R5,R1
0000002B 3F6601                                RTSD        #04H,R6-R6
                                               .LINE       "D:\RXC\work\list\now\sample.c",16
                                   ;      14 }
                                   ;      15
                                   ;      16 int func03(int p)
                                               .glb        _func03
0000002E                           _func03:                            ; function: func03
                                               .STACK      _func03=4
0000002E                           L14:
                                               .LINE       "D:\RXC\work\list\now\sample.c",18
                                   ;      17 {
                                   ;      18       return p+1;
0000002E 6211                                  ADD         #01H,R1
00000030 02                                    RTS
                                   ;      19 }
                                               .glb        _x
                                               .glb        _func01
                                               .SECTION    D,ROMDATA,ALIGN=4
                                               .glb        _y
00000000                           _y:                                 ; static: y
00000000 01000000                              .lword       00000001H
                                               .END
```

図 7.1 オブジェクト情報の出力例

(1) ロケーション情報(LOC.)

アセンブル時に決定できる範囲のオブジェクトコードのロケーションアドレスを出力します。

(2) オブジェクトコード情報(OBJ.)

ニーモニックに対応するオブジェクトコードを出力します。

(3) 行情報(0XMDA)

アセンブラがソースを処理した結果の情報を出力します。各記号の意味を下記に示します。

表7.4 **アセンブリソースの行情報**

0	X	M	D	A	内 容
0-30					**インクルードファイルのネストレベルを示します。**
	X				-show=conditions **指定時、条件アセンブルで条件が偽となった行を示します。**
		M			-show=expansions **指定時、マクロ命令の展開行であることを示します。**
		D			-show=definitions **指定時、マクロ命令の定義行であることを示します。**
			S		**分岐距離指定子** S **を選択したことを示します。**
			B		**分岐距離指定子** B **を選択したことを示します。**
			W		**分岐距離指定子** W **を選択したことを示します。**
			A		**分岐距離指定子** A **を選択したことを示します。**
				*	**条件分岐命令に対して代替命令を選択したことを示します。**

(4) ソース情報(SOURCE STATEMENT)

アセンブリソースファイルの内容を表示します。

(5) ラベル情報(C LABEL)

(6) アセンブラ命令列(INSTRUCTION OPERAND)

コンパイラの出力したアセンブラ命令列を表示します。

(7) アセンブリソースプログラムに対応するコメント(COMMENT)

(8) C/C++**ソース行番号**(LineNo.)

(9) C/C++**ソース**(C-SOURCE STATEMENT)

show=source オプションを指定した場合、C/C++ソースを出力します。

7.2.4 統計情報

統計情報の出力例を**図** 7.2 に示します。

```
Information List  (1)

TOTAL ERROR(S)    00000
TOTAL WARNING(S)  00000
TOTAL LINE(S)     00071   LINES

Section List  (2)

Attr         Size               Name
CODE     0000000047(0000002FH)  P
ROMDATA  0000000004(00000004H)  D
```

図 7.2 **統計情報の出力例**

(1) エラー、警告それぞれのメッセージ数と、ソース行の総数

(2) セクション情報(セクション属性、サイズ、セクション名)

7.2.5 コンパイラのコマンド指定情報

コンパイラを起動したときのコマンドで指定されたファイル名とオプションを表示します。コンパイラのコマンド指定情報は、リストファイルの先頭に出力されます。コマンド指定情報の出力例を**図** 7.3 に示します。

```
;*** CPU TYPE ***  (1)

;-CPU=RX600

;*** COMMAND PARAMETER ***  (2)

;-output=src=C:\tmp\elp1894\sample.src
;-nologo
;-show=source
;sample.c
```

図 7.3 **コマンド指定情報**

(1) 選択されているマイコン

(2) コンパイラに渡したファイル名とオプション

7.2.6 アセンブラのコマンド指定情報

アセンブラを起動したときのコマンドで指定されたファイル名とオプションを表示します。アセンブラのコマンド指定情報は、リストファイルの最後に出力されます。コマンド指定情報の出力例を**図** 7.4 に示します。

```
Cpu Type   (1)

-CPU=RX600

Command Parameter   (2)

-output=sample.obj
-nologo
-listfile=sample.lst
```

図 7.4 **コマンド指定情報**

(1) アセンブラで選択されているマイコン

(2) アセンブラに渡したファイル名とオプション

7.3 リンケージリストの参照方法

最適化リンケージエディタが出力するリンケージリストの内容と形式について説明します。

7.3.1 リンケージリストの構成

リンケージリストの構成と内容を表 7.5に示します。

表7.5 リンケージリストの構成と内容

No.	リストファイルへ表示する情報	内容	show オプション * 指定	show オプション省略時
1	オプション情報	コマンドライン、サブコマンドで指定したオプション列を表示	なし	出力する
2	エラー情報	エラーメッセージを表示	なし	出力する
3	リンケージマップ情報	セクション名、先頭／最終アドレス、サイズ、種別を表示	なし	出力する
4	シンボル情報	静的定義シンボル名、アドレス、サイズ、種別をアドレス順に表示 show=reference を指定した場合は、各シンボルの参照回数、最適化実行有無も表示	show=symbol show=reference	出力しない 出力しない
5	シンボル削除最適化情報	最適化で削除したシンボルを表示	show=symbol	出力しない
6	クロスリファレンス情報	シンボルの参照情報を表示	show=xreference	出力しない
7	合計セクションサイズ	RAM,ROM,およびプログラムセクションの合計サイズを表示	show=total_size	出力しない
8	ベクタ情報	ベクタ番号とアドレスの情報を表示	show=vector	出力しない
9	CRC 情報	CRC の演算結果および出力位置アドレスを表示	なし	CRC オプション指定時は常に出力

【注】 * show オプションは list オプションを指定した場合に有効です。

7.3.2 オプション情報

コマンドライン、サブコマンドファイルで指定したオプション列を出力します。オプション情報の出力例を**図** 7.5 に示します(optlnk -sub=test.sub -list -show 指定時)。

```
（test.subの内容）
INPUT test.obj
```

```
*** Options ***

-sub=test.sub
INPUT test.obj (2)   } (1)
-list                }
-show                }
```

図7.5 **オプション情報の出力例（リンケージリスト）**

(1) コマンドライン、サブコマンドで指定したオプション列を、指定順に出力します。

(2) サブコマンドファイルtest.sub内のサブコマンドです。

7.3.3 エラー情報

エラーメッセージを出力します。エラー情報の出力例を**図** 7.6 に示します。

```
*** Error Information ***

** L2310 (E) Undefined external symbol “strcmp” referred to in “test.obj”
```

図7.6 **エラー情報の出力例（リンケージリスト）**

(1) エラーメッセージを出力します。

7.3.4 リンケージマップ情報

各セクションの先頭／最終アドレス、サイズ、種別をアドレス順に出力します。リンケージマップ情報の出力例を**図** 7.7 に示します。

```
*** Mapping List ***

SECTION                          START      END        SIZE   ALIGN
  (1)                             (2)       (3)         (4)    (5)

P
                                 00001000  00001000       1   1
C
                                 00001004  00001007       4   4
D_2
                                 00001008  000014dd     4d6   2
B_2
                                 000014de  000050b3     3bd6  2
```

図7.7 **リンケージマップ情報の出力例(リンケージリスト)**

(1) セクション名を表示します。

(2) 先頭アドレスを表示します。

(3) 最終アドレスを表示します。

(4) セクションサイズを表示します。

(5) セクションのアライメント数を表示します。

7.3.5 シンボル情報

show=symbol を指定した場合、外部定義シンボルまたは静的内部定義シンボルのアドレス、サイズ、種別をアドレス順に出力します。また、show=reference を指定した場合は、各シンボルの参照回数、最適化実行の有無も出力します。シンボル情報の出力例を**図** 7.8 に示します。

```
*** Symbol List ***

SECTION=(1)
FILE=(2)          START          END            SIZE
                   (3)           (4)             (5)
  SYMBOL          ADDR           SIZE           INFO              COUNTS  OPT
  (6)              (7)           (8)             (9)               (10)   (11)

SECTION=P
FILE=test.obj
                  00000000       00000428           428
  _main
                  00000000              2       func ,g               0
  _malloc
                  00000000             32       func ,l               0
FILE=mvn3
                  00000428       00000490            68
  $MVN#3
                  00000428              0       none ,g               0
```

図7.8　**シンボル情報の出力例(リンケージリスト)**

(1) セクション名を表示します。

(2) ファイル名を表示します。

(3) (2)のファイルに含まれる該当セクションの先頭アドレスを表示します。

(4) (2)のファイルに含まれる該当セクションの最終アドレスを表示します。

(5) (2)のファイルに含まれる該当セクションのセクションサイズを表示します。

(6) シンボル名を表示します。

(7) シンボルアドレスを表示します。

(8) シンボルサイズを表示します。

(9) シンボル種別を次のように表示します。

データ種別：	func	……	関数名
	data	……	変数名
	entry	……	エントリ関数名
	none	……	未設定(ラベル、アセンブラシンボル)
宣言種別　：	g	……	外部定義
	l	……	内部定義

(10) シンボル参照回数を表示します。show=referenceを指定した場合のみ表示します。参照回数を表示しないときは、*を表示します。

(11) 最適化有無を次のように表示します。

ch	‥‥‥‥	最適化によって変更されたシンボル
cr	‥‥‥‥	最適化によって生成されたシンボル
mv	‥‥‥‥	最適化によって移動されたシンボル

7.3.6 シンボル削除最適化情報

シンボル削除最適化(optimize=symbol_delete)によって削除されたシンボルのサイズ、種別を出力します。シンボル削除最適化情報の出力例を**図** 7.9 に示します。

```
*** Delete Symbols ***

SYMBOL                          SIZE    INFO
  (1)                           (2)     (3)
  _Version
                                   4    data ,g
```

図7.9　**シンボル削除情報の出力例(リンケージリスト)**

(1) 削除シンボル名を表示します。

(2) 削除シンボルサイズを表示します。

(3) 削除シンボルの種別を以下のように表示します。

データ種別：	func	‥‥‥‥	関数名
	data	‥‥‥‥	変数名
宣言種別　：	g	‥‥‥‥	外部定義
	l	‥‥‥‥	内部定義

7.3.7 クロスリファレンス情報

show=xreference を指定した場合、シンボルの参照情報(クロスリファレンス情報)を出力します。クロスリファレンス情報の出力例を**図** 7.10 に示します。

```
 *** Cross Reference List ***

 No    Unit Name  Global.Symbol   Location    External Information
(1)      (2)          (3)          (4)              (5)
0001  a
      SECTION=P  _func
                                 00000100
                 _func1
                                 00000116
                 _main
                                 0000012c
                 _g
                                 00000136
      SECTION=B
                 _a
                                 00000190    0001(00000140:P)
                                             0002(00000178:P)
                                             0003(0000018c:P)
0002  b
      SECTION=P
                 _func01
                                 00000154    0001(00000148:P)
                 _func02
                                 00000166    0001(00000150:P)
0003  c
      SECTION=P
                 _func03
                                 00000184
```

図7.10 **クロスリファレンス情報の出力例(リンケージリスト)**

(1) Unit番号。オブジェクト単位の識別番号。

(2) オブジェクト名。リンク時の入力指定順になる。

(3) シンボル名。セクションごとに配置アドレスの昇順に出力される。

(4) シンボルの配置アドレス。

form=rel指定時は、セクション先頭からの相対値となる。

(5) 参照している外部シンボルのアドレスを表す。

出力形式は以下のようになる。

<Unit番号>(<アドレス or セクション内オフセット>:<セクション名>)

7.3.8 合計セクションサイズ

ROM セクション、RAM セクション、およびプログラムセクションの合計サイズを出力します。合計の出力例を**図** 7.11 に示します。

```
*** Total Section Size ***

RAMDATA SECTION:        00000660 Byte(s)
(1)
ROMDATA SECTION:        00000174 Byte(s)
(2)
PROGRAM SECTION:        000016d6 Byte(s)
(3)
```

図7.11 **合計セクションサイズの出力例(リンケージリスト)**

(1) RAMデータセクションの合計サイズ。

(2) ROMデータセクションの合計サイズ。

(3) プログラムセクションの合計サイズ。

7.3.9 ベクタ情報

show=vector を指定した場合、可変ベクタテーブルの内容を表示します。合計の出力例を**図** 7.12 に示します。

```
*** Variable Vector Table List ***

NO.     SYMBOL/ADDRESS
(1)     (2)
  0     $fdummy
  1     $fa
  2     00ff8800
  3     $fdummy
    :
  <省略>
```

図7.12 **ベクタ情報の出力例(リンケージリスト)**

(1) ベクタ番号。

(2) シンボルを表示します。シンボルが定義されていない場合はアドレスで表示します。

7.3.10 CRC情報

CRC オプション指定時に CRC の演算結果および出力位置アドレスを出力します。

```
*** CRC Code ***

CODE    : cb0b
(1)
ADDRESS : 00007ffe
(2)
```

図7.13 CRC **情報の出力例(リンケージリスト)**

(1) CRC演算結果

(2) CRCの演算結果の出力位置アドレス

7.4 ライブラリリストの参照方法

本節では、最適化リンケージエディタが出力するライブラリリストの内容と形式について説明します。

7.4.1 ライブラリリストの構成

ライブラリリストの構成と内容を**表** 7.6に示します。

表7.6 **ライブラリリストの構成と内容**

No.	リストの作成	内容	サブオプション*	show オプション省略時
1	オプション情報	コマンドライン、サブコマンドで指定したオプション列を表示	—	出力する
2	エラー情報	エラーメッセージを表示	—	出力する
3	ライブラリ情報	ライブラリ情報を表示	—	出力する
4	ライブラリ内モジュール、セクション、シンボル情報	ライブラリ内モジュールを表示	—	出力する
		show=symbol を指定した場合は、モジュール内シンボル名一覧も表示	show=symbol	出力しない
		show=section を指定した場合は、各モジュール内セクション名、シンボル名一覧も表示	show=section	出力しない

【注】 * **すべてのオプションは、**list **オプションを指定した場合に有効です。**

7.4.2 オプション情報

コマンドライン、サブコマンドファイルで指定したオプション列を出力します。オプション情報の出力例を**図** 7.14 に示します(optlnk -sub=test.sub -list –show を指定した場合)。

```
（test.subの内容）
form    library
in      adhry.obj
output  test.lib
```

```
*** Options ***

-sub=test.sub
form    library      ┐
in      adhry.obj    ┘(2)   ┐
output  test.lib             ┘(1)
-list                        ┐
-show                        ┘
```

図7.14 **オプション情報の出力例(ライブラリリスト)**

(1) コマンドライン、サブコマンドで指定したオプション列を、指定順に出力します。

(2) サブコマンドファイルtest.sub内のサブコマンドです。

7.4.3 エラー情報

エラー、ウォーニングなどのメッセージを出力します。エラー情報の出力例を**図** 7.15 に示します。

```
*** Error Information ***

** L1200 (W) Backed up file “main.lib” into “main.lbk”    (1)
```

図7.15 **エラー情報の出力例(ライブラリリスト)**

(1) ウォーニングメッセージを出力します。

7.4.4 ライブラリ情報

ライブラリの種別を出力します。ライブラリ情報の出力例を**図** 7.16 に示します。

```
*** Library Information ***

LIBRARY NAME=test.lib   (1)
CPU=SuperH              (2)
ENDIAN=Big              (3)
ATTRIBUTE=system        (4)
NUMBER OF MODULE=1      (5)
```

図7.16 **ライブラリ情報の出力例(ライブラリリスト)**

(1) ライブラリ名を表示します。

(2) マイコン名を表示します。

(3) エンディアン種別を表示します。

(4) ライブラリファイルの属性がシステムライブラリかユーザライブラリかを表示します。

(5) ライブラリ内モジュール数を表示します。

7.4.5 ライブラリ内モジュール、セクション、シンボル情報

ライブラリ内のモジュール一覧を出力します。

show=symbol を指定した場合はモジュール内シンボル名一覧を、show=section を指定した場合はモジュール内セクション名、シンボル名一覧を出力します。

ライブラリ内モジュール、セクション、シンボル情報の出力例を**図** 7.17 に示します。

```
*** Library List ***

MODULE           LAST UPDATE
 (1)                 (2)
  SECTION
    (3)
    SYMBOL
      (4)
adhry
                 29-Feb-2000 12:34:56
  P
   _main
   _Proc0
   _Proc1
  C
  D
   _Version
  B
   _IntGlob
   _CharGlob
```

図7.17 **ライブラリ内モジュール、セクション、シンボル情報の出力例(ライブラリリスト)**

(1) モジュール名を表示します。

(2) モジュールを登録した日付を表示します。モジュールが更新された場合は、最新の更新日付を表示します。

(3) モジュール内セクション名を表示します。

(4) セクション内をシンボル表示します。

8. プログラミング

8.1 プログラムの構造

8.1.1 セクション

アセンブラが出力するリロケータブルファイルの実行命令、データの各領域は、セクションを構成します。セクションは、メモリ上の配置を行う最小単位です。セクションの性質には、以下の項目があります。

- セクション属性

  code　実行命令を格納します。

  data　変更可能なデータを格納します。

  romdata　固定データを格納します。

- 形式種別

  相対アドレス形式 ・・・・・・・・ 最適化リンケージエディタで再配置可能なセクションです。

  絶対アドレス形式 ・・・・・・・・ アドレス決定済みのセクションです。最適化リンケージエディタで再配置できません。

- 初期値

  プログラム実行開始時の初期値の有無です。同一セクション内で初期値があるデータと初期値がないデータは混在できません。一つでも初期値があると、初期値のない領域は0で初期化します。

- 書き込み操作

  プログラム実行時における書き込み操作の可/不可を示します。

- アライメント数

  セクションの配置アドレスを補正するための値です。最適化リンケージエディタでは、各セクションの配置アドレスを、それぞれのアライメント数の倍数になるように補正します。

8.1.2 C/C++プログラムのセクション

C/C++プログラム、標準ライブラリの使用メモリ領域の種類とセクションとの対応を**表** 8.1に示します。

表 8.1 **メモリ領域の種類とその性質の概要**

No.	名称	セクション		形式	初期値	アライメント数	内容
		名称	属性	種別	書き込み操作		
1	プログラム領域	P*1*6	code	相対	有 不可	1byte*7	機械語を格納
2	定数領域	C*1*2*6*8	romdata	相対	有 不可	4byte	const 型のデータを格納
		C_2*1*2*6*8	romdata	相対	有 不可	2byte	
		C_1*1*2*6*8	romdata	相対	有 不可	1byte	
3	初期化データ領域	D*1*2*6*8	romdata	相対	有 可	4byte	初期値のあるデータを格納
		D_2*1*2*6*8	romdata	相対	有 可	2byte	
		D_1*1*2*6*8	romdata	相対	有 可	1byte	
4	未初期化データ領域	B*1*2*6*8	data	相対	無 可	4byte	初期値のないデータを格納
		B_2*1*2*6*8	data	相対	無 可	2byte	
		B_1*1*2*6*8	data	相対	無 可	1byte	
5	switch 文分岐テーブル領域	W*1*2	romdata	相対	有 不可	4byte	switch 文の分岐テーブルを格納
		W_2*1*2	romdata	相対	有 不可	2byte	
		W_1*1*2	romdata	相対	有 不可	1byte	
6	C++初期処理／後処理データ領域	C$INIT	romdata	相対	有 不可	4byte	グローバルクラスオブジェクトに対して呼び出されるコンストラクタおよびデストラクタのアドレスを格納
7	C++仮想関数表領域	C$VTBL	romdata	相対	有 不可	4byte	クラス宣言中に仮想関数があるときに仮想関数をコールするためのデータを格納

No.	名称	セクション		形式	初期値	アライメント数	内容
		名称	属性	種別	書き込み操作		
8	ユーザスタック領域	SU	data	相対	無 可	4byte	プログラム実行に必要な領域
9	割り込みスタック領域	SI	data	相対	無 可	4byte	プログラム実行に必要な領域
10	ヒープ領域	―	―	相対	無 可	―	ライブラリ関数 malloc、realloc、calloc、new で使用する領域
11	絶対アドレス変数領域	$ADDR_<section>_<address> *3	data	絶対	有/無 可/不可 *4	―	#pragma address 指定した変数を格納
12	可変ベクタ領域	C$VECT	romdata	相対	無 可	4byte	可変ベクタテーブル
13	リテラル領域	L*5	data	相対	有 可/不可	4byte	文字列リテラルおよび集成体の動的初期化で用いる初期化子を格納

【注】 *1 section オプションでセクション名を切り替えることができます。

*2 セクション名切り替えの際に、アライメント数が 4 のセクションを指定することで、アライメントが 1 または 2 のセクション名も変更されます。

*3 <section>は C,D,B のセクション名称、<address>は絶対アドレス値(16 進数)になります。

*4 初期値、書き込み操作は<section>の属性に従います。

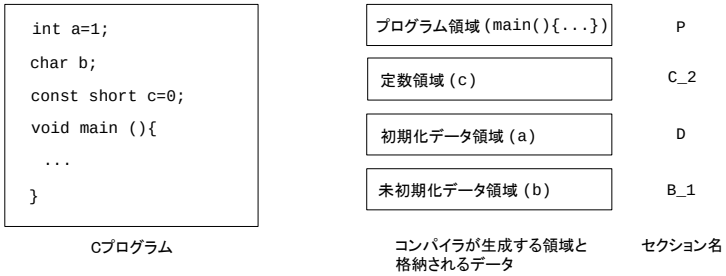
*5 section オプションでセクション名を変更することができます。このとき、変更後の名前に C セクションを選択することも可能です。

*6 #pragma section でセクション名を変更することができます。

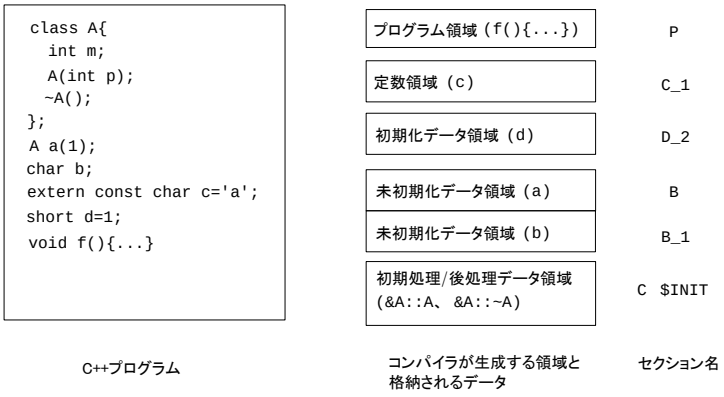
*7 instalign4 オプション、instalign8 オプション、#pragma instalign4 または #pragma instalign8 のいずれかを使用すると、アライメント数は 4 または 8 になります。

*8 #pragma endian で endian オプションと異なる指定のエンディアンを指定した場合、#pragma endian big であれば_B を、#pragma endian little であれば_L を、セクション名の後ろに付加した専用のセクションを生成し、該当データを格納します。

例1. Cプログラムとコンパイラ生成セクションとの対応をプログラム例を用いて示します。



例2. C++プログラムとコンパイラ生成セクションとの対応をプログラム例を用いて示します。



8.1.3 アセンブリプログラムのセクション

アセンブリプログラムでは、.SECTION 制御命令を用いてセクションの開始や属性を、.ORG 制御命令を用いてセクションの形式種別を、それぞれ宣言します。

各制御命令の詳細については「10.3 **アセンブラ制御命令の記述方法**」を参照してください。

例：アセンブリプログラムのセクション宣言例を示します。

```
            .SECTION        A,CODE,ALIGN=4    ;(1)

START:
            MOV.L           #CONST,R4
            MOV.L           [R4],R5
            ADD             #10,R5,R3
            MOV.L           #100,R4
            MOV.L           #ARRAY,R5
LOOP:
            MOV.L           R3,[R5+]
            SUB             #1,R4
            CMP             #0,R4
            BNE             LOOP
EXIT:
            RTS

;
            .SECTION        B,ROMDATA         ;(2)
            .ORG            02000H
            .glb            CONST
CONST:
            .LWORD          05H
;
            .SECTION        C,DATA,ALIGN=4    ;(3)
            .glb            BASE
BASE:
            .blkl           100
            .END
```

(1) **セクション名** A**、アライメント数** 4**、相対アドレス形式の** code **セクションを宣言しています。**

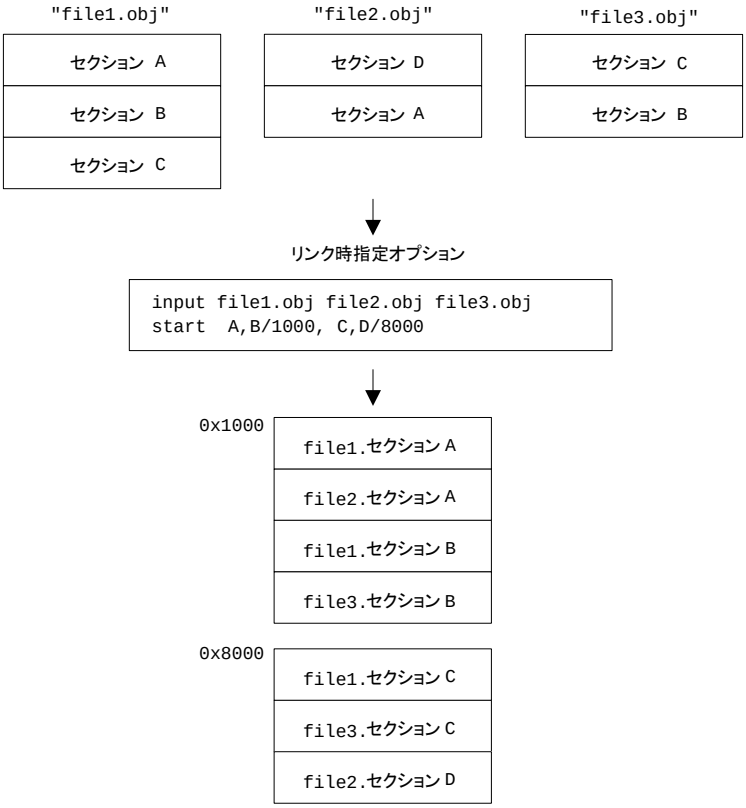
(2) **セクション名** B**、割り付けアドレス** 2000H**、絶対アドレス形式の** romdata **セクションを宣言しています。**

(3) **セクション名** C**、アライメント数** 4**、相対アドレス形式の** stack **セクションを宣言しています。**

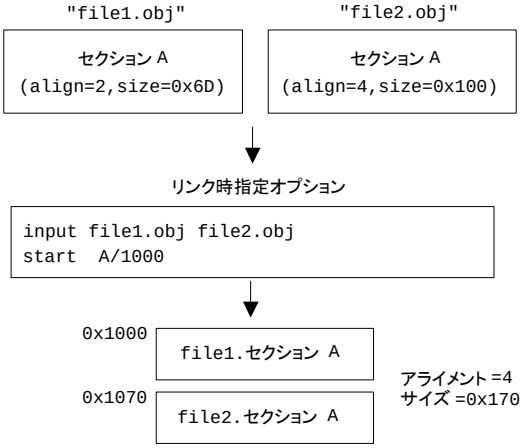
8.1.4 セクションの結合

最適化リンケージエディタでは、入力リロケータブルファイル内の同一セクションを結合し、start オプションによって指定されたアドレスに割り付けます。

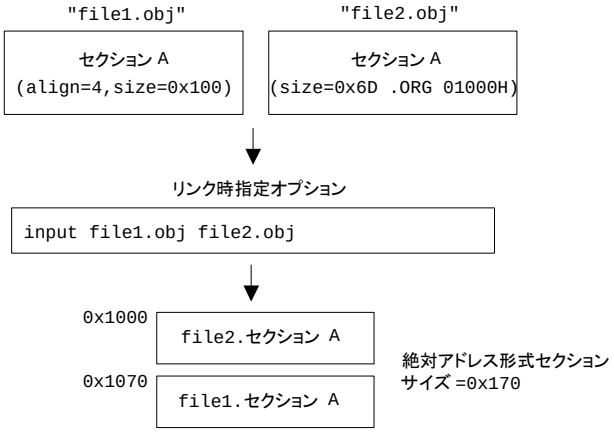
1. 異なるファイルの同名セクションは、ファイルの入力順に連続して割り付けます。



2. アライメント数の異なる同名セクションは、アライメント調整後に結合します。セクションのアライメント数は大きい方に合わせます。

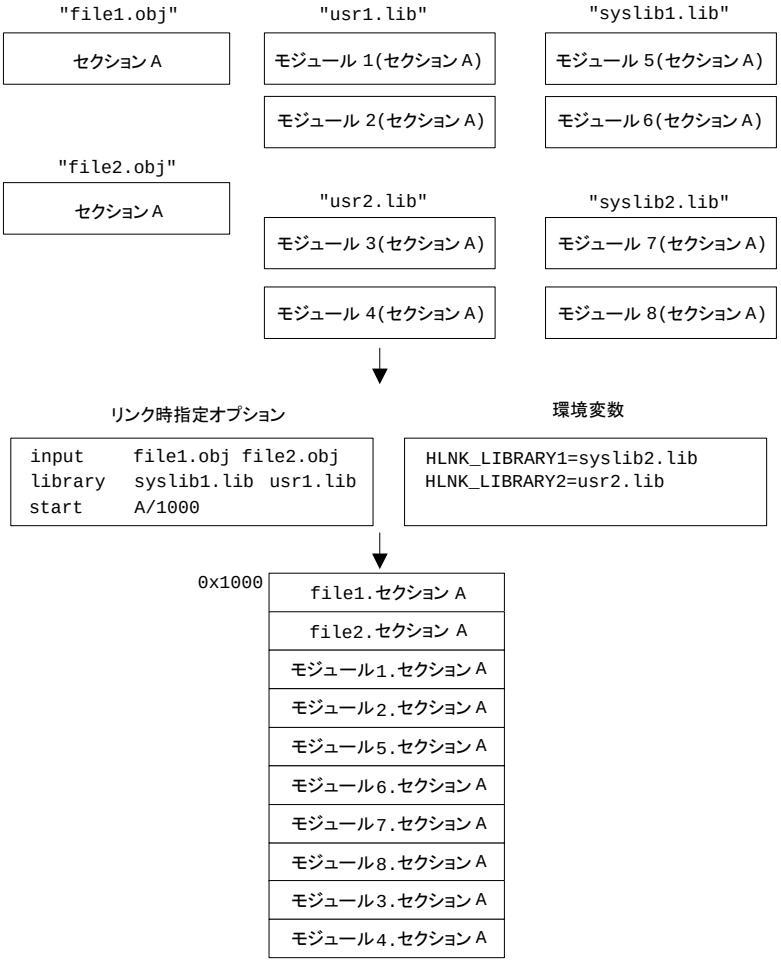


3. 同名セクションに絶対アドレス形式と相対アドレス形式が含まれている場合、絶対アドレス形式セクションの後に相対アドレス形式セクションを結合します。



4. 同名セクションの結合順序に関する規則は、優先度の高い順に以下の通りです。

- inputオプションまたはコマンドライン上の入力ファイル指定順
- libraryオプションのユーザライブラリ指定順およびライブラリ内モジュール入力順
- libraryオプションのシステムライブラリ指定順およびライブラリ内モジュール入力順
- 環境変数(HLNK_LIBRARY1～3)のライブラリ指定順およびライブラリ内モジュール入力順



8.2 関数呼び出しインタフェース

関数呼び出しを行う際の、コンパイラのレジスタおよびスタック領域を使用する規則について説明します。C/C++プログラムとアセンブリプログラム間で相互に関数呼び出しを行う場合は、これらの規則を守ってアセンブリプログラムを作成する必要があります。

- スタックに関する規則
- レジスタに関する規則
- 引数の設定、参照に関する規則
- リターン値の設定、参照に関する規則
- 外部名の相互参照方法

8.2.1 スタックに関する規則

(1) スタックポインタ

スタックポインタの指すアドレスよりも下位(0 番地の方向)のスタック領域に有効なデータを格納してはいけません。スタックポインタより下位アドレスに格納されたデータは、割り込み処理で破壊される可能性があります。

(2) スタックフレームの割り付け、解放

関数呼び出しが行われた時点(JSR または BSR 命令の実行直後)では、スタックポインタは呼び出した関数側で使用したスタックの最下位アドレスを指しています。このスタックポインタの指している領域より上位アドレスのデータの割り付け、設定は呼び出す側の関数の役目です。

関数のリターン時は、呼び出された関数で確保した領域を解放してから、通常 RTS 命令を用いて呼び出した関数へ返ります。これより上位アドレスの領域(リターン値アドレスおよび引数の領域)は、呼び出した側の関数で解放します。

図 8.1 は、関数呼び出し直後のスタックフレームの状態を説明したものです。

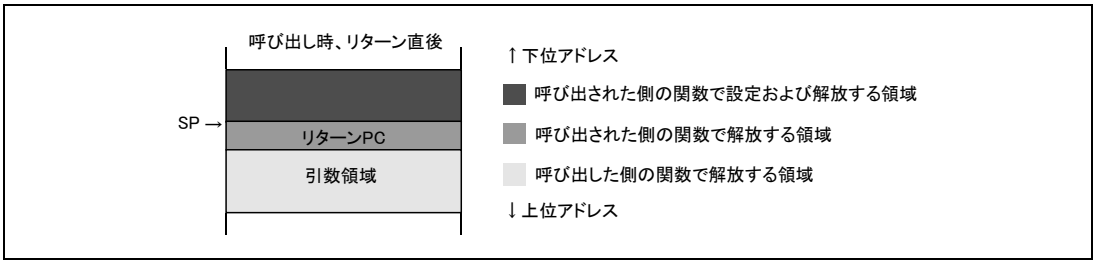


図 8.1 **スタックフレームの割り付け、解放に関する規則**

8.2.2 レジスタに関する規則

関数呼び出し前後において、レジスタの値が同一であることを保証するかどうかは、レジスタにより異なります。また、オプションにより特定の用途向けに使用するレジスタがあります。レジスタの使用規則を**表** 8.2 に示します。

表 8.2 **レジスタ使用規則**

レジスタ	関数呼び出し前後で値を保証	関数入口	関数出口	高速割り込み用レジスタ*1	ベースレジスタ *2	PID レジスタ *3
R0	保証する	スタックポインタ	スタックポインタ	-	-	-
R1	保証しない	引数 1	戻り値 1	-	-	-
R2	保証しない	引数 2	戻り値 2	-	-	-
R3	保証しない	引数 3	戻り値 3	-	-	-
R4	保証しない	引数 4	戻り値 4	-	-	-
R5	保証しない	-	(不定)	-	-	-
R6	保証する	-	(入口の値を保持)	-	-	-
R7	保証する	-	(入口の値を保持)	-	-	-
R8	保証する	-	(入口の値を保持)	-	○	-
R9	保証する	-	(入口の値を保持)	-	○	○
R10	保証する	-	(入口の値を保持)	○	○	○
R11	保証する	-	(入口の値を保持)	○	○	○
R12	保証する	-	(入口の値を保持)	○	○	○
R13	保証する	-	(入口の値を保持)	○	○	○
R14	保証しない	-	(不定)	-	-	-
R15	保証しない	構造体戻り値へのポインタ	(不定)	-	-	-
ISP USP	スタックポインタの場合は R0 と同じ。 そうでない場合は変化しません。*4			-	-	-
PC	-	プログラムカウンタ*5		-	-	-
PSW	保証しない	-	(不定)	-	-	-
FPSW	保証しない	-	(不定)	-	-	-
ACC	保証しない*6	-	(不定) *6	-	-	-
INTB BPC BPSW FINTV CPEN	-	変化しません*4	-	-	-	-

【注】 *1 R10～R13 の 4 本は、fint_register オプションにより、一部または全部が「高速割り込み機能」に使われることがあります。「高速割り込み機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

*2 R8～R13 の 6 本は、base オプションにより、一部または全部が「ベースレジスタ機能」に使われることがあります。「ベースレジスタ機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

*3 R9～R13 のうちの 1 本は、pid オプションにより「PID 機能」に使われることがあります。「PID 機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

*4 組み込み関数または#pragma inline_asm で、これらのレジスタを設定したり更新したりする場合を除きます。

*5 関数の呼び出しに使用する命令の仕様に従います。関数の呼び出しには、BSR, JSR, BRA および JMP のいずれかの命令を用います。

*6 ACC(アキュムレータ)を更新する命令は、RX のソフトウェアマニュアルを参照してください。

8.2.3 引数の設定、参照に関する規則

引数に対する一般的な規則と、引数の割り付け方について述べます。

引数が実際どのように割り付けられるかは、**「8.2.5 引数割り付けの具体例」**を参照ください。

(1) 引数の渡し方

引数の値を、必ずレジスタまたはスタック上の引数の割り付け領域にコピーしたあとで関数を呼び出します。呼び出した側の関数では、リターン後に引数の割り付け領域を参照することはありませんので、呼び出された側の関数で引数の値を変更しても呼び出した側の処理は直接には影響を受けません。

(2) 型変換の規則

(a) 関数原型によって型が宣言されている引数は、宣言された型に変換します。

(b) 関数原型によって型が宣言されていない引数の型変換は、以下の規則に従って変換します。

- 2バイト以下の整数型は、4バイト整数型に変換されます。
- float型の引数は、double型に変換します。
- 上記以外の引数は、変換しません。

例

```
void p(int,... );
void f( )
{
  char c;
  p(1.0, c);
}
```

cは、対応する引数の型宣言がないので、4バイト整数型に変換されます。

1.0は、対応する引数の型がint型なので、4バイト整数型に変換されます。

(3) 引数の割り付け領域

引数は、レジスタに割り付ける場合とスタック上の引数領域に割り付ける場合があります。引数の割り付け領域を**図** 8.2 に示します。

通常、ソースプログラムにおける引数の宣言順に、番号の小さいレジスタから順に割り付けを行い、レジスタが全て割り付いたらスタックに割り付けます。但し、可変個の引数を持つ関数など、レジスタが余っていてもスタックに割り付ける場合もあります。また、C++プログラムの非静的関数メンバの this ポインタは、常に R1 に割り付けられます。

引数割り付け領域の一般規則を**表** 8.3 にそれぞれ示します。

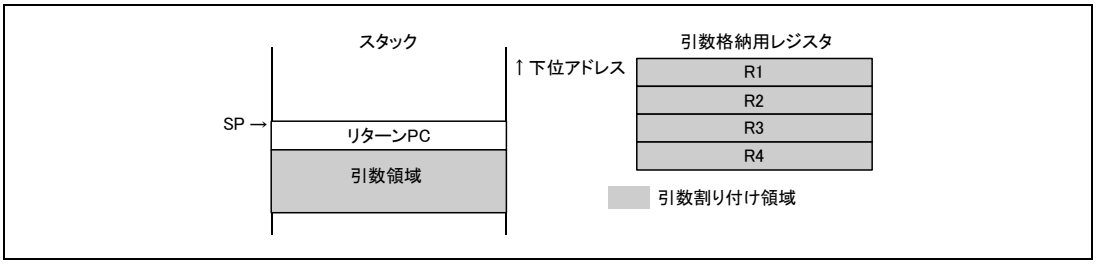


図 8.2　**引数の割り付け領域**

表 8.3　**引数割り付け領域の一般規則**

レジスタで渡される引数			**スタック渡しになる引数**
対象の型	**引数格納用レジスタ**	**割り付け方**	
signed char, (unsigned)char, bool, _Bool, (signed)short, unsigned short, (signed)int, unsigned int, (signed)long, unsigned long, float、double*1, long double*1, **ポインタ**, **データメンバへのポインタ**, **リファレンス**	R1～R4 **のうち** 1 **つ**	signed char, (signed)short **は符号拡張、**(unsigned)char, unsigned short **はゼロ拡張を行った結果を割り付ける** **その他の型はそのままレジスタに割り付ける**	(1) **引数の型がレジスタ渡しの対象の型以外のもの** (2) **関数原型により可変個の引数を持つ関数として宣言しているもの***3 (3) R1～R4 **のうち、まだ他の引数に割り当てられていないものの本数が、割り当てに必要な本数より少ない場合**
(signed)long long, unsigned long long, double*2, long double*2	R1～R4 **のうち** 2 **つ**	**下位** 4 **バイトを番号の少ない方に、上位** 4 **バイトを番号の大きい方に割り付ける**	
16 **バイト以内でサイズが** 4 **の倍数の構造体型、共用体型、クラス型**	R1～R4 **のうち、サイズを** 4 **で割った数**	**メモリイメージの先頭から** 4 **バイトずつ、レジスタ番号が増える方向に割り付ける**	

【注】　*1　dbl_size=8 **を指定しなかった場合です。**

*2　dbl_size=8 **を指定した場合です。**

*3　**関数原型により可変個の引数をもつ関数として宣言している場合、宣言の中で対応する型のない引数およびその直前の引数はスタック渡しになります。型のない引数は、**2 **バイト以下の整数は** long **型に、**float **型は** double **型にそれぞれ変換して、全て境界調整数が** 4 **の引数として取り扱います。**

例

```
int f2(int,int,int,int,...);
:
f2(a,b,c,x,y,z); → x、y、z はスタック渡しになります。
```

(4) スタック渡しとなる引数の割り付け方

表 8.3 で、スタック渡しとなる引数の、配置アドレス、およびスタックへの配置の仕方は以下となります。

- 各引数は、その境界調整数に応じたアドレスに配置します。
- 引数並びの左から右の順に、スタックが深くなる方向に配置されるように、スタックの引数用領域に格納します。すなわち、引数Aとその右隣の引数Bがともにスタック渡しとなる場合、引数Bのアドレスは、引数Aの配置アドレスに引数Aの占有サイズを加えたアドレスを、引数Bの境界調整数に整合させたアドレス、となります。

8.2.4 リターン値の設定、参照に関する規則

リターン値に対する一般的な規則と、リターン値の設定場所について述べます。

(1) リターン値の型変換

リターン値は、その関数の返す型に変換します。

例

```
long f( );
long f( )
{
    float x;
    return x;  <── 関数原型にしたがってリターン値はlong型に変換されます。
}
```

(2) リターン値の設定場所

関数のリターン値の型によっては、リターン値をレジスタに設定する場合とメモリに設定する場合があります。リターン値の型と設定場所の関係は**表** 8.4 を参照してください。

表 8.4 リターン値の型と設定場所

No.	リターン値の型	リターン値の設定場所
1	singed char, (unsigned)char, (singed)short, unsigned short, (singed)int, unsigned int, (signed)long, unsigned long, float, double*2, long double*2, ポインタ, bool, _Bool, リファレンス, データメンバへのポインタ	R1 但し、signed char, (signed)short は符号拡張、(unsigned)char, unsigned short はゼロ拡張を行った結果を設定
2	double*3, long double*3, (signed)long long, unsigned long long	R1, R2 下位 4 バイトを R1 に、上位 4 バイトを R2 に設定
3	16 バイト以内かつ 4 の倍数であるサイズの構造体、共用体、クラス型	メモリイメージの先頭から 4 バイトずつ R1,R2,R3,R4 の順に設定
4	3.以外の構造体、共用体、クラス型	リターン値設定領域(メモリ)*1

【注】 *1 関数のリターン値をメモリに設定する場合、リターン値はリターン値アドレスの指す領域に設定します。呼び出す側では、引数領域のほかにリターン値設定領域を確保し、そのアドレスを R15 に設定してから関数を呼び出します。

*2 dbl_size=8 を指定しなかった場合です。

*3 dbl_size=8 を指定した場合です。

8.2.5 引数割り付けの具体例

引数割り付けの具体例を示します。なお、アドレスは全ての図で右から左に向かって増加します(左側が上位アドレス)。

例1. レジスタ渡しの対象の型である引数は、宣言順にレジスタR1～R4に割り付けます。

途中でレジスタ渡しとならない引数があった場合、それ以降の引数はレジスタ渡しの対象となります。スタック上では、その引数の境界調整数に補正したアドレスに配置されます。

```
int f(
    unsigned char,
    long long,
    long long,
    short,
    int,
    char,
    short,
    char,
    char,
    short);
        :
f(1,2,3,4,5,6,7,8,9,10);
 /*
** 1, 2, 4 がレジスタ渡しとなる
*/
```

<レジスタ>

R1	0x000000(ゼロ拡張)	0x01
R2	0x00000002	
R3	0x00000000	
R4	0x0000(符号拡張)	0x0004

<スタック>

*(R0+0)	0x00000003		
*(R0+4)	0x00000000		
*(R0+8)	0x00000005		
*(R0+12)	0x0007	空領域	0x06
*(R0+16)	0x000A	0x09	0x08

例2. サイズが16バイト以下、かつ4の倍数である構造体および共用体型の引数は、レジスタ渡しの対象となります。それ以外の構造体および共用体型の引数は、スタック渡しとなります。

```
union U {int a[2]; int b;} u;
struct S {short d; char c[4];} s;
struct T {char g; char f[2]; char e;} t;
int f(union U, struct S, struct T);
:
f(u, s, t);
/*
** uは8バイトなのでレジスタ渡し
** sは6バイトなのでスタック渡し
** tは4バイトなのでレジスタ渡し
*/
```

<レジスタ>

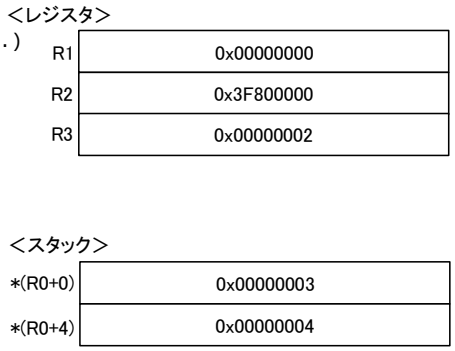
R1	u.a[0] (=u.b)			
R2	u.a[1]			
R3	e	f[1]	f[0]	g

<スタック>

*(R0+0)	s.c[1]	s.c[0]	s.d	
*(R0+4)	空領域		s.c[3]	s.c[2]

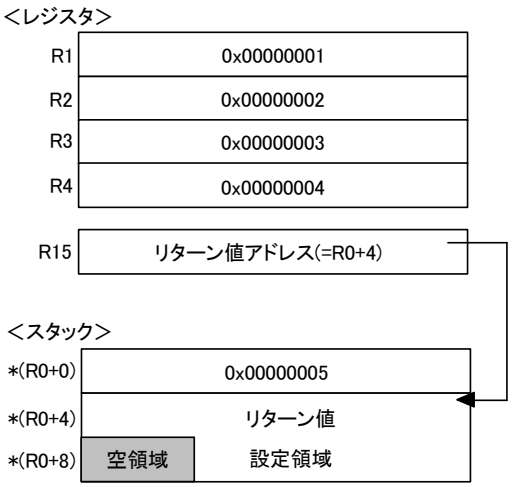
例3. 関数原型により可変個の引数を持つ関数として宣言している場合、対応する型のない引数およびその直前の引数は、宣言順にスタック渡しになります。

```
int f(int, float, int, int, ...)
    :
f(0, 1.0, 2, 3, 4)
```



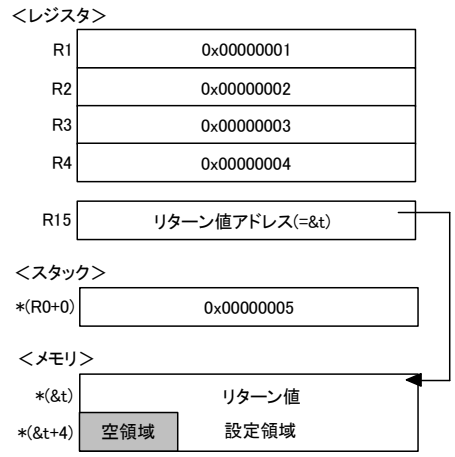
例4. 関数の返す型が16バイトを超える、または4の倍数でないサイズの構造体または共用体型の場合、R15にリターン値アドレスを設定します。

```
struct S{char a[7];};
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
f(1,2,3,4,5);
```



例5. リターン値をメモリに設定する場合、通常例4のようにスタックを確保して設定しますが、リターン値を変数に設定する場合、スタックを確保せず、その変数のメモリ領域に直接設定します。この場合、R15にはその変数のアドレスを設定します。

```
struct S{char a[7];}t;
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
t=f(1,2,3,4,5);
```



8.2.6 外部名の相互参照方法

C/C++プログラムの中で外部名として宣言されたものは、アセンブリプログラムとの間で相互に参照あるいは更新することができます。コンパイラは、次のものを外部名として扱います。

- 大域変数であって、かつ static 記憶クラスでないもの(C/C++プログラム)
- extern 記憶クラスで宣言されている変数名(C/C++プログラム)
- static 記憶クラスを指定されていない関数名(C プログラム)
- static 記憶クラスを指定されてない非メンバ非インライン関数名(C++プログラム)
- 非インラインメンバ関数名(C++プログラム)
- 静的データメンバ名(C++プログラム)

(1) アセンブリプログラムの外部名をC/C++プログラムで参照する方法

アセンブリプログラムでは、.glb を用いてシンボル名(先頭に下線"_"を付与)を外部定義宣言します。

C/C++プログラムでは、シンボル名(先頭に下線"_"がない)を「extern」宣言します。

アセンブリプログラム(定義する側)

```
      .glb    _a, _b
      .SECTION D,ROMDATA,ALIGN=4
 _a:  .LWORD 1
 _b:  .LWORD 1
      .END
```

C/C++プログラム(参照する側)

```
      extern int a,b;

      void f()
      {
        a+=b;
      }
```

(2) C/C++プログラムの外部(変数およびC関数)名をアセンブリプログラムから参照する方法

C/C++プログラムでは、変数名(先頭に下線"_"がない)を外部定義します。

アセンブリプログラムでは、~~.IMPORT~~ .GLB を用いて外部名(先頭に下線"_"を付与)を外部参照宣言します。

C/C++プログラム(定義する側)

```
int a;
```

アセンブリプログラム(参照する側)

```
        .GLB       _a
        .SECTION   P,CODE
        MOV.L      #A_a,R1
        MOV.L      [R1],R2
        ADD        #1,R2
        MOV.L      R2,[R1]
        RTS
        .SECTION   D,ROMDATA,ALIGN=4
A_a:    .LWORD     _a
        .END
```

(3) C++プログラムの外部(関数)名をアセンブリプログラムから参照する方法

アセンブリプログラムで参照する関数を「extern "C"」を用いて宣言することにより、(2)と同じ規則で参照できます。ただし、「extern "C"」を用いて宣言した関数は多重定義できません。

C++プログラム(呼び出される側)

```
 extern "C"
void sub ( )
{
    :
}
```

アセンブリプログラム(呼び出す側)

```
        .GLB    _sub
        .SECTION P,CODE
                :


PUSH.L  R13
MOV.L   4[R0],R1
MOV.L   R3,R12
MOV.L   #_sub,R14
JSR     R14
POP     R13
RTS
                :
        .END
```

8.3 スタートアッププログラムの作成

本章では、プログラムの実行に必要な環境を設定するための処理について説明します。ただし、プログラムを実行する環境はユーザシステムごとに異なりますので、ユーザシステムの仕様に合わせて実行環境の設定プログラムを作成する必要があります。

本章の内容は、標準となるスタートアップについて説明しています。PIC/PID 機能におけるアプリケーションに使用するスタートアップは特別な対応が必要ですので「8.4.7 アプリケーションのスタートアップ」も参照ください。

必要な手続きの概要は以下の通りです。

- 固定ベクタテーブルの設定

  パワーオンリセットで初期設定ルーチン (PowerON_Reset)が起動されるように、固定ベクタテーブルを設定します。固定ベクタテーブルには、リセットベクタの他、特権命令例外、アクセス例外、未定義命令例外、浮動小数点例外、ノンマスカブル割り込みの各処理ルーチンを登録することができます。

- 初期設定

  main関数に到達するまでに必要な手続きを行います。レジスタやセクションの初期化、各種初期設定ルーチンの呼び出しを行います。

- 低水準インタフェースルーチンの作成

  標準入出力(stdio.h、ios、streambuf、istream、ostream)、メモリ管理ライブラリ(stdlib.h、new)を使用する場合に必要なライブラリ関数とユーザシステムとの間のインタフェースルーチンです。

- 終了処理ルーチン(exit、atexit、abort)*1の作成

  プログラムの終了処理を行います。

【注】 *1 **プログラムの終了処理を行う C ライブラリ関数** exit、atexit、abort **関数を使用する場合は、ユーザシステムにあわせてこれらの関数を作成する必要があります。**
C++**プログラムを使用する場合、または C ライブラリ関数** assert **マクロを使用する場合、**abort **関数は必ず作成する必要があります。**

8.3.1 固定ベクタテーブルの設定

パワーオンリセットで、初期設定ルーチン PowerON_Reset が呼び出されるようにするためには、固定ベクタテーブルのリセットベクタに PowerON_Reset のアドレスを設定します。以下にそのコーディング例を示します。

固定ベクタテーブルには、リセットベクタの他、特権命令例外、アクセス例外、未定義命令例外、浮動小数点例外、ノンマスカブル割り込みの各処理ルーチンを登録することができます。

なお、固定ベクタテーブルの詳細については、ハードウェアマニュアル等を参照してください。

例

```
extern void PowerON_Reset_PC(void);

#pragma section VECTTBL  /* #pragma section宣言によりRESET_Vectorsを     */
                         /* CVECTTBLセクションに出力します。               */
                         /* リンク時にstartオプションでCVECTTBLセクションを  */
                         /* リセットベクタに割り付けるよう指定します。         */
void (*const RESET_Vectors[])(void)={
   PowerON_Reset_PC,
};
```

8.3.2 初期設定

初期設定ルーチン PowerON_Reset は、main 関数を実行する前、および実行した後に必要な手続きを記述する関数です。初期設定ルーチンの中で必要となる処理を、順番に記述します。

(1) 初期設定処理向けのPSWレジスタ初期化

初期設定処理を行うための、PSW レジスタ初期化を実施します。たとえば、初期設定処理中、割り込みを受け付けない設定にするために、PSW に対して割り込み禁止の設定をします。

リセット時の PSW は全 bit ゼロに初期化され、割り込み許可ビット(I ビット)も割り込み禁止状態(ゼロ)として初期化されています。

(2) スタックポインタの初期化

スタックポインタ(USP レジスタおよび ISP レジスタ)を初期化します。PowerON_Reset 関数に対して"#pragma entry"宣言することにより、コンパイラが自動的に ISP/USP 初期化コードを関数先頭に生成します。

PowerON_Reset 関数は、#pragma entry 宣言されているため、この手続きを記述する必要はありません。

(3) ベースレジスタに使用する汎用レジスタの初期化

コンパイラで base オプションを使用している場合、プログラム全体でベースアドレスとして使用する汎用レジスタを初期化する必要があります。PowerON_Reset 関数に対して"#pragma entry"宣言することにより、コンパイラが自動的に各レジスタへの初期化コードを関数先頭に生成します。

PowerON_Reset 関数は、#pragma entry 宣言されているため、この手続きを記述する必要はありません。

(4) 各種制御レジスタの初期化

可変ベクタテーブルの配置アドレスを、INTB に書き込みます。その他、必要に応じて、FINTV、FPSW、BPC、BPSW の初期化を行います。これらの初期化は、コンパイラの組み込み関数を使って行うことができます。

ただし、PSW だけは、割り込みマスク設定を維持するため、初期化はまだ行いません。

(5) セクションの初期化処理

RAM 領域セクションの初期化用ルーチン(_INITSCT)を呼び出します。未初期化データセクションはゼロ初期化されます。初期化データセクションは、ROM 領域の初期値を RAM 領域へコピーします。_INITSCT は、標準ライブラリとして提供されます。

初期化対象のセクションは、ユーザがセクション初期化用テーブル(DTBL,BTBL)へ記述する必要があります。_INITSCT 関数が使用するセクションの先頭アドレスおよび最終アドレスを、セクションアドレス演算子を用いて設定します。

セクション初期化用テーブルのセクション名は、未初期化データ領域を C$BSEC、初期化データ領域を C$DSEC で宣言します。

以下にコーディング例を示します。

例

```
#pragma section C C$DSEC          //セクション名を C$DSEC にします
extern const struct {
    void *rom_s;                  //初期化データセクションの ROM 上の先頭アドレスメンバ
    void *rom_e;                  //初期化データセクションの ROM 上の最終アドレスメンバ
    void *ram_s;                  //初期化データセクションの RAM 上の先頭アドレスメンバ
} DTBL[] = {__sectop("D"), __secend("D"), __sectop("R")};

#pragma section C C$BSEC          //セクション名を C$BSEC にします
extern const struct {
    void *b_s;                    //未初期化データセクションの先頭アドレスメンバ
    void *b_e;                    //未初期化データセクションの最終アドレスメンバ
} BTBL[] = {__sectop("B"), __secend("B")};
```

(6) ライブラリの初期化処理

C/C++言語ライブラリ関数使用時に、必要な初期化を実施するルーチン(_INITLIB)を呼び出します。

実際に使用する機能に合わせた必要最低限の初期設定を行うために、以下の指針を参考にしてください。

- 作成した低水準インタフェースルーチンの中で初期設定が必要な場合、低水準インタフェースルーチンの仕様に合わせた初期設定(_INIT_LOWLEVEL)が必要です。
- rand関数、strtok関数を使用する場合、標準入出力以外の初期設定(_INIT_OTHERLIB)が必要です。

ライブラリの初期設定を行うプログラム例を以下に示します。

```
#include <stdio.h>
#include <stdlib.h>
#define IOSTREAM 3
const size_t _sbrk_size = 520;          // ヒープ領域確保サイズの最小単位を指定します
                                        // (省略時：1024)
extern char *_s1ptr;

#ifdef __cplusplus
extern "C" {
#endif
void _INITLIB (void)
{
    _INIT_LOWLEVEL();                   // 低水準インタフェースルーチンの初期設定をします
    _INIT_IOLIB();                      // 入出力ライブラリの初期設定をします
    _INIT_OTHERLIB();                   // rand 関数、strtok 関数の初期設定をします
}

void _INIT_LOWLEVEL (void)
{                                       // 低水準ライブラリで必要な初期設定をしてください
}

void _INIT_OTHERLIB(void)
{
    srand(1);                           // rand 関数を使用する場合の初期設定です
    _s1ptr=NULL;                        // strtok 関数を使用する場合の初期設定です
}
#ifdef __cplusplus
}
#endif
```

【注】 *1 **標準入出力ファイルのファイル名を指定します。この名前は、低水準インタフェースルーチン「open」で使用します。**

*2 **コンソール等対話的な装置の場合、バッファリングを行わないためのフラグを立てます。**

(7) グローバルクラスオブジェクトの初期化

C++言語のプログラム開発の場合、グローバルに宣言されたクラスオブジェクトのコンストラクタを呼び出すためのルーチン(_CALL_INIT)を呼び出します。_CALL_INIT は、標準ライブラリとして提供されます。

(8) main関数実行向けのPSW初期化

PSW レジスタを初期化します。割り込みマスクの設定も、ここで解除します。

(9) PSWのPMビットの変更

リセット以降は特権モード(PSW の PM ビットがゼロ)で動作していますが、ユーザモードに切り替えたい場合は、組み込み関数の chg_pmusr を実行します。

chg_pmusr 関数にはいくつか注意事項がありますので、使用する場合は、9.2.2 組み込み関数の chg_pmusr の項目を参照ください。

（10）ユーザプログラムの実行

main 関数を実行します。

（11）グローバルクラスオブジェクトの後処理

C++言語のプログラム開発の場合、グローバルに宣言されたクラスオブジェクトのデストラクタを呼び出すためのルーチン(_CALL_END)を呼び出します。_CALL_END は、標準ライブラリとして提供されます。

8.3.3 初期設定ルーチンの記述例

「8.3.2 **初期設定**」で説明した、PowerON_Reset 関数のコーディング例を示します。

なお、統合開発環境で生成される実際の初期設定ルーチンは、「8.3.6 統合開発環境で生成されるスタートアッププログラム」を参照ください。

```
#include    <machine.h>
#include    <_h_c_lib.h>
#include    "typedefine.h"
#include    "stacksct.h"

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset(void);
void main(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus                              // Use SIM I/O
extern "C" {
#endif
extern void _INITLIB(void);
#ifdef __cplusplus
}
#endif

#define PSW_init  0x00010000
#define FPSW_init 0x00000100

#pragma section ResetPRG
#pragma entry PowerON_Reset
void PowerON_Reset(void)
```

```
{
            set_intb(__sectop("C$VECT"));
            set_fpsw(FPSW_init);

            _INITSCT();
            _INITLIB();
            nop();
            set_psw(PSW_init);
            main();
            brk();
}
```

8.3.4 低水準インタフェースルーチン

標準入出力、メモリ管理ライブラリを C/C++プログラムで使用する場合は、低水準インタフェースルーチンを作成しなければなりません。**表** 8.5 に C ライブラリ関数で使用している低水準インタフェースルーチンの一覧を示します。

表 8.5 **低水準インタフェースルーチンの一覧**

No.	名称	機能
1	open	**ファイルのオープン**
2	close	**ファイルのクローズ**
3	read	**ファイルからの読み込み**
4	write	**ファイルへの書き出し**
5	lseek	**ファイルの読み込み／書き出しの位置の設定**
6	sbrk	**メモリ領域の確保**
7	error_addr*	errno **アドレスの取得**
8	wait_sem*	**セマフォの確保**
9	signal_sem*	**セマフォの解放**

【注】 * **リエントラントライブラリを使用する場合に必要です。**

低水準インタフェースルーチンで必要な初期化は、プログラム起動時に行う必要があります。これは、ライブラリ初期設定処理_INITLIB の中の「_INIT_LOWLEVEL」という関数の中で行ってください。

以下、低水準入出力の基本的な考え方を説明したあと、各インタフェースルーチンの仕様を説明します。

【注】 **関数名** open、close、read、write、lseek、sbrk、error_addr、wait_sem、signal_sem **は低水準インタフェースルーチンの予約済み識別子です。ユーザプログラム中では使用しないでください。**

(1) 入出力の考え方

標準入出力ライブラリでは、ファイルを FILE 型のデータによって管理しますが、低水準インタフェースルーチンでは、実際のファイルと 1 対 1 に対応する正の整数を与え、これによって管理します。この整数をファイル番号といいます。

open ルーチンでは、与えられたファイル名に対してファイル番号を与えます。open ルーチンでは、この番号によってファイルの入出力ができるように、以下の情報を設定する必要があります。

- ファイルのデバイスの種類(コンソール、プリンタ、ディスクファイル等)。
  コンソールやプリンタ等の特殊なデバイスに対しては、特別なファイル名をシステムで決めておいてopen ルーチンで判定する必要があります。
- ファイルのバッファリングはバッファの位置、サイズ等の情報。
- ディスクファイルは、ファイルの先頭から次に読み込み/書き出しを行う位置までのバイトオフセット。

open ルーチンで設定した情報に基づいて、以後、入出力(read、write ルーチン)、読み込み/書き出し位置の設定(lseek ルーチン)を行います。

close ルーチンでは、出力ファイルのバッファリングを行っている場合はバッファの内容を実際のファイルに書き出し、open ルーチンで設定したデータの領域が再使用できるようにしてください。

(2) 低水準インタフェースルーチンの仕様

本項では低水準インタフェースルーチンを作成するための仕様を説明します。以下、各ルーチンごとに、ルーチンを呼び出す際のインタフェースとその動作および実現上の注意事項を示します。

各ルーチンのインタフェースは以下の形式で示します。なお、低水準インタフェースルーチンは必ず関数原型にしてください。また C++プログラム内で宣言する場合は「extern "C"」を付加してください。

凡例：

簡易説明

(ルーチン名)

説　明	(ルーチンの機能概要を示します)	
リターン値	正常：	(正常に終了した場合のリターン値を示します)
	異常：	(エラーが生じた場合のリターン値を示します)
引　数	(名前)	(意味)
	(インタフェースに示す引数名です)	(引数として渡される値を示します)

ファイルオープン

*long open(const char *name, long mode, long flg)*

説　明　第 1 引数のファイル名に対応するファイルを操作するための準備をします。
open ルーチンでは、後で読み込み/書き出しを行うために、ファイルの種類(コンソール、プリンタ、ディスクファイル等)を決定しなければなりません。ファイルの種類は、以後 open ルーチンで返したファイル番号を用いて読み込み/書き出しを行うたびに参照する必要があります。
第 2 引数の mode は、ファイルをオープンする時の処理の指定です。このデータの各ビットの意味について以下に示します。

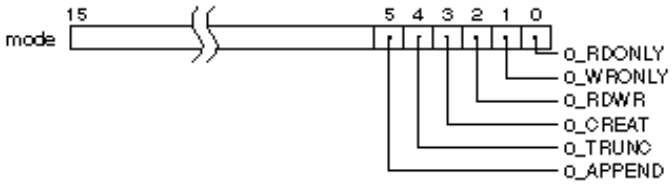


表 8.6　open ルーチン mode ビット説明

mode ビット	説明
O_RDONLY (0 ビット)	ビットが 1 のとき、ファイルを読み込み専用にオープン
O_WRONLY (1 ビット)	ビットが 1 のとき、ファイルを書き出し専用にオープン
O_RDWR (2 ビット)	ビットが 1 のとき、ファイルを読み込み、書き出し両用にオープン
O_CREAT (3 ビット)	ビットが 1 のとき、ファイル名で示すファイルが存在しない場合にファイルを新規に作成
O_TRUNC (4 ビット)	ビットが 1 のとき、ファイル名で示すファイルが存在する場合にファイルの内容を捨て、ファイルのサイズを 0 に更新
O_APPEND (5 ビット)	次に読み込み/書き出しを行うファイル内の位置を設定 ビットが 0 のとき：ファイルの先頭に設定 ビットが 1 のとき：ファイルの最後に設定

mode で示したファイルの処理の指定と実際のファイルの性質が矛盾する場合はエラーにしてください。正常にファイルがオープンできた場合は、以後の read、write、lseek、close ルーチンで使用されるファイル番号(正の整数)を返してください。ファイル番号と実際のファイルの対応は低水準インタフェースルーチンで管理する必要があります。オープンに失敗した場合は-1 を返してください。

リターン値　正常：　正常オープンしたファイルのファイル番号
　　異常：　-1

引　数　name　ファイルのファイル名を指す文字
　　mode　ファイルをオープンするときの処理の指定
　　flg　ファイルをオープンするときの処理の指定(常に 0777)

ファイルクローズ

long close(long fileno)

説　明　open ルーチンで得られたファイル番号が引数として渡されます。
open ルーチンで設定したファイル管理情報の領域を、再び使用できるように解放してください。また、低水準インタフェースルーチン内で出力ファイルのバッファリングを行っている場合は、バッファの内容を実際のファイルに書き出してください。
ファイルを正常にクローズできた場合は 0、失敗した場合は-1 を返してください。

リターン値　正常：　0
異常：　-1

引　数　fileno　クローズするファイル番号

データ読み込み

*long read(long fileno, unsigned char *buf, long count)*

説　明　第 1 引数(fileno)で示すファイルから、第 2 引数(buf)の指す領域へデータを読み込みます。読み込むデータのバイト数は第 3 引数(count)で示します。
ファイルが終了した場合、count で示されたバイト数以下のバイト数しか読み込むことができません。
ファイルの読み込み/書き出しの位置は、読み込んだバイト数だけ先に進みます。
正常に読み込みができた場合は、実際に読み込んだバイト数を返してください。読み込みに失敗した場合は-1 を返してください。

リターン値　正常：　実際に読み込んだバイト数
異常：　-1

引　数　fileno　読み込みの対象となるファイル番号
buf　読み込んだデータを格納する領域
count　読み込むバイト数

データ書き出し

*long write(long fileno, const unsigned char *buf, long count)*

説　明　第 2 引数(buf)の指す領域から、第 1 引数(fileno)の示すファイルにデータを書き出します。書き込むデータのバイト数は第 3 引数(count)で示します。
ファイルを書き出そうとしているデバイス(ディスク等)が満杯の時は、count で示されたバイト数以下のバイト数しか書き出すことができません。実際に書き出すことのできたバイト数が何度か連続して 0 バイトの場合、ディスクが満杯であると判断してエラー(-1)を返すように実現することをお勧めします。
ファイルの読み込み/書き出しの位置は、書き出したバイト数だけ先に進みます。
正常に書き出しができた場合は、実際に書き出したバイト数を返してください。書き出しに失敗した場合は-1 を返してください。

リターン値　正常：　実際に書き出されたバイト数
異常：　-1

引　数

fileno	書き出しの対象となるファイル番号
buf	書き出すデータ領域
count	書き出すバイト数

ファイル内位置設定

long lseek(long fileno, long offset, long base)

説　明　ファイルの読み込み/書き出しを行うファイル内の位置を、バイト単位で設定します。
新しいファイル内の位置は、第 3 引数(base)によって、以下の方法で計算し設定してください。

(1) base が 0 のとき	ファイルの先頭から offset バイトの位置に設定します。
(2) base が 1 のとき	現在の位置に offset バイトを加えた位置に設定します。
(3) base が 2 のとき	ファイルのサイズに offset バイトを加えた位置に設定します。

ファイルがコンソールやプリンタ等の対話的なデバイスの場合や、新しいオフセットの値が負になったり、(1)(2)のときファイルのサイズを超える場合はエラーにします。
正しくファイル位置を設定できた場合は、新しい読み込み/書き出し位置のファイルの先頭からのオフセットを、そうでない場合は-1 を返してください。

リターン値　正常：　新しいファイルの読み込み/書き出し位置のファイルの先頭からのオフセット(バイト単位)
異常：　-1

引　数

fileno	対象となるファイル番号
offset	読み込み/書き出しの位置を示すオフセット(バイト単位)
base	オフセットの起点

*char *sbrk(size_t size)*

説　明　メモリ領域を割り付けるサイズが引数として渡されます。
連続して sbrk ルーチンを呼び出す場合は、下位アドレスから順に連続した領域が割り付けられるようにしてください。割り付けるメモリ領域が不足した場合はエラーにしてください。
正常に割り付けができた場合は、割り付けた領域の先頭のアドレスを、割り付けに失敗した場合は、「(char *)-1」を返してください。

リターン値　正常：　割り付けた領域の先頭アドレス
異常：　(char *)-1

引　数　size　割り付けるデータのサイズ

*long *errno_addr(void)*

説　明　現在のタスクが持つエラー番号のアドレスを返却します。
標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

リターン値　現在のタスクが持つエラー番号のアドレス

セマフォ確保

long wait_sem(long semnum)

説　明　semnum で示されたセマフォを確保します。
確保できた場合は 1、確保できなかった場合は 0 を返してください。
標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

リターン値　正常：　1
異常：　0

引　数　semnum　セマフォ ID

セマフォ解放

long signal_sem(long semnum)

説　明　semnum で示されたセマフォを解放します。
解放できた場合は 1、解放できなかった場合は 0 を返してください。
標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

リターン値　正常：　1
異常：　0

引　数　semnum　セマフォ ID

(3) 低水準インタフェースルーチンコーディング例

```
/************************************************************************/
/*                                lowsrc.c:                             */
/*----------------------------------------------------------------------*/
/*     RX ファミリ  シミュレータ・デバッガ  インタフェースルーチン      */
/*           - 標準入出力(stdin,stdout,stderr)だけをサポートしています -*/
/************************************************************************/
#include <string.h>

/*  ファイル番号  */
#define STDIN  0                        /*  標準入力 (コンソール)  */
#define STDOUT 1                        /*  標準出力 (コンソール)  */
#define STDERR 2                        /*  標準エラー出力 (コンソール)*/

#define FLMIN 0                         /*  最小のファイル番号*/
#define FLMAX 3                         /*  ファイル数の最大値*/

/*  ファイルのフラグ  */
#define O_RDONLY 0x0001                 /*  読み込み専用*/
#define O_WRONLY 0x0002                 /*  書き出し専用*/
#define O_RDWR 0x0004                   /*  読み書き両用*/

/*  特殊文字コード  */
#define CR 0x0d                         /*  復帰*/
#define LF 0x0a                         /*  改行*/

/*  sbrk で管理する領域サイズ  */
#define HEAPSIZE 1024

/************************************************************************/
/*                          参照関数の宣言：                            */
/*    シミュレータ・デバッガでコンソールへの文字入出力を行うアセンブリプログラムの参照  */
/************************************************************************/
extern void charput(char);              /*  一文字入力処理*/
extern char charget(void);              /*  一文字出力処理*/

/************************************************************************/
/*                          静的変数の定義：                            */
/*              低水準インタフェースルーチンで使用する静的変数の定義    */
/************************************************************************/
char flmod[FLMAX];                      /*  オープンしたファイルのモード設定場所  */

union HEAP_TYPE{
   long dummy;                          /*  4 バイトアライメントにするためのダミー*/
   char heap[HEAPSIZE];                 /*  sbrk で管理する領域の宣言*/
};

static union HEAP_TYPE heap_area;

static char *brk=(char*)&heap_area;     /*  sbrk で割り付けた領域の最終アドレス  */
```

```
/************************************************************************/
/*                          open:ファイルのオープン                     */
/*                        リターン値：ファイル番号(成功)                */
/*                                    -1        (失敗)                  */
/************************************************************************/
long open(const char *name,           /*  ファイル名        */
         long mode,                   /*  ファイルのモード   */
         long flg)                    /*  処理の指定(未使用)*/
{
   /*  ファイル名に従ってモードをチェックし、ファイル番号を返す  */

   if (strcmp(name,"stdin")==0) {     /*  標準入力ファイル  */
      if ((mode&O_RDONLY)==0) {
         return (-1);
      }
      flmod[STDIN]=mode;
      return (STDIN);
   }

   else if (strcmp(name,"stdout")==0) {/*  標準出力ファイル  */
      if ((mode&O_WRONLY)==0) {
         return (-1);
      }
      flmod[STDOUT]=mode;
      return (STDOUT);
   }

   else if (strcmp(name,"stderr")==0){/* 標準エラー出力ファイル  */
      if ((mode&O_WRONLY)==0) {
         return (-1);
      }
      flmod[STDERR]=mode;
      return (STDERR);
   }

   else {
      return (-1);                    /* エラー  */
   }
}

/************************************************************************/
/*                          close:ファイルのクローズ                    */
/*                        リターン値： 0          (成功)                */
/*                                    -1        (失敗)                  */
/************************************************************************/
long close(long fileno)               /*  ファイル番号  */
{
   if (fileno<FLMIN || FLMAX<fileno) {/*  ファイル番号の範囲チェック  */
      return -1;
   }

   flmod[fileno]=0;                   /*  ファイルのモードリセット  */

   return 0;
}
```

```
/************************************************************************/
/*                          read:データの読み込み                       */
/*                  リターン値： 実際に読み込んだ文字数 (成功)          */
/*                               -1             (失敗)                  */
/************************************************************************/
long read(long fileno,              /*  ファイル番号          */
       unsigned char *buf,          /*  転送先バッファアドレス  */
       long  count)                 /*  読み込み文字数        */
{
   unsigned long i;

   /*  ファイル名に従ってモードをチェックし、一文字づつ入力してバッファに格納  */

   if (flmod[fileno]&O_RDONLY || flmod[fileno]&O_RDWR) {
      for (i=count;i>0;i--) {
         *buf=charget();
         if (*buf==CR) {            /*  改行文字の置き換え  */
            *buf=LF;
         }
      buf++;
      }
      return count;
   }

   else {
      return -1;
   }
}

/************************************************************************/
/*                          write:データの書き出し                      */
/*                  リターン値： 実際に書き出した文字数 (成功)          */
/*                               -1             (失敗)                  */
/************************************************************************/
long write(long fileno,             /*  ファイル番号  */
        const unsigned char *buf,   /*  転送元バッファアドレス  */
        long count)                 /*  書き出し文字数  */
{
   unsigned long i;
   unsigned char c;

   /*  ファイル名に従ってモードをチェックし、一文字づつ出力  */

   if (flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR) {
      for (i=count; i>0; i--) {
         c=*buf++;
         charput(c);
      }
      return count;
   }

   else {
      return -1;
   }
}
```

```
/************************************************************************/
/*                    lseek:ファイルの読み込み／書き出し位置の設定                    */
/*         リターン値：読み込み／書き出し位置のファイル先頭からのオフセット (成功)         */
/*                    -1                                    (失敗)         */
/*                (コンソール入出力では、lseek はサポートしていません)                */
/************************************************************************/
long lseek(long fileno,                 /*  ファイル番号            */
         long offset,                   /*  読み込み／書き出し位置    */
         long  base)                    /*  オフセットの起点          */
{
   return -1;
}

/************************************************************************/
/*                          sbrk:メモリ領域の割り付け                          */
/*                    リターン値： 割り付けた領域の先頭アドレス  (成功)                */
/*                             -1                          (失敗)                */
/************************************************************************/
char *sbrk(size_t size)                 /*  割り付ける領域のサイズ  */
{
   char *p;

   /*  空き領域のチェック  */

   if (brk+size>heap_area.heap+HEAPSIZE) {
      return (char *)-1;
   }

   p=brk;                               /* 領域の割り付け      */
   brk+=size;                           /* 最終アドレスの更新  */
   return p;
}
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           lowlvl.src                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  RX Family Simulator/Debugger Interface Routine   ;
;             - Inputs and outputs one character -                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          .GLB    _charput
          .GLB    _charget

SIM_IO    .EQU 0h

          .SECTION   P,CODE
;--------------------------------------------------------------------------
;  _charput:
;--------------------------------------------------------------------------
_charput:
          MOV.L      #IO_BUF,R2
          MOV.B      R1,[R2]
          MOV.L      #1220000h,R1
          MOV.L      #PARM,R3
          MOV.L      R2,[R3]
          MOV.L      R3,R2
          MOV.L      #SIM_IO,R3
          JSR        R3
          RTS

;--------------------------------------------------------------------------
;  _charget:
;--------------------------------------------------------------------------
_charget:
         MOV.L       #1210000h,R1
         MOV.L       #IO_BUF,R2
         MOV.L       #PARM,R3
         MOV.L       R2,[R3]
         MOV.L       R3,R2
         MOV.L       #SIM_IO,R3
         JSR         R3
         MOV.L       #IO_BUF,R2
         MOVU.B      [R2],R1
         RTS

;--------------------------------------------------------------------------
;  I/O Buffer
;--------------------------------------------------------------------------
          .SECTION  B,DATA,ALIGN=4
PARM:     .BLKL     1
          .SECTION  B_1,DATA
IO_BUF:   .BLKB     1
          .END
```

(4) リエントラントライブラリ用低水準インタフェースルーチン例

リエントラントライブラリ用低水準インタフェース例を示します。標準ライブラリ構築ツールで reent オプションを指定して作成したライブラリを使用する場合に必要になります。

wait_sem 関数、signal_sem 関数で NG が返った場合、errno に以下を設定し、ライブラリ関数からリターンします。

wait_sem	EMALRESM	malloc 用セマフォ資源確保に失敗しました
	ETOKRESM	strtok 用セマフォ資源確保に失敗しました
	EIOBRESM	_iob 用セマフォ資源確保に失敗しました
signal_sem	EMALFRSM	malloc 用セマフォ資源解放に失敗しました
	ETOKFRSM	strtok 用セマフォ資源解放に失敗しました
	EIOBRESM	_iob 用セマフォ資源解放に失敗しました

割り込みに関しては、セマフォ確保後により優先度の高い割り込みが発生し、セマフォ確保を行うとデッドロックが発生するため、リソースを共有するような処理が割り込みでネストしないようにしてください。

```
#define MALLOC_SEM            1     /* malloc 用セマフォ No.         */
#define STRTOK_SEM            2     /* strtok 用セマフォ No.         */
#define FILE_TBL_SEM          3     /* _iob 用セマフォ No.           */
#define SEMSIZE               4
#define TRUE                  1
#define FALSE                 0
#define OK                    1
#define NG                    0

extern long *errno_addr(void);
extern long wait_sem(long);
extern long signal_sem(long);

long sem_errno;
int force_fail_signal_sem = FALSE;
static int semaphore[SEMSIZE];

/*************************************************************************/
/*                      errno_addr:errno アドレスの取得                  */
/*                         リターン値：errno アドレス                    */
/*************************************************************************/
long *errno_addr(void)
{
    /* 現在のタスクの errno アドレスを返してください */
    return (&sem_errno);
}


/*************************************************************************/
/*                      wait_sem:指定されたセマフォの確保                */
/*                         リターン値：OK(=1) (成功)                     */
/*                                     NG(=0) (失敗)                     */
/*************************************************************************/
long wait_sem(long semnum)    /* セマフォ ID      */
{
    if((0 < semnum) && (semnum < SEMSIZE)) {
          if(semaphore[semnum] == FALSE) {
              semaphore[semnum] = TRUE;
              return(OK);
          }
    }
    return(NG);
}
```

```
/*****************************************************************************/
/*                     signal_sem:指定されたセマフォの解放                   */
/*                          リターン値：OK(=1) (成功)                        */
/*                                      NG(=0) (失敗)                        */
/*****************************************************************************/
long signal_sem(long semnum)  /* セマフォ ID      */
{
   if(!force_fail_signal_sem) {
        if((0 <= semnum) && (semnum < SEMSIZE)) {
           if( semaphore[semnum] == TRUE ) {
           semaphore[semnum] = FALSE;
           return(OK);
           }
        }
   }
   return(NG);
}
```

8.3.5 終了処理ルーチン

(1) 終了処理の登録と実行(atexit)ルーチンの作成例

終了処理の登録を行うライブラリ atexit 関数の作成法を示します。

atexit 関数では、引数として渡された関数のアドレスを、終了処理のテーブルに登録します。登録された関数の個数が限界値(ここでは、登録できる個数を 32 個とします)を超えた場合、あるいは同じ関数が二度以上登録された場合はリターン値として 0 以外(ここでは 1)を返します。そうでなければ 0 を返します。

以下にプログラム例を示します。

例:

```
#include <stdlib.h>

long _atexit_count=0 ;

void (*_atexit_buf[32])(void) ;

#ifdef __cplusplus
extern "C"
#endif
long atexit(void (*f)(void))
{
   int i;

   for(i=0; i<_atexit_count ; i++)             // 既に登録されていないかチェックします
      if(_atexit_buf[i]==f)
         return 1;
   if (_atexit_count==32)                      // 登録数の限界値をチェックします
      return 1;
   else {
      _atexit_buf[_atexit_count++]=f;          // 関数のアドレスを登録します
      return 0;
   }
}
```

(2) プログラムの終了(exit)ルーチンの作成例

プログラムの終了処理を行うライブラリ exit 関数の作成法を示します。プログラムの終了処理は、ユーザシステムによって異なりますので、以下のプログラム例を参考に、ユーザシステムの仕様に従った終了処理を作成してください。

exit 関数は、引数として渡されたプログラムの終了コードに従ってプログラムの終了処理を行い、プログラム起動時の環境に戻ります。ここでは、終了コードを外部変数に設定して、main 関数を呼び出す直前に setjmp 関数で退避した環境に戻ることによって実現します。プログラム実行前の環境に戻るためには、次の関数「callmain」を作成し、初期設定関数「PowerON_Reset」から関数「main」を呼び出す代わりに、関数「callmain」を呼び出してください。

以下にプログラム例を示します。

```
#include <setjmp.h>
#include <stddef.h>

extern long _atexit_count ;
extern void (*_atexit_buf[32])(void) ;
#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
int main(void);
extern jmp_buf _init_env ;
int _exit_code ;

#ifdef __cplusplus
extern "C"
#endif
void exit(int code)
{
    int i;
    _exit_code=code ;                    // _exit_code にリターンコードを設定します
    for(i=_atexit_count-1; i>=0; i--)    // atexit 関数で登録した関数を順次実行します
        (*_atexit_buf[i])();
    _CLOSEALL();                         // オープンした関数を全てクローズします
    longjmp(_init_env, 1) ;              // setjmp で退避した環境にリターンします
}
#ifdef __cplusplus
extern "C"
#endif
void callmain(void)
{
    // setjmp を用いて現在の環境を退避し、main 関数を呼び出します
    if(!setjmp(_init_env))
        _exit_code=main();               // exit 関数からのリターン時には処理を終了します
}
```

(3) 異常終了(abort)ルーチンの作成例

異常終了の場合は、ご使用になっているユーザシステムの仕様に従って、プログラムを異常終了させる処理を行ってください。

C++プログラムを使用する場合、以下のときにも abort 関数を呼び出します。

- 例外処理が正しく動作しなかった場合
- 純粋仮想関数自体をコールした場合
- dynamic_castに失敗した場合
- typeidに失敗した場合
- クラス配列のdelete時に情報が取れなかった場合
- クラスオブジェクトのデストラクタコール情報登録時に矛盾が発生した場合

以下、標準出力装置にメッセージを出力したあと、ファイルをクローズしてから無限ループしてリセットを待つプログラム例を示します。

例:

```
#include <stdio.h>

#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
#ifdef __cplusplus
extern "C"
#endif
void abort(void)
{
       printf("program is abort !!¥n");   // メッセージを出力します
       _CLOSEALL();                       // ファイルをクローズします
       while(1)   ;                       // 無限ループします
}
```

8.3.6 統合開発環境で生成されるスタートアッププログラム

統合開発環境で生成される実際のスタートアッププログラムの例として、CPU 種別として RX610 を選択したシミュレータ用の場合の例を示します。

(1) ソースファイル

スタートアッププログラムは、表 8.7 に示すファイルから構成されます。

表 8.7 統合開発環境で生成されるプログラムの一覧

	ファイル名	内容
(a)	resetprg.c	**初期設定ルーチン (リセットベクタ関数)**
(b)	intprg.c	**ベクタ関数の定義**
(c)	vecttbl.c	**固定ベクタテーブル**
(d)	dbsct.c	**セクションの初期化処理(テーブル)**
(e)	lowsrc.c	**低水準インタフェースルーチン(C 言語部分)**
(f)	lowlvl.src	**低水準インタフェースルーチン(アセンブリ言語部分)**
(g)	sbrk.c	**低水準インタフェースルーチン(sbrk 関数)**
(h)	typedefine.h	**型定義ヘッダ**
(i)	vect.h	**ベクタ関数のヘッダ**
(j)	stacksct.h	**スタックサイズの設定**
(k)	lowsrc.h	**低水準インタフェースルーチン(C 言語ヘッダ)**
(l)	sbrk.h	**低水準インタフェースルーチン(sbrk 関数のヘッダ)**

ファイル内容を、以下、(a) ～ (l) に示します。

(a) resetprg.c -- 初期設定ルーチン (リセットベクタ関数)

```
#include  <machine.h>
#include  <_h_c_lib.h>
//#include   <stddef.h>                        // Remove the comment when you use errno
//#include   <stdlib.h>                        // Remove the comment when you use rand()
#include  "typedefine.h"          // Define Types
#include  "stacksct.h"      // Stack Sizes (Interrupt and User)

#ifdef __cplusplus               // For Use Reset vector
extern "C" {
#endif
void PowerON_Reset(void);
void main(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus               // For Use SIM I/O
extern "C" {
#endif
extern void _INIT_IOLIB(void);
extern void _CLOSEALL(void);
#ifdef __cplusplus
}
```

```
#endif

#define PSW_init  0x00010000 // PSW bit pattern
#define FPSW_init 0x00000000// FPSW bit base pattern

//extern void srand(_UINT);      // Remove the comment when you use rand()
//extern _SBYTE *_s1ptr;         // Remove the comment when you use strtok()

//#ifdef __cplusplus                  // Use Hardware Setup
//extern "C" {
//#endif
//extern void HardwareSetup(void);
//#ifdef __cplusplus
//}
//#endif

//#ifdef __cplusplus             // Remove the comment when you use global class object
//extern "C" {                        // Sections C$INIT and C$END will be generated
//#endif
//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//#endif

#pragma section ResetPRG         // output PowerON_Reset to PResetPRG section

#pragma entry PowerON_Reset

void PowerON_Reset(void)
{
     set_intb(__sectop("C$VECT"));

#ifdef __ROZ                     // Initialize FPSW
#define _ROUND 0x00000001             // Let FPSW RMbits=01 (round to zero)
#else
#define _ROUND 0x00000000             // Let FPSW RMbits=00 (round to nearest)
#endif
#ifdef __DOFF
#define _DENOM 0x00000100             // Let FPSW DNbit=1 (denormal as zero)
#else
#define _DENOM 0x00000000             // Let FPSW DNbit=0 (denormal as is)
#endif
     set_fpsw(FPSW_init | _ROUND | _DENOM);

     _INITSCT();                      // Initialize Sections

     _INIT_IOLIB();                   // Use SIM I/O

//   errno=0;                         // Remove the comment when you use errno
//   srand((_UINT)1);                      // Remove the comment when you use rand()
//   _s1ptr=NULL;                     // Remove the comment when you use strtok()

//   HardwareSetup();                 // Use Hardware Setup
   nop();

//   _CALL_INIT();                    // Remove the comment when you use global class object

     set_psw(PSW_init);               // Set Ubit & Ibit for PSW
//   chg_pmusr();                     // Remove the comment when you need to change PSW PMbit
(SuperVisor->User)

     main();

     _CLOSEALL();                     // Use SIM I/O

//   _CALL_END();                     // Remove the comment when you use global class object

     brk();
}
```

(b) intprg.c -- **ベクタ関数の定義**

```
#include <machine.h>
#include "vect.h"
#pragma section IntPRG

// Exception(Supervisor Instruction)
void Excep_SuperVisorInst(void){/* brk(); */}

// Exception(Undefined Instruction)
void Excep_UndefinedInst(void){/* brk(); */}

// Exception(Floating Point)
void Excep_FloatingPoint(void){/* brk(); */}

// NMI
void NonMaskableInterrupt(void){/* brk(); */}

// Dummy
void Dummy(void){/* brk(); */}

// BRK
void Excep_BRK(void){ wait(); }
```

(c) vecttbl.c -- **固定ベクタテーブル**

```
#include "vect.h"

#pragma section C FIXEDVECT

void (*const Fixed_Vectors[])(void) = {
//;0xffffffd0 Exception(Supervisor Instruction)
   Excep_SuperVisorInst,
//;0xffffffd4 Reserved
   Dummy,
//;0xffffffd8 Reserved
   Dummy,
//;0xffffffdc Exception(Undefined Instruction)
   Excep_UndefinedInst,
//;0xffffffe0 Reserved
   Dummy,
//;0xffffffe4 Exception(Floating Point)
   Excep_FloatingPoint,
//;0xffffffe8 Reserved
   Dummy,
//;0xffffffec Reserved
   Dummy,
//;0xfffffff0 Reserved
   Dummy,
//;0xfffffff4 Reserved
   Dummy,
//;0xfffffff8 NMI
   NonMaskableInterrupt,
//;0xfffffffc RESET
//;<<VECTOR DATA START (POWER ON RESET)>>
//;Power On Reset PC
PowerON_Reset
//;<<VECTOR DATA END (POWER ON RESET)>>
};
```

(d) dbsct.c -- **セクションの初期化処理(テーブル)**

```
#include "typedefine.h"

#pragma unpack

#pragma section C C$DSEC
extern const struct {
   _UBYTE *rom_s;     /* Start address of the initialized data section in ROM */
   _UBYTE *rom_e;     /* End address of the initialized data section in ROM  */
   _UBYTE *ram_s;     /* Start address of the initialized data section in RAM */
}  _DTBL[] = {
   { __sectop("D"), __secend("D"), __sectop("R") },
   { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
   { __sectop("D_1"), __secend("D_1"), __sectop("R_1") }
};
#pragma section C C$BSEC
extern const struct {
   _UBYTE *b_s;       /* Start address of non-initialized data section */
   _UBYTE *b_e;       /* End address of non-initialized data section */
}  _BTBL[] = {
   { __sectop("B"), __secend("B") },
   { __sectop("B_2"), __secend("B_2") },
   { __sectop("B_1"), __secend("B_1") }
};

#pragma section

/*
** CTBL prevents excessive output of L1100 messages when linking.
** Even if CTBL is deleted, the operation of the program does not change.
*/
_UBYTE * const _CTBL[] = {
   __sectop("C_1"), __sectop("C_2"), __sectop("C"),
   __sectop("W_1"), __sectop("W_2"), __sectop("W")
};

#pragma packoption
```

(e) lowsrc.c -- **低水準インタフェースルーチン(C言語部分)**

```
#include <string.h>
#include <stdio.h>
#include <stddef.h>
#include "lowsrc.h"

#define STDIN  0
#define STDOUT 1
#define STDERR 2

#define FLMIN  0
#define _MOPENR    0x1
#define _MOPENW    0x2
#define _MOPENA    0x4
#define _MTRUNC    0x8
#define _MCREAT    0x10
#define _MBIN0x20
#define _MEXCL     0x40
#define _MALBUF    0x40
#define _MALFIL    0x80
#define _MEOF0x100
#define _MERR0x200
#define _MLBF0x400
#define _MNBF0x800
#define _MREAD     0x1000
#define _MWRITE    0x2000
#define _MBYTE     0x4000
```

```
#define _MWIDE    0x8000

#define O_RDONLY 0x0001
#define O_WRONLY 0x0002
#define O_RDWR  0x0004
#define O_CREAT  0x0008
#define O_TRUNC  0x0010
#define O_APPEND 0x0020

#define CR 0x0d
#define LF 0x0a

extern const long _nfiles;
char flmod[IOSTREAM];

unsigned char sml_buf[IOSTREAM];

#define FPATH_STDIN    "C:¥¥stdin"
#define FPATH_STDOUT   "C:¥¥stdout"
#define FPATH_STDERR   "C:¥¥stderr"

extern void charput(unsigned char);
extern unsigned char charget(void);

#include <stdio.h>
FILE *_Files[IOSTREAM];
char *env_list[] = {
   "ENV1=temp01",
   "ENV2=temp02",
   "ENV9=end",
   '¥0'
};

char **environ = env_list;

void _INIT_IOLIB( void )
{
   if( freopen( FPATH_STDIN, "r", stdin ) == NULL )
      stdin->_Mode = 0xffff;
   stdin->_Mode  = _MOPENR;
   stdin->_Mode |= _MNBF;
   stdin->_Bend = stdin->_Buf + 1;

   if( freopen( FPATH_STDOUT, "w", stdout ) == NULL )
      stdout->_Mode = 0xffff;
   stdout->_Mode |= _MNBF;
   stdout->_Bend = stdout->_Buf + 1;

   if( freopen( FPATH_STDERR, "w", stderr ) == NULL )
      stderr->_Mode = 0xffff;
   stderr->_Mode |= _MNBF;
   stderr->_Bend = stderr->_Buf + 1;
}

void _CLOSEALL( void )
{
   long i;
   for( i=0; i < _nfiles; i++ )
   {
      if( _Files[i]->_Mode & (_MOPENR | _MOPENW | _MOPENA ) )
      fclose( _Files[i] );
   }
}

long open(const char *name,
    long mode,
    long flg)
{

   if( strcmp( name, FPATH_STDIN ) == 0 )
   {
```

```
      if( ( mode & O_RDONLY ) == 0 ) return -1;
      flmod[STDIN] = mode;
      return STDIN;
   }
   else if( strcmp( name, FPATH_STDOUT ) == 0 )
   {
      if( ( mode & O_WRONLY ) == 0 ) return -1;
      flmod[STDOUT] = mode;
      return STDOUT;
   }
   else if(strcmp(name, FPATH_STDERR ) == 0 )
   {
      if( ( mode & O_WRONLY ) == 0 ) return -1;
      flmod[STDERR] = mode;
      return STDERR;
   }
   else return -1;
}

long close( long fileno )
{
   return 1;
}

long write(long  fileno,
    const unsigned char *buf,
    long  count)
{
   long   i;
   unsigned char   c;

   if(flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR)
   {
      if( fileno == STDIN ) return -1;
      else if( (fileno == STDOUT) || (fileno == STDERR) )
      {
         for( i = count; i > 0; --i )
         {
            c = *buf++;
            charput(c);
         }
         return count;
      }
      else return -1;
   }
   else return -1;
}

long read( long fileno, unsigned char *buf, long count )
{
      long i;
     if((flmod[fileno]&_MOPENR) || (flmod[fileno]&O_RDWR)){
         for(i = count; i > 0; i--){
              *buf = charget();
              if(*buf==CR){
                  *buf = LF;
              }
              buf++;
         }
         return count;
     }
     else {
         return -1;
     }
}

long lseek( long fileno, long offset, long base )
{
   return -1L;
}
```

(f) lowlvl.src -- **低水準インタフェースルーチン(アセンブリ言語部分)**

```
      .GLB   _charput
      .GLB   _charget

SIM_IO  .EQU 0h

      .SECTION  P,CODE
;------------------------------------------------------------------------
; _charput:
;------------------------------------------------------------------------
_charput:
      MOV.L    #IO_BUF,R2
      MOV.B    R1,[R2]
      MOV.L    #1220000h,R1
      MOV.L    #PARM,R3
      MOV.L    R2,[R3]
      MOV.L    R3,R2
      MOV.L    #SIM_IO,R3
      JSR      R3
      RTS

;------------------------------------------------------------------------
; _charget:
;------------------------------------------------------------------------
_charget:
     MOV.L     #1210000h,R1
     MOV.L     #IO_BUF,R2
     MOV.L     #PARM,R3
     MOV.L     R2,[R3]
     MOV.L     R3,R2
     MOV.L     #SIM_IO,R3
     JSR       R3
     MOV.L     #IO_BUF,R2
     MOVU.B    [R2],R1
     RTS

;------------------------------------------------------------------------
; I/O Buffer
;------------------------------------------------------------------------
      .SECTION  B,DATA,ALIGN=4
PARM:   .BLKL    1
      .SECTION  B_1,DATA
IO_BUF:  .BLKB    1
      .END
```

(g) sbrk.c -- **低水準インタフェースルーチン(sbrk関数)**

```
#include <stddef.h>
#include <stdio.h>
#include "typedefine.h"
#include "sbrk.h"

_SBYTE  *sbrk(size_t size);

//const size_t _sbrk_size=       /* Specifies the minimum unit of   */
                        /* the defined heap area      */

extern _SBYTE *_s1ptr;

union HEAP_TYPE {
   _SDWORD  dummy ;          /* Dummy for 4-byte boundary           */
   _SBYTE heap[HEAPSIZE];    /* Declaration of the area managed by sbrk   */
};
```

```
static union HEAP_TYPE heap_area ;

/* End address allocated by sbrk */
static _SBYTE *brk=(_SBYTE *)&heap_area;

/************************************************************************/
/*    sbrk:Memory area allocation                                       */
/*        Return value:Start address of allocated area (Pass)           */
/*                -1                        (Failure)      */
/************************************************************************/
_SBYTE  *sbrk(size_t size)              /* Assigned area size  */
{
   _SBYTE  *p;

   if(brk+size > heap_area.heap+HEAPSIZE){    /* Empty area size    */
      p = (_SBYTE *)-1;
   }
   else {
      p = brk;                      /* Area assignment    */
      brk += size;                   /* End address update  */
   }
   return p;
}
```

(h) typedefine.h -- **型定義ヘッダ**

```
typedef signed char _SBYTE;
typedef unsigned char _UBYTE;
typedef signed short _SWORD;
typedef unsigned short _UWORD;
typedef signed int _SINT;
typedef unsigned int _UINT;
typedef signed long _SDWORD;
typedef unsigned long _UDWORD;
typedef signed long long _SQWORD;
typedef unsigned long long _UQWORD;
```

(i) vect.h -- **ベクタ関数のヘッダ**

```
// Exception(Supervisor Instruction)
#pragma interrupt (Excep_SuperVisorInst)
void Excep_SuperVisorInst(void);

// Exception(Undefined Instruction)
#pragma interrupt (Excep_UndefinedInst)
void Excep_UndefinedInst(void);

// Exception(Floating Point)
#pragma interrupt (Excep_FloatingPoint)
void Excep_FloatingPoint(void);

// NMI
#pragma interrupt (NonMaskableInterrupt)
void NonMaskableInterrupt(void);

// Dummy
#pragma interrupt (Dummy)
void Dummy(void);

// BRK
#pragma interrupt (Excep_BRK(vect=0))
void Excep_BRK(void);

//;<<VECTOR DATA START (POWER ON RESET)>>
//;Power On Reset PC
```

```
extern void PowerON_Reset(void);
//;<<VECTOR DATA END (POWER ON RESET)>>
```

(j) stacksct.h -- **スタックサイズの設定**

```
// #pragma stacksize su=0x100    // Remove the comment when you use user stack
#pragma stacksize si=0x300
```

(k) lowsrc.h -- **低水準インタフェースルーチン**(C**言語ヘッダ**)

```
/*Number of I/O Stream*/
#define IOSTREAM 20
```

(l) sbrk.h -- **低水準インタフェースルーチン**(sbrk**関数のヘッダ**)

```
/* size of area managed by sbrk */
#define HEAPSIZE 0x400
```

(2) 実行コマンド

これらのファイルをビルドするのに必要なコマンド列の例を示します。

この例では、ユーザプログラム(main 関数を含む)は UserProgram.c、生成するロードモジュールやライブラリなどのファイル名を LoadModule(拡張子を除いた部分)とします。

```
lbgrx -cpu=rx600 -output=LoadModule.lib
ccrx -cpu=rx600 -output=obj UserProgram.c
ccrx -cpu=rx600 -output=obj resetprg.c
ccrx -cpu=rx600 -output=obj intprg.c
ccrx -cpu=rx600 -output=obj vecttbl.c
ccrx -cpu=rx600 -output=obj dbsct.c
ccrx -cpu=rx600 -output=obj lowsrc.c
asrx -cpu=rx600 lowlvl.src
ccrx -cpu=rx600 -output=obj sbrk.c
optlnk -rom=D=R,D_1=R_1,D_2=R_2 -list=LoadModule.map
-start=B_1,R_1,B_2,R_2,B,R,SI/01000,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D_1,D_2,D,P,PIntPRG,
W*,L/0FFFF8100,FIXEDVECT/0FFFFFFD0 -library=LoadModule.lib -output=LoadModule.abs
UserProgram.obj resetprg.obj intprg.obj vecttbl.obj dbsct.obj lowsrc.obj lowlvl.obj sbrk.obj
optlnk -output=LoadModule.sty -form=stype -output=LoadModule.mot LoadModule.abs
```

8.4 PIC/PID機能の利用

本章では、PIC/PID 機能の概要と、PIC/PID 機能を利用する場合のスタートアップの作成方法について説明します。

PIC/PID 機能は、一度リンクが完了して配置アドレスが確定した ROM 上のコードやデータを、リンクをやり直すことなく、任意のアドレスに配置して利用できるようにする機能です。

PIC は位置独立コード(Position Independent Code)、PID は位置独立データ(Position Independent Data)をそれぞれ意味します。PIC を生成する機能が PIC 機能、PID を生成する機能が PID 機能で、ここでは、これらの機能を総称して PIC/PID 機能と呼びます。

8.4.1 用語の定義

(1) マスタとアプリケーション

PIC/PID 機能では、ROM 上のコードやデータを PIC や PID にしたプログラムをアプリケーション、アプリケーションを実行させるのに必要なプログラムをマスタと呼びます。

マスタは、アプリケーションの起動処理のほか、アプリケーションから呼び出される共有ライブラリ、およびアプリケーションの RAM 領域を持ちます。PIC および PID はアプリケーションにのみ含まれ、マスタには含まれていません。

(2) 共有ライブラリ

マスタ内にあり、複数のアプリケーションから呼び出すことのできる関数群です。

(3) ジャンプテーブル

アプリケーションから共有ライブラリ関数への呼び出しを中継するプログラムです。

8.4.2 各オプションの機能

PIC/PID 機能と関連するオプションを説明します。

各オプションの機能詳細については、2 章(コンパイラ)、4 章(アセンブラ)、5 章(最適化リンケージエディタ)の各オプションの説明を参照ください。

(1) アプリケーションのコード生成(pic,pidオプション)

pic オプションを有効にしてコンパイルすると、PIC 機能が有効になり、コード領域(P セクション)が PIC になります。PIC は分岐先アドレスや関数アドレスの取得を全て PC 相対で行うため、リンク後も任意のアドレスに配置することができます。

pid オプションを有効にしてコンパイルすると、PID 機能が有効になり、ROM データ領域(C, C_2, C_1, W, W_2, W_1 および L セクション)が PID になります。プログラムは PID に対しその先頭アドレスを示すレジスタ(PID レジスタ)から相対のアクセスでアクセスします。ユーザはマスタで PID レジスタの設定値を変化させて、リンク後も PID を任意のアドレスに移動することができます。

なお、PIC 機能(pic オプション)と PID 機能(pid オプション)は、それぞれ独立した機能として動作できるように設計しておりますが、同時に有効にしたうえで、PIC と PID を隣接させてご利用いただくことを推奨します。PIC 機能と PID 機能を個別に利用したり、PIC と PID の相対距離を変更したアプリケーションのデバッグは、デバッガのバージョンによりサポートされない場合があります。本書でも PIC 機能と PID 機能を同時に有効にした例で説明しています。

(2) 共有ライブラリ対応(jump_entries_for_pic, nouse_pid_registerオプション)

アプリケーションからマスタにあるライブラリを呼び出すための機能です。

nouse_pid_register オプションは、マスタのコンパイル時に使用し、PID レジスタを使用しないコードを生成します。

jump_entries_for_pic オプションを、マスタのリンク時に最適化リンケージエディタに指定すると、アプリケーションから固定アドレスにあるライブラリ関数を呼び出すためのジャンプテーブルを生成します。

(3) RAM領域の共有(Fsymbolオプション)

マスタ上の変数を、リンク単位の違うアプリケーションからでも読み書きできるようにするための機能です。

マスタのリンク時に、Fsymbol オプションを最適化リンケージエディタに指定すると、アプリケーションから固定アドレスで変数を参照するためのシンボルテーブルを生成します。

8.4.3 アプリケーションに関する制限事項

(1) RAM領域

RAM 領域には PID 機能を適用できません。

(2) アプリケーションの同時実行

PIC/PID 機能を使うと、同じアプリケーションのコピーを複数個 ROM に置き、それぞれを実行できますが、RAM 領域が重なっているため、同じアプリケーションのコピーを同時に複数個実行することはできません。

(3) スタートアップ

アプリケーションに使用するスタートアップとしては、標準のスタートアップ（統合開発環境が生成。詳しくは「8.3 スタートアッププログラムの作成」を参照）はそのままでは使用できません。「8.4.7 アプリケーションのスタートアップ」を参考に、スタートアップを作成してください。

8.4.4 PIC/PID機能で必要なシステム依存処理

次の処理は、システム仕様に合わせてお客様にてご用意いただく必要があります。

(1) マスタの初期化

PIC/PID 機能を使用しない通常のプログラムと同様の処理を行います。

(2) マスタからアプリケーションを起動

アプリケーションの PID の先頭アドレスを PID レジスタに設定し、PIC の起動アドレスへ分岐することで、アプリケーションを起動します。

(3) アプリケーションの初期化

セクションの初期化を行い、アプリケーションの main 関数を実行します。

(4) アプリケーションの終了

main 関数が終了したら、マスタに処理を返します。

8.4.5 コード生成オプションの組み合わせ

マスタとアプリケーションをビルドするときは、構成するオブジェクト間で PIC/PID 機能に関係するオプション指定を合わせておく必要があります。

以下に、オブジェクトごとのコンパイル時のオプションの指定規則と、組み合わせて利用できるオブジェクトのオプション指定の制限について示します。

(1) マスタ

マスタをビルドするときは、PIC/PID 機能オプションを表 8.8 のように指定してください。

表 8.8 マスタ内の PIC/PID 機能オプションの指定規則

	オプション名	コンパイル時	リンク可能オブジェクトのオプション指定の条件
1	pic	× 指定不可	pic の指定なし
2	pid	× 指定不可	pid の指定なし
3	nouse_pid_register	○ 指定可	nouse_pid_register の指定が必須
4	fint_register	○ 指定可	同一パラメータの fint_register の指定が必須
5	base	○ 指定可	同一パラメータの base の指定が必須

(2) アプリケーション

アプリケーションをビルドするときは、PIC/PID 機能オプションを表 8.9 のように指定してください。

表 8.9 アプリケーション内の PIC/PID 機能オプションの指定規則

	オプション名	コンパイル時	リンク可能オブジェクトのオプション指定の条件
1	pic	○ 指定可	pic の指定が必須
2	pid	○ 指定可	pid の指定が必須
3	nouse_pid_register	× 指定不可	nouse_pid_register の指定なし
4	fint_register	○ 指定可	同一パラメータの fint_register の指定が必須
5	base	○ 指定可	同一パラメータの base*1 の指定が必須

【注】 *1 pid 指定時は base=rom=<レジスタ>の指定はできません。

(3) マスタとアプリケーション間

マスタとアプリケーションはそれぞれ PIC/PID 機能オプションを表 8.10 のように指定する必要があります。

表 8.10 マスタとアプリケーション間の PIC/PID 機能オプションの組み合わせ規則

	アプリケーションのオプション	マスタのオプション
1	pic	制限なし
2	pid	nouse_pid_register が必須
3	fint_register	同一パラメータの fint_register が必須
4	base	同一パラメータの base*1 が必須

【注】 *1 pid 指定時は base=rom=<レジスタ>の指定はできません。

8.4.6 マスタのスタートアップ

次の 2 点を除き、必要な処理は PIC/PID 機能を用いない通常のプログラムと同じです。「8.3 スタートアッププログラムの作成」に従ったスタートアップに、次の 2 つの内容を追加してください。

(1) アプリケーションの起動と復帰

main関数で、PIDレジスタを設定し、PICのエントリアドレスに分岐してアプリケーションを起動します。また、アプリケーションからマスタに戻れる手段を用意しておく必要があります。

(2) 使用する共有ライブラリ関数の参照

アプリケーションが利用する共有ライブラリは、あらかじめマスタでも参照しておく必要があります。

以下に、main 関数から PIC/PID アプリケーションを呼び出す例を示します。
なお、この例は次の条件に基づきます。

- アプリケーションの終了時、RTS命令でマスタに復帰できるものとします。
- アプリケーションは戻り値を持たないものとします。
- アプリケーションに対するPICの起動アドレス(PIC_entry)、およびPIDの先頭アドレス(PID_address)は、マスタをビルドする時点で既知および固定であるとします。
- PIDレジスタはR13であるものとします。
- アプリケーション側のセクション領域の初期化は、マスタ側では行わないこととします。
- 共有ライブラリとして、アプリケーションはprintf関数のみ使用するものとします。

例

```
/* マスタ側プログラム */
/* PIC/PID アプリケーションを起動する */
/* (アプリケーションが、戻り値を返さず、RTS で復帰するシステム仕様の場合) */
#include <stdio.h>
#pragma inline_asm Launch_PICPID_Application
void Launch_PICPID_Application(void *pic_entry, void *pid_address)
{
    MOV.L    R2,R13
    JSR      R1
}
int main()
{
    void *PIC_entry   = (void*)0x500000;  /* PIC の起動アドレス */
    void *PID_address = (void*)0x120000;  /* PID の先頭アドレス */

    /* (1)アプリケーションの起動と復帰 */
    Launch_PICPID_Application(PIC_entry, PID_address);

    return 0;
}

/* (2)アプリケーションで使用する共有ライブラリの参照 */
void *_dummy_ptr = (void*)printf;  /* printf 関数 */
```

8.4.7 アプリケーションのスタートアップ

アプリケーションでは、次の項目を設定してください。

【オプション】と書かれている項目は、不要な場合があります。

(1) エントリポイント(PICの起動アドレス)の用意

アプリケーションの起動アドレスです。

(2) スタックポインタの初期化【オプション】

マスタとスタックを共有する場合は、不要です。

必要な場合は、8.3.2(2)を参考に、設定を追加してください。

(3) ベースレジスタに使用する汎用レジスタの初期化【オプション】

ベースレジスタを使用しない場合は、不要です。

必要な場合は、8.3.2(3)を参考に、設定を追加してください。

(4) セクションの初期化処理【オプション】

マスタ側で初期化する場合は、不要です。

必要な場合は、後述の例を参考に、設定を追加してください。

なお、8.3.2(5)の方法はそのままでは使用できません。

(5) ライブラリの初期化処理【オプション】

標準ライブラリを使用しない場合は、不要です。

必要な場合は、8.3.2(6)を参考に、設定を追加してください。

(6) main関数向けPSW初期化【オプション】

必要に応じて、割り込みマスクやユーザモードへの移行を行います。

8.3.2(8)と(9)を参考に、設定を追加してください。

(7) ユーザプログラムの実行

main関数を実行します。

8.3.2(10)を参考に、設定してください。

以下、アプリケーション側のスタートアップ例を示します。

3 つのファイルに分かれています。

- startup_picpid.c … スタートアップ本体。
- initsct_pid.src … セクション初期化のPID版である_INITSCT_PIDです。

  8.3.2(5)で述べている_INITSCT関数をPID対応にしたものです。

  なお、PIDレジスタをR13に固定化しているので、PIDレジスタがR13ではない場合は、R13をPIDレジスタに書き換える必要があります。
- initiolib.c … 標準ライブラリの初期化を行う、_INITLIBを収録しています。

  8.3.2(6)をアプリケーション用に変更したものです。

[startup_picpid.c]

```
// マニュアル 8.3.2(5)セクションの初期化処理
#pragma section C C$DSEC //セクション名を C$DSEC にします
const struct {
    void *rom_s; //初期化データセクションの ROM 上の先頭アドレスメンバ
    void *rom_e; //初期化データセクションの ROM 上の最終アドレスメンバ
    void *ram_s; //初期化データセクションの RAM 上の先頭アドレスメンバ
} DTBL[] = {__sectop("D"), __secend("D"), __sectop("R")};
#pragma section C C$BSEC //セクション名を C$BSEC にします
const struct {
    void *b_s; //未初期化データセクションの先頭アドレスメンバ
    void *b_e; //未初期化データセクションの最終アドレスメンバ
} BTBL[] = {__sectop("B"), __secend("B")};

extern void main(void);
extern void _INITLIB(void); // マニュアル 8.3.2(6) ライブラリの初期化処理
#pragma entry application_pic_entry
void application_pic_entry(void)
{
    _INITSCT_PICPID();
    _INITLIB();
     main();
}
```

[initsct_pid.src]

```
; PID 対応 セクション初期化ルーチン
; ** 注意 ** PID レジスタのチェックが必要です
; このコードは PID レジスタが R13 であると想定しています.もし、PID レジスタが
; R13 でなければ、以下の R13 の記述を PID レジスタに変更してください。
    .glb   __INITSCT_PICPID
    .glb   __PID_TOP
    .section  C$BSEC,ROMDATA,ALIGN=4
    .section  C$DSEC,ROMDATA,ALIGN=4
    .section  P,CODE

__INITSCT_PICPID:             ; function: _INITSCT
    .STACK    __INITSCT_PICPID=28
    PUSHM     R1-R6
    ADD       #-__PID_TOP,R13,R6   ;   How long distance PID moves
;;;
```

```
;;; clear BBS(B)
;;;
    ADD      #TOPOF C$BSEC,  R6, R4
    ADD      #SIZEOF C$BSEC, R4, R5
    MOV.L    #0, R2
    BRA      next_loop1

loop1:
    MOV.L    [R4+], R1
    MOV.L    [R4+], R3
    CMP      R1, R3
    BLEU     next_loop1
    SUB      R1, R3
    SSTR.B
next_loop1:
    CMP      R4,R5
    BGTU     loop1

;;;
;;; copy DATA from ROM(D) to RAM(R)
;;;
    ADD      #TOPOF C$DSEC,  R6, R4
    ADD      #SIZEOF C$DSEC, R4, R5
    BRA      next_loop3

loop3:
    MOV.L    [R4+], R2
    MOV.L    [R4+], R3
    MOV.L    [R4+], R1
    CMP      R2, R3
    BLEU     next_loop3
    SUB      R2, R3
    ADD      R6, R2      ; Adjust for real address of PID
    SMOVF
next_loop3:
    CMP      R4, R5
    BGTU     loop3
    POPM     R1-R6
    RTS

    .end
```

[initiolib.c]

```
#include <stdio.h>
#include <stdlib.h>
#define IOSTREAM 3
const size_t _sbrk_size = 520; // ヒープ領域確保サイズの最小単位を指定します
// (省略時：1024)

void _INIT_LOWLEVEL(void);
void _INIT_OTHERLIB(void);

void _INITLIB (void)
{
    _INIT_LOWLEVEL(); // 低水準インタフェースルーチンの初期設定をします
    _INIT_IOLIB(); // 入出力ライブラリの初期設定をします
```

```
    _INIT_OTHERLIB(); // rand 関数、strtok 関数の初期設定をします
}
void _INIT_LOWLEVEL(void)
{ // 低水準ライブラリで必要な初期設定をしてください
}
void _INIT_OTHERLIB(void)
{
    srand(1); // rand 関数を使用する場合の初期設定です
}
```

9. C/C++言語仕様

9.1 言語仕様

9.1.1 コンパイラの仕様

言語仕様で規定していない処理系定義項目について、コンパイラの仕様を示します。

なお、本コンパイラが準拠する言語仕様については、「9.1.5 準拠する言語仕様」を参照下さい。

(1) 環境

表 9.1 環境の仕様

	項 目	コンパイラの仕様
1	main 関数への実引数の意味	規定しません。
2	対話的入出力装置の構成	規定しません。

(2) 識別子

表 9.2 識別子の仕様

	項 目	コンパイラの仕様
1	外部結合とならない識別子(内部名)の有効文字数	8189 文字まで有効です。
2	外部結合となる識別子(外部名)の有効文字数	8191 文字まで有効です。
3	外部結合となる識別子(外部名)の大文字と小文字の区別	大文字と小文字を区別します。

(3) 文字

表 9.3 文字の仕様

	項 目	コンパイラの仕様
1	ソース文字集合および実行環境文字集合の要素	どちらも ASCII 文字集合です。ただし、文字列、文字定数にはシフト JIS、EUC 漢字コード、Latin1 コードまたは UTF-8 コードを記述できます。
2	多バイト文字のコード化で使用されるシフト状態	シフト状態はサポートしていません。
3	プログラム実行時の文字集合の文字のビット数	ビット数は 8 ビットです。
4	文字定数内、文字列内のソース文字集合の文字と実行環境文字集合の文字との対応付け	同じ ASCII 文字に対応します。
5	言語で規定していない文字や拡張表記を含む整数文字定数の値	言語で規定する以外の文字、拡張表記はサポートしていません。
6	2 文字以上の文字を含む文字定数または 2 文字以上の多バイト文字を含む広角文字定数の値	文字定数は上位 2 バイトを有効とします。 広角文字定数はサポートしていません。 また、1 文字より多く指定した場合はウォーニングエラーを出力します。
7	多バイト文字を広角文字に変換するために使用される locale の仕様	locale はサポートしていません。
8	char 型の値	unsigned char 型と同じ値の範囲を持ちます。*1

【注】 *1 signed_char オプションを指定した場合、signed char 型と同じ値の範囲を持ちます。

(4) 整数

表 9.4 整数の仕様

	項 目	コンパイラの仕様
1	整数型の表現方法とその値	表 9.5 に示します。
2	整数の値がより短いサイズの符号付き整数型、または符号なし整数型を同一のサイズの符号付き整数型に変換したときの値(結果の値が変換先の型で表現できない場合)	整数の値の下位 4 バイト(変換後の型のサイズが 4 バイトの場合)、下位 2 バイト(変換後の型のサイズが 2 バイトの場合)あるいは下位 1 バイト(変換後の型のサイズが 1 バイトの場合)が変換後の値となります。
3	符号付き整数に対するビットごとの演算の結果	符号付きの値になります。
4	整数除算における剰余の符号	被除数の符号と同符号になります。
5	負の値を持つ符号付きスカラ型の右シフトの結果	符号ビットを保持します。

表 9.5 整数型とその値の範囲

	型	値の範囲	データサイズ
1	char*1	0～255	1 バイト
2	signed char	-128～127	1 バイト
3	unsigned char	0～255	1 バイト
4	short signed short	-32768～32767	2 バイト
5	unsigned short	0～65535	2 バイト
6	int*2 signed int*2	-2147483648～2147483647	4 バイト
7	unsigned int*2	0～4294967295	4 バイト
8	long signed long	-2147483648～2147483647	4 バイト
9	unsigned long	0～4294967295	4 バイト
10	long long signed long long	-9223372036854775808～9223372036854775807	8 バイト
11	unsigned long long	0～18446744073709551615	8 バイト

【注】 *1 signed_char オプションを指定した場合、signed char 型として扱います。

*2 int_to_short オプションを指定した場合、int 型は short 型、signed int 型は signed short 型、unsigned int 型は unsigned short 型としてそれぞれ扱います。

(5) 浮動小数点

表 9.6 浮動小数点の仕様

	項 目	コンパイラの仕様
1	浮動小数点型の表現方法とその値	浮動小数点型には、float型、double型とlong double型があります。浮動小数点型の内部表現や変換仕様、演算仕様等の性質は「9.1.3 浮動小数点型の仕様」で説明します。表 9.7 に、浮動小数点型の表現可能な値の限界値を示します。
2	整数を本来の値に正確に表現することができない浮動小数点型に変換したときの切り捨て方向	
3	浮動小数点型をより狭い浮動小数点型に変換したときの切り捨てまたは丸め方法	

表 9.7　浮動小数点型の限界値

	項　目		限界値	
			10 進数表現*1	内部表現（16 進数）
1	float 型の最大値		3.4028235677973364e+38f (3.4028234663852886e+38f)	7f7fffff
2	float 型の正の最小値		7.0064923216240862e-46f (1.4012984643248171e-45f)	00000001
3	double*2 long double*2	型の最大値	1.7976931348623158e+308 (1.7976931348623157e+308)	7fefffffffffffff
4	double*2 long double*2	型の正の最小値	4.9406564584124655e-324 (4.9406564584124654e-324)	0000000000000001

【注】　*1　10 進数表現の限界値は 0 または無限大にならない限界値です。また、( )内は理論値を示します。

*2　dbl_size=8 を指定した場合の解釈です。dbl_size=4 を指定した場合、double 型および long double 型は float 型と同じ値となります。

(6) 配列とポインタ

表 9.8　配列とポインタの仕様

	項　目	コンパイラの仕様
1	配列の大きさの最大値を保持するために必要な整数の型(size_t)	unsigned long 型
2	ポインタ型から整数型への変換 (ポインタ型のサイズ≧整数型のサイズ)	ポインタ型の下位バイトの値になります。
3	ポインタ型から整数型への変換 (ポインタ型のサイズ＜整数型のサイズ)	ゼロ拡張します。
4	整数型からポインタ型への変換 (整数型のサイズ≧ポインタ型のサイズ)	整数型の下位バイトの値となります。
5	整数型からポインタ型への変換 (整数型のサイズ＜ポインタ型のサイズ)	符号拡張します。
6	同じ配列内のメンバのポインタ間の差を保持するために必要な整数の型(ptrdiff_t)	int 型

(7) レジスタ

表 9.9 レジスタの仕様

	項 目	コンパイラの仕様
1	レジスタに割り付けることができる変数の型	char, signed char, unsigned char, bool, _Bool, short, unsigned short, int, unsigned int, long, unsigned long, long long, unsigned long long, float, ポインタ

(8) クラス、構造体、共用体、列挙型、ビットフィールド

表 9.10 クラス、構造体、共用体、列挙型、ビットフィールドの仕様

	項 目	コンパイラの仕様
1	異なる型のメンバでアクセスされる共用体型のメンバ参照	参照はできますが、値は保証しません。
2	クラス・構造体メンバのアライメント	クラス・構造体メンバ中のアライメント数の最大値がそのクラス・構造体のアライメント数になります。割り付け方の詳細な仕様は「9.1.2(2) 構造体/共用体、クラス型」を参照してください。
3	単なる int 型のビットフィールドの符号	unsigned int 型 *3
4	int 型のサイズ内のビットフィールドの割り付け順序	下位ビットから割り付けます。*1*2
5	int 型のサイズ内にビットフィールドが割り付けられているとき、次に割り付けるビットフィールドのサイズが int 型内の残っているサイズを超えたときの割り付け方	次の int 型の領域に割り付けます。*1
6	ビットフィールドで許される型指定子	char、unsigned char、bool、_Bool、short、unsigned short、int、unsigned int、long、unsigned long、enum、long long、unsigned long long
7	列挙型の値を表現する整数型	int 型です。*4

【注】 *1 ビットフィールドの割り付け方の詳細については、「9.1.2(3) ビットフィールド」を参照してください。

*2 bit_order=left オプションを指定した場合、上位ビットから割り付けられます。

*3 signed_bitfield オプションを指定した場合、signed int 型となります。

*4 auto_enum オプションを指定した場合、列挙値が収まる最小の型となります。詳細は、「2.5 マイコンオプション」の、auto_enum オプションの説明を参照ください。

(9) 型修飾子

表 9.11 型修飾子の仕様

	項 目	コンパイラの仕様
1	volatile 修飾したデータへのアクセスの種類	規定しません。

(10) 宣言

表 9.12 宣言の仕様

	項 目	コンパイラの仕様
1	基本型(算術型、構造体型、共用体型)を修飾する宣言子の数	16 個まで指定できます。

基本型を修飾する型の数の数え方を、以下に例を用いて示します。

例：

(i) `int a;` a は int 型(基本型)であり、基本型を修飾する型の数は 0 個です。

(ii) `char *f();` f は char 型(基本型)へのポインタ型を返す関数型であり、基本型を修飾する型の数は 2 個です。

(11) 文

表 9.13 文の仕様

	項 目	コンパイラの仕様
1	一つの switch 文中で指定できる case ラベルの数	2147483646 個まで指定できます。

(12) プリプロセッサ

表 9.14 プリプロセッサの仕様

	項 目	コンパイラの仕様
1	条件コンパイルの定数式内の単一文字の文字定数と実行環境文字集合の対応	プリプロセッサ文の文字定数と実行環境文字集合は一致します。
2	インクルードファイルの読み込み方法	「<」、「>」で囲まれたファイルは include オプションで指定されたパスから読み込みます。 ファイルが見つからない場合、環境変数 INC_RX 指定フォルダ、環境変数 BIN_RX 指定フォルダの順序で各フォルダを検索します。
3	二重引用符で囲まれたインクルードファイルのサポートの有無	サポートします。インクルードファイルをカレントフォルダから読み込みます。カレントフォルダになければ、本表 2 項の読み込み方法に従います。
4	ソースファイルの文字の並びの対応(マクロ展開後の文字列の空白文字)	空白文字列は、空白文字 1 文字として展開します。
5	#pragma の動作	「9.2.1 #pragma」を参照してください。
6	__DATE__、__TIME__の値	コンパイル開始時のホストマシンのタイマに基づく値が設定されます。

9.1.2 データの内部表現

本節では、型名と、データの内部表現の対応について述べます。データの内部表現は、以下の項目から成り立っています。

- データのサイズ

  データの占有する領域のサイズです。

- データのアライメント数

  データを割り付けるアドレスに関する制約です。任意のアドレスに割り付ける1バイトアライメント、偶数バイトに割り付ける2バイトアライメント、4の倍数バイトに割り付ける4バイトアライメントがあります。

- 値の範囲

  スカラ型(C言語)、基本型(C++言語)の値のとり得る範囲を示します。

- データの割り付け例

  構造体/共用体(C言語)、クラス型(C++言語)の要素となるデータの割り付け方を示します。

(1) スカラ型(C言語)、基本型(C++言語)

C言語におけるスカラ型および、C++言語における基本型の内部表現を**表**9.15に示します。

表 9.15 スカラ型・基本型の内部表現

	型名	サイズ (byte)	アライメント数 (byte)	符号の有無	値の範囲	
					最小値	最大値
1	char*1	1	1	無	0	2^8-1 (255)
2	signed char	1	1	有	-2^7 (-128)	2^7-1 (127)
3	unsigned char	1	1	無	0	2^8-1 (255)
4	short	2	2	有	-2^{15} (-32768)	2^{15}-1 (32767)
5	signed short	2	2	有	-2^{15} (-32768)	2^{15}-1 (32767)
6	unsigned short	2	2	無	0	2^{16}-1 (65535)
7	int*2	4	4	有	-2^{31}(-2147483648)	2^{31}-1 (2147483647)
8	signed int*2	4	4	有	-2^{31}(-2147483648)	2^{31}-1 (2147483647)
9	unsigned int*2	4	4	無	0	2^{32}-1 (4294967295)
10	long	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
11	signed long	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
12	unsigned long	4	4	無	0	2^{32}-1 (4294967295)
13	long long	8	4	有	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)

	型名	サイズ (byte)	アライメント数 (byte)	符号の有無	値の範囲	
					最小値	最大値
14	signed long long	8	4	有	-2^{63} (-9223372036854775808)	$2^{63}-1$ (9223372036854775807)
15	unsigned long long	8	4	無	0	$2^{64}-1$ (18446744073709551615)
16	float	4	4	有	-∞	+∞
17	double long double	4*4	4	有	-∞	+∞
18	size_t	4	4	無	0	$2^{32}-1$ (4294967295)
19	ptr_diff_t	4	4	有	-2^{31} (-2147483648)	$2^{31}-1$ (2147483647)
20	enum*3	4	4	有	-2^{31} (-2147483648)	$2^{31}-1$ (2147483647)
21	ポインタ	4	4	無	0	$2^{32}-1$ (4294967295)
22	bool*5 _Bool*8	1	1	ー*9	ー	ー
23	リファレンス*6	4	4	無	0	$2^{32}-1$ (4294967295)
24	データメンバへのポインタ*6	4	4	有	0	$2^{32}-1$ (4294967295)
25	関数メンバへのポインタ*6 *7	12	4	ー*9	ー	ー

【注】 *1 signed_char オプションを指定した場合、signed char 型と同じになります。

*2 int_to_short オプションを指定した場合、int 型は short 型と、signed int 型は signed short 型と、unsigned int 型は unsigned short 型とそれぞれ同じになります。

*3 auto_enum オプションを指定した場合、列挙値が収まる最小の型となります。

*4 dbl_size=8 を指定した場合、double 型および long double 型のサイズは 8 バイトになります。

*5 C++プログラム、または stdbool.h をインクルードした C99 プログラムのコンパイル時のみ有効です。

*6 C++プログラムのコンパイル時のみ有効です。

*7 関数メンバ・仮想関数メンバへのポインタは、以下のデータ構造で表現しています。

```
class PMF{
  public:
    long d;           // オブジェクトのオフセット値
    long i;           // 対象メンバ関数が仮想関数のときの仮想関数表中での
                      // インデックス
    union{
      void (*f)();    // 対象メンバ関数が非仮想関数のときの関数のアドレス
      long offset;    // 対象メンバ関数が仮想関数のときの仮想関数表のオブジェクト
```

```
                                    // 中のオフセット
        };
    };
```

*8 C99 コンパイルのみ有効です。_Bool 型は bool 型と同じ型としてコンパイルされます。

*9 符号の設定はありません。

(2) 構造体/共用体(C言語)、クラス型(C++言語)

本項では、C 言語における配列型、構造体型、共用体型および、C++言語におけるクラス型の内部表現について説明します。

表9.16 に構造体/共用体、クラス型の内部表現を示します。

表 9.16 構造体/共用体、クラス型の内部表現

	型名	アライメント数(byte)	サイズ(byte)	データの割り付け例		
1	配列型	配列要素のアライメント数	配列要素の数×要素サイズ	char a[10];	アライメント数 サイズ	1byte 10byte

2	構造体型	構造体メンバのアライメント数のうち最大値	メンバのサイズの和
			「(a) 構造体データの割り付け方」参照
				struct {
				char a,b;
				};
					アライメント数
					サイズ
						1byte
						2byte

3	共用体型	共用体メンバのアライメント数のうち最大値	最大メンバのサイズ
			「(b) 共用体データの割り付け方」参照
				union {
				char a,b;
				};
					アライメント数
					サイズ
						1byte
						1byte

4	クラス型	仮想関数がある場合:
		常に 4
		
		2)上記以外:
		データメンバのアライメント数のうち最大値
			データメンバ 、
			仮想関数表へのポインタ 、
			仮想基底クラスへのポインタの和
			「(c) クラスデータの割り付け方」参照
				class B: public A{
				virtual void f();
				};
				
				class A{
				char a;
				};
					アライメント数
					サイズ
					
					アライメント数
					サイズ
						4byte
						8byte
						
						1byte
						1byte

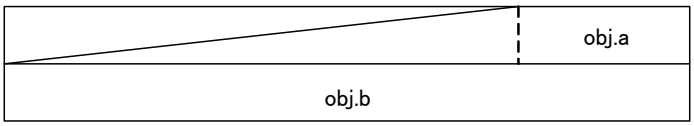
以下の例でサイズを明記していない  は、4 バイトを表します。  はパディングを表します。
また、アドレスの増える向きは、右から左とします（左側が上位アドレス）。

(a) 構造体データの割り付け方

構造体型の各メンバを割り付ける場合、そのメンバの型名の境界調整数に合わせるために直前のメンバとの間にパディングが生じる場合があります。

例：

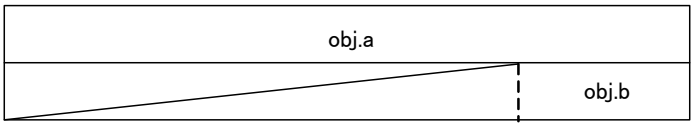
```
struct {
  char a;
  int b;
} obj;
```



構造体が4バイトのアライメント数を持ち、最後のメンバが1,2,3バイト目で終わっている場合、その次のバイトも含めて構造体型の領域として扱います。

例：

```
struct {
  int a;
  char b;
} obj;
```

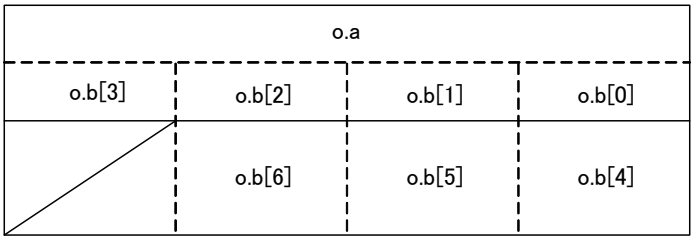


(b) 共用体データの割り付け方

共用体が4バイトのアライメント数を持ち、最大メンバのサイズが4の倍数バイトでない場合、4の倍数になるまで残りのバイトも含めて共用体型の領域として扱います。

例：

```
union {
  int a;
  char b[7];
} o;
```



(c) クラスデータの割り付け方

基底クラス、仮想関数がないクラスの場合、構造体データの割り付け規則に従ってデータメンバを割り付けます。

例：

```
class A{
   char data1;
   int data2;
 public:
   A();
   char getData1(){return data1;}
}obj;
```

アライメント数が1の基底クラスから派生したクラスの先頭メンバが1byteデータの場合、パディングを作らないようにデータメンバを割り付けます。

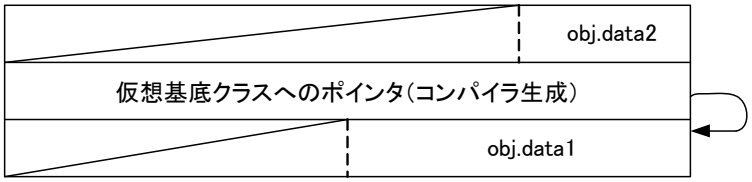
例：

```
class A{
   char data1;
};
class B:public A{
   char data2;
   short data3;
}obj;
```

クラスに仮想基底クラスがある場合、仮想基底クラスへのポインタを割り付けます。

例：

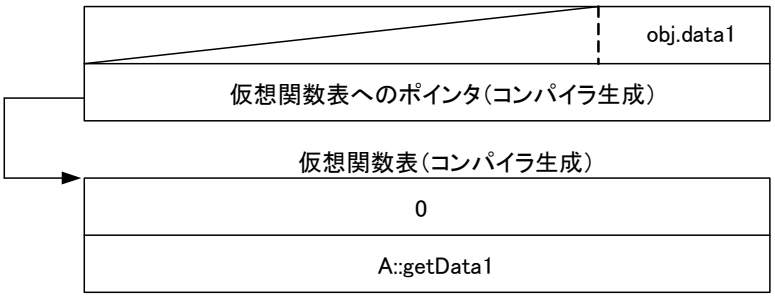
```
class A{
    short data1;
};
class B: virtual protected A{
    char data2;
}obj;
```



クラスに仮想関数がある場合、コンパイラは仮想関数表を生成し、仮想関数表へのポインタを割り付けます。

例：

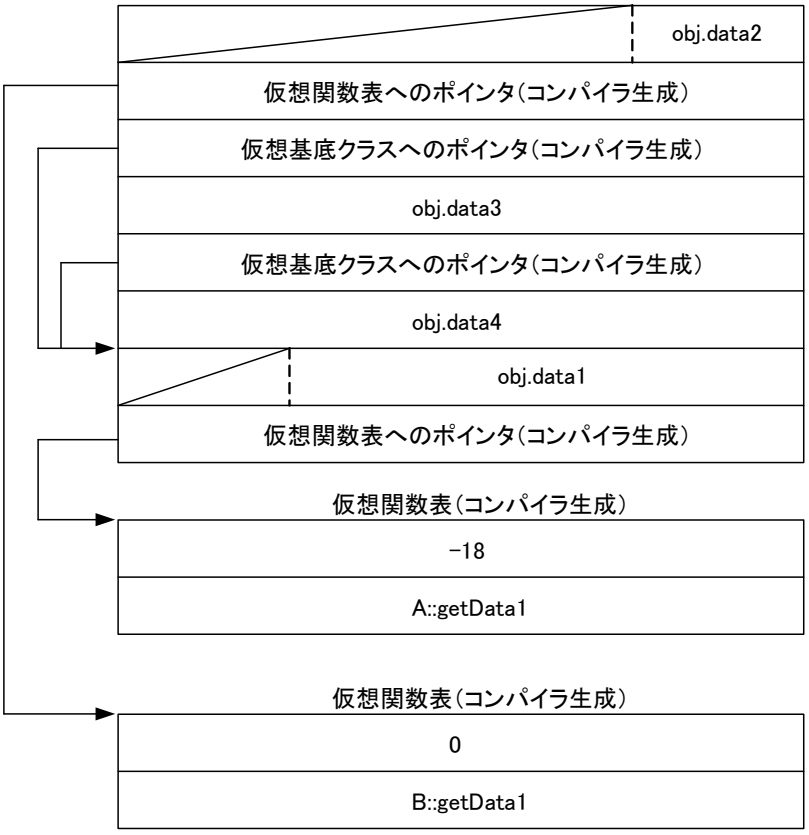
```
class A{
    char data1;
  public:
    virtual char getData1();
}obj;
```



仮想基底クラス、基底クラス、仮想関数があるクラスの例を示します。

例：

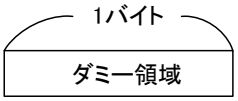
```
class A{
    char data1 ;
    virtual char getData1();
};
class B:virtual public A{
    char data2;
    char getData2();
    char getData1();
};
class C:virtual protected A{
    int data3;
};
class D:virtual public A,public B,public C{
  public:
    int data4;
    char getData1();
}obj;
```


obj.data2
仮想関数表へのポインタ(コンパイラ生成)
仮想基底クラスへのポインタ(コンパイラ生成)
obj.data3
仮想基底クラスへのポインタ(コンパイラ生成)
obj.data4
obj.data1
仮想関数表へのポインタ(コンパイラ生成)
仮想関数表(コンパイラ生成)
-18
A::getData1
仮想関数表(コンパイラ生成)
0
B::getData1

[1]空クラスの場合、1バイトのダミー領域を割り付けます。

例：

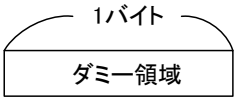
```
class A{
   void fun();
}obj;
```



空クラスを基底クラスに持つ空クラスの場合でも、ダミー領域は1バイトになります。

例：

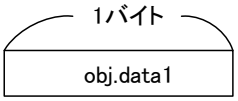
```
class A{
   void fun();
};
class B: A{
   void sub();
}obj;
```



空クラスのダミー領域は、クラスサイズが0の場合に割り付けます。基底クラスや派生クラスにデータメンバがある場合や、仮想関数があるクラスの場合には、ダミー領域は割り付けません。

例：

```
class A{
   void fun();
};
class B: A{
   char data1;
}obj;
```



(3) ビットフィールド

ビットフィールドは、構造体、共用体、クラスの中にビット幅を指定して割り付けるメンバです。

本項では、ビットフィールド特有の割り付け規則について説明します。

(a) ビットフィールドのメンバ

表9.17 にビットフィールドメンバの仕様を示します。

表 9.17 ビットフィールドメンバの仕様

	項目	仕様
1	ビットフィールドで許される型指定子	(unsigned) char、signed char、bool*1、_Bool*5、 (unsigned) short、signed short、enum、 (unsigned) int、signed int、 (unsigned) long、signed long、 (unsigned) long long、signed long long
2	宣言された型に拡張するときの符号の扱い*2	符号なし(unsigned)は、ゼロ拡張*3 符号付き(signed)は、符号拡張*4
3	符号指定なしの型の符号型	符号なし(unsigned) 但し、signed_bitfield オプションが指定された場合は、符号付き(signed)
4	enum 型の符号型	符号付き (signed) 但し、auto_enum オプションが指定された場合は、その結果の型に従う

【注】 *1 C++プログラム、または stdbool.h をインクルードした C99 プログラムのコンパイル時のみ bool を指定できます。

*2 ビットフィールドのメンバを使用する場合、ビットフィールドに格納したデータを宣言した型に拡張して使用します。符号付き(signed)で宣言されたサイズが 1 ビットのビットフィールドのデータは、データそのものを符号として解釈します。したがって、表現できる値は 0 と-1 だけになります。

*3 ゼロ拡張：拡張するときに上位のビットにゼロを補います。

*4 符号拡張：拡張するときにビットフィールドデータの最上位ビットを符号として解釈し、データより上位のビット全てに符号ビットを補います。

*5 C99 プログラムのみ有効です。_Bool 型は bool 型と同じ型としてコンパイルされます。

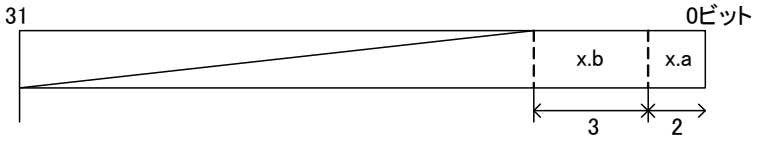
(b) ビットフィールドの割り付け方

ビットフィールドは、以下の 5 つの規則に従って割り付けます。

・ ビットフィールドのメンバは領域内で右(下位ビット側)から順に詰め込みます。

例：

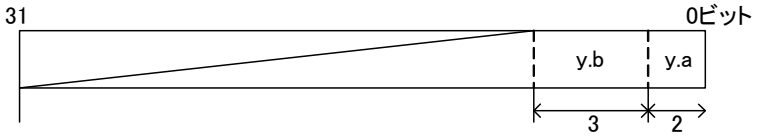
```
struct b1 {
  int a:2;
  int b:3;
} x;
```



・ 同じサイズの型指定子が連続している場合は、可能な限り同じ領域に詰め込みます。

例：

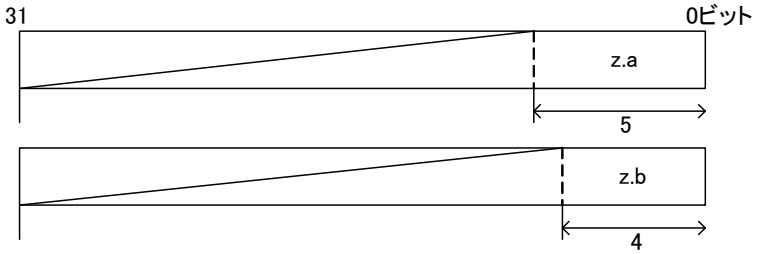
```
struct b1 {
  long            a:2;
  unsigned int    b:3;
} y;
```



・ 異なるサイズの型指定子で宣言されたメンバは、次の領域に割り付けます。

例：

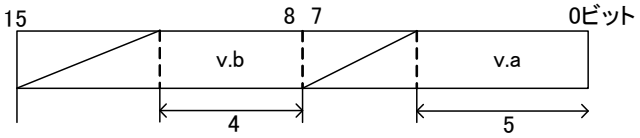
```
struct b1 {
  int   a:5;
  char  b:4;
} z;
```



・ 同じサイズの型指定子が連続していても、詰め込み先の領域の残りビットが、次のビットフィールドのサイズより小さい場合は、残りの領域は未使用領域となり、次の領域に割り付けます。

例：

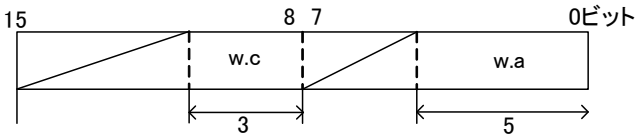
```
struct b2 {
  char a:5;
  char b:4;
} v;
```



・ ビット幅0のビットフィールドのメンバを指定すると、次のメンバからは、強制的に次の領域に割り付けます。

例：

```
struct b2 {
  char a:5;
  char  :0;
  char c:3;
} w;
```



【備考】 ビットフィールドメンバを上位ビット側から詰め込むことも可能です。
詳細は、「2. コンパイラオプション」の bit_order **オプション、「**9.2.1 #pragma**」の**#pragma bit_order **を参照してください。**

(4) Big Endianのメモリ割り付け

Big Endian でのメモリ上のデータ配列は以下のとおりです。

(a) 1 バイトデータ((signed)char、unsigned char、bool、_Bool型)

1 バイトデータの中のビット並び順は、Little Endian の場合も、Big Endian の場合も同じです。

(b) 2 バイトデータ((signed)short、unsigned short型)

2 バイトデータの中のバイト並び順は、Little Endian と Big Endian で上位、下位のバイトが逆になります。

例

0x100 番地に 2 バイトデータ 0x1234 がある場合

Little Endian:	0x100 番地：0x34	Big Endian:	0x100 番地：0x12
	0x101 番地：0x12		0x101 番地：0x34

(c) 4 バイトデータ((signed)int、unsigned int、(signed)long、unsigned long、float型)

4 バイトデータの中のバイト並び順は、Little Endian と Big Endian で 4 バイトのデータの順序が逆になります。

例

0x100 番地に 4 バイトデータ 0x12345678 がある場合

Little Endian:	0x100 番地：0x78	Big Endian:	0x100 番地：0x12
	0x101 番地：0x56		0x101 番地：0x34
	0x102 番地：0x34		0x102 番地：0x56
	0x103 番地：0x12		0x103 番地：0x78

(d) 8 バイトデータ((signed)long long、unsigned long long、double型)

8 バイトデータの中のバイト並び順は、Little Endian と Big Endian で 8 バイトのデータの順序が逆になります。

例

0x100 番地に 8 バイトデータ 0x0123456789abcdef がある場合

Little Endian:	0x100 番地：0xef	Big Endian:	0x100 番地：0x01
	0x101 番地：0xcd		0x101 番地：0x23
	0x102 番地：0xab		0x102 番地：0x45
	0x103 番地：0x89		0x103 番地：0x67
	0x104 番地：0x67		0x104 番地：0x89
	0x105 番地：0x45		0x105 番地：0xab
	0x106 番地：0x23		0x106 番地：0xcd
	0x107 番地：0x01		0x107 番地：0xef

(e) 構造体/共用体、クラス型データ

構造体/共用体、クラス型データの各メンバの割り付けは Little Endian のときと同様です。ただし、各メンバのバイト並び順はそのデータサイズの規則に従って反転します。

例

0x100 番地に、

```
struct {
    short a;
    int b;
}z = {0x1234, 0x56789abc};
```

がある場合

Little Endian:	0x100 番地：0x34	Big Endian:	0x100 番地：0x12
	0x101 番地：0x12		0x101 番地：0x34
	0x102 番地：パディング		0x102 番地：パディング
	0x103 番地：パディング		0x103 番地：パディング
	0x104 番地：0xbc		0x104 番地：0x56
	0x105 番地：0x9a		0x105 番地：0x78
	0x106 番地：0x78		0x106 番地：0x9a
	0x107 番地：0x56		0x107 番地：0xbc

(f) ビットフィールド

ビットフィールドの各領域の割り付けも Little Endian のときと同様です。ただし、各領域のバイト並び順はそのデータサイズの規則に従って反転します。

例

0x100 番地に、

```
struct {
    long a:16;
    unsigned int b:15;
    short c:5;
}y={1,1,1};
```

がある場合

Little Endian:	0x100 番地：0x01	Big Endian:	0x100 番地：0x00
	0x101 番地：0x00		0x101 番地：0x01
	0x102 番地：0x01		0x102 番地：0x00
	0x103 番地：0x00		0x103 番地：0x01
	0x104 番地：0x01		0x104 番地：0x00
	0x105 番地：0x00		0x105 番地：0x01
	0x106 番地：パディング		0x106 番地：パディング
	0x107 番地：パディング		0x107 番地：パディング

9.1.3 浮動小数点型の仕様

(1) 浮動小数点型の内部表現

コンパイラで扱う浮動小数点型の内部表現は、IEEE の形式に準拠しています。ここでは、IEEE 形式の浮動小数点型の内部表現の概要について述べます。

なお、本節では dbl_size=8 オプションが指定されたものとして説明しています。dbl_size=4 オプションが指定された場合は、double 型および long double 型の内部表現は、float 型と同じになります。

(a) 内部表現の形式

float 型は IEEE の単精度形式(32 ビット)、double 型と long double 型は IEEE の倍精度形式(64 ビット)で表現します。

(b) 浮動小数点データフォーマット

float型およびdouble型とlong double型の浮動小数点データフォーマットを**図** 9.1に示します。

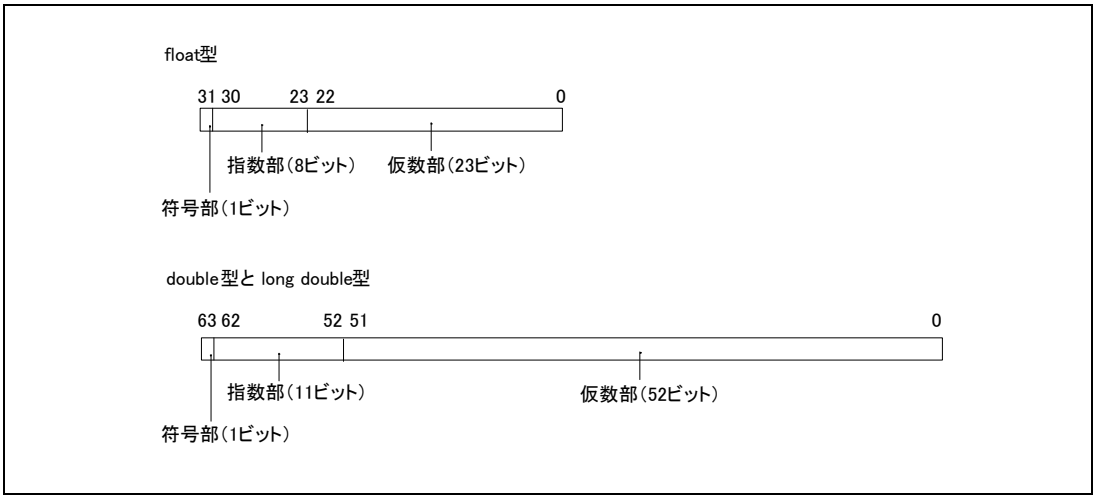


図 9.1　**浮動小数点データフォーマット**

内部表現の各構成要素の意味を以下に示します。

(i) 符号部

浮動小数点型の符号を示します。0のとき正、1のとき負を示します。

(ii) 指数部

浮動小数点型の指数を2のべき乗で示します。

(iii) 仮数部

浮動小数点型の有効数字に対応するデータです。

(c) 表現する値の種類

浮動小数点型は、通常の実数値のほかに、無限大等の値も表現することができます。浮動小数点型が表現する値の種類を以下に示します。

(i) 正規化数

指数部が0または全ビット1ではない場合です。通常の実数値を表現します。

(ii) 非正規化数

指数部が0で、仮数部が0でない場合です。絶対値の小さな実数値を表現します。

(iii)ゼロ

指数部および仮数部が0の場合です。値0.0を表現します。

(iv)無限大

指数部が全ビット1で仮数部が0の場合です。無限大を表現します。

(v) 非数

指数部が全ビット1で仮数部が0でない場合です。「0.0/0.0」、「∞/∞」、「∞-∞」等、結果が数値に対応しない演算の結果として得られます。

浮動小数点型の表現する値を決定する条件を**表** 9.18に示します。

表 9.18 浮動小数点型の表現する値の種類

仮数部	指数部		
	0	0 でも全ビット 1 でもない	全ビット 1
0	0	正規化数	無限大
0 以外	非正規化数		非数

【注】 非正規化数は、正規化数で表現できない範囲の絶対値の小さな浮動小数点型を表現しますが、正規化数と比較して有効桁数が少なくなっています。したがって、演算の結果あるいは途中結果が非正規化数となる場合、結果の有効桁数は保証しません。

denormalize=off を指定した場合、非正規化数は 0 として扱います。

denormalize=on を指定した場合、非正規化数は非正規化数のまま扱います。

(2) float型

float 型の内部表現は、1 ビットの符号部、8 ビットの指数部、23 ビットの仮数部からなります。

(i) 正規化数

符号部は、0(正)または1(負)で、値の符号を示します。

指数部は、1～254(2^8-2)の値をとります。実際の指数は、この値から127を引いた値で、その範囲は-126～127です。

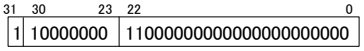
仮数部は、0～2^{23}-1の値をとります。実際の仮数は、2^{23}のビットを1と仮定し、その直後に小数点があるものとして解釈します。

正規化数の表現する値は、

$(-1)^{<符号部>} \times 2^{<指数部>-127} \times (1 + <仮数部> \times 2^{-23})$

となります。

例：



符号: －

指数: 10000000 (2) －127 ＝1

仮数: 1.11(2) ＝1.75

値: －1.75 × 2^1 ＝－3.5

(2) は2進数を意味します。

(ii) 非正規化数

符号部は0(正)または1(負)で、値の符号を示します。

指数部は0で、実際の指数は-126になります。

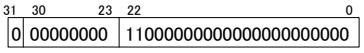
仮数部は、1～2^{23}-1で、実際の仮数は、2^{23}のビットを0と仮定し、その直後に小数点があるものとして解釈します。

非正規化数の表現する値は、

$(-1)^{<符号部>} \times 2^{-126} \times (<仮数部> \times 2^{-23})$

となります。

例：



符号: ＋

指数: －126

仮数: 0.11(2) ＝0.75

値: 0.75 × 2^{-126}

(2) は2進数を意味します。

(iii)ゼロ

符号部は0(正)または1(負)で、それぞれ+0.0、-0.0を示します。

指数部、仮数部はともに0です。

+0.0、-0.0は、ともに値としては0.0を示します。ゼロの符号による、各演算での機能の違いについては**「9.1.3(4) 浮動小数点演算の仕様」**を参照してください。

(iv) 無限大

符号部は0(正)または1(負)で、それぞれ+∞、-∞を示します。

指数部は255(2^8-1)です。

仮数部は0です。

(v) 非数

指数部は255(2^8-1)です。

仮数部は0以外の値です。

【注】 仮数部の最上位ビットが 1 の非数を qNaN、仮数部の最上位ビットが 0 の非数を sNaN と呼びます。その他の仮数フィールドの値、および符号部については規定していません。

(3) double型とlong double型

double 型と long double 型の内部表現は、1 ビットの符号部、11 ビットの指数部、52 ビットの仮数部からなります。

(i) 正規化数

符号部は0(正)または1(負)で、値の符号を示します。

指数部は、1～2046(2^{11}-2)の値をとります。実際の指数は、この値から1023を引いた値で、その範囲は-1022～1023です。

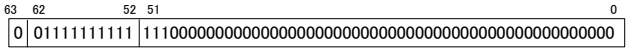
仮数部は、0～2^{52}-1の値となります。実際の仮数は、2^{52}のビットを1と仮定し、その直後に小数点があるものとして解釈します。

正規化数の表現する値は、

$(-1)^{<符号部>} \times 2^{<指数部>-1023} \times (1 + <仮数部> \times 2^{-52})$

となります。

例：



符号: ＋

指数: 1111111111(2) － 1023 ＝ 0

仮数: 1.111 (2) ＝ 1.875

値: 1.875 × 2^0 ＝ 1.875

(2) は2進数を意味します。

(ii) 非正規化数

符号部は0(正)または1(負)で、値の符号を示します。

指数部は0で、実際の指数は-1022になります。

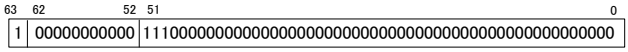
仮数部は、1～2^{52}-1で、実際の仮数は、2^{52}のビットを0と仮定し、その直後に小数点があるものとして解釈します。

非正規化数が表現する値は、

$(-1)^{<符号部>} \times 2^{-1022} \times (<仮数部> \times 2^{-52})$

となります。

例：



符号: －

指数: － 1022

仮数: 0.111 (2) ＝ 0.875

値: 0.875 × 2^{-1022}

(2) は2進数を意味します。

(iii)ゼロ

符号部は0(正)または1(負)で、それぞれ+0.0、-0.0を示します。

指数部、仮数部はともに0です。

+0.0、-0.0は、ともに値としては0.0を示します。ゼロの符号による、各演算での機能の違いについては**「9.1.3(4) 浮動小数点演算の仕様」**を参照してください。

(iv)無限大

符号部は0(正)または1(負)で、それぞれ+∞、-∞を示します。

指数部は2047(2^{11}-1)です。

仮数部は0です。

(v) 非数

指数部は2047(2^{11}-1)です。

仮数部は0以外の値です。

【注】 仮数部の最上位ビットが 1 の非数を qNaN、仮数部の最上位ビットが 0 の非数を sNaN と呼びます。その他の仮数フィールドの値、および符号部については規定していません。

(4) 浮動小数点演算の仕様

本項では、C/C++言語の機能として表現されている浮動小数点の四則演算、およびコンパイル時やCライブラリ関数の処理で生じる浮動小数点の10進表現と内部表現の間の変換の仕様について解説します。

(a) 四則演算の仕様

(i) 結果の値の丸め方

浮動小数点の四則演算の結果の正確な値が、内部表現の仮数の有効数字を超えた場合は、以下の規則に従って丸めを行います。

[1] 結果の値は、その値を近似する二つの浮動小数点型の内部表現のうち、近い方に向かって丸めます。

[2] 結果の値が、その値を近似する二つの浮動小数点型のちょうど中央になる場合は、仮数の最後の桁が0となる方向に丸めます。

(ii) オーバフロー、アンダフロー、無効演算時の処理

実行時のオーバフロー、アンダフロー、無効演算に対しては、以下の処理を行います。

[1] オーバフローの場合は、結果の符号に従って正または負の無限大になります。

[2] アンダフローの場合は、非正規化数となります。

[3] 無効演算は、符号が逆の無限大を加算した場合、符号が同じ無限大を減算した場合、ゼロと無限大を乗算した場合、ゼロをゼロで、あるいは無限大を無限大で除算した場合に生じます。
これらの場合、結果は非数になります。

[4] 浮動小数点型から整数へ変換したときにオーバフローが生じた場合、結果の値は保証しません。

【注】 定数式に関しては、コンパイル時に演算を行います。この時にオーバフロー、アンダフロー、無効演算を検出した場合は、ウォーニングレベルのエラーになります。

(iii)特殊な値(ゼロ、無限大、非数)の演算に関する注意事項

[1] 正のゼロと負のゼロの和は正のゼロとなります。

[2] 同符号のゼロの差は正のゼロになります。

[3] 被演算子の一方あるいは両方に非数を含む演算の結果は、常に非数になります。

[4] 比較演算においては、正のゼロと負のゼロは等しいものとして扱います。

[5] 被演算子の一方あるいは両方が非数であるような比較演算、等値演算の結果は、「!=」については常に真、その他は常に偽となります。

(b) 10 進表現と内部表現の間の変換

本項ではソースプログラム上の浮動小数点定数と内部表現の間の変換、あるいはCライブラリ関数による ASCII 文字列による浮動小数点型の 10 進表現と、内部表現の間の変換の仕様について解説します。

(i) 10進表現から内部表現に変換する場合、まず10進表現を10進表現の正規形に変換します。

10進表現の正規形は、「$\pm M \times 10^{\pm N}$」の形式で、M、Nの範囲は以下のとおりです。

[1] float型の正規形

$0 \leqq M \leqq 10^{9}-1$

$0 \leqq N \leqq 99$

[2] double型とlong double型の正規形

$0 \leqq M \leqq 10^{17}-1$

$0 \leqq N \leqq 999$

正規形に変換できない10進表現については、オーバフロー、またはアンダフローになります。また、10進表現が正規形よりも多くの有効数字を含んでいる場合は、下位の桁は切り捨てます。これらの場合、コンパイル時にはウォーニングレベルのエラーになり、実行時には対応するエラーの番号を変数errnoに設定します。

また、正規形に変換するためには、もとの10進表現のASCII文字列としての長さが511文字以下でなければなりません。そうでない場合、コンパイル時にはエラーになり、実行時には対応するエラーの番号を変数errnoに設定します。

内部表現から10進表現に変換する場合には、一度10進表現の正規形に変換してから、指定した書式に従ってASCII文字列に変換します。

(ii) 10進表現の正規形と内部表現の間の変換

10進表現の正規形と内部表現の間の変換は、指数が大きいときや小さいときには、正確な変換ができません。以下に、正確な変換ができる範囲と、その範囲外の場合での誤差の限界値について解説します。

[1] 正確な変換ができる範囲

以下に示す指数の範囲の浮動小数点型については、**「(a)(i) 結果の値の丸め方」**に示す丸めが正確に行われます。この範囲ではオーバフロー、アンダフローは生じません。

float型の場合：$0 \leqq M \leqq 10^{9}-1$、$0 \leqq N \leqq 13$

double型とlong double型の場合：$0 \leqq M \leqq 10^{17}-1$、$0 \leqq N \leqq 27$

[2] 誤差の限界値

[1]で示す範囲に入っていない値を変換する場合の誤差と、正確な丸めを行ったときの誤差の差は、有効数字の最小位桁の0.47倍を超えません。

また、[1]で示した範囲を超えている場合、変換の際にオーバフローやアンダフローが生じる場合があります。この場合、コンパイル時にはウォーニングレベルのエラーになり、実行時には対応するエラーの番号を変数errnoに設定します。

9.1.4 演算子の評価順序

式の中に複数の演算子がある場合、それらの演算子の評価順序は、優先順位と「右」または「左」で表わされる結合性によって決まります。

各演算子の優先順位と結合性を**表** 9.19 に示します。

表 9.19 **演算子の優先順位と結合性**

優先順位	演算子	結合性	適用される式
1	++ -- (後置) ( ) [ ] -> .	左	後置式
2	++ -- (前置) ! ~ + - * & sizeof	右	単項式
3	(型名)	右	キャスト式
4	* / %	左	乗除式
5	+ -	左	加減式
6	<< >>	左	ビット単位のシフト式
7	< <= > >=	左	関係式
8	== !=	左	等価式
9	&	左	ビット単位の AND 式
10	^	左	ビット単位の排他 OR 式
11	|	左	ビット単位の OR 式
12	&&	左	論理 AND 式
13	||	左	論理 OR 式
14	?:	左	条件式
15	= += -= *= /= %= <<= >>= &= |= ^=	右	代入式
16	,	左	カンマ式

9.1.5 準拠する言語仕様

(1) C言語仕様 (lang=c**オプション選択時**)

ANSI/ISO 9899-1990 American National Standard for Programming Languages -C

(2) C言語仕様 (lang=c99 **オプション選択時**)

ISO/IEC 9899:1999 INTERNATIONAL STANDARD Programming Languages - C

(3) C++仕様 (lang=cpp**オプション選択時**)

Microsoft(R) Visual C/C++ 6.0 と互換性のある言語仕様に基づいています。

9.2 拡張機能

コンパイラの拡張機能として、以下の機能をサポートしています。

- #pragma、キーワード
- 組み込み関数
- セクションアドレス演算子

9.2.1 #pragma、キーワード

#pragma およびキーワードの一覧を**表** 9.20 に示します。

なお、最適化に関わる#pragma は、条件により、適用されない場合もあります。当該最適化が適用されたかどうかは出力コードで確認してください。

表 9.20 #pragma、キーワード一覧

	対象	#pragma 拡張子*1	機能
1	メモリ配置	#pragma section	セクションの切り替え指定
2		#pragma stacksize	スタックセクションの作成
3	関数	#pragma interrupt	割り込み関数の作成
4		#pragma inline #pragma noinline	関数のインライン展開を指定
5		#pragma inline_asm	アセンブリ記述関数のインライン展開
6		#pragma entry	エントリ関数の作成
7		#pragma option	関数単位オプション指定
8	その他	#pragma bit_order	ビットフィールドの並び順指定
9		#pragma pack #pragma unpack #pragma packoption	構造体メンバおよびクラスメンバの境界調整数指定
10		#pragma address	変数に絶対アドレスを指定
11		#pragma endian	初期値のエンディアン指定
12		__evenaccess	変数の型のサイズのアクセスを保証
13		far *2 _far *2 near *2 _near *2	予約キーワード
14	関数	#pragma instalign4 #pragma instalign8 #pragma noinstalign	分岐先の命令実行向け整合を行う関数を指定

【注】 *1 各#pragma のキーワードでは、アルファベットの大文字・小文字を区別します。
このため、キーワードの小文字を大文字で書いたものは、警告 C5161 (W)となり受け付けられません。

*2 far, _far, near および_near は、キーワードとして予約されています。
修飾子として認識はしますが、コード生成には影響しません。

セクション切り替え指定

#pragma section

書　式　　#pragma section [<セクション種別>] [△<変更セクション名>]

<セクション種別>: { P | C | D | B }

説　明　　コンパイラの出力するセクション名を切り替えます。

セクション種別と変更セクションを指定した場合、セクション種別が P であればその#pragma 宣言以降に記述された関数のセクション名を変更します。セクション種別が C、D、または B の場合は、その#pragma 宣言以降に実体を定義した全てのセクション名を変更します。

変更セクション名のみを指定した場合、その#pragma 宣言以降にあるプログラム領域、定数領域、初期化データ領域、および未初期化データ領域のすべてのセクション名を変更します。この場合、各セクションの変更後のセクション名は、各デフォルトセクション名の後に<変更セクション名>の文字列を追加したセクション名となります。

セクション種別も変更セクション名も記述しなかった場合は、その#pragma 宣言以降にあるプログラム領域、定数領域、初期化データ領域、および未初期化データ領域の全てのセクション名をデフォルトセクション名に戻します。

各セクション種別のデフォルトセクション名は、section オプションの指定があればそれに従います。ない場合はセクション種別名をそのまま用います。

例 1　セクション名とセクション種別を指定した場合

```
#pragma section B Ba
int i;                          // Ba セクションに配置
void func(void)
{
(省略)
}

#pragma section B Bb
int j;                          // Bb セクションに配置
void sub(void)
{
(省略)
}
```

例 2 セクション種別を省略した場合

```
#pragma section abc
int a;                          // Babc セクションに配置
const int c=1;                  // Cabc セクションに配置
void f(void)                    // Pabc セクションに配置
{
    a=c;
}

#pragma section
int b;                          // B セクションに配置
void g(void)                    // P セクションに配置
{
    b=c;
}
```

備 考 #pragma section は関数定義の外で宣言しなければなりません。

次の項目のセクション名は変更できません。section オプションを使用してください。

(1) 文字列リテラル

(2) switch 文の分岐テーブル

1 ファイルあたりの#pragma section で指定できるセクション数は最大 2045 個です。

静的クラスメンバ変数のセクションを指定する場合は、クラスのメンバ宣言と実体の定義の両方に#pragma section の指定が必要になります。

例

```
/*
** クラスメンバ宣言
*/

 class A
 {
        private:

        // 初期値なし
        #pragma section DATA
static int data_;
#pragma section
```

```
// 初期値あり
#pragma section TABLE
static int table_[2];
#pragma section
  };

  /*
  ** 実体定義
  */

  // 初期値なし
  #pragma section DATA
  int A::data_;
  #pragma section

  // 初期値あり
  #pragma section TABLE
  int A::table_[2]={0, 1};
  #pragma section
```

スタックセクション作成

#pragma stacksize

書　式　#pragma stacksize {si=<定数> | su=<定数>}

説　明　si=<定数>を指定した場合、セクション名 SI、サイズ<定数>のスタックとして使用するデータセクションを作成します。

su=<定数>を指定した場合、セクション名 SU、サイズ<定数>のスタックとして使用するデータセクションを作成します。

例　C ソース：

```
#pragma stacksize si=100
#pragma stacksize su=200
```

コード展開例：

```
.SECTION    SI,DATA,ALIGN=4
.BLKB       100
.SECTION    SU,DATA,ALIGN=4
.BLKB       200
```

備　考　si, su 指定はファイル内でそれぞれ 1 回しか指定できません。

<定数>は必ず 4 の倍数を指定してください。

割り込み関数作成

#pragma interrupt

書　式　#pragma interrupt [(]<関数名>[(＜割り込み仕様＞[,…])][,…][)]

説　明　#pragma interrupt を用いて割り込み関数となる関数を宣言します。
関数名には、グローバル関数および静的関数メンバを指定できます。
割り込み仕様の一覧を**表** 9.21 に示します。

表 9.21　**割り込み仕様の一覧**

	項目	形式	オプション	指定内容
1	**ベクタテーブル指定**	vect=	<**ベクタ番号**>	**割り込み関数のアドレスを配置するベクタ番号**
2	**高速割り込み指定**	fint	**なし**	**高速割り込みに使用する関数の指定** RTFI **命令でリターン**
3	**割り込み関数レジスタ制限指定**	save	**なし**	**割り込み関数内で使用するレジスタの本数を制限し、退避・回復の数を減らす**
4	**多重割り込み許可指定**	enable	**なし**	**関数の先頭で** PSW **の** I **フラグを** 1 **にし、多重割り込みを許可する**
5	ACC **保存指定**	acc	**なし**	**割り込み関数内で** ACC **を退避・回復する**
6	ACC **非保存指定**	no_acc	**なし**	**割り込み関数内で** ACC **を退避・回復しない**

#pragma interrupt を用いて宣言した関数は、関数の処理の前後で全レジスタを保証(関数入口/出口において関数内で使用する全レジスタを退避･回復)し、通常 RTE 命令でリターンします。

割り込み仕様を指定しない場合は単純な割り込み関数として処理します。

ベクタテーブル指定(vect=)をした場合は C$VECT セクション内の指定したベクタテーブル番号位置にその関数アドレスを設定します。

高速割り込み指定(fint)をした場合は、RTFI 命令でリターンします。また、fint_register オプションを指定した場合は、オプションで指定したレジスタを退避、回復せずに割り込み関数で使用します。

割り込み関数レジスタ制限指定(save)をした場合は、割り込み関数内で使用するレジスタの本数を R1～R5、および R14～R15 に制限します。R6～R13 は割り込みで使用しないため、退避・回復命令は生成しません。

多重割り込み許可指定(enable)をした場合は、割り込み関数の先頭で PSW の I フラグを 1 にし、多重割り込みを許可します。

ACC 保存指定(acc)をした場合で、指定された関数から別の関数呼び出しがあったり、関数内で ACC を書き換える命令を使う場合は、ACC を退避および回復する命令を生成します。

ACC 非保存指定(no_acc)をした場合は、ACC を退避および回復する命令を生成しません。

acc および no_acc のどちらの指定もない場合は、コンパイルオプションに従います。

割り込み関数の定義に対して指定できる関数は、グローバル関数(C/C++言語)と静的関数メンバ(C++言語)です。

関数の返却値の型は void のみです。return 文の返却値を指定することはできません。指定があった場合はエラーを出力します。

例 1 正しい宣言と誤った宣言の例

```
#pragma interrupt (f1, f2)
void f1(){…}            // 正しい宣言です。
int f2(){…}             // 返却値の型が void ではないのでエラーになります。
```

例 2 通常の割り込み関数の例

C ソース：

```
#pragma interrupt func
void func(){ .... }
```

出力コード：

```
_func:
        PUSHM   R1-R3   ; 関数内で使用しているレジスタを退避
        ....
        (R1,R2,R3 を関数内で使用)
        ....
        POPM    R1-R3   ; 入口で退避したレジスタを回復
        RTE
```

例 3 関数呼び出しがある割り込み関数の例

関数内で使用しているレジスタに加えて、関数呼び出し前後で保証しないレジスタについても、割り込み関数の入口で退避し、出口で回復します。

C ソース：

```
#pragma interrupt func
void func(){
    ....
    sub();
    ....
  }
```

出力コード：

```
_func:
        PUSHM   R1-R5           ; R1～R5 を退避
        PUSHM   R14-R15         ; R14,R15 を退避
        ....
        MOV.L   #_sub,R15
        JSR     R15             ; 関数呼び出し
        ....
        POPM    R14-R15         ; R14,R15 を回復
        POPM    R1-R5           ; R1～R5 を回復
        RTE
```

例 4　割り込み仕様 fint を使用した場合の例

C ソース：fint_register=2 オプションを指定してコンパイル

```
#pragma interrupt func(fint)
void func1(){ .... }   // 割り込み関数
void func2(){ .... }   // 通常関数
```

出力コード：

```
_func1:
        PUSHM   R1-R3       ; 関数内で使用しているレジスタを退避
        ....                ; (但し、R12,R13 は退避しない)
        ....
        (R1,R2,R3,R12,R13 を関数内で使用)
        ....
        POPM    R1-R3       ; 入口で退避したレジスタを回復
        RTE

_func2:
        ....                ; #pragma interrupt fint 指定した関数以外では、
        ....                ; R12,R13 を使用しないコードを生成する
```

例 5　割り込み仕様 acc を使用した場合の例

```
void func(void);
#pragma interrupt accsaved_ih(acc)     /* "acc"を指定 */
void accsaved_ih(void)
{
        func();
}
```

出力コード：

```
 _accsaved_ih:              ; function: accsaved_ih
            .STACK          _accsaved_ih=44 ; ACC 退避分(8byte)を含む
            PUSHM           R1-R5
            PUSHM           R14-R15
            MVFACMI R4                  ; ACC 退避コード(1/4)
            SHLL #10H,R4                ; ACC 退避コード(2/4)
            MVFACHI R5                  ; ACC 退避コード(3/4)
            PUSHM     R4-R5             ; ACC 退避コード(4/4)
            BSR             _func
            POPM      R4-R5             ; ACC 回復コード(1/3)
            MVTACLO R4                  ; ACC 回復コード(2/3)
            MVTACHI R5                  ; ACC 回復コード(3/3)
            POPM            R14-R15
            POPM            R1-R5
            RTE
```

備　考　　最適化により削除される場合がありますので、static 関数は指定しないでください。

RX 命令セットの仕様のため、acc フラグで退避・回復できるのは、ACC の上位 48 ビットに限られます。ACC の下位 16 ビットは退避・回復されません。

各割り込み仕様は、小文字のみ指定できます。大文字を指定してもエラーになります。

割り込み仕様として vect を使った場合、指定の無い空きベクタのアドレスは 0 になります。このアドレスは、最適化リンケージエディタで任意のアドレスやシンボルに変更することができます。詳しくは、5.2.2 出力オプションの VECT および VECTN オプションの項目を参照してください。

＜acc、no_acc の用途＞

acc、no_acc は次のような用途を考慮したものです。

・ACC の補償を save_acc で行う場合の割り込み応答速度低下の対策(no_acc)

既存の割り込み関数で ACC の補償には save_acc オプションが有効だが、割り込み応答速度が悪化ことがあるため、不要な ACC の退避・回復を関数単位で抑止する手段をとして no_acc を用意する。

・ACC の退避・回復をソースコードで制御(acc と no_acc)

ACC の退避・回復の考慮が完了している割り込み関数には、明示的に acc、no_acc を選択しておくことで、ソースプログラムで ACC の退避・回復を save_acc に依存せずに定義できる。

#pragma inline, #pragma noinline

書　式　#pragma inline [(]<関数名>[,…][)]

#pragma noinline [(]<関数名>[,…][)]

説　明　#pragma inline は、インライン展開する関数を宣言します。

noinline オプションが指定された場合でも、#pragma inline 指定された関数はインライン展開の対象となります。

#pragma noinline は、inline オプションの指定を抑止する関数を宣言します。

関数名には、グローバル関数および静的関数メンバを指定できます。

#pragma inline で指定した関数名の関数と関数指定子 inline(C++言語および C(C99)言語)を指定した関数は、その関数を呼び出したところにインライン展開されます。

例　ソースファイル

```
#pragma inline(func)
static int func (int a, int b)
{
    return (a+b)/2;
}
int x;
main()
{
    x=func(10,20);
}
```

展開イメージ

```
int x;
main()
{
    int func_result;
    {
        int a_1=10, b_1=20;
        func_result=(a_1+b_1)/2;
    }
    x=func_result;
}
```

備　考　#pragma inline が指定された場合でも、以下のいずれかに該当する場合はインライン展開しません。

－可変引数を持つ関数である。

－関数内で仮引数のアドレスを参照している。

－展開対象関数のアドレスを介して呼び出しを行っている。

#pragma inline は、インライン展開されることを保証するものではありません。コンパイル時間やメモリ使用量の増大を考慮した制限により、インライン展開を抑止することがあります。なお、インライン展開が抑止される際に、noscope オプションを指定すると、インライン展開されるようになる場合があります。

#pragma inline は、関数本体の定義の前に指定してください。

#pragma inline で指定した関数に対しても外部定義を生成します。static 関数に#pragma inline を指定した場合、関数定義はインライン展開後に削除されます。

inline(C++言語および C(C99)言語)が指定された関数には、外部定義を生成しません。

#pragma inline_asm

書　式　　#pragma inline_asm[(]<関数名>[,…][)]

説　明　　#pragma inline_asm で宣言したアセンブリ記述関数をインライン展開します。

アセンブラ埋め込みインライン関数の呼び出し規則は通常関数の呼び出し規則と同様です。

例　　C ソース：

```
#pragma inline_asm func
static int func(int a, int b){
        ADD     R2,R1       ; アセンブリ記述
}
main(int *p){
*p = func(10,20);
}
```

出力コード：

```
_main:
        PUSH.L  R6
        MOV.L   R1,R6
        MOV.L   #20,R2
        MOV.L   #10,R1
        ADD     R2,R1       ; インライン展開
        MOV.L   R1,[R6]
        POP     R6
        RTS
```

備　考　　#pragma inline_asm は、関数本体の定義の前に指定してください。

#pragma inline_asm で指定した関数に対しても外部定義を生成します。

アセンブラ埋め込みインライン関数内で関数の出入口で保証するレジスタ（**表** 8.2 を参照）を使用する場合は、アセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタの退避・回復が必要です。

アセンブラ埋め込みインライン関数には、RX ファミリの命令およびテンポラリラベルだけを記述してください。テンポラリラベル以外のラベルを定義したり、アセンブラ制御命令を記述することはできません。

アセンブラ埋め込みインライン関数の最後に RTS を記述しないでください。

関数名に関数メンバを指定することはできません。

static 関数に#pragma inline_asm を指定した場合、関数定義はインライン展開後に削除されます。

アセンブリ記述は、プリプロセッサの処理対象となります。このため、アセンブリ言語で使用される命令やレジスタと同じ名前のマクロ(例: "MOV"や"R5"など)を#define でマクロ定義する場合はご注意ください。

エントリ関数作成

#pragma entry

書　式　　#pragma entry[(]<関数名>[)]

説　明　　<関数名>で指定した関数をエントリ関数として扱います。
エントリ関数では、レジスタの退避・回復コードを一切作成しません。
#pragma stacksize 宣言があると、関数先頭でスタックポインタの初期設定コードを出力します。
base オプションを指定した場合、オプションで指定したベースレジスタへの設定を行います。

例　　C ソース：-base=rom=R13 を指定

```
#pragma stacksize su=100
#pragma entry INIT
void INIT() {
:
}
```

出力コード：

```
.SECTION    SU,DATA,ALIGN=4
.BLKB       100
.SECTION    P,CODE
_INIT:
MVTC        (TOPOF SU + SIZEOF SU),USP
MOV.L       #__ROM_TOP,R13
```

備　考　　#pragma entry 指定は、関数の宣言前に行ってください。
ロードモジュール全体でエントリ関数を複数指定することはできません。

関数単位オプション指定

#pragma option

書　式　#pragma option [<オプション列>]

説　明　#pragma option を用いて、オプション列で指定したオプションを有効にします。
指定されたオプションはファイルの終わり、又は<オプション列>がない#pragma option が設定された部分まで適用されます。
#pragma option <オプション列>を指定すると、<オプション列>で指定された最適化を行います。使用可能な最適化は、**表** 9.22 の通りです。各最適化オプションについては**「第 2 章　C/C++コンパイラオプション」**を参照してください。

表 9.22　#pragma option **で使用可能な最適化オプション**

	オプション指定方法	オプション解除方法
1	const_div	noconst_div
2	optimize = {0 | 1 | 2}	**なし**
3	speed	size
	size	speed
4	loop=n　(n **は** 2**〜**32 **の整数**)	loop=1
5	case={ ifthen | table | auto }	**なし**
6	schedule	noschedule
7	scope	noscope

例　C ソース：コンパイラオプション指定なし（デフォルト）

```
#pragma option speed
void func1(){ ... }          /* デフォルト + -speed が有効                    */
#pragma option optimize=0
void func2(){ ... }          /* デフォルト + -speed + -optimize=0 が有効   */
#pragma option
void func3(){ ... }          /* デフォルトが有効                              */
```

#pragma bit_order

書　式　　#pragma bit_order [{left | right}]

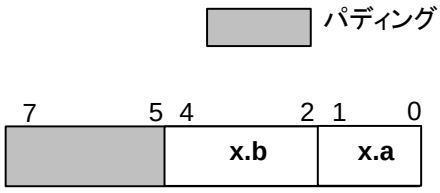
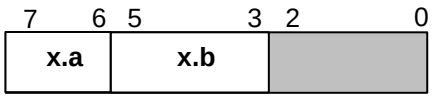
説　明　　ビットフィールドの並び順の切り替えを指定します。

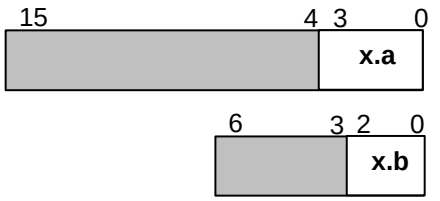
left を指定した場合は上位ビット側から、right を指定した場合は下位ビット側から、それぞれメンバが割り付けられます。

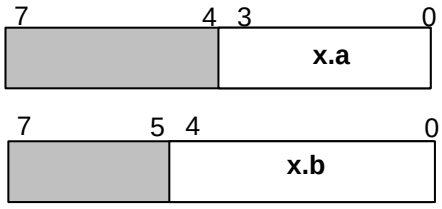
デフォルトの設定は right です。

left|right を省略すると、以降はオプションに従います。

例

C ソース	ビット配置
#pragma bit_order right struct tbl_r { 　unsigned char a:2; 　unsigned char b:3; } x;	
#pragma bit_order left struct tbl_l { 　unsigned char a:2; 　unsigned char b:3;} y;	

// 異なるサイズのメンバの場合
#pragma bit_order right
struct tbl_r {
　unsigned short a:4;
　unsigned char b:3;
} x;
	

// 型のサイズを超える場合
#pragma bit_order right
struct tbl_r {
　unsigned char a:4;
　unsigned char b:5;
} x;
	

構造体メンバおよびクラスメンバのアライメント数指定

#pragma pack
#pragma unpack
#pragma packoption

書　式　#pragma pack
#pragma unpack
#pragma packoption

説　明　ソースプログラム中の指定位置以降の構造体メンバおよびクラスメンバのアライメント数を指定します。本拡張子が指定されていない場合または#pragma packoption 指定位置以降で宣言された構造体メンバおよびクラスメンバのアライメント数は pack オプションの指定に従います。#pragma pack 拡張子とアライメント数の関係を**表** 9.23 に示します。

表 9.23　#pragma pack **とメンバのアライメント数**

メンバの型	#pragma pack	#pragma unpack	#pragma packoption **または指定なし**
(signed) char	1	1	1
(unsigned) short	1	2	pack **オプションに従う**
(unsigned) int *, (unsigned) long, (unsigned) long long, **浮動小数点型**, **ポインタ型**	1	4	pack **オプションに従う**

例

```
#pragma pack
struct S1 {
    char a;             /* バイトオフセット=0          */
    int b;              /* バイトオフセット=1          */
    char c;             /* バイトオフセット=5          */
} ST1;                  /* 合計サイズ 6 バイト         */

#pragma unpack
struct S2 {
    char a;             /* バイトオフセット=0          */
                        /* 3 バイト空き領域            */
    int b;              /* バイトオフセット=4          */
    char c;             /* バイトオフセット=8          */
                        /* 3 バイト空き領域            */
} ST2;                  /* 合計サイズ 12 バイト        */
```

備　考　#pragma pack を指定した構造体メンバおよびクラスのメンバはポインタを用いてアクセスすることはできません(ポインタを使用したメンバ関数内でのアクセスを含みます)。

例

```
#pragma pack
struct st {
    char x;
    int y;
} ST;
int *p=&ST.y; /* ST.y のアドレスが奇数になる場合があります */
void func(void) {
    ST.y=1; /* 正しくアクセスできます */
    *p=1; /* 正しくアクセスできない場合があります */
}
```

構造体、共用体、クラスメンバのアライメント数は pack オプションでも指定できます。オプションと#pragma の両方が指定された場合は、#pragma の指定を優先します。

#pragma address

書　式　　#pragma address [(]<変数名>=<絶対アドレス>[,…][)]

説　明　　指定した変数を指定したアドレスに割り付けます。その際、コンパイラが指定した変数ごとにセクションを設定し、リンク時に指定した絶対アドレスに割り付けます。連続したアドレスに変数を指定した場合、それらの変数は同一セクションにします。

例　　C ソース：

```
#pragma address X=0x7f00
int X;
main(){
    X=0;
}
```

出力コード：

```
_main:
    MOV.L       #0,R5
    MOV.L       #32512,R14      ; 0x7f00
    MOV.L       R5,[R14]
    RTS
    .SECTION    $ADDR_B_7F00,DATA
    .ORG        7F00H
    .glb        _X
_X:                             ; static: X
    .blkl       1
```

備　考　　#pragma address 指定は、変数の宣言前に行ってください。
構造体/共用体のメンバ、もしくは変数以外を指定した場合はエラーとなります。
#pragma address を同一の変数に対して複数回指定した場合はエラーとなります。
同一の変数に対して#pragma section を同時に指定した場合はエラーとなります。

初期値のエンディアン指定

#pragma endian

書　式　#pragma endian [{big | little}]

説　明　静的オブジェクトを格納する領域のエンディアンを指定します。

#pragma endian を記述した行から、ファイルの末尾か、または次の#pragma endian を記述した行の手前までに定義したものが対象となります。

big を指定した場合は big endian になります。オプション指定が endian=little である場合は、セクション名の後に_B をつけたセクションに配置されます。

little を指定した場合は little endian になります。オプション指定が endian=big である場合は、セクション名の後に_L をつけたセクションに配置されます。

big | little を省略すると、以降はオプションに従います。

例　endian=little オプション指定時(デフォルト)

C ソース：

```
#pragma endian big
int A=100;   /* D_B セクション       */
#pragma endian
int B=200;   /* D セクション  */
```

出力コード：

```
        .SECTION        D_B,ROMDATA,ALIGN=4
        .ENDIAN         BIG
        .glb            A
_A:                                 ; static: A
        .LWORD          00000064H
        .SECTION        D,ROMDATA,ALIGN=4
        .glb    _B
_B:                                 ; static: B
        .LWORD          000000C8H
```

備　考　endian オプションと異なる#pragma endian 対象のオブジェクトに、long long 型、double 型(dbl_size=8 オプション指定時)および long double 型(同)の領域を含む場合は、これらの領域に対するアドレスやポインタを用いた間接的なアクセスはしないでください。この場合の動作は保証されません。
もし、このような領域のアドレスを取得するコードを記述した場合は、警告を表示します。

endian オプションと異なる#pragma endian 対象のオブジェクトに、long long 型のビットフィールドを含む場合は、この領域に書き込みを行わないでください。この場合の動作は保証されません。
もし、このような領域への書き込みを行うコードを記述した場合は、警告を表示します。

次の項目のエンディアンは変更できません。endian オプションを使用してください。

(1) 文字列リテラル

(2) switch 文の分岐テーブル

(3) 外部参照宣言されたオブジェクト（初期化式なく extern 宣言されたオブジェクト）

指定サイズのアクセスを保証

__evenaccess

書　式　__evenaccess <型指定子> <変数名>

<型指定子> __evenaccess <変数名>

説　明　変数の型のサイズでアクセスすることを保証します。

保証対象サイズは、4 バイト以下の整数スカラ型(signed char, unsigned char, signed short, unsigned short, signed int, unsigned int, signed long, unsigned long)です。

例　C ソース：

```
#pragma address A=0xff0178
unsigned long __evenaccess A;
void test(void)
{
    A &= ~0x20;
}
```

出力コード（__evenaccess 非指定時）：

```
_test:
 MOV.L #16712056,R1
 BCLR #5,[R1]              ; 1 バイトメモリアクセス
 RTS
```

出力コード（__evenaccess 指定時）：

```
_test:
 MOV.L #16712056,R1
 MOV.L [R1],R5             ; 4 バイトメモリアクセス
 BCLR  #5,R5
 MOV.L R5,[ R1]            ; 4 バイトメモリアクセス
 RTS
```

備　考　構造体または共用体に指定した場合、全てのメンバに__evenaccess を指定したのと同じ効果になります。その場合、4 バイト以下の整数スカラ型メンバのアクセスサイズは保証しますが、構造体または共用体単位でのアクセスサイズは保証しません。

分岐先の命令実行向け整合を行う関数の指定

#pragma instalign4
#pragma instalign8
#pragma noinstalign

書　式　#pragma instalign4 [(]<関数名>[(＜分岐先の種類＞)][,…][)]

#pragma instalign8 [(]<関数名>[(＜分岐先の種類＞)][,…][)]

#pragma noinstalign [(]<関数名>[,…][)]

説　明　分岐先の命令実行向け整合を行う関数を指定します。

指定された関数に対し、#pragma instalign4 の場合は 4 バイト、#pragma instalign8 の場合は 8 バイトでそれぞれ配置アドレスを命令実行向け整合します。
#pragma noinstalign が指定された関数は、整合は行いません。

分岐先の種類は、以下から選択します(*1)。

指定なし:　関数先頭、switch 文の case および default ラベル

inmostloop:　各最内周ループの先頭、関数先頭、switch 文の case および default ラベル

loop:　各ループの先頭、関数先頭、switch 文の case および default ラベル

[注]

*1) ここに挙げたもの以外の分岐先は、命令実行向け整合の対象ではありません。たとえば、loop を選択した場合はループの先頭は対象ですが、ループ内にあるループを構成しない if 文などの分岐先は対象ではありません。

それぞれ、指定した関数ごとに有効になることを除き、instalign4, instalign8, noinstalign オプションと機能は同じです。これらのオプションと同時に指定した場合は、#pragma の指定が優先されます。

instalign4 または instalign8 を選択した関数が属するコードセクションは、そのアライメント数は 4(instalign4 の場合)または 8(instalign8 の場合)に変わります。同じコードセクション内に、instalign4 と instalign8 が指定された関数が混在する場合、そのコードセクションのアライメント数は 8 になります。

その他の内容については、instalign4, instalign8, noinstalign オプションと同様ですので、これらのオプションの項目を参照ください。

9.2.2 組み込み関数

以下の機能を、組み込み関数として提供します。

- 最大値、最小値
- データ内バイト入れ替え
- データ交換
- 積和演算
- 回転
- 特殊命令(BRK, WAIT, INT, NOP)
- BRK、WAITなどのRXファミリ用特殊命令
- 制御レジスタ設定、参照

組み込み関数は、通常の関数と同様に関数呼び出し形式で記述します。
組み込み関数の一覧を**表** 9.24 に示します。

表 9.24 **組み込み関数の一覧**

	項目	仕様	機能	ユーザモードの制限*1
1	最大値・最小値	signed long max(signed long data1, signed long data2)	最大値の選択	○
2		signed long min(signed long data1, signed long data2)	最小値の選択	○
3	バイト並べ替え	unsigned long revl(unsigned long data)	ロングワードデータをバイトリバース	○
4		unsigned long revw(unsigned long data)	ロングワードデータをワード毎にバイトリバース	○
5	データ交換	void xchg(signed long *data1, signed long *data2)	データ交換	○
6	積和演算	long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *add2)	積和演算(バイト)	○
7		long long rmpaw(long long init, unsigned long count, short *addr1, short *add2)	積和演算(ワード)	○
8		long long rmpal(long long init, unsigned long count, long *addr1, long *add2)	積和演算(ロングワード)	○
9	回転	unsigned long rolc(unsigned long data)	キャリーを含めて1ビット左回転	○
10		unsigned long rorc(unsigned long data)	キャリーを含めて1ビット右回転	○
11		unsigned long rotl(unsigned long data, unsigned long num)	左回転	○

	項目	仕様	機能	ユーザモードの制限*1
12		unsigned long rotr (unsigned long data, unsigned long num)	右回転	○
13	特殊命令	void brk(void)	BRK 命令例外	○
14		void int_exception(signed long num)	INT 命令例外	○
15		void wait(void)	プログラム実行停止	×
16		void nop(void)	NOP 命令に展開	○
17	プロセッサ割り込み優先レベル (IPL)	void set_ipl(signed long level)	割り込み優先レベルの設定	×
18		unsigned char get_ipl(void)	割り込み優先レベルの参照	○
19	プロセッサステータスワード(PSW)	void set_psw(unsigned long data)	PSW の設定	△
20		unsigned long get_psw(void)	PSW の参照	○
21	浮動小数点ステータスワード (FPSW)	void set_fpsw(unsigned long data)	FPSW の設定	○
22		unsigned long get_fpsw(void)	FPSW の参照	○
23	ユーザスタックポインタ(USP)	void set_usp(void *data)	USP の設定	○
24		void *get_usp(void)	USP の参照	○
25	割り込みスタックポインタ(ISP)	void set_isp(void *data)	ISP の設定	△
26		void *get_isp(void)	ISP の参照	○
27	割り込みテーブルレジスタ(INTB)	void set_intb (void *data)	INTB の設定	△
28		void *get_intb(void)	INTB の参照	○
29	バックアップ PSW(BPSW)	void set_bpsw(unsigned long data)	BPSW の設定	△
30		unsigned long get_bpsw(void)	BPSW の参照	○
31	バックアップ PC(BPC)	void set_bpc(void *data)	BPC の設定	△
32		void *get_bpc(void)	BPC の参照	○
33	高速割り込みベクタレジスタ (FINTV)	void set_fintv(void *data)	FINTV の設定	△
34		void *get_fintv(void)	FINTV の参照	○
35	有効桁 64bit の乗算	signed long long emul(signed long data1, signed long data2);	有効桁 64bit の符号付き乗算	○
36		unsigned long long emulu(unsigned long data1, unsigned long data2)	有効桁 64bit の符号なし乗算	○
37	プロセッサモード (PM)	void chg_pmusr(void)	ユーザモードへの切り換え	△
38	アキュムレータ (ACC)	void set_acc(signed long long data)	ACC の設定	○
39		signed long long get_acc(void)	ACC の参照	○

	項目	仕様	機能	ユーザモードの制限*1
40	割り込み許可ビットの制御	void setpsw_i(void)	割り込み許可ビットを 1 に設定	△
41		void clrpsw_i(void)	割り込み許可ビットを 0 に設定	△
42	積和演算	long macl(short *data1, short *data2, unsigned long count)	2byte データの積和演算	○
43		short macw1(short *data1, short *data2, unsigned long count) short macw2(short *data1, short *data2, unsigned long count)	固定小数点データ向けの積和演算	○

【注】 *1 RX のプロセッサモードがユーザモードの場合に制限があるものを示します。
×印の関数は特権命令例外が発生するため、ユーザモードでは使用しないでください。
△印の関数はユーザモードで実行しても効果はありません。

最大値の選択

signed long max(signed long data1, signed long data2)

説　明　　２つの入力値のうち大きい方を選択します（MAX 命令に展開します）。

ヘッダ　　<machine.h>

引　数　　data1　　入力値 1
　　data2　　入力値 2

リターン値　　data1 と data2 のうち大きい方の値

例

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
        ret = max(in1,in2);      // in1 と in2 のうち大きい方の値を ret に設定します。
}
```

最小値の選択

signed long min(signed long data1, signed long data2)

説　明　　２つの入力値のうち小さい方を選択します（MIN 命令に展開します）。

ヘッダ　　<machine.h>

引　数　　data1　　入力値 1
　　data2　　入力値 2

リターン値　　data1 と data2 のうち小さい方の値

例

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
        ret = min(in1,in2);          // in1 と in2 のうち小さい方の値を ret に設定します。
}
```

ロングワードデータをバイトリバース

unsigned long revl(unsigned long data)

説　明　4 バイトデータのバイト並び順をリバースします（REVL 命令に展開します）。

ヘッダ　<machine.h>

引　数　data　バイト並びをリバースするデータ

リターン値　data のバイト並びをリバースした値

例

```
#include <machine.h>
extern unsigned long ret,indata=0x12345678;
void main(void)
{
        ret = revl(indata);  // ret=0x78563412 となる
}
```

ロングワードデータをワード毎にバイトリバース

unsigned long revw(unsigned long data)

説　明　4 バイトデータの上位 2 バイトと下位 2 バイトでそれぞれのバイト並びをリバースします（REVW 命令に展開します）。

ヘッダ　<machine.h>

引　数　data　バイト並びをリバースするデータ

リターン値　data の上位 2 バイトと下位 2 バイトでそれぞれのバイト並びをリバースした値

例

```
#include <machine.h>
extern unsigned long ret;indata=0x12345678;
void main(void)
{
        ret = revw(indata);// ret=0x34127856  となる
}
```

*void xchg(signed long *data1, signed long *data2)*

説　明　　引数が指す領域の内容を入れ替えます（XCHG 命令に展開します）。

ヘッダ　　<machine.h>

引　数　　*data1　　入力値１
*data2　　入力値２

例

```
#include <machine.h>
extern signed long *in1,*in2;
void main(void)
{
    xchg (in1,in2);        // アドレス in1、アドレス in2 のデータを交換します。

}
```

積和演算(バイト)

*long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *add2)*

説　明　初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.B 命令に展開します）。

ヘッダ　<machine.h>

引　数

init	初期値
count	積和演算の回数
*addr1	乗数 1 の先頭アドレス
*addr2	乗数 2 の先頭アドレス

リターン値　init + Σ(data1[n] * data2[n]) の下位 64 ビットの結果　(n=0, 1, …, const-1)

例

```
#include <machine.h>
extern signed char data1[8],data2[8];
long long sum;
void main(void)
{
        sum=rmpab(0, 8, data1, data2);          // 0 を初期値とし、配列 data1,data2 の
                                                // 乗算結果を加算して sum に設定する
}
```

備考　RMPA 命令は 80bit の範囲で結果を計算しますが、本組み込み関数では 64bit 範囲のみ扱います。

積和演算(ワード)

*long long rmpaw(long long init, unsigned long count, short *addr1, short *add2)*

説　明　初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.W 命令に展開します）。

ヘッダ　<machine.h>

引　数

init	初期値
count	積和演算の回数
*addr1	乗数 1 の先頭アドレス
*addr2	乗数 2 の先頭アドレス

リターン値　init + Σ(data1[n] * data2[n]) の下位 64 ビットの結果　(n=0, 1, …, const-1)

例

```
#include <machine.h>
extern signed short data1[8],data2[8];
long long sum;
void main(void)
{
        sum=rmpaw(0, 8, data1, data2);          // 0 を初期値とし、配列 data1,data2 の
                                                // 乗算結果を加算して sum に設定する
}
```

備　考　RMPA 命令は 80bit の範囲で結果を計算しますが、本組み込み関数では 64bit 範囲のみ扱います。

積和演算(ロングワード)

*long long rmpal(long long init, unsigned long count, long *addr1, long *add2)*

説　明　初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.L 命令に展開します）。

ヘッダ　<machine.h>

引　数

init	初期値
count	積和演算の回数
*addr1	乗数 1 の先頭アドレス
*addr2	乗数 2 の先頭アドレス

リターン値　init + Σ(data1[n] * data2[n]) の下位 64 ビットの結果　(n=0, 1, …, const-1)

例

```
#include <machine.h>
extern signed long data1[8],data2[8];
long long sum;
void main(void)
{
    sum=rmpaw(0, 8, data1, data2);          // 0 を初期値とし、配列 data1,data2 の
                                            // 乗算結果を加算して sum に設定する
}
```

キャリーを含めて1 ビット左回転

unsigned long rolc(unsigned long data)

説　明　C フラグを含めて 1 ビット左回転した結果を返します（ROLC 命令に展開します）。
オペランドの外へ出たビットを C フラグに反映します。

ヘッダ　<machine.h>

引　数　data　左回転するデータ

リターン値　data を C フラグを含めて左に 1 ビット回転した結果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rolc(indata);             // indata を C フラグを含めて 1 ビット左回転し
                                        // ret に設定します。
}
```

キャリーを含めて1 ビット右回転

unsigned long rorc(unsigned long data)

説　明　C フラグを含めて 1 ビット右回転した結果を返します（RORC 命令に展開します）。
オペランドの外へ出たビットを C フラグに反映します。

ヘッダ　<machine.h>

引　数　data　右回転するデータ

リターン値　data を C フラグを含めて右に 1 ビット回転した結果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rorc(indata);             // indata を C フラグを含めて 1 ビット右回転し
                                        // ret に設定します。
}
```

左回転

unsigned long rotl(unsigned long data, unsigned long num)

説　明　任意ビット左回転した結果を返します（ROTL 命令に展開します）。
オペランドの外へ出たビットを C フラグに反映します。

ヘッダ　<machine.h>

引　数　data　左回転するデータ
num　回転するビット数

リターン値　data を左に num ビット回転した結果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rotl(indata, 31);          // indata を 31 ビット左回転し
                                         // ret に設定します。
}
```

右回転

unsigned long rotr (unsigned long data, unsigned long num)

説　明　任意ビット右回転した結果を返します（ROTR 命令に展開します）。
オペランドの外へ出たビットを C フラグに反映します。

ヘッダ　<machine.h>

引　数　data　右回転するデータ
num　回転するビット数

リターン値　data を右に num ビット回転した結果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rotr(indata, 31);      // indata を 31 ビット右回転し
                                     // ret に設定します。
}
```

BRK 命令例外

void brk(void)

説　明　　BRK 命令に展開します。

ヘッダ　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        brk();                  // BRK 命令
}
```

INT 命令例外

void int_exception(signed long num)

説　明　　INT　num　命令に展開します。

ヘッダ　　<machine.h>

引　数　　num　　INT 命令番号

例

```
#include <machine.h>
void main(void)
{
    int_exception(10);          // INT #10  命令
}
```

備　考　　num に設定できる数は、0～255 の整数のみです。

プログラム実行停止

void wait(void)

説　明　　WAIT 命令に展開します。

ヘッダ　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        wait();                         // WAIT 命令
}
```

備　考　　本関数は RX のプロセッサモードがユーザモードの場合は実行しないでください。実行すると、WAIT 命令の仕様により、RX の特権命令例外が発生します。

NOP 命令に展開

void nop(void)

説　明　　NOP 命令に展開します。

ヘッダ　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        nop();                          // NOP 命令
}
```

割り込み優先レベルの設定

void set_ipl(signed long level)

説　明　　割り込みマスクレベルを変更します。

ヘッダ　　<machine.h>

リターン値　　level　　設定する割り込みマスクレベル

例

```
#include <machine.h>
void main(void)
{
        set_ipl(7);                     // PSW.IPL  に 7 を設定
}
```

備　考　　デフォルトでは level には 0～15 の値が、-patch=rx610 を指定した場合は 0～7 の値がそれぞれ指定できます。
level が定数のとき、範囲外の値を指定した場合はエラーとなります。
本関数は RX のプロセッサモードがユーザモードの場合は使用しないでください。実行すると、MVTIPL 命令の仕様により、RX の特権命令例外が発生します。

割り込み優先レベルの参照

unsigned char get_ipl(void)

説　明　　割り込みマスクレベルを参照します。

ヘッダ　　<machine.h>

リターン値　　割り込みマスクレベル

例

```
#include <machine.h>
extern unsigned char level;
void main(void)
{
        level=get_ipl();                // PSW.IPL の値を取得し level に設定
}
```

PSW の設定

void set_psw(unsigned long data)

説　明　PSW を設定します。

ヘッダ　<machine.h>

引　数　data　設定値

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_psw(data);                  // PSW に data の値を設定
}
```

備　考　RX の命令セット仕様のため、PSW の PM ビットの書き込みは無視されます。また、RX のプロセッサモードがユーザモードの場合は、PSW への書き込みは無視されます。

PSW の参照

unsigned long get_psw(void)

説　明　PSW を参照します。

ヘッダ　<machine.h>

リターン値　PSW の値

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_psw();                  // PSW の値を取得し、ret に設定
}
```

備　考　最適化の作用により、get_psw の呼び出し箇所とは違うタイミングで PSW レジスタの値が取得される場合があります。このため、何らかの演算後に、本関数の戻り値に含まれる C,Z,S および O フラグのいずれかを利用するコードを記述した場合、その動作は保証されません。

FPSW の設定

void set_fpsw(unsigned long data)

説　明　　FPSW を設定します。

ヘッダ　　<machine.h>

引　数　　data　　設定値

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_fpsw(data);                 // FPSW に data の値を設定
}
```

FPSW の参照

unsigned long get_fpsw(void)

説　明　　FPSW を参照します。

ヘッダ　　<machine.h>

リターン値　　FPSW の値

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_fpsw();                 // FPSW の値を取得し、ret に設定
}
```

備　考　　最適化の作用により、get_fpsw の呼び出し箇所とは違うタイミングで FPSW レジスタの値が取得される場合があります。このため、何らかの演算後に、本関数の戻り値に含まれる CV,CO,CZ,CU,CX,CE,FV,FO,FZ,FU,FX および FS フラグのいずれかを利用するコードを記述した場合、その動作は保証されません。

USP の設定

*void set_usp(void *data)*

説　明　　USP を設定します。

ヘッダ　　<machine.h>

引　数　　data　　設定値

例

```
#include <machine.h>
extern void * data;
void main(void)
{
        set_usp(data);                  // USP に data の値を設定
}
```

USP の参照

*void *get_usp(void)*

説　明　　USP を参照します。

ヘッダ　　<machine.h>

リターン値　　USP の値

例

```
#include <machine.h>
extern void * ret;
void main(void)
{
        ret=get_usp();                  // USP の値を取得し、ret に設定
}
```

ISP の設定

*void set_isp(void *data)*

説　明　　ISP を設定します。

ヘッダ　　<machine.h>

引　数　　data　　設定値

例

```
#include <machine.h>
extern void * data;
void main(void)
{
        set_isp(data);                  // ISP に data の値を設定
}
```

備　考　　本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、ISP への書き込みは無視されます。

ISP の参照

*void *get_isp(void)*

説　明　　ISP を参照します。

ヘッダ　　<machine.h>

リターン値　　ISP の値

例

```
#include <machine.h>
extern void * ret;
void main(void)
{
        ret=get_isp();                  // ISP の値を取得し、ret に設定
}
```

INTB の設定

*void set_intb (void *data)*

説　明　INTB を設定します。

ヘッダ　<machine.h>

引　数　data　設定値

例

```
#include <machine.h>
extern void * data;
void main(void)
{
        set_intb (data);                // INTB に data の値を設定
}
```

備　考　本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、INTB への書き込みは無視されます。

INTB の参照

*void *get_intb(void)*

説　明　INTB を参照します。

ヘッダ　<machine.h>

リターン値　INTB の値

例

```
#include <machine.h>
extern void * ret;
void main(void)
{
        ret=get_intb();                 // INTB の値を取得し、ret に設定
}
```

BPSW の設定

void set_bpsw(unsigned long data)

説　明　　BPSW を設定します。

ヘッダ　　<machine.h>

引　数　　data　　設定値

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_bpsw (data);                // BPSW に data の値を設定
}
```

備　考　　本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、BPSW への書き込みは無視されます。

BPSW の参照

unsigned long get_bpsw(void)

説　明　　BPSW を参照します。

ヘッダ　　<machine.h>

リターン値　　BPSW の値

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_bpsw ();                // BPSW の値を取得し、ret に設定
}
```

BPC の設定

*void set_bpc(void *data)*

説　明　BPC を設定します。

ヘッダ　<machine.h>

引　数　data　設定値

例

```
#include <machine.h>
extern void * data;
void main(void)
{
        set_bpc(data);                  // BPC に data の値を設定
}
```

備　考　本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、BPC への書き込みは無視されます。

BPC の参照

*void *get_bpc(void)*

説　明　BPC を参照します。

ヘッダ　<machine.h>

リターン値　BPC の値

例

```
#include <machine.h>
extern void * ret;
void main(void)
{
        ret=get_bpc();                  // BPC の値を取得し、ret に設定
}
```

FINTV の設定

*void set_fintv(void *data)*

説　明　　FINTV を設定します。

ヘッダ　　<machine.h>

引　数　　data　　設定値

例

```
#include <machine.h>
extern void * data;
void main(void)
{
        set_fintv(data);                    // FINTV に data の値を設定
}
```

備　考　　本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、FINTV への書き込みは無視されます。

FINTV の参照

*void *get_fintv(void)*

説　明　　FINTV を参照します。

ヘッダ　　<machine.h>

リターン値　　FINTV の値

例

```
#include <machine.h>
extern void * ret;
void main(void)
{
        ret=get_fintv();                    // FINTV の値を取得し、ret に設定
}
```

有効桁 64bit の符号付き乗算

signed long long emul(signed long data1, signed long data2)

説　明　　有効桁 64bit の符号付き乗算を行います。

ヘッダ　　<machine.h>

リターン値　　符号付き乗算の結果（64bit 符号付き）

例

```
#include <machine.h>
extern signed long long ret;
extern signed long data1, data2;
void main(void)
{
        ret=emul(data1, data2);                 // data1 * data2  の値を計算し、ret に設定
}
```

有効桁 64bit の符号なし乗算

unsigned long long emulu(unsigned long data1, unsigned long data2)

説　明　　有効桁 64bit の符号なし乗算を行います。

ヘッダ　　<machine.h>

リターン値　　符号なし乗算の結果（64bit 符号なし）

例

```
#include <machine.h>
extern unsigned long long ret;
extern unsigned long data1, data2;
void main(void)
{
        ret=emulu(data1, data2);                // data1 * data2  の値を計算し、ret に設定
}
```

ユーザモードへの切り換え

void chg_pmusr(void)

説　明　RX のプロセッサモードをユーザに切り換えます。

ヘッダ　<machine.h>

例

```
#include <machine.h>
void main(void)
void Do_Main_on_UserMode(void)
{
        chg_pmusr();                    // プロセッサモードをユーザモードに切り換え
        main();         // main を実行
}
```

備　考　本関数はリセット処理関数あるいは割り込み関数のために用意されています。それ以外の関数での使用は推奨しません。

RX のプロセッサモードがユーザモードのときは、プロセッサモードは切り替わりません。

chg_pmusr 関数の実行により、スタックは割り込みスタックからユーザスタックに切り換えが起こりますので、本関数を呼び出す関数では、必ず次の条件を守ってください。守られない場合、本関数の実行前後のスタックの相違が起こるため、コードは正常に動作しません。

・呼び出し元に return することはできません。
・auto 変数を宣言することはできません。
・引数を宣言することはできません。

ACC の設定

void set_acc(signed long long data)

説　明　　ACC を設定します。

ヘッダ　　<machine.h>

引　数　　data　　ACC への設定値

例

```
#include <machine.h>
void main(void)
{
        signed long long data = 0x123456789ab0000LL;
        set_acc(data);                  // ACC に data の値を設定
}
```

ACC の参照

signed long long get_acc(void)

説　明　　ACC を参照します。

ヘッダ　　<machine.h>

リターン値　　ACC の値

例

```
/* get_acc と set_acc による ACC の退避/回復を利用したプログラムの例 */
#include <machine.h>
signed long func(signed long a, signed long b)
{
        signed long long bak_acc = get_acc();    // ACC の値を取得し、bak_acc に退避
        a *= b;                                  //  乗算(ACC を破壊する)
        set_acc(bak_acc);                        // bak_acc で退避していた値で ACC を回復
        return a;
}
```

備　考　　RX 命令セットの仕様のため、ACC の下位 16 ビットは取得できません。本関数は、これらのビットには 0 を返します。

割り込み許可ビットを 1 に設定

void setpsw_i(void)

説　明　PSW の割り込み許可ビット(I ビット)を 1 に設定します。

ヘッダ　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        setpsw_i();                     // 割り込み許可ビットを 1 にする
}
```

備　考　本関数で使用する SETPSW 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、割り込み許可ビットへの書き込みは無視されます。

割り込み許可ビットを 0 に設定

void clrpsw_i(void)

説　明　PSW の割り込み許可ビット(I ビット)を 0 に設定します。

ヘッダ　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        clrpsw_i();                     // 割り込み許可ビットを 0 にする
}
```

備　考　本関数で使用する CLRPSW 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、割り込み許可ビットへの書き込みは無視されます。

*long macl(short *data1, short *data2, unsigned long count)*

説　明　積和演算を行います。

2byte のデータ同士の積和演算を行い、その結果を 4byte で返します。

積和演算は、DSP 機能命令(MULLO,MACLO,MACHI)を使用して演算します。

積和演算の途中のデータは、ACC に 48bit 長データとして保持されます。

全ての積和演算が終わったら、ACC の内容を MVFACHI 命令で取り出し、組み込み関数の戻り値とします。

本組み込み関数を使用することにより、組み込み関数を使用せずに積和演算を記述する場合よりも、高速な積和演算処理が期待できます。

2byte の整数データの積和演算をする場合に利用できます。積和演算の演算結果には、飽和処理や丸め処理はされません。

ヘッダ　<machine.h>

引　数

data1	乗数 1 の先頭アドレス
data2	乗数 2 の先頭アドレス
count	積和演算の乗算回数

リターン値　Σ(data1[n]×data2[n])の演算結果

例

```
#include <machine.h>
short data1[3] = {a1, b1, c1};
short data2[3] = {a2, b2, c2};
void mac_calc()
{
        result = macl(data1, data2, 3);       /* a1*a2+b1*b2+c1*c2  の結果を求めます  */
}
```

備　考　積和演算に使用される各種 DSP 機能命令の詳細な内容を確認するには、プログラミングマニュアルを参照してください。

乗算回数が 0 の場合、組み込み関数の戻り値は 0 です。

本組み込み関数を使用する場合、ACC の内容が書き換わる割り込み処理では、ACC を退避回復してください。

ACC を退避回復する機能については、コンパイルオプションの save_acc、もしくは、拡張言語仕様の#pragma interrupt を参照してください。

*short macw1(short *data1, short *data2, unsigned long count)*
*short macw2(short *data1, short *data2, unsigned long count)*

説　明　2byte のデータ同士の積和演算を行い、その結果を 2byte で返します。

積和演算は、DSP 機能命令(MULLO,MACLO,MACHI)を使用して演算します。

積和演算の途中のデータは、ACC に 48bit 長データとして保持されます。

全ての積和演算が終わった後に、ACC の積和演算結果に対して丸め処理を行います。

macw1 関数は"RACW #1"命令、macw2 関数は"RACW #2"命令で丸め処理を行います。

丸め処理の処理内容は、以下の手順になります。

・ACC の内容を、macw1 関数は 1 ビット、macw2 関数は 2 ビット、左シフトします

・ACC の下位 32 ビットの最上位ビットを 0 捨 1 入します

・ACC の上位 32 ビットを、上限値 0x00007FFF、下限値を 0xFFFF8000 として飽和処理します

最後に、MVFACHI 命令で ACC から上位 32 ビットを取得し、組み込み関数の戻り値とします。

通常、固定小数点データ同士の乗算をする場合、乗算結果の小数点位置を調整する必要があります。たとえば、Q15 形式の固定小数点データ同士の乗算の場合、乗算結果を同じ Q15 形式にするためには、乗算結果を 1 ビット左シフトする必要があります。

この小数点位置を調整するための左シフトは、RACW 命令の左シフト動作によって実現できます。そのため、2byte の固定小数点データの積和演算をする場合に、本組み込み関数を利用することにより、容易に積和演算処理を実現できます。なお、macw1 と macw2 は演算結果の丸め方法が異なりますので、演算結果に求められる精度に応じて、使用する組み込み関数を選択してください。

ヘッダ　<machine.h>

引　数

	data1	乗数 1 の先頭アドレス
	data2	乗数 2 の先頭アドレス
	count	積和演算の乗算回数

リターン値　積和演算の演算結果を、RACW 命令で丸めた値

例

```
#include <machine.h>
short data1[3] = {a1, b1, c1};
short data2[3] = {a2, b2, c2};
void mac_calc()
{
        result = macw1(data1, data2, 3);
                /* a1*a2+b1*b2+c1*c2  の結果を、"RACW #1"命令で丸めた値を求めます。  */
}
```

備　考　積和演算に使用される各種 DSP 機能命令の詳細な内容を確認するには、プログラミングマニュアルを参照してください。

乗算回数が 0 の場合、組み込み関数の戻り値は 0 です。

本組み込み関数を使用する場合、ACC の内容が書き換わる割り込み関数では、 ACC を退避回復してください。

ACC を退避回復する機能については、コンパイルオプションの save_acc、もしくは、拡張言語仕様の#pragma interrupt を参照してください。

9.2.3 セクションアドレス演算子

セクションアドレス演算子の一覧を**表** 9.25 に示します。

表 9.25 **セクション演算子一覧**

	セクション演算子名	説明
1	__sectop("<**セクション名**>")	__sectop **で指定した<セクション名>の先頭アドレスを参照**
2	__secend("<**セクション名**>")	__secend **で指定した<セクション名>の末尾+1 アドレスを参照**
3	__secsize("<**セクション名**>")	__secsize **で指定した<セクション名>のサイズを生成**

セクションアドレス演算子

__sectop, __secend, __secsize

書　式　　__sectop("<セクション名>")
__secend("<セクション名>")
__secsize("<セクション名>")

説　明　　__sectop で指定した<セクション名>の先頭アドレスを参照します。
__secend で指定した<セクション名>の末尾+1 アドレスを参照します。
__secsize で指定した<セクション名>のサイズを生成します。

リターン値型　__sectop のリターン値型は、void * です。
__secend のリターン値型は、void * です。
__secsize のリターン値型は、unsigned long です。

例　　(1) __sectop, __secend

```
#include <machine.h>
#pragma section $DSEC
static const struct {
    void *rom_s; /*  初期化データセクションの ROM 上の先頭アドレス  */
    void *rom_e; /*  初期化データセクションの ROM 上の最終アドレス  */
    void *ram_s; /*  初期化データセクションの RAM 上の先頭アドレス  */
} DTBL[]={__sectop("D"), __secend("D"), __sectop("R")};

#pragma section $BSEC
static const struct {
    void *b_s; /*  未初期化データセクションの先頭アドレス  */
    void *b_e; /*  未初期化データセクションの最終アドレス  */
} BTBL[]={__sectop("B"), __secend("B")};
```

```
#pragma section
#pragma stacksize si=0x100
#pragma entry INIT
void main(void);
void INIT(void)
{
      _INITSCT();
      main();
      sleep();
}
```

(2) __secsize

```
/* size of section B */
unsigned int size_of_B = _ _secsize("B");
```

備　考　PIC/PID 機能が有効なアプリケーションの場合、__sectop および__secend はリンク時のアドレスで処理します。

PIC/PID 機能の詳細は、「2.5 マイコンオプション」の pic および pid オプションと、「8.4 PIC/PID 機能の利用」の項目をそれぞれ参照してください。

9.3 C/C++ライブラリ

9.3.1 標準Cライブラリ

(1) ライブラリの概要

C/C++言語の中で標準的に利用できる C ライブラリ関数の仕様について説明します。ここでは、ライブラリの構成を概説し、本節の読み方および用語について説明します。以降ではライブラリの構成に従って各ライブラリ関数の仕様を説明します。

【注】 本節で現れる double **型および** long double **型は、いずれもコンパイラの** dbl_size=8 **オプションを指定したものとして説明しています。**

(a) ライブラリの種類

ライブラリとは、入出力、文字列操作等の標準的な処理を C/C++言語の関数の形式で実現したものです。また、これらのライブラリは、各処理単位に対応した標準インクルードファイルを取り込むことによって使用可能となります。

標準インクルードファイルには、対応するライブラリの宣言とそれらを使用するために必要なマクロ名が定義されています。

表 9.26 にライブラリの種類と対応する標準インクルードファイルを示します。

表 9.26 **ライブラリの種類と対応する標準インクルードファイル**

	ライブラリの種類	内容	標準インクルードファイル
1	プログラム診断用ライブラリ	プログラムの診断情報の出力を行うライブラリです。	<assert.h>
2	文字操作用ライブラリ	文字の操作およびチェックを行うライブラリです。	<ctype.h>
3	数値計算用ライブラリ	三角関数等の数値計算を行うライブラリです。	<math.h> <mathf.h>
4	プログラムの制御移動用ライブラリ	関数間の制御の移動をサポートするライブラリです。	<setjmp.h>
5	可変個の実引数アクセス用ライブラリ	可変個の実引数を持つ関数に対し、その実引数へのアクセスをサポートするライブラリです。	<stdarg.h>
6	入出力用ライブラリ	入出力操作を行うライブラリです。	<stdio.h>
7	標準処理用ライブラリ	記憶域管理等の C プログラムでの標準的処理を行うライブラリです。	<stdlib.h>
8	文字列操作用ライブラリ	文字列の比較、複写等を行うライブラリです。	<string.h>
9	複素数計算ライブラリ	複素数の計算を行うライブラリです。	<complex.h>
10	浮動小数点環境ライブラリ	浮動小数点環境のライブラリです。	<fenv.h>
11	整数型の書式変換	最大幅の整数の操作、変換を行うライブラリです。	<inttypes.h>
12	多バイト文字、ワイド文字ライブラリ	多バイト文字の操作を行うライブラリです。	<wchar.h> <wctype.h>

また、以上の標準インクルードファイルの他にプログラムの作成作業の効率向上を図るため**表** 9.27 に示すマクロ名の定義だけからなる標準インクルードファイルがあります。

表 9.27 **マクロ名定義からなる標準インクルードファイル**

	標準インクルードファイル	内容
1	<stddef.h>	**各標準インクルードファイルで共通に使用するマクロ名を定義します。**
2	<limits.h>	**コンパイラの内部処理に関する各種制限値を定義します。**
3	<errno.h>	**ライブラリ関数においてエラーが発生した時に** errno **に設定する値を定義します。**
4	<float.h>	**浮動小数点型の限界に関する各種制限値を定義します。**
5	<iso646.h>	**代替つづりのマクロ名を定義します。**
6	<stdbool.h>	**論理型、および論理値に関するマクロを定義します。**
7	<stdint.h>	**指定した幅の整数型を宣言してマクロを定義します。**
8	<tgmath.h>	**型総称マクロを定義します。**

(b) ライブラリの説明形式

ライブラリの各関数を標準インクルードファイルごとに分類し、その標準インクルードファイルごとに説明します。その各分類は、まず、標準インクルードファイルの中で定義されているマクロ名や関数宣言に対する説明を行い(**図** 9.2 **参照**)、続いて各関数の説明を行う(**図** 9.3 **参照**)という形式で構成されています。

図 9.2 に標準インクルードファイルの説明形式、**図** 9.3 に関数の説明形式を示します。

項番 ＜標準インクルードファイル名＞

- 本インクルードファイルがもつ全体的な機能の概要を説明します。
- 本インクルードファイル内で定義・宣言される名前を名前種別(【型】、【定数】、【変数】、【関数】)に分類して説明します。マクロである場合、名前種別のタイトル(【】内)または名前の説明箇所に(マクロ)と表記しています。
- 処理系定義仕様がある場合や、本インクルードファイル内で宣言されている関数に共通する注意事項がある場合、説明を補足します。

図 9.2 **標準インクルードファイルの説明形式**

機能の概要を示します。

ライブラリ関数の型(リターン値および引数)を示します。

説　明	ライブラリ関数の機能を説明します。
ヘッダ	宣言元の標準インクルードファイル名です。
リターン値	正常：ライブラリ関数が正常終了したときの値です。 異常：ライブラリ関数が異常終了したときの値です。
引　数	引数の意味を説明します。
例	呼び出し手順を説明します。
エラー条件	ライブラリ関数の処理でリターン値からでは、判断できないエラーが発生する条件を示します。このようなエラーが発生したとき、エラーの種類に対応する、コンパイラごとに定義された値がerrno* に設定されます。
備　考	補足説明、または使用上の注意事項です。
処理系定義仕様	本コンパイラの処理方法です。

図 9.3　関数の説明形式

【注】 *errno **は、ライブラリ関数実行中に生じたエラーの種類を格納する変数です。詳細については「**9.3.1(2) <stddef.h>**」を参照してください。**

(c) ライブラリ関数の説明で使用する用語

(a) ストリーム入出力

データの入出力において、1文字ごとの入出力関数の呼び出しのたびに入出力装置を駆動したり、OSの機能を呼び出していたのでは、効率が悪くなります。そこで、通常はバッファと呼ばれる記憶域を用意しておき、バッファ内のデータに対して一括して入出力を行います。

一方、プログラムの側から見ると、1文字ごとに入出力関数を呼び出せた方が便利です。

ライブラリ関数では、バッファの管理を自動的に行うことにより、プログラム内でバッファの状態を意識することなしに、1文字単位の入出力を効率よく行うことができます。

このように、データの入出力を効率よく実現するために詳細な手段を意識せず、入出力をひとつのデータの流れ(ストリーム)と考えてプログラムを作ることのできる機能をストリーム入出力といいます。

(b) FILE構造体およびファイルポインタ

ストリーム入出力に必要なバッファやその他の情報は、一つの構造体の中に記憶されており、標準インクルードファイル<stdio.h>の中でFILEという名前で定義されています。

ストリーム入出力においては、ファイルはすべてFILE構造体のデータ構造を持つものとして扱います。このようなファイルをストリームファイルと呼びます。このファイル構造体へのポインタをファイルポインタと呼び、入出力ファイルを指定するために用います。

ファイルポインタは、

```
FILE *fp;
```

と定義します。

fopen関数等でファイルをオープンすると、ファイルポインタが得られますが、オープン処理が失敗するとNULLが返ってきます。NULLポインタを、他のストリーム入出力関数に指定すると、その関数は異常終了しますので、注意が必要です。ファイルをオープンした時は、必ずファイルポインタの値をチェックするようにしてください。

(c) 関数とマクロ

ライブラリ関数の実現方法としては、関数とマクロの二通りがあります。

関数は、通常のユーザ作成の関数と同じインタフェースを持ち、リンク時に取り込みます。

マクロは、その関数に関連した標準インクルードファイルの中で#define文を用いて定義されています。

マクロについては、以下の点に注意する必要があります。

- マクロは、プリプロセッサによって自動的に展開されてしまうので、ユーザが同じ名前の関数を宣言してもマクロを無効にすることはできません。
- マクロのパラメータとして副作用のある式(代入式、インクリメント、デクリメント)を指定した時、その効果は保証しません。

  例：

  パラメータの絶対値を求めるMACROを以下のようにマクロ定義します。

```
#define MACRO(a) ((a)>=0?(a):-(a))
```

と定義されている時、

```
X=MACRO(a++)
```

がプログラム内にあると、

```
X=((a++)>=0?(a++):-(a++))
```

と展開され、aは2回インクリメントされることになり、また結果の値も最初のaの値の絶対値とは異なります。

(d) EOF

getc関数、getchar関数、fgetc関数等のファイルからデータを入力する関数において、ファイル終了(End Of File)時に返される値です。EOFは、標準インクルードファイル<stdio.h>の中で定義されています。

(e) NULL

ポインタが何も指していない時の値です。NULLは、標準インクルードファイル<stddef.h>の中で定義されています。

(f) ヌル文字

C/C++言語における文字列の終わりは、文字"\0"によって示されることになっています。
ライブラリ関数における文字列のパラメータも、すべてこの約束に従っていなければなりません。
この文字列の終わりを示す文字"\0"を以下ヌル文字と呼びます。

(g) リターンコード

ライブラリ関数の中には、リターン値によって、指定された処理が成功したか、失敗したか等の結果を判断するものがあります。
このような場合のリターン値を特にリターンコードと呼びます。

(h) テキストファイルとバイナリファイル

多くのシステムでは、データを格納するために特殊なファイル形式を持っています。
これをサポートするために、ライブラリ関数にはテキストファイルとバイナリファイルの2種類のファイル形式があります。

- テキストファイル

  テキストファイルは、通常のテキストを格納するためのファイルで、行の集まりとして構成されています。テキストファイルの入力の時、行の区切りとして改行文字("\n")が入力されます。また、出力の時、改行文字を出力することにより、現在の行の出力を終了します。テキストファイルは、処理系ごとの標準的なテキストを格納するファイルの入出力を行うためのファイルです。テキストファイルでは、ライブラリ関数で入出力する文字は必ずしもファイル内の物理的なデータの並びと対応していません。

- バイナリファイル

  バイナリファイルは、バイトデータの列として構成されているファイルです。ライブラリ関数で入出力するデータは、ファイル内の物理的なデータの並びと対応しています。

(i) 標準入出力ファイル

入出力のライブラリ関数で、ファイルのオープン等の準備を行わずに標準的に使用できるファイルを標準入出力ファイルといいます。標準入出力ファイルには、標準入力ファイル(stdin)、標準出力ファイル(stdout)、標準エラー出力ファイル(stderr)があります。

- 標準入力ファイル (stdin)

  プログラムへの入力となる標準的なファイルです。

- 標準出力ファイル (stdout)

  プログラムからの出力となる標準的なファイルです。

- 標準エラー出力ファイル(stderr)

  プログラムからのエラーメッセージ等の出力を行うための標準的なファイルです。

(j) 浮動小数点型

浮動小数点型は、実数を近似して表現したものです。C/C++言語のソースプログラム上では浮動小数点型を10進数で表現していますが、計算機の内部では通常2進数で表現されます。
2進数の場合の浮動小数点型の表現は次のようになります。

$2^n \times m$ (n：整数、m：2進小数)

ここでnを浮動小数点型の指数部、mを仮数部といいます。浮動小数点型を一定のデータサイズで表現するために、nとmのビット数は通常固定されています。
以下、浮動小数点型に関する用語を説明します。

- 基数

  浮動小数点型が何進数で表現されているかを示す整数値です。通常、基数は2です。

- 丸め

  浮動小数点型よりも精度の高い演算の途中結果を浮動小数点型に格納する場合に丸めが行われます。丸めには、切り上げ、切り捨て、四捨五入(2進小数の場合は、0捨1入となります)があります。

- 正規化

  浮動小数点型を、$2^n \times m$の形式で表現する場合、同一の数値を表す異なる表現が可能です。

  例：

  $2^5 \times 1.0_{(2)}$ ($_{(2)}$は2進数を示します)

  $2^6 \times 0.1_{(2)}$

  どちらも同じ値です。

  通常は、有効桁数を確保するために、先頭の桁が0でないような表現を用います。これを正規化された浮動小数点型といい、浮動小数点型をこのような表現に変換する操作を正規化といいます。

- ガードビット

  浮動小数点型の演算の途中結果を保持する場合、通常は、丸めを行うために実際の浮動小数点型よりも1ビット多いデータを用意します。しかし、これだけでは桁落ち等が生じた時に正確な結果を求めることができま

せん。このために、もう1ビット設けて演算の途中結果を保持する手法があります。このビットをガードビットといいます。

(k) ファイルアクセスモード

ファイルをオープンする時にどのような処理をファイルに行うかを示す文字列です。文字列の種類には**表 9.28**に示す12種類があります。

表 9.28 ファイルアクセスモードの種類

	アクセスモード	意味
1	"r"	テキストファイルを読み込み用にオープンします。
2	"w"	テキストファイルを書き出し用にオープンします。
3	"a"	テキストファイルを追加用にオープンします。
4	"rb"	バイナリファイルを読み込み用にオープンします。
5	"wb"	バイナリファイルを書き出し用にオープンします。
6	"ab"	バイナリファイルを追加用にオープンします。
7	"r+"	テキストファイルを読み込み用でかつ更新用にオープンします。
8	"w+"	テキストファイルを書き出し用でかつ更新用にオープンします。
9	"a+"	テキストファイルを追加用でかつ更新用にオープンします。
10	"r+b"	バイナリファイルを読み込み用でかつ更新用にオープンします。
11	"w+b"	バイナリファイルを書き出し用でかつ更新用にオープンします。
12	"a+b"	バイナリファイルを追加用でかつ更新用にオープンします。

(l) 処理系定義

コンパイラが異なることによって定義が異なるという意味です。

(m) エラー指示子、ファイル終了指示子

ストリームファイルごとに、ファイルの入出力の際にエラーが生じたかどうかを示すエラー指示子と、入力ファイルが終了したかどうかを示すファイル終了指示子というデータを保持しています。

これらのデータは、それぞれferror関数、feof関数によって参照することができます。

ストリームファイルを扱う関数のうち、そのリターン値だけからではエラーの発生やファイルの終了の情報が得られないものがあります。エラー指示子とファイル終了指示子は、このような関数の実行後にファイルの状態を調べるために使用することができます。

(n) 位置指示子

ディスク上のファイル等、ファイル内の任意の位置からの読み書きができるストリームファイルは、現在読み書きしているファイル内の位置を示すデータを保持しています。これを位置指示子といいます。端末装置等、ファイル内の読み書きの位置を変更できないストリームファイルでは、位置指示子は使用しません。

(d) ライブラリ使用時の注意事項

ライブラリの中で定義されているマクロの内容は、コンパイラごとに異なります。

ライブラリを使用する場合、これらのマクロの内容を再定義した場合、動作は保証しません。

ライブラリは、すべての場合についてエラーを検出しているわけではありません。以降の説明に示す以外の形式でライブラリ関数を呼び出した場合、動作は保証しません。

(2) <stddef.h>

標準インクルードファイルの中で共通に使用されるマクロ名を定義します。

以下は、すべて処理系定義です。

種別	定義名	説明
型	ptrdiff_t	2 つのポインタを減算した結果の型です。
(マクロ)	size_t	sizeof 演算子による演算結果の型です。
定数 (マクロ)	NULL	ポインタが何も指していない時の値です。 これは、0 と等値演算子(==)による比較結果が真になるような値です。
変数 (マクロ)	errno	ライブラリ関数の処理中にエラーが発生した場合、そのライブラリごとに定義されたエラーコードがこの errno に設定されます。 ライブラリ関数を呼び出す前に errno に 0 を設定しておき、ライブラリ関数の処理終了後に errno に設定されているコードを調べることによってライブラリ関数の処理中に発生したエラーをチェックすることができます。
関数 (マクロ)	offsetof	構造体メンバの構造体先頭からのオフセット値をバイト単位で求めます。
型 (マクロ)	wchar_t	拡張文字を表す型です。

処理系定義仕様

	項目	
1	マクロ NULL の値	0（ただし、void 型へのポインタ型）
2	マクロ ptrdiff_t に適合する型	long 型
3	wchar_t に適合する型	short 型

(3) <assert.h>

プログラム中に診断機能を付け加えます。

種別	定義名	説明
関数 (マクロ)	assert	プログラム中に診断機能を付け加えます。

<assert.h>で定義される診断機能を無効にするためには、<assert.h>を取り込む前に NDEBUG というマクロ名を #define 文で定義してください(#define NDEBUG)。

【注】 assert というマクロ名に対して#undef 文を使用すると、それ以降の assert の呼び出しの効果は保証しません。

診断

void assert(long expression)

説 明 プログラム中に診断機能を付け加えます。

ヘッダ <assert.h>

引 数 expression 評価する式

例
```
#include <assert.h>
int expression;
    assert (expression);
```

備 考 assert マクロは、expression が真の時は値を返さずに処理を終了します。expression が偽の時は、診断情報をコンパイラによって定義された書式で標準エラーファイルに出力し、その後 abort 関数を呼び出します。
診断情報の中には、パラメータのプログラムテキスト、ソースファイル名、ソース行番号が含まれています。

処理系定義仕様 assert(expression)において、expression が偽の時メッセージを出力します。
なお、コンパイル時の lang オプションにより表示は変化します。

(1) -lang=c99 がないとき(C(C89)、C++、EC++言語の場合):
ASSERTION FAILED:Δ式ΔFILEΔ<ファイル名>,LINEΔ<行番号>

(2) -lang=c99 があるとき(C(C99)言語の場合):
ASSERTION FAILED:Δ式ΔFILEΔ<ファイル名>,LINEΔ<行番号>ΔFUNCNAMEΔ<関数名>

(4) <ctype.h>

文字に対して、その種類の判定や変換を行います。

種別	定義名	説明
関数	isalnum	英字または 10 進数字かどうかを判定します。
	isalpha	英字かどうかを判定します。
	iscntrl	制御文字かどうかを判定します。
	isdigit	10 進数字かどうかを判定します。
	isgraph	空白を除く印字文字かどうかを判定します。
	islower	英小文字かどうかを判定します。
	isprint	空白を含む印字文字かどうかを判定します。
	ispunct	特殊文字かどうかを判定します。
	isspace	空白類文字かどうかを判定します。
	isupper	英大文字かどうかを判定します。
	isxdigit	16 進数字かどうかを判定します。
	tolower	英大文字を英小文字に変換します。
	toupper	英小文字を英大文字に変換します。
	lsblank	空白文字またはタブ文字かを判定します。

上記の関数において、入力パラメータの値が unsigned char 型で表現できる範囲に含まれず、なおかつ EOF でない場合、その関数の動作は保証しません。

文字の種類の一覧を表 9.29 に示します。

表 9.29 文字の種類

	文字の種類	内容
1	英大文字	以下の 26 文字のいずれかの文字です。 'A'、'B'、'C'、'D'、'E'、'F'、'G'、'H'、'I'、'J'、'K'、'L'、'M'、 'N'、'O'、'P'、'Q'、'R'、'S'、'T'、'U'、'V'、'W'、'X'、'Y'、'Z'
2	英小文字	以下の 26 文字のいずれかの文字です。 'a'、'b'、'c'、'd'、'e'、'f'、'g'、'h'、'i'、'j'、'k'、'l'、'm'、 'n'、'o'、'p'、'q'、'r'、's'、't'、'u'、'v'、'w'、'x'、'y'、'z'
3	英字	英大文字と英小文字のいずれかの文字です。
4	10 進数字	以下の 10 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'
5	印字文字	空白(' ')を含む、ディスプレイ上に表示される文字のことです。 ASCII コードの 0x20～0x7E に対応します。
6	制御文字	印字文字以外の文字のことです。
7	空白類文字	以下の 6 文字のいずれかの文字です。 空白(' ')、書式送り('\f')、改行('\n')、復帰('\r')、水平タブ('\t')、垂直タブ('\v')
8	16 進数字	以下の 22 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'、 'A'、'B'、'C'、'D'、'E'、'F'、'a'、'b'、'c'、'd'、'e'、'f'
9	特殊文字	空白(' ')、英字、および 10 進数字を除く任意の印字文字のことです。
10	ブランク文字	以下の 2 文字のいずれかの文字です。 空白(' ')、水平タブ("\t')

処理系定義仕様

	項目	コンパイラの仕様
1	isalnum 関数、isalpha 関数、iscntrl 関数、islower 関数、isprint 関数、isupper 関数で判定される文字集合	unsigned char型で表現できる文字集合です。判定の結果、真になる文字を表9.30 に示します。

表 9.30 真となる文字の集合

	関数名	真となる文字
1	isalnum	'0'～'9', 'A'～'Z', 'a'～'z'
2	isalpha	'A'～'Z', 'a'～'z'
3	iscntrl	'\x00'～'\x1f', '\x7f'
4	islower	'a'～'z'
5	isprint	'\x20'～'\x7E'
6	isupper	'A'～'Z'

英字、10 進数字判定

long isalnum(long c)

説　明　　文字が英字または 10 進数字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英字または 10 進数字の時　　：0 以外
文字 c が英字または 10 進数字以外の時　　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isalnum(c);
```

英字判定

long isalpha(long c)

説　明　　文字が英字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英字の時　　：0 以外
文字 c が英字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isalpha(c);
```

制御文字判定

long iscntrl(long c)

説　明　　文字が制御文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が制御文字の時　　：0 以外
文字 c が制御文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=iscntrl(c);
```

10 進数字判定

long isdigit(long c)

説　明　　文字が 10 進数字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が 10 進数字の時　　：0 以外
文字 c が 10 進数字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isdigit(c);
```

空白を除く印字文字判定

long isgraph(long c)

説　明　　文字が空白(' ')を除く任意の印字文字かどうかを判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が空白を除く印字文字の時　　：0 以外
文字 c が空白を除く印字文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isgraph(c);
```

英小文字判定

long islower(long c)

説　明　　文字が英小文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英小文字の時　　：0 以外
文字 c が英小文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=islower(c);
```

印字文字判定

long isprint(long c)

説　明　　文字が空白文字(' ')を含む印字文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が空白文字を含む印字文字の時　　：0 以外
文字 c が空白文字を含む印字文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isprint (c);
```

特殊文字判定

long ispunct(long c)

説　明　　文字が特殊文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が特殊文字の時　　：0 以外
文字 c が特殊文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=ispunct(c);
```

空白類文字判定

long isspace(long c)

説　明　　文字が空白類文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が空白類文字の時　　：0 以外
文字 c が空白類文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isspace(c);
```

英大文字判定

long isupper(long c)

説　明　　文字が英大文字であるかどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英大文字の時　　：0 以外
文字 c が英大文字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isupper(c);
```

16 進数字判定

long isxdigit(long c)

説　明　　文字が 16 進数字かどうか判定します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が 16 進数字の時　　：0 以外
文字 c が 16 進数字以外の時　：0

引　数　　c　　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isxdigit(c);
```

英小文字変換

long tolower(long c)

説　明　　英大文字を対応する英小文字に変換します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英大文字の時　　：文字 c に対応する英小文字
文字 c が英大文字以外の時　：文字 c

引　数　　c　　変換する文字

例

```
#include <ctype.h>
int c, ret;
   ret=tolower(c);
```

英大文字変換

long toupper(long c)

説　明　　英小文字を対応する英大文字に変換します。

ヘッダ　　<ctype.h>

リターン値　　文字 c が英小文字の時　　：文字 c に対応する英大文字
文字 c が英小文字以外の時　：文字 c

引　数　　c　　変換する文字

例

```
#include <ctype.h>
int c, ret;
   ret=toupper(c);
```

ブランク判定

long isblank(long c)

説　明	空白文字またはタブ文字か判定します。
ヘッダ	<ctype.h>
リターン値	文字 c が空白文字またはタブ文字の時　：0 以外 文字 c が空白文字でもタブ文字でもない時　：0
引　数	c　判定する文字

例

```
#include <ctype.h>
int c, ret;
   ret=isblank(c);
```

(5) <float.h>

浮動小数点型の内部表現に関する各種制限値を定義します。

以下はすべて処理系定義です。

種別	定義名	定義値	説明
定数 (マクロ)	FLT_RADIX	2	指数部表現における基数です。
	FLT_ROUNDS	1	加算演算結果を丸めるかどうかを示します。本マクロの定義の意味は以下のとおりです。 ・演算結果を丸める場合：正の値 ・演算結果を切り捨てる場合：0 ・特に規定しない場合：-1 丸め、切り捨ての方法は、処理系定義です。
	FLT_GUARD	1	乗算演算結果においてガードビットを用いるかどうかを示します。本マクロの定義の意味は以下のとおりです。 ・ガードビットを用いる場合：1 ・ガードビットを用いない場合：0
	FLT_NORMALIZE	1	浮動小数点値を正規化するかどうかを示します。本マクロの定義の意味は以下のとおりです。 ・正規化する場合：1 ・正規化しない場合：0
	FLT_MAX	3.4028235677973364e+38F	float 型が浮動小数点値として表現できる最大値です。
	DBL_MAX	1.7976931348623158e+308	double 型が浮動小数点値として表現できる最大値です。
	LDBL_MAX	1.7976931348623158e+308	long double 型が浮動小数点値として表現できる最大値です。
	FLT_MAX_EXP	127	float 型が浮動小数点値として表現できる基数のべき乗の最大値です。
	DBL_MAX_EXP	1023	double 型が浮動小数点値として表現できる基数のべき乗の最大値です。
	LDBL_MAX_EXP	1023	long double 型が浮動小数点値として表現できる基数のべき乗の最大値です。
	FLT_MAX_10_EXP	38	float 型が浮動小数点値として表現できる 10 のべき乗の最大値です。
	DBL_MAX_10_EXP	308	double 型が浮動小数点値として表現できる 10 のべき乗の最大値です。
	LDBL_MAX_10_EXP	308	long double 型が浮動小数点値として表現できる 10 のべき乗の最大値です。
	FLT_MIN	1.175494351e-38F	float 型が浮動小数点値として表現できる正の値での最小値です。

種別	定義名	定義値	説明
定数 (マクロ)	DBL_MIN	2.2250738585072014e-308	double 型が浮動小数点値として表現できる正の値での最小値です。
	LDBL_MIN	2.2250738585072014e-308	long double 型が浮動小数点値として表現できる正の値での最小値です。
	FLT_MIN_EXP	-149	float 型が正の値として表現できる浮動小数点値の基数のべき乗の最小値です。
	DBL_MIN_EXP	-1074	double 型が正の値として表現できる浮動小数点値の基数のべき乗の最小値です。
	LDBL_MIN_EXP	-1074	long double 型が正の値として表現できる浮動小数点値の基数のべき乗の最小値です。
	FLT_MIN_10_EXP	-44	float 型が正の値として表現できる浮動小数点値の 10 のべき乗の最小値です。
	DBL_MIN_10_EXP	-323	double 型が正の値として表現できる浮動小数点値の 10 のべき乗の最小値です。
	LDBL_MIN_10_EXP	-323	long double 型が正の値として表現できる浮動小数点値の 10 のべき乗の最小値です。
	FLT_DIG	6	float 型の浮動小数点値の 10 進精度の最大桁数です。
	DBL_DIG	15	double 型の浮動小数点値の 10 進精度の最大桁数です。
	LDBL_DIG	15	long double 型の浮動小数点値の 10 進精度の最大桁数です。
	FLT_MANT_DIG	24	float 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	DBL_MANT_DIG	53	double 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	LDBL_MANT_DIG	53	long double 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	FLT_EXP_DIG	8	float 型の浮動小数点値を基数に合わせて表現した時の指数部の最大桁数です。
	DBL_EXP_DIG	11	double 型の浮動小数点値を基数に合わせて表現した時の指数部の最大桁数です。
	LDBL_EXP_DIG	11	long double 型の浮動小数点値を基数に合わせて表現した時の指数部の最大桁数です。
	FLT_POS_EPS	5.9604648328104311e-8F	float 型において、1.0＋x≠1.0 である最小の浮動小数点値 x を示します。

種別	定義名	定義値	説明
定数 (マクロ)	DBL_POS_EPS	1.1102230246251567e-16	double 型において、1.0+x≠1.0 である最小の浮動小数点値 x を示します。
	LDBL_POS_EPS	1.1102230246251567e-16	long double 型において、1.0+x≠1.0 である最小の浮動小数点値 x を示します。
	FLT_NEG_EPS	2.9802324164052156e-8F	float 型において、1.0−x≠1.0 である最小の浮動小数点値 x を示します。
	DBL_NEG_EPS	5.5511151231257834e-17	double 型において、1.0−x≠1.0 である最小の浮動小数点値 x を示します。
	LDBL_NEG_EPS	5.5511151231257834e-17	long double 型において、1.0−x≠1.0 である最小の浮動小数点値 x を示します。
	FLT_POS_EPS_EXP	-23	float 型において、1.0+(基数)^n≠1.0 となる最小の整数 n を示します。
	DBL_POS_EPS_EXP	-52	double 型において、1.0+(基数)^n≠1.0 となる最小の整数 n を示します。
	LDBL_POS_EPS_EXP	-52	long double 型において、1.0+(基数)^n≠1.0 となる最小の整数 n を示します。
	FLT_NEG_EPS_EXP	-24	float 型において、1.0−(基数)^n≠1.0 となる最小の整数 n を示します。
	DBL_NEG_EPS_EXP	-53	double 型において、1.0−(基数)^n≠1.0 となる最小の整数 n を示します。
	LDBL_NEG_EPS_EXP	-53	long double 型において、1.0−(基数)^n≠1.0 となる最小の整数 n を示します。
	DECIMAL_DIG	10	浮動小数点型数値の 10 進精度の最大桁数です。
	FLT_EPSILON	1E-5	float 型で表現可能な 1 より大きい最小の値と 1 との差を示します。
	DBL_EPSILON	1E-9	double 型で表現可能な 1 より大きい最小の値と 1 との差を示します。
	LDBL_EPSILON	1E-9	long double 型で表現可能な 1 より大きい最小の値と 1 との差を示します。

(6) <limits.h>

整数型データの内部表現に関する各種制限値を定義します。

以下はすべて処理系定義です。

種別	定義名	定義値	説明
定数 (マクロ)	CHAR_BIT	8	char 型が何ビットから構成されるかを示します。
	CHAR_MAX	127	char 型の変数が値として持つことができる最大値です。
		255 *1	
	CHAR_MIN	-128	char 型の変数が値として持つことができる最小値です。
		0 *1	
	SCHAR_MAX	127	signed char 型の変数が値として持つことができる最大値です。
	SCHAR_MIN	-128	signed char 型の変数が値として持つことができる最小値です。
	UCHAR_MAX	255U	unsigned char 型の変数が値として持つことができる最大値です。
	SHRT_MAX	32767	short 型の変数が値として持つことができる最大値です。
	SHRT_MIN	-32768	short 型の変数が値として持つことができる最小値です。
	USHRT_MAX	65535U	unsigned short 型の変数が値として持つことができる最大値です。
	INT_MAX	2147483647	int 型の変数が値として持つことができる最大値です。
		32767*2	
	INT_MIN	-2147483647-1	int 型の変数が値として持つことができる最小値です。
		-32768*2	
	UINT_MAX	4294967295U	unsigned int 型の変数が値として持つことができる最大値です。
		65535U*2	
	LONG_MAX	2147483647L	long 型の変数が値として持つことができる最大値です。
	LONG_MIN	-2147483647L-1L	long 型の変数が値として持つことができる最小値です。

種別	定義名	定義値	説明
定数 (マクロ)	ULONG_MAX	4294967295U	unsigned long 型の変数が値として持つことができる最大値です。
	LLONG_MAX	9223372036854775807LL	long long 型の変数が値として持つことができる最大値です。
	LLONG_MIN	–9223372036854775807LL-1LL	long long 型の変数が値として持つことができる最小値です。
	ULLONG_MAX	18446744073709551615ULL	unsigned long long 型の変数が値として持つことができる最大値です。

【注】 *1 signed_char オプションを指定した場合の変数が値として持つことができる値になります。

*2 int_to_short オプションを指定した場合の変数が値として持つことができる値になります。

(7) <errno.h>

ライブラリ関数においてエラーが発生したときに errno に設定する値を定義します。

以下は、すべて処理系定義です。

種別	定義名	説明
変数 (マクロ)	errno	int 型変数です。ライブラリ関数においてエラーが発生したときにエラー番号が設定されます。
定数 (マクロ)	ERANGE	「11.3 C 標準ライブラリ関数のエラーメッセージ」を参照してください。
	EDOM	
	ESTRN	
	PTRERR	
	ECBASE	
	ETLN	
	EEXP	
	EEXPN	
	EFLOATO	
	EFLOATU	
	EDBLO	
	EDBLU	
	ELDBLO	
	ELDBLU	
	NOTOPN	
	EBADF	
	ECSPEC	
	EFIXEDO	
	EFIXEDU	
	EACCUMO	
	EACCUMU	
	ELFIXEDO	
	ELFIXEDU	
	ELACCUMO	
	ELACCUMU	
	EILSEQ	

(8) <math.h>

各種の数値計算を行います。

以下の定数(マクロ)はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	EDOM	関数に入力するパラメータの値が関数内で定義している値の範囲を超える時、errno に設定する値です。
	ERANGE	関数の計算結果が double 型の値として表せない時、あるいはオーバフロー/アンダフローとなった時、errno に設定する値です。
	HUGE_VAL HUGE_VALF HUGE_VALL	関数の計算結果がオーバフローした時に、関数のリターン値として返す値です。
	INFINITY	正または符号なしの無限大を表す float 型の定数式に展開します。
	NAN	float 型の qNaN をサポートしている場合に定義されます。
	FP_INFINITE FP_NAN FP_NORMAL FP_SUBNORMAL FP_ZERO	浮動小数点数の値の排他的な種類を表します。
	FP_FAST_FMA FP_FAST_FMAF FP_FAST_FMAFL	Fma 関数が double 型のオペランドを持つ 1 回の乗算と加算が、同等以上の速度で実行される場合に定義されます。
	FP_ILOGB0 FP_ILOGBNAN	それぞれ 0 または非数の場合に ilogb で返される値の整数定数式に展開します。
	MATH_ERRNO MATH_ERREXCEPT	それぞれ整数定数 1 および 2 に展開します。
	math_errhandling	Int 型で値が、MATH_ERRNO,MATH_ERREXCEPT のビット単位の論理和の式に展開します。
型	float_t double_t	それぞれ float 型、double 型と同じ幅を持つ浮動小数点型です。
関数 (マクロ)	fpclassify	実引数の値を非数、無限大、正規化数、非正規化数、0 に分類します。
	isfinite	実引数が有限の値か判定します。
	isinf	実引数が無限大か判定します。
	isnan	実引数が非数か判定します。
	isnormal	実引数が正規化数か判定します。
	signbit	実引数の符号が負か判定します。
	isgreater	最初の引数が 2 番目の引数より大きいかどうかを判定します。
	isgreaterequal	最初の引数が 2 番目の引数以上かどうかを判定します。

種別	定義名	説明
関数 (マクロ)	isless	最初の引数が２番目の引数より小さいかどうかを判定します。
	Islessequal	最初の引数が２番目の引数以下かどうかを判定します。
	Islessgreater	最初の引数が２番目の引数より小さいまたは大きいを判定します。
	Isunordered	順序付けられていないかどうかを判定します。
関数	acos	浮動小数点値の逆余弦を計算します。
	acosf	
	acosl	
	asin	浮動小数点値の逆正弦を計算します。
	asinf	
	asinl	
	atan	浮動小数点値の逆正接を計算します。
	atanf	
	atanl	
	atan2	浮動小数点値どうしを除算した結果の値の逆正接を計算します。
	atan2f	
	atan2l	
	cos	浮動小数点値のラジアン値の余弦を計算します。
	cosf	
	cosl	
	sin	浮動小数点値のラジアン値の正弦を計算します。
	sinf	
	sinl	
	tan	浮動小数点値のラジアン値の正接を計算します。
	tanf	
	tanl	
	cosh	浮動小数点値の双曲線余弦を計算します。
	coshf	
	coshl	
	sinh	浮動小数点値の双曲線正弦を計算します。
	sinhf	
	sinhl	
	tanh	浮動小数点値の双曲線正接を計算します。
	tanhf	
	tanhl	
	exp	浮動小数点値の指数関数を計算します。
	expf	
	expl	

種別	定義名	説明
関数	frexp	浮動小数点値を[0.5,1.0]の値と 2 のべき乗の積に分解します。
	frexpf	
	frexpl	
	ldexp	浮動小数点値と 2 のべき乗の乗算を計算します。
	ldexpf	
	ldexpl	
	log	浮動小数点値の自然対数を計算します。
	logf	
	logl	
	log10	浮動小数点値の 10 を底とする対数を計算します。
	log10f	
	log10l	
	modf	浮動小数点値を整数部分と小数部分に分解します。
	modff	
	modfl	
	pow	浮動小数点値のべき乗を計算します。
	powf	
	powl	
	sqrt	浮動小数点値の正の平方根を計算します。
	sqrtf	
	sqrtl	
	ceil	浮動小数点値の小数点以下を切り上げた整数値を求めます。
	ceilf	
	ceill	
	fabs	浮動小数点値の絶対値を計算します。
	fabsf	
	fabsl	
	floor	浮動小数点値の小数点以下を切り捨てた整数値を求めます。
	floorf	
	floorl	
	fmod	浮動小数点値どうしを除算した結果の余りを計算します。
	fmodf	
	fmodl	
	acosh	浮動小数点値の双曲線逆余弦を計算します。
	acoshf	
	acoshl	

種別	定義名	説明
関数	asinh	浮動小数点値の双曲線逆正弦を計算します。
	asinhf	
	asinhl	
	atanh	浮動小数点値の双曲線逆正接を計算します。
	atanhf	
	atanhl	
	exp2	浮動小数点値の 2 の x 乗を計算します。
	exp2f	
	exp2l	
	expm1	自然対数の x 乗から 1 を引いた値を計算します。
	expm1f	
	expm1l	
	ilogb	符号あり int の値として x の指数を抽出します。
	ilogbf	
	ilogbl	
	log1p	実引数に 1 を加えた値の自然対数を計算します。
	log1pf	
	log1pl	
	log2	2 を底とする対数を計算します。
	log2f	
	log2l	
	logb	符号あり整数の値として x の指数を抽出します。
	logbf	
	logbl	
	scalbn	X×FLT_RADIXn を計算します。
	scalbnf	
	scalbnl	
	scalbln	
	scalblnf	
	scalblnl	
	cbrt	浮動小数点値の立方根を計算します。
	cbrtf	
	cbrtl	
	hypot	浮動小数点値の引数ごとに２乗し、その和を計算します。
	hypotf	
	hypotl	

種別	定義名	説明
関数	erf	誤差関数を計算します。
	erff	
	erfl	
	erfc	余誤差関数を計算します。
	erfcf	
	erfcl	
	lgamma	ガンマ関数の絶対値の自然対数を計算します。
	lgammaf	
	lgammal	
	tgamma	ガンマ関数を計算します。
	tgammaf	
	tgammal	
	nearbyint	浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。
	nearbyintf	
	nearbyintl	
	rint	nearbyint に対して、浮動小数点例外を生成することがあります。
	rintf	
	rintl	
	lrint	丸め方向に従って、最も近い整数値に丸めます。
	lrintf	
	lrintl	
	llrint	
	llrintf	
	llrintl	
	round	浮動小数点形式の最も近い整数値に丸めます。
	roundf	
	roundl	
	lround	最も近い整数値に丸めます。
	lroundf	
	lroundl	
	llround	
	llroundf	
	llroundl	
	trunc	浮動小数点形式の最も近い整数値に丸めます。
	truncf	
	truncl	

種別	定義名	説明
関数	remainder	IEEE60559 の剰余 x REM y を計算します。
	remainderf	
	remainderl	
	remquo	x/y と同符号で、商の絶対値を 2^n を法として合同である絶対値を計算します。
	remquof	
	remquol	
	copysign	絶対値、および符号が同じ値を生成します。
	copysignf	
	copysignl	
	nan	nan(“n 文字列”)は、strtod(“NAN(n 文字列)”, (char**) NULL)と等価です。
	nanf	
	nanl	
	nextafter	関数の型に変換して、実軸上の次に表現可能な値を求めます。
	nextafterf	
	nextafterl	
	nexttoward	2 番目の引数型が long double,引数同士が等しい場合に、２番目の引数をその関数の型に変換して返す以外は、nextafter 関数群と同じです。
	nexttowardf	
	nexttowardl	
	fdim	正の差を計算します。
	fdimf	
	fdiml	
	fmax	大きい方の値を求めます。
	fmaxf	
	fmaxl	
	fmin	小さいほうの値を求めます。
	fminf	
	fminl	
	fma	(x×y)+z をひとつの３項演算としてまとめて計算します。
	fmaf	
	fmal	

エラーが発生した時の動作を以下に説明します。

(1) 定義域エラー

関数に入力するパラメータの値が関数内で定義している値の範囲を超えている時、定義域エラーが発生します。この時errnoにはEDOMの値が設定されます。また、関数のリターン値は、処理系定義です。

(2) 範囲エラー

関数における計算結果がdouble型の値として表せない時には範囲エラーが発生します。この時、errnoにはERANGEの値が設定されます。また、計算結果がオーバフローの時は、正しく計算が行われた時と同様の符

号のHUGE_VAL、HUGE_VALF あるいは HUGE_VALLの値をリターン値として返します。逆に計算結果がアンダフローの時は、0をリターン値として返します。

【注】 1. <math.h>の関数の呼び出しによって定義域エラーが発生する可能性がある場合は、結果の値をそのまま用いるのは危険です。必ず errno をチェックしてから用いてください。

例：

```
.
.
.
1               x=asin(a);
2               if (errno==EDOM)
3                   printf("error\n");
4               else
5                   printf("result is : %lf\n",x);
.
.
.
```

1 行目で、asin 関数を使って逆正弦値を求めます。このとき、実引数 a の値が、asin 関数の定義域[-1.0, 1.0]の範囲を超えていると、errno に値 EDOM が設定されます。2 行目で定義域エラーが生じたかどうかの判定をします。定義域エラーが生じれば、3 行目で、error を出力します。定義域エラーが生じなければ 5 行目で、逆正弦値を出力します。

2. 範囲エラーが発生するかどうかは、コンパイラによって定まる、浮動小数点型の内部表現形式によって異なります。例えば無限大を値として表現できる内部表現形式を採用している場合、範囲エラーの生じないように<math.h>のライブラリ関数を実現することができます。

処理系定義仕様

	項目	コンパイラの仕様
1	数学関数の入力実引数が範囲を超えたときの数学関数が返す値	非数を返します。非数の形式は「9.1.3 浮動小数点型の仕様」を参照してください。
2	数学関数でアンダフローエラーが発生したときマクロ「ERANGE」の値が「errno」に設定されるかどうか	設定しません。
3	fmod 関数で第 2 実引数の値が 0 の場合、範囲エラーとなるかどうか	範囲エラーとなります。

逆余弦

double acos(double d)
float acosf(float d)
long double acosl(long double d)

説　明　浮動小数点値の逆余弦を計算します。

ヘッダ　<math.h>

リターン値　正常：d の逆余弦値
異常：定義域エラーの時は、非数を返します

引　数　d　逆余弦を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
   ret=acos(d);
```

エラー条件　d の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　acos 関数のリターン値の範囲は[0,π]です。

逆正弦

double asin(double d)
float asinf(float d)
long double asinl(long double)

説　明　浮動小数点値の逆正弦を計算します。

ヘッダ　<math.h>

リターン値　正常：d の逆正弦値
異常：定義域エラーの時は、非数を返します

引　数　d　逆正弦を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
   ret=asin(d);
```

エラー条件　d の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　asin 関数のリターン値の範囲は[-π/2,π/2]です。

double atan(double d)
float atanf(float d)
long double atanl(long double d)

説　明	浮動小数点値の逆正接を計算します。
ヘッダ	<math.h>
リターン値	d の逆正接値
引　数	d　　逆正接を求める浮動小数点値
例	#include <math.h> double d, ret; ret=atan(d);
備　考	atan 関数のリターン値の範囲は(-π/2,π/2)です。

double atan2(double y, double x)
float atan2f(float y, float x)
long double atan2l(long double y, long double x)

説　明　　浮動小数点値どうしを除算した結果の値の逆正接を計算します。

ヘッダ　　<math.h>

リターン値　　正常：y を x で除算したときの逆正接値
異常：定義域エラーの時は、非数を返します

引　数

x	除数
y	被除数

例

```
#include <math.h>
double x, y, ret;
   ret=atan2(y,x);
```

エラー条件　　x, y の値がともに 0.0 の時、定義域エラーになります。

備　考　　atan2 関数のリターン値の範囲は(-π,π)です。atan2 関数の示す意味を**図** 9.4 に示します。図に示すように、atan2 関数の結果は、点(x,y)と原点を通る直線と x 軸をなす角を求めます。y＝0.0 で x が負の時、結果はπ、x＝0.0 の時、y の値の正負に従って結果は±π/2 となります。

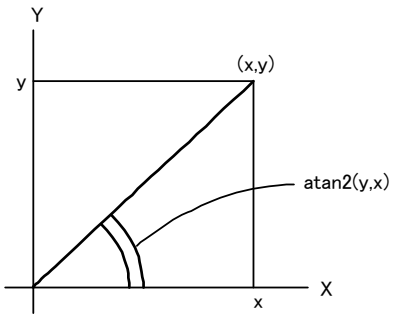


図 9.4　atan2 **関数の意味**

余弦

double cos(double d)
float cosf(float d)
long double cosl(long double d)

説　明　浮動小数点値のラジアン値の余弦を計算します。

ヘッダ　<math.h>

リターン値　d の余弦値

引　数　d　余弦を求めるラジアン値

例
```
#include <math.h>
double d, ret;
   ret=cos(d);
```

正弦

double sin(double d)
float sinf(float d)
long double sinl(long double d)

説　明　浮動小数点値のラジアン値の正弦を計算します。

ヘッダ　<math.h>

リターン値　d の正弦値

引　数　d　正弦を求めるラジアン値

例
```
#include <math.h>
double d, ret;
   ret=sin(d);
```

正接

double tan(double d)
float tanf(float d)
long double tanl(long double d)

説　明　　浮動小数点値のラジアン値の正接を計算します。

ヘッダ　　<math.h>

リターン値　　d の正接値

引　数　　d　　正接を求めるラジアン値

例

```
#include <math.h>
double d, ret;
    ret=tan(d);
```

双曲線余弦

double cosh(double d)
float coshf(float d)
long double coshl(long double d)

説　明　　浮動小数点値の双曲線余弦を計算します。

ヘッダ　　<math.h>

リターン値　　d の双曲線余弦値

引　数　　d　　双曲線余弦を求める浮動小数点値

例

```
#include <math.h>
double d, ret;
    ret=cosh(d);
```

双曲線正弦

double sinh(double d)
float sinhf(float d)
long double sinhl(long double d)

説　明　浮動小数点値の双曲線正弦を計算します。

ヘッダ　<math.h>

リターン値　d の双曲線正弦値

引　数　d　双曲線正弦を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
    ret=sinh(d);
```

双曲線正接

double tanh(double d)
float tanhf(float d)
long double tanhl(long double d)

説　明　浮動小数点値の双曲線正接を計算します。

ヘッダ　<math.h>

リターン値　d の双曲線正接値

引　数　d　双曲線正接を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
    ret=tanh(d);
```

指数関数

double exp(double d)
float expf(float d)
long double expl(long double d)

説　明　　浮動小数点値の指数関数を計算します。

ヘッダ　　<math.h>

リターン値　　d の指数関数値

引　数　　d　　指数関数を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
   ret=exp(d);
```

浮動小数点値を仮数、指数に分解

*double frexp(double value, long *exp)*
*float frexpf(float value, long *exp)*
*long double frexpl(long double value, long *exp)*

説　明　　浮動小数点値を[0.5,1.0)の値と 2 のべき乗の積に分解します。

ヘッダ　　<math.h>

リターン値　　value が 0.0 の時　：0.0
value が 0.0 でない時　：ret*2^(exp の指している領域の値)=value で定義される ret の値

引　数　　value　　[0.5,1.0)の値と 2 のべき乗の積に分解する浮動小数点値
exp　　2 のべき乗値を格納する記憶域へのポインタ

例
```
#include <math.h>
double ret, value;
long *exp;
   ret=frexp(value,exp);
```

備　考　　frexp 関数は、value を[0.5,1.0)の値と 2 のべき乗の積に分解します。exp の指す領域には、分解した結果の 2 のべき乗値を設定します。
リターン値 ret の値の範囲は[0.5,1.0)または 0.0 になります。
value が 0.0 ならば、exp の指す int 型の記憶域の内容と ret の値は 0.0 になります。

仮数、指数を浮動小数点値に変換

double ldexp(double e, long f)
float ldexpf(float e, long f)
long double ldexpl(long double e, long f)

説　明　　浮動小数点値と 2 のべき乗の積を計算します。

ヘッダ　　<math.h>

リターン値　　e*2^f の演算結果の値

引　数

e	2 のべき乗値を求める浮動小数点値
f	2 のべき乗値

例

```
#include <math.h>
double ret, e;
int f;
   ret=ldexp(e, f);
```

自然対数

double log(double d)
float logf(float d)
long double logl(long double d)

説　明　　浮動小数点値の自然対数を計算します。

ヘッダ　　<math.h>

リターン値　　正常：d の自然対数の値
異常：定義域エラーの時は、非数を返します

引　数

d	自然対数を求める浮動小数点値

例

```
#include <math.h>
double d, ret;
   ret=log(d);
```

エラー条件　　d の値が負の時、定義域エラーになります。
d の値が 0.0 の時、範囲エラーになります。

常用対数

double log10(double d)
float log10f(float d)
long double log10l(long double d)

説　明	浮動小数点値の 10 を底とする対数を計算します。
ヘッダ	<math.h>
リターン値	正常：d は 10 を底とする対数値 異常：定義域エラーの時は、非数を返します
引　数	d　　10 を底とする対数を求める浮動小数点値
例	#include <math.h> double d, ret; ret=log10(d);
エラー条件	d の値が負の値の時、定義域エラーになります。 d の値が 0.0 の時、範囲エラーになります。

浮動小数点値を整数部、小数部に分解

*double modf(double a, double *b)*
*float modff(float a, float *b)*
*long double modfl(long double a, long double *b)*

説　明	浮動小数点値を整数部分と小数部分に分解します。
ヘッダ	<math.h>
リターン値	a の小数部分
引　数	a　　整数部分と小数部分に分解する浮動小数点値 b　　整数部分を格納する記憶域を指すポインタ
例	#include <math.h> double a, *b, ret; ret=modf(a,b);

べき乗

double pow(double x, double y)
float powf(float x, float y)
long double powl(long double x, long double y)

説　明　　浮動小数点値のべき乗を計算します。

ヘッダ　　<math.h>

リターン値　　正常：x の y 乗の値
異常：定義域エラーの時は、非数を返します

引　数

x	べき乗される値
y	べき乗する値

例

```
#include <math.h>
double x, y, ret;
   ret=pow(x,y);
```

エラー条件　　x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。

平方根

double sqrt(double d)
float sqrtf(float d)
long double sqrtl(long double d)

説　明　　浮動小数点値の正の平方根を計算します。

ヘッダ　　<math.h>

リターン値　　正常：d の正の平方根の値
異常：定義域エラーの時は、非数を返します

引　数

d	正の平方根を求める浮動小数点値

例

```
#include <math.h>
double d, ret;
   ret=sqrt(d);
```

エラー条件　　d の値が負の値の時、定義域エラーになります。

切り上げ

double ceil(double d)
float ceilf(float d)
long double ceill(long double d)

説　明　浮動小数点値の小数点以下を切り上げた整数値を求めます。

ヘッダ　<math.h>

リターン値　d の小数点以下を切り上げた整数値

引　数　d　小数点以下を切り上げる浮動小数点値

例
```
#include <math.h>
double d, ret;
    ret=ceil(d);
```

備　考　ceil 関数は、d の値より大きいかまたは等しい最小の整数値を double 型の値として返す関数です。したがって d の値が負の値の時は小数点以下を切り捨てた時の値を返します。

絶対値

double fabs(double d)
float fabsf(float d)
long double fabsl(long double d)

説　明　浮動小数点値の絶対値を計算します。

ヘッダ　<math.h>

リターン値　d の絶対値

引　数　d　絶対値を求める浮動小数点値

例
```
#include <math.h>
double d, ret;
    ret=fabs(d);
```

切り捨て

double floor(double d)
float floorf(float d)
long double floorl(long double d)

説　明　　浮動小数点値の小数点以下を切り捨てた整数値を求めます。

ヘッダ　　<math.h>

リターン値　　d の小数点以下を切り捨てた整数値

引　数　　d　　小数点以下を切り捨てる浮動小数点値

例

```
#include <math.h>
double d, ret;
    ret=floor(d);
```

備　考　　floor 関数は、d の値を超えない範囲の整数の最大値を、double 型の値として返す関数です。
したがって d の値が負の値の時は小数点以下を切り上げた時の値を返します。

余り

double fmod(double x, double y)
float fmodf(float x, float y)
long double fmodl(long double x, long double y)

説　明　　浮動小数点値どうしを除算した結果の余りを計算します。

ヘッダ　　<math.h>

リターン値　　y の値が 0.0 の時　　：x
y の値が 0.0 でない時　：x を y で除算した結果の余り

引　数　　x　　被除数
y　　除数

例

```
#include <math.h>
double x, y, ret;
    ret=fmod(x,y);
```

備　考　　fmod 関数では、引数 x、y、リターン値 ret の間には、次に示す関係が成立します。
x=y*i+ret　(ただし i は整数値)
また、リターン値 ret の符号は x の符号と同じ符号になります。
x/y の商を表現できない場合、結果の値は保証しません。

双曲線逆余弦

double acosh(double d)
float acoshf(float d)
long double acoshl(long double d)

説　明　　双曲線逆余弦を計算します。

ヘッダ　　<math.h>

リターン値　　正常：d の双曲線逆余弦値
異常：定義域エラーの時は、非数を返します

引　数　　d　　双曲線逆余弦を求める浮動小数点値

例

```
#include <math.h>
double d, ret;
    ret=acosh(d);
```

エラー条件　　d の値が 1.0 未満の時、定義域エラーになります。

備　考　　acosh 関数のリターン値の範囲は[0,+∞]です。

双曲線逆正弦

double asinh(double d)
float asinhf(float d)
long double asinhl(long double d)

説　明　　双曲線逆正弦を計算します。

ヘッダ　　<math.h>

リターン値　　d の双曲線逆正弦値

引　数　　d　　双曲線逆正弦を求める浮動小数点値

例

```
#include <math.h>
double d, ret;
    ret=asinh(d);
```

double atanh(double d)
float atanhf(float d)
long double atanhl(long double d)

説　明	双曲線逆正接を計算します。
ヘッダ	<math.h>
リターン値	正常：　d の双曲線逆正接値 異常：　定義域エラーの場合は関数に応じて HUGE_VAL, HUGE_VALF, HUGE_VALL 範囲エラーの場合は非数
引　数	d　　双曲線逆正接を求める浮動小数点値
例	#include <math.h> double d, ret; ret=atanh(d);
エラー条件	d の値が[-1,+1]の範囲にない場合は定義域エラーになります。 d の値が-1 または 1 に等しい場合、範囲エラーになる可能性があります。

指数

double exp2(double d)
float exp2f(float d)
long double exp2l(long double d)

説　明　　2 の d 乗を計算します。

ヘッダ　　<math.h>

リターン値　　正常：　2 の指数関数値
異常：　範囲エラーの場合は 0 又は関数に応じて+HUGE_VAL, +HUGE_VALF, +HUGE_VALL

引　数　　d　　指数関数を求める浮動小数点数

例
```
#include <math.h>
double d, ret;
   ret=exp2(d);
```

エラー条件　　d の絶対値が大きすぎる場合、範囲エラーになります。

対数

double expm1(double d)
float expm1f(float d)
long double expm1l(long double d)

説　明　　自然対数の底 e の d 乗から 1 を引いた値を計算します。

ヘッダ　　<math.h>

リターン値　　正常：自然対数の底 e の d 乗から 1 を引いた値
異常：範囲エラーの場合は関数に応じて-HUGE_VAL, -HUGE_VALF, -HUGE_VALL

引　数　　d　　自然対数の底 e の指数となる値

例
```
#include <math.h>
double d, ret;
   ret=expm1(d);
```

エラー条件　　d の値が大きすぎる場合、範囲エラーになります。

備　考　　d の値が 0 に近い場合でも expm1(d)は exp(x)-1 よりも正確に計算できます。

long ilogb(double d)
long ilogbf(float d)
long ilogbl(long double d)

説　明　　d の指数を抽出します。

ヘッダ　　<math.h>

リターン値　　正常：　d の指数関数値
d が∞の場合は INT_MAX
d が非数の場合は FP_ILOGBNAN
d が 0 の場合は FP_ILOGBNAN
異常：　d が 0 で範囲エラーの場合は FP_ILOGB0

引　数　　d　　指数を抽出する値

例

```
#include <math.h>
double d;
int ret;
   ret = ilogb(d);
```

エラー条件　　d の値が 0 の場合、範囲エラーになることがあります。

double log1p(double d)
float log1pf(float d)
long double log1pl(long double d)

説　明	d に 1 を加えた値の e を底とする自然対数を計算します。
ヘッダ	<math.h>
リターン値	正常：　d に 1 を加えた値の自然対数 異常：　定義域エラーの場合は非数 範囲エラーの場合は関数に応じて-HUGE_VAL, -HUGE_VALF, -HUGE_VALL
引　数	d　　引数に 1 を加えた値の自然対数を計算する値
例	#include <math.h> double d; double ret; ret = log1p(d);
エラー条件	d の値が-1 より小さい場合、定義域エラーになります。 d の値が-1 の場合、範囲エラーになります。
備　考	d の値が 0 に近い場合でも log1p(d)は log(1+d)より正確な計算ができます。

対数抽出

double log2(double d)
float log2f(float d)
long double log2l(long double d)

説　明　　d の 2 を底とする対数を計算します。

ヘッダ　　<math.h>

リターン値　　正常：　d の 2 を底とする対数
異常：　定義域エラーの場合は非数

引　数　　d　　対数を計算する値

例

```
#include <math.h>
double d;
int ret;
    ret = log2(d);
```

エラー条件　　d の値が負の値の場合、定義域エラーになります。

指数部抽出

double logb(double d)
float logbf(float d)
long double logbl(long double d)

説　明　　d の浮動小数点数の内部表現における指数部を浮動小数点値として抽出します。

ヘッダ　　<math.h>

リターン値　　正常：　d の符号付き指数
異常：　範囲エラーの場合は関数に応じて-HUGE_VAL, -HUGE_VALF, -HUGE_VALL

引　数　　d　　指数を抽出する値

例

```
#include <math.h>
double d, ret;
    ret = logb(d);
```

エラー条件　　d の値が 0 の場合、範囲エラーになることがあります。

備　考　　d は常に正規化されているものとして処理します。

浮動小数点と FLT_RADIX の乗算

double scalbn(double d, long e)
float scalbnf(float d, long e)
long double scalbnl(long double d, long e)
double scalbln(double d, long e)
float scalblnf(float d, long int e)
long double scalblnl(long double d, long int e)

説　明　浮動小数点数に整数である基数の累乗を計算します。

ヘッダ　<math.h>

リターン値　正常：　d と FLT_RADIX を乗算した値と等価の値
異常：　範囲エラーの場合は関数に応じて-HUGE_VAL, -HUGE_VALF, -HUGE_VALL

引　数

d	FLT_RADIX を e 乗した値と乗算する値
e	FLT_RADIX を累乗する際に指数となる値

例

```
#include <math.h>
double d, ret;
int e;
    ret = scalbn(d,e);
```

エラー条件　d の値が 0 の場合、範囲エラーになることがあります。

備　考　実際に e を指数とした FLT_RADIX の累乗は計算しません。

立方根

double cbrt(double d)
float cbrtf(float d)
long double cbrtl(long double d)

説　明　浮動小数点値の立方根を計算します。

ヘッダ　<math.h>

リターン値　d の立方根値

引　数

d	立方根を求める値

例

```
#include <math.h>
double d, ret;
    ret = cbrt(d);
```

ユークリッド距離

double hypot(double d, double e)
float hypotf(float d, double e)
long double hypotl(long double d, double e)

説　明　　浮動小数点値の 2 乗の和の平方根を計算します。

ヘッダ　　<math.h>

リターン値　　正常：　d の 2 乗と e の 2 乗の和の平方根関数値
異常：　範囲エラーの場合は関数に応じて HUGE_VAL, HUGE_VALF, HUGE_VALL

引　数　　d, e　　2 乗の和の平方根を求める値

例

```
#include <math.h>
double d, e, ret;
   ret = hypot(d, e);
```

エラー条件　　結果がオーバーフローする場合に範囲エラーになることがあります。

誤差

double erf(double d)
float erff(float d)
long double erfl(long double d)

説　明　　浮動小数点値の誤差関数を計算します。

ヘッダ　　<math.h>

リターン値　　d の誤差関数値

引　数　　d　　誤差関数値を求める値

例

```
#include <math.h>
double d, ret;
   ret = erf(d);
```

余誤

double erfc(double d)
float erfcf(float d)
long double erfcl(long double d)

説　明　　浮動小数点値の余誤関数を計算します。

ヘッダ　　<math.h>

リターン値　　d の余誤関数値

引　数　　d　　余誤関数値を求める値

例

```
#include <math.h>
double d, ret;
    ret = erfc(d);
```

エラー条件　　d の絶対値が大きすぎる場合、範囲エラーが発生します。

ガンマ関数の対数

double lgamma(double d)
float lgammaf(float d)
long double lgammal(long double d)

説　明　　浮動小数点値のガンマ関数の対数を計算します。

ヘッダ　　<math.h>

リターン値　　正常：　d のガンマ関数の対数値
異常：　定義域エラーの場合は数学的に正しい符号が付与された
HUGE_VAL, HUGE_VALF, HUGE_VALL
範囲エラーの場合は+HUGE_VAL, +HUGE_VALF, +HUGE_VALL

引　数　　d　　ガンマ関数の対数値を求める値

例

```
#include <math.h>
double d, ret;
    ret = lgamma(d);
```

エラー条件　　d の絶対値が大きすぎる又は小さすぎる場合、範囲エラーを設定します。
d の値が負の整数又は 0 で、計算結果が表現できない場合、定義域エラーが発生します。

ガンマ

double tgamma(double d)
float tgammaf(float d)
long double tgammal(long double d)

説　明　浮動小数点値のガンマ関数を計算します。

ヘッダ　<math.h>

リターン値　正常：　d のガンマ関数値
異常：　定義域エラーの場合は d と同じ符号が付与された
HUGE_VAL, HUGE_VALF, HUGE_VALL
範囲エラーの場合は 0 又は
関数に応じて数学的に正しい符号が付与された
+HUGE_VAL, +HUGE_VALF, +HUGE_VALL

引　数　d　ガンマ関数値を求める値

例
```
#include <math.h>
double d, ret;
    ret = tgamma(d);
```

エラー条件　d の絶対値が大きすぎる又は小さすぎる場合、範囲エラーを設定します。
d の値が負の整数又は 0 で、計算結果が表現できない場合、定義域エラーが発生します。

整数変換

double nearbyint(double d)
float nearbyintf(float d)
long double nearbyintl(long double d)

説　明　浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。

ヘッダ　<math.h>

リターン値　d の浮動小数点形式の整数値に丸めた値

引　数　d　動小数点形式の整数値に丸める値

例
```
#include <math.h>
double d, ret;
    ret = nearbyint (d);
```

備　考　nearbyint 関数群は”不正確結果”浮動小数点例外を生成しません。

整数変換

double rint(double d)
float rintf(float d)
long double rintl(long double d)

説　明　　浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。

ヘッダ　　<math.h>

リターン値　　d の浮動小数点形式の整数値に丸めた値

引　数　　d　　動小数点形式の整数値に丸める値

例

```
#include <math.h>
double d, ret;
   ret = rint (d);
```

備　考　　rint 関数群は"不正確結果"浮動小数点例外を生成する場合があるという点のみで nearbyint 関数群と違います。

整数変換

long int lrint (double d)
long int lrintf(float d)
long int lrintl(long double d)
long long int llrint (double d)
long long int llrintf(float d)
long long int llrintl(long double d)

説　明　　浮動小数点値を丸め方向にしたがって、最も近い整数値に丸めます。

ヘッダ　　<math.h>

リターン値　　正常：　d を整数値に丸めた値
異常：　範囲エラーの場合は不定

引　数　　d　　整数に丸める値

例

```
#include <math.h>
double d;
long int ret;
   ret = lrint (d);
```

エラー条件　　d の絶対値が大きすぎる場合、範囲エラーが発生する場合があります。

備　考　　丸めた値がリターン値型の範囲外である場合のリターン値は未規定とします。

double round(double d)
float roundf(float d)
long double roundl(long double d)
long int lround(double d)
long int lroundf(float d)
long int lroundl(long double d)
long long int llround (double d)
long long int llroundf(float d)
long long int llroundl(long double d)

説　明	浮動小数点値を最も近い整数値に丸めます。
ヘッダ	`<math.h>`
リターン値	正常：　d を整数値に丸めた値 異常：　範囲エラーの場合は不定
引　数	d　　整数に丸める値

例	`#include <math.h>`
	`double d;`
	`long int ret;`
	`ret = lround(d);`

エラー条件	d の絶対値が大きすぎる場合、範囲エラーが発生する場合があります。
備　考	lround 関数群は d の値が中間にある場合、その時点の丸め方向とは関係なく 0 から遠い方向を選んで丸めます。丸めた値がリターン値型の範囲外である場合のリターン値は未規定とします。

整数変換

double trunc(double d)
float truncf(float d)
long double truncl(long double d)

説　明　　浮動小数点値を最も近い浮動小数点形式の整数値に丸めます。

ヘッダ　　<math.h>

リターン値　　d を切り捨てた浮動小数点形式の整数値

引　数　　d　　浮動小数点形式の整数に丸める値

例

```
#include <math.h>
double d, ret;
    ret = trunc(d);
```

備　考　　trunc 関数群は丸めた値の絶対値が d の絶対値より大きくならないようにします。

浮動小数点剰余計算

double remainder(double d1, double d2)
float remainderf(float d1, float d2)
long double remainderl(long double d1, long double d2)

説　明　　浮上小数点数同士の剰余を計算します。

ヘッダ　　<math.h>

リターン値　　d1 と d2 の剰余値

引　数　　d1　　剰余を求める値
d2

例

```
#include <math.h>
double d1, d2, ret;
    ret = remainder(d1, d2);
```

備　考　　remainder 関数群の剰余計算は IEEE 60559 の規定に沿っています。

浮動小数点剰余計算

*double remquo(double d1, double d2, long *q)*
*float remquof(float d1, float d2, long *q)*
*long double remquol(long double d1, long double d2, long *q)*

説　明　　浮動小数点値を最も近い整数値に丸めます。

ヘッダ　　<math.h>

リターン値　　d1 と d2 の剰余値

引　数

d1	整数に丸める値
d2	
q	剰余計算結果の商を格納する場所を指す値

例

```
#include <math.h>
double d1, d2, ret;
long q;
    ret = remquo(d1, d2, &q);
```

備　考　　q に格納される値は x/y と同じ符号と、2n(n は 3以上の処理系定義整数値)を法とする x/y の整数の商を持ちます。

符号コピー

double copysign(double d1, double d2)
float copysignf(float d1, float d2)
long double copysignl(long double d1, long double d2)

説　明　　絶対値が d1 に等しく、符号ビットが d2 に等しい値を生成します。

ヘッダ　　<math.h>

リターン値　　正常：　d1 の絶対値、d2 の符号の値
異常：　範囲エラーの場合は不定

引　数

d1	生成する絶対値の値
d2	生成する符号

例

```
#include <math.h>
double d1, d2, ret;
    ret = copysign(d1, d2);
```

備　考　　copysign 関数群は d1 が非数の場合 d2 の符号ビットを持った非数を生成します。

*double nan(const char *c)*
*float nanf(const char *c)*
*long double nanl(const char *c)*

説　明　　非数を返します。

ヘッダ　　<math.h>

リターン値　　c が示す内容をもつ qNaN または 0(qNaN 未サポート時)

引　数　　c　　文字列ポインタ

例

```
#include <math.h>
double ret;
const char *c;
    ret = nan(c);
```

備　考　　nan("c 文字列")の呼出しは，strtod("NAN(c 文字列)", (char**) NULL)と等価です。nanf 及び nanl の呼出しは，strtof 及び strtold のそれぞれに対応する呼出しと等価です。

浮動小数点数操作

double nextafter(double d1, double d2)
float nextafter f(float d1, float d2)
long double nextafterl(long double d1, long double d2)

説　明	実軸上で d1 から見て d2 に向かう方向で d1 のすぐ次の浮動小数点数表現を計算します。
ヘッダ	<math.h>
リターン値	正常：　表現可能な浮動小数点値 異常：　範囲エラーの場合、関数に応じて数学的に正しい符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL
引　数	d1　実軸上の浮動小数点値 d2　d1 から見て表現可能な浮動小数点値の存在する方向を示す値

例	`#include <math.h>`
	`double d1, d2, ret;`
	`ret = nextafter(d1, d2);`

エラー条件	d1 がその型で表現できる最大の有限な値であり，かつリターン値が無限大又はその型で表現できない場合，範囲エラーが発生することがあります。
備　考	nextafter 関数群は d1 と d2 が等しい場合、d2 を返します。

double nexttoward(double d1, long double d2)
float nexttowardf(float d1, long double d2)
long double nexttowardl(long double d1, long double d2)

説　明　　実軸上で d1 から見て d2 に向かう方向で d1 のすぐ次の浮動小数点数表現を計算します。

ヘッダ　　<math.h>

リターン値　　正常：　表現可能な浮動小数点値
異常：　範囲エラーの場合、関数に応じて数学的に正しい符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL

引　数

d1	実軸上の浮動小数点値
d2	d1 から見て表現可能な浮動小数点値の存在する方向を示す値

例

```
#include <math.h>
double d1, ret;
long double d2;
   ret = nexttoward(d1, d2);
```

エラー条件　　d1 がその型で表現できる最大の有限な値であり，かつリターン値が無限大又はその型で表現できない場合，範囲エラーが発生することがあります。

備　考　　nexttoward 関数群は d2 の値が long double であり、d1 と d2 が等しい場合は d2 を関数に応じて変換して返すという点以外は nextafter 関数群と等価です。

正の差

double fdim(double d1, double d2)
float fdimf(float d1, float d2)
long double fdiml(long double d1, long double d2)

説　明　　2引数間の正の差分を求めます。

ヘッダ　　<math.h>

リターン値　　正常：　2引数間の正の差分
異常：　範囲エラーの場合は HUGE_VAL, HUGE_VALF, HUGE_VALL

引　数　　d1　　正の差を求める値
d2

例

```
#include <math.h>
double d1, d2, ret;
   ret = fdim(d1, d2);
```

エラー条件　　リターン値がオーバーフローした場合に範囲エラーが発生することがあります。

最大値

double fmax(double d1, double d2)
float fmaxf(float d1, float d2)
long double fmaxl(long double d1, long double d2)

説　明　　2引数の大きい方を求めます。

ヘッダ　　<math.h>

リターン値　　2引数の大きい方

引　数　　d1　　大きさを比較する値
d2

例

```
#include <math.h>
double d1, d2, ret;
   ret = fmax(d1, d2);
```

備　考　　fmax 関数群は非数を，データが欠けているものとして認識します。一方の引数が非数で，もう一方が数値の場合，数値の値を返します。

最小値

double fmin(double d1, double d2)
float fminf(float d1, float d2)
long double fminl(long double d1, long double d2)

説　明　　2引数の小さい方を求めます。

ヘッダ　　<math.h>

リターン値　　2引数の小さい方

引　数　　d1　　大きさを比較する値
d2

例
```
#include <math.h>
double d1, d2, ret;
   ret = fmin(d1, d2);
```

備　考　　fmin 関数群は非数を，データが欠けているものとして認識します。一方の引数が非数で，もう一方が数値の場合，数値の値を返します。

積と和

double fma(double d1, double d2, double d3)
float fmaf(float d1, float d2, float d3)
long double fmal(long double d1, long double d2, long double d3)

説　明　　(d1*d2)+d3 を一つの 3 項演算としてまとめて計算します。

ヘッダ　　<math.h>

リターン値　　(d1*d2)+d3 を 3 項演算としてまとめて計算した結果

引　数　　d1, d2, d3　　浮動小数点値

例
```
#include <math.h>
double d1, d2, ret;
   ret = fma(d1, d2);
```

備　考　　fma 関数群は計算結果を無限の精度であるものとして計算し、FLT_ROUNDS の値が示す丸めモードに従って、1 回だけ丸めます。

(9) <mathf.h>

各種の数値計算を行います。

<mathf.h>では ANSI 規格規定外の単精度形式の数学関数の宣言とマクロの定義をしています。

各関数は float 型の実引数を受け取り、float 型の値を返します。

以下の定数(マクロ)はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	EDOM	関数に入力するパラメータの値が関数内で定義している値の範囲を超える時、errno に設定する値です。
	ERANGE	関数の計算結果が float 型の値として表せない時、あるいはオーバフロー/アンダフローとなった時、errno に設定する値です。
	HUGE_VALF	関数の計算結果がオーバフローした時に、関数のリターン値として返す値です。
関数	acosf	浮動小数点値の逆余弦を計算します。
	asinf	浮動小数点値の逆正弦を計算します。
	atanf	浮動小数点値の逆正接を計算します。
	atan2f	浮動小数点値どうしを除算した結果の値の逆正接を計算します。
	cosf	浮動小数点値のラジアン値の余弦を計算します。
	sinf	浮動小数点値のラジアン値の正弦を計算します。
	tanf	浮動小数点値のラジアン値の正接を計算します。
	coshf	浮動小数点値の双曲線余弦を計算します。
	sinhf	浮動小数点値の双曲線正弦を計算します。
	tanhf	浮動小数点値の双曲線正接を計算します。
	expf	浮動小数点値の指数関数を計算します。
	frexpf	浮動小数点値を[0.5, 1.0)の値と 2 のべき乗の積に分解します。
	ldexpf	浮動小数点値と 2 のべき乗の乗算を計算します。
	logf	浮動小数点値の自然対数を計算します。
	log10f	浮動小数点値の 10 を底とする対数を計算します。
	modff	浮動小数点値を整数部分と小数部分に分解します。
	powf	浮動小数点値のべき乗を計算します。
	sqrtf	浮動小数点値の正の平方根を計算します。
	ceilf	浮動小数点値の小数点以下を切り上げた整数値を求めます。
	fabsf	浮動小数点値の絶対値を計算します。
	floorf	浮動小数点値の小数点以下を切り捨てた整数値を求めます。
	fmodf	浮動小数点値どうしを除算した結果の余りを計算します。

エラーが発生した時の動作を以下に説明します。

(1) 定義域エラー

関数に入力するパラメータの値が関数内で定義している値の範囲を超えている時、定義域エラーが発生します。この時errnoにはEDOMの値が設定されます。また、関数のリターン値は、処理系定義です。

(2) 範囲エラー

関数における計算結果がfloat型の値として表せない時には範囲エラーが発生します。この時、errnoにはERANGEの値が設定されます。また、計算結果がオーバフローの時は、正しく計算が行われた時と同様の符号のHUGE_VALFの値をリターン値として返します。逆に計算結果がアンダフローの時は、0をリターン値として返します。

【注】 1. <mathf.h>**の関数の呼び出しによって定義域エラーが発生する可能性がある場合は、結果の値をそのまま用いるのは危険です。必ず** errno **をチェックしてから用いてください。**

例：

```
                .
                .
                .
1         x=asinf(a);
2         if (errno==EDOM)
3             printf("error\n");
4         else
5             printf("result is : %f\n",x);
                .
                .
                .
```

1 行目で、asinf **関数を使って逆正弦値を求めます。このとき、実引数** a **の値が、**asinf **関数の定義域**[-1.0,1.0]**の範囲を超えていると、**errno **に値** EDOM **が設定されます。**2 **行目で定義域エラーが生じたかどうかの判定をします。定義域エラーが生じれば、**3 **行目で、**error **を出力します。定義域エラーが生じなければ** 5 **行目で、逆正弦値を出力します。**

2. **範囲エラーが発生するかどうかは、コンパイラによって定まる、浮動小数点型の内部表現形式によって異なります。例えば無限大を値として表現できる内部表現形式を採用している場合、範囲エラーの生じないように**<mathf.h>**のライブラリ関数を実現することができます。**

処理系定義仕様

	項目	コンパイラの仕様
1	数学関数の入力実引数が範囲を超えたときの数学関数が返す値	非数を返します。非数の形式は「9.1.3 浮動小数点型の仕様」を参照してください
2	数学関数でアンダフローエラーが発生したときマクロ「ERANGE」の値が「errno」に設定されるかどうか	設定しません。
3	fmodf 関数で第 2 実引数の値が 0 の場合、範囲エラーとなるかどうか	範囲エラーとなります。

逆余弦

float acosf(float f)

説　明　浮動小数点値の逆余弦を計算します。

ヘッダ　<mathf.h>

リターン値　正常：f の逆余弦値
異常：定義域エラーの時は、非数を返します

引　数　f　逆余弦を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=acosf(f);
```

エラー条件　f の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　acosf 関数のリターン値の範囲は[0,π]です。

逆正弦

float asinf(float f)

説　明　浮動小数点値の逆正弦を計算します。

ヘッダ　<mathf.h>

リターン値　正常：f の逆正弦値
異常：定義域エラーの時は、非数を返します

引　数　f　逆正弦を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=asinf(f);
```

エラー条件　f の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　asinf 関数のリターン値の範囲は[-π/2,π/2]です。

float atanf(float f)

説　明	浮動小数点値の逆正接を計算します。
ヘッダ	<mathf.h>
リターン値	f の逆正接値
引　数	f　　逆正接を求める浮動小数点値
例	#include <mathf.h> float f, ret; ret=atanf(f);
備　考	atanf 関数のリターン値の範囲は(-π/2,π/2)です。

float atan2f(float y, float x)

説　明　　浮動小数点値どうしを除算した結果の値の逆正接を計算します。

ヘッダ　　<mathf.h>

リターン値　　正常：y を x で除算したときの逆正接値
異常：定義域エラーの時は、非数を返します

引　数

x	除数
y	被除数

例

```
#include <mathf.h>
float x, y, ret;
   ret=atan2f(y,x);
```

エラー条件　　x, y の値がともに 0.0 の時、定義域エラーになります。

備　考　　atan2f 関数のリターン値の範囲は(-π,π)です。atan2f 関数の示す意味を**図** 9.5 に示します。図に示すように、atan2f 関数の結果は、点(x,y)と原点を通る直線と x 軸をなす角を求めます。y＝0.0 で x が負の時、結果はπ、x＝0.0 の時、y の値の正負に従って結果は±π/2 となります。

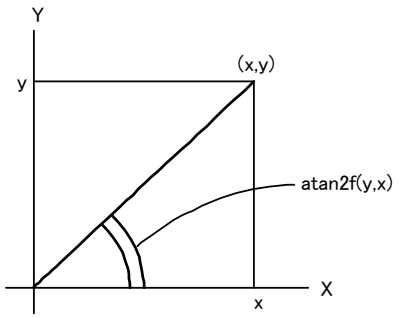


図 9.5　atan2f **関数の意味**

余弦

float cosf(float f)

説　明　　浮動小数点値のラジアン値の余弦を計算します。

ヘッダ　　<mathf.h>

リターン値　　f の余弦値

引　数　　f　　余弦を求めるラジアン値

例

```
#include <mathf.h>
float f, ret;
   ret=cosf(f);
```

正弦

float sinf(float f)

説　明　　浮動小数点値のラジアン値の正弦を計算します。

ヘッダ　　<mathf.h>

リターン値　　f の正弦値

引　数　　f　　正弦を求めるラジアン値

例

```
#include <mathf.h>
float f, ret;
   ret=sinf(f);
```

正接

float tanf(float f)

説　明　浮動小数点値のラジアン値の正接を計算します。

ヘッダ　<mathf.h>

リターン値　f の正接値

引　数　f　正接を求めるラジアン値

例
```
#include <mathf.h>
float f, ret;
   ret=tanf(f);
```

双曲線余弦

float coshf(float f)

説　明　浮動小数点値の双曲線余弦を計算します。

ヘッダ　<mathf.h>

リターン値　f の双曲線余弦値

引　数　f　双曲線余弦を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=coshf(f);
```

双曲線正弦

float sinhf(float f)

説　明　浮動小数点値の双曲線正弦を計算します。

ヘッダ　<mathf.h>

リターン値　f の双曲線正弦値

引　数　f　双曲線正弦を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=sinhf(f);
```

双曲線正接

float tanhf(float f)

説　明　浮動小数点値の双曲線正接を計算します。

ヘッダ　<mathf.h>

リターン値　f の双曲線正接値

引　数　f　双曲線正接を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=tanhf(f);
```

指数関数

float expf(float f)

説　明　　浮動小数点値の指数関数を計算します。

ヘッダ　　<mathf.h>

リターン値　　f の指数関数値

引　数　　f　　指数関数を求める浮動小数点値

例

```
#include <mathf.h>
float f, ret;
   ret=expf(f);
```

浮動小数点値を仮数、指数に分解

*float frexpf(float value, long *exp)*

説　明　　浮動小数点値を[0.5,1.0)の値と 2 のべき乗の積に分解します。

ヘッダ　　<mathf.h>

リターン値　　value が 0.0 の時　　：0.0
value が 0.0 でない時　：ret*2^(exp の指している領域の値)=value で定義される ret の値

引　数　　value　　[0.5,1.0)の値と 2 のべき乗の積に分解する浮動小数点値
exp　　2 のべき乗値を格納する記憶域へのポインタ

例

```
#include <mathf.h>
float ret, value;
long *exp;
   ret=frexpf(value,exp);
```

備　考　　frexpf 関数は、value を[0.5,1.0)の値と 2 のべき乗の積に分解します。exp の指す領域には、分解した結果の 2 のべき乗値を設定します。
リターン値 ret の値の範囲は[0.5,1.0)または 0.0 になります。
value が 0.0 ならば、exp の指す int 型の記憶域の内容と ret の値は 0.0 になります。

仮数、指数を浮動小数点値に変換

float ldexpf(float e, long f)

説　明　　浮動小数点値と 2 のべき乗の積を計算します。

ヘッダ　　<mathf.h>

リターン値　　e*2^f の演算結果の値

引　数　　e　　2 のべき乗との積を求める浮動小数点値
f　　2 のべき乗値

例

```
#include <mathf.h>
float ret, e;
int  f;
   ret=ldexpf(e, f);
```

自然対数

float logf(float f)

説　明　　浮動小数点値の自然対数を計算します。

ヘッダ　　<mathf.h>

リターン値　　正常：f の自然対数の値
異常：定義域エラーの時は、非数を返します

引　数　　f　　自然対数を求める浮動小数点値

例

```
#include <mathf.h>
float f, ret;
   ret=logf(f);
```

エラー条件　　f の値が負の時、定義域エラーになります。
f の値が 0.0 の時、範囲エラーになります。

常用対数

float log10f(float f)

説　明　　浮動小数点値の 10 を底とする対数を計算します。

ヘッダ　　<mathf.h>

リターン値　　正常：f は 10 を底とする対数値
異常：定義域エラーの時は、非数を返します

引　数　　f　　10 を底とする対数を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=log10f(f);
```

エラー条件　　f の値が負の値の時、定義域エラーになります。
f の値が 0.0 の時、範囲エラーになります。

浮動小数点値を整数部、小数部に分解

*float modff(float a, float *b)*

説　明　　浮動小数点値を整数部分と小数部分に分解します。

ヘッダ　　<mathf.h>

リターン値　　a の小数部分

引　数　　a　　整数部分と小数部分に分解する浮動小数点値
b　　整数部分を格納する記憶域を指すポインタ

例
```
#include <mathf.h>
float a, *b, ret;
   ret=modff(a,b);
```

べき乗

float powf(float x, float y)

説　明　浮動小数点値のべき乗を計算します。

ヘッダ　<mathf.h>

リターン値　正常：x の y 乗の値
異常：定義域エラーの時は、非数を返します

引　数

x	べき乗される値
y	べき乗する値

例

```
#include <mathf.h>
float x, y, ret;
   ret=powf(x, y);
```

エラー条件　x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。

平方根

float sqrtf(float f)

説　明　浮動小数点値の正の平方根を計算します。

ヘッダ　<mathf.h>

リターン値　正常：f の正の平方根の値
異常：定義域エラーの時は、非数を返します

引　数

f	正の平方根を求める浮動小数点値

例

```
#include <mathf.h>
float f, ret;
   ret=sqrtf(f);
```

エラー条件　f の値が負の値の時、定義域エラーになります。

切り上げ

float ceilf(float f)

説　明　浮動小数点値の小数点以下を切り上げた整数値を求めます。

ヘッダ　<mathf.h>

リターン値　f の小数点以下を切り上げた整数値

引　数　f　小数点以下を切り上げる浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=ceilf(f);
```

備　考　ceilf 関数は、f の値より大きい、または等しい最小の整数値を float 型の値として返す関数です。したがって f の値が負の値の時は小数点以下を切り捨てた時の値を返します。

絶対値

float fabsf(float f)

説　明　浮動小数点値の絶対値を計算します。

ヘッダ　<mathf.h>

リターン値　f の絶対値

引　数　f　絶対値を求める浮動小数点値

例
```
#include <mathf.h>
float f, ret;
   ret=fabsf(f);
```

切り捨て

float floorf(float f)

説　明　　浮動小数点値の小数点以下を切り捨てた整数値を求めます。

ヘッダ　　<mathf.h>

リターン値　　f の小数点以下を切り捨てた整数値

引　数　　f　　小数点以下を切り捨てる浮動小数点値

例

```
#include <mathf.h>
float f, ret;
   ret=floorf(f);
```

備　考　　floorf 関数は、f の値を超えない範囲の整数の最大値を、float 型の値として返す関数です。
したがって f の値が負の値の時は小数点以下を切り上げた時の値を返します。

余り

float fmodf(float x, float y)

説　明　　浮動小数点値どうしを除算した結果の余りを計算します。

ヘッダ　　<mathf.h>

リターン値　　y の値が 0.0 の時：x
y の値が 0.0 でない時：x を y で除算した結果の余り

引　数　　x　　被除数
y　　除数

例

```
#include <mathf.h>
float x, y, ret;
   ret=fmodf(x, y);
```

備　考　　fmodf 関数では、引数 x、y、リターン値 ret の間には、次に示す関係が成立します。
x=y*i+ret　(ただし i は整数値)
また、リターン値 ret の符号は x の符号と同じ符号になります。
x/y の商を表現できない場合、結果の値は、保証しません。

（10）<setjmp.h>

関数間の制御の移動をサポートします。

以下のマクロは、処理系定義です。

種別	定義名	説明
型 (マクロ)	jmp_buf	関数間の制御の移動を可能とする情報を保存しておくための記憶域に対応する型名です。
関数	setjmp	現在実行中の関数の jmp_buf で定義した実行環境を指定した記憶域に退避します。
	longjmp	setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。

setjmp 関数は現在の関数の実行環境を退避します。その後 longjmp 関数を呼び出すことにより、setjmp 関数を呼び出したプログラム上の位置に戻ることができます。

以下に setjmp、longjmp 関数を使用して関数間の制御の移動をサポートした例を示します。

```
1           #include <stdio.h>
2           #include <setjmp.h>
3           jmp_buf env;
4           void sub();
5           void main()
6           {
7
8              if (setjmp(env)!=0){
9                  printf("return from longjmp\n");
10                 exit(0);
11             }
12             sub();
13          }
14
15          void sub()
16          {
17             printf("subroutine is running \n");
18             longjmp(env,1);
19          }
```

【説明】

8 行目で setjmp 関数を呼んでいます。この時、setjmp 関数の呼び出された環境を、jmp_buf 型の変数 env に退避します。この時のリターン値は 0 なので、次に関数 sub が呼び出されます。

関数 sub の中で呼び出される longjmp 関数により、変数 env に退避した環境を回復します。その結果、プログラムは、あたかも 8 行目の setjmp 関数からリターンしたかのようにふるまいます。ただし、この時のリターン値は longjmp 関数の第 2 実引数で指定した値(1)になります。

その結果、9 行目以降が実行されます。

大域 goto 飛び先設定

long setjmp(jmp_buf env)

説　明　　現在実行中の関数の実行環境を、指定した記憶域に退避します。

ヘッダ　　<setjmp.h>

リターン値　　setjmp 関数を呼び出した時　　：0
longjmp 関数からのリターン時　：0 以外

引　数　　env　　実行環境を退避する記憶域へのポインタ

例

```
#include <setjmp.h>
int ret;
jmp_buf env;
   ret=setjmp(env);
```

備　考　　setjmp 関数により退避された実行環境は、longjmp 関数において使用されます。
setjmp 関数として呼び出された時のリターン値は 0 ですが、longjmp 関数からリターンしてきた時のリターン値は、longjmp 関数で指定した第 2 引数の値となります。
setjmp 関数を複雑な式から呼び出す場合、式の評価の途中結果等の現在の実行環境の一部が失われる可能性があります。setjmp 関数は setjmp 関数の結果と定数式の比較という形だけで使用し、複雑な式の中では呼び出さないようにしてください。
setjmp 関数へのポインタを使った間接呼び出しはしないでください。

大域 goto

void longjmp(jmp_buf env, long ret)

説　明　　setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。

ヘッダ　　<setjmp.h>

引　数　　env　　実行環境を退避した記憶域へのポインタ
ret　　setjmp 関数へのリターンコード

例

```
#include <setjmp.h>
int ret;
jmp_buf env;
   longjmp(env,ret);
```

備　考　　longjmp 関数は、同じプログラム中で最後に呼び出された setjmp 関数によって退避された関数の実行環境を第 1 引数 env で指定された記憶域から回復し、その setjmp 関数を呼び出したプログラムの位置に制御を移します。この時 longjmp 関数の第 2 引数 ret が setjmp 関数のリターン値として返ります。ただし、ret が 0 の時は setjmp 関数へのリターン値としては 1 が返ります。
setjmp 関数が呼び出されていない時、あるいは setjmp 関数を呼び出した関数がすでに return 文を実行している時は、longjmp 関数の動作は保証しません。

（11）<stdarg.h>

可変個の引数を持つ関数に対し、その引数の参照を可能にします。

以下はすべて処理系定義です。

種別	定義名	説明
型 (マクロ)	va_list	可変個の引数を参照するために、va_start, va_arg, va_end マクロで共通に使用される変数の型です。
関数 (マクロ)	va_start	可変個の引数の参照を行うため、初期設定処理を行います。
	va_arg	可変個の引数を持つ関数に対して、現在参照中引数の次の引数への参照を可能とします。
	va_end	可変個の引数を持つ関数の引数への参照を終了させます。
	va_copy	可変個の引数をコピーします。

本標準インクルードファイルで定義しているマクロを使用したプログラムの例を以下に示します。

```
1           #include <stdio.h>
2           #include <stdarg.h>
3
4           extern void prlist(int count,...);
5
6           void main()
7           {
8              prlist(1, 1);
9              prlist(3, 4, 5, 6);
10             prlist(5, 1, 2, 3, 4, 5);
11          }
12
13          void prlist(int count,...)
14          {
15             va_list ap;
16             int i;
17
18             va_start(ap,count);
19             for(i=0;i<count;i++)
20                 printf("%d",va_arg(ap,int));
21             putchar('\n');
22             va_end(ap);
23          }
```

【説明】

この例では、第 1 引数に出力するデータの数を指定し、以下の引数をその数だけ出力する関数 prlist を実現しています。

18 行目で、可変個の引数への参照を va_start で初期化します。その後引数を一つ出力するたびに、va_arg マクロによって次の引数を参照します(20 行目)。va_arg マクロでは、引数の型名(この場合は int 型)を第 2 引数に指定します。

引数の参照が終了したら、va_end マクロを呼び出します(22 行目)。

可変個引数取り出し開始

void va_start(va_list ap, parmN)

説　明　　可変個の実引数への参照を行うため、初期設定処理を行います。

ヘッダ　　<stdarg.h>

引　数

ap	可変個の引数にアクセスするための変数
parmN	最右端の引数の識別子

例

```
#include <stdarg.h>
void func(int count,...)
{
    va_list ap;
      va_start(ap,count);
}
```

備　考　　va_start マクロは、va_arg、va_end マクロによって使用される ap の初期化を行います。
また、parmN には、外部関数定義における引数の並びの最右端の引数の識別子、すなわち「,...」の直前の識別子を指定します。
可変個の名前のない引数を参照するためには、va_start マクロ呼び出しを一番初めに実行する必要があります。

可変個引数取り出し

type va_arg(va_list ap, type)

説　明　　可変個の実引数を持つ関数に対して、現在参照中の引数の次の引数への参照を可能とします。

ヘッダ　　<stdarg.h>

リターン値　　引数の値

引　数

ap	可変個の引数にアクセスするための変数
type	アクセスする引数の型

例

```
#include <stdarg.h>
va_list ap;
int ret;
    ret=va_arg(ap,int);
```

備　考　　va_start マクロで初期化した va_list 型の変数を第 1 引数に指定します。
ap の値は va_arg を使用する度に更新され、結果として可変個の引数が順次本マクロのリターン値として返されます。
第 2 引数 type は、参照する型を指定してください。
ap は va_start によって初期化された ap と同じでなければなりません。
type に char 型、unsigned char 型、short 型、unsigned short 型、float 型のように関数の引数に指定した時に型変換によってサイズが変わる型を指定した場合、正しく引数を参照することができなくなります。このような型を指定すると動作は保証しません。

可変個引数取り出し終了

void va_end(va_list ap)

説　明　　可変個の実引数を持つ関数の引数への参照を終了させます。

ヘッダ　　<stdarg.h>

引　数　　ap　　可変個の引数を参照するための変数

例
```
#include <stdarg.h>
va_list ap;
    va_end(ap);
```

備　考　　ap は va_start によって初期化された ap と同じでなければなりません。
また、関数からの return 前に va_end マクロが呼び出されない時は、その関数の動作は保証しません。

可変個引数のコピー

void va_copy(va_list dest, va_list src)

説　明　　可変個の実引数を持つ関数に対して、現在参照中の引数の複製を作ります。

ヘッダ　　<stdarg.h>

引　数　　dest　　可変個の引数を参照するための変数の複製
src　　可変個の引数を参照するための変数

例
```
#include <stdarg.h>
va_list ap, ap_sub;
    va_copy(ap_sub,ap);
```

備　考　　va_start マクロで初期化され、va_arg で使用された可変個の引数の状態を持つ第 2 引数 src に対し、第 1 引数 dest に複製を作ります。
src は va_start によって初期化された src と同じでなければなりません。
dest は、これ以降の va_arg マクロで可変個の引数を表す引数として使用することができます。

（12）<stdio.h>

ストリーム入出力用ファイルの入出力に関する処理を行います。

以下の定数(マクロ)はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	FILE	ストリーム入出力処理で必要とするバッファへのポインタやエラー指示子、終了指示子などの各種制御情報を保存しておく構造体の型です。
	_IOFBF	バッファ領域の使用方法として、入出力処理はすべてバッファ領域を使用することを示しています。
	_IOLBF	バッファ領域の使用方法として、入出力処理は行単位でバッファ領域を使用することを示しています。
	_IONBF	バッファ領域の使用方法として、入出力処理はバッファ領域を使用しないことを示しています。
	BUFSIZ	入出力処理において必要とするバッファの大きさです。
	EOF	ファイルの終わり(End Of File)すなわちファイルからそれ以上の入力が無いことを示しています。
	L_tmpnam*	tmpnam 関数によって生成される一時ファイル名の文字列を格納するのに十分な大きさの配列のサイズです。
	SEEK_CUR	ファイルの現在の読み書き位置を現在の位置からのオフセットに移すことを示しています。
	SEEK_END	ファイルの現在の読み書き位置をファイルの終了位置からのオフセットに移すことを示しています。
	SEEK_SET	ファイルの現在の読み書き位置をファイルの先頭位置からのオフセットに移すことを示しています。
	SYS_OPEN*	処理系が同時にオープンすることができることを保証するファイルの数です。
	TMP_MAX*	tmpnam 関数によって生成される一意なファイル名の個数の最大値です。
	stderr	標準エラーファイルに対するファイルポインタです。
	stdin	標準入力ファイルに対するファイルポインタです。
	stdout	標準出力ファイルに対するファイルポインタです。
関数	fclose	ストリーム入出力用ファイルをクローズします。
	fflush	ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。
	fopen	ストリーム入出力用ファイルを指定したファイル名によってオープンします。
	freopen	現在オープンされているストリーム入力出用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。
	setbuf	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

種別	定義名	説明
関数	setvbuf	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
	fprintf	書式に従ってストリーム入出力用ファイルへデータを出力します。
	vfprintf	可変個の引数リストを書式に従って指定したストリーム入出力用ファイルに出力します。

【注】 * 本処理系では、定義されません。

種別	定義名	説明
関数	printf	データを書式に従って変換し、標準出力ファイル(stdout)へ出力します。
	vprintf	可変個の引数リストを書式に従って標準出力ファイル(stdout)に出力します。
	sprintf	データを書式に従って変換し、指定した領域へ出力します。
	sscanf	指定した記憶域からデータを入力し、書式に従って変換します。
	snprintf	データを書式に従って変換し、配列に書き込みます。
	vsnprintf	可変個数の実引数並びを va_list で置き換えた snprintf と等価です。
	vfscanf	可変個数の実引数並びを va_list で置き換えた fscanf と等価です。
	vscanf	可変個数の実引数並びを va_list で置き換えた scanf と等価です。
	vsscanf	可変個数の実引数並びを va_list で置き換えた sscanf と等価です。
	fscanf	ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。
	scanf	標準入力ファイル(stdin)からデータを入力し、書式に従って変換します。
	vsprintf	可変個の引数リストを書式に従って指定した領域に出力します。
	fgetc	ストリーム入出力用ファイルから 1 文字入力します。
	fgets	ストリーム入出力用ファイルから文字列を入力します。
	fputc	ストリーム入出力用ファイルへ 1 文字出力します。
	fputs	ストリーム入出力用ファイルへ文字列を出力します。
	getc	(マクロ) ストリーム入出力用ファイルから 1 文字入力します。
	getchar	(マクロ) 標準入力ファイルから 1 文字入力します。
	gets	標準入力ファイルから文字列を入力します。
	putc	(マクロ) ストリーム入出力用ファイルへ 1 文字出力します。
	putchar	(マクロ) 標準出力ファイルへ 1 文字出力します。
	puts	標準出力ファイルへ文字列を出力します。
	ungetc	ストリーム入出力用ファイルへ 1 文字をもどします。
	fread	ストリーム入出力用ファイルから指定した記憶域にデータを入力します。
	fwrite	記憶域からストリーム入出力用ファイルにデータを出力します。
	fseek	ストリーム入出力用ファイルの現在の読み書き位置を移動させます。

種別	定義名	説明
関数	ftell	ストリーム入出力用ファイルの現在の読み書き位置を求めます。
	rewind	ストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
	clearerr	ストリーム入出力用ファイルのエラー状態をクリアします。
	feof	ストリーム入出力用ファイルが終わりであるかどうかを判定します。
	ferror	ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
	perror	標準エラーファイル(stderr)に、エラー番号に対応したエラーメッセージを出力します。
型	fpos_t	ファイル中の任意の位置を指定可能な型です。
定数(マクロ)	FOPEN_MAX	同時にオープン可能なファイル数です。
	FILENAME_MAX	保持可能なファイル名の最大長です。

処理系定義仕様

	項目	コンパイラの仕様
1	入力テキストの最終の行が終了を示す改行文字を必要とするかどうか	規定しません。低水準インタフェースルーチンの仕様によります。
2	改行文字の直前にかき出された空白文字は、読み込み時に読み込まれるかどうか	
3	バイナリファイルに書かれたデータに付加されるヌル文字の数	
4	追加モード時のファイル位置指定子の初期値	
5	テキストファイルへの出力によってそれ以降のファイルのデータが失われるかどうか	
6	ファイルバッファリングの仕様	
7	長さ 0 のファイルが存在するかどうか	
8	正当なファイル名の構成規則	
9	同時に同じファイルをオープンできるかどうか	
10	fprintf 関数における%p 書式変換の出力形式	16 進数出力となります
11	fscanf 関数における%p 書式変換の入力形式	16 進数入力となります。
	fscanf 関数での変換文字「－」の意味	先頭、最後あるいは「^」の直後でない場合、直前の文字と直後の範囲を示します。
12	fgetpos,ftell 関数で設定される errno の値	fgetpos 関数はサポートしていません。 ftell 関数については規定しません。 低水準インタフェースルーチンの仕様によります。
13	perror 関数が生成するメッセージ出力形式	メッセージの出力形式を(a)に示します。

(a) perror関数の出力形式は、

<文字列>：<errorに設定したエラー番号に対応するエラーメッセージ>

となります。

(b) printf関数、fprintf関数で、浮動小数点の無限大および非数を表示するときの形式を**表**9.31に示します。

表 9.31 **無限大および非数の表示形式**

	値	表示形式
1	**正の無限大**	＋＋＋＋＋＋
2	**負の無限大**	－－－－－－
3	**非数**	＊＊＊＊＊＊

ストリーム入出力用ファイルに対する一連の入出力処理を行ったプログラムの例を以下に示します。

```
1           #include <stdio.h>
2
3           void main()
4           {
5               int c;
6               FILE *ifp, *ofp;
7
8               if ((ifp=fopen("INPUT.DAT","r"))==NULL){
9                   fprintf(stderr,"cannot open input file\n");
10                  exit(1);
11              }
12              if ((ofp=fopen("OUTPUT.DAT","w"))==NULL){
13                  fprintf(stderr,"cannot open output file\n");
14                  exit(1);
15              }
16              while ((c=getc(ifp))!=EOF)
17                putc(c, ofp);
18              fclose(ifp);
19              fclose(ofp);
20          }
```

【説明】

ファイル INPUT.DAT の内容をファイル OUTPUT.DAT へコピーするプログラムです。

8 行目の fopen 関数で入力ファイル INPUT.DAT を、12 行目の fopen 関数で出力ファイル OUTPUT.DAT をオープンします。オープンに失敗した場合、fopen 関数のリターン値として NULL が返されますので、エラーメッセージを出力してプログラムを終了させます。

fopen 関数が正常に終了した時、オープンしたファイルの情報を格納するデータ(FILE 型)へのポインタが返されますので、これらを変数 ifp、ofp に設定します。

オープンが成功した後は、これらの FILE 型のデータを用いて入出力を行います。

ファイルの処理が終了したら、fclose 関数でファイルをクローズします。

ファイルクローズ

*long fclose(FILE *fp)*

説　明　ストリーム入出力用ファイルをクローズします。

ヘッダ　<stdio.h>

リターン値　正常：0
異常：0 以外

引　数　fp　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
int ret;
   ret=fclose(fp);
```

備　考　fclose 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルをクローズします。
fclose 関数は、ストリーム入出力用ファイルの出力ファイルがオープンされており、まだ出力されていないデータがバッファに残っている時は、それをファイルに出力してからクローズします。
また、入出力用のバッファがシステムによって自動的に割り付けられていた場合は、それを解放します。

バッファフラッシュ

*long fflush(FILE *fp)*

説　明　ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。

ヘッダ　<stdio.h>

リターン値　正常：0
異常：0 以外

引　数　fp　ファイルポインタ

例
```
#include  <stdio.h>
FILE *fp;
int ret;
   ret=fflush(fp);
```

備　考　fflush 関数は、ストリーム入出力用ファイルの出力ファイルがオープンされている時、ファイルポインタ fp で指定されたストリーム入出力用ファイルのバッファの未出力内容をファイルに出力します。また、入力ファイルがオープンされている時、ungetc 関数の指定を無効にします。

ファイルオープン

*FILE *fopen(const char *fname, const char *mode)*

説　明　ストリーム入出力用ファイルを、指定したファイル名によってオープンします。

ヘッダ　<stdio.h>

リターン値　正常：オープンしたファイルのファイル情報を指すファイルポインタ
異常：NULL

引　数

fname	ファイル名を示す文字列へのポインタ
mode	ファイルアクセスモードを示す文字列へのポインタ

例

```
#include <stdio.h>
FILE *ret;
const char *fname, *mode;
   ret=fopen(fname,mode);
```

備　考　fopen 関数は、fname が指す文字列をファイル名とするストリーム入出力用ファイルをオープンします。書き出しモードあるいは追加モードで存在しないファイルをオープンしようとした時は、可能な限り新しいファイルを作成します。また既存のファイルに対して書き出しモードでオープンした時は、ファイルの先頭から書き込みが行われ、以前に書き込まれていたファイルの内容は消去されます。
追加モードでオープンしたファイルは、そのファイルの終わりの位置から書き出しの処理が行われます。更新モードでオープンしたファイルは、このファイルに対して入力と出力の両方の処理を行うことができます。
ただし、出力処理は後に fflush、fseek、rewind 関数が実行されることなしに入力処理を続けることはできません。
また同様に入力処理の後に fflush、fseek、rewind 関数が実行されることなしに出力処理を続けることはできません。
また、ファイルアクセスモードを示す文字列の後にオープンの方法を指示する文字が付くこともあります。

ファイル再オープン

*FILE *freopen(const char *fname, const char *mode, FILE *fp)*

説　明　現在オープンされているストリーム入出力用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。

ヘッダ　<stdio.h>

リターン値　正常：fp
異常：NULL

引　数

fname	新しいファイル名を示す文字列へのポインタ
mode	ファイルアクセスモードを示す文字列へのポインタ
fp	現在オープンされているストリーム入出力用ファイルのファイルポインタ

例

```
#include <stdio.h>
const char *fname, *mode;
FILE *ret, *fp;
   ret=freopen(fname,mode,fp);
```

備　考　freopen 関数は、まず、ファイルポインタ fp の示すストリーム入出力用ファイルをクローズします(このクローズ処理が正しく行われない時でも以下の処理は続けます)。
次に、その fp の指す FILE 構造体を再使用して、ファイル名 fname で示すファイルを、ストリーム入出力用にオープンします。
freopen 関数は一時にオープンするファイル数が限られているときなどに有効です。
freopen 関数は通常、fp と同じ値を返しますが、エラーが発生した時は、NULL を返します。

バッファ領域設定

*void setbuf(FILE *fp, char buf[BUFSIZ])*

説　明　　ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

ヘッダ　　<stdio.h>

引　数

fp	ファイルポインタ
buf	バッファ領域へのポインタ

例

```
#include <stdio.h>
FILE *fp;
char buf[BUFSIZ];
   setbuf(fp,buf);
```

備　考　　setbuf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対して buf の指す記憶域を入出力用のバッファ領域として使用するように定義します。この結果、大きさが BUFSIZ のバッファ領域を使用した入出力処理が行われます。

バッファリング制御

*long setvbuf(FILE *fp, char *buf, long type, size_t size)*

説　明　　ストリーム入出力用のバッファ領域をユーザプログラムの側で定義して設定します。

ヘッダ　　<stdio.h>

リターン値　　正常：0
異常：0 以外

引　数

fp	ファイルポインタ
buf	バッファ領域へのポインタ
type	バッファの管理方式
size	バッファ領域の大きさ

例

```
#include <stdio.h>
FILE *fp;
char *buf;
int type, ret;
size_t size;
   ret=setvbuf(fp,buf,type,size);
```

備　考　　setvbuf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対して buf の指す記憶域を入出力用のバッファ領域として使用するように定義します。
このバッファ領域の使用方法としては、以下の三通りの方法があります。

(a) type に_IOFBF を指定した時
入出力処理はすべてバッファ領域を使用して行います。

(b) type に_IOLBF を指定した時
入出力処理は行単位でバッファ領域を使用して行います。すなわち、入出力データは、改行文字が書かれた時、バッファ領域が一杯になった時、入力が要求された時にバッファ領域から取り出されることになります。

(c) type に_IONBF を指定した時
入出力処理はバッファ領域を使用せず行います。

setvbuf 関数は通常 0 を返しますが、type あるいは size に不正な値が与えられた時、あるいはバッファ領域の使用方法等の要求が受け入れられなかった時には 0 以外の値を返します。

バッファ領域は、オープンされているストリーム入出力用ファイルがクローズされる前に解放してはいけません。また、setvbuf 関数は、ストリーム入出力用ファイルがオープンされてから入出力用処理が行われるまでの間で使用してください。

書式付きファイル出力

*long fprintf(FILE *fp, const char *control [, arg] …)*

説　明　　書式に従って、ストリーム入出力用ファイルへデータを出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

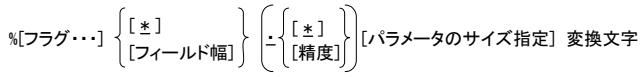
fp	ファイルポインタ
control	書式を示す文字列へのポインタ
arg,…	書式に従って出力されるデータの並び

例

```
#include <stdio.h>
FILE *fp;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=fprintf(fp,control,buffer);
```

備　考　　fprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、ファイルポインタ fp の示すストリーム入出力用ファイルへ出力します。
fprintf 関数は、通常は変換し出力したデータの個数を返しますが、エラー発生時には負の値を返します。
書式の仕様は以下のとおりです。

【書式の概要】
書式を表す文字列は、2 種類の文字列から構成されます。
・通常の文字
次の変換仕様を示す文字列以外の文字はそのまま出力されます。
・変換仕様
変換仕様は、%で始まる文字列で、後に続く引数の変換方法を指定します。変換仕様の形式は次の規則に従います。



この変換仕様に対して、実際に出力する引数が無い時は、その動作は保証しません。また、変換仕様よりも実際に出力する引数の個数が多い時は、余分な引数はすべて無視されます。

【変換仕様の説明】
(a) フラグ
符号を付けるなどの出力するデータに対する修飾を指定します。指定できるフラグの種類と意味を**表** 9.32 に示します。

表 9.32 フラグの種類と意味

	種類	意味
1	-	変換したデータの文字数が指定したフィールド幅より少ない時、そのデータをフィールド内で左詰めにして出力します。
2	+	符号付きのデータに変換する時、そのデータの符号に従って、変換したデータの先頭にプラスあるいはマイナス符号を付けます。
3	空白	符号付きのデータの変換において、変換したデータの先頭に符号が付かない時、そのデータの先頭に空白を付けます。 「+」と共に使用した時、本フラグは無視されます。
4	#	表 9.34 で説明する変換の種類に従って、変換後のデータに修飾を行います。 1. c，d，i，s，u 変換の時 本フラグは無視されます。 2. o 変換の時 変換したデータの先頭に 0 を付けます。 3. x(あるいは X)変換の時 変換したデータの先頭に 0x(あるいは 0X)を付けます。 4. e，E，f，g，G 変換の時 変換したデータに小数点以下がない時でも、小数点を出力します。 また、g，G 変換の時は、変換後のデータの後に付く 0 は取り除きません。

(b) フィールド幅
変換したデータを出力する文字数を任意の 10 進数で指定します。
変換したデータの文字数がフィールド幅より少ない時、フィールド幅までそのデータの先頭に空白が付けられます(ただし、フラグとして'-'を指定した時は、データの後に空白が付けられます)。
もし、変換したデータの文字数がフィールド幅より大きい時は、フィールド幅は、変換結果を出力できる幅に拡張されます。
また、フィールド幅指定の先頭が 0 で始まっている時は、出力するデータの先頭には空白ではなく文字「0」が付けられます。

(c) 精度
表 9.34 で説明する変換の種類に従って変換したデータの精度を指定します。
精度は、ピリオド(.)の後に 10 進整数を続ける形式で指定します。10 進整数を省略した時は、0 を指定したものと仮定します。
精度を指定した結果、フィールド幅の指定との間に矛盾が生じれば、フィールド幅の指定を無効とします。
各変換の種類と精度指定の意味を以下に示します。
・d，i，o，u，x，X 変換の時
　変換したデータの最小の桁数を示します。
・e，E，f 変換の時
　変換したデータの小数点以下の桁数を示します。
・g，G 変換の時
　変換したデータの最大有効桁数を示します。
・s 変換の時
　印字される最大文字数を示します。

(d) パラメータのサイズ指定
d, i, o, u, x, X, e, E, f, g, G 変換の時(**表** 9.34 参照)
変換するデータのサイズ(short 型、long 型、long long 型、long double 型)を指定します。これ以外の変換の時は、本指定を無視します。**表** 9.33 にサイズ指定の種類とその意味を示します。

表 9.33 **パラメータのサイズ指定の種類とその意味**

	種類	意味
1	h	d，i，o，u，x，X **変換において、変換するデータが** short **型あるいは** unsigned short **型であることを指定します。**
2	l	d，i，o，u，x，X **変換において、変換するデータが** long **型、**unsigned long **型あるいは、**double **型であることを指定します。**
3	L	e，E，f，g，G **変換において、変換するデータが** long double **型であることを指定します。**
4	ll	d，i，o，u，x，X **変換において、変換するデータが** long long **型あるいは、**unsigned long long **型であることを指定します。**n **変換において、変換するデータが** long long **型へのポインタ型であることを指定します。**

(e) 変換文字
変換するデータをどのような形式に変換するかを指定します。
もし、変換するデータが構造体や配列型の時や、それらの型を指すポインタの時は、s 変換で文字の配列を変換する時、p 変換でポインタを変換する時を除いてその動作は保証しません。**表** 9.34 に変換文字と変換方式を示します。この表に述べられていない英文字を変換文字として指定した時は、その動作は保証しません。また、それ以外の文字を指定した時の動作はコンパイラによって異なります。

表 9.34　変換文字と変換の方式

	変換文字	変換の種類	変換の方式	変換の対象とするデータの型	精度に対する注意事項
1	d	d 変換	int 型データを符号付き 10 進数の文字列に変換します。d 変換と i 変換は同じ仕様です。	int 型	精度指定は、最低で何文字出力されるかを示しています。もし、変換後の文字数が精度の値より少ない時は、文字列の先頭に 0 が付きます。また、精度を省略した時は、1 が仮定されます。さらに、0 の値を持つデータを精度に 0 を指定して変換し出力しようとした時は、何も出力されません。
2	i	i 変換		int 型	
3	o	o 変換	int 型データを符号なしの 8 進数の文字列に変換します。	int 型	
4	u	u 変換	int 型データを符号なしの 10 進数の文字列に変換します。	int 型	
5	x	x 変換	int 型データを符号なしの 16 進数に変換します。16 進文字には a，b，c，d，e，f を用います。	int 型	
6	X	X 変換	int 型データを符号なしの 16 進数に変換します。16 進文字には A，B，C，D，E，F を用います。	int 型	
7	f	f 変換	double 型データを「[-]ddd.ddd」の形式の 10 進数の文字列に変換します。	double 型	精度の指定は、小数点以降の桁数を表します。小数点以降の文字が存在する時には、必ず小数点の前に 1 桁の数字が出力されます。精度を省略した時は、6 が仮定されます。また、精度に 0 を指定した時は、小数点と小数点以降の文字は出力しません。出力するデータは丸められます。
8	e	e 変換	double 型データを「[-]d.ddde±dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	精度の指定は、小数点以降の桁数を表します。変換した文字は小数点の前に 1 桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される形式となります。精度を省略した時は 6 が仮定されます。また、精度に 0 を指定した時は、小数点以降の文字は出力しません。出力するデータは丸められます。
9	E	E 変換	double 型データを「[-]d.dddE±dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	

	変換文字	変換の種類	変換の方式	変換の対象とするデータの型	精度に対する注意事項
10	g	g 変換(あるいは G 変換)	変換する値と有効桁数を指定する精度の値から f 変換の形式で出力するか e 変換(あるいは E 変換)の形式で出力するかを決め double 型データを出力します。もし、変換されたデータの指数が-4 より小さいか、有効桁数を指定する精度より大きい時には e 変換(あるいは E 変換)の形式に変換します。	double 型	精度の指定は、変換されたデータの最大有効桁数を示します。
11	G			double 型	
12	c	c 変換	int 型のデータを unsigned char 型データとし、そのデータに対応する文字に変換します。	int 型	精度の指定は無効です。
13	s	s 変換	char 型へのポインタ型データが指す文字列を文字列の終了を示すヌル文字まで、あるいは、精度で指定された文字数分出力します(ただしヌル文字は出力されません。また、空白、水平タブ、改行文字は変換文字列に含まれません)。	char 型へのポインタ型	精度の指定は出力する文字数を示します。もし、精度が省略された時は、データが指す文字列のヌル文字までの文字が出力されます(ただし、ヌル文字は出力されません。また、空白、水平タブ、改行文字は変換文字列に含まれません)。
14	p	p 変換	データをポインタとして、コンパイラごとに定義された印字可能な文字列に変換します。	void 型へのポインタ	精度の指定は無効です。
15	n	データの変換は生じません。	データは int 型へのポインタ型とみなされ、このデータが指す記憶域にいままで、出力したデータの文字数を設定します。	int 型へのポインタ型	
16	%	データの変換は生じません。	%を出力します。	なし	

(f) フィールド幅あるいは精度に対する*指定
フィールド幅あるいは精度指定の値として*を指定することができます。この時は、この変換仕様に対応するパラメータの値がフィールド幅あるいは精度指定の値として使用されます。このパラメータが負のフィールド幅を持つ時は、正のフィールド幅にフラグ-が指定されたと解釈します。また、負の精度を持つ時は、精度が省略されたものと解釈します。

書式付き文字列出力

*long snprintf(char *restrict s, size_t n, const char *restrict control [, arg] …)*

説　明　　データを書式に従って変換し、指定した領域へ出力します。

ヘッダ　　<stdio.h>

リターン値　　変換した文字数

引　数

s	データを出力する記憶域へのポインタ
n	出力する文字数
control	書式を示す文字列へのポインタ
arg,…	書式に従って出力されるデータ

例

```
#include <stdio.h>
char *s;
size_t n;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
ret=snprintf(s,n,control,buffer);
```

備　考　　snprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、s の指す記憶域へ出力します。
変換して出力した文字列の最後には、ヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。書式の仕様の詳細は fprintf 関数を参照してください。

可変個引数書式付き文字列出力

*long vsnprintf(char *restrict s, size_t n, const char *restrict control, va_list arg)*

説　明　　データを書式に従って変換し、指定した領域へ出力します。

ヘッダ　　<stdarg.h>, <stdio.h>

リターン値　　変換した文字数

引　数

s	データを出力する記憶域へのポインタ
n	出力する文字数
control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
char *s;
size_t n;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++) {
      ret=vsnprintf(s,control,ap);
      va_arg(ap,int);
      s += ret;
    }
}
```

備　考　　vsnprintf 関数は，可変個引数を arg で置き換えた snprintf と等価です。
vsnprintf 関数の呼出し前に，va_start マクロで arg を初期化してください。
vsnprintf 関数は，va_end マクロを呼び出しません。

書式付きファイル入力

*long fscanf(FILE *fp, const char *control [, ptr] …)*

説　明　　ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

ヘッダ　　<stdio.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

fp	ファイルポインタ
control	書式を示す文字列へのポインタ
ptr,…	入力したデータを格納する記憶域へのポインタ

例

```
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret,buffer[10];
   ret=fscanf(fp,control,buffer);
```

備　考　　fscanf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルからデータを入力し、control が指す書式を文字列に従って変換、編集して、その結果を ptr の指す記憶域へ格納します。
データを入力するための書式の仕様を以下に示します。

【書式の概要】
書式を表す文字列は、以下の 3 種類の文字列から構成されます。
・空白文字
空白(' ')、水平タブ('\t')あるいは改行文字('\n')を指定すると、入力データを次の空白類文字でない文字まで読み飛ばす処理を行います。
・通常の文字
上の空白文字でも%でもない文字を指定すると、入力データを 1 文字入力します。ここで入力した文字は書式を表す文字列の中に指定した文字と一致していなければなりません。
・変換仕様
変換仕様は、%で始まる文字列で、書式を表す文字列の後に続く引数の指す領域に入力データを変換して格納する方法を指定します。変換仕様の形式は次の規則に従います。

　　%[*][フィールド幅][変換後のデータのサイズ]変換文字

書式中の変換仕様に対して入力したデータを格納する記憶域へのポインタがない時は、その動作は保証しません。また、書式が終了したにもかかわらず、入力データを格納する記憶域へのポインタが残っている時は、そのポインタは無視されます。

【変換仕様の説明】
・*指定
入力したデータを引数が指す記憶域に格納することを抑止します。
・フィールド幅
入力するデータの最大文字数を 10 進数字で指定します。
・変換後のデータのサイズ
d, i, o, u, x, X, e, E, f 変換の時(**表** 9.36 参照)、変換後のデータのサイズ(short 型、long 型、long long 型、long double 型)を指定します。これ以外の変換の時は、本指定を無視します。**表** 9.35 にサイズ指定の種類とその意味を示します。

表 9.35 変換後のデータのサイズ指定の種類とその意味

	種類	意味
1	h	d，i，o，u，x，X 変換において、変換後のデータは short 型であることを指定します。
2	l	d，i，o，u，x，X 変換において、変換後のデータは long 型であることを指定します。 また、e，E，f 変換において、変換後のデータは double 型であることを指定します。
3	L	e，E，f 変換において、変換後のデータは、long double 型であることを指定します。
4	ll	d，i，o，u，x，X 変換において、変換後のデータは long long 型であることを指定します。

・変換文字
入力するデータは、各変換文字が指定する変換の種類に従って変換します。ただし、空白類文字を読み込んだ場合、変換の対象として許されていない文字を読み込んだ場合、あるいは指定されたフィールド幅を超えた場合は処理を終了します。

表 9.36 変換文字と変換の内容

	変換文字	変換の種類	変換の方式	対応するパラメータの型名
1	d	d 変換	10 進数字の文字列を整数型データに変換します。	整数型
2	i	i 変換	先頭に符号が付いている 10 進数字の文字列、あるいは最後に u(U) または l(L)が付いている 10 進数字の文字列を整数型データに変換します。また、先頭が 0x(あるいは 0X)で始まっている文字列は、16 進数字として解釈し、文字列を int 型データに変換します。さらに、先頭が 0 で始まっている文字列は、8 進数字として解釈し文字列を int 型データに変換します。	整数型
3	o	o 変換	8 進数字の文字列を整数型データに変換します。	整数型
4	u	u 変換	符号なしの 10 進数字の文字列を整数型データに変換します。	整数型
5	x	x 変換	16 進数字の文字列を整数型データに変換します。	整数型
6	X	X 変換	x 変換と X 変換に意味の違いはありません。	
7	s	s 変換	空白、水平タブ、改行文字を読み込むまでをひとつの文字列として変換します。文字列の最後にはヌル文字を付加します(変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です)。	文字型
8	c	c 変換	1 文字を入力します。この時、入力する文字が空白類文字であっても読み飛ばすことはしません。もし、空白類文字以外の文字だけを読み込む時は、%1s と指定してください。また、フィールド幅が指定されている時は、その指定分の文字が読み込まれます。したがって、この時、変換したデータを格納する記憶域は、指定分の大きさが必要です。	char 型
9	e	e 変換	浮動小数点型を示す文字列を浮動小数点型データに変換します。e 変換と E 変換、g 変換と G 変換にそれぞれ意味の違いはありません。入力形式は strtod 関数で表現できる浮動小数点型です。	浮動小数点型
10	E	E 変換		
11	f	f 変換		
12	g	g 変換		
13	G	G 変換		
14	p	p 変換	fprintf 関数において、p 変換で変換される形式の文字列をポインタ型データに変換します。	void 型へのポインタ型
15	n	データの変換は生じません。	データの入力は行わず、いままでに入力したデータの文字数が設定されます。	整数型

	変換文字	変換の種類	変換の方式	対応するパラメータの型名
16	[	[変換	[の後に文字の集合、その後に]を指定します。この文字集合は、文字列を構成する文字の集合を定義しています。もし、文字集合の最初の文字が ^ でない時は、入力データはこの文字集合にない文字が最初に読み込まれるまでをひとつの文字列として入力します。もし、最初の文字が ^ の時は、 ^ を除いた文字集合の文字が最初に読み込まれるまでをひとつの文字列として入力します。入力した文字列の最後には自動的にヌル文字を付加します(変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です)。	文字型
17	%	データの変換は生じません。	%を読み込みます。	なし

変換文字が**表** 9.36 に示す文字以外の英文字の時は、その動作は保証しません。また、その他の文字の時は、その動作は処理系定義です。

書式付き出力

*long printf(const char *control [, arg] …)*

説　明　　データを書式に従って変換し、標準出力ファイル(stdout)へ出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

control	書式を示す文字列へのポインタ
arg,…	書式に従って出力されるデータ

例

```
#include <stdio.h>
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=printf(control,buffer);
```

備　考　　printf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、標準出力ファイル(stdout)へ出力します。
書式の仕様の詳細は fprintf 関数を参照してください。

可変個引数書式付きファイル入力

*long vfscanf(FILE *restrict fp, const char *restrict control , va_list arg)*

説　明　　ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <stdio.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

fp	ファイルポインタ
control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>

FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfscanf(fp, control, ap);
   va_end(ap);
}
```

備　考　　vfscanf 関数は可変個引数並びを arg で置き換えた fscanf と等価です。
vfscanf 関数の呼出し前に，va_start マクロで arg を初期化してください。
vfscanf 関数は va_end マクロを呼び出しません。

書式付き入力

*long scanf(const char *control [, ptr] …)*

説　明　標準入力ファイル(stdin)からデータを入力し、書式に従って変換します。

ヘッダ　<stdio.h>

リターン値　正常：入力変換に成功したデータの個数
異常：EOF

引　数

control	書式を示す文字列へのポインタ
ptr,…	入力変換したデータを格納する記憶域へのポインタ

例

```
#include <stdio.h>
const char *control="%d";
int ret,buffer[10];
   ret=scanf(control, buffer);
```

備　考　scanf 関数は、標準入力ファイル(stdin)からデータを入力し、control が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を ptr の指す記憶域へ格納します。
scanf 関数は、入力変換に成功したデータの個数をリターン値として返します。最初の変換の前に標準入力ファイルが終了した時には EOF を返します。
書式の仕様の詳細は fscanf 関数を参照してください。
%e 変換では、double 型の場合は l、long double 型の場合は L で指定します。デフォルトの型は float 型です。

可変個引数書式付きファイル入力

*long vscanf(const char *restrict control , va_list arg)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <stdio.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>

FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vscanf(control, ap);
   va_end(ap);
}
```

備　考　　vscanf 関数は，可変個数引数を arg で置き換えた scanf と等価です。
vscanf 関数の呼出し前に，va_start マクロで arg を初期化してください。
vscanf 関数は va_end マクロを呼びません。

書式付き文字列出力

*long sprintf(char *s, const char *control [, arg] …)*

説　明　　データを書式に従って変換し、指定した領域へ出力します。

ヘッダ　　<stdio.h>

リターン値　　変換した文字数

引　数

s	データを出力する記憶域へのポインタ
control	書式を示す文字列へのポインタ
arg,…	書式に従って出力されるデータ

例

```
#include <stdio.h>
char *s;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=sprintf(s,control,buffer);
```

備　考　　sprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、s の指す記憶域へ出力します。
変換して出力した文字列の最後には、ヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。
書式の仕様の詳細は fprintf 関数を参照してください。

書式付き文字列入力

*long sscanf(const char *s, const char *control [, ptr] …)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdio.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：EOF

引　数

s	入力するデータがある記憶域
control	書式を示す文字列へのポインタ
ptr,…	入力変換したデータを格納する記憶域へのポインタ

例

```
#include <stdio.h>
const char *s, *control="%d";
int ret,buffer[10];
   ret=sscanf(s,control,buffer);
```

備　考　　sscanf 関数は、s の指す記憶域からデータを入力し、control が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を ptr の指す記憶域へ格納します。
sscanf 関数は、入力変換に成功したデータの個数を返します。また、最初の変換の前に入力するデータが終了した時には EOF を返します。
書式の仕様の詳細は fscanf 関数を参照してください。

可変個引数書式付きファイル入力

*long vsscanf(const char *restrict s, const char *restrict control , va_list arg)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <stdio.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

s	入力するデータがある記憶域
control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>

const char *s, *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vsscanf(control, ap);
   va_end(ap);
}
```

備　考　　vsscanf 関数は，可変個数引数を arg で置き換えた sscanf と等価です。
vsscanf 関数の呼出し前に，va_start マクロで arg を初期化してください。
vsscanf 関数は va_end マクロを呼びません。

可変個引数ファイル出力

*long vfprintf(FILE *fp, const char *control, va_list arg)*

説　明　　可変個の引数リストを書式に従って、指定したストリーム入出力用ファイルに出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

fp	ファイルポインタ
control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control="%d";
int  ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfprintf(fp, control, ap);
   va_end(ap);
}
```

備　考　　vfprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、fp の示すストリーム入出力用ファイルへ出力します。
vfprintf 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。
また、vfprintf 関数では va_end マクロは呼び出しません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ)によって初期化されていなければなりません。

*long vprintf(const char *control, va_list arg)*

説　明　　可変個の引数リストを書式に従って標準出力ファイル(stdout)に出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vprintf(control, ap);
   va_end(ap);
}
```

備　考　　vprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、標準出力ファイルへ出力します。
vprintf 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。
また、vprintf 関数では va_end マクロは呼び出しません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ)によって初期化されていなければなりません。

可変個引数文字列出力

*long vsprintf(char *s, const char *control, va_list arg)*

説　明　　可変個の引数リストを書式に従って、指定した記憶域に出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：変換した文字数
異常：負の数

引　数

s	データを出力する記憶域へのポインタ
control	書式を示す文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
char *s;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++) {
      ret=vsprintf(s,control,ap);
      va_arg(ap,int);
      s += ret;
    }
}
```

備　考　　vsprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、s により指される記憶域へ出力します。
変換して出力した文字列の最後にヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ)によって初期化されていなければなりません。

ファイルから 1 文字入力

*long fgetc(FILE *fp)*

説　明	ストリーム入出力用ファイルから 1 文字入力します。
ヘッダ	<stdio.h>
リターン値	正常： ファイルの終了の時　　：EOF ファイルの終了でない時 ：入力した文字 異常： EOF
引　数	fp　　ファイルポインタ
例	#include <stdio.h> FILE *fp; int ret; ret=fgetc(fp);
エラー条件	読み込みエラーが発生した時、そのファイルに対してのエラー指示子が設定されます。
備　考	fgetc 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから 1 文字入力します。 fgetc 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の時は、EOF を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

ファイルから文字列入力

*char *fgets(char *s, long n, FILE *fp)*

説　明　　ストリーム入出力用ファイルから文字列を入力します。

ヘッダ　　<stdio.h>

リターン値　　正常： ファイルの終了の時　　：NULL
　　ファイルの終了でない時　：s
異常： NULL

引　数

s	文字列を入力する記憶域へのポインタ
n	文字列を入力する記憶域のバイト数
fp	ファイルポインタ

例

```
#include <stdio.h>
char *s, *ret;
int n;
FILE *fp;
   ret=fgets(s,n,fp);
```

備　考　　fgets 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから、ポインタ s の指す記憶域に文字列を入力します。
fgets 関数は、n-1 文字まであるいは改行文字を入力するまで、またはファイルの終わりになるまで文字を入力し、入力文字列の最後にヌル文字を付け加えます。
fgets 関数は通常、文字列を入力する記憶域へのポインタ s を返しますが、ファイルが終了した時やエラー発生の時は NULL を返します。
ファイルが終了した時は、s が指す記憶域の内容は変化しませんが、エラー発生の時は、s が指す記憶域の内容は保証しません。

ファイルに 1 文字出力

*long fputc(long c, FILE *fp)*

説　明	ストリーム入出力用ファイルへ 1 文字出力します。
ヘッダ	<stdio.h>
リターン値	正常：出力した文字 異常：EOF
引　数	c　　出力する文字 fp　　ファイルポインタ

例	`#include <stdio.h>`
	`FILE *fp;`
	`int c, ret;`
	`ret=fputc(c,fp);`

エラー条件	書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
備　考	fputc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力ファイルへ出力します。 fputc 関数は、通常出力した文字 c を返しますが、エラー発生の時は、EOF を返します。

ファイルに文字列出力

*long fputs(const char *s, FILE *fp)*

説　明　　ストリーム入出力用ファイルへ文字列を出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：0
異常：0 以外

引　数

s	出力する文字列へのポインタ
fp	ファイルポインタ

例

```
#include <stdio.h>
const char *s;
int ret;
FILE *fp;
   ret=fputs(s,fp);
```

備　考　　fputs 関数は、s の指すヌル文字の直前までの文字列をファイルポインタ fp の示すストリーム入出力用ファイルへ出力します。この時、文字列の終了を示すヌル文字は出力されません。
fputs 関数は、通常 0 を返しますが、エラー発生の時は、0 以外の値を返します。

ファイルから 1 文字入力

*long getc(FILE *fp)*

説　明　　ストリーム入出力用ファイルから 1 文字入力します。

ヘッダ　　<stdio.h>

リターン値　　正常： ファイルの終了の時　　：EOF
　　ファイルの終了でない時　：入力した文字
異常： EOF

引　数　　fp　　ファイルポインタ

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=getc(fp);
```

エラー条件　　読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

備　考　　getc 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから 1 文字入力します。
getc 関数は、通常入力した 1 文字を返しますがファイルの終了やエラー発生の時は、EOF を返します。またファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

1 文字入力

long getchar(void)

説　明　　標準入力ファイル(stdin)から、1 文字入力します。

ヘッダ　　<stdio.h>

リターン値　　正常： ファイルの終了の時　　：EOF
　　ファイルの終了でない時　：入力した文字
異常： EOF

例

```
#include <stdio.h>
int ret;
   ret=getchar();
```

エラー条件　　読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

備　考　　getchar 関数は標準入力ファイル(stdin)から 1 文字入力します。
getchar 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の時は EOF を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

文字列入力

*char *gets(char *s)*

説　明　　標準入力ファイル(stdin)から文字列を入力します。

ヘッダ　　<stdio.h>

リターン値　　正常： ファイルの終了の時　　：NULL
　　ファイルの終了でない時　：s
異常： NULL

引　数　　s　　文字列を入力する記憶域へのポインタ

例
```
#include <stdio.h>
char *ret, *s;
   ret=gets(s);
```

備　考　　gets 関数は、標準入力ファイル(stdin)から、s で始まる記憶域へ文字列を入力します。
gets 関数は、ファイルの終了か、改行文字を入力するまで文字を入力し、改行文字の代わりにヌル文字を付け加えます。
gets 関数は、通常文字列を入力する記憶域へのポインタ s を返しますが、標準入力ファイルの終了やエラー発生の時は、NULL を返します。
標準入力ファイルが終了した時は、s が指す記憶域の内容は変化しませんが、エラー発生の時は s が指す記憶域の内容は保証しません。

ファイルに 1 文字出力

*long putc(long c, FILE *fp)*

説　明　　ストリーム入出力用ファイルへ 1 文字出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：出力した文字
異常：EOF

引　数　　c　　出力する文字
fp　　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
int c, ret;
   ret=putc(c,fp);
```

エラー条件　　書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

備　考　　putc 関数は､文字 c をファイルポインタ fp の示すストリーム入出力ファイルへ出力します。
putc 関数は、通常出力した文字 c を返しますが、エラー発生の時は EOF を返します。

1 文字出力

long putchar(long c)

説　明　　標準出力ファイル(stdout)へ 1 文字出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：出力した文字
異常：EOF

引　数　　c　　出力する文字

例

```
#include <stdio.h>
int c, ret;
   ret=putchar(c);
```

エラー条件　　書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

備　考　　putchar 関数は、文字 c を標準出力ファイル(stdout)へ出力します。putchar マクロは、通常出力した文字 c を返しますが、エラー発生の時は EOF を返します。

文字列出力

*long puts(const char *s)*

説　明　　標準出力ファイル(stdout)へ文字列を出力します。

ヘッダ　　<stdio.h>

リターン値　　正常：0
異常：0 以外

引　数　　s　　出力する文字列へのポインタ

例

```
#include <stdio.h>
const char *s;
int ret;
   ret=puts(s);
```

備　考　　puts 関数は、s の指す文字列を標準出力ファイル(stdout)へ出力します。この時、文字列の終了を示す文字は出力されず、代わりに改行文字を出力します。
puts 関数は、通常 0 を返しますが、エラー発生の時は 0 以外の値を返します。

ファイルに 1 文字返却

*long ungetc(long c, FILE *fp)*

説　明　　ストリーム入出力用ファイルへ 1 文字を戻します。

ヘッダ　　<stdio.h>

リターン値　　正常：戻した文字
異常：EOF

引　数

c	戻す文字
fp	ファイルポインタ

例

```
#include <stdio.h>
int c, ret;
FILE *fp;
   ret=ungetc(c,fp);
```

備　考　　ungetc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力用ファイルへ戻します。
また、ここで戻された文字は、fflush，fseek，rewind 関数を呼び出さなければ次の入力データとなります。
ungetc 関数は、通常戻した文字 c を返しますが、エラー発生の時は EOF を返します。
ungetc 関数が fflush，fseek，rewind 関数を実行することなく 2 回以上呼び出された時の動作は保証しません。また、ungetc 関数が実行されるとファイルに対する現在の位置指示子が一つ戻されますが、この位置指示子がすでにファイルの先頭に位置している時は、位置指示子は保証しません。

ファイル読み込み

*size_t fread(void *ptr, size_t size, size_t n, FILE *fp)*

説　明　　ストリーム入出力用ファイルから、指定した記憶域にデータを入力します。

ヘッダ　　<stdio.h>

リターン値　　size もしくは n が 0 の時　：0
size,n がともに 0 でない時 ：入力に成功したメンバ数

引　数

ptr	データを入力する記憶域へのポインタ
size	1 メンバのバイト数
n	入力するメンバの数
fp	ファイルポインタ

例

```
#include <stdio.h>
void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
   ret=fread(ptr,size,n,fp);
```

備　考　　fread 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから ptr が指す記憶域に size で指定したバイト数を 1 メンバとしたデータを n メンバ入力します。この時、ファイルに対する位置指示子は入力したバイト数分進められます。
fread 関数は、実際に入力に成功したメンバ数を返しますので、通常 n と同じ値になります。しかし、ファイルが終了した時やエラー発生の時は、それまで入力に成功したメンバ数を返しますので、n より小さな値となります。ファイルの終了かエラー発生かの区別は、ferror, feof 関数を用いて行ってください。
size もしくは n が 0 の時、リターン値として 0 を返し、ptr の指す記憶域の内容は変化しません。また、エラーが発生した時、または、メンバの途中までしか入力できなかった時は、そのファイルの位置指示子は保証しません。

ファイル書き込み

*size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp)*

説　明　メモリ領域からストリーム入出力用ファイルにデータを出力します。

ヘッダ　<stdio.h>

リターン値　出力に成功したメンバ数

引　数

ptr	出力するデータを格納している記憶域へのポインタ
size	1 メンバのバイト数
n	出力するメンバの数
fp	ファイルポインタ

例

```
#include <stdio.h>
const void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
   ret=fwrite(ptr,size,n,fp);
```

備　考　fwrite 関数は、ptr の指す記憶域から、ファイルポインタ fp の示すストリーム入出力用ファイルに、size で指定したバイト数を 1 メンバとしたデータを n メンバ出力します。
この時、ファイルに対する位置指示子は出力したバイト数進められます。
fwrite 関数は、実際に出力に成功したメンバ数を返しますので、通常 n と同じ値になります。しかし、エラー発生の時はそれまで出力に成功したメンバ数を返しますので、n より小さな値となります。
エラー発生の時、そのファイルの位置指示子は保証しません。

ファイル読み書き位置移動

*long fseek(FILE *fp, long offset, long type)*

説　明　ストリーム入出力用ファイルの現在の読み書き位置を移動します。

ヘッダ　<stdio.h>

リターン値　正常：0
異常：0 以外

引　数

fp	ファイルポインタ
offset	オフセットの種類で指定された位置からのオフセット
type	オフセットの種類

例

```
#include <stdio.h>
FILE *fp;
long offset;
int type, ret;
   ret=fseek(fp,offset,type);
```

備　考　fseek 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置をオフセットの種類 type で指定した場所から offset バイト先の位置に移動します。
オフセットの種類を**表** 9.37 に示します。
fseek 関数は、通常は 0 を返しますが、不適当な要求に対しては 0 以外の値を返します。

表 9.37　**オフセットの種類**

	オフセットの種類	意味
1	SEEK_SET	**ファイルの先頭から** offset **バイト先の位置に移動します。この時、**offset **で指定する値は** 0 **か正でなければなりません。**
2	SEEK_CUR	**ファイルの現在位置から** offset **バイト先の位置に移動します。この時、**offset **で指定する値が正ならばファイルの後方に、負ならばファイルの先頭に向かって移動します。**
3	SEEK_END	**ファイルの終わりから** offset **バイト先の位置に移動します。この時** offset **で指定する値は** 0 **か負でなければなりません。**

テキストファイルの時は、オフセットの種類は SEEK_SET で、かつ offset は 0 かそのファイルに対する ftell 関数によって返された値でなければなりません。また、fseek 関数を呼び出すことによって ungetc 関数の効果はなくなりますので注意が必要です。

ファイル読み書き位置取得

*long ftell(FILE *fp)*

説　明　ストリーム入出力用ファイルの現在の読み書き位置を求めます。

ヘッダ　<stdio.h>

リターン値　現在の位置指示子の位置(テキストファイル)
ファイルの先頭から現在位置までのバイト数(バイナリファイル)

引　数　fp　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
long ret;
   ret=ftell(fp);
```

備　考　ftell 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置を求めます。
ftell 関数は、バイナリファイルの時、ファイルの先頭から現在位置までのバイト数を返しますが、テキストファイルの時は、ここで返した値が fseek 関数で使用できるように処理系定義の値を位置指示子の位置として返します。
ftell 関数を 2 回テキストファイルに適用した時、そのリターン値の差が実際のファイル上の隔たりを表すことにはなりません。

ファイル先頭に移動

*void rewind(FILE *fp)*

説　明　ストリーム入出力用ファイルの現在の読み書き位置を、ファイルの先頭に移動します。

ヘッダ　<stdio.h>

引　数　fp　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
   rewind(fp);
```

備　考　rewind 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
また、rewind 関数は、そのファイルに対する終了指示子とエラー指示子をクリアします。
rewind 関数を呼び出すことによって、ungetc 関数の効果はなくなりますので、注意が必要です。

エラー状態クリア

*void clearerr(FILE *fp)*

説　明　　ストリーム入出力用ファイルのエラー状態をクリアします。

ヘッダ　　<stdio.h>

引　数　　fp　　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
   clearerr(fp);
```

備　考　　clearerr 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対するエラー指示子と終了指示子をクリアします。

ファイル終了判定

*long feof(FILE *fp)*

説　明　　ストリーム入出力用ファイルが終わりであるかどうかを判定します。

ヘッダ　　<stdio.h>

リターン値　　ファイルが終わりの時　　：0 以外
ファイルが終わりでない時 ：0

引　数　　fp　　ファイルポインタ

例
```
#include <stdio.h>
FILE *fp;
int ret;
   ret=feof(fp);
```

備　考　　feof 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルが終了したかどうかを判定します。
feof 関数は、指定したストリーム入出力用ファイルに対するファイル終了指示子を調べ、設定されていればファイルが終わりであるとして、0 以外の値を返します。設定されていなければ、ファイルはまだ終わりではないとして 0 を返します。

ファイルエラー状態判定

*long ferror(FILE *fp)*

説　明　　ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。

ヘッダ　　<stdio.h>

リターン値　　ファイルがエラー状態の時　　：0 以外
ファイルがエラー状態でない時：0

引　数　　fp　　ファイルポインタ

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=ferror(fp);
```

備　考　　ferror 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
ferror 関数は、指定したストリーム入出力用ファイルに対するエラー指示子を調べ、設定されていれば、エラー状態にあるとして 0 以外の値を返します。設定されていなければ、エラー状態ではないとして 0 を返します。

エラーメッセージ出力

*void perror(const char *s)*

説　明　　標準エラーファイル(stderr)に、エラー番号に対応したエラーメッセージを出力します。

ヘッダ　　<stdio.h>

引　数　　s　　エラーメッセージへのポインタ

例

```
#include <stdio.h>
const char *s;
   perror(s);
```

備　考　　perror 関数は標準エラーファイル(stderr)へ s で示されるエラーメッセージと errno とを対応させ出力します。
出力するメッセージは、もし、s が NULL でなく、s の指す文字列がヌル文字でなければ、s の指す文字列にコロンと空白とその後に処理系定義のエラーメッセージを続け、最後に改行文字を付けた形式で出力されます。

（13）<stdlib.h>

C プログラムでの標準的処理を行う関数を定義しています。

以下のマクロは、処理系定義です。

種別	定義名	説明
型	onexit_t	onexit 関数で登録する関数の返す型および onexit 関数のリターン値の型です。
(マクロ)	div_t	div 関数のリターン値の構造体の型です。
	ldiv_t	ldiv 関数のリターン値の構造体の型です。
	lldiv_t	lldiv 関数のリターン値の構造体の型です。
定数	RAND_MAX	rand 関数において生成する擬似乱数整数の最大値です。
(マクロ)	EXIT_SUCCES S	成功終了状態を表します。
関数	atof	数を表現する文字列を double 型の浮動小数点値に変換します。
	atoi	10 進数を表現する文字列を int 型の整数値に変換します。
	atol	10 進数を表現する文字列を long 型の整数値に変換します。
	atoll	10 進数を表現する文字列を long long 型の整数値に変換します。
	strtod	数を表現する文字列を double 型の浮動小数点値に変換します。
	strtof	数を表現する文字列を float 型の浮動小数点値に変換します。
	strtold	数を表現する文字列を long double 型の浮動小数点値に変換します。
	strtol	数を表現する文字列を long 型の整数値に変換します。
	strtoul	数を表現する文字列を unsigned long 型の整数値に変換します。
	strtoll	数を表現する文字列を long long 型の整数値に変換します。
	strtoull	数を表現する文字列を unsigned long long 型の整数値に変換します。
	rand	0 から RAND_MAX の間の擬似乱数整数を生成します。
	srand	rand 関数で生成する擬似乱数列の初期値を設定します。
	calloc	記憶域を割り当てて、すべての割り当てられた記憶域を 0 で初期化します。
	free	指定された記憶域を解放します。
	malloc	記憶域を割り当てます。
	realloc	記憶域の大きさを指定された大きさに変更します。
	bsearch	2 分割検索を行います。
	qsort	ソートを行います。
	abs	int 型整数の絶対値を計算します。
	div	int 型整数の除算の商と余りを計算します。
	labs	long 型整数の絶対値を計算します。
	ldiv	long 型整数の除算の商と余りを計算します。
	llabs	long long 型整数の絶対値を計算します。
	lldiv	long long 型整数の除算の商と余りを計算します。

処理系定義仕様

	項目	コンパイラの仕様
1	calloc,malloc,realloc 関数でサイズが 0 のときの動作	NULL を返します。

文字列を double 型に変換

*double atof(const char *nptr)*

説　明　　数を表現する文字列を、double 型の浮動小数点値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　変換された double 型の浮動小数点値

引　数　　nptr　　変換する数を表現する文字列のポインタ

例

```
#include <stdlib.h>
const char *nptr;
double ret;
   ret=atof(nptr);
```

エラー条件　　変換後の値がオーバフロー/アンダフローをおこした時は errno を設定します。

備　考　　変換は、浮動小数点型の形式に合わない最初の文字までに対して行います。
atof 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtod 関数を使用してください。

文字列を int 型に変換

*long atoi(const char *nptr)*

説　明　　10 進数を表現する文字列を、int 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　変換された int 型の整数値

引　数　　nptr　　変換する数を表現する文字列のポインタ

例

```
#include <stdlib.h>
const char *nptr;
int ret;
   ret=atoi(nptr);
```

エラー条件　　変換後の値がオーバフローをおこした時は errno を設定します。

備　考　　変換は、10 進数の形式に合わない最初の文字までに対して行います。
atoi 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtol 関数を使用してください。

文字列を long 型に変換

*long atol(const char *nptr)*

説　明　　10 進数を表現する文字列を、long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　変換された long 型の整数値

引　数　　nptr　　変換する数を表現する文字列のポインタ

例

```
#include <stdlib.h>
const char *nptr;
long ret;
   ret=atol(nptr);
```

エラー条件　　変換後の値がオーバフローをおこした時は errno を設定します。

備　考　　変換は、10 進数の形式に合わない最初の文字までに対して行います。
atol 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtol 関数を使用してください。

文字列を long long 型に変換

*long long atoll (const char *nptr)*

説　明　　10 進数を表現する文字列を、long long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　変換された long long 型の整数値

引　数　　nptr　　変換する数を表現する文字列のポインタ

例

```
#include <stdlib.h>
const char *nptr;
long long ret;
   ret=atoll(nptr);
```

エラー条件　　変換後の値がオーバフローをおこした時は errno を設定します。

備　考　　変換は、10 進数の形式に合わない最初の文字までに対して行います。
atoll 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtoll 関数を使用してください。

文字列を double 型に変換

*double strtod(const char *nptr, char **endptr)*

説　明　　数を表現する文字列を double 型の浮動小数点値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
nptr が指している文字列が浮動小数点型を構成する文字で始まっている時
：変換された double 型の浮動小数点値
異常： 変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VAL
変換後の値がアンダフローの時：0

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

例

```
#include <stdlib.h>
const char *nptr;
char **endptr;
double ret;
   ret=strtod(nptr,endptr);
```

エラー条件　　変換後の値がオーバフロー/アンダフローをおこした時は errno を設定します。

備　考　　strtod 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを「9.1.3(4) **浮動小数点演算の仕様**」の規則に従って double 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

文字列を float 型に変換

*float strtof(const char *nptr, char **endptr)*

説　明　　数を表現する文字列を float 型の浮動小数点値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
nptr が指している文字列が浮動小数点型を構成する文字で始まっている時
：変換された float 型の浮動小数点値
異常： 変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VALF
変換後の値がアンダフローの時：0

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

例

```
#include <stdlib.h>
const char *nptr;
char **endptr;
float ret;
   ret=strtof(nptr,endptr);
```

エラー条件　　変換後の値がオーバフロー/アンダフローをおこした時は errno を設定します。

備　考　　strtof 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを「9.1.3(4) **浮動小数点演算の仕様**」の規則に従って float 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

*long double strtold(const char *nptr, char **endptr)*

説　明　　数を表現する文字列を long double 型の浮動小数点値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
nptr が指している文字列が浮動小数点型を構成する文字で始まっている時
：変換された long double 型の浮動小数点値
異常： 変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VALL
変換後の値がアンダフローの時：0

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

例

```
#include <stdlib.h>
const char *nptr;
char **endptr;
long double ret;
   ret=strtold(nptr,endptr);
```

エラー条件　　変換後の値がオーバフロー/アンダフローをおこした時は errno を設定します。

備　考　　strtold 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを「9.1.3(4) **浮動小数点演算の仕様**」の規則に従って long double 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

文字列を long 型に変換

*long strtol(const char *nptr, char **endptr, long base)*

説　明　　数を表現する文字列を long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された long 型の整数値
異常： 変換後の値がオーバフローの時：変換する文字列の符号に従って LONG_MAX
あるいは LONG_MIN

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <stdlib.h>
long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtol(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno を設定します。

備　考　　strtol 関数は、最初の数字から整数を構成しない最初の文字の前までを long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、**「**9.1.1(4) **整数」**の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A)から z(もしくは Z)までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X)も無視されます。

文字列を unsigned long 型に変換

*unsigned long strtoul (const char *nptr, char **endptr, long base)*

説　明　　数を表現する文字列を unsigned long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された unsigned long 型の整数値
異常： 変換後の値がオーバフローの時：ULONG_MAX

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <stdlib.h>
unsigned long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoul(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno を設定します。

備　考　　strtoul 関数は、最初の数字から整数を構成しない最初の文字の前までを unsigned long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「9.1.1(4) **整数**」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A)から z(もしくは Z)までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X)も無視されます。

文字列を long long 型に変換

*long long strtoll (const char *nptr, char **endptr, long base)*

説　明　　数を表現する文字列を long long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された long long 型の整数値
異常： 変換後の値がオーバフローの時：変換する文字列の符号に従って LLONG_MAX
あるいは LLONG_MIN

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <stdlib.h>
long long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoll(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno を設定します。

備　考　　strtoll 関数は、最初の数字から整数を構成しない最初の文字の前までを long long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、**「**9.1.1(4) **整数」**の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A)から z(もしくは Z)までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X)も無視されます。

文字列を unsigned long long 型に変換

*unsigned long long strtoull (const char *nptr, char **endptr, long base)*

説　明　　数を表現する文字列を unsigned long long 型の整数値に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された unsigned long long 型の整数値
異常： 変換後の値がオーバフローの時：ULLONG_MAX

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <stdlib.h>
unsigned long long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoull(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno を設定します。

備　考　　strtoull 関数は、最初の数字から整数を構成しない最初の文字の前までを unsigned long long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「9.1.1(4) **整数**」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A)から z(もしくは Z)までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X)も無視されます。

擬似乱数生成

long rand(void)

説　明　　0 から RAND_MAX の間の擬似乱数整数を生成します。

ヘッダ　　<stdlib.h>

リターン値　　擬似乱数整数値

例

```
#include <stdlib.h>
int ret;
    ret=rand();
```

擬似乱数列初期設定

void srand(unsigned long seed)

説　明　　rand 関数で生成する擬似乱数列の初期値を設定します。

ヘッダ　　<stdlib.h>

引　数　　seed　　擬似乱数列生成の初期値

例

```
#include <stdlib.h>
unsigned int seed;
    srand(seed);
```

備　考　　srand 関数は、rand 関数が擬似乱数列を生成するための初期値を設定します。したがって、rand 関数で擬似乱数値を生成している時に、再度 srand 関数で、同じ値の初期値を設定すると、擬似乱数列はくり返し生成されることになります。
rand 関数が srand 関数より先に呼ばれた時は、擬似乱数列の生成の初期値として 1 が設定されます。

初期化付き記憶域割り当て

*void *calloc(size_t nelem, size_t elsize)*

説　明　　記憶域を割り当てて、すべての割り当てられた記憶域を 0 で初期化します。

ヘッダ　　<stdlib.h>

リターン値　　正常：割り当てられた記憶域の先頭のアドレス
異常：記憶域の割り当てができなかった時、または引数のいずれかが 0 の時：NULL

引　数

nelem	要素の数
elsize	一つの要素の占めるバイト数

例

```
#include <stdlib.h>
size_t nelem, elsize;
void *ret;
   ret=calloc(nelem,elsize);
```

備　考　　elsize バイト単位の記憶域を nelem 個記憶域に割り当てます。また、その割り当てられた記憶域のすべてのビットは 0 で初期化されます。

記憶域解放

*void free(void *ptr)*

説　明　　指定された記憶域を解放します。

ヘッダ　　<stdlib.h>

引　数

ptr	解放する記憶域のアドレス

例

```
#include <stdlib.h>
void *ptr;
   free(ptr);
```

備　考　　ptr が指す記憶域を解放し、再度割り当てて使用することを可能とします。ptr が NULL であれば何もしません。
解放しようとした記憶域が、calloc、malloc、realloc 関数で割り当てられた記憶域でない時、または、すでに free、realloc 関数によって解放されていた時の動作は保証しません。また、解放された後の記憶域を参照した時の動作も保証しません。

記憶域割り当て

*void *malloc(size_t size)*

説　明　　記憶域を割り当てます。

ヘッダ　　<stdlib.h>

リターン値　　正常：割り当てられた記憶域の先頭アドレス
異常：記憶域の割り当てができなかった時、または size が 0 の時：NULL

引　数　　size　　割り当てる記憶域のバイト数

例

```
#include <stdlib.h>
size_t size;
void *ret;
   ret=malloc(size);
```

備　考　　size で示されるバイトの分だけ記憶域を割り当てます。

記憶域割り当てサイズ変更

*void *realloc(void *ptr, size_t size)*

説　明　　記憶域の大きさを指定された大きさに変更します。

ヘッダ　　<stdlib.h>

リターン値　　正常：変更した記憶域の先頭アドレス
異常：記憶域の割り当てができなかった時、または size が 0 の時：NULL

引　数　　ptr　　変更する記憶域の先頭アドレス
size　　変更後の記憶域のバイト数

例

```
#include <stdlib.h>
size_t size;
void *ptr, *ret;
   ret=realloc(ptr,size);
```

備　考　　ptr の指す記憶域の大きさを size で示されるバイト分の大きさの記憶域に変更します。もし、新しく割り当てられた記憶域の大きさが、変更前の記憶域の大きさより小さい時は、新しく割り当てられた記憶域の大きさまでの内容は変化しません。
ptr が calloc、malloc、realloc 関数で割り当てられた記憶域へのポインタでない時、またはすでに free、realloc 関数によって解放されている記憶域へのポインタの時、動作はされません。

二分割検索

*void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *))*

説　明　　二分割検索を行います。

ヘッダ　　<stdlib.h>

リターン値　　一致するメンバが検索できた時　　：一致したメンバへのポインタ
一致するメンバが検索できなかった時　：NULL

引　数

key	検索するデータへのポインタ
base	検索対象となるテーブルへのポインタ
nmemb	検索対象のメンバの数
size	検索対象のメンバのバイト数
compar	比較を行う関数へのポインタ

例

```
#include <stdlib.h>
const void *key, *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
void *ret;
    ret=bsearch(key,base,nmemb,size,compar);
```

備　考　　key の指すデータと一致するメンバを、base の指すテーブルの中で二分割検索法によって検索します。比較を行う関数は、比較する 2 つのデータへのポインタ p1(第 1 引数)、p2(第 2 引数)を受け取り、以下の仕様に従って結果を返してください。

- *p1<*p2 の時、負の値を返します。
- *p1==*p2 の時、0 を返します。
- *p1>*p2 の時、正の値を返します。

検索対象となる各メンバは、昇順に並んでいる必要があります。

ソート

*void qsort(const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *))*

説　明　　ソートを行います。

ヘッダ　　<stdlib.h>

引　数

base	ソート対象となるテーブルへのポインタ
nmemb	ソート対象のメンバの数
size	ソート対象のメンバのバイト数
compar	比較を行う関数へのポインタ

例

```
#include <stdlib.h>
const void *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
   qsort(base, nmemb, size, compar);
```

備　考　　base の指すテーブルのデータをソートします。データの並べる順序は、比較を行う関数へのポインタによって指定します。この関数は、比較する 2 つのデータへのポインタ p1(第 1 引数)、p2(第 2 引数)を受け取り、以下の仕様に従って結果を返してください。

- *p1<*p2 の時、負の値を返します。
- *p1==*p2 の時、0 を返します。
- *p1>*p2 の時、正の値を返します。

絶対値

long abs(long i)

説　明　　int 型整数の絶対値を求めます。

ヘッダ　　<stdlib.h>

リターン値　　i の絶対値

引　数

i	絶対値を求める整数

例

```
#include <stdlib.h>
int i, ret;
   ret=abs(i);
```

備　考　　i の絶対値を求めた結果、int 型整数値として表現できない時の動作は保証しません。

商と余り

div_t div(long numer, long denom)

説　明　　int 型整数の除算の商と余りを計算します。

ヘッダ　　<stdlib.h>

リターン値　　numer を denom で除算した結果の商と余り

引　数

numer	被除数
denom	除数

例

```
#include <stdlib.h>
int numer, denom;
div_t ret;
   ret=div(numer,denom);
```

絶対値

long labs(long j)

説　明　　long 型整数の絶対値を計算します。

ヘッダ　　<stdlib.h>

リターン値　　j の絶対値

引　数

j	絶対値を求める整数

例

```
#include <stdlib.h>
long j;
long ret;
   ret=labs(j);
```

備　考　　j の絶対値を求めた結果、long 型の整数値として表現できない時の動作は保証しません。

商と余り

ldiv_t ldiv(long numer, long denom)

説　明　　long 型整数の除算の商と余りを計算します。

ヘッダ　　<stdlib.h>

リターン値　　numer を denom で除算した結果の商と余り

引　数

numer	被除数
denom	除数

例

```
#include <stdlib.h>
long numer, denom;
ldiv_t ret;
   ret=ldiv(numer,denom);
```

絶対値

long long llabs(long long j)

説　明　　long long 型整数の絶対値を計算します。

ヘッダ　　<stdlib.h>

リターン値　　j の絶対値

引　数

j	絶対値を求める整数

例

```
#include <stdlib.h>
long long j;
long long ret;
   ret=llabs(j);
```

備　考　　j の絶対値を求めた結果、long long 型の整数値として表現できない時の動作は保証しません。

商と余り

lldiv_t lldiv(long long numer,long long denom)

説　明　　long long 型整数の除算の商と余りを計算します。

ヘッダ　　<stdlib.h>

リターン値　　numer を denom で除算した結果の商と余り

引　数

numer	被除数
denom	除数

例

```
#include <stdlib.h>
long long numer, denom;
lldiv_t ret;
   ret=lldiv(numer,denom);
```

（14）<string.h>

文字配列の操作に必要な種々の関数を定義します。

種別	定義名	説明
関数	memcpy	複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
	strcpy	複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。
	strncpy	複写元の文字列を指定された文字数分、複写先の記憶域に複写します。
	strcat	文字列の後に、文字列を連結します。
	strncat	文字列に文字列を指定した文字数分、連結します。
	memcmp	指定された 2 つの記憶域の比較を行います。
	strcmp	指定された 2 つの文字列を比較します。
	strncmp	指定された 2 つの文字列を指定された文字数分まで比較します。
	memchr	指定された記憶域において、指定された文字が最初に現われる位置を検索します。
	strchr	指定された文字列において、指定された文字が最初に現われる位置を検索します。
	strcspn	指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。
	strpbrk	指定された文字列において、別に指定された文字列中の文字が最初に現われる位置を検索します。
	strrchr	指定された文字列において指定された文字が最後に現われる位置を検索します。
	strspn	指定された文字列を先頭から調べ別に指定した文字列中の文字が先頭から何文字続くかを求めます。
	strstr	指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。
	strtok	指定した文字列をいくつかの字句に切り分けます。
	memset	指定された記憶域の先頭から指定された文字を指定された文字数分設定します。
	strerror	エラーメッセージを設定します。
	strlen	文字列の文字数を計算します。
	memmove	複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。複写元と複写先の記憶域が重なっていても、正しく複写されます。

処理系定義仕様

	項目	コンパイラの仕様
1	strerror 関数が返すエラーメッセージの内容	「11.3 C 標準ライブラリ関数のエラーメッセージ」を参照してください。

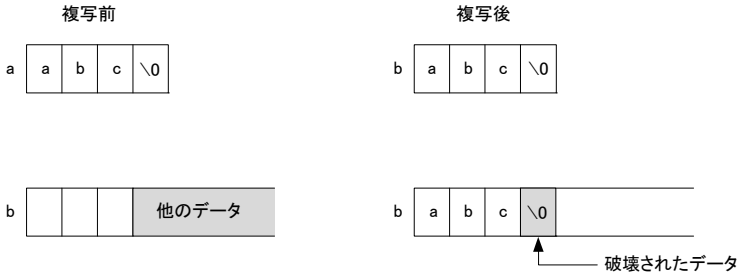
本標準インクルードファイル内で定義されている関数を使用する時は、以下の 2 つの事項に注意する必要があります。

(1) 文字列の複写を行う時、複写先の領域が複写元の領域よりも小さい場合、動作は保証しませんので注意が必要です。

例：

```
char a[]="abc";
char b[3];
            :
            :
strcpy(b,a);
```

この場合、配列aのサイズは(ヌル文字を含めて)4バイトです。したがって、strcpy関数によって複写を行うと、配列bの領域以外のデータを書き換えることになります。

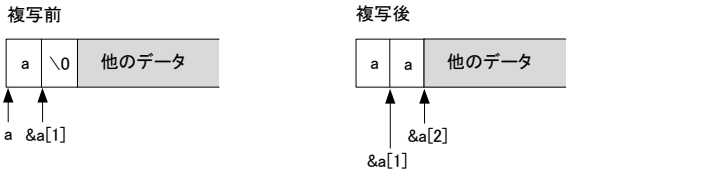


(2) 文字列の複写を行う時、複写元の領域と複写先の領域が重なっていると正しい動作は保証しませんので注意が必要です。

例：

```
int a[]="a";
            :
            :
strcpy(&a[1], a);
            :
            :
```

この場合、複写元の文字列がヌル文字に達する以前に、ヌル文字の上に文字'a'を書き込むことになります。したがって、複写元の文字列のデータに続くデータを書き換えることになります。



以下次々に文字を複写し続けます。

記憶域複写

*void *memcpy(void *s1, const void *s2, size_t n)*

説　明　　複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。

ヘッダ　　<string.h>

リターン値　　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の記憶域へのポインタ
n	複写する文字数

例

```
#include <string.h>
void *ret, *s1;
const void *s2;
size_t n;
   ret=memcpy(s1,s2,n);
```

文字列複写

*char *strcpy(char *s1, const char *s2)*

説　明　　複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。

ヘッダ　　<string.h>

リターン値　　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の文字列へのポインタ

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
   ret=strcpy(s1,s2);
```

文字列複写

*char *strncpy(char *s1, const char *s2, size_t n)*

説　明　複写元の文字列を指定された文字数分、複写先の記憶域に複写します。

ヘッダ　<string.h>

リターン値　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の文字列へのポインタ
n	複写する文字数

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
   ret=strncpy(s1,s2,n);
```

備　考　s2 で指された文字列の先頭から最高 n 文字を s1 で指される記憶域に複写します。s2 で指定された文字列の文字数が n 文字より短い時は、n 文字になるまでヌル文字が付加されます。s2 で指された文字列の文字数が n 文字より長い時は、s1 に複写された文字列はヌル文字で終了しないことになります。

文字列連結

*char *strcat(char *s1, const char *s2)*

説　明　文字列の後に、文字列を連結します。

ヘッダ　<string.h>

リターン値　s1 の値

引　数

s1	連結される文字列へのポインタ
s2	連結する文字列へのポインタ

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
   ret=strcat(s1,s2);
```

備　考　s1 で指された文字列の最後に、s2 で指された文字列を連結します。この時、s2 の指す文字列の最後を示すヌル文字も複写します。また、s1 で指された文字列の最後のヌル文字は削除されます。

文字列連結

*char *strncat(char *s1, const char *s2, size_t n)*

説　明　　文字列に文字列を指定した文字数分連結します。

ヘッダ　　<string.h>

リターン値　　s1 の値

引　数

s1	連結される文字列へのポインタ
s2	連結する文字列へのポインタ
n	連結する文字数

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
   ret=strncat(s1,s2,n);
```

備　考　　s2 で指された文字列の先頭から最高 n 文字を s1 で指された文字列の最後に付加します。s1 で指された文字列の最後のヌル文字は s2 の先頭文字で置き換えられます。
また、連結された後の文字列の最後には、必ずヌル文字が付加されます。

記憶域比較

*long memcmp(const void *s1, const void *s2, size_t n)*

説　明　　指定された 2 つの記憶域の内容を比較します。

ヘッダ　　<string.h>

リターン値

s1 で指された記憶域	>	s2 で指された記憶域の時：正の値
s1 で指された記憶域	==	s2 で指された記憶域の時：0
s1 で指された記憶域	<	s2 で指された記憶域の時：負の値

引　数

s1	比較される記憶域へのポインタ
s2	比較する記憶域へのポインタ
n	比較する記憶域の文字数

例

```
#include <string.h>
const void *s1, *s2;
size_t n;
int ret;
   ret=memcmp(s1,s2,n);
```

備　考　　s1 で指された記憶域と s2 で指された記憶域の、最初の n 文字分の内容を比較します。
この比較は処理系定義です。

文字列比較

*long strcmp(const char *s1, const char *s2)*

説　明　　指定された 2 つの文字列の内容を比較します。

ヘッダ　　<string.h>

リターン値　　s1 で指された文字列 > s2 で指された文字列の時：正の値
s1 で指された文字列 == s2 で指された文字列の時：0
s1 で指された文字列 < s2 で指された文字列の時：負の値

引　数　　s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ

例

```
#include <string.h>
const char *s1, *s2;
int ret;
   ret=strcmp(s1,s2);
```

備　考　　s1 で指された文字列と、s2 で指された文字列の内容を比較し、その結果をリターン値として設定します。
この比較は処理系定義です。

文字列比較

*long strncmp(const char *s1, const char *s2, size_t n)*

説　明　　指定された 2 つの文字列を指定された文字分まで比較します。

ヘッダ　　<string.h>

リターン値　　s1 で指された文字列 > s2 で指された文字列の時：正の値
s1 で指された文字列 == s2 で指された文字列の時：0
s1 で指された文字列 < s2 で指された文字列の時：負の値

引　数　　s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ
n　比較する文字数の最大値

例

```
#include <string.h>
const char *s1, *s2;
size_t n;
int ret;
   ret=strncmp(s1,s2,n);
```

備　考　　s1 で指された文字列と、s2 で指された文字列を最初の n 文字以内の範囲で、その内容を比較します。
この比較は処理系定義です。

記憶域内文字検索

*void *memchr(const void *s, long c, size_t n)*

説　明　　指定された記憶域において、指定された文字が最初に現われる位置を検索します。

ヘッダ　　<string.h>

リターン値　　検索の結果見つかった時　　：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s	検索を行う記憶域へのポインタ
c	検索する文字
n	検索を行う文字数

例

```
#include <string.h>
const void *s;
int c;
size_t n;
void *ret;
   ret=memchr(s,c,n);
```

備　考　　s で指定された記憶域の先頭から n 文字の中で最初に現われた c の文字と同一文字の位置へのポインタをリターン値として返します。

最初の文字位置

*char *strchr(const char *s, long c)*

説　明　　指定された文字列において、指定された文字が最初に現われる位置を検索します。

ヘッダ　　<string.h>

リターン値　　検索の結果見つかった時　　：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s	検索を行う文字列へのポインタ
c	検索する文字

例

```
#include <string.h>
const char *s;
int c;
char *ret;
   ret=strchr(s,c);
```

備　考　　s で指定された文字列中で最初に現われた c の文字と同一文字へのポインタをリターン値として返します。
s によって指される文字列の終了を現わすヌル文字も検索の対象として含まれます。

指定文字群が最初に現れるまでの文字数

*size_t strcspn(const char *s1, const char *s2)*

説　明　　指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

ヘッダ　　<string.h>

リターン値　　s2 が指す文字列を構成する文字以外の文字が構成される文字列 s1 の先頭からの長さ

引　数　　s1　　調べられる文字列へのポインタ
s2　　s1 を調べるための文字列へのポインタ

例

```
#include <string.h>
const char *s1, *s2;
size_t ret;
   ret=strcspn(s1,s2);
```

備　考　　s2 が指す文字列を構成する文字以外の文字が、文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の文字数をリターン値として返します。
s2 によって指される文字列の終了を表すヌル文字は、s2 で指された文字列の一部とはみなされません。

指定文字群が最初に現れる位置

*char *strpbrk(const char *s1, const char *s2)*

説　明　　指定された文字列内において、別に指定された文字列中の文字が最初に現われる位置を検索します。

ヘッダ　　<string.h>

リターン値　　検索の結果見つかった時　　：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

引　数　　s1　　検索を行う文字列へのポインタ
s2　　s1 内で検索する文字を示す文字列へのポインタ

例

```
#include <string.h>
const char *s1, *s2;
char *ret;
   ret=strpbrk(s1,s2);
```

備　考　　s1 で指された文字列において、s2 で指された文字列中の文字の一つが最初に現われる所を検索し、そのポインタをリターン値として返します。

最後の文字位置

*char *strrchr(const char *s, long c)*

説　明　　指定された文字列において、指定された文字が最後に現われる位置を検索します。

ヘッダ　　<string.h>

リターン値　　検索の結果見つかった時　　：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s	検索を行う文字列へのポインタ
c	検索する文字

例

```
#include <string.h>
const char *s;
int c;
char *ret;
   ret=strrchr(s,c);
```

備　考　　s で指された文字列の中で c で指定する文字と同一の文字が最後に現われた位置へのポインタをリターン値として返します。
s によって指される文字列の終了を表すヌル文字も検索の対象として含まれます。

指定文字群が連続する部分の長さ

*size_t strspn(const char *s1, const char *s2)*

説　明　　指定された文字列を先頭から調べ、別に指定した文字列中の文字が先頭から何文字続くかを求めます。

ヘッダ　　<string.h>

リターン値　　s1 の先頭から、s2 で指定した文字が続いている文字数

引　数

s1	調べられる文字列へのポインタ
s2	s1 を調べるための文字列へのポインタ

例

```
#include <string.h>
const char *s1, *s2;
size_t ret;
   ret=strspn(s1,s2);
```

備　考　　s2 が指す文字列を構成する文字が文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の文字数をリターン値として返します。

最初の文字列位置

*char *strstr(const char *s1, const char *s2)*

説　明　　指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。

ヘッダ　　<string.h>

リターン値　　検索の結果見つかったとき　　：見つけられた文字へのポインタ
検索の結果見つからなかったとき：NULL

引　数　　s1　　検索を行う文字列へのポインタ
s2　　検索する文字列へのポインタ

例

```
#include <string.h>
const char *s1, *s2;
char *ret;
   ret=strstr(s1,s2);
```

備　考　　s1 で指された文字列において、s2 で指された文字列が最初に現われる所を検索し、そのポインタをリターン値として返します。

char *strtok(char *s1, const char *s2)

説　明　　指定した文字列をいくつかの字句に切り分けます。

ヘッダ　　<string.h>

リターン値　　字句に切り分けられた時　　：切り分けた字句の先頭へのポインタ
字句に切り分けられなかった時：NULL

引　数

s1	いくつかの字句に切り分ける文字列へのポインタ
s2	文字列を切り分けるための文字からなる文字列へのポインタ

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
   ret=strtok(s1,s2);
```

備　考　　strtok 関数は文字列を切り分けるために連続的に呼び出されます。
(a) 最初の呼び出し時
s1 で指された文字列を先頭から s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。
(b) 2 回目以降の呼び出し時
以前に切り分けられた字句の次の文字から、s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。
2 回目以降の呼び出しの時は、第 1 引数には NULL を指定します。また、s2 で指された文字列は呼び出しのたびに異なっていてもかまいません。切り出された字句の最後にはヌル文字が付きます。
strtok 関数の使用例を以下に示します。
例：

```
1    #include <string.h>
2    static char s1[]="a@b,@c/@d";
3    char *ret;
4
5    ret=strtok(s1,"@");
6    ret=strtok(NULL,",@");
7    ret=strtok(NULL,"/@");
8    ret=strtok(NULL,"@");
```

【説明】
この例は、文字列「"a@b,@c/@d"」を strtok 関数を用いて a,b,c,d という字句に切り分けるプログラムを示しています。
2 行目で文字列 s1 に初期値として、文字列"a@b,@c/@d"を設定しています。
5 行目では、「@」を区切り文字として字句を切り分けるため、strtok 関数を呼び出します。この結果、文字'a'へのポインタがリターン値として得られ、文字'a'の次の最初の区切り文字である「@」にヌル文字を埋め込みます。この結果、文字列"a"が切り出されます。
以下、同一の文字列から次々に字句を切り出すために第 1 引数に NULL を指定して strtok 関数を呼び出します。
この結果、文字列"b"、"c"、"d"が次々に切り出されます。

文字繰り返し

*void *memset(void *s, long c, size_t n)*

説　明　　指定された記憶域の先頭から、指定された文字を指定された文字数分設定します。

ヘッダ　　<string.h>

リターン値　　s の値

引　数

s	文字が設定される記憶域へのポインタ
c	設定する文字
n	設定する文字数

例

```
#include <string.h>
void *s, *ret;
int c;
size_t n;
    ret=memset(s,c,n);
```

備　考　　s で指された記憶域に n 文字分、文字 c を設定します。

エラーメッセージ文字列

*char *strerror(long s)*

説　明　　エラー番号を指定して、それに対するエラーメッセージを返します。

ヘッダ　　<string.h>

リターン値　　エラー番号に対応するエラーメッセージ(文字列)へのポインタ

引　数

s	エラー番号

例

```
#include <string.h>
char *ret;
int s;
    ret=strerror(s);
```

備　考　　エラー番号 s に対応するエラーメッセージへのポインタをリターン値として返します。
エラーメッセージの内容に関しては処理系定義です。
リターン値として返されたエラーメッセージを修正した時、動作は保証しません。

文字列の文字数

*size_t strlen(const char *s)*

説　明　　文字列の文字数を計算します。

ヘッダ　　<string.h>

リターン値　　文字列の文字数

引　数　　s　　長さを求める文字列へのポインタ

例

```
#include <string.h>
const char *s;
size_t ret;
   ret=strlen(s);
```

備　考　　s が指す文字列の終了を表すヌル文字は、文字列の文字数としては計算に入れません。

記憶域移動

*void *memmove(void *s1, const void *s2, size_t n)*

説　明　　複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
また、複写元と複写先の記憶域が、重なっている部分があっても、複写元の重なっている部分を上書きする前に複写するので正しく複写されます。

ヘッダ　　<string.h>

リターン値　　s1 の値

引　数　　s1　　複写先の記憶域へのポインタ
s2　　複写元の記憶域へのポインタ
n　　複写する文字数

例

```
#include <string.h>
void *ret, *s1;
const void *s2;
size_t n;
   ret=memmove(s1,s2,n);
```

（15）<complex.h>

各種の複素数計算を行います。float 型の複素数の場合は、定義名の最後に’f’、long double 型の複素数の場合は、定義名の最後に’l’、double 型の複素数の場合は、定義名が関数名になります。

種別	定義名	説明
関数	cacos	複素数逆余弦を計算します。
	casin	複素数逆正弦を計算します。
	catan	複素数逆正接を計算します。
	ccos	複素数余弦を計算します。
	csin	複素数正弦を計算します。
	ctan	複素数正接を計算します。
	cacosh	複素数逆双曲線余弦を計算します。
	casinh	複素数逆双曲線正弦を計算します。
	catanh	複素数逆双曲線正接を計算します。
	ccosh	複素数双曲線余弦を計算します。
	csinh	複素数双曲線正弦を計算します。
	ctanh	複素数双曲線正接を計算します。
	cexp	複素数自然対数の底 e の z 乗を計算します。
	clog	複素数自然対数を計算します。
	cabs	複素数絶対値を計算します。
	cpow	複素数べき乗を計算します。
	csqrt	複素数平方根を計算します。
	carg	偏角を計算します。
	cimag	虚部を計算します。
	conj	虚部の符号を反転させて複素共役を計算します。
	cproj	リーマン球面上への射影を計算します。
	creal	実部を計算します。

複素数逆余弦

float complex cacosf(float complex z)
double complex cacos(double complex z)
long double complex cacosl(long double complex z)

説　明　　複素数逆余弦を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：z の逆余弦値
異常：定義域エラーの時は、非数を返します

引　数　　z　　複素数逆余弦を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=cacos(z);
```

エラー条件　　z の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　　cacos 関数のリターン値の実軸方向の範囲は[0,π]、虚軸方向の範囲は無限の区間です。

複素数逆正弦

float complex casinf(float complex z)
double complex casin(double complex z)
long double complex casinl(long double complex z)

説　明　　複素数逆正弦を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：z の複素数逆正弦値
異常：定義域エラーの時は、非数を返します

引　数　　z　　複素数逆正弦を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=casin(z);
```

エラー条件　　z の値が[-1.0,1.0]の範囲を超えている時、定義域エラーになります。

備　考　　casin 関数のリターン値の実軸方向の範囲は[-π/2,π/2]、虚軸方向の範囲は無限の空間です。

複素数逆正接

float complex catanf(float complex z)
double complex catan(double complex z)
long double complex catanl(long double complex z)

説　明　　複素数逆正接を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：z の複素数逆正接値

引　数　　z　　複素数逆正接を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=catan(z);
```

備　考　　catan 関数のリターン値の実軸方向の範囲は[-π/2,π/2]、虚軸方向の範囲は無限の空間です。

複素数余弦

float complex ccosf(float complex z)
double complex ccos(double complex z)
long double complex ccosl(long double complex z)

説　明　　複素数余弦を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数余弦値

引　数　　z　　複素数余弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=ccos(z);
```

複素数正弦

float complex csinf(float complex z)
double complex csin(double complex z)
long double complex csinl(long double complex z)

説　明　　複素数正弦を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数正弦値

引　数　　z　　複素数正弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=csin(z);
```

複素数正接

float complex ctanf(float complex z)
double complex ctan(double complex z)
long double complex ctanl(long double complex z)

説　明　　複素数正接を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数正接値

引　数　　z　　複素数正接を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=ctan(z);
```

複素数逆双曲線余弦

float complex cacoshf(float complex z)
double complex cacosh(double complex z)
long double complex cacoshl(long double complex z)

説　明　　複素数逆双曲線余弦を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：　z の複素数逆双曲線余弦値
異常：　定義域エラーの時は、非数を返します。

引　数　　z　　複素数逆双曲線余弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=cacosh(z);
```

エラー条件　　z の値が[-1.0, 1.0]の範囲を超えている時、定義域エラーになります。

備　考　　cacoshf 関数群のリターン値の範囲は[0, π]です。

複素数逆双曲線正弦

float complex casinhf(float complex z)
double complex casinh(double complex z)
long double complex casinhl(long double complex z)

説　明　　複素数逆双曲線正弦を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数逆双曲線正弦値

引　数　　z　　複素数逆双曲線正弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=casinh(z);
```

複素数逆双曲線正接

float complex catanhf(float complex z)
double complex catanh(double complex z)
long double complex catanhl(long double complex z)

説　明　　複素数逆双曲線正接を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数逆双曲線正接値

引　数　　z　　複素数逆双曲線正接を求める複素数

例

```
#include <complex.h>
double complex z, ret;
    ret=catanh(z);
```

複素数双曲線余弦

float complex ccoshf(float complex z)
double complex ccosh(double complex z)
long double complex ccoshl(long double complex z)

説　明　　複素数双曲線余弦を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数双曲線余弦値

引　数　　z　　双曲線余弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
    ret=ccosh(z);
```

複素数双曲線正弦

float complex csinhf(float complex z)
double complex csinh(double complex z)
long double complex csinhl(long double complex z)

説　明　　複素数双曲線正弦を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数双曲線正弦値

引　数　　z　　双曲線正弦を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=csinh(z);
```

複素数双曲線正接

float complex ctanhf(float complex z)
double complex ctanh(double complex z)
long double complex ctanhl(long double complex z)

説　明　　複素数双曲線正接を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数双曲線正接値

引　数　　z　　双曲線正接を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=ctanh(z);
```

複素数指数関数

float complex cexpf(float complex z)
double complex cexp(double complex z)
long double complex cexpl(long double complex z)

説　明　　複素数の指数関数を計算します。

ヘッダ　　<complex.h>

リターン値　　z の指数関数値

引　数　　z　　指数関数を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=cexp(z);
```

複素数自然対数

float complex clogf(float complex z)
double complex clog(double complex z)
long double complex clogl(long double complex z)

説　明　　複素数の自然対数を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：　z の複素数自然対数値
異常：　定義域エラーの時は、非数を返します。

引　数　　z　　複素数自然対数を求める複素数

例

```
#include <complex.h>
double complex z, ret;
   ret=clog(z);
```

エラー条件　　z の値が負の時、定義域エラーになります。
z の値が 0.0 の時、範囲エラーになります。

備　考　　clog 関数群のリターン値の実軸方向の範囲は無限の区間、虚軸方向の範囲は[-iπ,+iπ]です。

複素数絶対値

float cabsf(float complex z)
double cabs(double complex z)
long double cabsl(long double complex z)

説　明　　複素数絶対値を計算します。

ヘッダ　　<complex.h>

リターン値　　z の複素数絶対値

引　数　　z　　複素数絶対値を求める複素数

例

```
#include <complex.h>
double complex z, ret;
    ret=cabs(z);
```

複素数べき乗

float complex cpowf(float complex x, float complex y)
double complex cpow(double complex x, double complex y)
long double complex cpowl(long double complex x, long double complex y)

説　明　　複素数べき乗を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：　x の y 乗の値
異常：　定義域エラーの時は、非数を返します。

引　数　　x　　べき乗される値
y　　べき乗する値

例

```
#include <complex.h>
double complex x, y;
    ret=cpow(x, y);
```

エラー条件　　x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。

備　考　　cpow 関数群の第 1 仮引数に対する分岐切断線は負の実軸に沿っています。

複素数平方根

float complex csqrtf(float complex z)
double complex csqrt(double complex z)
long double complex csqrtl(long double complex z)

説　明　　複素数平方根を計算します。

ヘッダ　　<complex.h>

リターン値　　正常：　z の複素数平方根値
異常：　定義域エラーの時は、非数を返します。

引　数　　z　　平方根関数値を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=csqrt(z);
```

エラー条件　　z の値が負の値の時、定義域エラーになります。

備　考　　csqrt 関数群の分岐分断線は負の実軸に沿っています。
csqrt 関数群のリターン値の領域は虚軸を含む右半平面です。

偏角

float cargf(float complex z)
double carg(double complex z)
long double cargl(long double complex z)

説　明　　偏角を計算します。

ヘッダ　　<complex.h>

リターン値　　z の偏角値

引　数　　z　　偏角値を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=carg(z);
```

備　考　　carg 関数群の分岐切断線はは負の実軸に沿っています。
carg 関数群のリターン値の範囲は区間[-π,+π]です。

虚部

float cimagf(float complex z)
double cimag(double complex z)
long double cimagl(long double complex z)

説　明	虚部を計算します。
ヘッダ	<complex.h>
リターン値	実数としての z の虚部値
引　数	z　　虚部を求める複素数
例	#include <complex.h> double complex z, ret; ret=cimag(z);

複素共役

float complex conjf(float complex z)
double complex conj(double complex z)
long double complex conjl(long double complex z)

説　明	虚部の符号を反転させて複素共役を計算します。
ヘッダ	<complex.h>
リターン値	z の複素共役値
引　数	z　　複素共役値を求める複素数
例	#include <complex.h> double complex z, ret; ret=conj(z);

リーマン球面上への射影

float complex cprojf(float complex z)
double complex cproj(double complex z)
long double complex cprojl(long double complex z)

説　明　　リーマン球面上への射影を計算します。

ヘッダ　　<complex.h>

リターン値　　リーマン球面上への z の射影値

引　数　　z　　リーマン球面上への射影値を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=cproj(z);
```

実部

float crealf(float complex z)
double creal(double complex z)
long double creall(long double complex z)

説　明　　実部を計算します。

ヘッダ　　<complex.h>

リターン値　　z の実部値

引　数　　z　　実部値を求める複素数

例
```
#include <complex.h>
double complex z, ret;
   ret=creal(z);
```

（16）<fenv.h>

浮動小数点環境へアクセスします。

以下は、すべて処理系定義です。

種別	定義名	説明
型	fenv_t	浮動小数点環境全体の型です。
(マクロ)	fexcept_t	浮動小数点状態フラグの型です。
定数 (マクロ)	FE_DIVBYZERO	浮動小数点例外をサポートするときに定義されるマクロです。
	FE_INEXACT	
	FE_INVALID	
	FE_OVERFLOW	
	FE_UNDERFLOW	
	FE_ALL_EXCEPT	
定数 (マクロ)	FE_DOWNWARD	浮動小数点数の丸め方向のマクロです。
	FE_TONEAREST	
	FE_TOWARDZERO	
	FE_UPWARD	
定数 (マクロ)	FE_DFL_ENV	プログラム既定の浮動小数点環境です。
関数	feclearexcept	浮動小数点例外のクリアを試みます。
	fegetexceptflag	浮動小数点フラグの状態のオブジェクトへの格納を試みます。
	feraiseexcept	浮動小数点例外の生成を試みます。
	fesetexceptflag	浮動小数点フラグのセットを試みます。
	fetestexcept	浮動小数点フラグがセットされているか確認します。
	fegetround	丸め方向を取得します。
	fesetround	丸め方向を設定します。
	fegetenv	浮動小数点環境の取得を試みます。
	feholdexcept	浮動小数点環境を保存し、浮動小数点状態フラグをクリアし、浮動小数点例外について無停止モードに設定します。
	fesetenv	浮動小数点環境の設定を試みます。
	feupdateenv	浮動小数点例外の自動記憶域への保存、浮動小数点環境の設定、保存していた浮動小数点例外の生成を試みます。

例外クリア

long feclearexcept(long e)

説　明　浮動小数点例外のクリアを試みます。

ヘッダ　<fenv.h>

リターン値　正常：　0
異常：　0 以外

引　数　e　浮動小数点例外

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret, e;
   ret=feclearexcept(e);
```

備　考　本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

例外フラグ状態取得

*long fegetexceptflag(fexcept_t *f, long e)*

説　明　例外フラグの状態を取得します。

ヘッダ　<fenv.h>

リターン値　正常：　0
異常：　0 以外

引　数　f　例外フラグ状態の格納先を指すポインタ
e　状態を取得する例外フラグを表す値

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret;
fexcept_t f;
   ret=fegetexceptflag(&f, e);
```

備　考　本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

例外の生成

long feraiseexcept(long e)

説　明　　浮動小数点例外の生成を試みます。

ヘッダ　　<fenv.h>

リターン値　　正常：　0
異常：　0 以外

引　数　　e　　生成を試みる例外を指す値

例

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret, e;
   ret=feraiseexcept(e);
```

備　考　　feraiseexcept 関数が，“ オーバフロー ” 浮動小数点例外又は “ アンダフロー ” 浮動小数点例外を生成する際に “ 不正確結果 ” 浮動小数点例外を生成するかどうかは，処理系定義とします。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

例外フラグ状態設定

*long fesetexceptflag(const fexcept_t *f, long e)*

説　明　　例外フラグの状態を設定します。

ヘッダ　　<fenv.h>

リターン値　　正常：　0
異常：　0 以外

引　数　　f　　例外フラグ状態の取得元を指すポインタ
e　　状態を設定する例外フラグを表す値

例

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
fexcept_t f;
   fegetexceptflag(&f, FE_OVERFLOW) /* フラグ状態保存 */
   fesetexceptflag(&f,FE_OVERFLOW); /* フラグ状態設定 */
```

備　考　　fesetexceptflag 関数を呼ぶ前にフラグ状態の取得元を fegetexceptflag 関数にて設定してください。
fesetexceptflag 関数は浮動小数点例外を生成せず、フラグの状態だけを設定します。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

例外フラグ状態判定

long fetestexcept(long e)

説　明　　例外フラグの状態を判定します。

ヘッダ　　<fenv.h>

リターン値　　e と浮動小数点例外マクロのビット単位の論理和

引　数　　e　　状態を判定するフラグ(複数可)を表す値

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int e = fetestexcept(FE_INVALID | FE_OVERFLOW);
if (e & FE_INVALID) fnc1();
if (e & FE_OVERFLOW) fnc2();
```

備　考　　fetestexcept 関数は 1 回の呼び出しで複数個の浮動小数点例外を判定することができます。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

丸め方向取得

long fegetround(void)

説　明　　その時点の丸め方向を取得します。

ヘッダ　　<fenv.h>

リターン値　　正常：　0
異常：　丸め方向マクロ値が存在しない
　　又は丸め方向を決めることができない場合は負の値

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret = fgetround();
```

備　考　　本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

丸め方向設定

long fesetround(long rnd)

説　明　その時点の丸め方向を設定します。

ヘッダ　<fenv.h>

リターン値　成功した場合のみに 0

例
```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
#pragma STDC FENV_ACCESS ON
    int save_round;
    int setround_ok;
    save_round = fegetround();
    setround_ok = fesetround(round_dir);
    assert(setround_ok == 0);
    fesetround(save_round);
}
```

備　考　fesetround 関数に丸め方向マクロの値と等しくない変更を要求した場合
丸め方向を変更しません。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

浮動小数点環境取得

*long fegetenv(fenv_t *f)*

説　明　浮動小数点環境を取得します。

ヘッダ　<fenv.h>

リターン値　正常：　0
異常：　0 以外

引　数　f　浮動小数点環境格納先を指すポインタ

例
```
#include <fenv.h>
int ret, fenv_t f;
    ret=fegetenv(f);
```

備　考　本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

浮動小数点環境保存

*long feholdexcept(fenv_t *f)*

説　明　　浮動小数点環境を保存します。

ヘッダ　　<fenv.h>

リターン値　　成功した場合のみに 0

引　数　　f　　浮動小数点環境を指すポインタ

例

```
#include <fenv.h>
int ret, fenv_t f;
   ret=feholdexcept(&f);
```

備　考　　feholdexcept 関数は浮動小数点関数環境保存時に浮動小数点状態フラグをクリアし、すべての浮動小数点例外について、無停止(non-stop)モードを設定します。無停止モード設定時は浮動小数点例外発生時も実行を継続します。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

環境設定

*long fesetenv(const fenv_t *f)*

説　明　　浮動小数点環境を設定します。

ヘッダ　　<fenv.h>

リターン値　　正常：　0
異常：　0 以外

引　数　　f　　浮動小数点環境を指すポインタ

例

```
#include <fenv.h>
int ret, fenv_t f;
   ret=fesetenv(f);
```

備　考　　設定する環境は fegetenv 関数または feholdexcept 関数にて設定した環境か、浮動小数点環境マクロと等しい環境を指定していください。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

*long feupdateenv(const fenv_t *f)*

説　明　既出の例外を残したまま浮動小数点環境を設定します。

ヘッダ　<fenv.h>

リターン値　正常：　0
異常：　0 以外

引　数　f　設定する浮動小数点環境を指すポインタ

例

```
#include <fenv.h>
double f(double x)
{
#pragma STDC FENV_ACCESS ON
    double ret;
    fenv_t prev_env;
    if (feholdexcept(&prev_env))
        return /* 環境に問題があった */;
    // ret を計算する
    if (/* 見せかけのアンダフローかどうかを調べる */)
        if (feclearexcept(FE_UNDERFLOW))
            return /* 環境に問題があった */;
        if (feupdateenv(&prev_env))
            return /* 環境に問題があった */;
    return ret;
}
```

備　考　設定する浮動小数点環境は、fegetenv 関数または feholdexcept 関数の呼出しによって設定されたオブジェクトを指すか，又は浮動小数点環境マクロに等しいものにしてください。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

（17）<inttypes.h>

整数型を拡張します。

以下は、すべて処理系定義です。

種別	定義名	説明
型	Imaxdiv_t	imaxdiv 関数の返す値の型です。
(マクロ)		
変数	PRIdN	
(マクロ)	PRIdLEASTN	
	PRIdFASTN	
	PRIdMAX	
	PRIdPTR	
	PRIiN	
	PRIiLEASTN	
	PRIiFASTN	
	PRIiMAX	
	PRIiPTR	
	PRIoN	
	PRIoLEASTN	
	PRIoFASTN	
	PRIoMAX	
	PRIoPTR	
	PRIuN	
	PRIuLEASTN	
	PRIuFASTN	
	PRIuMAX	
	PRIuPTR	
	PRIxN	
	PRIxLEASTN	
	PRIxFASTN	
	PRIxMAX	
	PRIxPTR	
	PRIXN	
	PRIXLEASTN	
	PRIXFASTN	
	PRIXMAX	
	PRIXPTR	

種別	定義名	説明
変数	SCNdN	
(マクロ)	SCNdLEASTN	
	SCNdFASTN	
	SCNdMAX	
	SCNdPTR	
	SCNiN	
	SCNiLEASTN	
	SCNiFASTN	
	SCNiMAX	
	SCNiPTR	
	SCNoN	
	SCNoLEASTN	
	SCNoFASTN	
	SCNoMAX	
	SCNoPTR	
	SCNuN	
	SCNuLEASTN	
	SCNuFASTN	
	SCNuMAX	
	SCNuPTR	
	SCNxN	
	SCNxLEASTN	
	SCNxFASTN	
	SCNxMAX	
	SCNxPTR	
関数	imaxabs	絶対値を計算する。
	imaxdiv	商、剰余を計算する。
	strtoimax strtoumax	文字列最初の部分を intmax_t 型および uintmax_t 型表現に変換する以外は、strtol, strtoll, strtoul および strtoull 関数と等価。
	wcstoimax wcstoumax	ワイド文字列最初の部分を intmax_t 型および uintmax_t 型表現に変換する以外は、wcstol, wcstoll, wcstoul および wcstoull 関数と等価。

絶対値

intmax_t imaxabs(intmax_t a)

説　明　　絶対値を計算します。

ヘッダ　　<inttypes.h>

リターン値　　a の絶対値

引　数　　a　　絶対値を求める値

例

```
#include <inttypes.h>
intmax a, ret;
   ret=imaxabs(a);
```

除算

intmaxdiv_t imaxdiv(intmax_t n, intmax_t d)

説　明　　除算を行います。

ヘッダ　　<inttypes.h>

リターン値　　商と剰余から成る除算結果

引　数　　n　　除算をする値
d

例

```
#include <inttypes.h>
intmax_t n, m;
intmaxdiv_t ret;
   ret=imaxdiv(n, m);
```

文字列を intmax_t 型に変換

*intmax_t strtoimax(const char *nptr, char **endptr, long base)*
*uintmax_t strtoumax(const char *nptr, char **endptr, long base)*

説　明　　数を表現する文字列を intmax_t 型の整数に変換します。

ヘッダ　　<inttypes.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された intmax_t 型の整数値
異常： 変換後の値がオーバフローの時：INTMAX_MAX, INTMAX_MIN または UINTMAX_MAX

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <inttypes.h>
intmax_t ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoimax(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno に ERANGE を設定します。

備　考　　strtoimax 関数及び strtoumax 関数は文字列の最初の部分をそれぞれ intmax_t 型および uintmax_t 型整数に変換するという点を除いて、strtol 関数、strtoll 関数、strtoul 関数及び strtoull 関数と等価とします。

ワイド文字列を整数に変換

*intmax_t wcstoimax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)*
*uintmax_t wcstoumax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)*

説　明　　数を表現する文字列を intmax_t 型または uintmax_t 型の整数に変換します。

ヘッダ　　<stddef.h>, <inttypes.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された intmax_t 型の整数値
異常： 変換後の値がオーバフローの時：INTMAX_MAX, INTMAX_MIN または UINTMAX_MAX

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <stddef.h>
#include <inttypes.h>
intmax_t ret;
const char *nptr;
char **endptr;
int base;
   ret=wcstoimax(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno に ERANGE を設定します。

備　考　　wcstrtoimax 関数及び wcstrtoumax 関数は文字列の最初の部分をそれぞれ intmax_t 型および uintmax_t 型整数に変換するという点を除いて、wcstol 関数、wcstoll 関数、wcstoul 関数及び wcstoull 関数と等価とします。

（18）<iso646h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	and	&&
	and_eq	&=
	bitand	&
	bitor	|
	compl	~
	not	!
	not_eq	!=
	or	||
	or_eq	|=
	xor	^
	xor_eq	^=

（19）<stdbool.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ（変数）	bool	_Bool に展開します。
マクロ（定数）	true	1 に展開します。
	false	0 に展開します。
	__bool_true_false_are_defined	1 に展開します。

（20）<stdint.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	int_least8_t uint_least8_t int_least16_t uint_least16_t int_least32_t uint_least32_t int_least64_t uint_least64_t	8,16,32 および 64 ビットに対する、それぞれの符号あり/なし整数型を少なくとも格納できる大きさを持つ型です。
	int_fast8_t uint_fast8_t int_fast16_t uint_fast16_t int_fast32_t uint_fast32_t int_fast64_t uint_fast64_t	8,16,32 および 64 ビットに対する、それぞれの符号あり/なし整数型を最速で演算できる型です。
	intptr_t uintptr_t	void へのポインタを相互変換可能な符号あり/なし整数型です。
	intmax_t uintmax_t	すべての符号あり/なし整数型のすべての値を表現可能な符号あり/なし整数型です。
	int*N*_t uint*N*_t	N ビットの幅をもつ符号あり/なし整数型です。
	INT*N*_MIN	幅指定符号あり整数型の最小値です。
	INT*N*_MAX	幅指定符号あり整数型の最大値です。
	UINT*N*_MAX	幅指定符号なし整数型の最大値です。
	INT_LEAST*N*_MIN	最小幅指定符号あり整数型の最小値です。
	INT_LEAST*N*_MAX	最小幅指定符号あり整数型の最大値です。
	UINT_LEAST*N*_MAX	最小幅指定符号なし整数型の最大値です。
	INT_FAST*N*_MIN	最速最小幅指定符号あり整数型の最小値です。
	INT_FAST*N*_MAX	最速最小幅指定符号あり整数型の最大値です。
	UINT_FAST*N*_MAX	最速最小幅指定符号なし整数型の最大値です。
	INTPTR_MIN	ポインタ保持可能な符号あり整数型の最小値です。
	INTPTR_MAX	ポインタ保持可能な符号あり整数型の最大値です。
	UINTPTR_MAX	ポインタ保持可能な符号なし整数型の最大値です。

種別	定義名	説明
マクロ	INTMAX_MIN	最大幅符号あり整数型の最小値です。
	INTMAX_MAX	最大幅符号あり整数型の最大値です。
	UINTMAX_MAX	最大幅符号なし整数型の最大値です。
	PTRDIFF_MIN	-65535
	PTRDIFF_MAX	+65535
	SIG_ATOMIC_MIN	-127
	SIG_ATOMIC_MAX	+127
	SIZE_MAX	65535
	WCHAR_MIN	0
	WCHAR_MAX	65535U
	WINT_MIN	0
	WINT_MAX	4294967295U
関数 (マクロ)	INT*N*_C	Int_least*N*_t に対応する整数定数式に展開します。
	UINT*N*_C	uInt_least*N*_t に対応する整数定数式に展開します。
	INT_MAX_C	intmax_t の整数定数式に展開します。
	UINT_MAX_C	uintmax_t の整数定数式に展開します。

（21）<tgmath.h>

以下は、すべてマクロ定義です。

型総称マクロ	<math.h>の関数	<complex.h>の関数
acos	acos	cacos
asin	asin	casin
atan	atan	catan
acosh	acosh	cacosh
asinh	asinh	casinh
atanh	atanh	catanh
cos	cos	ccos
sin	sin	csin
tan	tan	ctan
cosh	cosh	ccosh
sinh	sinh	csinh
tanh	tanh	ctanh
exp	exp	cexp
log	log	clog
pow	pow	cpow
sqrt	sqrt	csqrt
fabs	fabs	cfabs
atan2	atan2	-
cbrt	cbrt	-
ceil	ceil	-
copysign	copysign	-
erf	erf	-
erfc	erfc	-
exp2	exp2	-
expm1	expm1	-
fdim	fdim	-
floor	floor	-
fma	fma	-
fmax	fmax	-
fmin	fmin	-
fmod	fmod	-
frexp	frexp	-
hypot	hypot	-
ilogb	ilogb	-

型総称マクロ	<math.h>の関数	<complex.h>の関数
ldexp	ldexp	-
lgamma	lgamma	-
llrint	llrint	-
llround	llround	-
log10	log10	-
log1p	log1p	-
log2	log2	-
logb	logb	-
lrint	lrint	-
lround	lround	-
nearbyint	nearbyint	-
nextafter	nextafter	-
nexttoward	nexttoward	-
remainder	remainder	-
remquo	remquo	-
rint	rint	-
round	round	-
scalbn	scalbn	-
scalbln	scalbln	-
tgamma	tgamma	-
trunc	trunc	-
carg	-	carg
cimag	-	cimag
conj	-	conj
cproj	-	cproj
creal		creal

（22）<wchar.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	mbstate_t	多バイト文字の並びとワイド文字の並びの間に必要な変換の状態を保持する型です。
	wint_t	拡張文字を保持する型です。
定数（マクロ）	WEOF	ファイルの終わりを表します。

種別	定義名	説明
関数	fwprintf	出力形式を変換して、ストリームへ出力します。
	vfwprintf	可変個数の実引数並びを va_list で置き換えた fwprintf と等価です。
	swprintf	出力形式を変換してワイド文字の配列に書き込みます。
	vswprintf	可変個数の実引数並びを va_list で置き換えた swprintf と等価です。
	wprintf	与えられた実引数の前に stdout を実引数として付加した fwprintf と等価です。
	vwprintf	可変個数の実引数並びを va_list で置き換えた wprintf と等価です。
	fwscanf	ワイド文字列の制御に従ってストリームから入力して変換し、オブジェクトに代入します。
	vfwscanf	可変個数の実引数並びを va_list で置き換えた fwscanf と等価です。
	swscanf	ワイド文字列の制御に従って変換し、オブジェクトに代入します。
	vswscanf	可変個数の実引数並びを va_list で置き換えた swscanf と等価です。
	wscanf	与えられた実引数の前に stdin を実引数として付加した fwscanf と等価です。
	vwscanf	可変個数の実引数並びを va_list で置き換えた wscanf と等価です。
	fgetwc	wchar_t 型として取り込み wint_t 型に変換します。
	fgetws	ワイド文字の列を配列に格納します。
	fputwc	ワイド文字を書き込みます。
	fputws	ワイド文字列を書き込みます。
	fwide	入出力の単位を設定します。
	getwc	fgetwc と等価です。
	getwchar	実引数に stdin を指定した getwc と等価です。
	putwc	fputwc と等価です。
	putwchar	第２引数に stdout を指定した putwc と等価です。
	ungetwc	ワイド文字をストリームに戻します。
	wcstod wcstof wcstold	ワイド文字列の最初の部分を double, float および long double 型の表現に変換します。
	wcstol wcstoll wcstoul wcstoull	ワイド文字列の最初の部分を long int, long long int, unsigned long int および unsigned long long int 型の表現に変換します。
	wcscpy	ワイド文字列をコピーします。
	wcsncpy	n 個以下のワイド文字をコピーします。
	wmemcpy	n ワイド文字をコピーします。
	wmemmove	n ワイド文字をコピーします。
	wcscat	ワイド文字列をコピーし、ワイド文字列の最後に付加します。

種別	定義名	説明
関数	wcsncat	n 個以下のワイド文字列をコピーし、ワイド文字列の最後に付加します。
	wcscmp	ワイド文字列同士を比較します。
	wcsncmp	n ワイド文字以下の配列を比較します。
	wmemcmp	n ワイド文字を比較します。
	wcschr	ワイド文字列の中でワイド文字を検索します。
	wcscspn	ワイド文字列の中で、ワイド文字列が含まれているかを検索します。
	wcspbrk	ワイド文字列の中で、ワイド文字列が含まれている最初の位置を検索します。
	wcsrchr	ワイド文字列の中でワイド文字が最後に現れる位置を検索します。
	wcsspn	ワイド文字列の中から、ワイド文字を含む先頭部分の最大の長さを計算します。
	wcsstr	ワイド文字列の中からワイド文字の並びをが最初に現れる位置を検索します。
	wcstok	ワイド文字列をワイド文字で区切られる字句の列に分割します。
	wmemchr	オブジェクトの先頭から n ワイド文字の中でワイド文字が最初に現れる位置を検索します。
	wcslen	ワイド文字列の長さを計算します。
	wmemset	n ワイド文字をコピーします。
	wctob	多バイト文字表現が１バイトに可能か判定します。
	mbsinit	初期変換状態か判定します。
	mbrlen	多バイト文字を構成するバイト数を計算します。
	mbrtowc	多バイト文字をワイド文字に変換します。
	wcrtomb	ワイド文字を多バイト文字に変換します。
	mbsrtowcs	多バイト文字の並びを対応するワイド文字の並びに変換します。
	wcsrtombs	ワイド文字の並びを対応する多バイト文字の並びに変換します。

ワイド文字版書式付きファイル出力

*long fwprintf(FILE *restrict fp, const wchar_t *restrict control [, arg] …)*

説　明　　書式に従って、ストリーム入出力用ファイルへデータを出力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：　変換し出力したワイド文字列数
異常：　負の値

引　数

fp	ファイルポインタ
control	書式を示すワイド文字列へのポインタ
arg, …	書式に従って出力されるデータの並び

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L”%ls”;
int ret;
wchar_t buffer[]=L”Hello World\n”;
   ret=fwprintf(fp, control, buffer);
```

エラー条件

備　考　　fwprintf 関数は fprintf 関数のワイド文字対応版です。

ワイド文字版可変個引数書式付きファイル出力

*long vfwprintf(FILE *restrict fp, const char *restrict control, va_list arg)*

説　明　　可変個の引数リストを書式に従って、指定したストリーム入出力用ファイルに出力します。

ヘッダ　　<stdarg.h>, <stdio.h>, <wchar.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

fp	ファイルポインタ
control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int  ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfprintf(fp, control, ap);
   va_end(ap);
}
```

備　考　　vfwprintf 関数は vfprintf 関数のワイド文字対応版です。

書式付きワイド文字列出力

*long swprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control [, arg] …)*

説　明　　データを書式に従って変換し、指定した領域へ出力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：　変換した文字数
異常：　表現形式エラー又は
n 個以上のワイド文字の書き込みが要求された場合は負の値

引　数

s	データを出力する記憶域へのポインタ
n	出力するワイド文字数
control	書式を示すワイド文字列へのポインタ
arg,…	書式に従って出力されるデータ

例

```
#include <stdio.h>
#include <wchar.h>
wchar_t s*;
size_t n=12;
const wchar_t *control=”%ls”;
int ret;
wchar_t buffer[]=”Hello World\n”;
   ret=swprintf(s, n, control, buffer);
```

エラー条件　　mbrtowc()関数に不正な多バイト文字列を渡した場合、表現形式エラーが発生します。

備　考　　swprintf 関数は sprintf 関数のワイド文字対応版です。

ワイド文字版可変個引数文字列出力

*long vswprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control, va_list arg)*

説　明　　可変個の引数リストを書式に従って、指定した記憶域に出力します。

ヘッダ　　<stdarg.h>, <wchar.h>

リターン値　　正常：変換した文字数
異常：負の数

引　数

s	データを出力する記憶域へのポインタ
n	出力するワイド文字数
control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <wchar.h>
wchar_t *s;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++) {
      ret=vswprintf(s, control, ap);
      va_arg(ap,int);
      s += ret;
   }
}
```

備　考　　vswprintf 関数は、vsprintf 関数のワイド文字対応版です。

書式付きワイド文字出力

*long wprintf(const wchar_t *restrict control [, arg] …)*

説　明　　データを書式に従って変換し、標準出力ファイル(stdout)へ出力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：変換し出力したワイド文字数
異常：負の値

引　数

control	書式を示す文字列へのポインタ
arg,…	書式に従って出力されるデータ

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *control=L"%ls";
int ret;
wchar_t buffer[]=L"Hello World\n";
   ret=wprintf(control,buffer);
```

備　考　　wprintf 関数は printf 関数のワイド文字対応版です。

可変個引数ワイド文字出力

*long vwprintf(const wchar_t *restrict control, va_list arg)*

説　明　　可変個の引数リストを書式に従って標準出力ファイル(stdout)に出力します。

ヘッダ　　<stdarg.h>, <wchar.h>

リターン値　　正常：変換し出力した文字数
異常：負の値

引　数

control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int ret;

void wprlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vwprintf(control, ap);
   va_end(ap);
}
```

備　考　　vwprintf 関数は vprintf 関数のワイド文字対応版です。

書式付きワイド文字ファイル入力

*long fwscanf(FILE *restrict fp, const wchar_t *restrict control [, ptr] …)*

説　明　　ストリーム入出力ファイルからデータを入力し、書式に従って変換します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：　入力変換に成功したデータの個数
異常：　入力データの変換を行う前に入力データが終了した時：EOF

引　数

fp	ファイルポインタ
control	書式を示すワイド文字列へのポインタ
ptr	入力したデータを格納する記憶域へのポインタ

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L”%d”;
int ret, buffer[10];
   ret=fwscanf(fp, control, buffer);
```

備　考　　fwscanf 関数は、fscanf 関数のワイド文字対応版です。

書式付き可変個引数ワイド文字ファイル入力

*long vfwscanf(FILE *restrict fp, const wchar_t *restrict control , va_list arg)*

説　明　　ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <stdio.h>, <wchar.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

fp	ファイルポインタ
control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfwscanf(fp, control, ap);
   va_end(ap);
}
```

備　考　　vfwscanf 関数は vfscanf 関数のワイド文字対応版です。

書式付きワイド文字列入力

*long swscanf(const wchar_t *restrict s, const wchar_t *restrict control [, ptr] …)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：EOF

引　数

s	入力するデータがある記憶域
control	書式を示すワイド文字列へのポインタ
ptr,…	入力変換したデータを格納する記憶域へのポインタ

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *s, *control=L"%d";
int ret,buffer[10];
ret=swscanf(s, control, buffer);
```

備　考　　swscanf 関数は sscanf 関数のワイド文字対応版です。

書式付き可変個引数ワイド文字列入力

*long vswscanf(const wchar_t *restrict s, const wchar_t *restrict control, va_list arg)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <wchar.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：EOF

引　数

s	入力するデータがある記憶域
control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <wchar.h>
const wchar_t *s, *control=L"%d";
int ret,buffer[10];
ret=vswscanf(s, control, buffer);
```

書式付きワイド文字入力

*long wscanf(const wchar_t *control [, ptr] …)*

説　明　　標準入力ファイル(stdin)からデータを入力し、書式に従って変換します。

ヘッダ　　<wchar.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：EOF

引　数

control	書式を示すワイド文字列へのポインタ
ptr,…	入力変換したデータを格納する記憶域へのポインタ

例

```
#include <wchar.h>
const wchar_t *control=L"%d";
int ret,buffer[10];
   ret=wscanf(control, buffer);
```

備　考　　wscanf 関数は scanf 関数のワイド文字対応版です。

書式付き可変個引数ワイド文字ファイル入力

*long vwscanf(const wchar_t *restrict control , va_list arg)*

説　明　　指定した記憶域からデータを入力し、書式に従って変換します。

ヘッダ　　<stdarg.h>, <wchar.h>

リターン値　　正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

引　数

control	書式を示すワイド文字列へのポインタ
arg	引数リスト

例

```
#include <stdarg.h>
#include <wchar.h>

FILE *fp;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vwscanf(control, ap);
   va_end(ap);
}
```

備　考　　vwscanf 関数は、vscanf 関数をワイド文字列の書式を使えるようにした関数です。

ファイルから1つのワイド文字を入力

*wint_t fgetwc(FILE *fp)*

説　明　　ストリーム入出力用ファイルから１つのワイド文字を入力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：ファイルの終了の時　　：EOF
　　ファイルの終了でない時　：入力したワイド文字
　　異常：EOF

引　数　　fp　　ファイルポインタ

例
```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wint_t ret;
   ret=fgetwc(fp);
```

エラー条件　　読み込みエラーが発生した時、そのファイルに対してのエラー指示子が設定されます。

備　考　　fgetwc 関数は fgetc 関数をワイド文字が入力できるようにした関数です。

ファイルからワイド文字列入力

*wchar_t *fgetws(wchar_t *restrict s, long n, FILE *fp)*

説　明　　ストリーム入出力用ファイルからワイド文字列を入力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：ファイルの終了の時　　：NULL
　　ファイルの終了でない時　：s
　　異常：NULL

引　数　　s　　ワイド文字列を入力する記憶域へのポインタ
　　n　　ワイド文字列を入力する記憶域のバイト数
　　fp　　ファイルポインタ

例
```
#include <stdio.h>
#include <wchar.h>
wchar_t *s, *ret;
int n;
FILE *fp;
   ret=fgetws(s,n,fp);
```

備　考　　fgetws 関数は fgets 関数をワイド文字列が入力できるように対応した関数です。

ファイルに 1 つのワイド文字出力

*wint_t fputwc(wchar_t c, FILE *fp)*

説　明　ストリーム入出力用ファイルへ１つのワイド文字を出力します。

ヘッダ　<stdio.h>, <wchar.h>

リターン値　正常：出力したワイド文字
異常：EOF

引　数

c	出力する文字
fp	ファイルポインタ

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wchar_t c;
wint_t ret;
   ret=fputwc(c,fp);
```

エラー条件　書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

備　考　fputwc 関数は fputc 関数のワイド文字対応版です。

ファイルにワイド文字列出力

*long fputws(const wchar_t *restrict s, FILE *restrict fp)*

説　明　ストリーム入出力用ファイルへワイド文字列を出力します。

ヘッダ　<stdio.h>, <wchar.h>

リターン値　正常：0
異常：EOF

引　数

s	出力するワイド文字列へのポインタ
fp	ファイルポインタ

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *s;
int ret;
FILE *fp;
   ret=fputws(s,fp);
```

備　考　fputws 関数は fputs 関数のワイド文字対応版です。

ファイルへの入力単位設定

*long fwide(FILE *fp, long mode)*

説　明　　ファイルへの入力単位を設定します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　ワイド文字単位が設定された場合は 0 より大きい値
バイト単位の場合は 0 より小さい値
入出力単位をもたない場合は 0

引　数

fp	ファイルポインタ
mode	入力単位を表す値

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
int mode, ret;
   ret=fwide(fp,mode);
```

備　考　　fwide 関数はストリーム入出力単位が既に決定されている場合、それを変更しません。

ファイルから 1 つのワイド文字入力

*long getwc(FILE *fp)*

説　明　　ストリーム入出力用ファイルから 1 つのワイド文字を入力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常： ファイルの終了の時　　：WEOF
　　ファイルの終了でない時　：入力した文字
異常： EOF

引　数

fp	ファイルポインタ

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
int ret;
   ret=getwc(fp);
```

エラー条件　　読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

備　考　　getwc 関数は fgetwc と等価ですが、マクロとして実装されているため、fp を 2 回以上評価することがあります。したがって、fp は副作用を伴わない式にしてください。

1 つのワイド文字入力

long getwchar(void)

説　明　　標準入力ファイル(stdin)から、1 つのワイド文字を入力します。

ヘッダ　　<wchar.h>

リターン値　　正常： ファイルの終了の時　　：WEOF
　　ファイルの終了でない時　：入力したワイド文字
異常： EOF

例

```
#include <wchar.h>
int ret;
   ret=getwchar();
```

エラー条件　　読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

備　考　　getwchar 関数は getchar 関数のワイド文字対応版です。

ファイルに 1 つのワイド文字出力

*wint_t putwc(wchar_t c, FILE *fp)*

説　明　　ストリーム入出力用ファイルへ 1 つのワイド文字を出力します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常： 出力したワイド文字
異常： WEOF

引　数

c	出力するワイド文字
fp	ファイルポインタ

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wchar_t c;
wint_t ret;
   ret=putwc(c,fp);
```

エラー条件　　書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

備　考　　putwc 関数は fputwc と等価ですが、マクロとして実装されているため、fp を 2 回以上評価することがあります。したがって、fp は副作用を伴わない式にしてください。

1 つのワイド文字出力

wint_t putwchar(wchar_t c)

説　明　　標準出力ファイル(stdout)へ１つのワイド文字を出力します。

ヘッダ　　<wchar.h>

リターン値　　正常：出力したワイド文字
異常：WEOF

引　数　　c　　出力するワイド文字

例
```
#include <wchar.h>
wint_t ret;
wchar_t c;
   ret=putwchar(c);
```

エラー条件　　書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

備　考　　putwchar 関数は putchar 関数のワイド文字対応版です。

ファイルに 1 つのワイド文字返却

*wint_t ungetwc(wint_t c, FILE *fp)*

説　明　　ストリーム入出力用ファイルへ１つのワイド文字を戻します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：戻したワイド文字
異常：WEOF

引　数　　c　　戻すワイド文字
fp　　ファイルポインタ

例
```
#include <stdio.h>
#include <wchar.h>
wint_t ret;
wchar_t c;
FILE *fp;
   ret=ungetwc(c,fp);
```

備　考　　ungetwc 関数は、ungetc 関数のワイド文字対応版です。

ワイド文字列を浮動小数点値に変換

*double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr)*
*float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr)*
*long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr)*

説　明　ワイド文字列の最初の部分を所定の型の浮動小数点値に変換します。

ヘッダ　<wchar.h>

リターン値　正常： nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
nptr が指している文字列が浮動小数点型を構成する文字で始まっている時
：変換された型の浮動小数点値
異常： 変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ
HUGE_VAL, HUGE_VALF, HUGE_VALL
変換後の値がアンダフローの時：0

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

例

```
#include <wchar.h>
const wchar_t *nptr;
wchar_t **endptr;
double ret;
   ret=wcstod(nptr,endptr);
```

エラー条件　変換後の値がオーバフロー/アンダフローをおこした時は errno を設定します。

備　考　wcstod 関数群は strtod 関数群のワイド文字対応版です。

ワイド文字列を整数値に変換

*long int wcstol(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)*
*long long int wcstoll(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)*
*unsigned long int wcstoul(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)*
*unsigned long long int wcstoull(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)*

説　明　　ワイド文字列の最初の部分を所定の型の整数値に変換します。

ヘッダ　　<wchar.h>

リターン値　　正常： nptr が指している文字列が整数を構成しない文字で始まっている時：0
nptr が指している文字列が整数を構成する文字で始まっている時
：変換された型の整数値
異常： 変換後の値がオーバフローの時：変換する文字列の符号に従って
LONG_MIN，LONG_MAX，LLONG_MIN，LLONG_MAX，ULONG_MAX 又は ULLONG_MAX

引　数

nptr	変換する数を表現する文字列へのポインタ
endptr	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base	変換の基数(0 又は 2～36)

例

```
#include <wchar.h>
long ret;
const wchar_t *nptr;
wchar_t **endptr;
int base;
   ret=wcstoull(nptr,endptr,base);
```

エラー条件　　変換後の値がオーバフローをおこした時は、errno を設定します。

備　考　　wcstol 関数群は strtol 関数群のワイド文字対応版です。

ワイド文字列複写

*wchar_t *wcscpy(wchar_t * restrict s1, const wchar_t * restrict s2)*

説　明　　複写元のワイド文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。

ヘッダ　　<wchar.h>

リターン値　　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の文字列へのポインタ

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
   ret=wcscpy(s1,s2);
```

備　考　　wcscpy 関数群は strcpy 関数群のワイド文字対応版です。

ワイド文字列複写

*wchar_t *wcsncpy(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n)*

説　明　　複写元のワイド文字列を指定された文字数分、複写先の記憶域に複写します。

ヘッダ　　<wchar.h>

リターン値　　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の文字列へのポインタ
n	複写する文字数

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
size_t n;
   ret=wcsncpy(s1,s2,n);
```

備　考　　wcsnpy 関数は strncpy 関数のワイド文字対応版です。

記憶域複写

*wchar_t *wmemcpy(wchar_t *restrict s1, const wchar_t *restrict s2, size_t n)*

説　明　複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。

ヘッダ　<wchar.h>

リターン値　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の記憶域へのポインタ
n	複写する文字数

例

```
#include <wchar.h>
wchar_t *ret, *s1;
const wchar_t *s2;
size_t n;
    ret=wmemcpy(s1,s2,n);
```

備　考　wmemcpy 関数は memcpy 関数のワイド文字対応版です。

記憶域移動

*wchar_t *wmemmove(wchar_t *s1, const wchar_t *s2, size_t n)*

説　明　複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
また、複写元と複写先の記憶域が、重なっている部分があっても、複写元の重なっている部分を上書きする前に複写するので正しく複写されます。

ヘッダ　<wchar.h>

リターン値　s1 の値

引　数

s1	複写先の記憶域へのポインタ
s2	複写元の記憶域へのポインタ
n	複写する文字数

例

```
#include <wchar.h>
wchar_t *ret, *s1;
const wchar_t *s2;
size_t n;
    ret=wmemmove(s1,s2,n);
```

備　考　wmemmove 関数は memmove 関数のワイド文字対応版です。

ワイド文字列文字列連結

*wchar_t *wcscat(wchar_t *s1, const wchar_t *s2)*

説　明　　文字列の後に、文字列を連結します。

ヘッダ　　<wchar.h>

リターン値　　s1 の値

引　数　　s1　　連結される文字列へのポインタ
s2　　連結する文字列へのポインタ

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
   ret=wcscat(s1,s2);
```

備　考　　wcscat 関数は strcat 関数のワイド文字対応版です。

文字列連結

*wchar_t *wcsncat(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n)*

説　明　　文字列に文字列を指定した文字数分連結します。

ヘッダ　　<wchar.h>

リターン値　　s1 の値

引　数　　s1　　連結される文字列へのポインタ
s2　　連結する文字列へのポインタ
n　　連結する文字数

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
size_t n;
   ret=wcsncat(s1,s2,n);
```

備　考　　wcsncat 関数は strncat 関数のワイド文字対応版です。

文字列比較

*long wcscmp(const wchar_t *s1, const wchar_t *s2)*

説　明　　指定された 2 つの文字列の内容を比較します。

ヘッダ　　<wchar.h>

リターン値　　s1 で指された文字列 > s2 で指された文字列の時：正の値
s1 で指された文字列 == s2 で指された文字列の時：0
s1 で指された文字列 < s2 で指された文字列の時：負の値

引　数　　s1　　比較される文字列へのポインタ
s2　　比較する文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
int ret;
   ret=wcscmp(s1,s2);
```

備　考　　wcscmp 関数は strcmp 関数のワイド文字対応版です。

文字列比較

*long wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n)*

説　明　　指定された 2 つの文字列を指定された文字分まで比較します。

ヘッダ　　<wchar.h>

リターン値　　s1 で指された文字列 > s2 で指された文字列の時：正の値
s1 で指された文字列 == s2 で指された文字列の時：0
s1 で指された文字列 < s2 で指された文字列の時：負の値

引　数　　s1　　比較される文字列へのポインタ
s2　　比較する文字列へのポインタ
n　　比較する文字数の最大値

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t n;
int ret;
   ret=wcsncmp(s1,s2,n);
```

備　考　　wcsncmp 関数は strncmp 関数のワイド文字対応版です。

記憶域比較

*long wmemcmp(const wchar_t * s1, const wchar_t * s2, size_t n)*

説　明　　指定された 2 つの記憶域の内容を比較します。

ヘッダ　　<wchar.h>

リターン値　　s1 で指された記憶域 > s2 で指された記憶域の時：正の値
s1 で指された記憶域 == s2 で指された記憶域の時：0
s1 で指された記憶域 < s2 で指された記憶域の時：負の値

引　数

s1	比較される記憶域へのポインタ
s2	比較する記憶域へのポインタ
n	比較する記憶域の文字数

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t n;
int ret;
    ret=wmemcmp(s1,s2,n);
```

備　考　　wmemcmp 関数は memcmp 関数のワイド文字対応版です。

最初の文字位置

*wchar_t *wcschr(const wchar_t *s, wchar_t c)*

説　明　　指定された文字列において、指定された文字が最初に現われる位置を検索します。

ヘッダ　　<wchar.h>

リターン値　　検索の結果見つかった時　　：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s	検索を行う文字列へのポインタ
c	検索する文字

例

```
#include <wchar.h>
const wchar_t *s;
int c;
char *ret;
    ret=wcschr(s,c);
```

備　考　　wcschr 関数は strchr 関数のワイド文字対応版です。

指定文字群が最初に現れるまでの文字数

*size_t wcscspn(const wchar_t *s1, const wchar_t *s2)*

説　明　指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

ヘッダ　<wchar.h>

リターン値　s2 が指す文字列を構成する文字以外の文字が構成される文字列 s1 の先頭からの長さ

引　数

s1	調べられる文字列へのポインタ
s2	s1 を調べるための文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t ret;
   ret=wcscspn(s1,s2);
```

備　考　wcscspn 関数は strcspn 関数のワイド文字対応版です。

指定文字群が最初に現れる位置

*wchar_t *wcspbrk(const wchar_t *s1, const wchar_t *s2)*

説　明　指定された文字列内において、別に指定された文字列中の文字が最初に現われる位置を検索します。

ヘッダ　<wchar.h>

リターン値　検索の結果見つかった時　：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s1	検索を行う文字列へのポインタ
s2	s1 内で検索する文字を示す文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
char *ret;
   ret=wcspbrk(s1,s2);
```

備　考　wcspbrk 関数は strpbrk 関数のワイド文字対応版です。

最後の文字位置

*wchar_t *wcsrchr(const wchar_t *s, wchar_t c)*

説　明　　指定された文字列において、指定された文字が最後に現われる位置を検索します。

ヘッダ　　<wchar.h>

リターン値　　検索の結果見つかった時　　：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

引　数　　s　　検索を行う文字列へのポインタ
c　　検索する文字

例

```
#include <wchar.h>
const wchar_t *s;
int c;
wchar_t *ret;
    ret=wcsrchr(s,c);
```

指定文字群が連続する部分の長さ

*size_t wcsspn(const wchar_t *s1, const wchar_t *s2)*

説　明　　指定された文字列を先頭から調べ、別に指定した文字列中の文字が先頭から何文字続くかを求めます。

ヘッダ　　<wchar.h>

リターン値　　s1 の先頭から、s2 で指定した文字が続いている文字数

引　数　　s1　　調べられる文字列へのポインタ
s2　　s1 を調べるための文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t ret;
    ret=wcsspn(s1,s2);
```

備　考　　wcsspn 関数は strspn 関数のワイド文字対応版です。

最初の文字列位置

*wchar_t *wcsstr(const wchar_t *s1, const wchar_t *s2)*

説　明　　指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。

ヘッダ　　<wchar.h>

リターン値　　検索の結果見つかったとき　　：見つけられた文字へのポインタ
検索の結果見つからなかったとき：NULL

引　数

s1	検索を行う文字列へのポインタ
s2	検索する文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
wchar_t *ret;
    ret=wcsstr(s1,s2);
```

字句切り分け

wchar_t wcstok(wchar_t * restrict s1, const wchar_t * restrict s2, wchar_t ** restrict ptr)*

説　明　　指定した文字列をいくつかの字句に切り分けます。

ヘッダ　　<wchar.h>

リターン値　　字句に切り分けられた時　　：切り分けた字句の先頭へのポインタ
字句に切り分けられなかった時：NULL

引　数

s1	いくつかの字句に切り分ける文字列へのポインタ
s2	文字列を切り分けるための文字からなる文字列へのポインタ
ptr	次の関数呼び出し時に検索を始める文字列へのポインタ

例

```
#include <wchar.h>
static wchar_t s1[] = L"?a???b,,,#c";
static wchar_t s2[] = L"＼t ＼t";
wchar_t *t, *p1, *p2;
t = wcstok(s1, L"?", &p1);    // t は字句 L"a"を指す
t = wcstok(NULL, L",", &p1);  // t は字句 L"??b"を指す
t = wcstok(s2, L" ＼t", &p2); // t は NULL ポインタとなる
t = wcstok(NULL, L"#,", &p1); // t は字句 L"c"を指す
t = wcstok(NULL, L"?", &p1);  // t は NULL ポインタとなる
```

備　考　　wcstok 関数は strtok 関数のワイド文字対応版です。
同じ文字列に対して 2 回目以降の検索をする場合は s1 に NULL を設定し、ptr には
前回の同文字列に対する関数呼び出しで取得した値を設定してください。

記憶域内文字検索

*wchar_t *wmemchr(const wchar_t *s, wchar_t c, size_t n)*

説　明　　指定された記憶域において、指定された文字が最初に現われる位置を検索します。

ヘッダ　　<wchar.h>

リターン値　　検索の結果見つかった時　　：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

引　数

s	検索を行う記憶域へのポインタ
c	検索する文字
n	検索を行う文字数

例

```
#include <wchar.h>
const wchar_t *s;
int c;
size_t n;
wchar_t *ret;
    ret=wmemchr(s,c,n);
```

備　考　　wmemchr 関数は memchr 関数のワイド文字対応版です。

ワイド文字列の文字数

*size_t wcslen(const wchar_t *s)*

説　明　　終端ナルワイド文字を除くワイド文字列の長さを計算します。

ヘッダ　　<wchar.h>

リターン値　　ワイド文字列の文字数

引　数

s	長さをワイド文字列へのポインタ

例

```
#include <wchar.h>
const wchar_t *s;
size_t ret;
    ret=wcslen(s);
```

備　考　　wcslen 関数は strlen 関数のワイド文字対応版です。

文字繰り返し

*wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n)*

説　明　　指定された記憶域の先頭から、指定された文字を指定された文字数分設定します。

ヘッダ　　<wchar.h>

リターン値　　s の値

引　数

s	文字が設定される記憶域へのポインタ
c	設定する文字
n	設定する文字数

例

```
#include <stdio.h>
#include <wchar.h>
wchar_t c, *s, *ret;
size_t n;
    ret=wmemset(s,c,n);
```

備　考　　wmemset 関数は memset 関数のワイド文字対応版です。

ワイド文字を 1 バイト表現に変換

long wctob(wint_t c)

説　明　　ワイド文字を１バイト表現に変換します。

ヘッダ　　<stdio.h>, <wchar.h>

リターン値　　正常：　ワイド文字の１バイト値
異常：　EOF

引　数

c	ワイド文字

例

```
#include <stdio.h>
#include <wchar.h>
wint_t c;
int ret;
    ret=wctob(c);
```

エラー条件　　ワイド文字が１バイトで表現できない場合は EOF を返します。

備　考　　wctob 関数は，c が拡張文字集合の要素であり，かつ初期シフト状態では多バイト文字表現が１バイトになるものに対応するかどうかを判定します。

変換状態関数

*long mbsinit(const mbstate_t *ps)*

説　明　　指定された mbstate_t オブジェクトが初期変換状態かどうか判定します。

ヘッダ　　<wchar.h>

リターン値　　初期化状態の場合 0 以外の値
それ以外の状態の場合は 0

引　数

ps	mbstate_t オブジェクトへのポインタ

例

```
#include <wchar.h>
const mbstate_t *mt;
int ret;
   ret=mbsinit(mt);
```

多バイト文字のバイト数取得

*size_t mbrlen(const char * restrict s, size_t n, mbstate_t *restrict ps)*

説　明　　指定された多バイト文字のバイト数を取得します。

ヘッダ　　<wchar.h>

リターン値

0	n 個以下のバイトによってナルワイド文字を認識した場合
1 以上 n 以下	n 個以下のバイトによって多バイト文字を認識した場合
(size_t)(-2)	n 個のバイトだけでは完全な多バイト文字を認識できない場合
(size_t)(-1)	不正な多バイト文字の並びに遭遇した場合

引　数

s	多バイト文字列へのポインタ
n	認識する多バイト文字の最大バイト数
ps	mbstate_t オブジェクトへのポインタ

例

```
#include <wchar.h>
const char *s;
size_t n;
const mbstate_t *mt;
int ret;
   ret=mbrlen(s, n, mt);
```

多バイト文字をワイド文字に変換

*size_t mbrtowc(wchar_t * restrict pwc, const char * restrict s, size_t n, mbstate_t * restrict ps)*

説　明	多バイト文字をワイド文字に変換します。	
ヘッダ	<wchar.h>	
リターン値	0	n 個以下のバイトによってナルワイド文字を認識した場合
	1 以上 n 以下	n 個以下のバイトによって多バイト文字を認識した場合
	(size_t)(-2)	n 個のバイトだけでは完全な多バイト文字を認識できない場合
	(size_t)(-1)	不正な多バイト文字の並びに遭遇した場合
引　数	pwc	取得したワイド文字を格納するワイド文字列へのポインタ
	s	多バイト文字列へのポインタ
	n	認識する多バイト文字の最大バイト数
	ps	mbstate_t オブジェクトへのポインタ

例

```
#include <wchar.h>
wchar_t *pwc;
const char *s;
size_t n, ret;
mbstate_t *ps;
   ret=mbrtowc(pwc, s, n, ps);
```

備　考　不正な多バイト文字の並びに遭遇した場合、マクロ EILSEQ の値を errno に格納し、変換状態は未規定とします。

ワイド文字を多バイト文字に変換

*size_t wcrtomb(char * restrict s, wchar_t wc, mbstate_t * restrict ps)*

説　明　ワイド文字を多バイト文字に変換します。

ヘッダ　<wchar.h>

リターン値　正常：　多バイト文字のバイト数
異常：　(size_t)(-1)　不正な多バイト文字の並びに遭遇した場合

引　数

s	多バイト文字列へのポインタ
wc	変換するワイド文字
ps	mbstate_t オブジェクトへのポインタ

例

```
#include <wchar.h>
wchar_t wc;
char *s;
size_t ret;
mbstate_t *ps;
   ret=wctomb(s, wc, ps);
```

エラー条件　不正な多バイト文字の並びに遭遇した場合、マクロ EILSEQ の値を errno に格納し、変換状態は未規定とします。

備　考　wcrtomb 関数が決定した多バイト文字のバイト数にはシフトシーケンスを含みます。バイト数は MB_CUR_MAX を超えません。変換結果がナルワイド文字であった場合は初期変換状態となりますが、必要であれば初期シフト状態に戻すためのシフトシーケンスをワイド文字の前に格納します。

多バイト文字文字列をワイド文字列に変換

*size_t mbstowcs(wchar_t * restrict pwcs, const char * restrict s, size_t n)*

説　明　　多バイト文字列をワイド文字列に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常：　ワイド文字列へ書き込まれた文字数
異常：　(size_t)(-1)　　不正な多バイト文字の並びに遭遇した場合

引　数

wcs	ワイド文字列へのポインタ
s	多バイト文字列へのポインタ
n	ワイド文字列へ格納されるワイド文字数

例

```
#include <stdlib.h>
wchar_t *pwcs;
const char *s;
size_t n, ret;
   ret=mbstowcs(pwcs, s, n);
```

備　考　　mbstowcs 関数は，s が指す配列中の初期シフト状態で始まる多バイト文字の並びを，対応するワイド文字の並びに変換し，n 個以下のワイド文字を pwcs が指す配列に格納します。ナル文字を発見した場合はナルワイド文字に変換し、変換処理を終了します。各多バイト文字は，mbtowc 関数の変換状態が影響を受けないことを除いて，mbtowc 関数の呼出しによる場合と同じ規則で変換します。領域の重なり合うオブジェクト間でコピーが行われる場合の動作は未定義とします。
正常なリターン値であっても終端文字分のバイトは含みません。
リターン値が n のとき，配列はナル文字で終わっていません。

ワイド文字列を多バイト文字列に変換

*size_t wcstombs(char * restrict s, const wchar_t * restrict pwcs, size_t n)*

説　明　　ワイド文字列を多バイト文字列に変換します。

ヘッダ　　<stdlib.h>

リターン値　　正常：　多バイト文字列へ書き込まれたバイト数
異常：　(size_t)(-1)　　不正な多バイト文字の並びに遭遇した場合

引　数

s	多バイト文字列へのポインタ
pwcs	ワイド文字列へのポインタ
n	多バイト文字列へ書き込むバイト数

例

```
#include <stdlib.h>
const char *s;
wchar_t *pwcs;
size_t n, ret;
   ret=wcstombs(s,pwcs,n);
```

備　考　　wcstombs 関数は、pwcs が指す配列中のワイド文字の列を、初期シフト状態から始まる対応する多バイト文字の並びに変換し，s が指す配列に格納します。ただし，多バイト文字が合計で n バイトの上限を超えるとき，又はナル文字が格納されたとき，配列への格納を終了します。各ワイド文字は，wctomb 関数の変換状態が影響を受けないことを除いて，wctomb 関数の呼出しによる場合と同じ規則で変換します。
領域の重なり合うオブジェクト間でコピーが行われた場合の動作は未定義とします。
正常なリターン値であっても終端文字分のバイトは含みません。
リターン値が n のとき，配列はナル文字で終わっていません。

9.3.2 EC++クラスライブラリ

(1) ライブラリの概要

C++プログラムから標準的に利用できる EC++クラスライブラリの仕様について説明します。ここでは、クラスライブラリの種類と対応する標準インクルードファイルについて説明します。以降では、ライブラリの構成に従って各クラスライブラリの仕様について説明します。

- ライブラリの種類

  表9.38にクラスライブラリの種類と対応する標準インクルードファイルを示します。

表 9.38 クラスライブラリの種類と標準インクルードファイルの対応

	ライブラリの種類	内容	標準インクルードファイル
1	ストリーム入出力用クラスライブラリ	入出力操作を行うライブラリです。	<ios>,<streambuf>, <istream>,<ostream>, <iostream>,<iomanip>
2	メモリ操作用ライブラリ	メモリの確保・解放を行うライブラリです。	<new>
3	複素数計算用クラスライブラリ	複素数データ演算を行うライブラリです。	<complex>
4	文字列操作用クラスライブラリ	文字列操作を行うライブラリです。	<string>

(2) ストリーム入出力用クラスライブラリ

ストリーム入出力用クラスライブラリに対応するヘッダファイルは以下の通りです。

- <ios>

  入出力用書式設定、入出力状態管理を行うデータメンバおよび関数メンバを定義します。

  iosクラスの他に、Initクラス、ios_baseクラスを定義します。

- <streambuf>

  ストリームバッファに対する関数を定義します。

- <istream>

  入力ストリームからの入力関数を定義します。

- <ostream>

  出力ストリームへの出力関数を定義します。

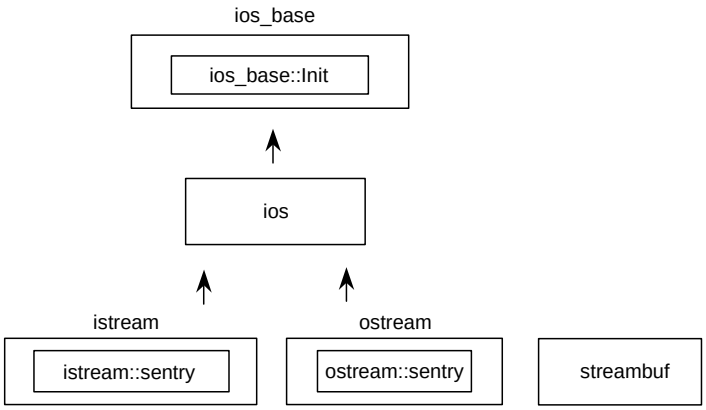
- <iostream>

  入出力関数を定義します。

- <iomanip>

  引数を持つマニピュレータを定義します。

これらのクラスの派生関係は次のようになります。矢印は、派生クラスから基底クラスを参照していることを示します。なお、streambuf クラスには派生関係はありません。



ストリーム入出力用クラスライブラリで共通に使用される型名を示します。

種別	定義名	説明
型	streamoff	long 型で定義された型です。
	streamsize	size_t 型で定義された型です。
	int_type	int 型で定義された型です。
	pos_type	long 型で定義された型です。
	off_type	long 型で定義された型です。

(a) ios_base::Initクラス

種別	定義名	説明
変数	init_cnt	ストリーム入出力オブジェクト数をカウントする静的データメンバです。 低水準インタフェースで 0 に初期化する必要があります。
関数	Init()	コンストラクタです
	~Init()	デストラクタです。

ios_base::Init::Init()

クラス Init のコンストラクタです。

init_cnt をインクリメントします。

ios_base::Init::~Init()

クラス Init のデストラクタです。

init_cnt をデクリメントします。

(b) ios_baseクラス

種別	定義名	説明
型	fmtflags	フォーマット制御情報を表す型です。
	iostate	ストリームバッファの入出力状態を表す型です。
	openmode	ファイルのオープンモードを表す型です。
	seekdir	ストリームバッファのシーク状態を表す型です。
変数	fmtfl	書式フラグです。
	wide	フィールド幅です。
	prec	出力時の精度(小数点以下の桁数)です。
	fillch	詰め文字です。
関数	void _ec2p_init_base()	初期化します。
	void _ec2p_copy_base(ios_base&ios_base_dt)	ios_base_dt をコピーします。
	ios_base()	コンストラクタです。
	~ios_base()	デストラクタです。
	fmtflags flags() const	書式フラグ(fmtfl)を参照します。
	fmtflags flags(fmtflags fmtflg)	fmtflg&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。
	fmtflags setf(fmtflags fmtflg)	fmtflg を書式フラグ(fmtfl)に設定します。
	fmtflags setf(fmtflags fmtflg, fmtflags mask)	mask&fmtflg を書式フラグ(fmtfl)に設定します。
	void unsetf(fmtflags mask)	~mask&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。
	char fill() const	詰め文字(fillch)を参照します。
	char fill(char ch)	ch を詰め文字(fillch)に設定します。
	int precision() const	精度(prec)を参照します。
	streamsize precision(streamsize preci)	preci を精度(prec)に設定します。
	streamsize width() const	フィールド幅(wide)を参照します。
	streamsize width(streamsize wd)	wd をフィールド幅(wide)に設定します。

ios_base::fmtflags

入出力に関するフォーマット制御情報を定義します。

fmtflags の各ビットマスクの定義は以下のようになります。

```
const ios_base::fmtflags ios_base::boolalpha    = 0x0000;
const ios_base::fmtflags ios_base::skipws       = 0x0001;
const ios_base::fmtflags ios_base::unitbuf      = 0x0002;
const ios_base::fmtflags ios_base::uppercase    = 0x0004;
const ios_base::fmtflags ios_base::showbase     = 0x0008;
const ios_base::fmtflags ios_base::showpoint    = 0x0010;
const ios_base::fmtflags ios_base::showpos      = 0x0020;
const ios_base::fmtflags ios_base::left         = 0x0040;
const ios_base::fmtflags ios_base::right        = 0x0080;
const ios_base::fmtflags ios_base::internal     = 0x0100;
const ios_base::fmtflags ios_base::adjustfield  = 0x01c0;
const ios_base::fmtflags ios_base::dec          = 0x0200;
const ios_base::fmtflags ios_base::oct          = 0x0400;
const ios_base::fmtflags ios_base::hex          = 0x0800;
const ios_base::fmtflags ios_base::basefield    = 0x0e00;
const ios_base::fmtflags ios_base::scientific   = 0x1000;
const ios_base::fmtflags ios_base::fixed        = 0x2000;
const ios_base::fmtflags ios_base::floatfield   = 0x3000;
const ios_base::fmtflags ios_base::_fmtmask     = 0x3fff;
```

ios_base::iostate

ストリームバッファの入出力状態を定義します。

iostate の各ビットマスクの定義は以下のようになります。

```
const ios_base::iostate ios_base::goodbit    = 0x0;
const ios_base::iostate ios_base::eofbit     = 0x1;
const ios_base::iostate ios_base::failbit    = 0x2;
const ios_base::iostate ios_base::badbit     = 0x4;
const ios_base::iostate ios_base::_statemask = 0x7;
```

ios_base::openmode

ファイルのオープンモードを定義します。

openmode の各ビットマスクの定義は以下のようになります。

```
const ios_base::openmode ios_base::in   = 0x01;  入力用のファイルを open します。
const ios_base::openmode ios_base::out  = 0x02;  出力用のファイルを open します。
const ios_base::openmode ios_base::ate  = 0x04;  オープン後一度だけ eof に seek します。
const ios_base::openmode ios_base::app  = 0x08;  書き込む度に eof に seek します。
```

const ios_base::openmode ios_base::trunc = 0x10; ファイルを上書きモードで open します。

const ios_base::openmode ios_base::binary = 0x20; ファイルをバイナリモードで open します。

ios_base::seekdir

ストリームバッファのシーク状態を定義します。

引き続き入力または出力を行うためのストリーム内の位置を決定します。

seekdir の各ビットマスクの定義は以下のようになります。

```
const ios_base::seekdir ios_base::beg = 0x0;
const ios_base::seekdir ios_base::cur = 0x1;
const ios_base::seekdir ios_base::end = 0x2;
```

void ios_base::_ec2p_init_base()

以下の値で初期設定します。

```
fmtfl   = skipws | dec;
wide    = 0;
prec    = 6;
fillch  = ' ';
```

void ios_base::_ec2p_copy_base(ios_base& ios_base_dt)

ios_base_dt をコピーします。

ios_base::ios_base()

クラス ios_base のコンストラクタです。

Init::Init()を呼び出します。

ios_base::~ios_base()

クラス ios_base のデストラクタです。

ios_base::fmtflags ios_base::flags() const

書式フラグ(fmtfl)を参照します。

リターン値は、書式フラグ(fmtfl)です。

ios_base::fmtflags ios_base::flags(fmtflags fmtflg)

fmtflg&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。

リターン値は、設定前の書式フラグ(fmtfl)です。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg)

fmtflg を書式フラグ(fmtfl)に設定します。

リターン値は、設定前の書式フラグ(fmtfl)です。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg, fmtflags mask)

mask&fmtflg の値を書式フラグ(fmtfl)に設定します。

リターン値は、設定前の書式フラグ(fmtfl)です。

void ios_base::unsetf(fmtflags mask)

~mask&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。

char ios_base::fill() const

詰め文字(fillch)を参照します。

リターン値は、詰め文字(fillch)です。

char ios_base::fill(char ch)

ch を詰め文字として設定します。

リターン値は、設定前の詰め文字(fillch)です。

int ios_base::precision() const

精度(prec)を参照します。

リターン値は、精度(prec)です。

streamsize ios_base::precision(streamsize preci)

preci を精度(prec)に設定します。

リターン値は、設定前の精度(prec)です。

streamsize ios_base::width() const

フィールド幅(wide)を参照します。

リターン値は、フィールド幅(wide)です。

streamsize ios_base::width(streamsize wd)

wd をフィールド幅(wide)に設定します。

リターン値は、設定前のフィールド幅(wide)です。

(c) iosクラス

種別	定義名	説明
変数	sb	streambuf オブジェクトへのポインタです。
	tiestr	ostream オブジェクトへのポインタです。
	state	streambuf への状態フラグです。
関数	ios()	コンストラクタです。
	ios(streambuf* sbptr)	
	void init(streambuf* sbptr)	初期設定を行います。
	virtual ~ios()	デストラクタです。
	operator void*() const	エラー有無(!state&(badbit | failbit))を判定します。
	bool operator!() const	エラー有無(state&(badbit | failbit))を判定します。
	iostate rdstate() const	状態フラグ(state)を参照します。
	void clear(iostate st = goodbit)	指定された状態(st)を除いて状態フラグ(state)をクリアします。
	void setstate(iostate st)	st を状態フラグ(state)に設定します。
	bool good() const	エラー有無(state==goodbit)を判定します。
	bool eof() const	入力ストリームの最後かどうか(state&eofbit)を判定します。
	bool bad() const	エラー有無(state&badbit)を判定します。
	bool fail() const	入力テキストが要求パターンと不一致であるかどうか (state&(badbit | failbit))判定します。
	ostream* tie() const	ostream オブジェクトへのポインタ(tiestr)を参照します。
	ostream* tie(ostream* tstrptr)	tstrptr を ostream オブジェクトへのポインタ(tiestr)に設定します。
	streambuf* rdbuf() const	streambuf オブジェクトへのポインタ(sb)を参照します。
	streambuf* rdbuf(streambuf* sbptr)	sbptr を streambuf オブジェクトへのポインタ(sb)に設定します。
	ios& copyfmt(const ios& rhs)	rhs の状態フラグ(state)をコピーします。

ios::ios()

クラス ios のコンストラクタです。

init(0)を呼び出し、初期値をそのメンバオブジェクトに設定します。

ios::ios(streambuf* sbptr)

クラス ios のコンストラクタです。

init(sbptr)を呼び出し、初期値をそのメンバオブジェクトに設定します。

void ios::init(streambuf* sbptr)

sbptr を sb に設定します。

state、tiestr を 0 に設定します。

virtual ios::~ios()

クラス ios のデストラクタです。

ios::operator void*() const

エラー有無(!state&(badbit | failbit))を判定します。

リターン値は次のとおりです。

エラー有の場合：false

エラー無の場合：true

bool ios::operator!() const

エラー有無(state&(badbit | failbit))を判定します。

リターン値は次のとおりです。

エラー有の場合：true

エラー無の場合：false

iostate ios::rdstate() const

状態フラグ(state) を参照します。

リターン値は、状態フラグ(state)です。

void ios::clear(iostate st = goodbit)

指定された状態(st)を除いて状態フラグ(state)をクリアします。

streambuf オブジェクトへのポインタ(sb)が 0 のときは、状態フラグ(state)に badbit を設定します。

void ios::setstate(iostate st)

st を状態フラグ(state)に設定します。

bool ios::good() const

エラー有無(state==goodbit)を判定します。

リターン値は次のとおりです。

エラー有の場合：false

エラー無の場合：true

bool ios::eof() const

入力ストリームの最後かどうか(state&eofbit)を判定します。

リターン値は次のとおりです。

入力ストリームの最後の場合　　：true

入力ストリームの最後以外の場合：false

bool ios::bad() const

エラー有無(state&badbit)を判定します。

リターン値は次のとおりです。

エラー有の場合：true

エラー無の場合：false

bool ios::fail() const

入力テキストが要求パターンと不一致であるかどうか(state&(badbit | failbit))を判定します。

リターン値は次のとおりです。

不一致の場合：true

一致の場合　：false

ostream* ios::tie() const

ostream オブジェクトへのポインタ(tiestr)を参照します。

リターン値は、ostream オブジェクトへのポインタ(tiestr)です。

ostream* ios::tie(ostream* tstrptr)

tstrptr を ostream オブジェクトへのポインタ(tiestr)に設定します。

リターン値は、設定前の ostream オブジェクトへのポインタ(tiestr)です。

streambuf* ios::rdbuf() const

streambuf オブジェクトへのポインタ(sb)を参照します。

リターン値は、streambuf オブジェクトへのポインタ(sb)です。

streambuf* ios::rdbuf(streambuf* sbptr)

sbptr を streambuf オブジェクトへのポインタ(sb)に設定します。

リターン値は、設定前の streambuf オブジェクトへのポインタ(sb)です。

ios& ios::copyfmt(const ios& rhs)

rhs の状態フラグ(state)をコピーします。

リターン値は*this です。

(d) iosクラスマニピュレータ

種別	定義名	説明
関数	ios_base& showbase(ios_base& str)	基数表示接頭辞モードに設定します。
	ios_base& noshowbase(ios_base& str)	基数表示接頭辞モードをクリアします。
	ios_base& showpoint (ios_base& str)	小数点生成モードに設定します。
	ios_base& noshowpoint (ios_base& str)	小数点生成モードをクリアします。
	ios_base& showpos(ios_base& str)	+記号生成モードに設定します。
	ios_base& noshowpos(ios_base& str)	+記号生成モードをクリアします。
	ios_base& skipws(ios_base& str)	空白読み飛ばしモードに設定します。
	ios_base& noskipws(ios_base& str)	空白読み飛ばしモードをクリアします。
	ios_base& uppercase(ios_base& str)	大文字変換モードに設定します。
	ios_base& nouppercase(ios_base& str)	大文字変換モードをクリアします。
	ios_base& internal(ios_base& str)	内部補充モードに設定します。
	ios_base& left(ios_base& str)	左側補充モードに設定します。
	ios_base& right(ios_base& str)	右側補充モードに設定します。
	ios_base& dec(ios_base& str)	10 進モードに設定します。
	ios_base& hex(ios_base& str)	16 進モードに設定します。
	ios_base& oct(ios_base& str)	8 進モードに設定します。
	ios_base& fixed(ios_base& str)	固定小数点モードに設定します。
	ios_base& scientific(ios_base& str)	科学表記法モードに設定します。

ios_base& showbase(ios_base& str)

データのはじめに基数を表示させるモードに設定します。

16 進数のときは、0x を行の先頭に付加します。10 進数のときは、そのまま出力します。

8 進数のときは、0 を行の先頭に付加します。

リターン値は str です。

ios_base& noshowbase(ios_base& str)

データのはじめに基数を表示させるモードをクリアします。

リターン値は str です。

ios_base& showpoint(ios_base& str)

小数点を出力するモードに設定します。

精度の指定がない場合、小数点以下 6 桁で表示します。

リターン値は str です。

ios_base& noshowpoint(ios_base& str)

小数点を出力するモードをクリアします。

リターン値は str です。

ios_base& showpos(ios_base& str)

+記号生成出力モード(正の数に対して+の符号を付加)に設定します。

リターン値は str です。

ios_base& noshowpos(ios_base& str)

+記号生成出力モードをクリアします。

リターン値は str です。

ios_base& skipws(ios_base& str)

空白読み飛ばし入力モード(連続する空白をスキップ)に設定します。

リターン値は str です。

ios_base& noskipws(ios_base& str)

空白読み飛ばし入力モードをクリアします。

リターン値は str です。

ios_base& uppercase(ios_base& str)

大文字変換出力モードに設定します。

16 進の基数表現が大文字の 0X になり、数値自体も大文字になります。

浮動小数点の指数表現も大文字の E になります。

リターン値は str です。

ios_base& nouppercase(ios_base& str)

大文字変換出力モードをクリアします。

リターン値は str です。

ios_base& internal(ios_base& str)

フィールド幅(wide)の範囲で出力時に

符号、基数

詰め文字(fill)

数値

の順で出力します。

リターン値は str です。

ios_base& left(ios_base& str)

フィールド幅(wide)の範囲で出力時に左詰めします。

リターン値は str です。

ios_base& right(ios_base& str)

フィールド幅(wide)の範囲で出力時に右詰めします。

リターン値は str です。

ios_base& dec(ios_base& str)

変換基数を 10 進モードに設定します。

リターン値は str です。

ios_base& hex(ios_base& str)

変換基数を 16 進モードに設定します。

リターン値は str です。

ios_base& oct(ios_base& str)

変換基数を 8 進モードに設定します。

リターン値は str です。

ios_base& fixed(ios_base& str)

固定小数点出力モードに設定します。

リターン値は str です。

ios_base& scientific(ios_base& str)

科学表記法出力モード(指数表記)に設定します。

リターン値は str です。

(e) streambufクラス

種別	定義名	説明
定数	eof	ファイル終了を示します。
変数	_B_cnt_ptr	バッファの有効データ長へのポインタです。
	B_beg_ptr	バッファのベースポインタへのポインタです。
	_B_len_ptr	バッファの長さへのポインタです。
	B_next_ptr	バッファの次の読み出し位置へのポインタです。
	B_end_ptr	バッファの終端位置へのポインタです。
	B_beg_pptr	制御バッファの先頭位置へのポインタです。
	B_next_pptr	バッファの次の読み出し位置へのポインタです。
	C_flg_ptr	ファイルの入出力制御フラグへのポインタです。
関数	char* _ec2p_getflag() const	ファイル入出力制御フラグのポインタを参照します。
	char*& _ec2p_gnptr()	バッファの次の読み出し位置へのポインタを参照します。
	char*& _ec2p_pnptr()	バッファの次の書き込み位置へのポインタを参照します。
	void _ec2p_bcntplus()	バッファの有効データ長をインクリメントします。
	void _ec2p_bcntminus()	バッファの有効データ長をデクリメントします。
	void _ec2p_setbPtr(char** begptr, char** curptr, long* cntptr, long* lenptr, char* flgptr)	streambuf のポインタを設定します。
	streambuf()	コンストラクタです。
	virtual ~streambuf()	デストラクタです。
	streambuf* pubsetbuf(char* s, streamsize n)	ストリーム入出力用のバッファを確保します。この関数では setbuf(s,n)[*1] を呼び出します。
	pos_type pubseekoff(off_type off, ios _base::seekdir way, ios_base::openmode which = ios_base::in | ios_base::out)	way で指定された方法で入出力ストリームの読み書き位置を移動させます。この関数では seekoff(off,way,which)[*1] を呼び出します。
	pos_type pubseekpos(pos_type sp, ios_base::openmode which = ios_base::in | ios_base::out)	ストリームの先頭から現在の位置までのオフセットを求めます。この関数では seekpos(sp,which)[*1] を呼び出します。
	int pubsync()	出力ストリームをフラッシュします。この関数では sync()[*1] を呼び出します。
	streamsize in_avail()	入力ストリームの最後尾から現在位置までのオフセットを求めます。

種別	定義名	説明
関数	int_type snextc()	次の一文字を読み込みます。
	int_type sbumpc()	一文字読み込みポインタを次に設定します。
	int_type sgetc()	一文字読み込みます。
	int sgetn(char* s, streamsize n)	s の指す記憶領域に n 個の文字を設定します。
	int_type sputbackc(char c)	読み込み位置をプットバックします。
	int sungetc()	読み込み位置をプットバックします。
	int sputc(char c)	文字 c を挿入します。
	int_type sputn(const char* s, streamsize n)	s の指す n 個の文字を挿入します。
	char* eback() const	入力ストリームの先頭ポインタを求めます。
	char* gptr() const	入力ストリームの次ポインタを求めます。
	char* egptr() const	入力ストリームの最後尾ポインタを求めます。
	void gbump(int n)	入力ストリームの次ポインタを n 進めます。
	void setg(char* gbeg, char* gnext, char* gend)	入力ストリームの各ポインタを代入します。
	char* pbase() const	出力ストリームの先頭ポインタを求めます。
	char* pptr() const	出力ストリームの次ポインタを求めます。
	char* epptr() const	出力ストリームの最後尾ポインタを求めます。
	void pbump(int n)	出力ストリームの次ポインタを n 進めます。
	void setp(char* pbeg, char* pend)	出力ストリームの各ポインタを設定します。
	virtual streambuf* setbuf(char* s, streamsize n)[*1]	派生する各クラスごとに、個別に定義する演算を実行します。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))[*1]	ストリーム位置を変更します。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out)) [*1]	ストリーム位置を変更します。
	virtual int sync()[*1]	出力ストリームをフラッシュします。
	virtual int showmanyc()[*1]	入力ストリームの有効な文字数を求めます。
	virtual streamsize xsgetn(char* s, streamsize n)	s の指す記憶領域に n 個の文字を設定します。
	virtual int_type underflow()[*1]	ストリーム位置を動かさずに一文字読み込みます。
	virtual int_type uflow()[*1]	次ポインタの一文字を読み込みます。
	virtual int_type pbackfail(int_type c = eof)[*1]	c によって示される文字をプットバックします。

種別	定義名	説明
関数	virtual streamsize xsputn(const char* s, streamsize n)	s の指す n 個の文字を挿入します。
	virtual int_type overflow(int_type c = eof)*1	c を出力ストリームに挿入します。

【注】 *1 このクラスでは処理を定義していません。

streambuf::streambuf()

コンストラクタです。

以下の値で初期化します。

_B_cnt_ptr = B_beg_ptr = B_next_ptr = B_end_ptr = C_flg_ptr = _B_len_ptr = 0

B_beg_pptr = &B_beg_ptr

B_next_pptr = &B_next_ptr

virtual streambuf::~streambuf()

デストラクタです。

streambuf* streambuf::pubsetbuf(char* s, streamsize n)

ストリーム入出力用のバッファを確保します。

この関数では setbuf(s,n)を呼び出します。

リターン値は、*this です。

pos_type streambuf::pubseekoff(off_type off, ios _base::seekdir way, ios_base::openmode which = (ios_base::openmode)(ios_base::in | ios_base::out))

way で指定された方法で入出力ストリームの読み書き位置を移動させます。

この関数では seekoff(off,way,which)を呼び出します。

リターン値は、新たに設定されたストリームの位置です。

pos_type streambuf::pubseekpos(pos_type sp, ios_base::openmode which = (ios_base::openmode)(ios_base::in | ios_base::out))

ストリームの先頭から現在の位置までのオフセットを求めます。

現在のストリームポインタから sp だけ移動します。

この関数では seekpos(sp,which)を呼び出します。

リターン値は、先頭からのオフセットです。

int streambuf::pubsync()

出力ストリームをフラッシュします。

この関数では sync()を呼び出します。

リターン値は 0 です。

streamsize streambuf::in_avail()

入力ストリームの最後尾から現在位置までのオフセットを求めます。

リターン値は次のとおりです。

読み込み位置が有効の場合 ：最後尾から現在位置までのオフセット
読み込み位置が無効の場合 ：0(showmanyc()を呼び出します)

int_type streambuf::snextc()

一文字読み込みます。読み込んだ文字が eof でなければ、次の一文字を読み込みます。
リターン値は次のとおりです。

eof でない場合 ：読み込んだ文字
eof の場合 ：eof

int_type streambuf::sbumpc()

一文字読み込みポインタを次に設定します。
リターン値は次のとおりです。

読み込み位置が無効でない場合 ：読み込んだ文字
読み込み位置が無効の場合 ：eof

int_type streambuf::sgetc()

一文字読み込みます。
リターン値は次のとおりです。

読み込み位置が無効でない場合 ：読み込んだ文字
読み込み位置が無効の場合 ：eof

int streambuf::sgetn(char* s, streamsize n)

s の指す記憶領域に n 個の文字を設定します。
文字列中に eof を検出した場合、設定を終了します。
リターン値は、設定した文字数です。

int_type streambuf::sputbackc(char c)

読み込み位置が正常で読み込み位置のプットバックデータが c と同一の場合、読み込み位置をプットバックします。
リターン値は次のとおりです。

プットバックできた場合 ：c の値
プットバックできなかった場合 ：eof

int streambuf::sungetc()

読み込み位置が正常である場合、読み込み位置をプットバックします。
リターン値は次のとおりです。

プットバックできた場合 ：プットバックした値
プットバックできなかった場合 ：eof

int streambuf::sputc(char c)

文字 c を挿入します。
リターン値は次のとおりです。

書き込み位置が正しい場合 ：c の値
書き込み位置が不正な場合 ：eof

int_type streambuf::sputn(const char* s, streamsize n)

s の指す n 個の文字を挿入します。
バッファが n より小さい場合は、バッファサイズ分だけ挿入します。
リターン値は、挿入された文字数です。

char* streambuf::eback() const

入力ストリームの先頭ポインタを求めます。
リターン値は、先頭ポインタです。

char* streambuf::gptr() const

入力ストリームの次ポインタを求めます。
リターン値は、次ポインタです。

char* streambuf::egptr() const

入力ストリームの最後尾ポインタを求めます。
リターン値は、最後尾ポインタです。

void streambuf::gbump(int n)

入力ストリームの次ポインタを n 進めます。

void streambuf::setg(char* gbeg, char* gnext, char* gend)

入力ストリームの各ポインタに、以下の設定を行います。

```
*B_beg_pptr    = gbeg;
*B_next_pptr   = gnext;
B_end_ptr      = gend;
*_B_cnt_ptr    = gend-gnext;
*_B_len_ptr    = gend-gbeg;
```

char* streambuf::pbase() const

出力ストリームの先頭ポインタを求めます。
リターン値は、先頭ポインタです。

char* streambuf::pptr() const

出力ストリームの次ポインタを求めます。
リターン値は、次ポインタです。

char* streambuf::epptr() const

出力ストリームの最後尾ポインタを求めます。
リターン値は、最後尾ポインタです。

void streambuf::pbump(int n)

出力ストリームの次ポインタを n 進めます。

void streambuf::setp(char* pbeg, char* pend)

出力ストリームの各ポインタに、以下の設定を行います。

*B_beg_pptr = pbeg;

*B_next_pptr = pbeg;

B_end_ptr = pend;

*_B_cnt_ptr = pend-pbeg;

*_B_len_ptr = pend-pbeg;

virtual streambuf* streambuf::setbuf(char* s, streamsize n)

streambuf から派生する各クラスごとに、個別に定義する演算を実行します。

リターン値は*this です。このクラスでは処理を定義していません。

virtual pos_type streambuf::seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode)(ios_base::in | ios_base::out))

ストリーム位置を変更します。

リターン値は-1 です。このクラスでは処理を定義していません。

virtual pos_type streambuf::seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode)(ios_base::in | ios_base::out))

ストリーム位置を変更します。

リターン値は-1 です。このクラスでは処理を定義していません。

virtual int streambuf::sync()

出力ストリームをフラッシュします。

リターン値は 0 です。このクラスでは処理を定義していません。

virtual int streambuf::showmanyc()

入力ストリームの有効な文字数を求めます。

リターン値は 0 です。このクラスでは処理を定義していません。

virtual streamsize streambuf::xsgetn(char* s, streamsize n)

s の指す記憶領域に n 個の文字を設定します。

バッファが n より小さい場合は、バッファサイズ分だけ設定します。

リターン値は、入力された文字数です。

virtual int_type streambuf::underflow()

ストリーム位置を動かさずに一文字読み込みます。

リターン値は eof です。このクラスでは処理を定義していません。

virtual int_type streambuf::uflow()

次ポインタの一文字を読み込みます。

リターン値は eof です。このクラスでは処理を定義していません。

virtual int_type streambuf::pbackfail(int_type c = eof)

c によって示される文字をプットバックします。

リターン値は eof です。このクラスでは処理を定義していません。

virtual streamsize streambuf::xsputn(const char* s, streamsize n)

s の指す n 個の文字を挿入します。

バッファが n より小さい場合は、バッファサイズ分だけ挿入します。

リターン値は、挿入された文字数です。

virtual int_type streambuf::overflow(int_type c = eof)

c を出力ストリームに挿入します。

リターン値は eof です。このクラスでは処理を定義していません。

(f) istream::sentryクラス

種別	定義名	説明
変数	ok_	入力可能状態か否かを意味します。
関数	sentry(istream& is, bool noskipws = false)	コンストラクタです。
	~sentry()	デストラクタです。
	operator bool()	ok_を参照します。

istream::sentry::sentry(istream& is, bool noskipws = _false)

内部クラス sentry のコンストラクタです。

good()が非 0 の場合、フォーマット付きまたはフォーマットなし入力を可能にします。

tie()が非 0 の場合、関連する出力ストリームをフラッシュします。

istream::sentry::~sentry()

内部クラス sentry のデストラクタです。

istream::sentry::operator bool()

ok_を参照します。

リターン値は ok_です。

(g) istreamクラス

種別	定義名	説明
変数	chcount	最後にコールされた入力関数が抽出した文字数です。
関数	int _ec2p_getistr(char* str, unsigned int dig, int mode)	str を dig が示す基数で変換します。
	istream(streambuf* sb)	コンストラクタです。
	virtual ~istream()	デストラクタです。
	istream& operator>>(bool& n)	抽出した文字を n に格納します。
	istream& operator>>(short& n)	
	istream& operator>>(unsigned short& n)	
	istream& operator>>(int& n)	
	istream& operator>>(unsigned int& n)	
	istream& operator>>(long& n)	
	istream& operator>>(unsigned long& n)	
	istream& operator>>(long long& n)	
	istream& operator>>(unsigned long long& n)	
	istream& operator>>(float& n)	
	istream& operator>>(double& n)	
	istream& operator>>(long double& n)	
	istream& operator>>(void*& p)	void を指すポインタに変換して p に格納します。
	istream& operator>>(streambuf* sb)	文字を抽出し、sb の指す記憶領域へ格納します。
	streamsize gcount() const	chcount(抽出文字数)を求めます。
	int_type get()	文字を抽出します。
	istream& get(char& c)	文字を抽出し c に格納します。
	istream& get(signed char& c)	
	istream& get(unsigned char& c)	
	istream& get(char* s, streamsize n)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
	istream& get(signed char* s, streamsize n)	
	istream& get(unsigned char* s, streamsize n)	
	istream& get(char* s, streamsize n, char delim)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。文字列内に'delim'を検出したら、入力を終了します。
	istream& get(signed char* s, streamsize n, char delim)	
	istream& get(unsigned char* s, streamsize n, char delim)	

種別	定義名	説明
関数	istream& get(streambuf& sb)	文字列を抽出し、sb の指す記憶領域に格納します。
	istream& get(streambuf& sb, char delim)	文字列を抽出し、sb の指す記憶領域に格納します。途中で文字'delim'を検出したら、入力を終了します。
	istream& getline(char* s, streamsize n)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
	istream& getline(signed char* s, streamsize n)	
	istream& getline(unsigned char* s, streamsize n)	
	istream& getline(char* s, streamsize n, char delim)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。途中で文字'delim'を検出したら、入力を終了します。
	istream& getline(signed char* s, streamsize n, char delim)	
	istream& getline(unsigned char* s, streamsize n, char delim)	
	istream& ignore(streamsize n = 1, int_type delim = streambuf::eof)	n 個の文字を読み飛ばします。途中で文字'delim'を検出したら、読み飛ばし処理を中止します。
	int_type peek()	次の入手可能な入力文字を求めます。
	istream& read(char* s, streamsize n)	サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
	istream& read(signed char* s, streamsize n)	
	istream& read(unsigned char* s, streamsize n)	
	streamsize readsome(char* s, streamsize n)	サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
	streamsize readsome(signed char* s, streamsize n)	
	streamsize readsome(unsigned char* s, streamsize n)	
	istream& putback(char c)	文字を入力ストリームに戻します。
	istream& unget()	入力ストリームの位置を戻します。
	int sync()	入力ストリームがあるかどうかを調べます。この関数は streambuf::pubsync()を呼び出します。
	pos_type tellg()	入力ストリームの位置を調べます。この関数は streambuf::pubseekoff(0,cur,in)を呼び出します。

種別	定義名	説明
関数	istream& seekg(pos_type pos)	現在のストリームポインタから pos だけ移動します。この関数は streambuf::pubseekpos(pos)を呼び出します。
	istream& seekg(off_type off, ios_base::seekdir dir)	dir で指定された方法で入力ストリームの読み込み位置を移動します。この関数は streambuf::pubseekoff(off,dir)を呼び出します。

int istream::_ec2p_getistr(char* str, unsigned int dig, int mode)

str を dig が示す基数で変換します。

リターン値は、変換した基数です。

istream::istream(streambuf* sb)

クラス istream のコンストラクタです。

ios::init(sb)を呼び出します。

chcount=0 の設定を行います。

virtual istream::~istream()

クラス istream のデストラクタです。

istream& istream::operator>>(bool& n)
istream& istream::operator>>(short& n)
istream& istream::operator>>(unsigned short& n)
istream& istream::operator>>(int& n)
istream& istream::operator>>(unsigned int& n)
istream& istream::operator>>(long& n)
istream& istream::operator>>(unsigned long& n)
istream& istream::operator>>(long long& n)
istream& istream::operator>>(unsigned long long& n)
istream& istream::operator>>(float& n)
istream& istream::operator>>(double& n)
istream& istream::operator>>(long double& n)

抽出した文字を n に格納します。

リターン値は*this です。

istream& istream::operator>>(void*& p)

抽出した文字を void*型に変換し、p の指す記憶領域に格納します。

リターン値は*this です。

istream& istream::operator>>(streambuf* sb)

文字を抽出し、sb の指す記憶領域に格納します。

抽出文字がない場合は、setstate(failbit)を呼び出します。

リターン値は*this です。

streamsize istream::gcount() const

chcount(抽出文字数)を参照します。

リターン値は chcount です。

int_type istream::get()

文字を抽出します。

リターン値は次のとおりです。

抽出可能の場合 ：抽出した文字

抽出不可の場合 ：setstate(failbat)を呼び出して、streambuf::eof

istream& istream::get(char& c)
istream& istream::get(signed char& c)
istream& istream::get(unsigned char& c)

文字を抽出し c に格納します。抽出した文字が streambuf::eof の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::get(char* s, streamsize n)
istream& istream::get(signed char* s, streamsize n)
istream& istream::get(unsigned char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::get(char* s, streamsize n, char delim)
istream& istream::get(signed char* s, streamsize n, char delim)
istream& istream::get(unsigned char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。

文字列内に'delim'を検出したら、終了します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::get(streambuf& sb)

文字列を抽出し、sb の指す記憶領域に格納します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::get(streambuf& sb, char delim)

文字列を抽出し、sb の指す記憶領域に格納します。

途中で文字'delim'を検出したら、終了します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::getline(char* s, streamsize n)
istream& istream::getline(signed char* s, streamsize n)
istream& istream::getline(unsigned char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::getline(char* s, streamsize n, char delim)
istream& istream::getline(signed char* s, streamsize n, char delim)
istream& istream::getline(unsigned char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。

途中で文字'delim'を検出したら、終了します。

ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。

リターン値は*this です。

istream& istream::ignore(streamsize n = 1, int_type delim = streambuf::eof)

n 個の文字を読み飛ばします。

途中で文字'delim'を検出したら、読み飛ばし処理を中止します。

リターン値は*this です。

int_type istream::peek()

次の入力可能な入力文字を求めます。

リターン値は次のとおりです。

ok_==false の場合	：streambuf::eof
ok_!=false の場合	：rdbuf()->sgetc()

istream& istream::read(char* s, streamsize n)
istream& istream::read(signed char* s, streamsize n)
istream& istream::read(unsigned char* s, streamsize n)

ok_!=false の場合、サイズ n の文字列を抽出し、s の指す記憶領域に格納します。

抽出した文字数が n と異なる場合、eofbit を設定します。

リターン値は*this です。

streamsize istream::readsome(char* s, streamsize n)
streamsize istream::readsome(signed char* s, streamsize n)
streamsize istream::readsome(unsigned char* s, streamsize n)

サイズ n の文字列を抽出し、s の指す記憶領域に格納します。

文字数がストリームサイズより大きければ、ストリームサイズ分格納します。

リターン値は、抽出した文字数です。

istream& istream::putback(char c)

文字 c を入力ストリームに戻します。プットバックした文字が streambuf::eof の場合は、badbit を設定します。

リターン値は*this です。

istream& istream::unget()

入力ストリームのポインタをひとつ戻します。

抽出した文字が streambuf::eof の場合、badbit を設定します。

リターン値は*this です。

int istream::sync()

入力ストリームがあるかどうかを調べます。

この関数は streambuf::pubsync()を呼び出します。

リターン値は次のとおりです。

入力ストリームがない場合 ：streambuf::eof

入力ストリームがある場合 ：0

pos_type istream::tellg()

入力ストリームの位置を調べます。

この関数は streambuf::pubseekoff(0,cur,in)を呼び出します。

リターン値は次のとおりです。

ストリームの先頭からのオフセット

ただし、入力処理にエラーが発生した場合は-1

istream& istream::seekg(pos_type pos)

現在のストリームポインタから pos だけ移動します。

この関数は streambuf::pubseekpos(pos)を呼び出します。

リターン値は*this です。

istream& istream::seekg(off_type off, ios_base::seekdir dir)

dir で指定された方法で入力ストリームの読み込み位置を移動します。

この関数は streambuf::pubseekoff(off,dir)を呼び出します。

入力処理にエラーがある場合は処理は行いません。

リターン値は*this です。

(h) istreamクラスマニピュレータ

種別	定義名	説明
関数	istream& ws(istream& is)	空白類を読み飛ばします。

istream& ws(istream& is)

空白類を読み飛ばします。

リターン値は is です。

(i) istreamメンバ外関数

種別	定義名	説明
関数	istream& operator>>(istream& in, char* s)	文字列を抽出し、s の指す記憶領域に格納します。
	istream& operator>>(istream& in, signed char* s)	
	istream& operator>>(istream& in, unsigned char* s)	
	istream& operator>>(istream& in, char& c)	文字を抽出し、c に格納します。
	istream& operator>>(istream& in, singed char& c)	
	istream& operator>>(istream& in, unsigned char& c)	

istream& operator>>(istream& in, char* s)
istream& operator>>(istream& in, signed char* s)
istream& operator>>(istream& in, unsigned char* s)

文字列を抽出し、s の指す記憶領域に格納します。

(フィールド幅-1)個の文字を格納したか、または入力ストリームに streambuf::eof が現れたか、または次の入力可能な文字 c が isspace(c)==1 の場合、処理は終了します。格納文字数が 0 の場合は failbit を設定します。

リターン値は in です。

istream& operator>>(istream& in, char& c)
istream& operator>>(istream& in, singed char& c)
istream& operator>>(istream& in, unsigned char& c)

文字を抽出し、c に格納します。

抽出入力がない場合、failbit を設定します。

リターン値は in です。

(j) ostream::sentryクラス

種別	定義名	説明
変数	ok_	出力可能状態か否かを意味します。
	_ _ec2p_os	ostream オブジェクトへのポインタです。
関数	sentry(ostream& os)	コンストラクタです。
	~sentry()	デストラクタです。
	operator bool()	ok_を参照します。

ostream::sentry::sentry(ostream& os)

内部クラス sentry のコンストラクタです。

good()が非 0 かつ tie()が非 0 なら flush()を呼び出します。_ _ec2p_os に os を設定します。

ostream::sentry::~sentry()

内部クラス sentry のデストラクタです。

_ _ec2p_os->flags() & ios_base::unitbuf が真なら、flush()を呼び出します。

ostream::sentry::operator bool()

ok_を参照します。

リターン値は ok_です。

(k) ostreamクラス

種別	定義名	説明
関数	ostream(streambuf* sbptr)	コンストラクタです。
	virtual ~ostream()	デストラクタです。
	ostream& operator<<(bool n)	n を出力ストリームに挿入します。
	ostream& operator<<(short n)	
	ostream& operator<<(unsigned short n)	
	ostream& operator<<(int n)	
	ostream& operator<<(unsigned int n)	
	ostream& operator<<(long n)	
	ostream& operator<<(unsigned long n)	
	ostream& operator<<(long long n)	
	ostream& operator<<(unsigned long long n)	
	ostream& operator<<(float n)	
	ostream& operator<<(double n)	
	ostream& operator<<(long double n)	
	ostream& operator<<(void* n)	
	ostream& operator<<(streambuf* sbptr)	sbptr の出力列を出力ストリームに挿入します。
	ostream& put(char c)	文字 c を出力ストリームに挿入します。
	ostream& write(const char* s, streamsize n)	s の n 個の文字を出力ストリームに挿入します。
	ostream& write(const signed char* s, streamsize n)	
	ostream& write(const unsigned char* s, streamsize n)	
	ostream& flush()	出力ストリームをフラッシュします。この関数は streambuf::pubsync()を呼び出します。
	pos_type tellp()	現在の書き込み位置を求めます。この関数は streambuf::pubseekoff(0,cur,out)を呼び出します。
	ostream& seekp(pos_type pos)	ストリームの先頭から現在の位置までのオフセットを求めます。現在のストリームポインタから pos だけ移動します。この関数は streambuf::pubseekpos(pos)を呼び出します。
	ostream& seekp(off_type off, seekdir dir)	dir を基準として、ストリームの書き込み位置を off 分だけ移動します。この関数は streambuf::pubseekoff(off,dir)を呼び出します。

ostream::ostream(streambuf* sbptr)

コンストラクタです。

ios(sbptr)を呼び出します。

virtual ostream::~ostream()

デストラクタです。

ostream& ostream::operator<<(bool n)
ostream& ostream::operator<<(short n)
ostream& ostream::operator<<(unsigned short n)
ostream& ostream::operator<<(int n)
ostream& ostream::operator<<(unsigned int n)
ostream& ostream::operator<<(long n)
ostream& ostream::operator<<(unsigned long n)
ostream& ostream::operator<<(long long n)
ostream& ostream::operator<<(unsigned long long n)
ostream& ostream::operator<<(float n)
ostream& ostream::operator<<(double n)
ostream& ostream::operator<<(long double n)
ostream& ostream::operator<<(void* n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。

sentry::ok_==false のとき、failbit を設定します。

リターン値は*this です。

ostream& ostream::operator<<(streambuf* sbptr)

sentry::ok_==true のとき、sbptr の出力列を出力ストリームに挿入します。

sentry::ok_==false のとき、failbit を設定します。

リターン値は*this です。

ostream& ostream::put(char c)

sentry::ok_==true かつ rdbuf()->sputc(c)!=streambuf::eof のとき、c を出力ストリームに挿入します。

上記以外のとき、badbit を設定します。

リターン値は*this です。

ostream& ostream::write(const char* s, streamsize n)
ostream& ostream::write(const signed char* s, streamsize n)
ostream& ostream::write(const unsigned char* s, streamsize n)

sentry::ok_==true かつ rdbuf()->sputn(s, n)==n のとき、s の n 個の文字を出力ストリームに挿入します。

上記以外のとき、badbit を設定します。

リターン値は*this です。

ostream& ostream::flush()

出力ストリームをフラッシュします。

この関数は streambuf::pubsync()を呼び出します。

リターン値は*this です。

pos_type ostream::tellp()

現在の書き込み位置を求めます。

この関数は streambuf::pubseekoff(0,cur,out)を呼び出します。

リターン値は次のとおりです。

現在のストリームの位置

ただし、処理中にエラーが発生した場合は-1

ostream& ostream::seekp(pos_type pos)

エラーがないとき、ストリームの先頭から現在の位置までのオフセットを求めます。

また、現在のストリームポインタから pos だけ移動します。

この関数は streambuf::pubseekpos(pos)を呼び出します。

リターン値は*this です。

ostream& ostream::seekp(off_type off, seekdir dir)

エラーがないとき、dir を基準として off 分ストリームの位置を移動します。

この関数は streambuf::pubseekoff(off,dir)を呼び出します。

リターン値は*this です。

(l) ostreamクラスマニピュレータ

種別	定義名	説明
関数	ostream& endl(ostream& os)	改行を挿入し、出力ストリームをフラッシュします。
	ostream& ends(ostream& os)	ヌルコードを挿入します。
	ostream& flush(ostream& os)	出力ストリームをフラッシュします。

ostream& endl(ostream& os)

ストリームに改行文字を挿入します。

出力ストリームをフラッシュします。この関数は flush()を呼び出します。

リターン値は os です。

ostream& ends(ostream& os)

出力ストリームにヌルコードを挿入します。

リターン値は os です。

ostream& flush(ostream& os)

出力ストリームをフラッシュします。この関数は streambuf::sync()を呼び出します。

リターン値は os です。

（m）ostreamメンバ外関数

種別	定義名	説明
関数	ostream& operator<<(ostream& os, char s)	s を出力ストリームに挿入します。
	ostream& operator<<(ostream& os, signed char s)	
	ostream& operator<<(ostream& os, unsigned char s)	
	ostream& operator<<(ostream& os, const char* s)	
	ostream& operator<<(ostream& os, const singed char* s)	
	ostream& operator<<(ostream& os, const unsigned char* s)	

ostream& operator<<(ostream& os, char s)
ostream& operator<<(ostream& os, signed char s)
ostream& operator<<(ostream& os, unsigned char s)
ostream& operator<<(ostream& os, const char* s)
ostream& operator<<(ostream& os, const singed char* s)
ostream& operator<<(ostream& os, const unsigned char* s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。

上記以外のとき、failbit を設定します。

リターン値は os です。

(n) smanipクラスマニピュレータ

種別	定義名	説明
関数	smanip resetiosflags(ios_base::fmtflags mask)	mask 値で指定されたフラグをクリアします。
	smanip setiosflags(ios_base::fmtflags mask)	書式フラグ(fmtfl)を設定します。
	smanip setbase(int base)	出力時に用いる基数を設定します。
	smanip setfill(char c)	詰め文字(fillch)を設定します。
	smanip setprecision(int n)	精度(prec)を設定します。
	smanip setw(int n)	フィールド幅(wide)を設定します。

smanip resetiosflags(ios_base::fmtflags mask)

mask 値で指定されたフラグをクリアします。

リターン値は、入出力対象のオブジェクトです。

smanip setiosflags(ios_base::fmtflags mask)

書式フラグ(fmtfl)を設定します。

リターン値は、入出力対象のオブジェクトです。

smanip setbase(int base)

出力時に用いる基数を設定します。

リターン値は、入出力対象のオブジェクトです。

smanip setfill(char c)

詰め文字(fillch)を設定します。

リターン値は、入出力対象のオブジェクトです。

smanip setprecision(int n)

精度(prec)を設定します。

リターン値は、入出力対象のオブジェクトです。

smanip setw(int n)

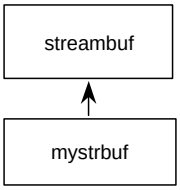
フィールド幅(wide)を設定します。

リターン値は、入出力対象のオブジェクトです。

(o) EC++入出力ライブラリの使用例

istream, ostream のオブジェクトの初期化時に streambuf のかわりに mystrbuf クラスのオブジェクトへのポインタを使うことにより入出力ストリームが使用可能になります。

クラスの派生関係は次のようになります。矢印は、派生クラスから基底クラスを参照していることを示します。



種別	定義名	説明
変数	_file_Ptr	ファイルポインタです。
関数	mystrbuf()	コンストラクタです。streambuf バッファの初期化を行います。
	mystrbuf(void* ptr)	
	virtual ~mystrbuf()	デストラクタです。
	void* myfptr() const	FILE 型構造体へのポインタを返します。
	mystrbuf* open(const char* filename, int mode)	ファイル名とモードを指定して、ファイルをオープンします。
	mystrbuf* close()	ファイルのクローズを行います。
	virtual streambuf* setbuf(char* s, streamsize n)	ストリーム入出力用のバッファを確保します。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	ストリームポインタの位置を変えます。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	ストリームポインタの位置を変えます。
	virtual int sync()	ストリームをフラッシュします。
	virtual int showmanyc()	入力ストリームの有効な文字数を返します。
	virtual int_type underflow()	ストリーム位置を動かさずに一文字読み込みます。
	virtual int_type pbackfail(int_type c = streambuf::eof)	c によって示される文字をプットバックします。

種別	定義名	説明
関数	virtual int_type overflow(int_type c = streambuf::eof)	c によって示される文字を挿入します。
	void _Init(_f_type* fp)	初期処理です。

例：

```
#include <istream>
#include <ostream>
#include <mystrbuf>
#include <string>
#include <new>
#include <stdio.h>
void main(void)
{
    mystrbuf myfin(stdin);
    mystrbuf myfout(stdout);
    istream mycin(&myfin);
    ostream mycout(&myfout);

    int i;
    short s;
    long l;
    char c;
    string str;

    mycin >> i >> s >> l >> c >> str;
    mycout << "This is EC++ Library." << endl
           << i << s << l << c << str << endl;
    return;
}
```

(3) メモリ管理用ライブラリ

メモリの管理用ライブラリに対応するヘッダファイルは以下の通りです。

- <new>

  メモリの確保・解放を行う関数を定義します。

_ec2p_new_handler 変数に例外処理関数のアドレスを設定することにより、メモリ確保に失敗した場合、例外処理を実行することができます。

_ec2p_new_handler は static 変数で、初期値は NULL です。このハンドラを使用することにより、リエントラント性は失われます。

例外処理関数に要求される動作：

- 割り当て可能な領域を作成して返します。
- 作成できない場合の動作は規定されていません。

種別	定義名	説明
型	new_handler	void 型を返す関数へのポインタ型です。
変数	_ec2p_new_handler	例外処理関数へのポインタです。
関数	void* operator new(size_t size)	size 分の領域を確保します。
	void* operator new[ ](size_t size)	size 分の配列領域を確保します。
	void* operator new(size_t size, void* ptr)	ptr の指している領域を記憶領域として割り当てます。
	void* operator new[ ](size_t size, void* ptr)	ptr の指している領域を配列領域として割り当てます。
	void operator delete(void* ptr)	領域を解放します。
	void operator delete[ ](void* ptr)	配列領域を解放します。
	new_handler set_new_handler(new_handler new_P)	_ec2p_new_handler に例外処理関数アドレス(new_P)を設定します。

void* operator new(size_t size)

size バイト分の領域を割り当てます。

領域割り当てに失敗し、かつ new_handler が設定されていれば、new_handler を呼び出します。

リターン値は次のとおりです。

領域確保に成功した場合　：void 型へのポインタ

領域確保に失敗した場合　：NULL

void* operator new[ ](size_t size)

size 分の配列領域を確保します。

領域割り当てに失敗し、かつ new_handler が設定されていれば、new_handler を呼び出します。

リターン値は次のとおりです。

領域確保に成功した場合　：void 型へのポインタ

領域確保に失敗した場合 ：NULL

void* operator new(size_t size, void* ptr)

ptr の指している領域を記憶領域として割り当てます。

リターン値は ptr です。

void* operator new[ ](size_t size, void* ptr)

ptr の指している領域を配列領域として割り当てます。

リターン値は ptr です。

void operator delete(void* ptr)

ptr が指す記憶領域を解放します。ptr が NULL のときは何もしません。

void operator delete[ ](void* ptr)

ptr が指す配列領域を解放します。ptr が NULL のときは何もしません。

new_handler set_new_handler(new_handler new_P)

_ec2p_new_handler に new_P を設定します。

リターン値は_ec2p_new_handler です。

(4) 複素数計算用クラスライブラリ

複素数計算用クラスライブラリに対応するヘッダファイルは以下のとおりです。

- <complex>

  float_complexクラス、double_complexクラスを定義します。

これらのクラスには派生関係はありません。

(a) float_complexクラス

種別	定義名	説明
型	value_type	float 型です。
変数	_re	float 精度の実数部を定義します。
	_im	float 精度の虚数部を定義します。
関数	float_complex(float re = 0.0f, float im = 0.0f)	コンストラクタです。
	float_complex(const double_complex& rhs)	
	float real() const	実数部(_re)を求めます。
	float imag() const	虚数部(_im)を求めます。
	float_complex& operator=(float rhs)	rhs を実数部にコピーします。虚数部は 0.0f を設定します。
	float_complex& operator+=(float rhs)	rhs を実数部に加算し、和を*this に格納します。
	float_complex& operator-=(float rhs)	rhs を実数部から減算し、差を*this に格納します。
	float_complex& operator*=(float rhs)	rhs を乗算し、積を*this に格納します。
	float_complex& operator/=(float rhs)	rhs で除算し、商を*this に格納します。
	float_complex& operator=(const float_complex& rhs)	rhs をコピーします。
	float_complex& operator+=(const float_complex& rhs)	rhs を加算し、和を*this に格納します。
	float_complex& operator-=(const float_complex& rhs)	rhs を減算し、差を*this に格納します。
	float_complex& operator*=(const float_complex& rhs)	rhs を乗算し、積を*this に格納します。
	float_complex& operator/=(const float_complex& rhs)	rhs で除算し、商を*this に格納します。

float_complex::float_complex(float re = 0.0f, float im = 0.0f)

クラス float_complex のコンストラクタです。

以下の値で初期化します。

_re = re;

_im = im;

float_complex::float_complex(const double_complex& rhs)

クラス float_complex のコンストラクタです。

以下の値で初期化します。

_re = (float)rhs.real();

_im = (float)rhs.imag();

float float_complex::real() const

実数部を求めます。

リターン値は、this->_re です。

float float_complex::imag() const

虚数部を求めます。

リターン値は、this->_im です。

float_complex& float_complex::operator=(float rhs)

rhs を実数部(_re)にコピーします。虚数部(_im)は 0.0f を設定します。

リターン値は*this です。

float_complex& float_complex::operator+=(float rhs)

rhs を実数部(_re)に加算し、結果を実数部(_re)に格納します。虚数部(_im)の値は変わりません。

リターン値は*this です。

float_complex& float_complex::operator-=(float rhs)

rhs を実数部(_re)から減算し、結果を実数部(_re)に格納します。虚数部(_im)の値は変わりません。

リターン値は*this です。

float_complex& float_complex::operator*=(float rhs)

rhs と乗算し、結果を*this に格納します。

(_re=_re*rhs, _im=_im*rhs)

リターン値は*this です。

float_complex& float_complex::operator/=(float rhs)

rhs で除算し、結果を*this に格納します。

(_re=_re/rhs, _im=_im/rhs)

リターン値は*this です。

float_complex& float_complex::operator=(const float_complex& rhs)

rhs をコピーします。

リターン値は*this です。

float_complex& float_complex::operator+=(const float_complex& rhs)

rhs を加算し、結果を*this に格納します。

リターン値は*this です。

float_complex& float_complex::operator-=(const float_complex& rhs)

rhs を減算し、結果を*this に格納します。

リターン値は*this です。

float_complex& float_complex::operator*=(const float_complex& rhs)

rhs と乗算し、結果を*this に格納します。

リターン値は*this です。

float_complex& float_complex::operator/=(const float_complex& rhs)

rhs で除算し、結果を*this に格納します。

リターン値は*this です。

(b) float_complexメンバ外関数

種別	定義名	説明
関数	float_complex operator+(const float_complex& lhs)	lhs の単項＋演算を行います。
	float_complex operator+(const float_complex& lhs, const float_complex& rhs)	lhs と rhs を加算した結果を返却します。
	float_complex operator+(const float_complex& lhs, const float& rhs)	
	float_complex operator+(const float& lhs, const float_complex& rhs)	
	float_complex operator-(const float_complex& lhs)	lhs の単項－演算を行います。
	float_complex operator-(const float_complex& lhs, const float_complex& rhs)	lhs から rhs を減算した結果を返却します。
	float_complex operator-(const float_complex& lhs, const float& rhs)	
	float_complex operator-(const float& lhs, const float_complex& rhs)	
	float_complex operator*(const float_complex& lhs, const float_complex& rhs)	lhs と rhs を乗算した結果を返却します。
	float_complex operator*(const float_complex& lhs, const float& rhs)	
	float_complex operator*(const float& lhs, const float_complex& rhs)	
	float_complex operator/(const float_complex& lhs, const float_complex& rhs)	lhs を rhs で除算した結果を返却します。
	float_complex operator/(const float_complex& lhs, const float& rhs)	
	float_complex operator/(const float& lhs, const float_complex& rhs)	

種別	定義名	説明
関数	bool operator==(const float_complex& lhs, const float_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator==(const float_complex& lhs, const float& rhs)	
	bool operator==(const float& lhs, const float_complex& rhs)	
	bool operator!=(const float_complex& lhs, const float_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator!=(const float_complex& lhs, const float& rhs)	
	bool operator!=(const float& lhs, const float_complex& rhs)	
	istream& operator>>(istream& is, float_complex& x)	u,(u),または(u,v) (u:実数部、v:虚数部)形式の x を入力します。
	ostream& operator<<(ostream& os, float_complex& x)	x を u,(u)または (u,v) (u:実数部、v:虚数部)形式で出力します。
	float real(const float_complex& x)	実数部を求めます。
	float imag(const float_complex& x)	虚数部を求めます。
	float abs(const float_comlex& x)	絶対値を求めます。
	float arg(const float_complex& x)	位相角度を求めます。
	float norm(const float_complex& x)	２乗の絶対値を求めます。
	float_complex conj(const float_complex& x)	共役複素数を求めます。
	float_complex polar(const float& rho, const float& theta)	大きさが rho で位相角度が theta の複素数に対応する float_complex 値を求めます。
	float_complex cos(const float_complex& x)	複素余弦を求めます。
	float_complex cosh(const float_complex& x)	複素双曲余弦を求めます。
	float_complex exp(const float_complex& x)	指数関数を求めます。
	float_complex log(const float_complex& x)	自然対数を求めます。
	float_complex log10(const float_complex& x)	常用対数を求めます。

種別	定義名	説明
関数	float_complex pow(const float_complex& x, int y)	x の y 乗を求めます。
	float_complex pow(const float_complex& x, const float& y)	
	float_complex pow(const float_complex& x, const float_complex& y)	
	float_complex pow(const float& x, const float_complex& y)	
	float_complex sin(const float_complex& x)	複素正弦を求めます。
	float_complex sinh(const float_complex& x)	複素双曲正弦を求めます。
	float_complex sqrt(const float_complex& x)	右半空間における範囲での平方根を求めます。
	float_complex tan(const float_complex& x)	複素正接を求めます。
	float_complex tanh(const float_complex& x)	複素双曲正接を求めます。

float_complex operator+(const float_complex& lhs)

lhs の単項＋演算を行います。

リターン値は lhs です。

float_complex operator+(const float_complex& lhs, const float_complex& rhs)
float_complex operator+(const float_complex& lhs, const float& rhs)
float_complex operator+(const float& lhs, const float_complex& rhs)

lhs と rhs を加算した結果を返却します。

リターン値は、float_complex(lhs)+=rhs です。

float_complex operator-(const float_complex& lhs)

lhs の単項－演算を行います。

リターン値は、float_complex(-lhs.real(),-lhs.imag())です。

float_complex operator-(const float_complex& lhs, const float_complex& rhs)
float_complex operator-(const float_complex& lhs, const float& rhs)
float_complex operator-(const float& lhs, const float_complex& rhs)

lhs から rhs を減算した結果を返却します。

リターン値は、float_complex(lhs)-=rhs です。

float_complex operator*(const float_complex& lhs, const float_complex& rhs)
float_complex operator*(const float_complex& lhs, const float& rhs)
float_complex operator*(const float& lhs, const float_complex& rhs)

lhs と rhs を乗算した結果を返却します。

リターン値は、float_complex(lhs)*=rhs です。

float_complex operator/(const float_complex& lhs, const float_complex& rhs)
float_complex operator/(const float_complex& lhs, const float& rhs)
float_complex operator/(const float& lhs, const float_complex& rhs)

lhs を rhs で除算した結果を返却します。

リターン値は、float_complex(lhs)/=rhs です。

bool operator==(const float_complex& lhs, const float_complex& rhs)
bool operator==(const float_complex& lhs, const float& rhs)
bool operator==(const float& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。

リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag()です。

bool operator!=(const float_complex& lhs, const float_complex& rhs)
bool operator!=(const float_complex& lhs, const float& rhs)
bool operator!=(const float& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。

リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()です。

istream& operator>>(istream& is, float_complex& x)

u,(u), または(u,v) (u は実数部、v は虚数部)の形式の x を入力します。入力値は float_complex に変換されます。

u,(u),(u,v)形式以外が入力された場合は、is.setstate(ios_base::failbit)を呼びます。

リターン値は is です。

ostream& operator<<(ostream& os, const float_complex& x)

x を os に出力します。

出力形式は u,(u)または(u,v) (u は実数部、v は虚数部)です。

リターン値は os です。

float real(const float_complex& x)

実数部を求めます。

リターン値は x.real()です。

float imag(const float_complex& x)

虚数部を求めます。

リターン値は x.imag()です。

float abs(const float_complex& x)

絶対値を求めます。

リターン値は、$(|x.real()|^2 + |x.imag()|^2)^{1/2}$ です。

float arg(const float_complex& x)

位相角度を求めます。

リターン値は、atan2f(x.imag() , x.real())です。

float norm(const float_complex& x)

2 乗の絶対値を求めます。

リターン値は、$|x.real()|^2 + |x.imag()|^2$ です。

float_complex conj(const float_complex& x)

共役複素数を求めます。

リターン値は、float_complex(x.real(), (-1)*x.imag())です。

float_complex polar(const float& rho, const float& theta)

大きさが rho で位相角度(偏角)が theta の複素数に対応する float_complex 値を求めます。

リターン値は、float_complex(rho*cosf(theta), rho*sinf(theta))です。

float_complex cos(const float_complex& x)

複素余弦を求めます。

リターン値は、float_complex(cosf(x.real())*coshf(x.imag()), (-1)*sinf(x.real())*sinhf(x.imag()))です。

float_complex cosh(const float_complex& x)

複素双曲余弦を求めます。

リターン値は、cos(float_complex((-1)*x.imag(), x.real()))です。

float_complex exp(const float_complex& x)

指数関数を求めます。

リターン値は、expf(x.real())*cosf(x.imag()),expf(x.real())*sinf(x.imag())です。

float_complex log(const float_complex& x)

(e を底とする)自然対数を求めます。

リターン値は、float_complex(logf(abs(x)), arg(x))です。

float_complex log10(const float_complex& x)

(10 を底とする)常用対数を求めます。

リターン値は、float_complex(log10f(abs(x)), arg(x)/logf(10))です。

float_complex pow(const float_complex& x, int y)
float_complex pow(const float_complex& x, const float& y)
float_complex pow(const float_complex& x, const float_complex& y)
float_complex pow(const float& x, const float_complex& y)

x の y 乗を求めます。

pow(0,0)のとき、定義域エラーになります。

リターン値は次のとおりです。

float_complex pow(const float_complex& x,const float_complex& y)の場合 ：exp(y*logf(x))
上記以外 ：exp(y*log(x))

float_complex sin(const float_complex& x)

複素正弦を求めます。

リターン値は、float_complex(sinf(x.real())*coshf(x.imag()), cosf(x.real())*sinhf(x.imag()))です。

float_complex sinh(const float_complex& x)

複素双曲正弦を求めます。

リターン値は、float_complex(0,-1)*sin(float_complex((-1)*x.imag(),x.real())) です。

float_complex sqrt(const float_complex& x)

右半空間における範囲での平方根を求めます。

リターン値は、float_complex(sqrtf(abs(x))*cosf(arg(x)/2), sqrtf(abs(x))*sinf(arg(x)/2))です。

float_complex tan(const float_complex& x)

複素正接を求めます。

リターン値は、sin(x)/cos(x)です。

float_complex tanh(const float_complex& x)

複素双曲正接を求めます。

リターン値は、sinh(x)/cosh(x)です。

(c) double_complexクラス

種別	定義名	説明
型	value_type	double 型です。
変数	_re	double 精度の実数部を定義します。
	_im	double 精度の虚数部を定義します。
関数	double_complex(double re = 0.0, double im = 0.0)	コンストラクタです。
	double_complex(const float_complex&)	
	double real() const	実数部を求めます。
	double imag() const	虚数部を求めます。
	double_complex& operator=(double rhs)	rhs を実数部にコピーします。虚数部は 0.0 を設定します。
	double_complex& operator+=(double rhs)	rhs を実数部に加算し、和を*this に格納します。
	double_complex& operator-=(double rhs)	rhs を実数部から減算し、差を*this に格納します。
	double_complex& operator*=(double rhs)	rhs を乗算し、積を*this に格納します。
	double_complex& operator/=(double rhs)	rhs で除算し、商を*this に格納します。
	double_complex& operator=(const double_complex& rhs)	rhs をコピーします。
	double_complex& operator+=(const double_complex& rhs)	rhs を加算し、和を*this に格納します。
	double_complex& operator-=(const double_complex& rhs)	rhs を減算し、差を*this に格納します。
	double_complex& operator*=(const double_complex& rhs)	rhs を乗算し、積を*this に格納します。
	double_complex& operator/=(const double_complex& rhs)	rhs で除算し、商を*this に格納します。

double_complex::double_complex(double re = 0.0, double im = 0.0)

クラス double_complex のコンストラクタです。

以下の値で初期化します。

_re = re;

_im = im;

double_complex::double_complex(const float_complex&)

クラス double_complex のコンストラクタです。

以下の値で初期化します。

_re = (double)rhs.real();

_im = (double)rhs.imag();

double double_complex::real() const

実数部を求めます。

リターン値は、this->_re です。

double double_complex::imag() const

虚数部を求めます。

リターン値は、this->_im です。

double_complex& double_complex::operator=(double rhs)

rhs を実数部(_re)にコピーします。虚数部(_im)は 0.0 を設定します。

リターン値は*this です。

double_complex& double_complex::operator+=(double rhs)

rhs を実数部(_re)に加算し、結果を実数部(_re)に格納します。虚数部(_im)の値は変わりません。

リターン値は*this です。

double_complex& double_complex::operator-=(double rhs)

rhs を実数部(_re)から減算し、結果を実数部(_re)に格納します。虚数部(_im)の値は変わりません。

リターン値は*this です。

double_complex& double_complex::operator*=(double rhs)

rhs と乗算し、結果を*this に格納します。

(_re=_re*rhs, _im=_im*rhs)

リターン値は*this です。

double_complex& double_complex::operator/=(double rhs)

rhs で除算し、結果を*this に格納します。

(_re=_re/rhs, _im=_im/rhs)

リターン値は*this です。

double_complex& double_complex::operator=(const double_complex& rhs)

rhs をコピーします。

リターン値は*this です。

double_complex& double_complex::operator+=(const double_complex& rhs)

rhs を加算し、結果を*this に格納します。

リターン値は*this です。

double_complex& double_complex::operator-=(const double_complex& rhs)

rhs を減算し、結果を*this に格納します。

リターン値は*this です。

double_complex& double_complex::operator*=(const double_complex& rhs)

rhs と乗算し、結果を*this に格納します。

リターン値は*this です。

double_complex& double_complex::operator/=(const double_complex& rhs)

rhs で除算し、結果を*this に格納します。

リターン値は*this です。

（d） double_complexメンバ外関数

種別	定義名	説明
関数	double_complex operator+(const double_complex& lhs)	lhs の単項＋演算を行います。
	double_complex operator+(const double_complex& lhs, const double_complex& rhs)	lhs と rhs を加算し、和を lhs に格納します。
	double_complex operator+(const double_complex& lhs, const double& rhs)	
	double_complex operator+(const double& lhs, const double_complex& rhs)	
	double_complex operator-(const double_complex& lhs)	lhs の単項－演算を行います。
	double_complex operator-(const double_complex& lhs, const double_complex& rhs)	lhs から rhs を減算し、差を lhs に格納します。
	double_complex operator-(const double_complex& lhs, const double& rhs)	
	double_complex operator-(const double& lhs, const double_complex& rhs)	
	double_complex operator*(const double_complex& lhs, const double_complex& rhs)	lhs と rhs を乗算し、積を lhs に格納します。
	double_complex operator*(const double_complex& lhs, const double& rhs)	
	double_complex operator*(const double& lhs, const double_complex& rhs)	
	double_complex operator/(const double_complex& lhs, const double_complex& rhs)	lhs を rhs で除算し、商を lhs に格納します。
	double_complex operator/(const double_complex& lhs, const double& rhs)	
	double_complex operator/(const double& lhs, const double_complex& rhs)	

種別	定義名	説明
関数	bool operator==(const double_complex& lhs, const double_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator==(const double_complex& lhs, const double& rhs)	
	bool operator==(const double& lhs, const double_complex& rhs)	
	bool operator!=(const double_complex& lhs, const double_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator!=(const double_complex& lhs, const double& rhs)	
	bool operator!=(const double& lhs, const double_complex& rhs)	
	istream& operator>>(istream& is, double_complex& x)	u,(u)または(u,v) (u:実数部、v:虚数部)形式の x を入力します。
	ostream& operator<<(ostream& os, const double_complex& x)	x を u,(u)または (u,v) (u:実数部、v:虚数部)形式で出力します。
	double real(const double_complex& x)	実数部を求めます。
	double imag(const double_complex& x)	虚数部を求めます。
	double abs(const double_comlex& x)	絶対値を求めます。
	double arg(const double_complex& x)	位相角度を求めます。
	double norm(const double_complex& x)	２乗の絶対値を求めます。
	double_complex conj(const double_complex& x)	共役複素数を求めます。
	double_complex polar(const double& rho, const double& theta)	大きさが rho で位相角度が theta の複素数に対応する double_complex 値を求めます。
	double_complex cos(const double_complex& x)	複素余弦を求めます。
	double_complex cosh(const double_complex& x)	複素双曲余弦を求めます。
	double_complex exp(const double_complex&)	指数関数を求めます。

種別	定義名	説明
関数	double_complex log(const double_complex& x)	自然対数を求めます。
	double_complex log10(const double_complex& x)	常用対数を求めます。
	double_complex pow(const double _complex& x, int y)	x の y 乗を求めます。
	double_complex pow(const double _complex& x, const double & y)	
	double_complex pow(const double _complex& x, const double _complex& y)	
	double_complex pow(const double & x, const double _complex& y)	
	double_complex sin(const double_complex& x)	複素正弦を求めます。
	double_complex sinh(const double_complex& x)	複素双曲正弦を求めます。
	double_complex sqrt(const double_complex& x)	右半空間における範囲での平方根を求めます。
	double_complex tan(const double_complex& x)	複素正接を求めます。
	double_complex tanh(const double_complex& x)	複素双曲正接を求めます。

double_complex operator+(const double_complex& lhs)

lhs の単項＋演算を行います。

リターン値は lhs です。

double_complex operator+(const double_complex& lhs, const double_complex& rhs)
double_complex operator+(const double_complex& lhs, const double& rhs)
double_complex operator+(const double& lhs, const double_complex& rhs)

lhs と rhs を加算し、結果を lhs に格納します。

リターン値は、double_complex(lhs)+=rhs です。

double_complex operator-(const double_complex& lhs)

lhs の単項－演算を行います。

リターン値は、double_complex(-lhs.real(), -lhs.imag())です。

double_complex operator-(const double_complex& lhs, const double_complex& rhs)
double_complex operator-(const double_complex& lhs, const double& rhs)
double_complex operator-(const double& lhs, const double_complex& rhs)

lhs から rhs を減算し、結果を lhs に格納します。

リターン値は、double_complex(lhs)-=rhs です。

double_complex operator*(const double_complex& lhs, const double_complex& rhs)
double_complex operator*(const double_complex& lhs, const double& rhs)
double_complex operator*(const double& lhs, const double_complex& rhs)

lhs と rhs を乗算し、結果を lhs に格納します。

リターン値は、double_complex(lhs)*=rhs です。

double_complex operator/(const double_complex& lhs, const double_complex& rhs)
double_complex operator/(const double_complex& lhs, const double& rhs)
double_complex operator/(const double& lhs, const double_complex& rhs)

lhs を rhs で除算し、結果を lhs に格納します。

リターン値は、double_complex(lhs)/=rhs です。

bool operator==(const double_complex& lhs, const double_complex& rhs)
bool operator==(const double_complex& lhs, const double& rhs)
bool operator==(const double& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。

リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag()です。

bool operator!=(const double_complex& lhs, const double_complex& rhs)
bool operator!=(const double_complex& lhs, const double& rhs)
bool operator!=(const double& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。

リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()です。

istream& operator>>(istream& is, double_complex& x)

u,(u)または(u,v) (u は実数部、v は虚数部)の形式の複素数 x を入力します。入力値は double_complex に変換されます。

u,(u),(u,v)形式以外が入力された場合は、is.setstate(ios_base::failbit)を呼びます。

リターン値は is です。

ostream& operator<<(ostream& os, const double_complex& x)

x を os に出力します。

出力形式は u,(u)または(u,v) (u は実数部、v は虚数部)です。

リターン値は os です。

double real(const double_complex& x)

実数部を求めます。

リターン値は x.real()です。

double imag(const double_complex& x)

虚数部を求めます。

リターン値は x.imag()です。

double abs(const double_complex& x)

絶対値を求めます。

リターン値は、$(|x.real()|^2 + |x.imag()|^2)^{1/2}$ です。

double arg(const double_complex& x)

位相角度を求めます。

リターン値は、atan2(x.imag() , x.real())です。

double norm(const double_complex& x)

2 乗の絶対値を求めます。

リターン値は、$|x.real()|^2 + |x.imag()|^2$ です。

double_complex conj(const double_complex& x)

共役複素数を求めます。

リターン値は、double_complex(x.real(), (-1)*x.imag())です。

double_complex polar(const double& rho, const double& theta)

大きさが rho で位相角度(偏角)が theta の複素数に対応する double_complex 値を求めます。

リターン値は、double_complex(rho*cos(theta), rho*sin(theta))です。

double_complex cos(const double_complex& x)

複素余弦を求めます。

リターン値は、double_complex(cos(x.real())*cosh(x.imag()), (-1)*sin(x.real())*sinh(x.imag()))です。

double_complex cosh(const double_complex& x)

複素双曲余弦を求めます。

リターン値は、cos(double_complex((-1)*x.imag(), x.real()))です。

double_complex exp(const double_complex& x)

指数関数を求めます。

リターン値は、exp(x.real())*cos(x.imag()),exp(x.real())*sin(x.imag())です。

double_complex log(const double_complex& x)

(e を底とする)自然対数を求めます。

リターン値は、double_complex(log(abs(x)), arg(x))です。

double_complex log10(const double_complex& x)

(10 を底とする)常用対数を求めます。

リターン値は、double_complex(log10(abs(x)), arg(x)/log(10))です。

double_complex pow(const double_complex& x, int y)
double_complex pow(const double_complex& x, const double& y)
double_complex pow(const double_complex& x, const double_complex& y)
double_complex pow(const double& x, const double_complex& y)

x の y 乗を求めます。

pow(0,0)のとき、定義域エラーになります。

リターン値は、exp(y*log(x))です。

double_complex sin(const double_complex& x)

複素正弦を求めます。

リターン値は、double_complex(sin(x.real())*cosh(x.imag()), cos(x.real())*sinh(x.imag()))です。

double_complex sinh(const double_complex& x)

複素双曲正弦を求めます。

リターン値は、double_complex(0,-1)*sin(double_complex((-1)*x.imag(),x.real()))です。

double_complex sqrt(const double_complex& x)

右半空間における範囲での平方根を求めます。

リターン値は、double_complex(sqrt(abs(x))*cos(arg(x)/2), sqrt(abs(x))*sin(arg(x)/2))です。

double_complex tan(const double_complex& x)

複素正接を求めます。

リターン値は、sin(x)/cos(x)です。

double_complex tanh(const double_complex& x)

複素双曲正接を求めます。

リターン値は、sinh(x)/cosh(x)です。

(5) 文字列操作用クラスライブラリ

文字列操作用クラスライブラリに対応するヘッダファイルは以下の通りです。

- <string>

  stringクラスを定義します。

本クラスには派生関係はありません。

(a) stringクラス

種別	定義名	説明
型	iterator	char*型です。
	const_iterator	const char*型です。
定数	npos	文字列の最大長(UINT_MAX 文字)です。
変数	s_ptr	オブジェクトが文字列を格納している領域へのポインタです。
	s_len	オブジェクトが格納している文字列長です。
	s_res	オブジェクトが文字列を格納するために確保している領域のサイズです。
関数	string(void)	コンストラクタです。
	string::string(const string& str, size_t pos = 0, size_t n = npos)	
	string::string(const char* str, size_t n)	
	string::string(const char* str)	
	string::string(size_t n, char c)	
	~string()	デストラクタです。
	string& operator=(const string& str)	str を代入します。
	string& operator=(const char* str)	
	string& operator=(char c)	c を代入します。
	iterator begin()	文字列の先頭ポインタを求めます。
	const_iterator begin() const	
	iterator end()	文字列の最後尾ポインタを求めます。
	const_iterator end() const	
	size_t size() const	格納されている文字列の文字列長を求めます。
	size_t length() const	
	size_t max_size() const	確保している領域のサイズを求めます。
	void resize(size_t n, char c)	格納可能な文字列の文字数を n に変更します。
	void resize(size_t n)	格納可能な文字列の文字数を n に変更します。

種別	定義名	説明
関数	size_t capacity() const	確保している領域のサイズを求めます。
	void reserve(size_t res_arg = 0)	領域の再割り当てを行います。
	void clear()	格納されている文字列を clear します。
	bool empty() const	格納している文字列の文字数が 0 かチェックします。
	const char& operator[](size_t pos) const	s_ptr[pos]を参照します。
	char& operator[](size_t pos)	
	const char& at(size_t pos) const	
	char& at(size_t pos)	
	string& operator+=(const string& str)	str の文字列を追加します。
	string& operator+=(const char* str)	
	string& operator+=(char c)	c の文字を追加します。
	string& append(const string& str)	str の文字列を追加します。
	string& append(const char* str)	
	string& append(const string& str, size_t pos, size_t n)	オブジェクトの位置 pos に str の文字列を n 文字分追加します。
	string& append(const char* str, size_t n)	文字列 str の n 文字分を追加します。
	string& append(size_t n, char c)	n 個の文字 c を追加します。
	string& assign(const string& str)	str の文字列を代入します。
	string& assign(const char* str)	
	string& assign(const string& str, size_t pos, size_t n)	位置 pos に文字列 str の n 文字分を代入します。
	string& assign(const char* str, size_t n)	文字列 str の n 文字分を代入します。
	string& assign(size_t n, char c)	n 個の文字 c を代入します。
	string& insert(size_t pos1, const string& str)	位置 pos1 に str の文字列を挿入します。
	string& insert(size_t pos1, const string& str, size_t pos2, size_t n)	位置 pos1 に str の文字列の位置 pos2 から n 文字分を挿入します。
	string& insert(size_t pos, const char* str, size_t n)	pos の位置に文字列 str を n 文字分挿入します。
	string& insert(size_t pos, const char* str)	pos の位置に文字列 str を挿入します。

種別	定義名	説明
関数	string& insert(size_t pos, size_t n, char c)	位置 pos に n 個の文字 c の文字列を挿入します。
	iterator insert(iterator p, char c = char())	p が指す文字列の前に文字 c を挿入します。
	void insert(iterator p, size_t n, char c)	p が指す文字の前に、n 個の文字 c を挿入します。
	string& erase(size_t pos = 0, size_t n = npos)	位置 pos から n 個分取り除きます。
	iterator erase(iterator position)	position により参照された文字を取り除きます。
	iterator erase(iterator first, iterator last)	範囲[first, last]において文字を取り除きます。
	string& replace(size_t pos1, size_t n1, const string& str)	位置 pos1 から n1 文字分の文字列を、str の文字列で置き換えます。
	string& replace(size_t pos1, size_t n1, const char* str)	
	string& replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)	位置 pos1 から n1 文字分の文字列を、str の位置 pos2 から n2 文字分の文字列で置き換えます。
	string& replace(size_t pos, size_t n1, const char* str, size_t n2)	位置 pos から n1 文字分の文字列を、n2 個の str の文字列で置き換えます。
	string& replace(size_t pos, size_t n1, size_t n2, char c)	位置 pos から n1 文字分の文字列を、n2 個の文字 c で置き換えます。
	string& replace(iterator i1, iterator i2, const string& str)	位置 i1 から i2 までの文字列を str の文字列で置き換えます。
	string& replace(iterator i1, iterator i2, const char* str)	

種別	定義名	説明
関数	string& replace(iterator i1, iterator i2, const char* str, size_t n)	位置 i1 から i2 までの文字列を str の文字列の n 文字分で置き換えます。
	string& replace(iterator i1, iterator i2, size_t n, char c)	位置 i1 から i2 までの文字列を n 個の文字 c で置き換えます。
	size_t copy(char* str, size_t n, size_t pos = 0) const	位置 pos に文字列 str の n 文字分の文字列をコピーします。
	void swap(string& str)	str の文字列と交換します。
	const char* c_str() const	文字列を格納している領域へのポインタを参照します。
	const char* data() const	
	size_t find(const string& str, size_t pos = 0) const	位置 pos 以降で str の文字列と同じ文字列が最初に現れる位置を検索します。
	size_t find(const char* str, size_t pos = 0) const	
	size_t find(const char* str, size_t pos, size_t n) const	位置 pos 以降で str の n 文字分と同じ文字列が最初に現れる位置を検索します。
	size_t find(char c, size_t pos = 0) const	位置 pos 以降で文字 c が最初に現れる位置を検索します。
	size_t rfind(const string& str, size_t pos = npos) const	位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。
	size_t rfind(const char* str, size_t pos = npos) const	
	size_t rfind(const char* str, size_t pos, size_t n) const	位置 pos 以前で str の n 文字分と同じ文字列が最後に現れる位置を検索します。
	size_t rfind(char c, size_t pos = npos) const	位置 pos 以前で文字 c が最後に現れる位置を検索します。

種別	定義名	説明
関数	size_t find_first_of(const string& str, size_t pos = 0) const	位置 pos 以降で文字列 str に含まれる任意の文字が最初に現れる位置を検索します。
	size_t find_first_of(const char* str, size_t pos = 0) const	
	size_t find_first_of(const char* str, size_t pos, size_t n) const	位置 pos 以降で文字列 str の n 文字分に含まれる任意の文字が最初に現れる位置を検索します。
	size_t find_first_of(char c, size_t pos = 0) const	位置 pos 以降で文字 c が最初に現れる位置を検索します。
	size_t find_last_of(const string& str, size_t pos = npos) const	位置 pos 以前で文字列 str に含まれる任意の文字が最後に現れる位置を検索します。
	size_t find_last_of(const char* str, size_t pos = npos) const	
	size_t find_last_of(const char* str, size_t pos, size_t n) const	位置 pos 以前で文字列 str の n 文字分に含まれる任意の文字が最後に現れる位置を検索します。
	size_t find_last_of(char c, size_t pos = npos) const	位置 pos 以前で文字 c が最後に現れる位置を検索します。
	size_t find_first_not_of(const string& str, size_t pos = 0) const	位置 pos 以降で str 中の任意の文字と異なった文字が最初に現れる位置を検索します。
	size_t find_first_not_of(const char* str, size_t pos = 0) const	
	size_t find_first_not_of(const char* str, size_t pos, size_t n)	位置 pos 以降で str の先頭から n 文字までの任意の文字と異なった文字が最初に現れる位置を検索します。
	size_t find_first_not_of(char c, size_t pos = 0) const	位置 pos 以降で文字 c と異なった文字が最初に現れる位置を検索します。
	size_t find_last_not_of(const string& str, size_t pos = npos) const	位置 pos 以前で str 中の任意の文字と異なった文字が最後に現れる位置を検索します。
	size_t find_last_not_of(const char* str, size_t pos = npos) const	

種別	定義名	説明
関数	size_t find_last_not_of(const char* str, size_t pos, size_t n) const	位置 pos 以前で str の先頭から n 文字までの任意の文字と異なった文字が最後に現れる位置を検索します。
	size_t find_last_not_of(char c, size_t pos = npos) const	位置 pos 以前で文字 c と異なった文字が最後に現れる位置を検索します。
	string substr(size_t pos = 0, size_t n = npos) const	格納された文字列に対し、範囲[pos,n]の文字列を持つオブジェクトを生成します。
	int compare(const string& str) const	文字列と str の文字列を比較します。
	int compare(size_t pos1, size_t n1, const string& str) const	位置 pos1 から n1 文字分の文字列と str を比較します。
	int compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const	位置 pos1 から n1 文字分の文字列と str の位置 pos2 から n2 文字分の文字列を比較します。
	int compare(const char* str) const	str と比較します。
	int compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const	位置 pos1 から n1 文字分の文字列と str の n2 文字分の文字列を比較します。

string::string(void)

以下のように設定します。

s_ptr = 0;

s_len = 0;

s_res = 1;

string::string(const string& str, size_t pos = 0, size_t n = npos)

str をコピーします。ただし、s_len は、n と s_len の小さい方の値になります。

string::string(const char* str, size_t n)

以下に設定します。

s_ptr = str;

s_len = n;

s_res = n+1;

string::string(const char* str)

以下に設定します。

s_ptr = str;

s_len = str の文字列長;

s_res = str の文字列長+1;

string::string(size_t n, char c)

以下に設定します。

s_ptr = 文字数 n で文字 c の文字列;

s_len = n;

s_res = n+1;

string::~string()

クラス string のデストラクタです。

文字列を格納している領域を解放します。

string& string::operator=(const string& str)

str のデータを代入します。

リターン値は*this です。

string& string::operator=(const char* str)

str から string オブジェクトを生成し、そのデータを代入します。

リターン値は*this です。

string& string::operator=(char c)

c から string オブジェクトを生成し、そのデータを代入します。

リターン値は*this です。

string::iterator string::begin()
string::const_iterator string::begin() const

文字列の先頭ポインタを求めます。

リターン値は、文字列の先頭ポインタです。

string::iterator string::end()
string::const_iterator string::end() const

文字列の最後尾ポインタを求めます。

リターン値は、文字列の最後尾ポインタです。

size_t string::size() const
size_t string::length() const

格納されている文字列の文字列長を求めます。

リターン値は、格納されている文字列の文字列長です。

size_t string::max_size() const

確保している領域のサイズを求めます。

リターン値は、確保している領域のサイズです。

void string::resize(size_t n, char c)

オブジェクトが格納可能な文字列の文字数を n に変更します。

n<=size()のとき、長さを n にした元の文字列と置き換えます。

n>size()のとき、元の文字列の後ろに長さ n になるまで c をつめた文字列と置き換えます。

n<=max_size()である必要があります。

n>max_size()の場合、n=max_size()として計算します。

void string::resize(size_t n)

オブジェクトが格納可能な文字列の文字数を n に変更します。

n<=size()のとき、長さを n にしたもとの文字列と置き換えます。

n<=max_size()である必要があります。

size_t string::capacity() const

確保している領域のサイズを求めます。

リターン値は、確保している領域のサイズです。

void string::reserve(size_t res_arg = 0)

記憶領域の再割り当てを行います。

reserve()後、capacity()は reserve()の引数より大きいかまたは等しくなります。

再割り当てを行うと、すべての参照、ポインタ、この数列の中の要素の参照する iterator を無効にします。

void string::clear()

格納されている文字列をクリアします。

bool string::empty() const

格納している文字列の文字数が 0 かチェックします。

リターン値は次のとおりです。

格納している文字列長が 0 の場合	：true
格納している文字列長が 0 以外の場合	：false

const char& string::operator[](size_t pos) const
char& string::operator[](size_t pos)
const char& string::at(size_t pos) const
char& string::at(size_t pos)

s_ptr[pos]を参照します。

リターン値は次のとおりです。

n<s_len の場合	：s_ptr [pos]
n>=s_len の場合	：'\0'

string& string::operator+=(const string& str)

str が格納している文字列を追加します。

リターン値は*this です。

string& string::operator+=(const char* str)

str から string オブジェクトを生成し、その文字列を追加します。

リターン値は*this です。

string& string::operator+=(char c)

c から string オブジェクトを生成し、その文字列を追加します。

リターン値は*this です。

string& string::append(const string& str)
string& string::append(const char* str)

str の文字列をオブジェクトに追加します。

リターン値は*this です。

string& string::append(const string& str, size_t pos, size_t n)

オブジェクトの位置 pos に str の文字列を n 文字分追加します。

リターン値は*this です。

string& string::append(const char* str, size_t n)

文字列 str の n 文字分を追加します。

リターン値は*this です。

string& string::append(size_t n, char c)

n 個の文字 c を追加します。

リターン値は*this です。

string& string::assign(const string& str)
string& string::assign(const char* str)

str の文字列を代入します。

リターン値は*this です。

string& string::assign(const string& str, size_t pos, size_t n)

位置 pos に文字列 str の n 文字分を代入します。

リターン値は*this です。

string& string::assign(const char* str, size_t n)

文字列 str の n 文字分を代入します。

リターン値は*this です。

string& string::assign(size_t n, char c)

n 個の文字 c を代入します。

リターン値は*this です。

string& string::insert(size_t pos1, const string& str)

位置 pos1 に str の文字列を挿入します。

リターン値は*this です。

string& string::insert(size_t pos1, const string& str, size_t pos2, size_t n)

位置 pos1 に str の文字列の位置 pos2 から n 文字分を挿入します。

リターン値は*this です。

string& string::insert(size_t pos, const char* str, size_t n)

pos の位置に文字列 str を n 文字分挿入します。

リターン値は*this です。

string& string::insert(size_t pos, const char* str)

pos の位置に文字列 str を挿入します。

リターン値は*this です。

string& string::insert(size_t pos, size_t n, char c)

位置 pos に n 個の文字 c の文字列を挿入します。

リターン値は*this です。

string::iterator string::insert(iterator p, char c = char())

p が指す文字列の前に、文字 c を挿入します。

リターン値は、挿入された文字です。

void string::insert(iterator p, size_t n, char c)

p が指す文字の前に、n 個の文字 c を挿入します。

string& string::erase(size_t pos = 0, size_t n = npos)

位置 pos から n 個分取り除きます。

リターン値は*this です。

iterator string::erase(iterator position)

position により参照された文字を取り除きます。

リターン値は次のとおりです。

削除要素の次の iterator がある場合 ：削除要素の次の iterator

削除要素の次の iterator がない場合 ：end()

iterator string::erase(iterator first, iterator last)

範囲[first, last]において文字を取り除きます。

リターン値は次のとおりです。

last の次の iterator がある場合 ：last の次の iterator

last の次の iterator がない場合 ：end()

string& string::replace(size_t pos1, size_t n1, const string& str)
string& string::replace(size_t pos1, size_t n1, const char* str)

位置 pos1 から n1 文字分の文字列を、str の文字列で置き換えます。

リターン値は*this です。

string& string::replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)

位置 pos1 から n1 文字分の文字列を、str の位置 pos2 から n2 文字分の文字列で置き換えます。

リターン値は*this です。

string& string::replace(size_t pos, size_t n1, const char* str, size_t n2)

位置 pos から n1 文字分の文字列を、n2 個の str の文字列で置き換えます。

リターン値は*this です。

string& string::replace(size_t pos, size_t n1, size_t n2, char c)

位置 pos から n1 文字分の文字列を、n2 個の文字 c で置き換えます。

リターン値は*this です。

string& string::replace(iterator i1, iterator i2, const string& str)
string& string::replace(iterator i1, iterator i2, const char* str)

位置 i1 から i2 までの文字列を str の文字列で置き換えます。

リターン値は*this です。

string& string::replace(iterator i1, iterator i2, const char* str, size_t n)

位置 i1 から i2 までの文字列を、str の n 文字分の文字列で置き換えます。

リターン値は*this です。

string& string::replace(iterator i1, iterator i2, size_t n, char c)

位置 i1 から i2 までの文字列を、n 個の文字 c で置き換えます。

リターン値は*this です。

size_t string::copy(char* str, size_t n, size_t pos = 0) const

位置 pos に文字列 str の n 文字分の文字列をコピーします。

リターン値は rlen です。

void string::swap(string& str)

str の文字列と交換します。

const char* string::c_str() const
const char* string::data() const

文字列を格納している領域へのポインタを参照します。

リターン値は s_ptr です。

size_t string::find(const string& str, size_t pos = 0) const
size_t string::find(const char* str, size_t pos = 0) const

位置 pos 以降で str の文字列と同じ文字列が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find(const char* str, size_t pos, size_t n) const

位置 pos 以降で str の n 文字分と同じ文字列が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find(char c, size_t pos = 0) const

位置 pos 以降で文字 c が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::rfind(const string& str, size_t pos = npos) const
size_t string::rfind(const char* str, size_t pos = npos) const

位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::rfind(const char* str, size_t pos, size_t n) const

位置 pos 以前で文字列 str の n 文字分と同じ文字列が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::rfind(char c, size_t pos = npos) const

位置 pos 以前で文字 c が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_of(const string& str, size_t pos = 0) const
size_t string::find_first_of(const char* str, size_t pos = 0) const

位置 pos 以降で文字列 str に含まれる任意の文字が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_of(const char* str, size_t pos, size_t n) const

位置 pos 以降で文字列 str の n 文字分に含まれる任意の文字が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_of(char c, size_t pos = 0) const

位置 pos 以降で文字 c が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_of(const string& str, size_t pos = npos) const
size_t string::find_last_of(const char* str, size_t pos = npos) const

位置 pos 以前で文字列 str に含まれる任意の文字が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_of(const char* str, size_t pos, size_t n) const

位置 pos 以前で文字列 str の n 文字分に含まれる任意の文字が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_of(char c, size_t pos = npos) const

位置 pos 以前で文字 c が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(const string& str, size_t pos = 0) const
size_t string::find_first_not_of(const char* str, size_t pos = 0) const

位置 pos 以降で str 中の任意の文字と異なった文字が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(const char* str, size_t pos, size_t n) const

位置 pos 以降で str の先頭から n 文字までの任意の文字と異なった文字が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(char c, size_t pos = 0) const

位置 pos 以降で文字 c と異なった文字が最初に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(const string& str, size_t pos = npos) const
size_t string::find_last_not_of(const char* str, size_t pos = npos) const

位置 pos 以前で str 中の任意の文字と異なった文字が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(const char* str, size_t pos, size_t n) const

位置 pos 以前で str の先頭から n 文字までの任意の文字と異なった文字が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(char c, size_t pos = npos) const

位置 pos 以前で文字 c と異なった文字が最後に現れる位置を検索します。

リターン値は、文字列のオフセットです。

string string::substr(size_t pos = 0, size_t n = npos) const

格納された文字列に対し、範囲[pos,n]の文字列を持つオブジェクトを生成します。

リターン値は、範囲[pos,n]の文字列を持つオブジェクトです。

int string::compare(const string& str) const

文字列と str の文字列を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：0

文字列が異なる場合 ：this->s_len>str.s_len のとき 1

this->s_len < str.s_len のとき-1

int string::compare(size_t pos1, size_t n1, const string& str) const

位置 pos1 から n1 文字分の文字列と str を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：0

文字列が異なる場合 ：this->s_len>str.s_len のとき 1

this->s_len < str.s_len のとき-1

int string::compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const

位置 pos1 から n1 文字分の文字列と str の位置 pos2 から n2 文字分の文字列を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：0

文字列が異なる場合 ：this->s_len>str.s_len のとき 1

this->s_len < str.s_len のとき-1

int string::compare(const char* str) const

str と比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：0

文字列が異なる場合 ：this->s_len>str.s_len のとき 1

this->s_len < str.s_len のとき-1

int string::compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const

位置 pos1 から n1 文字分の文字列と str の n2 文字分の文字列を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：0

文字列が異なる場合 ：this->s_len>str.s_len のとき 1

this->s_len < str.s_len のとき-1

(b) stringクラスマニピュレータ

種別	定義名	説明
関数	string operator+(const string& lhs, const string& rhs)	lhs の文字列(または文字)に rhs の文字列(または文字)を追加し、オブジェクトを生成してその文字列を格納します。
	string operator+(const char* lhs, const string& rhs)	
	string operator+(char lhs, const string& rhs)	
	string operator+(const string& lhs, const char* rhs)	
	string operator+(const string& lhs, char rhs)	
	bool operator==(const string& lhs, const string& rhs)	lhs の文字列と rhs の文字列を比較します。
	bool operator==(const char* lhs, const string& rhs)	
	bool operator==(const string& lhs, const char* rhs)	
	bool operator!=(const string& lhs, const string& rhs)	lhs の文字列と rhs の文字列を比較します。
	bool operator!=(const char* lhs, const string& rhs)	
	bool operator!=(const string& lhs, const char* rhs)	
	bool operator<(const string& lhs, const string& rhs)	lhs の文字列長と rhs の文字列長を比較します。
	bool operator<(const char* lhs, const string& rhs)	
	bool operator<(const string& lhs, const char* rhs)	
	bool operator>(const string& lhs, const string& rhs)	lhs の文字列長と rhs の文字列長を比較します。
	bool operator>(const char* lhs, const string& rhs)	
	bool operator>(const string& lhs, const char* rhs)	
	bool operator<=(const string& lhs, const string& rhs)	lhs の文字列長と rhs の文字列長を比較します。
	bool operator<=(const char* lhs, const string& rhs)	
	bool operator<=(const string& lhs, const char* rhs)	
	bool operator>=(const string& lhs, const string& rhs)	lhs の文字列長と rhs の文字列長を比較します。
	bool operator>=(const char* lhs, const string& rhs)	
	bool operator>=(const string& lhs, const char* rhs)	
	void swap(string& lhs, string& rhs)	lhs の文字列と rhs の文字列を交換します。
	istream& operator>>(istream& is, string& str)	文字列を str に抽出します。
	ostream& operator<<(ostream& os, const string& str)	文字列を挿入します。

種別	定義名	説明
関数	istream& getline(istream& is, string& str, char delim)	is から文字列を抽出し，str に付加します。途中で文字'delim'を検出したら、入力を終了します。
	istream& getline(istream& is, string& str)	is から文字列を抽出し，str に付加します。途中で改行文字を検出したら、入力を終了します。

string operator+(const string& lhs, const string& rhs)
string operator+(const char* lhs, const string& rhs)
string operator+(char lhs, const string& rhs)
string operator+(const string& lhs, const char* rhs)
string operator+(const string& lhs, char rhs)

lhs の文字列(または文字)に rhs の文字列(または文字)を追加し、オブジェクトを生成してその文字列を格納します。

リターン値は、結合した文字列を格納するオブジェクトです。

bool operator==(const string& lhs, const string& rhs)
bool operator==(const char* lhs, const string& rhs)
bool operator==(const string& lhs, const char* rhs)

lhs の文字列と rhs の文字列を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：true
文字列が異なる場合 ：false

bool operator!=(const string& lhs, const string& rhs)
bool operator!=(const char* lhs, const string& rhs)
bool operator!=(const string& lhs, const char* rhs)

lhs の文字列と rhs の文字列を比較します。

リターン値は次のとおりです。

文字列が同一の場合 ：false
文字列が異なる場合 ：true

bool operator<(const string& lhs, const string& rhs)
bool operator<(const char* lhs, const string& rhs)
bool operator<(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。

リターン値は次のとおりです。

lhs.s_len < rhs.s_len の場合 ：true
lhs.s_len >= rhs.s_len の場合 ：false

bool operator>(const string& lhs, const string& rhs)
bool operator>(const char* lhs, const string& rhs)
bool operator>(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。

リターン値は次のとおりです。

lhs.s_len > rhs.s_len の場合 ：true

lhs.s_len <= rhs.s_len の場合 ：false

bool operator<=(const string& lhs, const string& rhs)
bool operator<=(const char* lhs, const string& rhs)
bool operator<=(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。

リターン値は次のとおりです。

lhs.s_len <= rhs.s_len の場合 ：true

lhs.s_len > rhs.s_len の場合 ：false

bool operator>=(const string& lhs, const string& rhs)
bool operator>=(const char* lhs, const string& rhs)
bool operator>=(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。

リターン値は次のとおりです。

lhs.s_len >= rhs.s_len の場合 ：true

lhs.s_len < rhs.s_len の場合 ：false

void swap(string& lhs, string& rhs)

lhs の文字列と rhs の文字列を交換します。

istream& operator>>(istream& is, string& str)

文字列を str に抽出します。

リターン値は is です。

ostream& operator<<(ostream& os, const string& str)

文字列を挿入します。

リターン値は os です。

istream& getline(istream& is, string& str, char delim)

is から文字列を抽出し、str に付加します。

途中で文字'delim'を検出したら、入力を終了します。

リターン値は is です。

istream& getline(istream& is, string& str)

is から文字列を抽出し、str に付加します。

途中で改行文字を検出したら、入力を終了します。

リターン値は is です。

9.3.3　リエントラントライブラリ

標準ライブラリ構築ツールで reent オプションを指定して作成したライブラリは、rand、srand 関数を除いてすべてリエントラントに実行できます。

reent オプションを指定していない場合について、**表** 9.39 にリエントラントライブラリ一覧を示します。表中、△で示した関数は、errno 変数を設定しますので、プログラム中で errno を参照していなければリエントラントに実行できます。

リエントラント欄　○：リエントラント　×：ノンリエントラント　△：errno を設定

表 9.39　**リエントラントライブラリ一覧**

標準インクルードファイル	関数名	リエントラント
stddef.h	offsetof	○
assert.h	assert	×
ctype.h	isalnum	○
	isalpha	○
	iscntrl	○
	isdigit	○
	isgraph	○
	islower	○
	isprint	○
	ispunct	○
	isspace	○
	isupper	○
	isxdigit	○
	tolower	○
	toupper	○
math.h	acos	△
	asin	△
	atan	△
	atan2	△
	cos	△
	sin	△
	tan	△
	cosh	△
	sinh	△
	tanh	△
	exp	△
math.h	frexp	△
	ldexp	△
	log	△
	log10	△
	modf	△
	pow	△
	sqrt	△
	ceil	△
	fabs	△
	floor	△
	fmod	△
mathf.h	acosf	△
	asinf	△
	atanf	△
	atan2f	△
	cosf	△
	sinf	△
	tanf	△
	coshf	△
	sinhf	△
	tanhf	△
	expf	△
	frexpf	△
	ldexpf	△
	logf	△
	log10f	△

標準インクルードファイル	関数名	リエントラント
mathf.h	modff	△
	powf	△
	sqrtf	△
	ceilf	△
	fabsf	△
	floorf	△
	fmodf	△
setjmp.h	setjmp	○
	longjmp	○
stdarg.h	va_start	○
	va_arg	○
	va_end	○
stdio.h	fclose	×
	fflush	×
	fopen	×
	freopen	×
	setbuf	×
	setvbuf	×
	fprintf	×
	fscanf	×
	printf	×
	scanf	×
	sprintf	△
	sscanf	△
	vfprintf	×
	vprintf	×
	vsprintf	△
	fgetc	×
	fgets	×
	fputc	×
	fputs	×
	getc	×
	getchar	×

標準インクルードファイル	関数名	リエントラント
stdio.h	gets	×
	putc	×
	putchar	×
	puts	×
	ungetc	×
	fread	×
	fwrite	×
	fseek	×
	ftell	×
	rewind	×
	clearerr	×
	feof	×
	ferror	×
	perror	×
stdlib.h	atof	△
	atoi	△
	atol	△
	atoll	△
	strtod	△
	strtol	△
	strtoul	△
	strtoll	△
	strtoull	△
	rand	×
	srand	×
	calloc	×
	free	×
	malloc	×
	realloc	×
	bsearch	○
	qsort	○
	abs	○
	div	△

標準インクルードファイル	関数名	リエントラント
string.h	labs	○
	llabs	○
	ldiv	△
	lldiv	△
	memcpy	○
	strcpy	○
	strncpy	○
	strcat	○
	strncat	○
	memcmp	○
	strcmp	○
	strncmp	○

標準インクルードファイル	関数名	リエントラント
string.h	memchr	○
	strchr	○
	strcspn	○
	strpbrk	○
	strrchr	○
	strspn	○
	strstr	○
	strtok	×
	memset	○
	strerror	○
	strlen	○
	memmove	○

9.3.4 未サポートライブラリ

C 言語仕様で定義しているライブラリのうち、本コンパイラでサポートしていないライブラリを**表** 9.40 に示します。

表 9.40 **サポートしていないライブラリ**

	ヘッダファイル	ライブラリ名
1	locale.h*[1]	setlocale、localeconv
2	signal.h*[1]	signal、raise
3	stdio.h	remove、rename、tmpfile、tmpnam、fgetpos、fsetpos
4	stdlib.h	abort、atexit、exit、_Exit、getenv、system、 mblen、mbtowc、wctomb、mbstowcs、wcstombs
5	string.h	strcoll、strxfrm
6	time.h*[1]	clock、difftime、mktime、time、asctime、ctime、gmtime、localtime、 strftime
7	wctype.h	iswalnum、iswalpha、iswblank、iswcntrl、iswdigit、iswgraph、 iswlower、iswprintf、iswpunct、iswspace、iswupper、iswxdigit、iswctype、 wctype、towlower、towupper、towctrans、wctrans
8	wchar.h	wcsftime、wcscoll、wcsxfrm、wctob、mbrtowc、wcrtomb、 mbsrtowcs, wcsrtombs

【注】 *1 **ヘッダファイルをサポートしません。**

10. アセンブラの言語仕様

10.1 プログラムの記述方法

10.1.1 予約語

アセンブラでは、アセンブラ制御命令やニーモニックなどと同一の文字列を予約語として扱います。予約語は特別な機能を持っているため、アセンブリ言語ファイル中でラベル名やシンボル名に使用することはできません。また、予約語は大文字と小文字を区別しません。"ABS"と"abs"は同じ予約語となります。

予約語には以下のものがあります。

(1) アセンブラ制御命令

アセンブラ制御命令と、ピリオド(.)で始まる文字列全てを予約語とします。

(2) ニーモニック

RXファミリのニーモニック全てを予約語とします。

(3) レジスタ・フラグ名

RXファミリのレジスタ名およびフラグ名全てを予約語とします。

(4) 演算子

本章で説明している全ての演算子を予約語とします。

(5) システムラベル

アセンブラが生成する、ピリオド2つから始まる名前をシステムラベルといい、システムラベルは全て予約語として扱います。

10.1.2 名前

名前はアセンブリ言語ファイルの中で任意に定義し使用できます。

名前は次の種類に分けられます。

表 10.1 名前の種類

名前の種類	内容
ラベル名	アドレスを値として持つ名前
シンボル名	定数を値として持つ名前 (シンボル名には、ラベル名も含みます)
セクション名	.SECTION 制御命令で定義されるセクションの名前
ロケーションシンボル名	ロケーションシンボル'$'が記述されている行のオペレーション部の先頭アドレスを示します
マクロ名	マクロの定義名

名前の記述規則

- 名前の文字数に制限はありません。
- 名前は大文字と小文字を区別して扱います。"LAB"と"Lab"は異なる名前として扱われます。
- 名前には英数字とアンダーライン(_)、ドル($)が使用できます。
- 名前の先頭文字に数字は使用できません。
- 名前に予約語を使用することはできません。

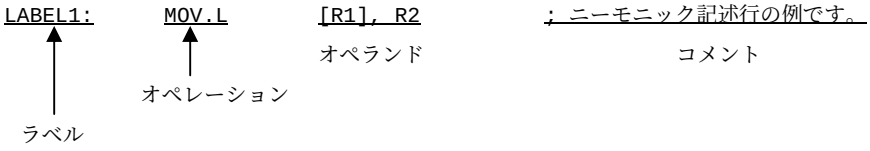
※ただし、セクション名にのみ予約語のフラグ名(U,I,O,S,Z,C)を使用することができます。

10.1.3 ニーモニック記述行の構成

ニーモニック記述行の構成を以下に記します。

[ラベル][オペレーション[△オペランド]][コメント]

(コーディング例)



(1) ラベル

ニーモニック記述行のアドレスに対して名前を定義します。

(2) オペレーション

ニーモニック、制御命令を書きます。

(3) オペランド

オペレーションの実行対象となるものを書きます。オペランドの個数と種類はオペレーションによって決まります。オペランドを必要としないオペレーションもあります。

(4) コメント

プログラムを分かりやすくするための注釈を書きます。

10.1.4 ラベルの記述方法

ラベルを記述する場合、名前の最後に必ずコロン(:)を付けます。

- 記述例

```
LABEL1:
```

定義されているセクション名と同じ名前のシンボルを定義することはできません。セクションとシンボルを同じ名前で定義した場合、後で定義したものは A2118 エラーとなります。

10.1.5 オペレーション部の記述方法

- 記述方法

  ニーモニック[サイズ指定子(分岐距離指定子)]

- 内容

命令は、以下２つの要素から構成されます。

(1) ニーモニック ・・・ 命令の動作を示します。

(2) サイズ指定子 ・・・ 処理対象のデータサイズを指定します。

(1) ニーモニック

ニーモニックは命令の動作を示します。

(例)

```
MOV ・・・ 転送命令
ADD ・・・ 算術演算命令(加算命令)
```

(2) サイズ指定子

サイズ指定子は、命令コードのオペランドサイズを指定するものです。

- 記述方法

```
.size
```

- 内容

  オペランドの演算サイズを指定します。正確にはその命令が処理を行うために読み出すデータのサイズを指定します。sizeは下記表の通りです。

表 10.2 サイズ指定子

size	内容
B	バイト(8 ビット)
W	ワード(16 ビット)
L	ロングワード(32 ビット)

size は大文字小文字いずれも可。

(例) MOV.B #0, R3 ・・・ バイト指定

本指定子は、「RX ファミリソフトウェアマニュアル」の命令フォーマットで (.size) が明記されているものについてのみ指定が可能でかつ必須となります。

(3) 分岐距離指定子

分岐距離指定子は、分岐命令および相対サブルーチン分岐命令で指定します。

- 記述方法

  `.length`

- 内容

  lengthは下記表の通りです。

表 10.3 分岐距離指定子

length	内容	
S	3 ビット PC 前方相対	(+3〜+10)
B	8 ビット PC 相対	(-128〜+127)
W	16 ビット PC 相対	(-32768〜+32767)
A	24 ビット PC 相対	(-8388608〜+8388607)
L	レジスタ相対	(-2147483648〜+2147183647)

length は大文字小文字いずれも可。

(例)

BRA.W label ・・・ 16 ビット相対指定

BRA.L R1 ・・・ レジスタ相対指定

本指定子は省略可能です。省略した場合は次の条件を全て満たした場合のみ、もっともオペコードが小さくなるように(S/B/W/A の中から)アセンブラがコードを選択します。

(1) オペランドが、レジスタ以外で記述されている場合

(2) オペランドが、アセンブル時に分岐距離が確定する分岐先である場合

(例) ラベル＋アセンブル時確定値
ラベル－アセンブル時確定値
アセンブル時確定値＋ラベル

(3) オペランドのラベルが同一セクション内で定義されている場合

また、オペランドがレジスタの場合、分岐距離指定子 Lが選択されます。

条件分岐命令の場合、分岐距離が規定の範囲を超えているときは条件を反転してコードを生成します。

各命令で指定可能な分岐距離指定子は、下記表の通りです。

表 10.4　分岐命令ごとの分岐距離指定子

命令		.S	.B	.W	.A	.L
BCnd	(Cnd=EQ/Z)	○	○	○	×	×
	(Cnd=NE/NZ)	○	○	○	×	×
	(Cnd=**上記以外**)	×	○	×	×	×
BRA		○	○	○	○	○
BSR		×	×	○	○	○

10.1.6　オペランド部の記述方法

(1) 数値

プログラムに記述できる数値の種類は以下 5 つがあります。

記述した値は 32 ビット符号付きで処理されます。(浮動小数点数を除く)

(a) 2 進数

0～1 のいずれかの数字で記述し、接尾辞として B または b を添付します。

- 記述例

```
1011000B
1011000b
```

(b) 8 進数

0～7 までの数字で記述し、接尾辞として O または o を添付します。

- 記述例

```
60702O
60702o
```

(c) 10 進数

0～9 までの数字で記述します。

- 記述例

```
9243
```

(d) 16 進数

0～9, A～F, a～f で記述し、接尾辞として H または h を添付します。

アルファベットで始まる数値の場合は接頭辞として 0 を添付します。

- 記述例

```
0A5FH
5FH
0a5fh
5fh
```

(e) 浮動小数点数

浮動小数点数は制御命令 ".FLOAT"と".DOUBLE" のオペランドにのみ記述できます。

浮動小数点数は式に記述できません。

浮動小数点数で表される次の範囲の値を記述できます。

FLOAT (32bits) 1.17549435×10^{-38} ～ 3.40282347×10^{38}

DOUBLE (64bits) $2.2250738585072014\times10^{-308}$ ～ $1.7976931348623157\times10^{308}$

- 記述方法

(仮数部)E(指数部)

(仮数部)e(指数部)

- 記述例

```
3.4E35      ;3.4×10**35
3.4e-35     ;3.4×10**-35
-.5E20      ;-0.5×10**20
5e-20       ;5.0×10**-20
```

(2) 式

数値、シンボルおよび演算子を組み合わせた式を記述できます。

- 演算子と数値の間には空白文字またはタブを記述できます。
- 演算子は複数組み合わせて記述できます。
- シンボル値として式を記述する場合は、式の値がアセンブル時に確定するように式を記述する必要があります。
- 式の項に文字定数は使用できません。
- 演算結果の範囲は、-2147483648～2147483647となります。演算結果がこの範囲を超えた場合のオーバフローの判断は行いません。

(a) 演算子

プログラムに記述できる演算子の一覧を示します。

- 単項演算子

表 10.5 単項演算子

演算子	機能
+	続く値を正の値として扱います。
-	続く値を負の値として扱います。
~	続く値の論理否定値を扱います。
SIZEOF	オペランドに指定したセクションのサイズ（バイト数）を値として扱います。
TOPOF	オペランドに指定したセクションの開始アドレス値として扱います。

SIZEOF, TOPOF は、オペランドとの間に空白文字またはタブを記述します。

(例) SIZEOF program

- 二項演算子

表 10.6 二項演算子

演算子	機能
+	左辺値と右辺値を加算します。
-	左辺値から右辺値を減算します。
*	左辺値と右辺値を乗算します。
/	左辺値を右辺値で除算します。
%	左辺値を右辺値で割った余りを扱います。
>>	左辺値を右辺値回右へビットシフトします。
<<	左辺値を右辺値回左へビットシフトします。
&	左辺値と右辺値のビット毎の論理積値を扱います。
|	左辺値と右辺値のビット毎の論理和値を扱います。
^	左辺値と右辺値のビット毎の排他的論理和値を扱います。

- 条件演算子

条件演算子は制御命令".IF", ".ELIF" のオペランドにだけ記述できます。

表 10.7 条件演算子

演算子	機能
>	左辺値が右辺値より大きいことを評価します。
<	左辺値が右辺値より小さいことを評価します。
>=	左辺値が右辺値以上であることを評価します。
<=	左辺値が右辺値以下であることを評価します。
==	左辺値が右辺値と等しいことを評価します。
!=	左辺値が右辺値と等しくないことを評価します。

- 演算優先順位変更演算子

表 10.8 演算優先順位変更演算子

演算子	機能
()	() で囲った演算を最優先で行います。一つの式に複数の() が記述されている場合は、左側が優先されます。 () はネストした記述ができます。

(b) 式の演算優先順位

オペランドに記述されている式について、次に示す優先順位に従い演算を行った結果の数値を値として扱います。

- 演算子がもつ優先順位の高いものから演算します。演算子の優先順位を以下表に示します。
- 同一の優先順位を持つ演算子は、左から順に演算を行います。
- ()で囲ったものが、優先順位が一番高くなります。

表 10.9 式の演算優先順位

優先順位	演算子の種類	演算子
1	演算優先順位変更演算子	()
2	単項演算子	+, -, ~, SIZEOF, TOPOF
3	二項演算子 1	*, /, %
4	二項演算子 2	+, -
5	二項演算子 3	>>, <<
6	二項演算子 4	&
7	二項演算子 5	|, ^
8	条件演算子	>, <, >=, <=, ==, !=

(3) アドレッシングモード

命令のオペランド部に記述できるアドレッシングモードは以下 3 つがあります。

(a) 一般命令アドレッシング

- レジスタ直接

  指定したレジスタが演算の対象となります。R0～R15、SPを記述することができます。SPはR0と解釈します。(R0=SP)

  ```
  Rn (Rn=R0～R15, SP)
  ```

  - 記述例

  ```
  ADD  R1, R2
  ```

- 即値

  #immで示した即値は整数を表します。

  #uimmで示した即値は符号なし整数を表します。

  #simmで示した即値は符号付き整数を表します。

  #imm:n、#uimm:n、および#simm:n は、n ビット長の即値を表します。

  ```
  #imm:8, #uimm:8, #simm:8, #imm:16, #simm:16, #simm:24, #imm:32
  ```

【注】 RTSD **命令の**#uimm:8 **は、確定値でなければなりません。**

  - 記述例

  ```
  MOV.L  #-100, R2                              ; #simm:8
  ```

- レジスタ間接

  レジスタの値が演算対象の実効アドレスになります。実効アドレスの範囲は、00000000h～FFFFFFFFhです。

  ```
  [Rn] (Rn=R0～R15, SP)
  ```

  - 記述例

  ```
  ADD  [R1], R2
  ```

- レジスタ相対

  ディスプレースメント(dsp)を32ビットにゼロ拡張した値と、レジスタ値を加算した結果が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h～FFFFFFFFh です。dsp:n は、nビット長のディスプレースメントを表します。

  dspの値は以下規則によりスケーリングされた値で指定します。アセンブラではスケーリング前の値に戻し、命令のビットパターンに埋め込みます。

表 10.10 dsp 値スケーリング規則

命令	規則
サイズ指定子をとる転送命令	サイズ指定子.B/.W/.L に応じてそれぞれ、1/2/4 倍
サイズ拡張指定子をとる演算命令	サイズ拡張指定子.B/.UB/.W/.UW/.L に応じてそれぞれ、1/1/2/2/4 倍
ビット操作命令	1 倍
上記以外	4 倍

dsp:8[Rn], dsp:16[Rn] (Rn=R0～R15, SP)

- 記述例

```
ADD  400[R1], R2          ; dsp:8[Rn] (400/4 = 100)
```

サイズ指定子 W/L で 2/4 の倍数でない場合、アセンブル時確定値はアセンブラエラー
アセンブル時未確定値はリンク時エラー

(b) 拡張命令アドレッシング

- 短縮即値

#immで示した即値が演算の対象となります。即値がアセンブル時確定値でない場合はエラー処理されます。

#imm:1

このアドレッシングは、DSP機能命令（RACW）のsrcにのみ使用されます。1または2を記述できます。

- 記述例

```
RACW  #1                  ; RACW  #imm:1
```

#imm:2

#immで示した2 ビット即値が演算の対象となります。このアドレッシングは、コプロセッサ命令(MVFCP,MVTCP,OPECP)のコプロセッサ番号指定にのみ使用されます。

- 記述例

```
MVTCP  #3, R1, #4:16        ; MVTCP  #imm:2, Rn, #imm:16
```

#imm:3

#immで示した3 ビット即値が演算の対象となります。このアドレッシングは、ビット操作命令(BCLR,BMCnd,BNOT,BSET,BTST)のビット番号指定に使用されます。

- 記述例

```
BSET   #7, R10              ; BSET   #imm:3, Rn
```

#imm:4

ADD,AND,CMP,MOV,MUL,OR,SUB命令のソースに使用される場合は、#immで示した4 ビット即値を32ビットにゼロ拡張した結果が演算の対象となります。

MVTIPL命令の割り込み優先レベル指定に使用される場合は、#immで示した4ビット即値が演算の対象となります。

- 記述例

```
ADD   #15, R8                ; ADD   #imm:4, Rn
```

#imm:5

#immで示した5 ビット即値が演算の対象となります。このアドレッシングは、ビット操作命令(BCLR, BMCnd, BNOT, BSET, BTST)のビット番号指定、シフト命令(SHAR,SHLL,SHLR)のシフト幅指定、および、ローテート命令(ROTL,ROTR)のローテート幅指定にのみ使用されます。

- 記述例

```
BSET  #31, R10               ; BSET   #imm:5, Rn
```

- 短縮レジスタ相対

5ビットディスプレースメント(dsp)を32ビットにゼロ拡張した値と、レジスタ値を加算した結果が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h～FFFFFFFFh です。
dspの値はサイズ指定子.B/.W/.L に応じて、それぞれ1/2/4倍した値で指定します。dspの値がアセンブル時確定値でない場合、エラー処理をします。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

```
dsp:5[Rn] (Rn=R0～R7, SP)
```

- 記述例

```
MOV.L  R3,124[R1]            ; dsp:5[Rn] (124/4 = 31)
```

※src/dest **のレジスタも** R0**～**R7 **でなくてはなりません。**

- ポストインクリメントレジスタ間接

レジスタの値に、サイズ指定子.B/.W/.L に応じて、それぞれ1/2/4 を加算します。更新前のレジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h～FFFFFFFFh です。このアドレッシングは、MOV,MOVU命令にのみ使用されます。

```
[Rn+] (Rn=R0～R15, SP)
```

- 記述例

```
MOV.L [R3+],R1
```

- プリデクリメントレジスタ間接

  レジスタの値に、サイズ指定子.B/.W/.L に応じて、それぞれ1/2/4 を減算します。更新後のレジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h～FFFFFFFFh です。このアドレッシングは、MOV,MOVU命令にのみ使用されます。

  ```
  [-Rn] (Rn=R0～R15, SP)
  ```

  - 記述例

  ```
  MOV.L [-R3],R1
  ```

- インデックス付きレジスタ間接

  インデックスレジスタ(Ri)の値をサイズ指定子.B/.W/.L に応じて、それぞれ1/2/4 倍した値とベースレジスタ(Rb)の値を加算した結果の下位32 ビットが演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h～FFFFFFFFh です。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

  ```
  [Ri,Rb] (Ri=R0～R15, SP) (Rb=R0～R15, SP)
  ```

  - 記述例

  ```
  MOV.L [R3,R1],R2
  MOV.L R3, [R1,R2]
  ```

(c) 特定命令アドレッシング

- 制御レジスタ直接

  指定した制御レジスタが演算の対象となります。

  このアドレッシングは、MVTC,POPC,PUSHC,MVFC 命令にのみ使用されます。

  ```
  PSW, FPSW, USP, ISP, INTB, BPSW, BPC, FINTV, PC, CPEN
  ```

  - 記述例

  ```
  STC  PSW,R2
  ```

- PSW直接

  指定したフラグ、または、ビットが演算の対象となります。このアドレッシングは、CLRPSW,SETPSW 命令にのみ使用されます。

  ```
  U, I, O, S, Z, C
  ```

  - 記述例

  ```
  CLRPSW  U
  ```

- プログラムカウンタ相対

  分岐命令の分岐先を指定するためのアドレッシング。

  `Rn(Rn=R0～R15, SP)`

  プログラムカウンタの値と、Rnの値を符号付きで加算した結果が実効アドレスとなります。Rnの値の範囲は、-2147483648～2147483647です。実効アドレスの範囲は、00000000h～FFFFFFFFh です。このアドレッシングモードは、BRA(.L)、BSR(.L)命令に使用されます。

  `label(PC + pcdsp:3)`

  分岐命令の分岐先アドレスを表します。指定したシンボル、数値が実効アドレスとなります。

  指定した分岐先アドレスからプログラムカウンタの値を引いたものをディスプレースメント(pcdsp)として命令のビットパターンに埋め込みます。

  分岐距離指定子が“ .S ”の場合、プログラムカウンタの値とディスプレースメントの値を符号なしで加算した結果の下位32 ビットが実効アドレスとなります。

  pcdsp の範囲は、3 ≤ pcdsp:3 ≤ 10です。

  実効アドレスの範囲は、00000000h～FFFFFFFFh です。このアドレッシングは、BRA,BCnd(Cnd==EQ,NE,Z,NZ のみ)のみに使用できます。

  `label(PC + pcdsp:8/pcdsp:16/pcdsp:24)`

  分岐命令の分岐先アドレスを表します。指定したシンボル、数値が実効アドレスとなります。

  指定した分岐先アドレスからプログラムカウンタの値を引いたものをディスプレースメント(pcdsp)として命令のビットパターンに埋め込みます。

  分岐距離指定子が“ .B ”または“ .W ”または“ .A ”の場合、プログラムカウンタの値とディスプレースメントの値を符号付きで加算した結果の下位32 ビットが実効アドレスとなります。

  pcdsp の範囲は以下の通りです。

  “ .B ”の場合 –128 ≤ pcdsp:8 ≤ +127

  “ .W ”の場合 –32768 ≤ pcdsp:16 ≤ +32767

  “ .A ”の場合 –8388608 ≤ pcdsp:24 ≤ +8388607

  実効アドレスの範囲は、00000000h～FFFFFFFFh です。

(4) ビット長指定子

ビット長指定子はオペランドの即値、またはディスプレースメントのサイズを指定します。

- 記述方法

```
:width
```

- 内容

本指定子はオペランドに記述された即値、またはディスプレースメントの直後に指定します。

アセンブラは指定されたビット長のアドレッシングモードを選択します。

本指定子を省略した場合はアセンブラがもっとも効率のよいビット長を選択します。
本指定子が記述されている場合には最適選択は行わず、指定されたビット長とします。

本指定子はアセンブラ制御命令のオペランドには記述できません。

即値、ディスプレースメントの式と本指定子の間には、１つ以上の空白文字を入れることができます。

命令フォーマットに存在しないビット長が指定された場合は、エラー処理されます。

width に指定できるものは次の通りです。

2: 有効ビット長が2ビットであることを表します。

#imm:2

3: 有効ビット長が3ビットであることを表します。

#imm:3

4: 有効ビット長が4ビットであることを表します。

#imm:4

5: 有効ビット長が5ビットであることを表します。

#imm:5, dsp:5

8: 有効ビット長が8ビットであることを表します。

#uimm:8, #simm:8, dsp:8

16: 有効ビット長が16ビットであることを表します。

#uimm:16, #simm:16, dsp:16

24: 有効ビット長が24ビットであることを表します。

#simm:24

32: 有効ビット長が32ビットであることを表します。

#imm:32

(5) サイズ拡張指定子

サイズ拡張指定子は、演算命令でソースがメモリオペランドの場合、メモリオペランドのサイズと拡張方法を指定するために付加されます。

- 記述方法

```
.memex
```

- 内容

本指定子はメモリオペランドの直後に記述し、間に空白文字を入れることはできません。

サイズ拡張指定子は特定の命令と、メモリオペランドの組み合わせに対してのみ有効で、有効でない命令とオペランドの組み合わせで指定した場合は、エラー処理をします。

指定可能な命令とオペランドの組み合わせは、RX ファミリソフトウェアマニュアルの命令フォーマットのオペランドに .memex が付いているパターンのみです。

省略時はビット操作命令では'B'として扱い、それ以外の命令では'L'として扱います。

指定可能なサイズ拡張指定子と効果を、以下表に記します。

表 10.11 サイズ拡張指定子

サイズ拡張指定子	効果
B	8 ビットデータを 32 ビット符号拡張
UB	8 ビットデータを 32 ビットゼロ拡張
W	16 ビットデータを 32 ビット符号拡張
UW	16 ビットデータを 32 ビットゼロ拡張
L	32 ビットデータをロード

(記述例)

```
ADD   [R1].B, R2
AND   125[R1].UB, R2
```

10.1.7 コメントの記述方法

セミコロン(;)の後に続けて記述します。セミコロンから行末までをコメントと見なします。

- 記述例

```
ADD    R1, R2        ; R2 に R1 を加えます。
```

10.2 命令の最適選択

10.2.1 命令フォーマットの最適選択

RX ファミリの命令には、同一処理に対して複数の命令フォーマットを持つものがあります。

アセンブラでは、命令およびアドレッシングモードの指定に応じて、最短コードの命令フォーマットを選択する最適選択を行います。

(1) 即値について

アセンブラではオペランドに即値を持つ命令である場合、オペランドに指定された即値の範囲に従い選択可能なアドレッシングから最適選択を行います。以下に即値の範囲について優先順位の高い順に示します。

表 10.12 即値の範囲

#imm	10 進記法	16 進記法
#imm:1	1～2	1H～2H
#imm:2	0～3	0H～3H
#imm:3	0～7	0H～7H
#imm:4	0～15	0H～0FH
#imm:5	0～31	0H～1FH
#imm:8	-128～255	-80H～0FFH
#uimm:8	0～255	0H～0FFH
#simm:8	-128～127	-80H～7FH
#imm:16	-32768～65535	-8000H～0FFFFH
#simm:16	-32768～32767	-8000H～7FFFH
#simm:24	-8388608～8388607	-800000H～7FFFFFH
#imm:32	-2147483648～4294967295	-80000000H～0FFFFFFFFH

【注】 *1 16 進表記は 32 ビット表記も可能です。
例：10 進表記"-127"、16 進表記"-7FH"は"0FFFFFF81H"と表記できます。

*2 INT 命令の src の#imm の範囲は 0～255 となります。

*3 RTSD 命令の src の#imm の範囲は#uimm:8 を 4 倍した値となります。

(2) ADC, SBB命令

ADC, SBB 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

【注】 最適選択対象とならない命令フォーマットとオペランドは記述していません。表内の処理サイズは、特に明記がない場合は"L"となります。

表 10.13 ADC, SBB 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
ADC src,dest	#simm:8	-	Rd	4
	#simm:16	-	Rd	5
	#simm:24	-	Rd	6
	#imm:32	-	Rd	7
ADC/SBB src,dest	dsp:8[Rs].L	-	Rd	4
	dsp:16[Rs].L	-	Rd	5

SBB 命令では、src に即値を指定することはできません。

(3) ADD命令

ADD 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.14 ADD 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
(1) ADD src,dest	#uimm:4	-	Rd	2
	#simm:8	-	Rd	3
	#simm:16	-	Rd	4
	#simm:24	-	Rd	5
	#imm:32	-	Rd	6
	dsp:8[Rs].memex	-	Rd	3(memex =UB), 4(memex ≠ UB)
	dsp:16[Rs].memex	-	Rd	4(memex =UB), 5(memex ≠ UB)
(2) ADD src,src2,dest	#simm:8	Rs	Rd	3
	#simm:16	Rs	Rd	4
	#simm:24	Rs	Rd	5
	#imm:32	Rs	Rd	6

(4) AND, OR, SUB, MUL命令

AND, OR, SUB, MUL 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.15 AND, OR, SUB および MUL 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
AND/OR/SUB/MUL	#uimm:4	-	Rd	2
src,dest	#simm:8	-	Rd	3
	#simm:16	-	Rd	4
	#simm:24	-	Rd	5
	#imm:32	-	Rd	6
	dsp:8[Rs].memex	-	Rd	3(memex = UB), 4(memex ≠ UB)
	dsp:16[Rs].memex	-	Rd	4(memex = UB), 5(memex ≠ UB)

SUB 命令では、src に#simm:8/16/24, #imm32 を指定することはできません。

(5) BMCnd命令

BMCnd 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.16 BMCnd 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ[バイト]
		src	src2	dest	
BMCnd src,dest	B	#imm:3	-	dsp:8[Rs].B	4
	B	#imm:3	-	dsp:16[Rs].B	5

(6) CMP命令

CMP 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.17　CMP 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ[バイト]
		src	src2	dest	
CMP src,src2	L	#uimm:4	Rd	-	2
	L	#uimm:8	Rd	-	3
	L	#simm:8	Rd	-	3
	L	#simm:16	Rd	-	4
	L	#simm:24	Rd	-	5
	L	#imm:32	Rd	-	6
	L	dsp:8[Rs].memex	Rd	-	3(memex = UB), 4(memex ≠ UB)
	L	dsp:16[Rs].memex	Rd	-	4(memex = UB), 5(memex ≠UB)

(7) DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, TST, XOR命令

DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, MUL, TST, XOR 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.18　DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, TST および XOR 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
DIV/DIVU/	#simm:8	-	Rd	4
EMUL/EMULU/ITOF/	#simm:16	-	Rd	5
MAX/MIN/TST/XOR	#simm:24	-	Rd	6
	#imm:32	-	Rd	7
src,dest	dsp:8[Rs].memex	-	Rd	4(memex=UB), 5(memex ≠ UB)
	dsp:16[Rs].memex	-	Rd	5(memex=UB), 6(memex ≠ UB)

ITOF 命令では、src に#simm:8/16/24, #imm32 を指定することはできません。

(8) FADD, FCMP, FDIV, FMUL, FTOI命令

FADD, FCMP, FDIV, FMUL, FTOI 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.19 FADD, FCMP, FDIV, FMUL および FTOI 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
FADD/FCMP/FDIV/	#imm:32	-	Rd	7
FMUL/FTOI	dsp:8[Rs].L	-	Rd	4
src,dest	dsp:16[Rs].L	-	Rd	5

FTOI 命令では、src に#imm32 を指定することはできません。

(9) MVTC, STNZ, STZ命令

MVTC, STNZ, STZ 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.20 MVTC, STNZ および STZ 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
MVTC/STNZ/STZ	#simm:8	-	Rd	4
src,dest	#simm:16	-	Rd	5
	#simm:24	-	Rd	6
	#imm:32	-	Rd	7

（10） MOV命令

MOV 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.21　MOV 命令の命令フォーマット

命令フォーマット	size	処理サイズ	対象			コードサイズ [バイト]
			src	src2	dest	
MOV(.size) src,dest	B/W/L	size	Rs(Rs=R0-R7)	-	dsp:5[Rd](Rd=R0-R7)	2
	B/W/L	L	dsp:5[Rs](Rs=R0-R7)	-	Rd(Rd=R0-R7)	2
	B/W/L	L	#uimm:8	-	dsp:5[Rd](Rd=R0-R7)	3
	L	L	#uimm:4	-	Rd	2
	L	L	#uimm:8	-	Rd	3
	L	L	#simm:8	-	Rd	3
	L	L	#simm:16	-	Rd	4
	L	L	#simm:24	-	Rd	5
	L	L	#imm:32	-	Rd	6
	B	B	#imm:8	-	[Rd]	3
	W/L	W/L	#simm:8	-	[Rd]	3
	W	W	#imm:16	-	[Rd]	4
	L	L	#simm:16	-	[Rd]	4
	L	L	#simm:24	-	[Rd]	5
	L	L	#imm:32	-	[Rd]	6
	B	B	#imm:8	-	dsp:8[Rd]	4
	W/L	W/L	#simm:8	-	dsp:8[Rd]	4
	W	W	#imm:16	-	dsp:8[Rd]	5
	L	L	#simm:16	-	dsp:8[Rd]	5
	L	L	#simm:24	-	dsp:8[Rd]	6
	L	L	#imm:32	-	dsp:8[Rd]	7
	B	B	#imm:8	-	dsp:16[Rd]	5
	W/L	W/L	#simm:8	-	dsp:16[Rd]	5
	W	W	#imm:16	-	dsp:16[Rd]	6
	L	L	#simm:16	-	dsp:16[Rd]	6
	L	L	#simm:24	-	dsp:16[Rd]	7
	L	L	#imm:32	-	dsp:16[Rd]	8
	B/W/L	L	dsp:8[Rs]	-	Rd	3
	B/W/L	L	dsp:16[Rs]	-	Rd	4
	B/W/L	size	Rs	-	dsp:8[Rd]	3
	B/W/L	size	Rs	-	dsp:16[Rd]	4

命令フォーマット	size	処理サイズ	対象			コードサイズ[バイト]
			src	src2	dest	
MOV(.size) src,dest	B/W/L	size	[Rs]	-	dsp:8[Rd]	3
	B/W/L	size	[Rs]	-	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	-	[Rd]	3
	B/W/L	size	dsp:16[Rs]	-	[Rd]	4
	B/W/L	size	dsp:8[Rs]	-	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	-	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	-	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	-	dsp:16[Rd]	6

（11） MOVU命令

MOVU 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.22　MOVU 命令の命令フォーマット

命令フォーマット	size	処理サイズ	対象			コードサイズ[バイト]
			src	src2	dest	
MOVU(.size) src,dest	B/W	L	dsp:5[Rs](Rs=R0-R7)	-	Rd(Rd=R0-R7)	2
	B/W	L	dsp:8[Rs]	-	Rd	3
	B/W	L	dsp:16[Rs]	-	Rd	4

（12） PUSH命令

PUSH 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.23　PUSH 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
PUSH src	dsp:8[Rs]	-	-	3
	dsp:16[Rs]	-	-	4

（13） ROUND命令

ROUND 命令に対して､最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.24　ROUND 命令の命令フォーマット

命令フォーマット	対象			コードサイズ[バイト]
	src	src2	dest	
ROUND src,dest	dsp:8[Rs]	-	Rd	4
	dsp:16[Rs]	-	Rd	5

（14） SCCnd命令

SCCnd 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.25　SCCnd 命令の命令フォーマット

命令フォーマット	size	対象			コードサイズ[バイト]
		src	src2	dest	
SCCnd(.size) src,dest	B/W/L	-	-	dsp:8[Rd]	4
	B/W/L	-	-	dsp:16[Rd]	5

（15） XCHG命令

XCHG 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.26　XCHG 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ[バイト]
		src	src2	dest	
XCHG src,dest	L	dsp:8[Rs].memex	-	Rd	4(memex = UB), 5(memex ≠ UB)
	L	dsp:16[Rs].memex	-	Rd	5(memex = UB), 6(memex ≠ UB)

（16） BCLR, BNOT, BSET, BTST命令

BCLR, BNOT, BSET, BTST 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 10.27　BCLR, BNOT, BSET および BTST 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ[バイト]
		src	src2	dest	
BCLR/BNOT/BSET/BTST	B	#imm:3	-	dsp:8[Rd].B	3
src,dest	B	#imm:3	-	dsp:16[Rd].B	4
	B	Rs	-	dsp:8[Rd].B	4
	B	Rs	-	dsp:16[Rd].B	5

10.2.2 分岐命令の最適選択

(1) 相対無条件分岐命令(BRA)の最適選択

(a) 指定可能な分岐距離指定子

.S	3ビットPC相対(PC+pcdsp:3,3 ≤ pcdsp:3 ≤ 10)
.B	8ビットPC相対(PC+pcdsp:8,-128 ≤ pcdsp:8 ≤ 127)
.W	16ビットPC相対(PC+pcdsp:16,-32768 ≤ pcdsp:16 ≤ 32767)
.A	24ビットPC相対(PC+pcdsp:24,-8388608 ≤ pcdsp:24 ≤ 8388607)
.L	レジスタ相対(PC+Rs,-2147483648 ≤ Rs ≤ 2147483647)

※レジスタ相対はオペランドがレジスタの場合のみ選択され、最適選択によって選択されることはありません。

(b) 最適選択

- アセンブラでは相対無条件分岐命令のオペランドが分岐最適化対象条件を満たす場合、最短の分岐距離を選択します。条件については、**「10.1.5 (3) 分岐距離指定子」**を参照してください。
- 条件を満たさないものについては、24ビットPC相対(.A)を選択します。

(2) 相対サブルーチン分岐命令(BSR)の最適選択

(a) 指定可能な分岐距離指定子

.W	16ビットPC相対(PC+pcdsp:16,-32768 ≤ pcdsp:16 ≤ 32767)
.A	24ビットPC相対(PC+pcdsp:24,-8388608 ≤ pcdsp:24 ≤ 8388607)
.L	レジスタ相対(PC+Rs,-2147483648 ≤ Rs ≤ 2147483647)

※レジスタ相対はオペランドがレジスタの場合のみ選択され、最適選択によって選択されることはありません。

(b) 最適選択

- アセンブラでは相対サブルーチン分岐命令のオペランドが分岐最適化対象条件を満たす場合、最短の分岐距離を選択します。条件については、**「10.1.5 (3) 分岐距離指定子」**を参照してください。
- 条件を満たさないものについては、24ビットPC相対(.A)を選択します。

(3) 条件分岐命令(BCnd)の最適選択

(a) 指定可能な分岐距離指定子

BEQ.S	3ビットPC相対(PC+pcdsp:3,3 ≤ pcdsp:3 ≤ 10)
BNE.S	3ビットPC相対(PC+pcdsp:3,3 ≤ pcdsp:3 ≤ 10)
BCnd.B	8ビットPC相対(PC+pcdsp:8,-128 ≤ pcdsp:8 ≤ 127)
BEQ.W	16ビットPC相対(PC+pcdsp:16,-32768 ≤ pcdsp:16 ≤ 32767)
BNE.W	16ビットPC相対(PC+pcdsp:16,-32768 ≤ pcdsp:16 ≤ 32767)

(b) 最適選択

- アセンブラでは条件分岐命令のオペランドが分岐最適化対象条件を満たす場合、論理を反転した条件分岐命令と最適な分岐距離の相対無条件分岐命令を組み合わせた最適な条件分岐コードを選択して生成します。
- 条件を満たさないものについては、8ビットPC相対(.B) または16ビットPC相対(.W)を選択します。

(c) 変換される条件分岐命令に対する代替命令

表 10.28 条件分岐命令の代替規則

条件分岐命令	代替分岐命令	条件分岐命令	代替分岐命令
BNC/BLTU dest	BC ..xx BRA.A dest ..xx:	BC/BGEU dest	BNC ..xx BRA.A dest ..xx:
BLEU dest	BGTU ..xx BRA.A dest ..xx:	BGTU dest	BLEU ..xx BRA.A dest ..xx:
BNZ/BNE dest	BZ ..xx BRA.A dest ..xx:	BZ/BEQ dest	BNZ ..xx BRA.A dest ..xx:
BPZ dest	BN ..xx BRA.A dest ..xx:	BO dest	BNO ..xx BRA.A dest ..xx:
BGT dest	BLE ..xx BRA.A dest ..xx:	BLE dest	BGT ..xx BRA.A dest ..xx:
BGE dest	BLT ..xx BRA.A dest ..xx:	BLT dest	BGE ..xx BRA.A dest ..xx:

上記は相対無条件分岐命令の分岐距離が 24 ビット PC 相対の場合を記します。

"..xx"ラベルおよび相対無条件分岐命令は内部的に処理されるものであり、アセンブルリストファイルにはコードのみ生成されます。

10.3 アセンブラ制御命令の記述方法

アセンブラ制御命令には、一般のアセンブラ制御命令(以降、単にアセンブラ制御命令と呼びます)と高級言語用アセンブラ制御命令が存在します。

10.3.1 アドレス制御命令

アセンブラがアドレス更新をする場合の指示を行う制御命令です。

絶対アドレス形式セクション内のアドレスを除いて、アセンブラが制御を行うアドレスはリロケータブル値となります。

表 10.29 アドレス制御命令

制御命令	機能内容
.ORG	開始アドレスを宣言します。 本制御命令を記述したセクションは、絶対アドレス形式セクションとなります。
.OFFSET	セクション先頭からのオフセットを指定します。 本制御命令は相対アドレス形式セクションでのみ記述できます。
.ENDIAN	セクションのエンディアンを指定します。
.BLKB	1 バイト単位で RAM 領域を確保します。
.BLKW	2 バイト単位で RAM 領域を確保します。
.BLKL	4 バイト単位で RAM 領域を確保します。
.BLKD	8 バイト単位で RAM 領域を確保します。
.BYTE	1 バイト長のデータを ROM 領域に格納します。
.WORD	2 バイト長のデータを ROM 領域に格納します。
.LWORD	4 バイト長のデータを ROM 領域に格納します。
.FLOAT	4 バイトで表される浮動小数点数データを ROM 領域に格納します。
.DOUBLE	8 バイトで表される浮動小数点数データを ROM 領域に格納します。
.ALIGN	ロケーションカウンタを境界調整数の倍数に補正します。

アドレス宣言

.ORG

書　式　　.ORG△<数値>

説　明　　本制御命令を記述したセクションを絶対アドレス形式とします。

本制御命令を記述したセクションのアドレスはアブソリュート値になります。

本制御命令を記述した直後の行から記述したニーモニックのコードが格納されるアドレスを決定します。

本制御命令の直後の行から記述した領域確保制御命令で、確保されるメモリのアドレスを決定します。

例

```
.SECTION        value,ROMDATA
.ORG            0FF00H
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.ORG            0FF80H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

以下の場合は.SECTION の直後に.ORG が記述されていないため、エラーとなります。

```
.SECTION        value,ROMDATA
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.ORG            0FF80H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

備　考　　本制御命令は、必ずセクション制御命令の直後に記述してください。

".SECTION"を記述した直後の行に".ORG"の記述がない場合は、そのセクションは相対アドレス形式セクションとなります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドに記述できる値は、0～0FFFFFFFFH の範囲の数値です。

オペランドには式、シンボルを記述できます。ただし、アセンブル時に確定する値でなければなりません。

本制御命令は、相対アドレス形式指定を行ったセクション内には記述できません。

絶対アドレス形式セクション内であれば複数回記述できます。ただし本制御命令記述行のアドレスよりも小さい値を指定した場合にはエラーとなります。

.OFFSET

書　式　　.OFFSET△<数値>

説　明　　セクション先頭からのオフセットを指定します。

本制御命令を記述した直後の行から記述したニーモニックのコードが格納される、セクション先頭からのオフセットを決定します。

本制御命令の直後の行から記述した領域確保制御命令で確保されるメモリの、セクション先頭からのオフセットを決定します。

例

```
.SECTION        value,ROMDATA
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.OFFSET         80H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

以下の場合は.OFFSET記述行のオフセットよりも小さい値が指定されているため、エラーとなります。

```
.SECTION        value,ROMDATA
.OFFSET         80H
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.OFFSET         70H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

備　考　　制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドに記述できる値は、0～0FFFFFFFFH の範囲の数値です。

オペランドには式、シンボルを記述できます。ただし、アセンブル時に確定する値でなければなりません。

本制御命令は、絶対アドレス形式指定を行ったセクション内には記述できません。

相対アドレス形式セクション内であれば複数回記述できます。ただし本制御命令記述行のオフセットよりも小さい値を指定した場合にはエラーとなります。

.ENDIAN

書　式　　.ENDIAN△BIG

.ENDIAN△LITTLE

説　明　　本制御命令を記述したセクションのエンディアンを指定します。

.ENDIAN BIG を記述したセクションのデータのバイト並びは Big Endian になります。

.ENDIAN LITTLE を記述したセクションのデータのバイト並びは Little Endian になります。

本制御命令が記述されていないセクションのデータのバイト並びは、-endian オプションに依存します。

例　　
```
.SECTION        value,ROMDATA
.ORG            0FF00H
.ENDIAN         BIG
.BYTE           "abcdefghijklmnopqrstuvwxyz"
```

以下の場合は.SECTION または.ORG の直後に.ENDIAN が記述されていないため、エラーとなります。

```
.SECTION        value,ROMDATA
.ORG            0FF00H
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.ENDIAN         BIG
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

備　考　　本制御命令は必ず.SECTION 制御命令、またはそれに続く.ORG 制御命令の直後に記述してください。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

セクション属性が CODE のセクションに本制御命令を追加することはできません。

1 バイト長の領域を確保

.BLKB

書　式　△.BLKB△<オペランド>

△<ラベル名:>△.BLKB△<オペランド>

説　明　1 バイト単位で、指定したバイト数の RAM 領域を確保します。

確保した RAM のアドレスに、ラベル名を定義することもできます。

例

```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKB 1
work2:          .BLKB symbol
                .BLKB symbol+1
```

備　考　本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",DATA"を記述することでセクション属性が DATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドの値は、アセンブル時に確定しなければなりません。

領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。

ラベル名には、必ずコロン(:)を記述してください。

2 バイト長の領域を確保

.BLKW

書　式　△.BLKW△<オペランド>

△<ラベル名:>△.BLKW△<オペランド>

説　明　2 バイト単位で、指定した個数の RAM 領域を確保します。

確保した RAM のアドレスに、ラベル名を定義することもできます。

例

```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKW 1
work2:          .BLKW symbol
                .BLKW symbol+1
```

備　考　本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",DATA"を記述することでセクション属性が DATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドの値は、アセンブル時に確定しなければなりません。

領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。

ラベル名には、必ずコロン(:)を記述してください。

4 バイト長の領域を確保

.BLKL

書　式　△.BLKL△<オペランド>

△<ラベル名:>△.BLKL△<オペランド>

説　明　4 バイト単位で、指定した個数の RAM 領域を確保します。

確保した RAM のアドレスに、ラベル名を定義することもできます。

例

```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKL 1
work2:          .BLKL symbol
                .BLKL symbol+1
```

備　考　本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",DATA"を記述することでセクション属性が DATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドの値は、アセンブル時に確定しなければなりません。

領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。

ラベル名には、必ずコロン(:)を記述してください。

8 バイト長の領域を確保

.BLKD

書　式　△.BLKD△<オペランド>

△<ラベル名:>△.BLKD△<オペランド>

説　明　8 バイト単位で、指定した個数の RAM 領域を確保します。

確保した RAM のアドレスに、ラベル名を定義することもできます。

例

```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKD 1
work2:          .BLKD symbol
                .BLKD symbol+1
```

備　考　本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",DATA"を記述することでセクション属性が DATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドの値は、アセンブル時に確定しなければなりません。

領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。

ラベル名には、必ずコロン(:)を記述してください。

1 バイト長のデータを格納

.BYTE

書　式　△.BYTE△<オペランド>

△<ラベル名:>△.BYTE△<オペランド>

説　明　1 バイト長の固定データを ROM に格納します。

データを格納したアドレスに、ラベル名を定義することもできます。

例　<endian=little オプション指定時>

```
.SECTION value,ROMDATA
.BYTE 1
.BYTE "data"
.BYTE symbol
.BYTE symbol+1
.BYTE 1,2,3,4,5
.END
```

<endian=big オプション指定時>

```
.SECTION  program,CODE,ALIGN=4
MOV.L  R1,R2
.ALIGN 4
.BYTE 080H,00H,00H,00H
.END
```

備　考　本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",ROMDATA"を記述することでセクション属性が ROMDATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドに文字・文字列を記述するときは、クォーテーション(‘)またはダブルクォーテーション(“)で囲ってください。このとき格納されるデータは、文字の ASCII コードになります。

ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン(:)を記述してください。

endian=big オプション指定時、本制御命令は次の条件に当てはまるセクション内にのみ記述できます。条件に当てはまらないセクション内に記述した場合はエラーとなります。

(1) ROMDATA セクション

```
.SECTION data,ROMDATA
```

(2) セクション定義の際のアドレス補正に 4、もしくは 8 を指示している相対アドレス形式の CODE セクション

```
.SECTION program,CODE,ALIGN=4
```

(3) 絶対アドレス形式の CODE セクション

```
.SECTION program,CODE
.ORG 0fff00000H
```

endian=big オプション指定時、本制御命令をセクション属性が CODE のセクションに記述する場合、直前の行にアドレス補正制御命令(.ALIGN 4)を記述し、4 バイト境界に配置されるようにしてください。記述されていない場合、アセンブラはウォーニングを出力し、自動的に 4 バイト境界に配置します。

2 バイト長のデータを格納

.WORD

書　式　△.WORD△<オペランド>

△<ラベル名:>△.WORD△<オペランド>

説　明　2 バイト長の固定データを ROM に格納します。

データを格納したアドレスに、ラベル名を定義することもできます。

例

```
.SECTION value,ROMDATA
.WORD             1
.WORD             symbol
.WORD             symbol+1
.WORD             1,2,3,4,5
.END
```

備　考　本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",ROMDATA"を記述することでセクション属性が ROMDATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドに文字・文字列を記述することはできません。

ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン(:)を記述してください。

4 バイト長のデータを格納

.LWORD

書　式　△.LWORD△<オペランド>

△<ラベル名:>△.LWORD△<オペランド>

説　明　4 バイト長の固定データを ROM に格納します。

データを格納したアドレスに、ラベル名を定義することもできます。

例

```
.SECTION value,ROMDATA
.LWORD          1
.LWORD          symbol
.LWORD          symbol+1
.LWORD          1,2,3,4,5
.END
```

備　考　本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",ROMDATA"を記述することでセクション属性が ROMDATA となります。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドには数値、シンボル、式を記述できます。

オペランドに文字・文字列を記述することはできません。

ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン(:)を記述してください。

4 バイト長のデータを格納

.FLOAT

書　式　　△.FLOAT△<数値>

△<ラベル名:>△.FLOAT△<数値>

説　明　　4 バイト長の固定データを ROM に格納します。

データを格納したアドレスに、ラベル名を定義することもできます。

例

```
                    .FLOAT  5E2
constant:           .FLOAT  5e2
```

備　考　　本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",ROMDATA"を記述することでセクション属性が ROMDATA となります。

オペランドに浮動小数点数を記述してください。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン(:)を記述してください。

8 バイト長のデータを格納

.DOUBLE

書　式　　△.DOUBLE△<数値>

△<ラベル名:>△.DOUBLE△<数値>

説　明　　8 バイト長の固定データを ROM に格納します。

データを格納したアドレスに、ラベル名を定義することもできます。

例

```
                    .DOUBLE  5E2
constant:           .DOUBLE  5e2
```

備　考　　本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて",ROMDATA"を記述することでセクション属性が ROMDATA となります。

オペランドに浮動小数点数を記述してください。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン(:)を記述してください。

.ALIGN

書　式　　△.ALIGN△<アライメント補正値>

<アライメント補正値>:[2|4|8]

説　明　　本制御命令を記述した直後の行のコードを格納するアドレスを 2、4 または 8 バイトアライメントに補正します。

セクション属性が CODE または、ROMDATA の場合は、アドレスを補正した結果、空になったところに NOP のコード（03H）を書き込みます。

セクション属性が DATA の場合は、アドレス補正のみ行います。

例

```
.SECTION program,CODE,ALIGN=4
MOV.L  R1, R2
.ALIGN  4          ; アドレスを 4 の倍数に補正
RTS
.END
```

備　考　　本制御命令は、次の条件に当てはまるセクション内に記述できます。

(1) セクション定義の際にアドレス補正を指示している相対アドレス形式セクション

```
.SECTION program,CODE,ALIGN=4
```

(2) 絶対アドレス形式セクション

```
.SECTION program,CODE
.ORG 0fff00000H
```

相対アドレス形式のセクションで.SECTION 制御命令行で ALIGN 指定のされていないセクションに本制御命令を記述した場合は、ウォーニングが出力されます。

指定した値がセクションの境界調整数よりも大きい場合は、ウォーニングが出力されます。

10.3.2 アセンブラ制御命令

制御命令自身はデータを生成しません。命令に対する機械語コードの生成を制御する制御命令です。アドレスの更新は行いません。

表 10.30 アセンブラ制御命令

制御命令	機能内容
.EQU	シンボルを設定します。
.END	アセンブリ言語ファイルの終了を指定します。
.INCLUDE	本制御命令を記述した位置に、指定したファイルの内容を読み込みます。

数値シンボル定義

.EQU

書　式　　<名前>△.EQU△<数値>

説　明　　シンボルに 32 ビット符号付き整数値(-2147483648～2147483647)の範囲の値を定義します。

本制御命令でシンボルを定義することにより、シンボリックデバッグ機能が使用できます。

例

```
symbol  .EQU 1
symbol1 .EQU symbol+symbol
symbol2 .EQU 2
```

備　考　　シンボルに定義できる値は、アセンブル時に確定しなければなりません。

制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

シンボル定義のオペランドには、シンボルを記述できます。ただし、前方参照となるシンボル名は記述できません。

オペランドには式を記述できます。

シンボルはグローバル指定ができます。

本制御命令と.DEFINE 制御命令で同名のシンボルを宣言した場合、先に記述した方が優先されます。

アセンブリ言語ファイル終了宣言

.END

書　式　　.END

説　明　　アセンブリ言語ファイルの終了を宣言します。

本制御命令を記述した行以降の記述内容はアセンブルリストファイルに出力するのみで、コード生成などの処理は行いません。

例　　.END

備　考　　本制御命令は、１つのアセンブリ言語ファイルに必ず１つ記述する必要があります。

インクルードファイル指定

.INCLUDE

書　式　　.INCLUDE△<インクルードファイル名>

説　明　　アセンブリ言語ファイルの行に、インクルードファイルの内容全てを読み込みます。

本制御命令で読み込まれたインクルードファイルの内容は、読み込んだアセンブリ言語ファイル内に記述した場合と、同じ１つのアセンブリ言語ファイルとして処理されます。

インクルードファイルは 30 レベルまでネストできます。

インクルードファイル名に絶対パスを記述した場合は、記述したディレクトリ内のファイルを検索します。

ファイルが見つからない場合はエラーとなります。

インクルードファイル名に絶対パスを記述していない場合は、次に示す順序でファイルを検索します。

(1) アセンブラ起動時にコマンド行で指定したアセンブリ言語ファイル名にディレクトリ指定がない場合は、.INCLUDE 制御命令で指定されたインクルードファイル名を検索します。アセンブラ起動時にコマンド行で指定したアセンブリ言語ファイル名にディレクトリ指定がある場合は、.INCLUDE 制御命令で指定されたインクルードファイル名にコマンド行で指定されたディレクトリ名を付加して検索します。

(2) アセンブラオプション-include で指定されたディレクトリを検索します。

(3) 環境変数 INC_RXA に設定されているディレクトリを検索します。

例　　.INCLUDE initial.src

.INCLUDE ..FILE@.inc

備　考　　制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドのインクルードファイル名には、必ずファイル拡張子を記述してください。

オペランドには、制御命令"..FILE"や"@"を含む文字列が記述できます。

ファイル名は、その先頭を除き、空白文字を含むことができます。

ファイル名をダブルクォーテーション「"」で囲わないでください。
自分自身をインクルードファイルに指定することはできません。

10.3.3 リンク制御命令

プログラムを複数のファイルに分割して記述するリロケータブルアセンブルを実行するための制御命令です。

表 10.31 リンク制御命令

制御命令	機能内容
.SECTION	アドレスを再配置するための最小の単位となるセクションを定義します。
.GLB	シンボルが外部シンボルであることを宣言します。
.RVECTOR	シンボルを可変ベクタに登録します。

.SECTION

書　式　.SECTION△<セクション名>

.SECTION△<セクション名>,<セクション属性>

.SECTION△<セクション名>,<セクション属性>,ALIGN=[2|4|8]

.SECTION△<セクション名>,ALIGN=[2|4|8]

<セクション属性>:[CODE|ROMDATA|DATA]

説　明　セクションの宣言、再開を指定します。

(1) セクションの宣言

セクション名、セクション属性を指定し、セクションの始まりを定義します。

(2) セクションの再開

ソースプログラム中にすでに存在しているセクションを再開します。セクションの再開ではすでに存在するセクション名を指定します。セクション属性とアライメント補正値は最初に宣言したものを継続します。

'ALIGN='指定がある場合、指定されたセクションに対してアライメント補正値を変更することができます。

ALIGN 指定をした相対アドレス形式セクションまたは絶対アドレス形式セクションに、制御命令".ALIGN"が記述できます。

ALIGN 指定がない場合、そのセクションの境界調整数は 1 となります。

例

```
.SECTION        program,CODE
NOP
.SECTION        ram,DATA
.BLKB           10
.SECTION        tbl1,ROMDATA
.BYTE           "abcd"
.SECTION        tbl2,ROMDATA,ALIGN=8
.LWORD          11111111H,22222222H
.END
```

備　考　セクション名は必ず記述してください。

メモリ領域を確保したりメモリにデータを格納するアセンブラ制御命令を記述する場合は、必ず本制御命令でセクションを定義してください。

ニーモニックを記述する場合は必ず、本制御命令でセクションを定義してください。

セクション属性と ALIGN は、セクション名の後に記述してください。

セクション属性および、ALIGN 指定をする場合は、カンマで区切って記述してください。

セクション属性と ALIGN の記述順序は任意です。

セクション属性は、'CODE','ROMDATA','DATA'のいずれかを記述できます。

セクション属性は省略できます。このとき、アセンブラはセクション属性を CODE として処理

します。

注意事項　-endian=big 指定時、絶対アドレス形式の CODE セクションの開始アドレスには 4 の倍数以外の値を指定することはできません。

-endian=big 指定時、絶対アドレス形式の CODE セクションはウォーニングを出力し、セクションサイズが 4 の倍数になるようにアセンブラがセクション末尾に NOP(0x03)を書き込みます。

定義されているセクション名と同じ名前のシンボルを定義することはできません。セクションとシンボルを同じ名前で定義した場合、後で定義したものは A2118 エラーとなります。

$iop という名前のセクション名を定義することはできません。$iop を定義した場合 A2049 エラーとなります。

グローバル宣言

.GLB

書　式　.GLB△<名前>

.GLB△<名前>[,<名前> …]

説　明　本制御命令で指定したラベルおよびシンボルがグローバルであることを宣言します。

本制御命令で指定したラベルおよびシンボルで、ファイル内で定義されていないものは、外部のファイルで定義されているものとして処理します。

本制御命令で指定したラベルおよび、シンボルで、ファイル内で定義されているものは、外部から参照できるように処理します。

例

```
.GLB    name1,name2,name3
.GLB    name4
.SECTION program
MOV.L   #name1,R1
```

備　考　制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

オペランドにグローバルラベルとするラベル名を記述します。

オペランドにグローバルシンボルとするシンボル名を記述します。

オペランドに複数のシンボル名を記述する場合は、カンマ(,)で区切って記述してください。

.RVECTOR

書　式　　.RVECTOR△<番号>,<名前>

説　明　　本制御命令で指定したラベルおよびシンボルを、可変ベクタとして登録します。

本制御命令の<番号>には、ベクタ番号として 0～255 の定数を記述することができます。

本制御命令の<名前>には、ファイル内で定義されたラベルまたはシンボルを指定することができます。

登録された可変ベクタは、最適化リンケージエディタにより、ひとつの C$VECT セクションにまとめられます。

例

```
    .RVECTOR   50,_rvfunc
_rvfunc:
    MOV.L  #0,R1
    RTE
```

備　考　　制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

10.3.4 アセンブルリスト制御命令

アセンブルリストファイルに出力する情報や、アセンブルリストファイルのフォーマットの制御を行う制御命令です。なお、コード生成には影響を与えません。

表 10.32 アセンブルリスト制御命令

制御命令	機能内容
.LIST	アセンブルリストファイルを生成する際に、アセンブリ言語ファイルの行単位でアセンブルリストファイルへの出力を行うか行わないかを制御します。

アセンブルリスト出力制御命令

.LIST

書　式　　.LIST△[ON|OFF]

説　明　　アセンブルリストファイルへの行の出力を停止(OFF)することができます。
リストへの行の出力を停止している範囲においても、エラー発生行についてはアセンブルリストファイルに出力します。
アセンブルリストファイルへの行の出力を開始(ON)することができます。
本制御命令を指定しない場合は、全ての行をアセンブルリストファイルに出力します。

例　　.LIST ON
.LIST OFF

備　考　　制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
行の出力を停止する場合は、オペランドに'OFF'を記述してください。
行の出力を開始する場合は、オペランドに'ON'を記述してください。

10.3.5 条件アセンブル制御命令

条件アセンブル制御命令を使って、指定した範囲の行のアセンブルを行うか、行わないかを指定できます。

表 10.33 条件アセンブル制御命令

制御命令	機能内容
.IF	条件アセンブルブロックの始まりを示します。条件の判定を行います。
.ELIF	二つ以上の条件ブロックを記述する場合に、二つ目以降の条件を判定します。
.ELSE	全ての条件が偽である場合に、アセンブルを行うブロックの始まりを示します。
.ENDIF	条件アセンブルブロックの終了を示します。

条件アセンブル命令

.IF, .ELIF, .ELSE, .ENDIF

書　式

```
.IF△条件式
ボディ
.ELIF△条件式
ボディ
.ELSE
ボディ
.ENDIF
```

説　明

.IF, .ELIF に記述した条件に従いアセンブルを行うブロックを制御します。

.IF, .ELIF のオペランドに記述した条件を判定し、真であれば以降に続くボディをアセンブルします。

条件が真である場合にアセンブルされる行は、制御命令".ELIF",".ELSE"および".ENDIF"行の前までです。

条件アセンブルブロック内には、アセンブリ言語ファイルに記述可能な全ての命令を記述できます。

条件式の結果によって、条件アセンブルが行われます。

例 ＜条件式の記述例＞

```
sym < 1
sym+2 < data1
sym+2 < data1+2
'smp1' == name
```

＜条件アセンブル記述例＞

```
.IF               TYPE==0
.byte             "Proto Type Mode"
.ELIF             TYPE>0
.byte             "Mass Produciton Mode"
.ELSE
.byte             "Debug Mode"
.ENDIF
```

備 考 .IF, .ELIF 制御命令のオペランドには、必ず条件式を記述してください。

.IF, .ELIF 制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

条件式は、制御命令のオペランドに 1 つだけ記述できます。

条件式には、必ず条件演算子を記述してください。

次に示す演算子が記述できます。

表 10.34 .IF および.ELIF 制御命令の条件演算子

条件演算子	内容
>	左辺値が右辺値より大きい場合に真となります
<	左辺値が右辺値より小さい場合に真となります
>=	左辺値が右辺値より大きいか等しい場合に真となります
<=	左辺値が右辺値より小さいか等しい場合に真となります
==	左辺値と右辺値が等しい場合に真となります
!=	左辺値と右辺値が等しくない場合に真となります

条件式は符号付き 32 ビットで演算します。

条件演算子の左辺および右辺には、シンボルが記述できます。

条件演算子の左辺および右辺には、式が記述できます。式は、**「10.1.6 オペランド部の記述方法」**の**「(2) 式」**に従って記述してください。

条件演算子の左辺および右辺には、文字列が記述できます。文字列は、必ずシングルクォーテーション(')またはダブルクォーテーション(")で囲って記述してください。このとき、文字列の大小は、文字コードの値で判定されます。

例)

"ABC"<"CBA" -> 414243 < 434241 で真となります。

"C" < "A" -> 43 < 41 で偽となります。

条件演算子の前後には、空白文字またはタブが記述できます。

条件式は、制御命令".IF"および".ELIF"のオペランドに記述できます。

演算結果のオーバフローは判断しません。

シンボルは、前方参照（本制御命令行より後に定義されているシンボルを参照）はできません。

前方参照のシンボルや、未定義のシンボルを記述した場合は、値を 0 として式を判定します。

10.3.6 拡張機能制御命令

コード生成には影響を与えない制御命令です。

表 10.35 拡張機能制御命令

制御命令	機能内容
.ASSERT	オペランドに記述した文字列を標準エラー出力またはファイルに出力します。
?	テンポラリラベルの定義と参照を指定します。
@	@の前後の文字列を連結し、1 つの文字列として扱います。
..FILE	アセンブラが処理を行っているアセンブリ言語ファイル名を示します。
.STACK	指定したシンボルに対してスタック値を定義します。
.LINE	行番号を変更します。
.DEFINE	置き換えシンボルを定義します。

指定文字列を出力

.ASSERT

書　式　　.ASSERT△“<文字列>”

.ASSERT△“<文字列>”>△<ファイル名>

.ASSERT△“<文字列>”>>△<ファイル名>

説　明　　オペランドに記述した文字列をアセンブル時に、標準エラー出力に出力します。

ファイル名を指定した場合は、オペランドに記述した文字列をファイルに出力します。

ファイル名に絶対パスを記述した場合は、記述したディレクトリにファイルを生成します。

ファイル名に絶対パスを記述していない場合

(1) output オプションで指定したファイル名にディレクトリ指定がない場合は、本制御命令で指定されたファイルをカレントディレクトリに生成します。

(2) output オプションで指定したファイル名にディレクトリ指定がある場合は、本制御命令で指定されたファイル名に output オプションで指定されたファイルのディレクトリを付加したファイルを生成します。

(3) output オプションが指定されていない場合、アセンブラ起動時にコマンド行で指定したファイルと同じディレクトリにファイルを生成します。

ファイル名に制御命令”..FILE”を記述した場合は、アセンブラ起動時にコマンド行で指定したファイルと同じディレクトリにファイルを生成します。

例　　sample.dat ファイルにメッセージを出力します。

```
.ASSERT "string" > sample.dat
```

sample.dat ファイルにメッセージを追加します。

```
.ASSERT "string" >> sample.dat
```

現在処理中のファイルと同じ名前で拡張子を除くファイル名のファイルにメッセージを出力します。

```
.ASSERT "string" > ..FILE
```

備　考　　オペランドと制御命令の間には、必ず空白文字またはタブを記述してください。

オペランドの文字列は必ずダブルクォーテーションで囲ってください。

文字列をファイルに出力するときは、">"または">>"に続けてファイル名を指定してください。

>は、新規にファイルを生成して、そのファイルにメッセージを出力します。以前に同一名のファイルがある場合は、そのファイルに上書きされます。

>>は、ファイルの内容に追加して、メッセージを出力します。指定したファイルが存在しない場合は、新しくファイルを生成します。

">"または">>"の前後には、空白文字またはタブを記述できます。

ファイル名に制御命令"..FILE"を記述できます。

テンポラリラベル

?

書　式　?:

△<ニーモニック>△?+

△<ニーモニック>△?-

説　明　テンポラリラベルを定義します。

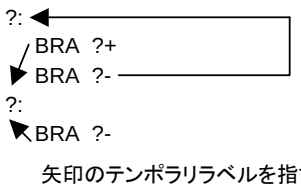
直前または直後に定義されたテンポラリラベルを参照します。

同一ファイル内で定義および参照が可能です。

ファイル内に 65535 個までのテンポラリラベルが定義できます。このとき、ファイル内に ”.INCLUDE ”が記述されている場合は、インクルードファイル内のテンポラリラベルを含み 65535 個までの記述ができます。

アセンブルリストファイルには、テンポラリラベルとして変換された結果が出力されます。

例



備　考　テンポラリラベルとして定義したい行に“?:”を記述してください。

直前に定義したテンポラリラベルを参照したい場合は、命令のオペランドに“?-”を記述してください。

直後に定義したテンポラリラベルを参照したい場合は、命令のオペランドに“?+”を記述してください。

参照できるラベルは、直前と直後のラベルだけです。

文字列の連結

@

書　式　　<文字列>@<文字列>[@<文字列> ...]

説　明　　マクロ引数、マクロ変数、予約シンボル、制御命令"..FILE"の展開ファイル名および指定文字列を連結します。

例　　ファイル名の連結例：

現在処理中のファイル名が sample1.src の場合、sample.dat ファイルにメッセージを出力します。

```
.ASEERT   "sample" > ..FILE@.dat
```

文字列の連結例：

```
mov_nibble .MACRO p1,src,dest
 MOV.@p1 src,dest
 .ENDM


mov_nibble  W,R1,R2         ; マクロ呼び出し


 MOV.W R1,R2 ;              マクロ展開後コード
```

備　考　　本制御命令の前後に記述した空白文字およびタブは、文字列として連結します。

本制御命令の前後には、文字列が記述できます。

@を文字データ(40H)として記述する場合は、ダブルクォーテーション(")で囲んでください。@を含む文字列をシングルクォーテーション(')で囲った場合は、@の前後の文字列を連結します。

一行に複数回記述できます。

連結した文字列を名前とする場合は、本制御命令の前後に空白文字およびタブを記述しないでください。

ソースファイル名情報に置き換え

..FILE

書　式　　..FILE

説　明　　アセンブラが処理中のファイル名に展開されます（アセンブリ言語ファイル名またはインクルードファイル）。

例　　アセンブリ言語ファイル名が"sample.src"の場合、"sample"ファイルにメッセージを出力します。

```
.ASSERT "sample" > ..FILE
```

アセンブリ言語ファイル名が"sample.src"の場合、"sample.inc"ファイルをインクルードします。

```
.INCLUDE ..FILE@.inc
```

上記の行が、"sample.src"ファイルでインクルードしている"incl.inc"内に記述されている場合、通常、"incl.mes"に文字列を出力します。

```
.ASSERT "sample" > ..FILE@.mes
```

備　考　　制御命令".ASSERT"および制御命令".INCLUDE"のオペランドに記述できます。

本制御命令で読み込まれるファイル名は、ファイルの拡張子およびパスを除いた部分です。

指定シンボルに対してスタック値を設定

.STACK

書　式　　.STACK△<名前>=<数値>

説　明　　シンボルに対して、Call Walker で表示するスタック使用量を定義します。

例　　.STACK　SYMBOL=100H

備　考　　1 つのシンボルに対して定義できるスタック値は 1 度のみ有効とします。

2 度以上指定した場合は、その定義を無効とします。また、指定できるスタック値は、

0H～0FFFFFFFCH の範囲の 4 の倍数のみとし、それ以外を指定した場合はその定義を無効とします。

＜数値＞は定数値とし、かつ前方参照シンボル、外部参照シンボル、相対アドレスシンボルを使わずに指定してください。

行番号変更

.LINE

書　式　　.LINE△<ファイル名>,<行番号>

.LINE△<行番号>

説　明　　アセンブラのエラーメッセージあるいはデバッグ時に参照する行番号とファイル名を変更します。

プログラム内の最初の.LINE 以降は次の.LINE まで行番号、ファイル名を更新しません。

コンパイラは、デバッグオプションを指定してアセンブリ言語ファイルを出力する時に C 言語ソースファイル行に対応する.LINE を生成します。

ファイル名を省略するとファイル名は変更されず、行番号だけが変更されます。

例　　.LINE　"C:¥asm¥test.c",5

置き換えシンボルの定義

.DEFINE

書　式　　<シンボル名>△.DEFINE△<文字列>

<シンボル名>△.DEFINE△'<文字列>'

<シンボル名>△.DEFINE△"<文字列>"

説　明　　シンボルに文字列を定義します。

シンボルは再定義が可能です。

例

```
X_HI    .DEFINE   R1
        MOV.L  #0, X_HI
```

備　考　　空白文字またはタブを含む文字列を定義する場合は、必ずシングルクォーテーション(')または、ダブルクォーテーション(")で囲って記述してください。

本制御命令で定義されたシンボルは、外部参照指定ができません。

本制御命令と.EQU 制御命令で同名のシンボルを宣言した場合、先に記述した方が優先されます。

10.3.7 マクロ制御命令

マクロ機能および繰り返しマクロ機能を定義するための制御命令です。

表 10.36 マクロ制御命令

制御命令	機能内容
.MACRO	マクロ名を定義します。マクロボディの始まりを定義します。
.EXITM	マクロボディの展開を中止します。
.LOCAL	マクロ内ローカルラベルを宣言します。
.ENDM	マクロボディの終了を示します。
.MREPEAT	繰り返しマクロボディの始まりを示します。
.ENDR	繰り返しマクロボディの終了を示します。
..MACPARA	マクロ呼び出しの実引数の個数を値として持ちます。
..MACREP	繰り返しマクロボディの展開回数を値として持ちます。
.LEN	指定した文字列の文字数を値として持ちます。
.INSTR	指定した文字列の中で指定した文字列の始まる位置を値として持ちます。
.SUBSTR	指定した文字列の中で指定した位置から指定した文字数分の文字を切り出します。

マクロ定義

.MACRO

書　式

```
[マクロ定義]
△<マクロ名>△.MACRO[<仮引数>[,...]]
△ボディ
△.ENDM
[マクロ呼び出し]
△<マクロ名>△[<実引数>[,...]]
```

説　明

マクロ名を定義します。
マクロ定義の始まりを示します。

例

・例 1

```
[マクロ定義例]
name    .MACRO  string
        .BYTE   'string'
        .ENDM
 [マクロ呼び出し例 1]
name    "name,address"
```

```
.BYTE   'name,address'
```

[マクロ呼び出し例 2]

```
name    (name,address)

.BYTE   '(name,address)'
```

・例 2

```
mac             .MACRO  p1,p2,p3
        .IF     ..MACPARA == 3
                .IF     'p1' == 'byte'
                        MOV.B  #p2,[p3]
                .ELSE
                        MOV.W  #p2,[p3]
                .ENDIF
        .ELIF   ..MACPARA == 2
                .IF     'p1' == 'byte'
                        MOV.B  #p2,[R3]
                .ELSE
                        MOV.W  #p2,[R3]
                .ENDIF
        .ELSE
                MOV.W  R3,R1
        .ENDIF
                .ENDM

                mac     word,10,R3    ; マクロ呼び出し

        .IF     3 == 3                ; マクロ展開後コード
        .ELSE
                        MOV.W  #10,[R3]
        .ENDIF
```

備　考　マクロ名は必ず記述してください。

マクロ名は、「10.1.2　名前」の「名前の記述規則」に従ってください。

マクロ仮引数の名前は、「10.1.2　名前」の「名前の記述規則」に従ってください。

マクロ仮引数の名前は、ネストしているマクロ定義を含めて、異なる名前で定義してください。

仮引数を複数定義する場合は、仮引数をカンマ(,)で区切って記述してください。

制御命令".MACRO"のオペランドに記述した仮引数は、必ずマクロボディ内に記述してください。

マクロ名と実引数の間には、必ず空白文字またはタブを記述してください。

実引数は、マクロ呼び出しの際に仮引数に対応させて記述してください。

特殊文字を実引数に記述する場合は、ダブルクォーテーションで囲って記述してください。

実引数には、ラベル、グローバルラベルおよびシンボルが記述できます。

実引数には式が記述できます。

仮引数と実引数は、左から記述されている順に置き換えられます。

仮引数が定義されていて、マクロ呼び出しで実引数の記述がない場合は、仮引数にあたる部分のコードは出力されません。

仮引数の数が、実引数の数より多い場合は、対応する実引数がない仮引数にあたる部分のコードは出力されません。

ボディに記述した仮引数をシングルクォーテーション(')で囲った場合は、対応する実引数をシングルクォーテーションで囲って出力されます。

１つの実引数がカンマ(,)を含む場合に、括弧(())で囲った場合は、括弧を含めて変換されます。

実引数の数が、仮引数の数より多い場合は、対応する仮引数がない実引数については処理されません。

ダブルクォーテーションで囲った文字列は、全てその文字列そのものを示します。仮引数をダブルクォーテーションで囲わないでください。

仮引数は 80 個まで記述できます。

1 行に記述できる文字数の範囲内で最大 80 個まで記述できます。

実引数と仮引数の数が合わない場合は、アセンブラはウォーニングメッセージを出力します。

マクロ展開の中止

.EXITM

書　式　　<マクロ名>△.MACRO
△ボディ
△.EXITM
△ボディ
△.ENDM

説　明　　マクロボディの展開を中止し、最も近い".ENDM"に制御を渡します。

例

```
data1           .MACRO  value
        .IF     value == 0
                .EXITM
        .ELSE
                .BLKB   value
        .ENDIF
        .ENDM

data1           0       ; マクロ呼び出し

.IF             0 == 0  ; マクロ展開後コード
                .EXITM
.ENDIF
```

備　考　　マクロ定義のボディ内に記述してください。

マクロ内ローカルラベル宣言

.LOCAL

書　式　　.LOCAL△<ラベル名>[,...]

説　明　　オペランドに記述されたラベルがマクロローカルラベルであることを宣言します。

マクロローカルラベルは、異なるマクロ定義およびマクロ定義外であれば、同一の名前を複数個記述できます。

例

```
name    .MACRO
  .LOCAL        m1      ; 'm1' is macro local label
m1:
  nop
  bra   m1
  .ENDM
```

備　考　　本制御命令は、必ずマクロボディ内に記述してください。

本制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

本制御命令によるマクロローカルラベル宣言は、ラベル名を定義するより前に記述してください。

マクロローカルラベル名は、**「**10.1.2　**名前」**の**「名前の記述規則」**に従ってください。

本制御命令のオペランドは、カンマで区切って複数のラベルを記述できます。このときの最大ラベル数は 100 個までです。

マクロ定義がネストしている場合は、マクロ定義内で定義を行っているマクロ内のマクロローカルラベルは、同一名を使用できません。

インクルードファイルの内容を含む、1 つのアセンブリ言語ファイルに記述できるマクロローカルラベルは 65535 個までです。

マクロ定義の終了

.ENDM

書　式　　<マクロ名>△.MACRO
△ボディ
△.ENDM

説　明　　1 つのマクロ定義のボディが終了することを示します。

例

```
lda     .MACRO
        MOV.L  #value,R3
        .ENDM

lda     0       ; MOV.L  #0,R3 に展開される
```

繰り返しマクロの開始

.MREPEAT

書　式　　[<ラベル>:]△.MREPEAT△<数値>
△ボディ
△.ENDR

説　明　　繰り返しマクロの始まりを示します。
ボディを指定した数値回、繰り返して展開します。
繰り返し回数は、1 から 65535 の間の数を指定できます。
65535 レベルまでのネストができます。
本制御命令を記述した場所に、マクロボディを展開します。

例

```
rep     .MACRO  num
        .MREPEAT   num
                .IF     num > 49
                .EXITM
                .ENDIF
              nop
              .ENDR
  .ENDM

  rep   3       ; マクロ呼び出し

  nop           ; マクロ展開後コード
  nop
  nop
```

備　考　　オペランドは必ず記述してください。
本制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
本制御命令行の先頭にラベルを記述できます。
オペランドには、シンボルを記述できます。
前方参照となるシンボルは記述できません。
オペランドには、式が記述できます。
ボディには、マクロ定義およびマクロ呼び出しが記述できます。
ボディ内に制御命令".EXITM"を記述できます。

繰り返しマクロ終了

.ENDR

書　式　　[<ラベル>:]△.MREPEAT△<数値>
△ボディ
△.ENDR

説　明　　繰り返しマクロの終了を示します。

備　考　　必ず制御命令".MREPEAT"に対応させて記述してください。

マクロ実引数の数に置き換え

..MACPARA

書　式　　..MACPARA

説　明　　マクロ呼び出しの実引数の個数を示します。
".MACRO"によるマクロ定義のボディ内に記述できます。

例　　マクロ実引数の数を判断して、条件アセンブルを行います。

```
        .GLB    mem
name    .MACRO  f1,f2
        .IF     ..MACPARA == 2
        ADD  f1,f2
        .ELSE
        ADD  R3,f1
        .ENDIF
        .ENDM

        name  mem    ; マクロ呼び出し

        .ELSE        ; マクロ展開後コード
        ADD  R3,mem
        .ENDIF
```

備　考　　本制御命令は式の項として記述できます。
".MACRO"によるマクロボディの外に記述した場合、値は 0 となります。

現在のマクロ繰り返し回数に置き換え

..MACREP

書　式　　..MACREP

説　明　　繰り返しマクロが展開されている回数を示します。
".MREPEAT"によるマクロ定義のボディ内に記述できます。
条件アセンブルのオペランドに記述できます。

例

```
mac     .MACRO  value,reg
        .MREPEAT   value
        MOV.B   #0,..MACREP[reg]
        .ENDR
        .ENDM

        mac     3,R3    ; マクロ呼び出し

        .MREPEAT   3    ; マクロ展開後コード
        MOV.B   #0,1[R3]
        MOV.B   #0,2[R3]
        MOV.B   #0,3[R3]
        .ENDR
        .ENDM
```

備　考　　本制御命令は式の項として記述できます。
".MACRO"によるマクロボディの外に記述した場合、値は 0 となります。

指定文字列の長さに置き換え

.LEN

書　式　　.LEN△{"<文字列>"}
.LEN△{'<文字列>'}

説　明　　オペランドに記述した文字列の長さを示します。

例

```
bufset  .MACRO  f1
buffer: .BLKB  .LEN{'f1'}
                .ENDM

        bufset  Sample    ; マクロ呼び出し

buffer:         .BLKB   6   ; マクロ展開後コード
```

備　考　　オペランドは、必ず{}で囲ってください。
本制御命令とオペランドの間に空白文字またはタブが記述できます。
文字列には、空白文字およびタブを含む文字が記述できます。
文字列は、必ずクォーテーションで囲って記述してください。
本制御命令を式の項に記述できます。
マクロの実引数の文字列長を求める場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲った場合は、仮引数として指定した文字列の長さを示します。

文字列の開始位置に置き換え

.INSTR

書　式　　.INSTR△{ "<文字列>","<検出文字列>",<検出開始位置> }
.INSTR△{ '<文字列>','<検出文字列>',<検出開始位置> }

説　明　　オペランドで指定した文字列のなかで、検出文字列が始まる位置を示します。
文字列の検索を開始する位置を指定できます。

例　　指定した文字列(japanese)の先頭(top)からの、"se"文字列の位置(7)を取り出します。

```
top     .EQU   1
point_set       .MACRO  source,dest,top
point   .EQU    .INSTR{'source','dest',top}
        .ENDM
  point_set  japanese,se,1   ; マクロ呼び出し

  point         .EQU   7       ; マクロ展開後コード
```

備　考　　オペランドは、必ず{}で囲ってください。
文字列、検出文字列および検索開始位置は、必ず記述してください。
文字列、検出文字列および検索開始位置は、カンマで区切って記述してください。
カンマの前後には、空白文字およびタブは記述できません。
検索開始位置は、シンボルを記述できます。
検索開始位置を 1 とした場合は、文字列の先頭を示します。
本制御命令は、式の項に記述できます。
文字列よりも、検索文字列が長い場合の値は 0 となります。文字列のなかに、検索文字列が含まれていなかった場合の値は 0 となります。文字列の長さよりも、検索開始位置の値が大きかった場合の値は 0 となります。
マクロの実引数を検出条件としてマクロを展開したい場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲って記述した場合は、その文字列を検出条件としてマクロを展開します。

.SUBSTR

書　式　　.SUBSTR△{ "<文字列>",<切り出し開始位置>,<切り出し文字数> }

.SUBSTR△{ '<文字列>',<切り出し開始位置>,<切り出し文字数> }

説　明　　文字列の指定した位置から、指定した文字列を取り出します。

例　　マクロの実引数として与えられた文字列の長さを、".MREPEAT"のオペランドに与えます。

"..MACREP"は、".BYTE"の行を１回展開するごとに、1→2→3→4 と増加します。
したがって、マクロの実引数として与えられた文字列の先頭から順に１文字ずつ、".BYTE"のオペランドに与えることになります。

```
name    .MACRO  data
        .MREPEAT   .LEN{'data'}
        .BYTE   .SUBSTR{'data',..MACREP,1}
        .ENDR
        .ENDM

        name  ABCD      ; マクロ呼び出し

        .BYTE  "A"      ; マクロ展開後コード
        .BYTE  "B"
        .BYTE  "C"
        .BYTE  "D"
```

備　考　　オペランドは、必ず{}で囲ってください。

文字列、切り出し開始位置および切り出し文字数は、必ず記述してください。

文字列、切り出し開始位置および切り出し文字数は、カンマで区切って記述してください。

切り出し開始位置および切り出し文字数には、シンボルが記述できます。切り出し開始位置を１とした場合は、文字列の先頭を示します。

文字列には、空白文字およびタブを含む文字が記述できます。

文字列は、必ずクォーテーションで囲って記述してください。

文字列の長さよりも切り出し開始位置の値が大きい場合の値は０となります。文字列の長さよりも切り出し文字数の値が大きい場合の値は０となります。切り出し文字数を０とした場合の値は０となります。

マクロの実引数を切り出し条件としてマクロを展開したい場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲って記述した場合は、その文字列を切り出し条件としてマクロを展開します。

10.3.8 コンパイラ専用制御命令

コンパイラがアセンブリ言語ソースファイルを生成する際、C 言語の機能をアセンブラで適切に処理させるため、下記の制御命令を出力することがあります。

コンパイラが生成したアセンブリ言語ソースファイルを利用する場合、これらの制御命令の内容を変更せず、そのまま使用してください。また、ユーザアセンブリプログラム作成時には、これらの制御命令は使用しないでください。

表 10.37 コンパイラ専用制御命令

制御命令	内容
._LINE_TOP	#pragma inline_asm で指定された関数を展開した場合に出力されます。
._LINE_END	
.SWSECTION	switch 文で分岐テーブルを使用した場合に出力されます。
.SWMOV	
.SWITCH	
.INSTALIGN	instalign4, instalign8 オプション、または#pragma instalign4, #pragma instalign8 を使用した場合に出力されます。

11. コンパイラのエラーメッセージ

11.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
エラー内容

エラーレベルは、エラーの重要度に従い、5 種類に分類されます。

エラーレベル		動作
(I)	インフォメーション	処理を継続します。
(W)	ウォーニング	処理を継続します。
(E)	エラー	オプション解析処理を継続し、処理を中断します。
(F)	フェータル	処理を中断します。
(-)	インターナル	処理を中断します。

11.2 メッセージ一覧

C0005 (I) Precision lost
代入式において、右辺の式の値を左辺の型へ変換する時に、精度が失われる可能性があります。

C0006 (I) Conversion in argument
関数の引数の式が、関数原型で指定した引数の型に変換されます。

C0008 (I) Conversion in return
リターン文の式が、関数の返す値の型に変換されます。

C0011 (I) Used before set symbol : "変数名" in "関数名"
値の設定されていない局所変数を参照しています。

C0101 (I) Optimizing range divided in function "関数名"
"関数名"の最適化範囲が複数に分割されました。

C0102 (I) Register is not allocated to "変数名" in "関数名"

register 記憶クラスを持つ変数にレジスタを割り付けることができませんでした。

C1026 (W) Address of packed member

pack=1 指定ありの構造体メンバのアドレスを取得しています。

C1300 (W) Command parameter specified twice

同じコンパイラオプションを 2 度以上指定しています。同じコンパイラオプションの中で最後に指定したものを有効とします。

C1301 (W) "オプション" option ignored

"オプション"を無視してコンパイルをします。

C1308 (W) Duplicate number specified in option "オプション" : "番号"

"オプション"で同じ番号を指定しています。

C1309 (W) Section name “SI” or “SU” specified

"セクション名"に、”SI”または”SU”を指定しています。指定したセクション名で出力します。

C1315 (W) File_inline "ファイル名" ignored by same file as source file

コンパイル対象のファイルが file_inline オプションで指定されています。file_inline オプションを無視してコンパイルをします。

C1316 (W) "該当マクロ" is not a valid predefined macro name

マクロ名"<マクロ名>"はプリディファインドマクロではありません。undefine オプションの指定を無効とします。

C1317 (W) "オプション 1" and "オプション 2" are specified

互いに意味が矛盾する"オプション 1"と"オプション 2"を共に指定しています。
オプションは共に有効ですが、意図した組み合わせであるかどうか確認してください。

C1402 (W) #pragma section ignored

#pragama section 指定を無視します。

C1410 (W) A struct/union/class has different pack specifications
ひとつの構造体/共用体/クラスの中に、異なる pack 値を持つものが混在しています。

C1600 (W) Debugging information describing location of "名前" is lost
"名前"のシンボル情報が出力されませんでした。

C1800 (W) Variable "変数名" type mismatch in files
"変数名"で示す変数の型がファイル間で異なっています。
file_inline オプションの指定をやめてください。

C1801 (W) Using “機能項目” at influence the code generation of "NC" compiler
NC コンパイラとの互換性に影響のある“機能項目”が使用されています。

C1802 (E)(W) Using “機能項目” at influence the code generation of "H8" compiler
H8 コンパイラとの互換性に影響のある“機能項目”が使用されています。

C1803 (W) Address taken "変数名". It may cause an upset endian indirect reference
endian オプションと異なるエンディアンの 8 バイト変数"変数名"のアドレスが取得されました。エンディアン処理が正しくない間接参照を引き起こす可能性があります。

C1804 (W) Using incompatible int type
C++コンパイル時に、int_to_short オプションは無効になるため、C++コンパイルと C コンパイルで int 型のサイズが異なります。C++プログラム内で C プログラムの外部名を参照している可能性がある場合に、本メッセージを出力します。

C1805 (W) "シンボル名" is not confirmed in ROM by map option
const 修飾子付きで宣言した外部参照シンボル"シンボル名"が、map オプションでは ROM 上のシンボルと確認できませんでした。

C1806 (W) "シンボル名" is regarded in ROM by map seciton
const 修飾子なしで宣言した外部参照シンボル"シンボル名"は、map オプションによって ROM 上のシンボルと判定しました。

C1807 (E)(W) Using “機能項目” at influence the code generation of "SuperH" compiler
SuperH コンパイラとの互換性に影響ある“機能項目”(オプションや#pragma など)が使用されています。

C1950 (W) Nothing to compile, assemble or link (input and output combination)
コンパイル、アセンブルまたはリンク処理のいずれも行う必要がありません。入力ファイルの構成とoutput オプション指定の組み合わせを確認してください。Ignored argument(s):以下に処理を行わなかった一覧を表示します。

C2021 (E) Invalid number specified in option "オプション" : "番号"
"オプション"指定で無効な値を指定しています。値の範囲を確認してください。

C2022 (E) Error level message cannot be changed : "change_message"
Error レベルのメッセージは、メッセージレベルを変更できません。

C2023 (E) Same register is used at base option.
base オプションの異なる領域に対して同じレジスタが指定されています。

C2024 (E) Base register is already used at fint_register option.
fint_register オプションで使用禁止としたレジスタが base オプションで指定されています。

C2025 (E) Base option address constant overflow
base オプションのアドレス値が 0x00000000～0xffffffff の範囲を超えています。

C2026 (E) Illegal register of base option
base オプションのレジスタ番号が誤っています。R8～R13 以外を指定しています。

C2027 (E) Cannot read specified file "ファイル名"
ファイルが正常に読み込めません。ファイルの指定が間違っていないか確認してください。

C2028 (E) Base register conflicts with option "オプション名
base オプションで指定されたレジスタは、"オプション名"で指定されたレジスタで既に使われています。

C2203 (E) Illegal member reference for "."
演算子.の左側の式の型が構造体型または共用体型ではありません。

C2240 (E) Illegal section naming
セクションの命名に誤りがあります。用途の異なるセクションに同じ名前を付けています。

C2450 (E) Illegal #pragma option declaration
#pragma option 宣言に誤りがあります。

C2550 (E) Assignment of ROM section object "変数名"

ROM セクション上にある"変数名"に書込みを行いました。

リンク時に-rom オプションが適切に適用されていない可能性があります。

C2700 (E) Function "関数名" in #pragma interrupt already declared

割り込み関数宣言#pragma interrupt で指定した関数が、すでに通常関数として宣言されています。

C2701 (E) Multiple interrupt for one function

1 つの関数に対して割り込み関数宣言#pragma interrupt を重複して宣言しています。

C2703 (E) Illegal #pragma interrupt declaration

割り込み関数宣言#pragma interrupt の仕様に誤りがあります。

C2704 (E) Illegal reference to interrupt function

割り込み関数を不正に参照しています。

C2710 (E) Section name too long

指定したセクション名の文字数が限界値を超えています。

C2711 (E) Section name table overflow

指定したセクションの数が限界値を超えています。

C2714 (E) Usable stack area overflow

スタックへのアクセスで、SP 相対アドレッシングで参照できない範囲をアクセスしようとしたために、命令を生成できませんでした。

配列のインデックスに負数を設定している、もしくは自動変数領域が大きすぎるのが原因と思われます。ソース記述を見直してください。

C2800 (E) Illegal parameter number in in-line function

組み込み関数で使用する引数の数が一致しません。

C2801 (E) Illegal parameter type in in-line function

組み込み関数で引数の型が一致しません。

C2802 (E) Parameter out of range in in-line function

組み込み関数で引数の大きさが指定可能範囲を超えています。

C2803 (E) Invalid offset value in in-line function
組み込み関数で引数の指定が不適当です。

C2804 (E) Illegal in-line function
指定された cpu オプションでは使用できない組み込み関数があります。

C2806 (E) Multiple #pragma for one function
1 つの関数に対して複数の矛盾した#pragma 指定をしています。

C2831 (E) Multiple #pragma entry declaration
#pragma entry 宣言が複数存在しています。

C2833 (E) Multiple #pragma stacksize declaration
si または su 指定の#pragma stacksize 宣言が複数存在しています。

C2854 (E) Illegal address in #pragma address
指定アドレスが以下のいずれかに該当しています。
(1) 異なる変数に対して、同一アドレスを指定している。
(2) 異なる変数に対して、変数のアドレスが重なっている。

C2860 (E) Missing #pragma oscall for “関数名”
関数"サービスコール名"に必要な#pragma oscall の指定がありません。

C3009 (F) String literal too long
文字列の文字数が限界値を超えています。文字列の文字数は、連続して指定した文字列を連結した後のバイト数です。ここでいう文字列の文字数とは、ソースプログラム上の長さではなく文字列のデータに含まれるバイト数で、拡張表記も 1 文字に数えます。

C3019 (F) Cannot open source file "ファイル名"
ソースファイルをオープンすることができません。

C3020 (F) Source file input error "ファイル名"
ソースファイルまたはインクルードファイルを読み込むことができません。

C3021 (F) Memory overflow
コンパイラが内部で使用するメモリ領域を割り当てることができません。

C3023 (F) Type nest too deep

基本型を修飾する型(ポインタ型、配列型、関数型)の数が限界値を超えています。

C3024 (F) Array dimension too deep

配列の次元数が限界値を超えています。

C3025 (F) Source file not found

コマンドラインの中にソースファイル名の指定がありません。

C3030 (F) Too many compound statements

1 関数における複文の数が限界値を超えています。

C3031 (F) Data size overflow

配列または構造体の大きさが、限界値を超えています。

C3203 (F) Assembly source line too long

出力するアセンブリソースの 1 行が長すぎます。

C3204 (F) Illegal stack access

関数内で使用するスタックのサイズ(局所変数領域、レジスタ退避領域その他関数呼び出しのためのパラメータプッシュ領域等含む)または、その関数呼び出しのためのパラメータ領域が 2G バイトを超えています。

C3300 (F) Cannot open internal file

以下、3 つの場合のいずれかでエラーが起こっている可能性があります。

(1)コンパイラが内部で生成する中間ファイルをオープンすることができません。

(2)中間ファイルと同じ名前のファイルが既に存在しています。

(3)コンパイラが内部で使用するファイルをオープンすることができません。

C3301 (F) Cannot close internal file

コンパイラが内部で生成する中間ファイルをクローズすることができません。コンパイラのインストール手順に誤りがないことを確認してください。

C3302 (F) Cannot input internal file

コンパイラが内部で生成する中間ファイルを読み込むことができません。コンパイラのインストール手順に誤りがないことを確認してください。

C3303 (F) Cannot output internal file

コンパイラが内部で生成する中間ファイルに書き込むことができません。ディスクの空き容量を増やしてください。

C3304 (F) Cannot delete internal file

コンパイラが内部で生成する中間ファイルを削除することができません。コンパイラが生成する中間ファイルをアクセスしていないかを確認してください。

C3305 (F) Invalid command parameter "オプション"

コンパイラオプションの指定方法が誤っています。

C3306 (F) Interrupt in compilation

コンパイル処理中に標準入力端末から(Ctrl)+C コマンドによる割り込みを検出しました。

C3307 (F) Compiler version mismatch

コンパイラを構成するファイル間のバージョンが一致していません。インストールガイドの組み込み方法を参照し、コンパイラ本体を再インストールしてください。

C3308 (F) Cannot create file "ファイル名"

コンパイラが生成するファイルを作成できません。

C3320 (F) Command parameter buffer overflow

コマンドラインの指定が 4096 文字を超えています。

C3321 (F) Illegal environment variable

以下の 4 つの場合のいずれかでエラーが起こっています。

(1)環境変数 BIN_RX が設定されていません。

(2)環境変数 BIN_RX にコンパイラの実行ファイルパス名が指定されていません。

(3)環境変数 BIN_RX の設定でファイル名の規約に反した指定をしているか、パス名の長さが 118 文字を超えています。

(4)環境変数 CPU_RX に、"RX600"以外の設定がされています。

C3322 (F) Lacking cpu specification

CPU の指定がされていません。cpu オプションまたは環境変数 CPU_RX で CPU を指定してください。

C3900 (E) Input file not found. – "ファイル名"

入力指定されたファイル名がありません。

C3901 (E) Input file read error. – "ファイル名"
入力ファイルに読み込みエラーが発生しました。

C3902 (E) Invalid file name. – "ファイル名"
入力ファイル名に利用できない文字が指定されています。

C3903 (E) Invalid option. – "オプション指定"
オプション指定が正しくありません。

C3905 (E) Cannot build temporary file.
一時ファイルが作成できません。コンパイラの環境設定に問題がないか確認ください。

C3906 (E) Memory overflow.
コンパイラで使用するメモリが不足しています。

C3907 (E) Tool execute error.
コンパイラ、アセンブラ、または最適化リンケージエディタのいずれかの起動に失敗しました。

C3908 (E) Cannot delete temporary file.
一時ファイルが削除できません。コンパイラの環境設定に問題がないか確認ください。

C4000-C4999 (-) Internal error
コンパイラの内部処理で何らかの障害が生じました。本コンパイラをお求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。

C5001 (E) Last line of file ends without a newline
ファイルの最終行の末尾に改行文字がありません。

C5002 (E) Last line of file ends with a backslash
ファイルの最終行の末尾がバックスラッシュになっています。

C5003 (F) #include file "ファイル名" includes itself
自分自身のファイル"ファイル名"をインクルードしています。

C5004 (F) Out of memory

コンパイルに必要なメモリが不足しています。システムのメモリを増やすか、他のアプリケーションを終了してください。

C5005 (F) Could not open source file "名前"

ファイル"名前"をオープンできませんでした。ファイル名が正しいか確認してください。

C5006 (E) Comment unclosed at end of file

コメントの終了指定*/がありません。

C5007 (E) (I) Unrecognized token

認識できない字句があります(マクロの場合は(I)となります)。

C5008 (E) (I) Missing closing quote

文字列の終了指定"がありません(マクロの場合は(I)となります)。

C5009 (I) Nested comment is not allowed

/* */コメントがネストしています。

C5010 (E) "#" not expected here

#が行の先頭、プリプロセッサ以外に指定されています。

C5011 (E)(W) Unrecognized preprocessing directive

認識できないプリプロセッサのキーワードがあります。

C5012 (E)(W) Parsing restarts here after previous syntax error

字句の解析を再開しました。

C5013 (E) (F) Expected a file name

ファイル名が必要です。#include 文では(F)、#line 文では(E)となります。

C5014 (E) Extra text after expected end of preprocessing directive

プリプロセッサ文の後にさらにテキストが記述されています。

C5016 (F) "名前" is not a valid source file name

ファイル"名前"が有効でありません。

C5017 (E) Expected a "]"
"]"がありません。

C5018 (E) Expected a ")"
")"がありません。

C5019 (E) Extra text after expected end of number
数値の後ろにさらにテキストが記述されています。

C5020 (E) Identifier "名前" is undefined
シンボル"名前"の定義がありません。

C5021 (W) Type qualifiers are meaningless in this declaration
意味のない型限定子を指定しています。型限定子を無効にします。

C5022 (E) Invalid hexadecimal number
16 進数の記述に誤りがあります。

C5023 (E) Integer constant is too large
整数定数の値が大きすぎます。

C5024 (E) Invalid octal digit
8 進数の記述に誤りがあります。

C5025 (E) Quoted string should contain at least one character
文字定数が空です。

C5026 (E) Too many characters in character constant
文字定数中の文字数が多すぎます。

C5027 (W) Character value is out of range
文字の値が範囲を超えています。超えた値は切り捨てられます。

C5028 (E) Expression must have a constant value
式の値が定数でありません。

C5029 (E) Expected an expression
式が必要です。

C5030 (E) Floating constant is out of range
浮動小数点型の値が範囲を超えています。

C5031 (E)(W) Expression must have integral type
式の型は整数型でなければなりません。

C5032 (E) Expression must have arithmetic type
式の型は算術型でなければなりません。

C5033 (E) Expected a line number
#line 文には行番号が必要です。

C5034 (E) Invalid line number
#line 文の行番号が有効でありません。

C5035 (F) #error directive: "行番号"
#error 文が適用されました。

C5036 (E) The #if for this directive is missing
#if 文の指定方法に誤りがあります。

C5037 (E) The #endif for this directive is missing
#endif 行の指定方法に誤りがあります。

C5038 (E)(W) Directive is not allowed -- an #else has already appeared
#else 文はすでに出現しました。本指定を読み飛ばします。

C5039 (E)(W) Division by zero
ゼロ除算が発生しました。

C5040 (E) Expected an identifier
識別子が必要です。

C5041 (E) Expression must have arithmetic or pointer type
式の型は算術型またはポインタ型でなければなりません。

C5042 (E)(W) Operand types are incompatible ("型 1" and "型 2")
"型 1"と"型 2"のオペランドの型が適合しません。

C5044 (E) Expression must have pointer type
式の型はポインタ型でなければなりません。

C5045 (W) #undef may not be used on this predefined name
システムで定義しているマクロ名を取り消すことはできません。#undef 指定を無効にします。

C5046 (W) “マクロ名” is predefined; attempted redefinition ignored
システムで定義しているマクロ名を再定義することはできません。#define 指定を無効にします。

C5047 (W) Incompatible redefinition of macro "名前" (declared at line "行番号")
マクロ"名前"の再定義が以前の定義と異なります。再定義したマクロを有効にします。

C5049 (E) Duplicate macro parameter name
マクロのパラメータ名を二重定義しています。

C5050 (E) "##" may not be first in a macro definition
#define マクロの最初に##が指定されています。

C5051 (E) "##" may not be last in a macro definition
#define マクロの最後に##が指定されています。

C5052 (E) Expected a macro parameter name
#に続くマクロ引数がありません。

C5053 (E) Expected a ":"
":"が必要です。

C5054 (W) Too few arguments in macro invocation
マクロ展開時の実引数が足りません。

C5055 (W) Too many arguments in macro invocation
マクロ展開時の実引数が多すぎます。

C5056 (E) Operand of sizeof may not be a function
sizeof 演算のオペランドに関数を指定できません。

C5057 (E) This operator is not allowed in a constant expression
この演算子は定数式中に指定できません。

C5058 (E) This operator is not allowed in a preprocessing expression
この演算子はプリプロセッサの式中で指定できません。

C5059 (E) Function call is not allowed in a constant expression
定数式中で関数呼び出しはできません。

C5060 (E) This operator is not allowed in an integral constant expression
この演算子は整数型定数式中で指定できません。

C5061 (W) Integer operation result is out of range
整数演算の結果が値の範囲を超えました。オーバーフローした上位ビットを無視した値を仮定します。

C5062 (W) Shift count is negative
シフトカウントが負の値です。指定された通りに演算します。

C5063 (W) Shift count is too large
シフトカウントが有効ビット数を超えています。指定された通りに演算します。

C5064 (W) Declaration does not declare anything
宣言を指定するシンボルがありません。宣言を無視します。

C5065 (E)(W) Expected a ";"
";"が必要です。

C5066 (E) Enumeration value is out of "int" range
列挙型メンバの値が int 型の範囲を超えました。

C5067 (E) Expected a "}"
"}"が必要です。

C5068 (W) Integer conversion resulted in a change of sign
符号変換を伴った整数型変換が実施されました。ビット列をそのまま設定します。

C5069 (W) Integer conversion resulted in truncation
上位バイト側を切り捨てる整数型変換が実施されました。切り捨て後の値を設定します。

C5070 (E) Incomplete type is not allowed
不完全型が指定されています。

C5071 (E) Operand of sizeof may not be a bit field
sizeof 演算子のオペランドにビットフィールドが指定されています。

C5075 (E) Operand of "*" must be a pointer
*演算子のオペランドの型がポインタ型ではありません。

C5076 (W) Argument to macro is empty
関数マクロに対して引数が指定されていません。

C5077 (E) This declaration has no storage class or type specifier
記憶クラスまたは型の指定がありません。

C5078 (E) A parameter declaration may not have an initializer
パラメーター宣言には初期化子を指定できません。

C5079 (E) Expected a type specifier
型指定子が必要です。

C5080 (E)(W) A storage class may not be specified here
ここでは記憶クラスを指定することはできません。

C5081 (E) More than one storage class may not be specified
記憶クラスを複数指定することはできません。

C5082 (W) Storage class is not first
記憶クラスがデータ型の前に指定されていません。

C5083 (W) Type qualifier specified more than once
const/volatile 限定子を複数指定しています。余分な指定を無視します。

C5084 (E) Invalid combination of type specifiers
型の組み合わせが正しくありません。

C5085 (W) Invalid storage class for a parameter
仮引数に不当な記憶クラスを指定しています。

C5086 (E) Invalid storage class for a function
関数に不当な記憶クラスを指定しています。

C5087 (E) A type specifier may not be used here
型を指定することはできません。

C5088 (E) Array of functions is not allowed
関数を要素とする配列は指定できません。

C5089 (E) Array of void is not allowed
void 型を要素とする配列は指定できません。

C5090 (E) Function returning function is not allowed
関数型をリターン型とする関数は指定できません。

C5091 (E) Function returning array is not allowed
配列をリターン型とする関数は指定できません。

C5092 (E) Identifier-list parameters may only be used in a function definition
識別子リストパラメータは関数定義以外の場所で使用できません。

C5093 (E) Function type may not come from a typedef
typedef 宣言された関数型を使用することはできません。

C5094 (E) The size of an array must be greater than zero
配列のサイズは 0 より大きな値でなければなりません。

C5095 (E) Array is too large
配列のサイズが大きすぎます。

C5096 (W) A translation unit must contain at least one declaration
翻訳単位内には最低１つの宣言が必要です。

C5097 (E) A function may not return a value of this type
関数はこの型の値を返すことができません。

C5098 (E) An array may not have elements of this type
配列はこの型を要素とすることができません。

C5099 (E)(W) A declaration here must declare a parameter
この関数宣言はパラメータを宣言する必要があります。

C5100 (E) Duplicate parameter name
仮引数の名前が重複しています。

C5101 (E) "名前" has already been declared in the current scope
同一スコープ内にすでに"名前"の宣言が存在します。

C5102 (E) Forward declaration of enum type is nonstandard
enum 型の前方宣言は標準形式ではありません。

C5103 (E) Class is too large
クラスのサイズが大きすぎます。

C5104 (E) Struct or union is too large
構造体または共用体のサイズが大きすぎます。

C5105 (E) Invalid size for bit field
ビットフィールドのサイズが不正です。

C5106 (E) Invalid type for a bit field
ビットフィールドの型が不正です。

C5107 (E)(W) Zero-length bit field must be unnamed
長さ 0 のビットフィールドには名前をつけられません。

C5108 (W) Signed bit field of length 1
符号付整数型の長さ 1 のビットフィールドが指定されています。指定された型で処理します。

C5109 (E) Expression must have (pointer-to-) function type
式は関数型へのポインタ型でなければなりません。

C5110 (E) Expected either a definition or a tag name
宣言の定義またはタグ名が必要です。

C5111 (W) Statement is unreachable
実行されない文です。最適化により削除される可能性があります。

C5112 (E) Expected "while"
while キーワードが必要です。

C5114 (E)(W) Entity-kind "名前" was referenced but not defined
参照される"名前"の定義がありません。

C5115 (E) A continue statement may only be used within a loop
continue 文はループの中で有効です。

C5116 (E) A break statement may only be used within a loop or switch
break 文はループまたは switch 文の中で有効です。

C5117 (W) Non-void entity-kind "名前" should return a value
void 型でない関数がリターン値を返しません。リターン値は不定です。

C5118 (E) A void function may not return a value
void 型を返す関数はリターン値を返すことはできません。

C5119 (E) Cast to type "型" is not allowed
"型"へのキャストは指定できません。

C5120 (E) Return value type does not match the function type
リターン値と関数の型が合いません。

C5121 (E) A case label may only be used within a switch
case ラベルを switch 文以外で使用しています。

C5122 (E) A default label may only be used within a switch
default ラベルを switch 文以外で使用しています。

C5123 (E) Case label value has already appeared in this switch
case ラベルの値がすでに switch 文の中に存在します。

C5124 (E) Default label has already appeared in this switch
default ラベルの値がすでに switch 文の中に存在します。

C5125 (E) Expected a "("
"("が必要です。

C5126 (E) Expression must be an lvalue
式は左辺値でなければなりません。

C5127 (E) Expected a statement
文が必要です。

C5128 (W) Loop is not reachable from preceding code
実行されない繰り返し文です。

C5129 (E) A block-scope function may only have extern storage class
ブロック内で宣言された関数は extern 記憶クラスでなければなりません。

C5130 (E) Expected a "{"
"{"が必要です。

C5131 (E) Expression must have pointer-to-class type
式はクラスへのポインタ型でなければなりません。

C5132 (E) Expression must have pointer-to-struct-or-union type
式は構造体または共用体へのポインタ型でなければなりません。

C5133 (E) Expected a member name
メンバ名が必要です。

C5134 (E) Expected a field name
フィールド名が必要です。

C5135 (E) Entity-kind "名前" has no member "メンバ名"
"名前"は"メンバ名"を持ちません。

C5136 (E) Entity-kind "名前" has no field "フィールド名"
"名前"は"フィールド名"を持ちません。

C5137 (E)(W) Expression must be a modifiable lvalue
式は修正可能な左辺値でなければなりません。

C5138 (E)(W) Taking the address of a register field is not allowed
レジスタフィールドのアドレスを参照することはできません。

C5139 (E) Taking the address of a bit field is not allowed
ビットフィールドのアドレスを参照することはできません。

C5140 (E)(W) Too many arguments in function call
関数呼び出しの実引数の数が多すぎます。

C5141 (E) Unnamed prototyped parameters not allowed when body is present
定義された関数のプロトタイプ宣言のパラメータに名前がありません。

C5142 (E) Expression must have pointer-to-object type
式はオブジェクトへのポインタ型でなければなりません。

C5143 (F) Program too large or complicated to compile
プログラムが大きすぎるかまたは複雑すぎます。

C5144 (E) A value of type "型 1" cannot be used to initialize an entity of type "型 2"
初期値の"型 1"と変数の"型 2"が異なります。

C5145 (E) Entity-kind "名前" may not be initialized
"名前"を初期化することはできません。

C5146 (E) Too many initializer values
初期値の数が多すぎます。

C5147 (E)(W) Declaration is incompatible with "名前" (declared at line "行番号")
前に宣言した"名前"の型が合致しません。

C5148 (E) Entity-kind "名前" has already been initialized
すでに"名前"の初期値が設定されています。

C5149 (E) A global-scope declaration may not have this storage class
大域的なスコープでの宣言にはこの記憶クラスを指定できません。

C5150 (E) A type name may not be redeclared as a parameter
型名を仮引数で再宣言することはできません。

C5151 (E) A typedef name may not be redeclared as a parameter
型名を仮引数で再宣言することはできません。

C5152 (W) Conversion of nonzero integer to pointer
ゼロ以外の整数をポインタに変換しようとしました。

C5153 (E) Expression must have class type
式はクラス型でなければなりません。

C5154 (E) Expression must have struct or union type
式は構造体または共用体型でなければなりません。

C5155 (W) Old-fashioned assignment operator
古いスタイルの代入オペレータが使用されました。

C5156 (W) Old-fashioned initializer
古いスタイルの初期化子が使用されました。

C5157 (E)(W) Expression must be an integral constant expression
式は整数型の定数式でなければなりません。

C5158 (E) Expression must be an lvalue or a function designator
式は左辺値または関数名でなければなりません。

C5159 (E) Declaration is incompatible with previous "名前" (declared at line "行番号")
前に使用した"名前"の型と合致しません。

C5160 (E) Name conflicts with previously used external name "名前"
前に使用した外部名"名前"と名前が重複しています。

C5161 (W) Unrecognized #pragma
認識できない#pragma 指定があります。#pragma 指定を無視します。

C5163 (F) Could not open temporary file "名前"
テンポラリファイル"名前"をオープンできませんでした。コンパイラの環境設定やホスト環境のファイルシステム異常がないか確認ください。

C5164 (F) Name of directory for temporary files is too long ("名前")
テンポラリファイルの"名前"が長すぎます。

C5165 (E) Too few arguments in function call
関数呼び出しの実引数の数が足りません。

C5166 (E) Invalid floating constant
浮動小数点定数の指定が不正です。

C5167 (E) Argument of type "型1" is incompatible with parameter of type "型2"
実引数の型"型1"と仮引数の型"型2"とが合致しません。

C5168 (E) A function type is not allowed here
関数型は許されません。

C5169 (E)(W) Expected a declaration
宣言が必要です。

C5170 (W) Pointer points outside of underlying object
ポインタが指している領域がオブジェクトの範囲を超えています。

C5171 (E) Invalid type conversion
キャストの型が不正です。

C5172 (W)(I) External/internal linkage conflict with previous declaration
前の宣言と外部/内部リンケージが異なります。内部リンケージが仮定されます。

C5173 (E)(W) Floating-point value does not fit in required integral type
浮動小数点型の値を整数型に変換するときに値の範囲を超えました。

C5174 (I) Expression has no effect
効果のない式です。最適化で削除される可能性があります。

C5175 (E)(W) Subscript out of range
配列のインデックスが範囲を超えています。指定されたインデックスで処理を継続します。

C5177 (W) Entity-kind "名前" was declared but never referenced
参照されない宣言があります。

C5178 (W) "&" applied to an array has no effect
配列名の前に"&"があります。無視します。

C5179 (W) Right operand of "%" is zero
%演算子の右辺が値０です。指定された式で評価します。

C5180 (W)(I) Argument is incompatible with formal parameter
引数が古い形式のパラメータと合致しません。

C5181 (W) Argument is incompatible with corresponding format string conversion
引数が対応する文字列変換形式と合致しません。

C5182 (F) Could not open source file "名前" (no directories in search list)
ファイル"名前"をオープンできませんでした。フォルダが存在するかどうか確認してください。

C5183 (E) Type of cast must be integral
キャストの型は整数型でなければなりません。

C5184 (E) Type of cast must be arithmetic or pointer
キャストの型は算術型またはポインタ型でなければなりません。

C5185 (I) Dynamic initialization in unreachable code
初期化式は実行されません。実行時に初期値は設定されません。

C5186 (W) Pointless comparison of unsigned integer with zero
0 と符号なし整数の無意味な比較をしています。指定された通りに式を評価します。

C5187 (I) Use of "=" where "==" may have been intended
"=="が意図される式で"="が使われています。指定された通りに式を評価します。

C5188 (W) Enumerated type mixed with another type
列挙型が他の列挙型またはデータ型に変換されています。

C5189 (F) Error while writing "ファイル名" file
ファイルの書き込みに失敗しました。

C5190 (F) Invalid intermediate language file
不正な中間言語ファイルです。

C5191 (W) Type qualifier is meaningless on cast type
キャストの型に意味のない型限定子を指定しています。指定された型を無視します。

C5192 (W) Unrecognized character escape sequence
認識できないエスケープシーケンス文字を指定しています。値をそのまま使用します。

C5193 (I) Zero used for undefined preprocessing identifier
プリプロセッサ文の式評価に値 0 が使われました。指定された通りに式を評価します。

C5194 (E) Expected an asm string
asm 文字列が必要です。

C5195 (E) An asm function must be prototyped
asm 関数はプロトタイプ宣言されている必要があります。

C5196 (E) An asm function may not have an ellipsis
asm 関数のパラメータに省略記号(…)は使用できません。

C5219 (F) Error while deleting file "ファイル名"
ファイル"ファイル名"を削除することができません。

C5220 (E) Integral value does not fit in required floating-point type
整数値を要求された浮動小数点型に変換できません。

C5221 (E) Floating-point value does not fit in required floating-point type
浮動小数点型を要求された浮動小数点型に変換できません。無限大の値とみなします。

C5222 (E) Floating-point operation result is out of range
浮動所数点演算の結果が値の範囲を超えました。オーバーフローした上位ビットを無視した値を仮定します。

C5223 (W) Function 関数名 declared implicitly
関数が暗黙的に宣言されました。

C5224 (W) The format string requires additional arguments
フォーマット文字列で要求する引数より実引数の数が足りません。

C5225 (W) The format string ends before this argument
フォーマット文字列が要求する引数より実引数の数が多すぎます。

C5226 (W) Invalid format string conversion
フォーマット変換の形式が実引数の型と異なります。

C5227 (E) Macro recursion
再帰的なマクロの展開レベルが 300 を超えています。

C5228 (W) Trailing comma is nonstandard
リストの最後の要素に与える値の直後にコンマをつけるのは標準形式ではありません。

C5229 (W) Bit field cannot contain all values of the enumerated type
ビットフィールドが列挙型全ての値を保持できません。結果は切り捨てられます。

C5230 (W) Nonstandard type for a bit field
ビットフィールドとして標準形式でないデータ型を使用しています。

C5231 (W) Declaration is not visible outside of function
関数プロトタイプ宣言内のタイプ宣言は関数の外からは見えません。

C5232 (W) Old-fashioned typedef of "void" ignored
古い形式である void の typedef は無効になります。

C5233 (W) Left operand is not a struct or union containing this field
左オペランドの構造体または共用体には無いフィールドを指定しました。

C5234 (W) Pointer does not point to struct or union containing this field
ポインタの指す構造体または共用体には無いフィールドを指定しました。

C5235 (E) Variable "名前" was declared with a never-completed type
変数"名前"が不完全型のまま宣言されました。

C5236 (W) (I) Controlling expression is constant
制御式が定数です(I)。制御式がアドレス定数です(W)。指定された通りに式を評価します。

C5237 (I) Selector expression is constant
switch 文の制御式が定数です。

C5238 (E) Invalid specifier on a parameter
引数宣言で不正な指定子を使用しています。

C5239 (E) Invalid specifier outside a class declaration
クラス宣言外で不正な指定子を使用しています。

C5240 (E) Duplicate specifier in declaration
1 つの宣言内で指定子を重複して使用しています。

C5241 (E) A union is not allowed to have a base class
union 型は基底クラスを持つことはできません。

C5242 (E) Multiple access control specifiers are not allowed
アクセス指定子が重複して使われています。

C5243 (E) Class or struct definition is missing
class 定義の括弧の対応がとれません。

C5244 (E) Qualified name is not a member of class "型" or its base classes
限定名がクラスまたは基底クラスのメンバの"型"でありません。

C5245 (E) A nonstatic member reference must be relative to a specific object
非静的メンバの参照がオブジェクトに対応していません。

C5246 (E) A nonstatic data member may not be defined outside its class
非静的データメンバはクラス外で定義できません。

C5247 (E) Entity-kind "名前" has already been defined
"名前"はすでに定義されています。

C5248 (E) Pointer to reference is not allowed
リファレンス型へのポインタ型は許されません

C5249 (E) Reference to reference is not allowed
リファレンス型へのリファレンス型は許されません。

C5250 (E) Reference to void is not allowed
void 型へのリファレンス型は許されません。

C5251 (E) Array of reference is not allowed
リファレンス型の配列は許されません。

C5252 (E) Reference entity-kind "名前" requires an initializer
リファレンス型の定義"名前"には初期値が必要です。

C5253 (E) Expected a ","
カンマ","が必要です。

C5254 (E) Type name is not allowed
型名は許されません。

C5255 (E) Type definition is not allowed
型の定義は許されません。

C5256 (E) Invalid redeclaration of type name "名前" (declared at line "行番号")
型名"名前"を再定義することはできません。

C5257 (E) Const entity-kind "名前" requires an initializer
const 型の定義"名前"には初期値が必要です。

C5258 (E) "this" may only be used inside a nonstatic member function
"this"が非静的メンバ関数以外で使われています。

C5259 (E) Constant value is not known
const 型の値が不明です。

C5260 (W) Explicit type is missing ("int" assumed)
型を指定していません。int 型を仮定します。

C5261 (I) Access control not specified ("名前" by default)
基底クラスのアクセス制御指定がありません。アクセス制御指定"名前"が仮定されます。

C5262 (E)(W) Not a class or struct name
基底クラスで指定されたクラスまたは構造体がありません。

C5263 (E) Duplicate base class name
基底クラスを二重に指定しています。

C5264 (E) Invalid base class
基底クラスが不正です。

C5265 (E) Entity-kind "名前" is inaccessible
"名前"をアクセスすることはできません。

C5266 (E) "名前" is ambiguous
指定された"名前"が曖昧です。

C5268 (E) Declaration may not appear after executable statement in block
宣言がブロックの実行文の後にありません。

C5269 (E) Conversion to inaccessible base class "型" is not allowed
参照不可能な基底クラス "型" に変換できません。

C5274 (E) Improperly terminated macro invocation
マクロ呼び出しの途中でファイルが終了しました。

C5276 (E) Name followed by "::" must be a class or namespace name
::に続く名前はクラス名または namespace 名でなければなりません。

C5277 (E) Invalid friend declaration
フレンド宣言の指定が正しくありません。

C5278 (E) A constructor or destructor may not return a value
コンストラクタやデストラクタはリターン値を持てません。

C5279 (E) Invalid destructor declaration
デストラクタの宣言が正しくありません。

C5280 (E)(W) Declaration of a member with the same name as its class
クラス名と同じ名前のメンバ名を宣言しています。
(W) 非 static 変数名
(E) static 変数名,typedef 名,enum メンバなど

C5281 (E) Global-scope qualifier (leading "::") is not allowed
グローバルなスコープ決定演算子は許されません。

C5282 (E) The global scope has no "名前"
"名前"がグローバルなスコープに宣言されていません。

C5283 (E) Qualified name is not allowed
限定名は許されません。

C5284 (E)(W) NULL reference is not allowed

NULL へのリファレンスは許されません。指定された通りに式を評価します。

C5285 (E) Initialization with "{...}" is not allowed for object of type "型"

"型"のオブジェクトに{}形式の初期化は許されません。

C5286 (E) Base class "型" is ambiguous

基底クラスの型が曖昧です。

C5287 (E) Derived class "型" contains more than one instance of class "型"

派生型が複数の同一クラス"型"を含みます。

C5288 (E) Cannot convert pointer to base class "型 1" to pointer to derived class "型 2" -- base class is virtual

仮想基底クラス"型 1"のポインタ型を派生クラス"型 2"のポインタ型に変換することはできません。

C5289 (E) No instance of constructor "名前" matches the argument list

コンストラクタ"名前"の引数が一致しません。

C5290 (E) Copy constructor for class "型" is ambiguous

クラス"型"のコピーコンストラクタが曖昧です。

C5291 (E) No default constructor exists for class "型"

クラス"型"のデフォルトコンストラクタは存在しません。

C5292 (E) "名前" is not a nonstatic data member or base class of class "型"

"名前"が非静的データメンバまたは基底クラス"型"でありません。

C5293 (E) Indirect nonvirtual base class is not allowed

仮想でない間接基底クラスは許されません。

C5294 (E) Invalid union member -- class "型" has a disallowed member function

union メンバに指定できないクラス"型"のメンバ関数があります。

C5296 (E)(W) Invalid use of non-lvalue array
左辺値でない配列の使用が不正です。

C5297 (E) Expected an operator
演算子が必要です。

C5298 (E) Inherited member is not allowed
継承されたメンバを使用することはできません。

C5299 (E) Cannot determine which instance of entity-kind "名前" is intended
オーバーロード関数の"名前"を決定できません。

C5300 (E)(W) A pointer to a bound function may only be used to call the function
メンバ関数へのポインタを関数呼び出し以外に使用しています。

C5301 (E) Typedef name has already been declared (with same type)
typdef の名前がすでに同じタイプで定義されています。

C5302 (E) Entity-kind "名前" has already been defined
関数"名前"はすでに定義されています。

C5304 (E) No instance of entity-kind "名前" matches the argument list
関数"名前"の引数が一致しません。

C5305 (E) Type definition is not allowed in function return type declaration
関数のリターン型の宣言で型の定義をすることはできません。

C5306 (E) Default argument not at end of parameter list
デフォルト引数の宣言がパラメータリストの最後ではありません。

C5307 (E) Redefinition of default argument
デフォルト引数を再定義しています。

C5308 (E) More than one instance of "名前" matches the argument list:
引数リストが一致するためオーバーロード関数"名前"が曖昧です。

C5309 (E) More than one instance of constructor "名前" matches the argument list:
引数リストが一致するためコンストラクタ"名前"があいまいです。

C5310 (E) Default argument of type "型1" is incompatible with parameter of type "型2"
デフォルト値の"型1"が引数の"型2"に合致しません。

C5311 (E) Cannot overload functions distinguished by return type alone
リターン型が異なる関数をオーバーロードすることはできません。

C5312 (E) No suitable user-defined conversion from "型1" to "型2" exists
適切な利用者定義変換"型1"から"型2"が存在しません。

C5313 (E) Type qualifier is not allowed on this function
関数に型限定子(const,volatile)を指定することはできません。

C5314 (E) Only nonstatic member functions may be virtual
静的メンバ関数に virtual を指定しています。

C5315 (E) The object has cv-qualifiers that are not compatible with the member function
オブジェクトの型限定子(const,volatile)がメンバ関数の型限定子と合致しません。

C5316 (E) Program too large to compile (too many virtual functions)
仮想関数の数が多すぎます。

C5317 (E) Return type is not identical to nor covariant with return type "型" of overridden virtual function entity-kind "名前"
仮想関数"名前"のリターン型"型"が異なります。

C5318 (E) Override of virtual entity-kind "名前" is ambiguous
仮想関数"名前"の置き換えが曖昧です。

C5319 (E) Pure specifier ("= 0") allowed only on virtual functions
純粋指定子"=0"を仮想関数以外に指定しています。

C5320 (E) Badly-formed pure specifier (only "= 0" is allowed)
純粋指定子の形式が正しくありません。"=0"だけが許されます。

C5321 (E) Data member initializer is not allowed
データメンバの初期化指定が正しくありません。

C5322 (E) Object of abstract class type "型" is not allowed:
抽象クラス"型"のオブジェクトは定義できません。

C5323 (E) Function returning abstract class "型" is not allowed:
抽象クラス"型"を返す関数は定義できません。

C5324 (I) Duplicate friend declaration
フレンド宣言が重複して指定されています。

C5325 (E) Inline specifier allowed on function declarations only
inline 指定子は関数宣言でのみ有効です。

C5326 (E)(W) "inline" is not allowed
inline 指定は許されません。

C5327 (E) Invalid storage class for an inline function
inline 関数の記憶クラスが不正です。

C5328 (E) Invalid storage class for a class member
クラスメンバの記憶クラスが不正です。

C5329 (E) Local class member entity-kind "名前" requires a definition
局所クラスメンバ"名前"の定義がありません。

C5330 (E) Entity-kind "名前" is inaccessible
"名前"をアクセスできません。

C5332 (E) Class "型" has no copy constructor to copy a const object
クラス"型"に const 型オブジェクトをコピーするコピーコンストラクタがありません。

C5333 (E) Defining an implicitly declared member function is not allowed
暗黙宣言されたメンバ関数を定義することはできません。

C5334 (E) Class "型" has no suitable copy constructor
クラス"型"に適切なコピーコンストラクタが存在しません。

C5335 (E)(W) Linkage specification is not allowed
リンケージ指定子を指定することはできません。

C5336 (E) Unknown external linkage specification
認識できないリンケージ指定が指定されました。

C5337 (E) Linkage specification is incompatible with previous "名前" (declared at line "行番号")
前に指定されたリンケージ指定子"名前"と合致しません。

C5338 (E) More than one instance of overloaded function "名前" has "C" linkage
C リンケージを持ったオーバーロード関数"名前"が複数あります。

C5339 (E) Class "型" has more than one default constructor
クラス"型"は複数のデフォルトコンストラクタを持っています。

C5340 (E) Value copied to temporary, reference to temporary used
値がローカルな領域にコピーされました。ローカルな領域への参照が使用されます。

C5341 (E) "operator 演算子" must be a member function
演算子関数"演算子"はメンバ関数でなければなりません。

C5342 (E) Operator may not be a static member function
静的メンバ関数の演算子関数は許されません。

C5343 (E) No arguments allowed on user-defined conversion
利用者定義変換に引数は許されません。

C5344 (E) Too many parameters for this operator function
演算子関数の引数の数が多すぎます。

C5345 (E) Too few parameters for this operator function
演算子関数の引数の数が足りません。

C5346 (E) Nonmember operator requires a parameter with class type
メンバ関数でない演算子関数はクラス型を引数に持つ必要があります。

C5347 (E) Default argument is not allowed
デフォルト引数は許されません。

C5348 (E) More than one user-defined conversion from "型1" to "型2" applies:
"型1"から"型2"への利用者定義型変換があいまいです。

C5349 (E) No operator "演算子" matches these operands
演算子関数"演算子"のオペランドが一致しません。

C5350 (E) More than one operator "演算子" matches these operands:
演算子関数"演算子"のオペランドが曖昧です。

C5351 (E) First parameter of allocation function must be of type "size_t"
operator new の第1パラメータは size_t 型でなければなりません。

C5352 (E) Allocation function requires "void *" return type
operator new のリターン型は void *型でなければなりません。

C5353 (E) Deallocation function requires "void" return type
operator delete のリターン型は void 型でなければなりません。

C5354 (E) First parameter of deallocation function must be of type "void *"
operator delete の第1パラメータは void *型でなければなりません。

C5356 (E) Type must be an object type
型はオブジェクト型でなければなりません。

C5357 (E) Base class "型" has already been initialized
基底クラスはすでに初期化されています。

C5359 (E) Entity-kind "名前" has already been initialized
"名前"はすでに初期化されています。

C5360 (E) Name of member or base class is missing
メンバ名または基底クラスに誤りがあります。

C5363 (E) Invalid anonymous union -- nonpublic member is not allowed
無名 union のメンバが公開メンバでありません。

C5364 (E) Invalid anonymous union -- member function is not allowed
無名 union にメンバ関数は許されません。

C5365 (E) Anonymous union at global or namespace scope must be declared static
グローバルまたは namespace スコープの無名 union は static 宣言が必要です。

C5366 (E) Entity-kind "名前" provides no initializer for:
"名前"に初期化指定はできません。

C5367 (E) Implicitly generated constructor for class "型" cannot initialize:
暗黙に生成されたクラス"型"のコンストラクタを初期化することはできません。

C5368 (W) Entity-kind "名前" defines no constructor to initialize the following:
"名前"は初期化のためのコンストラクタを定義していません。

C5369 (E) Entity-kind "名前" has an uninitialized const or reference member
"名前"の const またはリファレンスメンバが初期化されていません。

C5370 (W) Entity-kind "名前" has an uninitialized const field
"名前"の const フィールドが初期化されていません。

C5371 (E) Class "型" has no assignment operator to copy a const object
const オブジェクトをコピーするクラス"型"の代入演算子関数が定義されていません。

C5372 (E) Class "型" has no suitable assignment operator
クラス"型"に適当な代入演算が定義されていません。

C5373 (E) Ambiguous assignment operator for class "型"
クラス"型"の代入演算子関数があいまいです。

C5375 (E) Declaration requires a typedef name
typedef 名の宣言が必要です。

C5377 (W) "virtual" is not allowed
virtual を指定することはできません。

C5378 (E) "static" is not allowed
static を指定することはできません。

C5380 (E) Expression must have pointer-to-member type
式はメンバへのポインタ型でなければなりません。

C5381 (I) Extra ";" ignored
余分な";"を無視します。

C5382 (W) In-class initializer for nonstatic member is nonstandard
非スタティックなメンバを初期化するのは標準形式ではありません。

C5384 (E) No instance of overloaded "名前" matches the argument list
オーバーロード関数"名前"の引数リストが一致しません。

C5386 (E) No instance of entity-kind "名前" matches the required type
要求される型のオーバーロード関数"名前"がありません。

C5388 (E) "operator->" for class "型 1" returns invalid type "型 2"
クラス"型 1"の operator->演算関数のリターン型"型 2"が正しくありません。

C5389 (E) A cast to abstract class "型" is not allowed:
抽象クラス"型"へのキャストは許されません。

C5390 (E) Function "main" may not be called or have its address taken
main 関数の呼び出し、またはアドレスの取得を行ってはいけません。

C5391 (E) A new-initializer may not be specified for an array
配列を new によって初期化することはできません。

C5392 (E) Member function "名前" may not be redeclared outside its class
メンバ関数"名前"がクラスの外側で再宣言されました。

C5393 (E) Pointer to incomplete class type is not allowed
不完全クラスへのポインタ型は許されません。

C5394 (E) Reference to local variable of enclosing function is not allowed
ローカルクラスを囲む関数の局所変数へのリファレンスは許されません。

C5397 (E) Implicitly generated assignment operator cannot copy:
暗黙に生成された代入演算子関数がオブジェクトを正しくコピーすることができません。

C5398 (W) Cast to array type is nonstandard (treated as cast to “型”)
配列型へのキャストは標準形式ではありません。(“型”へのキャストと仮定します)

C5399 (I) Entity-kind "名前" has an operator newxxxx() but no default operator deletexxxx()
"名前"が operator new を持ちますがデフォルトの operator delete を持ちません。

C5400 (I) Entity-kind "名前" has a default operator deletexxxx() but no operator newxxxx()
"名前"がデフォルトの operator delete を持ちますが operator new を持ちません。

C5401 (E) Destructor for base class "型" is not virtual
基底クラス"型"のデストラクタが virtual でありません。

C5403 (E) Invalid redeclaration of member “関数名”
メンバ関数の不正な再宣言です。

C5404 (E) Function "main" may not be declared inline
main 関数を inline 宣言することはできません。

C5405 (E) Member function with the same name as its class must be a constructor
クラス名と同じ名前のメンバ関数はコンストラクタでなければなりません。

C5407 (E) A destructor may not have parameters
デストラクタは引数を持つことができません。

C5408 (E) Copy constructor for class "型1" may not have a parameter of type "型2"
クラス"型1"のコピーコンストラクタは"型2"の引数を持つことはできません。

C5409 (E) Entity-kind "名前" returns incomplete type "型"
関数"名前"のリターン型が不完全型"型"です。

C5410 (E) Protected entity-kind "名前" is not accessible through a "型" pointer or object
限定公開名"名前"は"型"へのポインタやオブジェクトを経由してアクセスすることはできません。

C5411 (E) A parameter is not allowed
仮引数は許されません。

C5412 (E) An "asm" declaration is not allowed here
asm 宣言は許されません。

C5413 (E) No suitable conversion function from "型1" to "型2" exists
"型1"から"型2"への適切な変換関数が存在しません。

C5414 (W) Delete of pointer to incomplete class
不完全型クラスへのポインタは削除されました。

C5415 (E) No suitable constructor exists to convert from "型1" to "型2"
"型1"から"型2"へ変換する適切なコンストラクタが存在しません。

C5416 (E) More than one constructor applies to convert from "型1" to
"型2":
"型1"から"型2"へ変換するコンストラクタが曖昧です。

C5417 (E) More than one conversion function from "型1" to "型2" applies:
"型1"から"型2"への変換関数が曖昧です。

C5418 (E) More than one conversion function from "型" to a built-in type applies:
"型"から組み込み型への変換関数が曖昧です。

C5424 (E) A constructor or destructor may not have its address taken
コンストラクタまたはデストラクタのアドレスを参照することはできません。

C5427 (E) Qualified name is not allowed in member declaration
限定名をメンバ宣言のなかで使用できません。

C5429 (E) The size of an array in "new" must be non-negative
new で指定された配列のサイズに負の値は許されません。

C5430 (W) Returning reference to local temporary
関数内にローカルな領域のリファレンスをリターン値にしています。

C5432 (E) "enum" declaration is not allowed
列挙型宣言は許されません。

C5433 (E) Qualifiers dropped in binding reference of type "型 1" to initializer of type "型 2"
const/volatile 限定の型"型 2"が参照型"型 1"の初期値に指定されました。

C5434 (E) A reference of type "型 1" (not const-qualified) cannot be initialized with a value of type "型 2"
const 型修飾されない型"型 1"へのリファレンスを"型 2"の値で初期化できません。

C5435 (E) A pointer to function may not be deleted
関数へのポインタを削除することはできません。

C5436 (E) Conversion function must be a nonstatic member function
変換関数は非静的メンバ関数でなければなりません。

C5437 (E) Template declaration is not allowed here
このスコープ内でテンプレート宣言は許されません。

C5438 (E) Expected a "<"
"<"が必要です。

C5439 (E) Expected a ">"
">"が必要です。

C5440 (E) Template parameter declaration is missing
テンプレートの引数宣言が正しくありません。

C5441 (E) Argument list for entity-kind "名前" is missing
テンプレート"名前"の実引数リストが正しくありません。

C5442 (E) Too few arguments for entity-kind "名前"
テンプレート"名前"の実引数が足りません。

C5443 (E) Too many arguments for entity-kind "名前"
テンプレートの実引数が多すぎます。

C5445 (E) Entity-kind "名前 1" is not used in declaring the parameter types of entity-kind "名前 2"
テンプレート"名前 2"の引数"名前 1"が使用されません。

C5449 (E) More than one instance of entity-kind "名前" matches the required type
オーバーロード関数"名前"が曖昧です。

C5450 (E) The type "long long" is nonstandard
long long 型は標準形式ではありません。

C5451 (E) Omission of “class” is nonstandard
“class”の無い friend 宣言は標準形式ではありません。

C5452 (E) Return type may not be specified on a conversion function
変換関数のリターン型が指定されていません。

C5456 (E) Excessive recursion at instantiation of entity-kind "名前"
テンプレート"名前"のインスタンスが再帰的に生成されます。

C5457 (E) "名前" is not a function or static data member
"名前"が関数または静的データメンバではありません。

C5458 (E) Argument of type "型 1" is incompatible with template parameter of type "型 2"
実引数の型"型 1"がテンプレートの引数"型 2"に合致しません。

C5459 (E) Initialization requiring a temporary or conversion is not allowed
初期化にテンポラリや変換を要求することは許されません。

C5460 (W) Declaration of “変数名” hides function parameter
関数内の変数宣言が関数の引数を隠しました。

C5461 (E) Initial value of reference to non-const must be an lvalue
const 型を持たないリファレンスの初期値は左辺値でなければなりません。

C5463 (E) "template" is not allowed
"template"指定は許されません。

C5464 (E) "型" is not a class template
"型"がクラステンプレートではありません。

C5466 (E) "main" is not a valid name for a function template
"main"は関数テンプレートの名前に使用できません。

C5467 (E) Invalid reference to entity-kind "名前" (union/nonunion mismatch)
"名前"の参照が不正です。

C5468 (E) A template argument may not reference a local type
テンプレートの実引数はローカルな型を参照できません。

C5469 (E) Tag kind of "名前 1" is incompatible with declaration of entity-kind "名前 2" (declared at line "行番号")
タグ名"名前 1"の種類と"名前 2"の宣言が合致しません。

C5470 (E) The global scope has no tag named "名前"
グローバルスコープにタグ名"名前"がありません。

C5471 (E) Entity-kind "名前 1" has no tag member named "名前 2"
"名前 1"はタグメンバ"名前 2"を持ちません。

C5473 (E) Entity-kind "名前" may be used only in pointer-to-member declaration
typedef 名"名前"はメンバへのポインタ型の宣言の中で使用されなければなりません。

C5475 (E) A template argument may not reference a non-external entity
テンプレートの実引数は外部名以外を参照できません。

C5476 (E) Name followed by "::~" must be a class name or a type name
::~に続く名前はクラス名または型名でなければなりません。

C5477 (E) Destructor name does not match name of class "型"
クラス名"型"とデストラクタ名が合致しません。

C5478 (E) Type used as destructor name does not match type "型"
デストラクタ名で使われた型と"型"が合致しません。

C5479 (I) Entity-kind "名前" redeclared "inline" after being called
関数が呼ばれたあとに inline"名前"を宣言しています。以降 inline 指定を有効にします。

C5481 (E) Invalid storage class for a template declaration
テンプレート宣言の記憶クラス指定が正しくありません。

C5484 (E) Invalid explicit instantiation declaration
テンプレートの実引数が不正です。

C5485 (E) Entity-kind "名前" is not an entity that can be instantiated
テンプレート"名前"を実体化できません。

C5486 (E) Compiler generated entity-kind "名前" cannot be explicitly instantiated
コンパイラが生成した関数を実体化することはできません。

C5487 (E)(I) Inline entity-kind "名前" cannot be explicitly instantiated
インライン関数"名前"を実体化することはできません。

C5489 (E) Entity-kind "名前" cannot be instantiated -- no template definition was supplied
テンプレート定義がないため"名前"を実体化することはできません。

C5490 (E) Entity-kind "名前" cannot be instantiated -- it has been explicitly specialized
"名前"を実体化することはできません。

C5493 (E) No instance of entity-kind "名前" matches the specified type
オーバーロード関数"名前"と指定された型が合致しません。

C5494 (E)(W) Declaring a void parameter list with a typedef is nonstandard
typedef された void パラメータリストを宣言するのは標準形式ではありません。

C5496 (E) Template parameter "名前" may not be redeclared in this scope
テンプレート引数"名前"がスコープ内で再宣言されています。

C5497 (W) Declaration of "名前" hides template parameter
"名前"の宣言はテンプレート引数を隠蔽します。

C5498 (E) Template argument list must match the parameter list
テンプレート実引数と仮引数が合致しません。

C5500 (E) Extra parameter of postfix "operatorxxxx" must be of type "int"
後置演算関数の第 2 パラメータの型は int 型でなければなりません。

C5501 (E) An operator name must be declared as a function
演算子名は関数として宣言しなければなりません。

C5502 (E) Operator name is not allowed
演算子名は許されません。

C5503 (E) Entity-kind "名前" cannot be specialized in the current scope
スコープ内で"名前"が曖昧です。

C5504 (E) Nonstandard form for taking the address of a member function
メンバ関数のアドレスを取得するのは標準形式ではありません。

C5505 (E) Too few template parameters -- does not match previous declaration
テンプレートの引数が足りません。

C5506 (E) Too many template parameters -- does not match previous declaration
テンプレートの引数が多すぎます。

C5507 (E) Function template for operator delete(void *) is not allowed
operator delete(void *)の関数テンプレートは許されません。

C5508 (E) Class template and template parameter may not have the same name
クラステンプレートとテンプレートの引数が同じ名前です。

C5510 (E) A template argument may not reference an unnamed type
テンプレートの実引数が名前付けされていない型を参照しています。

C5511 (E) Enumerated type is not allowed
列挙型は許されません。

C5512 (W) Type qualifier on a reference type is not allowed
リファレンス型に const/volatile 修飾を指定することはできません。

C5513 (E)(W) A value of type "型 1" cannot be assigned to an entity of type "型 2"
型不一致のため"型 1"の値を"型 2"の実体に代入することができません。
(W) 型 1 と型 2 がそれぞれ、互いに互換性のない型へのポインタ
(E) 型 1 と型 2 が、互いに互換性のない型

C5514 (W) Pointless comparison of unsigned integer with a negative constant
負の定数と符号なし整数を比較しています。

C5515 (E) Cannot convert to incomplete class "型"
不完全型"型"への型変換はできません。

C5516 (E) Const object requires an initializer
const 型のオブジェクトには初期値が必要です。

C5517 (E) Object has an uninitialized const or reference member
オブジェクトが未初期化の const 型メンバあるいはリファレンス型メンバを持ちます。

C5518 (E) Nonstandard preprocessing directive
標準形式ではないプリプロセッサのキーワードがあります。

C5519 (E) Entity-kind "名前" may not have a template argument list
"名前"はテンプレート実引数を持つことができません。

C5520 (E)(W) Initialization with "{...}" expected for aggregate object
集成型のオブジェクトは{...}の形式で初期化しなければなりません。

C5521 (E) Pointer-to-member selection class types are incompatible ("型1" and "型2")
メンバへのポインタ型のクラスの型が"型1"と"型2"で合致しません。

C5522 (W) Pointless friend declaration
自分自身へのフレンド宣言をしています。

C5523 (W) "." used in place of "::" to form a qualified name
“.” が スコープ解決子 ”::” の代わりに使用されています。

C5525 (W) A dependent statement may not be a declaration
条件式はスコープを持ちません。

C5526 (E) A parameter may not have void type
void 型の引数は指定できません。

C5529 (E) This operator is not allowed in a template argument expression
テンプレートの実引数式に指定された演算は許されません。

C5530 (E) Try block requires at least one handler
try 文に対応する catch 文がありません。

C5531 (E) Handler requires an exception declaration
catch 文の(...)には例外宣言が必要です。

C5532 (E) Handler is masked by default handler
デフォルトハンドラによってハンドラがマスクされました。

C5533 (W) Handler is potentially masked by previous handler for type "型"
"型"を持つ前のハンドラによってハンドラがマスクされる可能性があります。

C5534 (I) Use of a local type to specify an exception
ローカルな型を使用した例外処理が指定されています。

C5535 (I) Redundant type in exception specification
例外処理中に冗長な型の指定があります。

C5536 (E) Exception specification is incompatible with that of previous entity-kind "名前" (declared at line "行番号"):
例外処理指定が前の指定"名前"と合致しません。

C5540 (E) Support for exception handling is disabled
例外処理を行うオプション(exception)が指定されていません。

C5541 (W) Omission of exception specification is incompatible with previous entity-kind "名前" (declared at line "行番号")
例外処理の省略形が前の"名前"と合致しません。

C5542 (F) Could not create instantiation request file "名前"
テンプレートを実体化するのに使用するファイル"名前"を作成することができませんでした。

C5543 (E) Non-arithmetic operation not allowed in nontype template argument
対応するテンプレートの実引数に非算術型変換は許されません。

C5544 (E) Use of a local type to declare a nonlocal variable
ローカルでない変数にローカルな型を指定しています。

C5545 (E) Use of a local type to declare a function
関数宣言にローカルな型を指定しています。

C5546 (E) Transfer of control bypasses initialization of:
初期化処理が行われません。

C5548 (E) Transfer of control into an exception handler
例外ハンドラ処理が実行されます。

C5549 (I) Entity-kind "名前" is used before its value is set
"名前"に値を設定する前に使用しています。

C5550 (W) Entity-kind "名前" was set but never used
"名前"が使用されませんでした。

C5551 (E) Entity-kind "名前" cannot be defined in the current scope
"名前"はこのスコープ内で定義できません。

C5552 (W) Exception specification is not allowed
例外処理指定は許されません。例外処理を無効にします。

C5553 (W) External/internal linkage conflict for entity-kind "名前" (declared at line "行番号")
"名前"の外部/内部リンケージ指定が衝突します。外部リンケージを設定します。

C5554 (W) Entity-kind "名前" will not be called for implicit or explicit conversions
変換関数"名前"は暗黙的にも明示的にも呼ばれることはありません。

C5555 (E) Tag kind of "名前" is incompatible with template parameter of type "型"
タグ"名前"の種類とテンプレートの引数の"型"が合致しません。

C5556 (E) Function template for operator new(size_t) is not allowed
operator new(size_t)の関数テンプレートは許されません。

C5558 (E) Pointer to member of type "型" is not allowed
メンバへのポインタ型"型"が誤っています。

C5559 (E) Ellipsis is not allowed in operator function parameter list
省略指定(...)は演算子関数の引数リストに指定できません。

C5560 (E) “キーワード” is reserved for future use as a keyword
キーワードは将来実装される予約語です。

C5563 (F) Invalid preprocessor output file
プリプロセッサ出力に使用できないファイル名です。

C5598 (E) A template parameter may not have void type
テンプレートの引数に void 型は指定できません。

C5599 (E) Excessive recursive instantiation of entity-kind “名前” due to instantiate-all mode

instantiate-all モードの指定によってテンプレート”名前”のインスタンスが再帰的に生成されます。

C5601 (E) A throw expression may not have void type

throw 式に void 型は指定できません。

C5603 (E) Parameter of abstract class type "型" is not allowed:

抽象クラス"型"の引数は許されません。

C5604 (E) Array of abstract class "型" is not allowed:

抽象クラス"型"の配列は許されません。

C5605 (E) Floating-point template parameter is nonstandard

浮動小数点のテンプレートパラメータは標準形式ではありません。

C5606 (E) This pragma must immediately precede a declaration

この pragma は宣言の前に記述しなければいけません。

C5607 (E) This pragma must immediately precede a statement

この pragma は式の直前に記述しなければいけません。

C5608 (E) This pragma must immediately precede a declaration or statement

この pragma は宣言または式の直前に記述しなければいけません。

C5609 (E) This kind of pragma may not be used here

この種類の pragma はここで使用してはいけません。

C5611 (W) Overloaded virtual function "名前 1" is only partially overridden in entity-kind "名前 2"

"名前 1"のオーバーロード仮想関数は"名前 2"の中で一部の仮想関数だけが置き換えの対象になります。指定された通りに処理を継続します。

C5612 (E) Specific definition of inline template function must precede its first use

インライン指定されたテンプレート関数は呼び出しの前に定義しなければなりません。

C5615 (E) Parameter type involves pointer to array of unknown bound
引数の型に要素数の指定がない配列へのポインタがふくまれています。

C5616 (E) Parameter type involves reference to array of unknown bound
引数の型に要素数の指定がない配列への参照が含まれています。

C5617 (W) Pointer-to-member-function cast to pointer to function
メンバ関数ポインタを関数ポインタにキャストしています。

C5618 (I) Struct or union declares no named members
構造体または共用体に名前付きのメンバが含まれていません。

C5619 (E) Nonstandard unnamed field
標準形式ではない名前の無いフィールドです。

C5620 (E) Nonstandard unnamed member
標準形式ではない名前の名前の無いメンバです。

C5624 (E) "名前" is not a type name
"名前"は型の名前ではありません。

C5641 (F) "名前" is not a valid directory
"名前"が正しいフォルダでありません。

C5642 (F) Cannot build temporary file name
コンパイラが使用するテンポラリファイルを作成できません。

C5643 (E) “restrict” is not allowed
“restrict”を指定することはできません。

C5644 (E) A pointer or reference to function type may not be qualified by "restrict"
関数へのポインタまたは参照型は”restrict”によって修飾してはいけません。

C5647 (E) Conflicting calling convention modifiers
呼び出し規約修飾子が競合しています。

C5650 (W) Calling convention specified here is ignored
ここで指定された呼び出し規約は無視されます。

C5651 (E) A calling convention may not be followed by a nested declarator
呼び出し規約の後にネストされた宣言子が続いてはいけません。

C5652 (I) Calling convention is ignored for this type
この型に対する呼び出し規約は無視されます。

C5654 (E) Declaration modifiers are incompatible with previous declaration
宣言子が前に宣言されたものと互換性がありません。

C5656 (E) Transfer of control into a try block
外側のブロックから try ブロックに制御が移ります。

C5657 (W) Inline specification is incompatible with previous "名前" (declared at line "行番号")
インライン指定が前の宣言"名前"と合致しません。

C5658 (E) Closing brace of template definition not found
テンプレート定義の閉じ括弧がありません。

C5660 (E) Invalid packing alignment value
pack の値が不正です。

C5661 (E) Expected an integer constant
整数定数がありません。

C5662 (W) Call of pure virtual function
純粋仮想関数が関数を呼び出しています。

C5663 (E) Invalid source file identifier string
#pragma 指定の構文に誤りがあります。

C5664 (E) A class template cannot be defined in a friend declaration
フレンド宣言内でクラステンプレートを定義することはできません。

C5665 (E) "asm" is not allowed
asm 指定子は使用できません。

C5666 (E) "asm" must be used with a function definition
asm 指定子は関数定義と共に指定してください。

C5667 (E) "asm" function is nonstandard
asm 関数は標準形式ではありません。

C5668 (E) Ellipsis with no explicit parameters is nonstandard
省略指定(...)のみのパラメータは標準形式ではありません。

C5669 (E) "&..." is nonstandard
“&...” のパラメータは標準形式ではありません。

C5670 (E) Invalid use of "&..."
“&...” が不正に使われています。

C5673 (E) A reference of type "型 1" cannot be initialized with a value of type "型 2"
const/volatile 型"型 1"のリファレンスは"型 2"の値で初期化できません。

C5674 (E) Initial value of reference to const volatile must be an lvalue
const/volatile 型のリファレンスの初期値は左辺値でなければなりません。

C5676 (W) Using out-of-scope declaration of “シンボル名”
Using 宣言がシンボルのスコープ外です。

C5678 (I) Call of entity-kind "名前" (declared at line "行番号") cannot be inlined
関数呼び出し"名前"がインライン展開されませんでした。

C5679 (I) Entity-kind "名前" cannot be inlined
関数"名前"はインライン展開されません。

C5691 (E)(W) “シンボル”, required for copy that was eliminated, is inaccessible
コピーコンストラクタにアクセスできません。

C5692 (E)(W) “シンボル”, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue

コピーコンストラクタを呼び出すことができません。

C5693 (E) <typeinfo> must be included before typeid is used

typeid を使うためには<typeinfo>をインクルードしなければなりません。

C5694 (E) "名前" cannot cast away const or other type qualifiers

"名前"のキャストの結果 const などの属性がなくなります。

C5695 (E) The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *

dynamic_cast の型は完全クラス型へのポインタ型またはリファレンス型か void *型でなければなりません。

C5696 (E) The operand of a pointer dynamic_cast must be a pointer to a complete class type

dynamic_cast ポインタのオペランドは完全クラス型へのポインタ型でなければなりません。

C5697 (E) The operand of a reference dynamic_cast must be an lvalue of a complete class type

dynamic_cast のリファレンスのオペランドは完全クラス型の左辺値でなければなりません。

C5698 (E) The operand of a runtime dynamic_cast must have a polymorphic class type

実行時 dynamic_cast のオペランドはポリモフィックなクラス型でなければなりません。

C5701 (E) An array type is not allowed here

配列型は許されません。

C5702 (E) Expected an "="

代入式が必要です。

C5703 (E) Expected a declarator in condition declaration

宣言子が必要です。

C5704 (E) "名前", declared in condition, may not be redeclared in this scope

このスコープ内で"名前"を再宣言することはできません。

C5705 (E) Default template arguments are not allowed for function templates
関数テンプレートにデフォルトの実引数を指定することはできません。

C5706 (E) Expected a "," or ">"
","または">"が必要です。

C5707 (E) Expected a template parameter list
テンプレートの引数リストが必要です。

C5708 (W) Incrementing a bool value is deprecated
bool 型の値をインクリメントしています。値をインクリメントして処理を継続します。

C5709 (E) bool type is not allowed
bool 型の値をデクリメントすることはできません。

C5710 (E) Offset of base class "名前 1" within class "名前 2" is too large
クラス"名前 2"内の基底クラス"名前 1"のサイズが大きすぎます。

C5711 (E) Expression must have bool type (or be convertible to bool)
式の型は bool 型か bool 型へ変換可能な型でなければなりません。

C5717 (E) The type in a const_cast must be a pointer, reference, or pointer to member to an object type
const_cast の型はポインタ型、リファレンス型またはメンバへのポインタ型でなければなりません。

C5718 (E) A const_cast can only adjust type qualifiers; it cannot change the underlying type
const_cast は const/volatile 以外の型を調整することはできません。

C5719 (E) mutable is not allowed
mutable の指定は許されません。

C5720 (W) Redeclaration of entity-kind "名前" is not allowed to alter its access
"名前"の再宣言でアクセス指定を変更することはできません。前の宣言のアクセス指定を有効にします。

C5722 (W) Use of alternative token "<:" appears to be unintended
2 文字表記"<:"が使用されました。"["と解釈します。

C5723 (W) Use of alternative token "%:" appears to be unintended
2 文字表記"%:"が使用されました。"#"と解釈します。

C5724 (E) namespace definition is not allowed
namespace の定義はファイルスコープまたは namespace スコープ内で許されます。

C5725 (E) Name must be a namespace name
namespace の名前が正しくありません。

C5726 (E) Namespace alias definition is not allowed
namespace の別名定義はここでは許されません。

C5727 (E) namespace-qualified name is required
namespace の限定名が要求されます。

C5728 (E) A namespace name is not allowed
namespace 名は許されません。

C5730 (E) Entity-kind "名前" is not a class template
"名前"はクラステンプレートのメンバではありません。

C5731 (E) Array with incomplete element type is nonstandard
不完全な要素型を持つ配列は標準形式ではありません。

C5732 (E) Allocation operator may not be declared in a namespace
operator new 関数が namespace 内で宣言されています。

C5733 (E) Deallocation operator may not be declared in a namespace
operator delete 関数が namespace 内で宣言されています。

C5734 (E) Entity-kind "名前 1" conflicts with using-declaration of entity-kind "名前 2"
名前"名前 1"が using 宣言名"名前 2"と衝突します。

C5735 (E) Using-declaration of entity-kind "名前 1" conflicts with entity-kind "名前 2" (declared at line "行番号")

using 宣言の名前が衝突します。

C5737 (W) Using-declaration ignored -- it refers to the current namespace

現在の namespace スコープの名前を using 宣言しています。using 宣言を無視します。

C5738 (E) A class-qualified name is required

クラスの限定名が要求されています。

C5741 (W) Using-declaration of entity-kind "名前" ignored

using 宣言"名前"は無効です。

C5742 (E) Entity-kind "名前 1" has no actual member "名前 2"

"名前 1"に"名前 2"のメンバは存在しません。

C5748 (W) Calling convention specified more than once

呼び出し規約が 1 回以上指定されています。

C5749 (E) A type qualifier is not allowed

型修飾子を指定できません。

C5750 (E) Entity-kind "名前" (declared at line "行番号") was used before its template was declared

"名前"はテンプレートが宣言される前に使われました。

C5751 (E) Static and nonstatic member functions with same parameter types cannot be overloaded

同じ引数の型を持つ静的メンバ関数と非静的メンバ関数はオーバーロードすることはできません。

C5752 (E) No prior declaration of entity-kind "名前"

namespace テンプレート関数"名前"の宣言がありません。

C5753 (E) A template-id is not allowed

ここではテンプレート(template 名<template 実引数>)は許されません。

C5754 (E) A class-qualified name is not allowed
ここではクラス限定名は許されません。

C5755 (E) Entity-kind "名前" may not be redeclared in the current scope
このスコープ内で"名前"を再宣言することはできません。

C5756 (E) Qualified name is not allowed in namespace member declaration
namespace メンバの宣言で指定された限定名は許されません。

C5757 (E) Entity-kind "名前" is not a type name
"名前"は型名ではありません。

C5758 (E) Explicit instantiation is not allowed in the current scope
現在のスコープ範囲でインスタンスを明示的に生成することはできません。

C5759 (E) “シンボル名” cannot be explicitly instantiated in the current scope
シンボルは現在のスコープで明示的にインスタンス化できません。

C5760 (W) “シンボル” explicitly instantiated more than once
シンボルを具現化できませんでした。

C5761 (E) Typename may only be used within a template
typename キーワードはテンプレート内でのみ使用できます。

C5765 (E) Nonstandard character at start of object-like macro definition
非標準の文字列がオブジェクト的マクロ定義の始まりに含まれています。

C5766 (W) Exception specification for virtual entity-kind "名前 1" is incompatible with
that of overridden entity-kind "名前 2"
仮想関数の例外指定"名前 1"が"名前 2"に合致しません。

C5767 (W) Conversion from pointer to smaller integer
ポインタをポインタサイズより小さい型に変換しています。

C5768 (W) Exception specification for implicitly declared virtual entity-kind "名前 1" is incompatible with that of overridden entity-kind "名前 2"

コンパイラが生成する暗黙の仮想関数"名前 1"の例外指定が"名前 2"に合致しません。

C5769 (E) “シンボル 1”, implicitly called from “シンボル 2”, is ambiguous

operator delete の呼び出しが曖昧です。

C5771 (E) "explicit" is not allowed

explicit はクラス宣言内のコンストラクタにのみ指定できます。

C5772 (E) Declaration conflicts with "名前" (reserved class name)

予約されたクラス名 type_info と衝突します。

C5773 (E) Only "()" is allowed as initializer for array entity-kind "名前"

配列"名前"の初期化指定が正しくありません。

C5774 (E) "virtual" is not allowed in a function template declaration

関数テンプレートに virtual 指定はできません。

C5775 (E) Invalid anonymous union -- class member template is not allowed

無名 union の指定が正しくありません。

C5776 (E) Template nesting depth does not match the previous declaration of entity-kind "名前"

テンプレートのパラメータのネストが前の宣言"名前"と合致しません。

C5777 (E) This declaration cannot have multiple "template <...>" clauses

この宣言に複数のテンプレート宣言はできません。

C5779 (E) "名前", declared in for-loop initialization, may not be redeclared in this scope

for 文の初期化式で宣言された"名前"をこのスコープ内で再宣言できません。

C5780 (W) Reference is to “シンボル 1” -- under old for-init scoping rules it would have been “シンボル 2”

“シンボル 1”を参照しています。

C5782 (E) Definition of virtual entity-kind "名前" is required here

仮想関数の定義"名前"が必要です。

C5783 (W) Empty comment interpreted as token-pasting operator "##"

空のコメントは字句連結オペレータ”##”と仮定します。

C5784 (E) A storage class is not allowed in a friend declaration

フレンド宣言に記憶クラスを指定することはできません。

C5785 (E) Template parameter list for "名前" is not allowed in this declaration

この宣言内に"名前"のテンプレートの引数並びは許されません。

C5786 (E) entity-kind "名前" is not a valid member class or function template

"名前"は有効なメンバまたは関数テンプレートではありません。

C5787 (E) Not a valid member class or function template declaration

有効なメンバまたは関数テンプレート宣言ではありません。

C5788 (E) A template declaration containing a template parameter list may not be followed by an explicit specialization declaration

テンプレート関数の定義の後にテンプレート引数並びを含むテンプレート宣言は指定できません。

C5789 (E) Explicit specialization of entity-kind "名前 1" must precede the first use of entity-kind "名前 2"

明示的なテンプレートの実体の定義"名前 1"は最初のテンプレート"名前 2"を使用する前になければなりません。

C5790 (E) Explicit specialization is not allowed in the current scope

明示的なテンプレートの実体の定義はこのスコープでは許されません。

C5791 (E) Partial specialization of entity-kind "名前" is not allowed

テンプレート"名前"の部分的な定義は許されません。

C5792 (E) Entity-kind "名前" is not an entity that can be explicitly specialized
"名前"はテンプレートのインスタンスではありません。

C5793 (E) Explicit specialization of entity-kind "名前" must precede its first use
明示的なテンプレートの実体"名前"の定義は最初の使用より前になければなりません。

C5794 (W) Template parameter "テンプレート引数" may not be used in an elaborated type specifier
class 指定にテンプレート引数を使用することはできません。class 指定を無効にしてテンプレートを有効にします。

C5795 (E) Specializing "名前" requires "template<>" syntax
"名前"のテンプレートの実体定義は tempalte<>形式が要求されます。

C5799 (E) Specializing “シンボル名” without "template<>" syntax is nonstandard
“template<>”なしでシンボルを特殊化するのは標準形式ではありません。

C5800 (E) This declaration may not have extern "C" linkage
この宣言は extern "C"リンケージを持つことはできません。

C5801 (E) "名前" is not a class or function template name in the current scope
"名前"はこのスコープ内ではクラステンプレートまたは関数テンプレートではありません。

C5802 (W) Specifying a default argument when redeclaring an unreferenced function template is nonstandard
未参照の関数テンプレートを再宣言するときにデフォルト引数を指定しています。デフォルト引数を無視します。

C5803 (E) Specifying a default argument when redeclaring an already referenced function template is not allowed
すでに参照された関数テンプレートを再宣言するときにデフォルト引数を指定しています。

C5804 (E) Cannot convert pointer to member of base class "型 1" to pointer to member of derived class "型 2" -- base class is virtual
仮想基底クラス"型 1"のメンバポインタを派生クラス"型 2"のメンバポインタに変換することはできません。

C5805 (E) Exception specification is incompatible with that of entity-kind "名前" (declared at line "行番号"):
throw 例外指定は"名前"の例外指定と合致しません。

C5806 (W) Omission of exception specification is incompatible with entity-kind "名前" (declared at line "行番号")
throw 例外指定の省略は"名前"の例外指定と合致しません。"名前"を有効にします。

C5807 (E) Unexpected end of default argument expression
デフォルト引数式が正しくありません。

C5808 (E) Default-initialization of reference is not allowed
リファレンス型のデフォルトの初期化は許されません。

C5809 (E) Uninitialized entity-kind "名前" has a const member
未初期化の"名前"が const 型メンバを持ちます。

C5810 (E) Uninitialized base class "型" has a const member
未初期化の基底クラス"型"が const 型メンバを持ちます。

C5811 (E) Const entity-kind "名前" requires an initializer -- class "型" has no explicitly declared default constructor
const 型の"名前"には初期化指定が必要です。クラス"型"が明示的に宣言されたデフォルトコンストラクタを持ちません。

C5812 (E)(W) Const object requires an initializer -- class "型" has no explicitly declared default constructor
const 型オブジェクトには初期化指定が必要です。クラス"型"が明示的に宣言されたデフォルトコンストラクタを持ちません。

C5815 (I) Type qualifier on return type is meaningless
テンプレートで実体化されるリターン型に意味のない修飾型を指定しています。修飾型を有効にします。

C5816 (E) In a function definition a type qualifier on a "void" return type is not allowed
関数定義において“void”型の戻り値に型修飾子を指定することはできません。

C5817 (E) Static data member declaration is not allowed in this class
局所クラスは静的データメンバを持つことはできません。

C5818 (E) Template instantiation resulted in an invalid function declaration
テンプレートで実体化された関数宣言が正しくありません。

C5819 (E) "..." is not allowed
“...” は使用できません。

C5822 (E) Invalid destructor name for type "型"
"型"のデストラクタ名が正しくありません。

C5824 (E) Destructor reference is ambiguous -- both entity-kind "名前 1" and entity-kind "名前 2" could be used
"名前 1"と"名前 2"が使われました。デストラクタの参照があいまいです。

C5825 (W) Virtual inline entity-kind "名前" was never defined
仮想インラインメンバ関数"名前"の定義がありません。

C5826 (W) Entity-kind "名前" was never referenced
関数の引数"名前"は参照されません。

C5827 (E) Only one member of a union may be specified in a constructor initializer list
共用体の一つのメンバのみをコンストラクタの初期化で指定できます。

C5828 (E) Support for "new[]" and "delete[]" is disabled
new[] と delete[] はサポートされていません。

C5829 (W) "double" used for "long double" in generated C code
C コード生成時に ”long double” は “double” に変換されます。

C5830 (W) "シンボル” has no corresponding operator deletes (to be called if an exception is thrown during initialization of an allocated object)
対応する operator delete がありません。

C5831 (W)(I) Support for placement delete is disabled
operator delete 関数の型が正しくありません。処理を継続します。

C5832 (E) No appropriate operator delete is visible
適当な operator delete 関数が見つかりません。

C5833 (E) Pointer or reference to incomplete type is not allowed
不完全型へのポインタまたはリファレンス型は許されません。

C5834 (E) Invalid partial specialization -- entity-kind "名前" is already fully specialized
すでに特別化された"名前"を部分特別化しています。

C5835 (E) Incompatible exception specifications
例外指定の型が合致しません。

C5836 (W) Returning reference to local variable
局所変数のリファレンスをリターン値に指定しています。指定された処理を継続します。

C5837 (W) Omission of explicit type is nonstandard ("int" assumed)
型指定がありません。int 型を仮定します。

C5838 (E) More than one partial specialization matches the template argument list of entity-kind "名前"
部分特別化テンプレート"名前"のテンプレート実引数があいまいです。

C5840 (E) A template argument list is not allowed in a declaration of a primary template
プライマリテンプレート宣言にテンプレート実引数は指定できません。

C5841 (E) Partial specializations may not have default template arguments
部分特別化テンプレートはデフォルトのテンプレート引数を持つことはできません。

C5842 (E) Entity-kind "名前 1" is not used in template argument list of entity-kind "名前 2"
部分特別化テンプレート"名前 1"は"名前 2"のテンプレート実引数に使用されません。

C5843 (E) The type of partial specialization template parameter entity-kind "名前" depends on another template parameter

部分特別化テンプレート"名前"のテンプレート仮引数が別のテンプレート仮引数に依存しています。

C5844 (E) The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter

部分特別化テンプレートのテンプレート実引数がテンプレート仮引数に依存する非型の実引数を含んでいます。

C5845 (E) This partial specialization would have been used to instantiate entity-kind "名前"

この部分特別化テンプレートはプライマリテンプレート"名前"を実体化しようとしています。

C5846 (E) This partial specialization would have been made the instantiation of entity-kind "名前" ambiguous

この部分特別化テンプレートは"名前"の実体化があいまいになります。

C5847 (E) Expression must have integral or enum type

式の型は整数型か列挙型でなければなりません。

C5848 (E) Expression must have arithmetic or enum type

式の型は算術型か列挙型でなければなりません。

C5849 (E) Expression must have arithmetic, enum, or pointer type

式の型は算術型、列挙型もしくはポインタ型でなければなりません。

C5850 (E) Type of cast must be integral or enum

キャストの型は整数型か列挙型でなければなりません。

C5851 (E) Type of cast must be arithmetic, enum, or pointer

キャストの型は算術型、列挙型もしくはポインタ型でなければなりません。

C5852 (E) Expression must be a pointer to a complete object type

式の型は完全オブジェクト型へのポインタ型でなければなりません。

C5854 (E) A partial specialization nontype argument must be the name of a nontype parameter or a constant

部分特別化テンプレートの非型テンプレート実引数は非型の仮引数名か定数でなければなりません。

C5855 (E)(W) Return type is not identical to return type "型" of overridden virtual function entity-kind "名前"

関数のリターン型がオーバーライドされた仮想関数"名前"のリターン型"型"と同一でありません。

C5857 (E) A partial specialization of a class template must be declared in the namespace of which it is a member

部分特別化テンプレートはそのメンバを含む namespace の中で宣言しなければなりません。

C5858 (E) Entity-kind "名前" is a pure virtual function

"名前"は純粋仮想関数です。

C5859 (E) Pure virtual entity-kind "名前" has no overrider

純粋仮想関数"名前"はオーバーライドされません。

C5861 (E) Invalid character in input line

行中に不正な文字が現れました。

C5862 (E) Function returns incomplete type "型"

関数のリターン型"型"が不完全型です。

C5863 (I) Effect of this "#pragma pack" directive is local to “シンボル”

#pragma pack ディレクティブの影響はシンボル内にとどまります。

C5864 (E) "名前" is not a template

"名前"はテンプレートでありません。

C5865 (E) A friend declaration may not declare a partial specialization

部分特別化テンプレートはフレンド宣言内で指定できません。

C5866 (I) Exception specification ignored

例外指定は無視されます。

C5867 (W) Declaration of "size_t" does not match the expected type "型"
size_t 型が期待する"型"と異なります。

C5868 (E) Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)
2 つのテンプレート実引数リストの最後に指定する">>"は間に空白が必要です。

C5869 (E) Could not set locale to allow processing of multibyte characters
多バイト文字にロケール設定ができませんでした。

C5870 (W) Invalid multibyte character sequence
不正な 2 バイト文字があります。

C5871 (E) Template instantiation resulted in unexpected function type of "型 1" (the meaning of a name may have changed since the template declaration -- the type of the template is "型 2")
"型 2"を持つテンプレートの実体化の結果、期待されない型"型 1"の関数が作られました。

C5872 (E) Ambiguous guiding declaration -- more than one function template no matches type “型”
テンプレート関数が曖昧です。

C5873 (E) Non-integral operation not allowed in nontype template argument
非型のテンプレート実引数に非整数型の演算は許されません。

C5875 (E) Embedded C++ does not support templates
Embedded C++仕様はテンプレート機能をサポートしません。

C5876 (E) Embedded C++ does not support exception handling
Embedded C++仕様は例外処理機能をサポートしません。

C5877 (E) Embedded C++ does not support namespaces
Embedded C++仕様は namespace 機能をサポートしません。

C5878 (E) Embedded C++ does not support run-time type information
Embedded C++仕様はランタイム型情報機能をサポートしません。

C5879 (E) Embedded C++ does not support the new cast syntax
Embedded C++仕様は新形式のキャスト機能をサポートしません。

C5880 (E) Embedded C++ does not support using-declarations
Embedded C++仕様は using 宣言機能をサポートしません。

C5881 (E) Embedded C++ does not support "mutable"
Embedded C++仕様は mutable 機能をサポートしません。

C5882 (E) Embedded C++ does not support multiple or virtual inheritance
Embedded C++仕様は多重継承/仮想継承機能をサポートしません。

C5885 (E) "型1" cannot be used to designate constructor for "型2"
"型1"はコンストラクタの"型2"で使用することはできません。

C5886 (E) Invalid suffix on integral constant
整数定数への接尾辞が不正です。

C5890 (E) Variable length array with unspecified bound is not allowed
可変長配列に大きさが指定されていません。

C5891 (E) An explicit template argument list is not allowed on this declaration
この宣言内では明示的なテンプレート実引数は許されません。

C5892 (E) An entity with linkage cannot have a type involving a variable length array
リンケージ指定子がある宣言は可変長配列を含む型を持つことはできません。

C5893 (E) A variable length array cannot have static storage duration
可変長配列は静的記憶期間を持つことができません。

C5894 (E) Entity-kind "名前" is not a template
"名前"はテンプレートではありません。

C5896 (E) Expected a template argument
テンプレートの実引数が期待されます。

C5898 (E) Nonmember operator requires a parameter with class or enum type
非メンバ演算子関数にはクラスまたは列挙型の仮引数が要求されます。

C5900 (E) Using-declaration of entity-kind "名前" is not allowed
"名前"の using 宣言は許されません。

C5901 (E) Qualifier of destructor name "型 1" does not match type "型 2"
"型 1"のデストラクタの限定名が"型 2"に一致しません。

C5902 (W) Type qualifier ignored
型限定名が不正です。型限定名を無効にします。

C5907 (E) Option "nonstd_qualifier_deduction" can be used only when compiling C++
“nonstd_qualifier_deduction”オプションは C++コンパイル時のみ使用できます。

C5912(W) Ambiguous class member reference – “シンボル 1” used in preference to “シンボル 2”
曖昧なクラスメンバの参照です。シンボル 1 をシンボル 2 に優先して参照します。

C5915 (E) A segment name has already been specified
すでに指定されたセグメント名です。

C5916 (E) Cannot convert pointer to member of derived class "型 1" to pointer to member of base class "型 2" -- base class is virtual
派生クラス"型 1"のメンバへのポインタ型を仮想基底クラス"型 2"のメンバへのポインタ型に変換できません。

C5919 (F) Invalid output file: "名前"
テンプレート情報ファイルの"名前"が不正です。コンパイラの環境設定やホスト環境のファイルシステム異常がないか確認ください。

C5920 (F) Cannot open output file: "名前"
テンプレート情報ファイル"名前"をオープンすることができません。コンパイラの環境設定やホスト環境のファイルシステム異常がないか確認ください。

C5925 (W) Type qualifiers on function types are ignored
関数型への型修飾子を無視します。

C5926 (F) Cannot open definition list file: "名前"
ファイル"名前"をオープンすることができません。コンパイラの環境設定やホスト環境のファイルシステム異常がないか確認ください。

C5928 (E) Incorrect use of va_start
va_start の使用方法に誤りがあります。

C5929 (E) Incorrect use of va_arg
va_arg の使用方法に誤りがあります。

C5930 (E) Incorrect use of va_end
va_end の使用方法に誤りがあります。

C5934 (E) A member with reference type is not allowed in a union
参照型は共用体のメンバにできません。

C5935 (E) "typedef" may not be specified here
typedef を指定することはできません。

C5936 (W) Redeclaration of entity-kind "名前" alters its access
"名前"の再宣言でアクセス指定を変更しています。再定義されたアクセス指定を有効にします。

C5937 (E) A class or namespace qualified name is required
クラスまたは namespace の限定名が要求されます。

C5938 (E) Return type "int" omitted in declaration of function "main"
int 型の戻り値は main 関数の宣言において除外されます。

C5939 (E) pointer-to-member representation “シンボル 1” is too restrictive for “シンボル 2”
メンバへのポインタの宣言が正しくありません。

C5940 (W) Missing return statement at end of non-void entity-kind "名前"
void 型以外をリターンする関数"名前"が return 文を持ちません。return 値は不定になります。

C5941 (W) Duplicate using-declaration of "名前" ignored
using 宣言"名前"を重複指定しています。重複した using 宣言を無効にします。

C5942 (W) enum bit-fields are always unsigned, but enum “名前” includes negative enumerator
列挙型のビットフィールドは常に unsigned ですが、列挙型 “名前” には値が負の列挙定数が含まれています。

C5946 (E) Name following "template" must be a member template
"template"に続く名前はメンバテンプレートでなければなりません。

C5947 (E) Name following "template" must have a template argument list
"template"に続く名前はテンプレート実引数でなければなりません。

C5948 (E)(W) Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope
非標準形式のローカルクラスのフレンド宣言です。クラスの定義内に前方宣言がありません。

C5949 (I) Specifying a default argument on this declaration is nonstandard
この宣言にデフォルト引数を指定するのは標準形式ではありません。

C5951 (E)(W) Return type of function "main" must be "int"
main 関数の戻り値は int でなければいけません。

C5952 (E) A template parameter may not have class type
テンプレート仮引数にクラス型名は指定できません。

C5953 (E) A default template argument cannot be specified on the declaration of a member of a class template
クラステンプレートのメンバ宣言にデフォルトのテンプレート実引数を指定できません。

C5954 (E) A return statement is not allowed in a handler of a function try block of a constructor
コンストラクタの try ブロックのハンドラ内にリターン文は許されません。

C5955 (E) Ordinary and extended designators cannot be combined
in an initializer designation
指示子が正しくありません。

C5956 (E) The second subscript must not be smaller than the first
2 番目の添え字は 1 番目の添え字より大きくなければいけません。

C5959 (W) Declared size for bit field is larger than the size of the bit field type;
truncated to "ビット数" bits
指定されたビット数がビットフィールドの型の"ビット数"を超えています。ビット数をビットフィールドの型に合わせて処理を継続します。

C5960 (E) Type used as constructor name does not match type "型"
コンストラクタ名として使用された型が"型"と一致しません。

C5961 (W) Use of a type with no linkage to declare a variable with linkage
リンケージを持たない型を使用してリンケージを持つ変数として宣言しています。リンケージを持つものとします。

C5962 (W) Use of a type with no linkage to declare a function
リンケージを持たない型を使用してリンケージを持つ関数として宣言しています。リンケージを持つものとします。

C5963 (E) Return type may not be specified on a constructor
コンストラクタにリターン型を指定できません。

C5964 (E) Return type may not be specified on a destructor
デストラクタにリターン型を指定できません。

C5965 (E) Incorrectly formed universal character name
universal character の形式が正しくありません。

C5966 (E) Universal character name specifies an invalid character
universal character で指定された文字が不正です。

C5967 (E) A universal character name cannot designate a character in the basic character set

基本文字集合内で universal character を文字として指定することはできません。

C5968 (E) This universal character is not allowed in an identifier

識別子にこの universal character は許されません。

C5969 (E) The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros

__VA_ARGS__識別子 は可変個引数を持つマクロの置換リスト内以外に記述できません。

C5970 (W) The qualifier on this friend declaration is ignored

このフレンド宣言への修飾子は無視されます。

C5971 (E) Array range designators cannot be applied to dynamic initializers

配列範囲名は動的初期化子に適用できません。

C5972 (E) Property name cannot appear here

プロパティ名はここに存在できません。

C5973 (W) "inline" used as a function qualifier is ignored

関数修飾子として使用された”inline”を無視します。

C5975 (E) A variable-length array type is not allowed

可変長配列型は使用できません。

C5976 (E) A compound literal is not allowed in an integral constant expression

複合リテラルは整数定数式で使用することはできません。

C5977 (E) A compound literal of type “型” is not allowed

指定の複合リテラル型は使用できません。

C5978 (E) A template friend declaration cannot be declared in a local class

テンプレートのフレンド関数は局所クラスで宣言できません。

C5979 (E) Ambiguous "?" operation: second operand of type "型 1" can be converted to third operand type "型 2", and vice versa

三項演算子"?:"の第 2 式の"型 1"と第 3 式の"型 2"が互いに変換可能な型であいまいです。

C5980 (E) Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type

オブジェクトを呼び出していますが operator()関数または関数へのポインタ型変換関数が定義されていません。

C5982 (E) There is more than one way an object of type "型" can be called for the argument list

実引数リストから呼ぶことができる"型"のオブジェクトが 2 つ以上あります。

C5983 (E) typedef name has already been declared (with similar type)

typedef 名はすでに同等の型で宣言されています。

C5984 (W) Operator new and operator delete cannot be given internal linkage

operator new/operator delete が static で定義されています。

C5985 (E) Storage class "mutable" is not allowed for anonymous unions

mutable を無名共用体に指定することはできません。

C5987 (E) Abstract class type "型" is not allowed as catch type:

抽象クラスを catch で受けることはできません。

C5988 (E) A qualified function type cannot be used to declare a nonmember function or a static member function

修飾付き関数型を非メンバ関数や static メンバ関数の宣言に使用することはできません。

C5989 (E) A qualified function type cannot be used to declare a parameter

修飾付き関数型を関数パラメータ指定に使用することはできません。

C5990 (E) Cannot create a pointer or reference to qualified function type

修飾付き関数型へのポインタ型や参照型を作ることはできません。

C5991 (W) Extra braces are nonstandard

集合型の初期化子リストに余分な'{'があります。

C5992 (E) Invalid macro definition:
不正なマクロ定義です。

C5993 (W) Subtraction of pointer types “シンボル名 1” and “シンボル名 2” is nonstandard
ポインタ型のシンボル 1 とシンボル 2 の減算は標準形式ではありません。

C5994 (E) An empty template parameter list is not allowed in a template parameter declaration
空テンプレートパラメータを持つテンプレートをテンプレートパラメータに指定することはできません。

C5995 (E) Expected "class"
テンプレートパラメータに指定するクラステンプレートはクラスを必要とします。

C5996 (E) The "class" keyword must be used when declaring a template parameter
テンプレートパラメータに指定するクラステンプレートは構造体ではいけません。

C5997 (W) "関数名 1" is hidden by "関数名 2" -- virtual function override intended?
"関数名 1"が"関数名 2"を隠しています。仮想関数をオーバーライドしようとしていないか確認してください。

C5998 (E) A qualified name is not allowed for a friend declaration that is a function definition
friend 指定付き関数定義において、名前空間の名前付き関数名を指定することはできません。

C5999 (E) "型 1" is not compatible with "型 2"
指定したクラステンプレートはテンプレートパラメータと形式が一致しません。

C6000 (W) A storage class may not be specified here
ここには記憶域クラス指定子を指定することはできません。

C6001 (E) Class member designated by a using-declaration must be visible in a direct base class
クラスメンバの using 指定は参照可能な直接基底クラスでなければなりません。

C6006 (E) A template parameter cannot have the same name as one of its template parameters
テンプレートパラメータに指定するクラステンプレート名が、それ自身のテンプレートパラメータ名と同じです。

C6007 (E) Recursive instantiation of default argument
テンプレート関数のデフォルト引数のインスタンスが再帰的に生成されます。

C6009 (E) "インスタンス名" is not an entity that can be defined
実体のないインスタンスを生成しようとしています。

C6010 (E) Destructor name must be qualified
不正なデストラクタ名です。

C6011 (E) Friend class name may not be introduced with "typename"
フレンドクラスの名前を ”typename” に続けて記述してはいけません。

C6012 (E) A using-declaration may not name a constructor or destructor
using 宣言でコンストラクタまたはデストラクタを指定してはいけません。

C6013 (E) A qualified friend template declaration must refer to a specific previously declared template
限定フレンドテンプレートは参照前に定義しておく必要があります。

C6014 (E) Invalid specifier in class template declaration
不正な指定子がクラステンプレート宣言に含まれています。

C6015 (E) Argument is incompatible with formal parameter
引数が定義された引数と互換性がありません。

C6017 (E) Loop in sequence of "operator->" functions starting at class “シンボル”
operator->が正しくありません。

C6018 (E) "クラス名" has no member class "メンバ名"
クラスにないメンバを使っています。

C6019 (E) The global scope has no class named "クラス名"
クラス内の名前にファイルスコープ演算子を使っています。

C6020 (E) Recursive instantiation of template default argument
テンプレートのデフォルト引数で再帰的にインスタンスを生成します。

C6021 (E) Access declarations and using-declarations cannot appear in unions
union で using 指定は使えません。

C6022 (E) "名前" is not a class member
クラスのメンバではありません。

C6023 (E) Nonstandard member constant declaration is not allowed
非標準形式の const メンバは宣言することができません。

C6028 (W) Invalid redeclaration of nested class
クラス内でクラスを二重定義しています。

C6029 (E) Type containing an unknown-size array is not allowed
サイズが未定の配列を持つ構造体または共用体はメンバにできません。

C6030 (W) A variable with static storage duration cannot be defined within an inline function
静的なスコープを持つ変数はインライン関数内に宣言できません。

C6031 (W) An entity with internal linkage cannot be referenced within an inline function with external linkage
内部リンケージを持つ識別子は外部リンケージを持つインライン関数内で参照することはできません。

C6032 (E) Argument type “型” does not match this type-generic function macro
引数の型がジェネリック関数生成マクロの型に合いません。

C6034 (E) Friend declaration cannot add default arguments to previous declaration
フレンド関数が宣言された場合、フレンド関数の定義にデフォルト引数をいれることはできません。

C6035 (E) "テンプレート名" cannot be declared in this scope
このスコープではテンプレートを宣言することができません。

C6036 (E) The reserved identifier “シンボル” may only be used inside a function
関数外で__FUNC__を使用しています。

C6037 (E) This universal character cannot begin an identifier
この汎用文字で識別子名を始めることはできません。

C6038 (E) Expected a string literal
文字列リテラルがありません。

C6039 (E) Unrecognized STDC pragma
認識できない STDC プラグマです。

C6040 (E) Expected "ON", "OFF", or "DEFAULT"
“ON”、”OFF”、”DEFAULT”がありません。

C6041 (E) A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope
STDC プラグマが現れるのはグローバルスコープ内の宣言の間、いかなる式の間またはブロックスコープ内の宣言の間だけです。

C6042 (E) Incorrect use of va_copy
va_copy マクロの使用方法が不正です。

C6043 (E) “型” can only be used with floating-point types
“型"が浮動小数点型以外の型と使用しています。

C6044 (E) Complex type is not allowed
複素数型を使えません。

C6045 (E) Invalid designator kind
不正なフィールド識別子です。

C6046 (W) Floating-point value cannot be represented exactly
浮動小数点数値に誤差が生じています。

C6047 (E) Complex floating-point operation result is out of range
複素数型浮動小数点演算の結果が表現可能な値の範囲を超えました。

C6048 (E) Conversion between real and imaginary yields zero
実数と虚数の相互変換後の値が 0 になりました。

C6049 (E) An initializer cannot be specified for a flexible array member
可変長配列メンバに初期化子を指定することはできません。

C6050 (W) imaginary *= imaginary sets the left-hand operand to zero
虚数 *= 虚数は左辺値を 0 にします。

C6051 (E)(W) Standard requires that “シンボル” be given a type by a subsequent declaration ("int" assumed)
暗黙の型は使用できません。

C6052 (E) A definition is required for inline “シンボル”
インライン関数の定義がありません。

C6053 (W) Conversion from integer to smaller pointer
整数がより小さいサイズのポインタへ変換されました。

C6054 (E) A floating-point type must be included in the type specifier for a _Complex or _Imaginary type
浮動小数点型は複素数または虚数型の指定子に含まれてなければいけません。

C6055 (E) Types cannot be declared in anonymous unions
型を無名共用体内で宣言することはできません。

C6056 (W) Returning pointer to local variable
ローカル変数へのポインタを返しています。

C6057 (W) Returning pointer to local temporary
ローカルな領域へのポインタを返しています。

C6061 (E) Declaration of “シンボル名” is incompatible with a declaration in another translation unit
“シンボル名”の宣言はもう一つの翻訳単位内の宣言と互換性がありません。

C6062 (E) The other declaration is "行”
別の宣言があります。

C6065 (E) A field declaration cannot have a type involving a variable length array
フィールド宣言は可変長配列が存在する型を含むことができません。

C6066 (E) declaration of “インスタンス” had a different meaning during compilation of “シンボル”
コンパイル時に宣言が異なっています。

C6067 (E) Expected "template"
“template”がありません。

C6072 (E)(W) A declaration cannot have a label
宣言はラベルを持つことはできません。

C6075 (E) “インスタンス名” already defined during compilation of “シンボル”
コンパイル時にすでに定義されています。

C6076 (E) “シンボル” already defined in another translation unit
すでに別の翻訳単位で定義されています。

C6081 (E) A field with the same name as its class cannot be declared in a class with a user-declared constructor
クラス名と同じ名前のメンバを宣言することはできません。

C6083 (F) Exported template file ファイル名 is corrupted
エクスポートされたテンプレートファイルは破損しています。

C6086 (E) the object has cv-qualifiers that are not compatible with the member “シンボル”
オブジェクトの持つ CV 修飾子はメンバ“シンボル”と互換性がありません。

C6087 (E) No instance of “クラス名” matches the argument list and object (the object has cv-qualifiers that prevent a match)
“クラス名”のインスタンスは引数リストとオブジェクトと合致しません。(オブジェクトの持つ CV 修飾子が合致を抑止しています)

C6089 (E) There is no type with the width specified
幅が指定された型がありません。

C6105 (W) #warning directive: “文字列”
“文字列”を出力しました。

C6139 (E) The "template" keyword used for syntactic disambiguation may only be used within a template
キーワード”template”を構文上の曖昧さを解消するのに使用できるのは template 内のみです。

C6144 (E) Storage class must be auto or register
記憶クラスは auto または register でなければいけません。

C6145 (W) “型 1” would have been promoted to “型 2” when passed through the ellipsis parameter; use the latter type instead
型 1 は型 2 へと拡張されます。型 2 を使用します。

C6146 (E) “シンボル” is not a base class member
基底クラスのメンバではありません。

C6151 (F) Mangled name is too long
マングルされた名前が長すぎます。

C6158 (E) void return type cannot be qualified
void 型の戻り値は修飾できません。

C6161 (E) A member template corresponding to “シンボル” is declared as a template of a different kind in another translation unit
テンプレート宣言が他コンパイル単位と異なっています。

C6163 (E) va_start should only appear in a function with an ellipsis parameter
va_start が使用されるのは省略記号を引数とする関数のみです。

C6192 (W) Null (zero) character in input line ignored
入力ライン中の null 文字が無視されました。

C6193 (W) Null (zero) character in string or character constant
文字列または文字定数内に null 文字が含まれています。

C6194 (W) Null (zero) character in header name
ヘッダ名に null 文字が含まれています。

C6197 (W) The prototype declaration of “シンボル” is ignored after this unprototyped redeclaration
関数原型を無視します。

C6201 (E) Typedef “シンボル” may not be used in an elaborated type specifier
詳述型指定子に使用できません。

C6203 (E) Parameter “引数名” may not be redeclared in a catch clause of function try block
“引数名”を try ブロックの catch 句の中で再宣言してはいけません。

C6204 (E) The initial explicit specialization of “シンボル名” must be declared in the namespace containing the template
シンボルに対する最初の明示的な特殊化はテンプレートを含む名前空間の中に宣言されなければいけません。

C6206 (E) "template" must be followed by an identifier
“template”の後には識別子が必要です。

C6211 (W) Nonstandard cast to array type ignored
非標準形式の配列型へのキャストが無視されました。

C6212 (E) This pragma cannot be used in a _Pragma operator (a #pragma directive must be used)
このプラグマは_Pragma operator 内では使用できません。(#pragma ディレクティブを使用してください)

C6213 (W) Field uses tail padding of a base class
フィールドは基底クラスの終端パディングを使用しています。

C6218 (W) Base class “クラス名 1” uses tail padding of base class “クラス名 2”
基底クラス 1 は基底クラス 2 の終端パディングを使用しています。

C6222 (W) Invalid error number
不正なエラー番号です。

C6223 (W) Invalid error tag
不正なエラータグです。

C6224 (W)Expected an error number or error tag
エラー番号またはエラータグがありません。

C6227 (E) Transfer of control into a statement expression is not allowed
式文への制御の転移はできません。

C6229 (E) This statement is not allowed inside of a statement expression
この式は式文内にあってはいけません。

C6230 (E) A non-POD class definition is not allowed inside of a statement expression
非 POD クラスは式文内に定義できません。

C6235 (W) Nonstandard conversion between pointer to function and pointer to data
非標準形式の変換がポインタ関数と不完全なオブジェクト間で行われました。

C6254 (E) Integer overflow in internal computation due to size or complexity of “型”
データ型のサイズまたは複雑さに伴い、内部の計算結果にて整数のオーバーフローが発生しました。

C6255 (E) Integer overflow in internal computation
内部の計算結果にて整数のオーバーフローが発生しました。

C6273 (W) Alignment-of operator applied to incomplete type
オペレータのアライメントが不完全な型に対して適用されました。

C6280 (E) Conversion from inaccessible base class “クラス名” is not allowed
派生クラスにプライベートで継承された基底クラス型のポインタを継承クラス型のポインタへ変換することはできません。

C6282 (E) String literals with different character kinds cannot be concatenated
違う種類の文字列リテラルを連結することはできません。

C6285 (W) Nonstandard qualified name in namespace member declaration
非標準形式の修飾子名が名前空間のメンバの宣言に使用されています。

C6290 (W) Non-POD class type passed through ellipsis
非 POD クラス型が省略記号に渡されています。

C6291 (E) A non-POD class type cannot be fetched by va_arg
非 POD 型のクラスは va_arg によって取得することができません。

C6292 (E) The 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal
固定小数点リテラルにおいて、‘u’ または ’U’ 型の接尾辞は ‘l’ または ‘L’ の接尾辞の前に現れなければいけません。

C6294 (W) Integer operand may cause fixed-point overflow
整数オペランドは固定小数点オーバーフローを起こす可能性があります。

C6295 (E) Fixed-point constant is out of range
固定小数点定数が表現可能な範囲を超えています。

C6296 (W) Fixed-point value cannot be represented exactly
固定小数点では 16 進数表記を完全に表現することができません。

C6297 (W) Constant is too large for long long; given unsigned long long type (nonstandard)
定数は long long 型としては大きすぎます。 Unsigned の long long 型に変更します。(非標準形式)

C6301 (W) “シンボル” declares a non-template function -- add <> to refer to a template instance
非テンプレート関数を宣言しています。

C6302 (W) Operation may cause fixed-point overflow
演算によって固定小数点オーバーフローが起こる可能性があります。

C6303 (E) Expression must have integral, enum, or fixed-point type
式には整数型、列挙型または固定小数点型を含んでください。

C6304 (E) Expression must have integral or fixed-point type
式には整数型または固定小数点型を含んでください。

C6307 (W) Class member typedef may not be redeclared
クラスメンバの typedef を再宣言してはいけません。

C6308 (W) Taking the address of a temporary
ローカルな領域のアドレスを取得しています。

C6310 (W) Fixed-point value implicitly converted to floating-point type
固定小数点値が浮動小数点型に暗黙的に変換されました。

C6311 (E) Fixed-point types have no classification
浮動小数点型の区分がありません。

C6312 (E) A template parameter may not have fixed-point type
テンプレート引数には固定小数点型を指定できません。

C6313 (E) Hexadecimal floating-point constants are not allowed
16 進数の浮動小数点定数は使用できません。

C6315 (E) Floating-point value does not fit in required fixed-point type
浮動小数点数値は要求された固定小数点型に収まりません。

C6316 (W) Value cannot be converted to fixed-point value exactly
値を固定小数点値にすると誤差が生じます。

C6317 (E) Fixed-point conversion resulted in a change of sign
負の整数値を固定小数点型へ変換した結果、正の値になりました。

C6318 (E) Integer value does not fit in required fixed-point type
整数値は要求された固定小数点型に収まりません。

C6319 (E)(W) Fixed-point operation result is out of range
固定小数点演算の結果が表現可能な値の範囲をこえました。

C6320 (E) Multiple named address spaces
同一の名前アドレス空間が複数存在します。

C6321 (E) Variable with automatic storage duration cannot be stored in a named address space
局所的なスコープを持つ変数は名前付きアドレス空間に保持することはできません。

C6322 (E) Type cannot be qualified with named address space
名前付きアドレス空間によって型を識別することはできません。

C6323 (E) Function type cannot be qualified with named address space
名前付きアドレス空間によって関数型を識別することはできません。

C6324 (E) Field type cannot be qualified with named address space
フィールド型は名前付き空間によって識別することはできません。

C6325 (E) Fixed-point value does not fit in required floating-point type
固定小数点値は要求された浮動小数点型に収まりません。

C6326 (E) Fixed-point value does not fit in required integer type
固定小数点値は要求された整数型に収まりません。

C6327 (E) Value does not fit in required fixed-point type
値は要求された固定小数点値に収まりません。

C6335 (F) Cannot open predefined macro file: “ファイル名”
定義済みマクロファイルを開けません。

C6336 (F) Invalid predefined macro entry at line “行数”: “マクロ名”
不正な定義済みマクロの entry 宣言が ”行数” にあります。

C6337 (F) Invalid macro mode name “マクロモード名”
不正なマクロモード名です。

C6338 (F) Incompatible redefinition of predefined macro “マクロ名”
互換性の無い定義済みマクロの再定義です。

C6342 (W) const_cast to enum type is nonstandard
const_cast で列挙型をキャストするのは標準形式ではありません。

C6344 (E) A named address space qualifier is not allowed here
名前付きアドレス空間識別子はここで使用できません。

C6345 (E) An empty initializer is invalid for an array with unspecified bound
空の初期化子で境界が指定されていない配列を初期化するのは不正です。

C6346 (W) Function returns incomplete class type “クラス名”
関数が不正なクラス型を返しています。

C6348 (I) Declaration hides “変数名”
局所変数が他の局所変数の宣言によって隠蔽されました。

C6349 (E) A parameter cannot be allocated in a named address space
引数は名前付きアドレス空間に配置できません。

C6350 (E) Invalid suffix on fixed-point or floating-point constant
不正な接尾辞が固定または浮動小数点定数についています。

C6351 (E) A register variable cannot be allocated in a named address space
レジスタ変数は名前付きアドレス空間に配置できません。

C6352 (E) Expected "SAT" or "DEFAULT"
“SAT”または”DEFAULT”がありません。

C6353 (I) “シンボル名” has no corresponding member operator delete “シンボル名” (to be called if an exception is thrown during initialization of an allocated object)
シンボルは new オペレータの対となる delete オペレータを持ちません。(取得したオブジェクトの初期化時に例外が発生した場合に呼ばれます。)

C6355 (E) A function return type cannot be qualified with a named address space
関数の戻り値を名前付きアドレス空間で修飾することはできません。

C6361 (W) Negation of an unsigned fixed-point value
符号なしの固定小数点を無効にします。

C6365 (E) Named-register variables cannot have void type

名前付きレジスタ変数は void 型にできません。

C6372 (E) Nonstandard qualified name in global scope declaration

非標準形式で修飾した名前がグローバルなスコープに宣言されています。

C6373 (W) Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)

64 ビット整数型がより小さい整数型へと暗黙的に変換されています。移植性の問題になる可能性があります。

C6374 (W) Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)

64 ビット整数型がより小さい整数型へと明示的に変換されています。移植性の問題になる可能性があります。

C6375 (W) Conversion from pointer to same-sized integral type (potential portability problem)

ポインタから同サイズの整数型へと変換しています。移植性の問題になる可能性があります。

C6380 (E)(I) Virtual “関数名” was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace)

仮想関数の定義がありません。また、無名空間のメンバである為それ以外の場所で定義することができません。

C6381 (E)(I) Carriage return character in source line outside of comment or character/string literal

改行文字がコメントまたは文字列リテラル以外のところにあります。

C6382 (E) Expression must have fixed-point type

式に固定小数点を含めなくてはいけません。

C6386 (W) Storage specifier ignored

記憶クラス指定子を無視します。

C6396 (W) White space between backslash and newline in line splice ignored

行接合部のバックスラッシュと改行の間の空白を無視します。

C6398 (E) Invalid member for anonymous member class -- class “シンボル” has a disallowed member function

無名のメンバクラスに対して不正なメンバ関数を宣言しています。

C6400 (W) Positional format specifier cannot be zero

位置フォーマット指定子に 0 を指定することはできません。

C6403 (E) A variable-length array is not allowed in a function return type

可変長配列を関数の戻り値型とすることはできません。

C6404 (E) Variable-length array type is not allowed in pointer to member of type “型”

クラスメンバへのポインタとして可変長配列型メンバへのポインタは禁止されています。

C6405 (E) The result of a statement expression cannot have a type involving a variable-length array

式文の演算結果に可変長配列型が含まれてはいけません。

C6420 (E)(W) Some enumerator values cannot be represented by the integral type underlying the enum type

整数型で表せない列挙値です。

C6421 (E) Default argument is not allowed on a friend class template declaration

デフォルト引数をフレンドクラスのテンプレート宣言に指定することはできません。

C6422 (W) Multicharacter character literal (potential portability problem)

複数文字リテラルです。移植性の問題を引き起こす可能性があります。

C6424 (E) Second operand of offsetof must be a field

マクロ offsetof の二番目のオペランドはフィールドでなくてはいけません。

C6425 (E) Second operand of offsetof may not be a bit field

マクロ offsetof の二番目のオペランドはフィールドであってはいけません。

C6426 (E) Cannot apply offsetof to a member of a virtual base

マクロ offsetof を仮想基底クラスのメンバに適用することはできません。

C6427 (W) offsetof applied to non-POD types is nonstandard
マクロ offsetof を非 POD 型に適用するのは標準形式ではありません。

C6428 (E) Default arguments are not allowed on a friend declaration of a member function
デフォルト引数をフレンド宣言のメンバ関数に指定することはできません。

C6429 (E) Default arguments are not allowed on friend declarations that are not definitions
デフォルト引数を定義ではないフレンド宣言に指定することはできません。

C6430 (E) Redeclaration of “関数名” previously declared as a friend with default arguments is not allowed
デフォルト引数を持つフレンドとしてすでに宣言した関数を再宣言することはできません。

C6431 (E) Invalid qualifier for “シンボル” (a derived class is not allowed here)
限定子が正しくありません。

C6432 (E) Invalid qualifier for definition of class “クラス名”
不正な修飾子をクラスの定義に指定しました。

C6439 (E) Template argument list of “シンボル” must match the parameter list
テンプレート引数リストに合わなければなりません。

C6440 (E) An incomplete class type is not allowed
不完全なクラス型です。

C6445 (E) Invalid redefinition of “シンボル名”
列挙型が再定義されています。

C6449 (E) Explicit specialization of “シンボル” must precede its first use “シンボル 2”
テンプレートをすでに具現化しています。

C6623 (W) The destructor for “クラス 1” has been suppressed because the destructor for “クラス 2” is inaccessible
クラス 2 のデストラクタにアクセスできないため、クラス 1 のデストラクタは抑制されました。

C6648 (W) '=' assumed following macro name “マクロ名” in command-line definition
コマンドライン定義内のマクロ名の後ろには'='がついているとみなします。

C6649 (E)(W) White space is required between the macro name “マクロ名” and its replacement text
”マクロ名” とその置換テキストの間には空白が必要です。

C6655 (E) “シンボル” cannot be declared inline after its definition “定義名”
inline が抑止されているため、シンボルは inline 関数として宣言することができません。

C6671 (W) __assume expression with side effects discarded
副作用のある __assume 式が破棄されました。

C6674 (E) __evenaccess qualifier is applied to only integer type
__evenaccess 修飾子は整数タイプのみに指定できます。

C6675 (E) Expected a section name string
__sectop/__secend/__secsize にセクション名がありません。

C6676 (E) Expected a section name
セクション名がありません。

C6677 (E) Invalid pragma declaration
#pragma の構文が不正です。

C6678 (E) "シンボル名" has already been specified by other pragma
このシンボルは既に他の#pragma 指定がされています。

C6679 (E) Pragma may not be specified after definition
シンボル定義後の宣言にのみ#pragma 指定することはできません。

C6680 (E) Invalid kind of pragma is specified to this symbol
不正な#pragma を指定しました。

C6681 (I) This pragma has no effect
この#pragma は無効です。

C6682 (E) “シンボル名” must be qualified for function type
シンボルは関数型でなければいけません。

C6683 (E) Illegal “プラグマ名” specifier
不正な#pragma です。

C6684 (E) Multiple pointer qualifiers
ポインタ型修飾子が重複しています。

C6685 (E) __ptr16 must be qualified for data pointer type
__ptr16 はデータポインタ型以外を修飾できません。

C6686 (E) Invalid binary digit
不正な 2 進数です。

C6687 (W) This pragma “名前” is ignored
“名前”という#pragma は無視されます。

C6688 (E) "this" pointer of “クラス名” is cast implicitly to near pointer
“this”を暗黙的に near ポインタでキャストしました。

C6689 (E) Can not specify near or far for member
メンバ関数に対して near または far を指定することはできません。

C6690 (E) A member “関数名” qualified with near or far is declared
メンバ関数に対して near または far が指定されています。

C6691 (E) near or far specifier on a reference type is not allowed
near または far 指定を参照タイプに指定することはできません。

C6692 (E) can not specify near or far for member function
メンバ関数に対して near または far を指定することはできません。

C6693 (E) can not specify near or far for function types
関数タイプに対して near または far を指定することはできません。

C6698 (E) Incorrect PIC address usage

PIC 機能使用時のアドレスの使用方法が不適切です。PIC 機能が有効な場合、関数のアドレスを静的初期化や集成体の初期化式に使用することはできません。

C6699 (E) Incorrect PID address usage

PID 機能使用時のアドレスの使用方法が不適切です。PID 機能が有効な場合、定数領域やリテラルセクションにある変数のアドレスを静的初期化や集成体の初期化式に使用することはできません。

11.3 C標準ライブラリ関数のエラーメッセージ

ライブラリ関数の中には、ライブラリ関数を実行中にエラーが発生した場合、標準ライブラリのヘッダファイル<errno.h>で定義しているマクロ errno にエラー番号を設定するものがあります。
エラー番号には対応するエラーメッセージが定義してあり、エラーメッセージを出力することができます。エラーメッセージを出力するプログラム例を以下に示します。

例

```
#include      <stdio.h>
#include      <string.h>
#include      <stdlib.h>
#include      <errno.h>

main()
{
    FILE *fp;

    fp=fopen("file", "w");
    fp=NULL;

    fclose(fp);                    /* error occurred     */

    printf("%s¥n", strerror(errno));  /* print error message */
}
```

説明

(1) fclose 関数に値 NULL のファイルポインタを実引数として渡しているので、エラーとなります。このとき errno に対応するエラー番号が設定されます。

(2) strerror 関数は、エラー番号を実引数として渡すと、対応するエラーメッセージの文字列のポインタを返します。printf 関数の文字列出力指定によりエラーメッセージを出力します。

標準ライブラリエラーメッセージ一覧

エラー番号	エラーメッセージ／説明	エラー番号を設定する関数
0x22 (ERANGE)	Data out of range **オーバーフローが発生しました。**	frexp, ldexp, modf, ceil, floor, fmod, atof, atoi, atol, atoll, atolfixed, atolaccum, strtod, strtol, strtoul, strtoll, stroull, strtolfixed, strtolaccum, perror, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, acos, acosf, asin, asinf, atan, atan2, atan2f, atanf, ceilf, cos, cosf, cosh, coshf, exp, expf, floorf, fmodf, ldexpf, log, log10, log10f, logf, modff, pow, powf, sin, sinf, sinh, sinhf, sqrt, sqrtf, tan, tanf, tanh, tanhf, fabs, fabsf, frexpf
0x21 (EDOM)	Data out of domain **数学関数の引数に対する結果の値が定義されません。**	acos, acosf, asin, asinf, atan, atan2, atan2f, atanf, ceil, ceilf, cos, cosf, cosh, coshf, exp, expf, floor, floorf, fmod, fmodf, ldexp, ldexpf, log, log10, log10f, logf, modf, modff, pow, powf, sin, sinf, sinh, sinhf, sqrt, sqrtf, tan, tanf, tanh, tanhf, fabs, fabsf, frexp, frexpf
0x450 (ESTRN)	Too long string **文字列の文字数が** 512 **文字を超えています。**	atof, atoi, atol, atoll, atolfixed, atolaccum, strtod, strtol, strtoul, strtoll, stroull, strtolfixed, strtolaccum
0x04B0 (ECBASE)	Invalid radix **基数の指定が誤っています。**	strtol, strtoul, strtoll, strtoull
0x04B2 (ETLN)	Number too long **数値を表現する文字列の文字数が有効桁数を超えています。**	atof, atolfixed, atolaccum, strtod, strtolfixed, strtolaccum, fscanf, scanf, sscanf
0x04B4 (EEXP)	Exponent too large **指数部の桁数が** 3 **桁を超えています。**	atof, strtod, fscanf, scanf, sscanf
0x04B6 (EEXPN)	Normalized exponent too large **文字列を一度** IEEE **規格の** 10 **進形式に正規化したとき指数部の桁数が** 3 **桁を超えています。**	atof, strtod, fscanf, scanf, sscanf
0x04BA (EFLOATO)	Overflow out of float float **型の** 10 **進数値が**,float **型の範囲を超えています(オーバーフロー)。**	fscanf, scanf, sscanf
0x04C4 (EFLOATU)	Underflow out of float float **型の** 10 **進数値が**,float **型の範囲を超えています(アンダーフロー)。**	fscanf, scanf, sscanf
0x04E2 (EDBLO)	Overflow out of double double **型の** 10 **進数値が，**double **型の範囲を超えています(オーバーフロー)。**	fscanf, scanf, sscanf
0x04EC (EDBLU)	Underflow out of double double **型の** 10 **進数値が，**double **型の範囲を超えています(アンダーフロー)。**	fscanf, scanf, sscanf
0x04F6 (ELDBLO)	Overflow out of long double long double **型の** 10 **進数値が，**long double **型の範囲を超えています(オーバーフロー)。**	fscanf, scanf, sscanf

エラー番号	エラーメッセージ／説明	エラー番号を設定する関数
0x0500 (ELDBLU)	Underflow out of long double long double 型の 10 進数値が，long double 型の範囲を超えています(アンダーフロー)。	fscanf, scanf, sscanf

12. アセンブラのエラーメッセージ

12.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
エラー内容

エラーレベルは、エラーの重要度に従い、3 種類に分類されます。

エラーレベル		動作
(W)	ウォーニング	処理を継続します。
(E)	エラー	処理を中断します。
(F)	フェータル	処理を中断します。

12.2 メッセージ一覧

A1000 (W) '.ALIGN' with not 'ALIGN' specified relocatable section
ALIGN 指定がないセクション内に制御命令".ALIGN "が記述されています。
制御命令".ALIGN"の記述位置を確認してください。制御命令".ALIGN"を記述するセクションのセクション定義行に ALIGN 指定を記述してください。

A1001 (W) Destination address may be changed
分岐先が期待するものと異なる位置になる可能性があります。
アドレッシングモードが最適選択されないように分岐命令のオペランドを記述してください。

A1002 (W) Floating point value is out of range
浮動小数点数が範囲外です。
浮動小数点数の記述を確認してください。範囲外は無視します。

A1003 (W) Location counter exceed
ロケーションカウンタが 0FFFFFFFFh を超えました。
.ORG のオペランド値を確認してください。ソースを記述し直してください。

A1004 (W) '.ALIGN' size is different
アライメント値が異なります。
アライメント値を確認してください。

A1006 (W) Data in 'CODE' section align in 4byte
endian=big 時、CODE セクション中データ領域の開始位置は 4 バイト境界に補正されます。

A1007 (W) Data size in 'CODE' section align in 4byte
endian=big 時、CODE セクション中データ領域のサイズは 4 の倍数に補正されます。

A1009 (W) Multiple symbols
.STACK でシンボルへのスタック値指定を重複して行なっています。

A1010 (W) Section attribute mismatch
セクションの属性が異なっています。

A1011 (W) Use PM instruction
特権命令を使用しています。

A1012 (W) Use FPU instruction
浮動小数点演算命令を使用しています。

A1013 (W) Use DSP instruction
DSP 機能命令を使用しています。

A1014 (W) Too many actual macro parameters
マクロ実引数の数が多すぎます。
余分な実引数は無視されます。

A1015 (W) Actual macro parameters are not enough
マクロ実引数の数がマクロ仮引数の数より少なくなっています。
該当する実引数のない仮引数は無効となります。

A1016 (W) '.END' statement is in include file

インクルードファイルに .END 記述があります。

インクルードファイル内には、.END は記述できません。記述を削除してください。

.END を無視して処理します。

A2000 (E) No space after mnemonic or directive

ニーモニック、アセンブル制御命令の直後に空白文字がありません。

命令とオペランドの間に、空白文字を記述してください。

A2001 (E) ',' is missing

','の記述がありません。

オペランドの区切りには、カンマを記述してください。

A2002 (E) Characters exist in expression

命令又は式中に余分な文字があります。

式の記述規則を確認してください。

A2003 (E) Size specifier is missing

サイズ指定子がありません。

サイズ指定子を記述してください。

A2004 (E) Invalid operand(s) exist in instruction

命令に無効なオペランドがあります。

命令のオペランドの記述方法を確認して、記述し直してください。

A2005 (E) Operand type is not appropriate

オペランドの種類が間違っています。

オペランドの記述方法を確認して、記述し直してください。

A2006 (E) Size specifier is not appropriate

サイズ指定子の記述に間違いがあります。

サイズ指定子を記述し直してください。

A2007 (E) Operand label is not in the same section

分岐先が同一セクション内にありません。

同一セクション内の分岐先にしか分岐できません。ニーモニックを記述し直してください。

A2008 (E) Illegal displacement value

ディスプレースメント値が間違っています。

サイズ指定子が W のときは、2 の倍数、L のときは、4 の倍数にしてください。

A2009 (E) FPU instruction or FPSW is used

浮動小数点演算(FPU)命令または FPSW を使用しています。CPU 種別を確認してください

A2022 (E) Symbol name is missing

EQU 制御命令行にシンボル名が未記述です。

A2023 (E) Illegal directive command is used

不正な制御命令を記述しています。

正しい制御命令に記述し直してください。

A2024 (E) No ';' at the top of comment

コメント先頭に ; が記述されていません。

コメントの先頭には、セミコロンを記述してください。ニーモニック又はオペランドの記述に誤りがないか確認してください。

A2026 (E) 'CODE' section in big endian is not appropriate

endian=big 時、絶対属性の CODE セクション開始アドレスに 4 の倍数以外の値を指定しています。

絶対属性の CODE セクション開始アドレスには 4 の倍数の値を指定してください。

A2027 (E) Illegal character code

文字コードが正しくありません。

A2028 (E) Unrecognized character escape sequence

認識できないエスケープシーケンスがあります。

A2029 (E) Invalid description in #pragma inline_asm function

アセンブリ記述関数内のアセンブリ言語に、使用できない記述があります。

C 言語ソースファイルで、#pragma inline_asm を指定した関数内の記述を確認してください。

A2040 (E) Include nesting over

インクルードのネストレベルが深すぎます。

インクルードのネストレベルが 30 以下になるように記述し直してください。

A2041 (E) Can't open include file 'XXXX'

インクルードファイルをオープンできません。

インクルードファイル名を確認してください。インクルードファイルの格納ディレクトリを確認してください。

A2042 (E) Including the include file in itself

インクルードファイル内で、自身をインクルードしています。

インクルードファイル名を確認して、記述し直してください。

A2049 (E) Invalid reserved word exist in operand

オペランド中に予約語が記述されています。

予約語は、オペランドに記述できません。オペランドを記述し直してください。

A2050 (E) Operand value is not defined

オペランドの値が未定義です。

オペランドには確定値を記述してください。

A2051 (E) '{' is missing

'{'の記述がありません。

A2052 (E) Addressing mode specifier is not appropriate

アドレッシングモード指定子の記述に間違いがあります。

アドレッシングモード指定子の記述方法を確認してください。

A2053 (E) Reserved word is missing

予約語の記述がありません。

A2054 (E) ']' is missing

'] ' の記述がありません。

'[' に対応する ']' を記述してください。

A2055 (E) Right quote is missing

右側の引用符がありません。

引用符を記述してください。

A2056 (E) The value is not constant

値がアセンブル時確定値ではありません。

アセンブル時に確定するような、式、シンボル名又はラベル名を記述してください。

A2057 (E) Quote is missing

文字列に対する引用符の記述がありません。

文字列は引用符で囲って記述してください。

A2058 (E) Illegal operand is used

オペランドが間違っています。

オペランドの記述方法を確認して、記述し直してください。

A2059 (E) Operand number is not enough

オペランドが不足しています。

オペランドの記述方法を確認して、記述し直してください。

A2060 (E) Too many macro nesting

マクロのネスティングが多すぎます。

マクロのネスティングレベルを 65535 レベル以下にしてください。ソース記述を確認してください。

A2061 (E) Too many macro local label definition

マクロ内ローカルラベルの定義が多すぎます。

マクロ内ローカルラベル数を 1 ファイル内に 65535 個以下にしてください。

A2062 (E) '.MACRO' is missing for '.ENDM'

.ENDM に対する .MACRO がありません。

.ENDM の記述位置を確認してください。

A2063 (E) '.MREPEAT' is missing for '.ENDR'

.ENDR に対する .MREPEAT がありません。

.ENDR の記述位置を確認してください。

A2064 (E) '.MACRO' or '.MREPEAT' is missing for '.EXITM'

.EXITM に対する .MACRO 又は .MREPEAT がありません。

.EXITM の記述位置を確認してください。

A2065 (E) No macro name

マクロ名がありません。

マクロ定義には、マクロ名を記述してください。

A2066 (E) Too many formal parameter

マクロの仮引数の定義数が多すぎます。

マクロの仮引数の数を 80 以下にしてください。

A2067 (E) Illegal macro parameter

マクロ引数に不正な記述があります。

マクロ引数の記述内容を確認してください。

A2068 (E) Source line is too long

ソース行が長すぎます。

ソース行の記述内容を確認してください。

A2069 (E) '.MACRO' is missing for '.LOCAL'

.LOCAL に対する .MACRO がありません。

.LOCAL の記述位置を確認してください。.LOCAL は、マクロブロック内にしか記述できません。

A2070 (E) No '.ENDM' statement

.ENDM 記述がありません。

.ENDM の記述位置を確認してください。.ENDM を記述してください。

A2071 (E) No '.ENDR' statement

.ENDR 記述がありません。

.ENDR の記述位置を確認してください。.ENDR を記述してください。

A2072 (E) ')' is missing

')'の記述がありません。

'('に対応する')'を記述してください。

A2073 (E) Operand expression is not completed

オペランド記述に不足があります。

オペランドの記述方法を確認して、記述し直してください。

A2074 (E) Syntax error in expression
式の記述に間違いがあります。
式の記述方法を確認して、記述し直してください。

A2075 (E) String value exist in expression
式中に文字列式が記述されています。
式を記述し直してください。

A2076 (E) Division by zero
除数０による除算が行われています。
式を記述し直してください。

A2077 (E) No '.END' statement
.END の記述がありません。
ソースプログラムの最後の行に.END を記述してください。

A2078 (E) The specified address overlaps at 'アドレス値'
指定された'アドレス値'でアドレス割り付けが重複しています。
.ORG、.OFFSET の指定内容を見直してください。
C/C++ソースの場合は'アドレス値'で複数の変数が重複しています。
'アドレス値'に割り付けようとしている変数を確認してください。

A2080 (E) '.IF' is missing for '.ELSE'
.ELSE に対する .IF がありません。
.ELSE の記述位置を確認してください。

A2081 (E) '.IF' is missing for '.ELIF'
.ELIF に対する .IF がありません。
.ELIF の記述位置を確認してください。

A2082 (E) '.IF' is missing for '.ENDIF'
.ENDIF に対する .IF がありません。
.ENDIF の記述位置を確認してください。

A2083 (E) Too many nesting level of condition assemble
条件アセンブルのネスティングが多すぎます。
条件アセンブルの記述を確認してください。

A2084 (E) No '.ENDIF' statement

ソースファイル内に IF 文に対応した ENDIF がありません。

ソースの記述を確認してください。

A2088 (E) Can't open '.ASSERT' message file 'XXXX'

.ASSERT の出力ファイルをオープンできません。

ファイル名を確認してください。

A2089 (E) Can't write '.ASSERT' message file 'XXXX'

.ASSERT の出力ファイルに書き込みできません。

ファイルのパーミッションを確認してください。

A2090 (E) Too many temporary label

テンポラリラベルの個数が多すぎます。

テンポラリラベルをラベル名に置き換えて記述してください。

A2091 (E) Temporary label is undefined

テンポラリラベルが未定義です。

テンポラリラベルの定義を行ってください。

A2100 (E) Value is out of range

値が範囲外です。

レジスタなどのビット長に合った値を記述してください。

A2111 (E) Symbol is undefined

シンボルが未定義です。

未定義のシンボル名は使用できません。前方参照となるシンボル名は記述できません。

シンボル名を確認してください。

A2112 (E) Symbol is missing

シンボルの記述がありません。

シンボル名を記述してください。

A2113 (E) Symbol definition is not appropriate

シンボルの定義に間違いがあります。

シンボル定義方法を確認して記述し直してください。

A2114 (E) Symbol has already defined as another type
シンボルは既に同一名で異なる制御命令で定義されています。
シンボル名を変更してください。

A2115 (E) Symbol has already defined as the same type
シンボルは、すでに定義されています。
シンボル名を変更してください。

A2116 (E) Symbol is multiple defined
シンボルが二重定義です。マクロ名と他の名前が重複しています。
シンボル名を変更してください。

A2117 (E) Invalid label definition
無効なラベル記述をしています。
ラベル定義を記述し直してください。

A2118 (E) Invalid symbol definition
無効なシンボル記述をしています。
シンボルの定義を記述し直してください。

A2119 (E) Reserved word is used as label or symbol
予約語をラベル又はシンボルに用いています。
ラベル又はシンボル名を記述し直してください。

A2130 (E) No '.SECTION' statement
'.SECTION 'の記述がありません。
ソースプログラムには、必ず１つ以上の.SECTION を記述してください。

A2131 (E) Section type is not appropriate
セクション属性の記述に適合しない命令や制御命令を記述しています。

A2132 (E) Section has already determined as attribute
セクションは既に相対属性に確定しています。制御命令".ORG"は記述できません。
セクションの属性を確認してください。

A2133 (E) Section attribute is not defined

セクションの属性が未定義です。このセクション内では制御命令".ALIGN"は記述できません。
制御命令".ALIGN"は、絶対アドレス属性セクション又は ALIGN 指定のある相対アドレス属性セクション内に記述してください。

A2134 (E) Section name is missing

セクション名がありません。
オペランドにセクション名を記述してください。

A2135 (E) 'ALIGN' is multiple specified in '.SECTION'

.SECTION 定義行に複数の 'ALIGN ' 指定があります。
余分な'ALIGN'指定を削除してください。

A2136 (E) Section type is multiple specified

セクション定義行でセクション属性の指定が重複しています。
セクション定義行には"CODE","DATA",ROMDATA"の指定は 1 つだけ記述してください。

A2137 (E) Too many operand

オペランドが余分にあります。
オペランドの記述内容を確認してください。

A3000 (F) Can't create file 'filename'

'filename'ファイルが生成できません。
ディレクトリ容量を確認してください。

A3001 (F) Can't open file 'filename'

'filename'ファイルがオープンできません。
ファイル名を確認してください。

A3002 (F) Can't write file 'filename'

'filename'ファイルに書き込むことができません。
ファイルのパーミッションを確認してください。

A3003 (F) Can't read file 'filename'

ファイルを読み込むことができません。
ファイルのパーミッションを確認してください。

A3004 (F) Can't create Temporary file

テンポラリファイルが生成できません。

カレントディレクトリ以外にテンポラリファイルを作成するように、環境変数'TMP_RX'にディレクトリを指定してください。

A3005 (F) Can't open Temporary file

テンポラリファイルがオープンできません。

‘TMP_RX’で指定したディレクトリを確認してください。

A3006 (F) Can't read Temporary file

テンポラリファイルを読み込むことができません。

‘TMP_RX’で指定したディレクトリを確認してください。

A3007 (F) Can't write Temporary file

テンポラリファイルに書き込むことができません。

‘TMP_RX’で指定したディレクトリを確認してください。

A3008 (F) Illegal file name 'filename'

ファイル名が不正です。

ファイル名の記述規則に従ったファイル名を指定してください。

A3100 (F) Command line is too long

コマンド行の文字数が多すぎます。

コマンドを入力し直してください。

A3101 (F) Invalid option 'xx' is used

無効なコマンドオプション xx を使用しています。

指定したオプションは存在しません。コマンドを入力し直してください。

A3102 (F) Ignore option 'xx'

無効なオプションが指定されています。

A3103 (F) Option 'xx' is not appropriate

コマンドオプション xx の記述が正しくありません。

コマンドオプションを指定し直してください。

A3104 (F) No input files specified

入力ファイルの指定がありません。

入力ファイルを指定してください。

A3105 (F) Source files number exceed 80

ファイルの数が 80 を超えています。

複数回に分けてアセンブルを実行してください。

A3106 (F) Lacking cpu specification

CPU の指定がされていません。

cpu オプションまたは環境変数 CPU_RX で CPU を指定してください。

A3110 (F) Multiple register base/fint_register

base と fint_register オプションで指定レジスタが重複しています。

A3111 (F) Multiple register base/pid

base と pid オプションで指定レジスタが重複しています。

A3112 (F) Multiple register base/nouse_pid_register

base と nouse_pid_register オプションで指定レジスタが重複しています。

A3200 (F) Error occurred in executing 'xxx'

xxx の実行でエラーが発生しました。

再度 asrx を実行し直してください。

A3201 (F) Not enough memory

メモリが足りません。

ファイルを分割して実行し直してください。又はメモリを増設してください。

A3202 (F) Can't find work dir

ワークディレクトリが見つかりません。

環境変数 TMP_RX が正しく設定されているかを確認してください。

A4000-A4999 (-) Internal error

アセンブラの内部処理で何らかの障害が生じました。本製品をお求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。

13. 最適化リンケージエディタのエラーメッセージ

13.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
エラー内容

エラーレベルは、エラーの重要度に従い、5 種類に分類されます。

	エラーレベル		動作
L0000 - L0999 P0000 - P0999	(I)	インフォメーション	処理を継続します。
L1000 - L1999 P1000 - P1999	(W)	ウォーニング	処理を継続します。
L2000 - L2999 P2000 - P2999	(E)	エラー	オプション解析処理を継続し、処理を中断します。
L3000 - L3999 P3000 - P3999	(F)	フェータル	処理を中断します。
L4000 - P4000 -	(-)	インターナル	処理を中断します。

L で始まるエラー番号は、最適化リンケージエディタ出力メッセージです。

P で始まるエラー番号は、プレリンカ出力メッセージです。P で始まるエラー番号は、nomessage オプションや change_message オプションで指定できません。

13.2 エラーの返値

最適化リンケージエディタは処理を終了した際、処理結果によって次の値を OS に返します。

返値	内容
0	正常終了、インフォメーションメッセージ出力後終了、およびウォーニング出力後終了します。
1	エラー、フェータル、インターナルおよび強制終了します。

13.3 メッセージ一覧

L0001 (I) Section "セクション" created by optimization "最適化"
"最適化"の最適化によって、"セクション"を作成しました。

L0002 (I) Symbol "シンボル" created by optimization "最適化"
"最適化"の最適化によって、"シンボル"を作成しました。

L0003 (I) "ファイル"-"シンボル" moved to "セクション" by optimization
variable_access の最適化によって、"ファイル"内の"シンボル"を移動しました。

L0004 (I) "ファイル"-"シンボル" deleted by optimization
symbol_delete の最適化によって、"ファイル"内の"シンボル"を削除しました。

L0005 (I) The offset value from the symbol location has been changed by optimization :
"ファイル"-"セクション"-"シンボル±offset"
"シンボル±offset"の範囲で最適化によるサイズ変更があったため offset 値を変更しました。問題ないか確認してください。offset 値の変更を抑止したい場合は、"ファイル"のアセンブル時に goptimize オプション指定を外してください。

L0100 (I) No inter-module optimization information in "ファイル"
"ファイル"内にモジュール間最適化情報がありません。"ファイル"をモジュール間最適化の対象外にします。モジュール間最適化の対象にする場合は、コンパイル、アセンブル時に goptimize オプションを指定してください。ただし、asmsh には goptimize オプションはありません。

L0101 (I) No stack information in "ファイル"
"ファイル"内にスタック情報がありません。"ファイル"はアセンブラ出力ファイルまたは SYSROF->ELF コンバートファイルの可能性があります。最適化リンケージエディタが出力するスタック情報ファイルに当該ファイルの内容は含まれません。

L0102 (I) Stack size "サイズ" specified to the undefined symbol "シンボル" in "ファイル"
"ファイル"内の未定義シンボル"シンボル"に、スタックサイズ "サイズ" が指定されています。

L0103 (I) Multiple stack sizes specified to the symbol "シンボル"
シンボル"シンボル"は、複数のスタックサイズが指定されています。

L0300 (I) Mode type "モード種別１" in "ファイル" differ from "モード種別２"
異なるモード種別のファイルを入力しました。

L0400 (I) Unused symbol "ファイル"-"シンボル"
"ファイル"内の"シンボル"は使用されていません。

L0500 (I) Generated CRC code at "アドレス"
"アドレス"に CRC コードを出力しました.

L0510 (I) Section "セクション" was moved other area specified in option "cpu=<メモリ属性>"
セクションを分割せずに cpu=<メモリ属性>にしたがって " セクション " を配置しました。

L0511 (I) Sections "セクション名","分割後のセクション名" are Non-contiguous
"セクション名"のセクションを分割し、"分割後のセクション名"のセクションを生成しました。

L1000 (W) Option "オプション" ignored
"オプション"は無効です。"オプション"を無視します。

L1001 (W) Option "オプション 1" is ineffective without option "オプション 2"
"オプション 1"は"オプション 2"が必要です。"オプション 1"を無視します。

L1002 (W) Option "オプション 1" cannot be combined with option "オプション 2"
"オプション 1"と"オプション 2"は同時に指定できません。"オプション 1"を無視します。

L1003 (W) Divided output file cannot be combined with option "オプション"
"オプション"指定時、出力ファイルの分割指定はできません。オプションの指定を無視します。先頭入力ファイル名を出力ファイル名として使用します。

L1004 (W) Fatal level message cannot be changed to other level : "番号"
Fatal レベルメッセージはレベル変更できません。"番号"の指定を無視します。change_message オプションで変更できるエラーは、Information/Warning/Error レベルです。

L1005 (W) Subcommand file terminated with end option instead of exit option
end オプションの後に処理指定がありません。exit オプションを仮定して処理します。

L1006 (W) Options following exit option ignored
exit オプションの後のオプションを無視しました。

L1007 (W) Duplicate option : "オプション"

"オプション"が重複しています。最後に指定したオプションを有効にします。

L1008 (W) Option "オプション" is effective only in cpu type "マイコン種別"

"オプション"は"マイコン種別"以外では無効です。"オプション"を無視します。

L1010 (W) Duplicate file specified in option "オプション" : "ファイル名"

"オプション"で同じファイルを 2 度指定しました。2 度目の指定を無視します。

L1011 (W) Duplicate module specified in option "オプション" : "モジュール"

"オプション"で同じモジュールを 2 度指定しました。2 度目の指定を無視します。

L1012 (W) Duplicate symbol/section specified in option "オプション" : "名前"

"オプション"で同じシンボル名またはセクション名を 2 度指定しました。2 度目の指定を無視します。

L1013 (W) Duplicate number specified in option "オプション" : "番号"

"オプション"で同じエラー番号を指定しました。最後に指定した方を有効にします。

L1100 (W) Cannot find "名前" specified in option "オプション"

"オプション"で指定したシンボル名またはセクション名が見つかりません。"名前"の指定を無視します。

L1101 (W) "名前" in rename option conflicts between symbol and section

rename オプションで指定した"名前"がセクション名とシンボル名の両方に存在します。
シンボル名を変更の対象にします。

L1102 (W) Symbol "シンボル" redefined in option "オプション"

"オプション"で指定したシンボルはすでに定義されています。そのまま処理を続けます。

L1103 (W) Invalid address value specified in option "オプション" : "アドレス"

"オプション"で指定した"アドレス"は無効な値です。"アドレス"の指定を無視します。

L1104 (W) Invalid section specified in option "オプション" : "セクション"

"オプション"で無効なセクションを指定しています。以下を確認してください。

- -outputオプションは、初期値のないセクションを指定できません。
- -jump_entries_for_picオプションは、プログラムセクション以外を指定できません。

L1110 (W) Entry symbol "シンボル" in entry option conflicts

entry オプションで指定した"シンボル"以外のシンボルがコンパイル、アセンブル時にエントリシンボルとして指定されています。オプション指定を優先します。

L1120 (W) Section address is not assigned to "セクション"

"セクション"のアドレス指定がありません。"セクション"を最後尾に配置します。
optlnk オプション-start を使用して、セクションのアドレスを設定してください。

L1121 (W) Address cannot be assigned to absolute section "セクション" in start option

"セクション"は絶対アドレスセクションです。絶対アドレスセクションに対するアドレス指定を無視します。

L1122 (W) Section address in start option is incompatible with alignment : "セクション"

start オプションで指定した"セクション"のアドレスはアライメント数と矛盾しています。アライメント数に合わせてセクションアドレスを補正します。

L1130 (W) Section attribute mismatch in rom option :
"セクション 1,セクション 2"

rom オプションで指定した"セクション 1"と"セクション 2"の属性、アライメント数が異なります。"セクション 2"のアライメント数はどちらか大きい方を有効とします。

L1140 (W) Load address overflowed out of record-type in option "オプション"

アドレス値よりも小さい record 形式を指定しました。指定した record 形式を超える範囲は、別の record 形式で出力します。

L1141 (W) Cannot fill unused area from "アドレス" with the specified value

空きエリアのサイズが space オプションで指定された値の倍数となっていないため、"アドレス"以降に指定データを出力できませんでした。

L1150 (W) Sections in "オプション" option have no symbol

"オプション" で指定したセクションは外部定義シンボルがありません。

L1160 (W) Undefined external symbol "シンボル"

未定義の"シンボル"を参照しています。

L1170 (W) Specified SBR addresses conflict

異なる複数の SBR アドレスが指定されました。SBR=USER として処理します。

L1171 (W) Least significant byte in SBR="定数" ignored

SBR オプションで指定されたアドレス"定数"の下位 8bit は無効です。

L1180 (W) Directive command "制御命令" is duplicated in "ファイル"

複数のソースファイルに、"制御命令"を記述しています。

"制御命令"は、複数記述することはできません。

L1181 (W) Fail to write "出力コード種別"

出力ファイルへの、"出力コード種別"の書き込みが失敗しました。

出力ファイルに、"出力コード種別"の書き込み先アドレスが含まれていない可能性があります。

出力コード種別:

ID コード書き込み失敗時・・・「"ID Code"」

→ L1181 Fail to write "ID Code"

PROTECT/OFSREG コード書き込み失敗時・・・「"Protect Code" or "OFSREG Code"」

→ L1181 Fail to write "Protect Code" or "OFSREG Code"

CRC コード書き込み失敗時・・・「"CRC Code"」

→ L1181 Fail to write "CRC Code"

L1182 (W) Cannot generate vector table section "セクション"

入力ファイル内に、ベクタテーブル"セクション"があります。リンカは、"セクション"を自動生成しません。

L1183 (W) Interrupt number "ベクタ番号" of "セクション" is defined in input file

VECTN オプションで記述したベクタ番号は、入力ファイル内で定義済みです。入力ファイルの内容を優先して、処理を継続します。

L1190 (W) Section "セクション" was moved other area specified in option "cpu=<メモ属性>"

外部変数アクセス最適化によりオブジェクトサイズが変更されたため、次の cpu 指定範囲の”セクション”を移動しました。

L1191 (W) Area of "FIX" is within the range of the area specified by “cpu=<メモリ属性>” :"<start>-<end>"

cpu オプションで、メモリ属性 FIX と FIX 以外の<start>-<end>範囲が重複していたため、FIX を有効にしました。

L1192 (W) Bss Section "セクション名" is not initialized

初期値なしのデータセクション”セクション名”は、初期設定プログラムで初期化できません。-cpu 指定範囲、ポインタ変数のサイズ指定を見直してください。

L1193 (W) Section "セクション名" specified in option "オプション" is ignored

-cpu=stride の機能で分割したセクションの、後半部への"オプション"指定は無効となります。後半部のセクションは"オプション"で指定しないでください。

L1194 (W) Section "セクション" in relocation "ファイル"-"セクション"-"オフセット" is changed.

"セクション""ファイル""オフセット" の位置にある"セクション"を参照していたリロケーションが、分割した後半セクションを参照するよう変更しました。分割しないようにするには、"セクション"を contiguous_section オプションで指定してください。

L1200 (W) Backed up file "ファイル 1" into "ファイル 2"

入力ファイル"ファイル 1"は書き換えられました。
書き換える前の"ファイル 1"の内容は"ファイル 2"にバックアップされています。

L1300 (W) No debug information in input files

入力ファイル内にデバッグ情報がありません。debug,sdebug,compress オプション指定を無視します。コンパイル、アセンブル時に該当するオプションを指定しているか確認してください。

L1301 (W) No inter-module optimization information in input files

入力ファイル内にモジュール間最適化情報がありません。optimize オプションを無視します。コンパイル、アセンブル時に goptimize オプションを指定してください。

L1302 (W) No stack information in input files

入力ファイル内にスタック情報がありません。stack オプションを無視します。入力ファイルがアセンブラ出力ファイルまたは SYSROF->ELF コンバートファイルの場合は、stack オプションは無効です。

L1303 (W) No rts information in input files

.rts ファイルを生成可能な入力ファイルがありません。
.rts ファイルを生成せずに処理を終了します。

L1304 (W) No utl information in input files

utl ファイルを生成するための情報が入力されませんでした。

L1305 (W) Entry address in "ファイル" conflicts : "アドレス"

異なるエントリーアドレスのファイルが複数入力されています。

L1310 (W) "セクション" in "ファイル" is not supported in this tool

"ファイル"内に非サポートセクションがありました。"セクション"を無視します。

L1311 (W) Invalid debug information format in "ファイル"

"ファイル"内のデバッグ情報は dwarf2 ではありません。debug 情報を削除します。

L1320 (W) Duplicate symbol "シンボル" in "ファイル"

"シンボル"は重複しています。先に入力したファイル内シンボルを優先します。

L1321 (W) Entry symbol "シンボル" in "ファイル" conflicts

エントリシンボル定義のあるオブジェクトファイルを複数入力しました。先に入力したファイル内のエントリシンボルを有効にします。

L1322 (W) Section alignment mismatch : "セクション"

アライメント数の異なる同名セクションを入力しました。アライメント数は最大の指定を有効にします。

L1323 (W) Section attribute mismatch : "セクション"

属性の異なる同名セクションを入力しました。絶対セクションと相対セクションの場合は、絶対セクションとして扱います。read/write 属性が異なる場合は、どちらも許可します。

L1324 (W) Symbol size mismatch : "シンボル" in "ファイル"

サイズの異なるコモンシンボルまたは定義シンボルが入力されました。定義シンボルを優先します。コモンシンボル同士の場合は、先に入力したファイル内シンボルを優先します。

L1325 (W) Symbol attribute mismatch : "シンボル"："ファイル"

“ファイル”内の”シンボル”が、他のファイルの同名シンボルと属性が一致していません。シンボルを確認してください。

L1326 (W) Reserved symbol "シンボル”is defined in "ファイル”

予約された名称のシンボル"シンボル"が"ファイル"内で定義されています。

L1327 (W) Section alignment in option "aligned_section" is small : "セクション"
aligned_section オプション指定時のアライメント数 16 の方が、"セクション"のアライメント数より小さいため、指定セクションに対するオプション指定を無視します。

L1330 (W) Cpu type "マイコン種別1" in "ファイル" differ from "マイコン種別2"
異なるマイコン種別のファイルを入力しました。マイコン種別を H8SX として処理を継続します。

L1400 (W) Stack size overflow in register optimization
レジスタ最適化で、スタックアクセスコードがコンパイラのスタック量制限値を超えました。レジスタ最適化指定を無視します。

L1401 (W) Function call nest too deep
関数の呼び出しネストが深すぎるため、レジスタ最適化を実施できません。

L1402 (W) Parentheses specified in option "start" with optimization
start オプションで括弧"()"を記述した場合、最適化機能は使用できません。
最適化機能を無効にします。

L1410 (W) Cannot optimize "ファイル"-"セクション" due to multi label relocation operation
複数ラベルのリロケーション演算を持つセクションは最適化できません。"ファイル"内の"セクション"を最適化対象外にします。

L1420 (W) "ファイル" is newer than "プロファイル"
"ファイル"は"プロファイル"より後に更新されました。プロファイル情報を無視します。

L1430 (W) Cannot generate effective bls file for compiler optimization
無効な bls ファイルが生成されました。コンパイル時に、外部変数アクセス最適化(map オプション)を指定しても、この最適化は実施できません。
コンパイラの外部変数アクセス最適化(map オプション)には、以下の制限があります。該当する内容がないかを確認し、セクション配置を見直してください。
コンパイル時に base オプションを使用している場合、プログラムセクションの直後にデータセクションを配置すると、外部変数アクセス最適化が実施できない場合があります。
※bls ファイルは"外部シンボル割り付け情報ファイル"を指します。コンパイラの map オプションに使用するための情報ファイルです。

L1500 (W) Cannot check stack size

スタックセクションがないため、コンパイル時の stack オプションで指定したスタックサイズの整合性をチェックできません。コンパイル時の stack オプションの整合性をチェックするためにはコンパイル時、アセンブル時に goptimize オプション指定が必要です。

L1501 (W) Stack size overflow : "スタックサイズ"

スタックセクションサイズが、コンパイル時に stack オプションで指定した"スタックサイズ"を超えました。コンパイル時のオプションを変更するか、スタック量を削減できるようにプログラムを変更してください。

L1502 (W) Stack size in "ファイル" conflicts with that in another file

複数のファイルで異なるスタックサイズを指定されています。コンパイル時のオプションを確認してください。

L1510 (W) Input file was compiled with option "smap" and option "map" is specified at linkage

"smap を指定してコンパイルしたファイルがあります。smap を指定したファイルは、2 回目のビルドで map オプションを指定してコンパイルしないでください。

P1600 (W) An error occurred during name decoding of "インスタンス"

"インスタンス"はデコードできませんでした。エンコード名でメッセージ出力します。

L2000 (E) Invalid option : "オプション"

P2000 (E) Invalid option : "オプション"

"オプション"はサポートしていません。

L2001 (E) Option "オプション" cannot be specified on command line

"オプション"はコマンドライン上では指定できません。サブコマンドファイル内で指定してください。

L2002 (E) Input option cannot be specified on command line

コマンドライン上で input オプションを指定しました。コマンドライン上での入力ファイル指定は input オプション無しで指定してください。

L2003 (E) Subcommand option cannot be specified in subcommand file

サブコマンドファイル内に subcommand オプションを指定しました。subcommand オプションはネストできません。

L2004 (E) Option "オプション 1" cannot be combined with option "オプション 2"
"オプション 1"と"オプション 2"は同時に指定できません。

L2005 (E) Option "オプション" cannot be specified while processing "プロセス"
"プロセス"処理に対して"オプション"は指定できません。

L2006 (E) Option "オプション 1" is ineffective without option "オプション 2"
"オプション 1"は"オプション 2"が必要です。

L2010 (E) Option "オプション" requires parameter
"オプション"はパラメータ指定が必要です。

L2011 (E) Invalid parameter specified in option "オプション" : "パラメータ"
"オプション"で無効なパラメータを指定しました。

L2012 (E) Invalid number specified in option "オプション" : "値"
"オプション"指定で無効な値を指定しました。値の範囲を確認してください。

L2013 (E) Invalid address value specified in option "オプション" : "アドレス"
"オプション"で指定した"アドレス"は無効な値です。0～FFFFFFFF の間の 16 進数で指定してください。

L2014 (E) Illegal symbol/section name specified in "オプション" : "名前"
"オプション"で指定したセクションまたはシンボル名に不正文字が使用されています。セクション/シンボル名で使用できるのは数字、英字、_、$(先頭は数字以外)です。

L2016 (E) Invalid alignment value specified in option "オプション" : "アライメント数"
"オプション"で指定した"アライメント数"は無効な値です。
1,2,4,8,16 または 32 を指定してください。

L2017 (E) Cannot output “セクション” specified in option “オプション”
"オプション"で指定した"セクション"のコードの一部を出力できません。命令コードのエンディアンを変換したことにより、"セクション"内命令コードの一部が非連続となりました。非連続部分の命令コードが属しているセクションは、リンケージリストからセクションアドレスを 4 バイト境界で確認の上、出力するセクションがどのセクションとエンディアン変換を行っているか確認してください。

L2020 (E) Duplicate file specified in option "オプション" : "ファイル"
"オプション"指定で同じファイルを 2 度指定しました。

L2021 (E) Duplicate symbol/section specified in option
"オプション" : "名前"
"オプション"指定で同じシンボル名またはセクション名を 2 度指定しました。

L2022 (E) Address ranges overlap in option "オプション" : "アドレス範囲"
"オプション"で指定した"アドレス範囲"が重複しています。

L2100 (E) Invalid address specified in cpu option : "アドレス"
cpu オプションで cpu では指定できないアドレスを指定しました。

L2101 (E) Invalid address specified in option "オプション" : "アドレス"
"オプション"で指定した"アドレス"は cpu で指定できるアドレス範囲、または cpu オプションで指定した範囲を超えました。

L2110 (E) Section size of second parameter in rom option is not 0 : "セクション"
rom オプションの第 2 パラメータにサイズが 0 でない"セクション"を指定しました。

L2111 (E) Absolute section cannot be specified in "オプション" option : "セクション"
"オプション"で絶対アドレスセクションを指定しました。

L2112 (E) "セクション 1" and "セクション 2" cannot mapped as ROM/RAM in "ファイル"
"ファイル名"で指定された"セクション 1"と"セクション 2"は ROM/RAM 対応となりません。

L2113 (E) Option "rom" and internal information in the file are conflicted
rom オプションの指定と内部情報が矛盾しています。

L2120 (E) Library "ファイル" without module name specified as input file
入力ファイルとしてモジュール名なしのライブラリファイルを指定しました。

L2121 (E) Input file is not library file : "ファイル(モジュール)"
入力ファイルで指定した"ファイル(モジュール)"はライブラリファイルではありません。

L2130 (E) Cannot find file specified in option "オプション" : "ファイル"
"オプション"で指定したファイルが見つかりません。

L2131 (E) Cannot find module specified in option "オプション" : "モジュール"
"オプション"で指定したモジュールがありません。

L2132 (E) Cannot find "名前" specified in option "オプション"
"オプション"で指定したシンボルまたはセクションが存在しません。

L2133 (E) Cannot find defined symbol "名前" in option "オプション"
"オプション"で指定した外部定義シンボルが存在しません。

L2140 (E) Symbol/section "名前" redefined in option "オプション"
"オプション"で指定したシンボル、セクションはすでに定義されています。

L2141 (E) Module "モジュール" redefined in option "オプション"
"オプション"で指定したモジュールはすでに登録されています。

L2142 (E) Interrupt number "ベクタ番号" of "セクション" has multiple definition
ベクタテーブル"セクション"の、ベクタ番号定義が複数入力されました。ベクタ番号には、ひとつのアドレスしか設定できません。ソースファイルの記述を見直してください。

L2143 (E) Invalid vector number specified : "number"
number で示すベクタ番号は指定できません。
#pragma special で指定したベクタ番号を見直してください。

L2200* (E) Illegal object file : "ファイル"
ELF フォーマット以外を入力しました。
* P2200 と表示される場合があります。

L2201 (E) Illegal library file : "ファイル"
"ファイル"はライブラリファイルではありません。

L2202 (E) Illegal cpu information file : "ファイル"
"ファイル"はマイコン情報ファイルではありません。

L2203 (E) Illegal profile information file : "ファイル"
"ファイル"はプロファイル情報ファイルではありません。

L2210 (E) Invalid input file type specified for option "オプション" : "ファイル(種別)"
"オプション"指定時に処理できない"ファイル(種別)"を入力しました。

L2211 (E) Invalid input file type specified while processing
"プロセス" : "ファイル(種別)"
"プロセス"処理に対して処理できない"ファイル(種別)"を入力しました。

L2212 (E) "オプション" cannot be specified for inter-module optimization
information in "ファイル"
"ファイル"内にモジュール間最適化情報があるため、"オプション"オプションは使用できません。コンパイル、アセンブル時に goptimize オプションを使用しないでください。

L2220 (E) Illegal mode type "モード種別" in "ファイル"
異なる"モード種別"のファイルを入力しました。

L2221 (E) Section type mismatch : "セクション"
属性(初期値有無)の異なる同名セクションを入力しました。

L2223 (E) Cpu type "CPU 種別 1" in "ファイル" is incompatible with "CPU 種別 2"
異なる CPU 種別を入力しました。
一部の仕様に互換性がないため、リンクしても動作が保証できません。

L2300 (E) Duplicate symbol "シンボル" in "ファイル"
"シンボル"は重複しています。

L2301 (E) Duplicate module "モジュール" in "ファイル"
"モジュール"は重複しています。

L2310 (E) Undefined external symbol "シンボル" referenced in "ファイル"
"ファイル"内で未定義の"シンボル"を参照しています。

L2311 (E) Section "セクション 1" cannot refer to overlaid section : "セクション 2"-"シンボル"
同一アドレスを指定したオーバーレイセクション間でシンボル参照がありました。
"セクション 1"と"セクション 2"を同じアドレスに割り付けないでください。

L2320 (E) Section address overflowed out of range : "セクション"

"セクション"のアドレスが使用可能なアドレス範囲を超えました。

L2321 (E) Section "セクション１" overlaps section "セクション２"

"セクション 1"と"セクション 2"のアドレスが重複しました。start オプションのアドレス指定を変更してください。

L2322 (E) Section size too large: "セクション"

セクション"セクション"のサイズが大きすぎます。

$TBR セクションのサイズは 1024 バイト以内でなければなりません。

L2323 (E) Section "セクション 1(アドレス範囲)" overlaps with section "セクション 2(アドレス範囲)" in physical space

物理メモリの配置上で、"セクション 1"と"セクション 2"が重複しています。

各セクションの配置アドレスを見直してください。

<アドレス範囲> : <セクションの開始アドレス>-<セクションの終端アドレス>

L2330 (E) Relocation size overflow : "ファイル"-"セクション"-"オフセット"

リロケーション演算結果がリロケーションサイズを超えました。分岐先が届かない、特定のアドレスに配置しなければならないシンボルを参照しているなどが考えられます。コンパイル、アセンブルリストで、"セクション"の"オフセット"位置の参照シンボルが正しい位置に配置されているか確認してください。

L2331 (E) Division by zero in relocation value calculation : "ファイル"-"セクション"-"オフセット"

リロケーション演算に 0 除算が発生しました。コンパイル、アセンブルリストで、"セクション"の"オフセット"位置の演算に問題がないか確認してください。

L2332 (E) Relocation value is odd number : "ファイル"-"セクション"-"オフセット"

リロケーション演算結果が奇数になりました。コンパイル、アセンブルリストで、"セクション"の"オフセット"位置の演算に問題がないか確認してください。

L2340 (E) Symbol name "ファイル"-"セクション"-"シンボル..." is too long

"セクション"内の"シンボル"の文字数がアセンブラの翻訳限界を超えました。

シンボルアドレスファイルを出力する場合は、アセンブラの翻訳限界文字数以下になるようなシンボル名としてください。

L2400 (E) Global register in "ファイル" conflicts : "シンボル","レジスタ"

"ファイル"内で指定したグローバルレジスタにはすでに別のシンボルが割り付いています。

L2401 (E) near8,near16 symbol "シンボル" is outside near memory area

"シンボル"は near8、near16 の範囲に割りついていません。start 指定を変更するか、コンパイル時の near 指定を外して、正しいアドレス計算ができるようにしてください。

L2402 (E) Number of register parameter conflicts with that in another file : "関数"

"関数"は複数のファイルで異なるレジスタパラメータ数を指定されています。

L2403 (E) Fast interrupt register in "ファイル" conflicts with that in another file

"ファイル"内で指定した高速割り込み用汎用レジスタ番号が、他ファイルと統一されていません。高速割り込み用汎用レジスタ番号を他ファイルに合わせて、再度コンパイルして下さい。

L2404 (E) Base register "ベースレジスタ種別" in "ファイル" conflicts with that in another file

"ファイル"内で指定した"ベースレジスタ種別"用のレジスタ番号が、他ファイルと統一されていません。ベースレジスタ番号を他ファイルに合わせて、再度コンパイルして下さい。

L2405 (E) Option "コンパイルオプション" conflicts with that in other files

"コンパイルオプション"の指定が入力ファイル間で統一されていません。
コンパイルオプションを見直してください。

L2410 (E) Address value specified by map file differs from one after linkage as to "シンボル"

"シンボル"のアドレス値がコンパイル時に使用した外部シンボル割り付け情報ファイル内のアドレスとリンク後のアドレスで異なっています。下記の(1)～(3)を確認してください。

(1) コンパイル時の map オプション指定前後でプログラムを変更している場合は、プログラムの変更をやめてください。

(2) optlnk の最適化によって、コンパイル時の map オプション指定前後のシンボル並び順が変わることがあります。コンパイル時 map オプションを無効にするか、optlnk の最適化オプションを無効にしてください。

(3) tbr オプションまたは#pragma tbr 使用時、コンパイラの最適化によって、コンパイル時の map オプション指定後のシンボルが削除されることがあります。コンパイル時 map オプションを無効にするか、tbr オプションまたは#pragma tbr を無効にしてください。

L2411 (E) Map file in "ファイル" conflicts with that in another file

入力ファイル間でコンパイル時に異なる外部シンボル割り付け情報ファイルを使用しています。

L2412 (E) Cannot open file : "ファイル"

"ファイル"(外部シンボル割り付け情報ファイル)がオープンできません。ファイル名およびアクセス権が正しいか確認してください。

L2413 (E) Cannot close file : "ファイル"

"ファイル"(外部シンボル割り付け情報ファイル)がクローズできません。ディスク容量に空きがない可能性があります。

L2414 (E) Cannot read file : "ファイル"

"ファイル"(外部シンボル割り付け情報ファイル)が読みこめません。ディスク容量に空きがない可能性があります。

L2415 (E) Illegal map file : "ファイル"

"ファイル"(外部シンボル割り付け情報ファイル)のフォーマットが不正です。ファイル名が正しいか確認してください。

L2416 (E) Order of functions specified by map file differs from one after linkage as to "関数名"

関数"関数名"は、コンパイル時に使用した外部シンボル割り付け情報ファイル内の情報とリンク後の配置とで、他の関数との並び順が異なっています。関数内 static 変数のアドレスが、外部シンボル割り付け情報ファイルとリンク後の結果とで異なっている可能性があります。

L2417 (E) Map file is not the newest version: "ファイル名"

bls ファイルが最新バージョンではありません。

L2420 (E) "ファイル 1" overlap address "ファイル 2" : "アドレス"

ファイル 1 とファイル 2 のアドレスが重複しています。

P2500 (E) Cannot find library file : "ファイル"

ライブラリとして指定した"ファイル"がありません。

P2501 (E) "インスタンス" has been referenced as both an explicit specialization and a generated instantiation

すでに定義が存在しているインスタンスに対して、インスタンス生成を要求しています。
"インスタンス"を使用しているファイルに対して、form=relocate でリロケータブルファイルを作成していないか確認してください。

P2502 (E) "インスタンス" assigned to "ファイル 1" and "ファイル 2"

"ファイル 1"と"ファイル 2"に"インスタンス"定義が重複しています。
"インスタンス"を使用しているファイルに対して、form=relocate でリロケータブルファイルを作成していないか確認してください。

L3000 (F) No input file

入力ファイルがありません。

L3001 (F) No module in library

ライブラリ内のモジュール数が 0 になりました。

L3002 (F) Option "オプション 1" is ineffective without option "オプション 2"

"オプション 1"は"オプション 2"が必要です。

L3004 (F) Unsupported inter-module optimization information type “タイプ” in “ファイル”

ファイル内にサポートしていないモジュール間最適化情報”タイプ”がありました。コンパイラ、アセンブラのバージョンが正しいか確認してください。

P3005 (F) Instantiation loop

インスタンス生成処理がループしています。
入力ファイル名が別ファイルのファイルと一致している可能性があります。拡張子を除いたファイル名が一致しないようにファイル名を変更してください。

P3007 (F) Cannot create instantiation request file "ファイル"

インスタンス生成処理用の中間ファイルを作成できません。
オブジェクト作成フォルダ以下のアクセス権が正しいか確認してください。

P3008 (F) Cannot change to directory "フォルダ"

"フォルダ"に移動できません。"フォルダ"が存在するか確認してください。

P3009 (F) File "ファイル" is read-only

"ファイル"は読み取り専用です。アクセス権を変更してください。

L3100 (F) Section address overflow out of range : "セクション"

"セクション"のアドレスが使用可能な上限の領域を超えました。

start オプションのアドレス指定を変更してください。

アドレス空間の詳細については各マイコンのハードウェアマニュアルを参照してください。

L3102 (F) Section contents overlap in absolute section "セクション"

絶対アドレスセクションのセクション内データアドレスが重複しています。ソースプログラムを修正してください。

L3110 (F) Illegal cpu type "マイコン種別" in "ファイル"

異なるマイコン種別のファイルを入力しました。

L3111 (F) Illegal encode type "エンディアン種別" in "ファイル"

異なるエンディアン種別のファイルを入力しました。

L3112 (F) Invalid relocation type in "ファイル"

"ファイル"内にサポートしていないリロケーションタイプがありました。コンパイラ、アセンブラのバージョンが正しいか確認してください。

L3120 (F) Illegal size of the absolute code section : "セクション" in "ファイル"

"ファイル"に存在する絶対アドレスプログラムセクション"セクション" のサイズが不正です。CPU 種別が RX ファミリでビッグエンディアンの場合は、絶対アドレスプログラムセクションのサイズが 4 の倍数になるように変更してください。

L3200 (F) Too many sections

セクション数が翻訳限界を超えました。複数ファイル出力を指定すると解決できる可能性があります。

L3201 (F) Too many symbols

シンボル数が翻訳限界を超えました。複数ファイル出力を指定すると解決できる可能性があります。

L3202 (F) Too many modules

モジュール数が翻訳限界を超えました。ライブラリを分けて作成してください。

L3203 (F) Reserved module name “optlnk_generates”

optlnk_generates_** (**は、01～99 までの数値)は、最適化リンケージエディタで使用する予約名称です。.obj/.rel ファイル名およびライブラリ内モジュール名として使用しています。ファイル名およびライブラリ内モジュール名で使用している場合は、変更してください。

L3300* (F) Cannot open file : "ファイル"

"ファイル"をオープンできません。ファイル名およびアクセス権が正しいか、確認してください。

* P3300 と表示される場合があります。

L3301 (F) Cannot close file : "ファイル"

"ファイル"をクローズできません。ディスク容量に空きがない可能性があります。

L3302 (F) Cannot write file : "ファイル"

"ファイル"に書き込めません。ディスク容量に空きがない可能性があります。

L3303* (F) Cannot read file : "ファイル"

"ファイル"を読めません。空ファイルを入力したか、ディスク容量に空きがない可能性があります。

* P3303 と表示される場合があります。

L3310* (F) Cannot open temporary file

中間ファイルをオープンできません。HLNK_TMP 指定が正しいか確認してください。

またはディスク容量に空きがない可能性があります。

* P3310 と表示される場合があります。

L3311 (F) Cannot close temporary file

中間ファイルをクローズできません。ディスク容量に空きがない可能性があります。

L3312 (F) Cannot write temporary file

中間ファイルに書き込めません。ディスク容量に空きがない可能性があります。

L3313 (F) Cannot read temporary file

中間ファイルを読めません。ディスク容量に空きがない可能性があります。

L3314 (F) Cannot delete temporary file

中間ファイルを削除できません。ディスク容量に空きがない可能性があります。

L3320* (F) Memory overflow

最適化リンケージエディタが内部で使用するメモリが不足しています。メモリを増やしてください。

* P3320 と表示される場合があります。

L3400 (F) Cannot execute "ロードモジュール"

"ロードモジュール"を起動できません。"ロードモジュール"のパスが設定されているか確認してください。

L3410 (F) Interrupt by user

標準入力端末から「(Ctrl)+C」キーによる割り込みを検出しました。

L3420 (F) Error occurred in "ロードモジュール"

"ロードモジュール"実行中にエラーが発生しました。

P3500 (F) Bad instantiation request file -- instantiation assigned to more than one file

インスタンス生成処理用の中間ファイルに誤りがあります。

リンク対象ファイルを再コンパイルしてください。

P3505 (F) corrupted template information file or instantiation request file

テンプレート処理用中間ファイル、またはインスタンス生成処理用の中間ファイルのデータが誤っています。これらのファイルの編集はしないでください。

L4000* (-) Internal error : ("内部エラー番号") "ファイル 行番号" / "コメント"

最適化リンケージエディタの処理中に内部的な問題が発生しました。

メッセージ内の内部エラー番号、ファイル、行番号、コメントを添えて、販売元のサポートセンターまでご連絡ください。

* P4000 と表示される場合があります。

14. 翻訳限界

14.1 コンパイラの翻訳限界

コンパイラの翻訳限界を**表** 14.1 に示します。

ソースプログラムを作成する際は、この翻訳限界の範囲で作成してください。

表 14.1 **コンパイラの翻訳限界**

	分類	項目	翻訳限界
1	起動	define オプションで指定可能なマクロ名総数	制限なし(メモリ容量に依存)
2		ファイル名の文字数	制限なし(OS に依存)
3	ソース プログラム	1 行の文字数	32768 文字
4		1 ファイルあたりのソースプログラムの行数	制限なし(メモリ容量に依存)
5		コンパイル可能なソースプログラムの総行数	制限なし(メモリ容量に依存)
6	プリプロ セッサ	#include 文のネストの深さ	制限なし(メモリ容量に依存)
7		#define 文のマクロ名総数	制限なし(メモリ容量に依存)
8		マクロ定義、マクロ呼び出しのパラメータの個数	制限なし(メモリ容量に依存)
9		マクロ名の再置き換えの数	制限なし(メモリ容量に依存)
10		条件コンパイルのネストのレベル数	制限なし(メモリ容量に依存)
11		#if, #elif 文で指定可能な演算子、非演算子の合計数	制限なし(メモリ容量に依存)
12	宣言	関数定義の個数	制限なし(メモリ容量に依存)
13		外部結合となる識別子(外部名)の数	制限なし(メモリ容量に依存)
14		1 関数内で有効な識別子(内部名)の数	制限なし(メモリ容量に依存)
15		基本型を修飾するポインタ、配列、および関数宣子の数	16 個
16		配列の次元数	6 次元
17		配列・構造体のサイズ	2147483647 バイト
18	文	複文のネストの深さ	制限なし(メモリ容量に依存)
19		繰り返し文(while 文、do 文、for 文)、選択文(if 文、switch 文)の組み合わせによるネストの深さ	4096 レベル
20		1 関数内で記述可能な複文の数	2048 個
21		1 関数内で指定可能な goto ラベルの数	2147483646 個
22		switch 文の数	2048 個
23		switch 文のネストの深さ	2048 レベル
24		1 つの switch 文内で指定可能な case ラベルの数	2147483646 個
25		for 文のネストの深さ	2048 レベル
26	式	文字列の文字数	32766 文字
27		関数定義、関数呼び出しでパラメータの個数	2147483646 個
28		1 つの式で指定可能な演算子と非演算子の合計数	約 500 個
29	標準 ライブラリ	open 関数で一度にオープンできるファイルの数	可変*[1]
30	セクション	セクション名長*[2]	8146 文字
31		1 ファイルあたりの#pragma section で指定できるセクション数	2045 個

【注】 *1 **詳細は「**8.3.2 **初期設定」を参照してください。**

*2 オブジェクト生成時に用いるアセンブラの 1 行文字数の制限を受けるため、#pragma section や section オプションで指定できる長さはこれより小さくなります。

14.2 アセンブラの翻訳限界

アセンブラの翻訳限界を表 14.2 に示します。

表 14.2 アセンブラの翻訳限界

	項目	翻訳限界
1	1 行文字数	8190 文字
2	シンボル長	1 行文字数*1
3	シンボル数	制限なし(メモリ容量に依存)
4	外部参照シンボル数	制限なし(メモリ容量に依存)
5	外部定義シンボル数	制限なし(メモリ容量に依存)
6	セクションの最大サイズ	0FFFFFFFFH バイト
7	セクション数	65265 個(デバッグ情報あり)、65274 個(デバッグ情報なし)
8	ファイルインクルード	ネストは 30 レベル
9	文字列長	1 行文字数*1
10	ファイル名の文字数	1 行文字数*1
11	環境変数設定文字数	2048 バイト
12	マクロ定義数	65535 個

【注】 *1 同じ行に指定した文字列の長さにより、これよりも小さい値となります。

15. ご利用上の注意事項

本章では、本コンパイラパッケージをご利用いただく上での注意事項を述べます。

15.1 コーディング上の注意事項

(1) 関数原型について

関数を呼び出す際には、呼び出される関数の関数原型を行ってください。関数原型を行わない場合、パラメータの受け渡しが正しく行えない場合があります。

例 1

float 型パラメータをもつ関数（dbl_size=8 を指定した場合）

```
void g()
{
            float a;
            ...
            f(a);                                   // a は double 型に変換されます。
}
void f(float x)
{...}
```

例 2

スタック渡しとなる signed char、(unsigned)char、(signed)short、および unsigned short 型のパラメータをもつ関数

```
void h();
void g()
{
            char a,b;
            ...
            h(1,2,3,4,a,b);                         // a,b は int 型に変換されます。
}
void h(int a1, int a2, int a3, int a4, char a5, char a6)
{...}
```

(2) 引数に型情報のない関数宣言

同じ関数に対して関数宣言(関数定義を含む)を複数行うとき、引数並びに型を記述しない形式と、型を記述する形式を両方使用しないでください。

この場合、呼び出す関数と呼び出される関数とで引数の解釈に違いが生じるため、生成コードが型を正しく処理できない場合があります。

コンパイル時に C5147 のエラーメッセージが表示された場合、この問題に該当している可能性がありますので、引数並びに型を記述する形式に変更するか、生成コードを確認して引数の受け渡しに問題がないかを確認してください。

例

old_style を異なる形式で記述しているために、引数 d と e の型の意味が呼び出す関数と呼び出される関数で異なるため、引数の受け渡しが正しく行われません。

```
extern int old_style(int,int,int,short,short); /* 関数宣言: 引数並びに型を記述する形式 */
int old_style(a,b,c,d,e)    /* 関数定義: 引数並びに型を記述しない形式 */
 int a,b,c;
 short d,e;
{
    return a + b + c + d + e;
}
int result;
func()
{
    result = old_style(1,2,3,4,5);
}
```

(3) C/C++言語で評価順序を規定していない式

C/C++言語規格で評価順序が規定されていない式を用いる際、評価順序で結果が変わるようなコーディングをした場合は動作保証しません。

例

```
a[i]=a[++i]  ;  代入式の右辺を先に評価するか後に評価するかで左辺の値が変わります。

sub(++i, i)  ;  関数の第 1 引数を先に評価するか後に評価するかで第 2 引数の値が変わります。
```

(4) オーバーフロー演算、ゼロ除算

オーバーフロー演算や浮動小数点のゼロ除算があっても、エラーメッセージを出力しません。ただし、一つの定数または定数どうしの演算でのオーバーフロー演算があれば、コンパイル時にエラーメッセージを出力します。

例

```
 void main()
 {
   int ia;
   int ib;
   float fa;
   float fb;

   ib=32767;
   fb=3.4e+38f;

  /* 定数または定数どうしの演算時はオーバーフローに対する                    */
  /* コンパイルエラーメッセージを出力します                                  */

  ia=99999999999;      /* (W) 定数のオーバーフローを検出します              */
  fa=3.5e+40f;         /* (E) 浮動小数点演算のオーバーフローを検出します    */

  /* 実行時のオーバーフローに対するエラーメッセージは出力しません            */

  ib=ib+32767;         /* 演算結果のオーバーフローを無視します              */
  fb=fb+3.4e+38f;      /* 浮動小数点演算結果のオーバーフローを無視します    */
}
```

(5) const型変数への書き込み

const 型の変数を宣言していても、型変換で const 型でない型に変換して代入した場合や、分割コンパイルしたプログラムの間で、型を統一して扱っていない場合は、const 型変数への書き込みをコンパイラでチェックできませんので、注意が必要です。

例

```
const char *p;          /* ライブラリ関数 strcat の第 1 引数は char 型への          */
:                       /* ポインタ型なので、引数の指す領域が書き換わる          */
strcat(p, "abc");       /* ことがあります                                        */
```

ファイル 1

```
const int i;
```

ファイル 2

```
extern int i;           /* 変数 i は、ファイル 2 では const 型で宣言していま      */
:                       /* せんのでファイル 2 の中で書き込んでもエラーに          */
i=10;                   /* なりません                                            */
```

(6) 数学関数ライブラリの精度について

acos(x)、asin(x)関数では x　1 で誤差が大きくなりますので注意が必要です。

誤差範囲は以下のとおりです。

acos(1.0 - ε)における絶対誤差	倍精度 2^{-39} ($\varepsilon = 2^{-33}$)
	単精度 2^{-21} ($\varepsilon = 2^{-19}$)
asin(1.0 - ε)における絶対誤差	倍精度 2^{-39} ($\varepsilon = 2^{-28}$)
	単精度 2^{-21} ($\varepsilon = 2^{-16}$)

(7) 最適化により削除される可能性のあるコーディング

連続した同一変数の参照や、結果を使用しない式を記述した場合、コンパイラの最適化により冗長コードとして削除される場合があります。常にアクセスを保証する場合は、宣言時に volatile を指定してください。

例：

```
[1] b=a;                /* 1 行目の式は冗長コードとして削除されることがあります  */
    b=a;
[2] while(1)a;          /* 変数 a の参照およびループ文は冗長コードとして削除される*/
                        /* ことがあります                                        */
```

(8) C89 とC99 の動作の差異

C99 では、選択文および反復文は、{}で囲まれます。そのため、C89 と C99 で動作が異なることがあります。

例：

```
enum {a,b};
int g(void)
{
     if(sizeof(enum{b,a}))
         return a;
     return b;
}
```

上記を-lang=c99 を指定してコンパイルすると、以下の解釈となります。

```
enum {a,b};
int g(void)
{
     {
         if(sizeof(enum{b,a}))
         return a;
     }
     return b;
}
```

-lang=c では、g()=0 となりますが、-lang=c99 では、g()=1 となります。

(9) オーバーフローを伴う演算および型変換に関する注意事項

数値演算や型変換を行う場合、その型で取り扱える値の範囲外(オーバーフロー)とならないようにご注意ください。オーバーフローが発生すると、得られる結果がコンパイルオプションなどの条件によって変化する場合があります。

標準の C 言語では、オーバーフローを伴う演算処理の結果は未定義となっており、コンパイル条件などにより得られる結果が異なる場合があります。

演算処理を行うプログラムでは、オーバーフローを発生させないようにご注意ください。

実際に結果が異なる例を次に示します。

例：float 型→unsigned short 型への変換

```
float f = 2147483648.0f;
unsigned short ui2;
void ex1func(void)
{
       ui2 = f; /* float → unsigned short への変換 */
}
```

ex1func 関数を実行して得られる ui2 の値は、FPU あり(-fpu)と FPU なし(-nofpu)とで次のように異なります。

FPU あり: ui2 = 65535

FPU なし: ui2 = 0

これは、float 型から unsigned short 型への変換の方法が FPU ありと FPU なしで異なるためです。

15.2 Cプログラムを C++コンパイラでコンパイルするときの注意事項

(1) 関数原型

関数を使用する前に関数原型が必要です。そのときには、仮引数の型も必ず宣言してください。

```
extern void func1();
void g()
{
    func1(1); // エラー
}
```

```
extern void func1(int);
void g()
{
    func1(1); // OK
}
```

(2) constオブジェクトのリンケージ

const オブジェクトのリンケージは、C プログラムでは外部結合であるのに対し、C++プログラムでは内部結合になります。また、const オブジェクトは初期値を必要とします。

```
const cvalue1;      // エラー

const cvalue2 = 1; // 内部結合
```

```
const cvalue1=0;
   // 初期値を与えます

extern const cvalue2 = 1;
   // Cプログラムと同様に外部結合に
   // なります
```

(3) void*からの代入

C++プログラムでは、明示的なキャストを用いないと他のオブジェクト型へのポインタ(関数へのポインタ、メンバへのポインタを除く)へ代入できません。

```
void func(void *ptrv, int *ptri)
{
    ptri = ptrv; //エラー
}
```

```
void func(void *ptrv, int *ptri)
{
    ptri = (int *)ptrv; //OK
}
```

15.3 オプションに関する注意事項

(1) 指定の統一が必要なオプションについて

指定の統一が必要なオプションを以下(a) (b)に示します。これらのオプション指定の異なるリロケータブルファイルおよびライブラリファイルをリンクした場合、実行時の動作は保証しません。

(a) cpu、endian、base、およびfint_registerの4つのオプションは、コンパイラ、アセンブラ、およびライブラリジェネレータで統一してください。

(b) 「2.5 マイコンオプション」に該当する(a)以外のオプションについては、コンパイラおよびライブラリジェネレータで統一してください。

15.4 旧バージョン・旧リビジョンとの互換性

バージョンもしくはリビジョン変更に伴う互換性に関する影響について説明します。

15.4.1 V.1.00 との互換性

(1) 組み込み関数の仕様変更について

アドレスを表す引数や戻り値を持つ組み込み関数については、その型を従来の unsigned long から void * に変更しました。変更になった関数を表 15.1 に示します。

表 15.1 型が変更された組み込み関数の一覧

	項目	仕様	機能	変更内容	
				箇所	内容
1	ユーザスタックポインタ(USP)	void set_usp(void *data)	USP の設定	引数	unsigned long → void *
2		void *get_usp(void)	USP の参照	戻り値	unsigned long → void *
3	割り込みスタックポインタ(ISP)	void set_isp(void *data)	ISP の設定	引数	unsigned long → void *
4		void *get_isp(void)	ISP の参照	戻り値	unsigned long → void *
5	割り込みテーブルレジスタ(INTB)	void set_intb (void *data)	INTB の設定	引数	unsigned long → void *
6		void *get_intb(void)	INTB の参照	戻り値	unsigned long → void *
7	バックアップ PC(BPC)	void set_bpc(void *data)	BPC の設定	引数	unsigned long → void *
8		void *get_bpc(void)	BPC の参照	戻り値	unsigned long → void *
9	高速割り込みベクタレジスタ (FINTV)	void set_fintv(void *data)	FINTV の設定	引数	unsigned long → void *
10		void *get_fintv(void)	FINTV の参照	戻り値	unsigned long → void *

この変更により、V.1.00 でこれらの関数を利用されていたプログラムでは、V.1.01 では型が合わない等の警告やエラーになる場合があります。この場合は、キャストを追加または削除して型を合わせてください。

例として、V.1.00 で標準的に使用されていたスタートアッププログラム例を示します。この例は V.1.01 では C5167(W)の警告メッセージが表示されますが、キャストをはずし型を合わせることで警告を回避できます。

例

set_intb 関数の利用例

```
#include <machine.h>
#pragma entry Reset_Program
void PowerON_Reset_PC(void)
{
        ...
        set_intb((unsigned long)__sectop("C$VECT")); // 警告 C5167(W)になる
        ...
```

```
}
```

V.1.01 用の記述に変更した例

```
#include <machine.h>
#pragma entry Reset_Program
void PowerON_Reset_PC(void)
{
        ...
        set_intb(__sectop("C$VECT")); // キャスト (unsignd long)を削除
        ...
}
```

(2) Lセクション追加について（sectionオプション、Startオプション）

V.1.01 では、文字列リテラルなどのリテラル領域を収録する L セクションを導入しました。

セクションが増えたことで、リンク時に L セクションが末尾に並ぶため、最適化リンケージエディタからアドレスエラーL3100(F)が発生する場合があります。

これを回避するためには、次のいずれかの方法を採ってください。

(a) リンク時の最適化リンケージエディタのStartオプションに指定するセクション列にLを追加する

例)

V.1.00 での指定例

-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D*,P,PIntPRG,W*/0FFFF8100,FIXEDVECT/0FFFFFFD0

変更例 (C の後に L を追加する)

-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000,C_1,C_2,C,L,C$*,D*,P,PIntPRG,W*/0FFFF8100,FIXEDVECT/0FFFFFFD0

(b) コンパイル時に-section=L=Cを選択する

コンパイル時に-section=L=C を指定することで、リテラル領域の出力先が C セクションに変更され、V.1.00 互換のセクション構成にすることができます。

なお、上記のリンク時の Start オプションを変更する方法に比べ、コード効率に影響が出る場合があります。

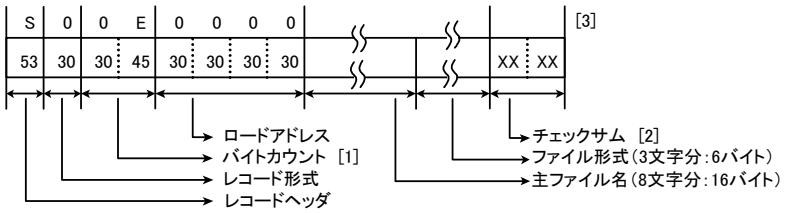
16. 付録

16.1 モトローラS形式、インテルHEX形式ファイル

本節では、最適化リンケージエディタによって出力されるモトローラ S 形式ファイルおよび、インテル HEX 形式ファイルについて説明します。

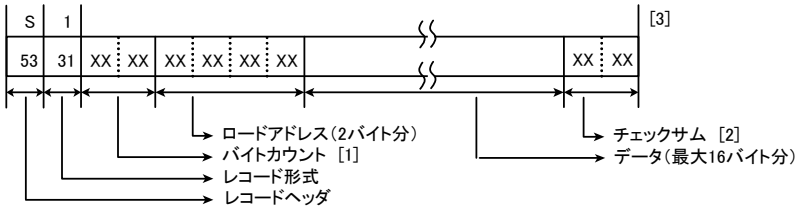
16.1.1 モトローラS形式ファイル

(a) ヘッダレコード(S0レコード)

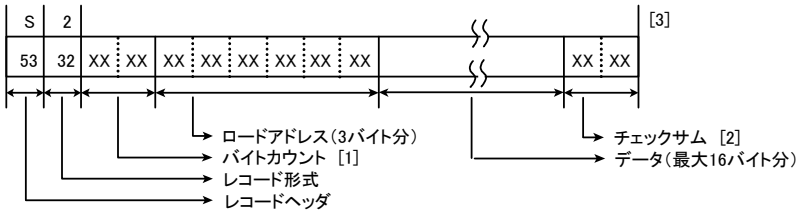


(b) データレコード(S1, S2, S3レコード)

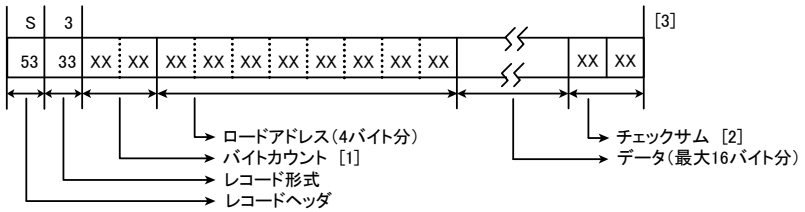
(i) ロードアドレスが0～FFFFの場合



(ii) ロードアドレスが10000～FFFFFFの場合

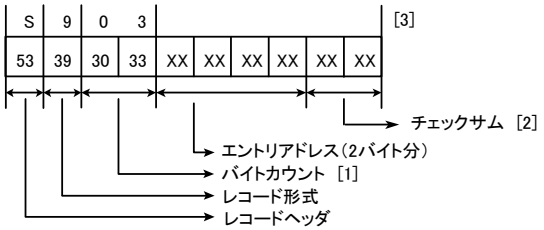


(iii) ロードアドレスが1000000～FFFFFFFFの場合

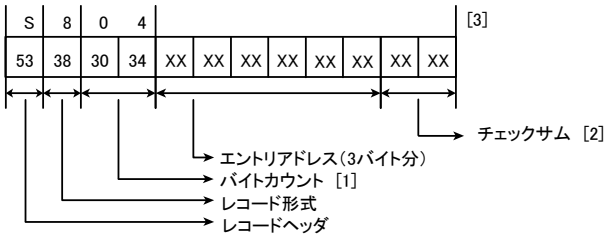


(c) エンドレコード(S9, S8, S7レコード)

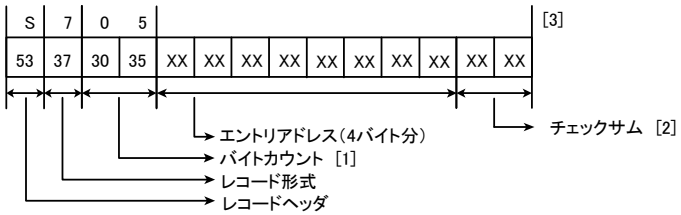
(i) エントリアドレスが0～FFFFの場合



(ii) エントリアドレスが10000～FFFFFFの場合



(iii) エントリアドレスが1000000～FFFFFFFFの場合



【注】 [1] ロードアドレス(またはエントリアドレス)からチェックサムまでのバイト数
[2] バイトカウンタからチェックサムの前までのデータ値をバイト単位に加算した結果の1の補数
[3] チェックサムの直後に改行コードが付加される

16.1.2 インテルHEX形式ファイル

各データレコードの実行アドレスは以下のように求めます。

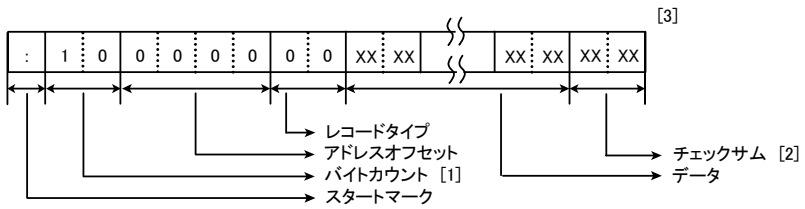
(1) セグメントアドレスの場合

(セグメントベースアドレス << 4) + (データレコードのアドレスオフセット)

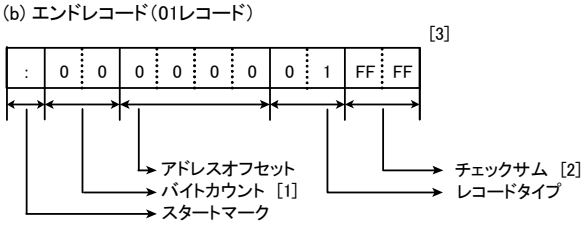
(2) リニアアドレスの場合

(リニアベースアドレス << 16) + (データレコードのアドレスオフセット)

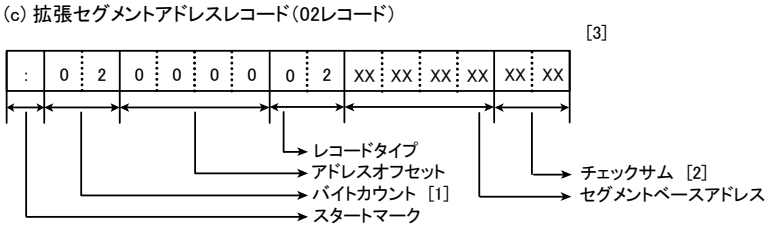
(a) データレコード(00レコード)



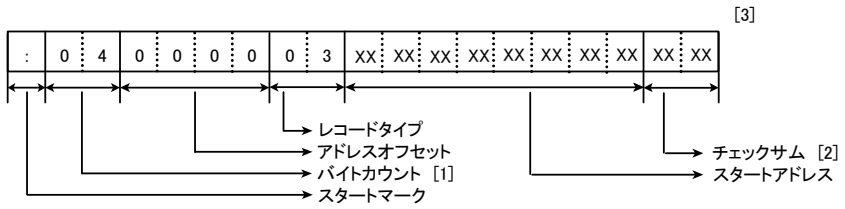
(b) エンドレコード(01レコード)



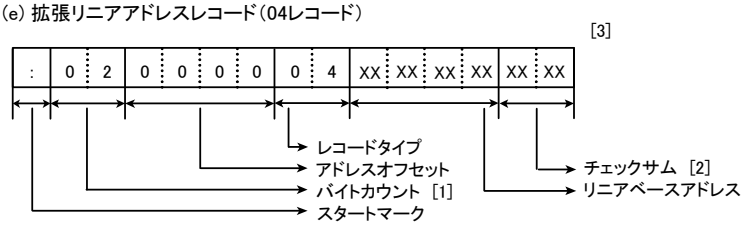
(c) 拡張セグメントアドレスレコード(02レコード)



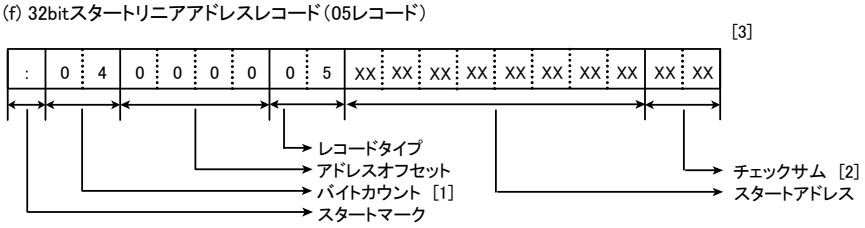
(d) スタートアドレスレコード(03レコード)



(e) 拡張リニアアドレスレコード(04レコード)



(f) 32bitスタートリニアアドレスレコード(05レコード)



【注】 [1] レコードタイプの次のデータから、チェックサムまでのバイト数
[2] バイトカウンタからチェックサムの前までのデータを、16進数で加算した結果の2の補数(下位8bitが有効)
[3] チェックサムの直後に改行コードが付加される

16.2 ASCIIコード一覧表

表 16.1 ASCII コード一覧表

下位 4 ビット	上位 4 ビット							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

RX ファミリ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ
コンパイラパッケージ
ユーザーズマニュアル

発行年月日　2011 年 2 月 23 日　Rev.1.00

発行　ルネサス エレクトロニクス株式会社
〒211-8668 神奈川県川崎市中原区下沼部 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

(03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

RX ファミリ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ コンパイラパッケージ V.1.01 ユーザーズマニュアル

ルネサス エレクトロニクス株式会社

R20UT0570JJ0100

RX ファミリ C/C++コンパイラパッケージ V.1.02 Release 01
注意事項、ユーザーズマニュアル修正および機能追加の項目

RX ファミリ C/C++コンパイラ V.1.02 Release 01 には、添付ユーザーズマニュアル(R20UT0570JJ0100)に、以下に示す注意事項、修正および機能追加の項目がございますので、当マニュアルご参照の際にはこれらの内容も合わせてご一読くださるようお願い申し上げます。

1. 注意事項

1.1 C1804 メッセージが出力される場合の注意事項

[C/C++コンパイラ]

C 標準ヘッダをインクルードしたファイルを C++または EC++コンパイルしたとき、int_to_short オプションを指定すると C1804 メッセージが出力されることがあります。これは、C++および EC++コンパイル時は、int_to_short オプションの指定は無効になるためです。

【注意】

C と C++(EC++)との間で共通にアクセスするデータは、int 型ではなく long 型または short 型で宣言してください。

1.2 MVTC,POPC 命令を使用する場合の注意事項

[アセンブラ]

アセンブリ言語において、MVTC,POPC 命令に対してプログラムカウンタ(PC)は指定できません。

1.3 delete オプションをリンク時に指定する場合の注意事項

[最適化リンケージエディタ]

delete オプションで指定した関数シンボルが削除された場合、削除された関数定義の次の関数定義の関数名に対して、デバッグ時にエディタ上でブレークポイントを設定することができません。ラベルウィンドウからブレークポイントを設定するか、関数のプログラム行で指定してください。

1.4 ファイル名に関する注意事項

[最適化リンケージエディタ]

最適化リンケージエディタでは、括弧記号"("および")"をオプション指定で使用するので、ファイル名に括弧記号を含まないようにしてください。

1.5 I/O ライブラリを使用する場合の注意事項

[統合開発環境(プロジェクト生成時)]

統合開発環境でプロジェクトを生成するときに、以下の画面で［I/O ライブラリ使用］を有効にする場合には、同時に［ヒープメモリ使用］も有効にしてください。

[I/O ライブラリ使用］を有効にした状態で［ヒープメモリ使用］を無効にすると、以下のリンクエラーとなります。

```
L2310 (E) Undefined external symbol "_sbrk" referenced in "xgetmem"
```

もし上記のリンクエラーが出るプロジェクトを生成した場合には、以下の C 言語ソースファイルをプロジェクトに追加登録して回避してください。

```
#include <stddef.h>
#include <stdio.h>

#define HEAPSIZE 0x400

signed char  *sbrk(size_t size);

union HEAP_TYPE {
    signed long  dummy ;
    signed char  heap[HEAPSIZE];
};

static union HEAP_TYPE heap_area ;

/* End address allocated by sbrk     */
static signed char *brk=(signed char *)&heap_area;

signed char  *sbrk(size_t size)
{
    signed char  *p;

    if(brk+size > heap_area.heap+HEAPSIZE){
        p = (signed char *)-1;
    }
    else {
        p = brk;
        brk += size;
    }
    return p;
}
```

1.6 ユーザスタックを作成しない方法

[統合開発環境(プロジェクト生成時)]

統合開発環境でプロジェクトを新規作成すると、デフォルトではユーザスタックを作成する設定になっています。ユーザスタックを使用されない場合は、以下の画面で［ユーザスタック使用］を無効にしてください。

1.7 スタックサイズとプロセッサモードのデフォルト設定変更

V.1.01 以降では、統合開発環境でプロジェクトを新規作成したときの次の項目のデフォルト設定を変更しています。なお、V.1.00 で作成したプロジェクトを V.1.01 以降で利用する場合はこれらの項目は変化しません。

(1) スタックサイズ

ユーザスタックと割り込みスタックのデフォルトのサイズを、それぞれ 256(0x100)バイトと 768(0x300)バイトに変更しました。

(2) プロセッサモード

スタートアッププログラムを、デフォルトでユーザモードに切り換えないように変更しました。これにより、main 関数はスーパーバイザモードのままで動作します。なお、この場合はユーザスタックを使用しないため、上記「1.6 ユーザスタックを使用しない方法」でユーザスタックを作成しないようにすることができます。

1.8 標準ライブラリのご利用に関するご注意【V.1.01 からリビジョンアップされたお客様】

V.1.01 Release 00 で標準ライブラリをご利用のお客様の場合、V.1.02 Release 00 以降にリビジョンアップ後は、環境変数 TMP_RX を V.1.01 Release 00 とは異なるディレクトリに変更してご利用いただくようお願いします。これは、V.1.01 Release 00 以降のライブラリジェネレータ lbgrx は、ライブラリ作成時の中間結果を TMP_RX が示すディレクトリに保存し、次回のライブラリ作成時に再利用するためです。
この対応を行わない場合、lbgrx が生成する標準ライブラリが、リビジョンアップした環境で生成したライブラリになりません。
なお、統合環境 High-performance Embedded Workshop をご利用の場合は、次の対応を行ってください。

【統合環境(High-performance Embedded Workshop)で標準ライブラリをご利用の場合の対応手順】

次の(1)～(3)の手順を、V.1.02 Release 00 にリビジョンアップ後に 1 回実施ください。

(1) コマンドプロンプトを開きます。(以降、コマンドプロンプト上で作業を行います。)

(2) コマンドラインで `dir %TEMP%¥*.pgl` を実行し、次のような、数字とアルファベットの並びを持ち、かつ拡張子が.pgl というファイルがひとつまたは複数表示されることを確認してください。

例) `dir %TEMP%¥.pgl` の表示結果

```
2011/08/09 15:47         825,346 800004008010000225a40409694ab0200000000.pgl
```

(3) (2)で表示されたファイルの個数だけ、次のように del コマンドを使ってファイルをひとつずつ削除してください。

例) ファイルを削除するコマンドの例 (1 ファイル分)

```
del %TEMP%¥800004008010000225a40409694ab0200000000.pgl
```

[ご参考]

手順(2)で、拡張子.pgl のファイルが手順(2)の例で示した形式以外に表示されていなければ、次のように一括して削除することもできます。

例) ファイルを一括で削除するコマンドの例

```
del %TEMP%¥*.pgl
```

1.9 PIC/PID 機能ご利用に関する制限事項【V.1.01 以降】

1.9.1 pic および pid オプション

ライブラリジェネレータ lbgrx に pic または pid オプションを指定して、標準ライブラリを作成することができますが、その際に、次の警告が 1 回または複数回表示されます。

```
C1301 (W) "-pic" option ignored   (pic オプションを指定した場合)
C1301 (W) "-pid" option ignored   (pid オプションを指定した場合)
```

これらの警告は、EC++ライブラリに対して、pic, pid オプションが無効になるため出力されます。

1.9.2 nouse_pid_register オプション

PID 機能を利用するとき、マニュアルに従って、マスタに含まれる全てのファイルに nouse_pid_register を指定すると、下記をアセンブルする際にエラーが発生します。

(1) PID レジスタにアドレスを代入するプログラム

(2) 標準ライブラリ(ライブラリジェネレータ lbgrx に指定)

それぞれ次の方法で対応をお願いします。

【(1)の対応】

PID レジスタにアドレスを代入するその機能だけをひとつの C ソースもしくはアセンブラファイルにして、-nouse_pid_register を指定せずにコンパイルまたはアセンブルしてください。そして、他のマスタのファイル(-nouse_pid_register 付き)とリンクして利用してください。

【(2)の対応】

次の(a)(b)いずれかの方法で対応ください。

(a) 標準ライブラリをマスタではなくアプリケーションとリンクする

- ジャンプテーブル(8.4.2(2)参照)を利用する必要はありません。
- ライブラリジェネレータ lbgrx に pid オプションを指定して標準ライブラリを作成してください。

(b)標準ライブラリをマスタとリンクする場合

(i) ライブラリジェネレータ lbgrx に nouse_pid_register オプションをつけずに標準ライブラリを作成してください。

(ii) 作成したジャンプテーブルを、次の例に従って、全てのエントリの JMP R14 を変更してから使用してください。

例) `_printf` エントリの変更

[変更前]

```
_printf:
   MOV.L     #0ffff90cfH,R14   ; アドレス 0ffff90cfH は例です。
   JMP       R14
```

[変更後]

```
_printf:
   MOV.L     #0ffff90cfH,R14
   PUSH.L    R13       ; PID レジスタが R13 の場合
   JSR       R14
   POP       R13       ; PID レジスタが R13 の場合
   RTS
```

1.10 C++言語(EC++含む)で math.h の一部関数(frexp,ldexp,scalbn,remquo)を使用する場合の注意事項

C++/EC++コンパイル時に、math.h の一部の関数(frexp,ldexp,scalbn,remquo)の実引数を int 型にすると、実行時に無限ループとなるオブジェクトが生成されます。

発生条件：

次の条件(1)(2)を全て満たす場合が該当します。

(1) C++ソース(拡張子が.cpp)または、-lang=cpp オプションが有効である。

(2) math.h をインクルードして、以下の関数をそれぞれの条件で呼び出している。

(a) frexp(double, long *) の第２引数の値を (int *)型とする

ただし、第１引数が float 型で、-dbl_size=8 オプション指定時を除く

(b) ldexp(double, long) の第２引数の値を int 型とする

ただし、第１引数が float 型で、-dbl_size=8 オプション指定時を除く

(c) scalbn(double, long) の第２引数の値を int 型とする

ただし、第１引数が float 型で、-dbl_size=8 オプション指定時を除く

(d) remquo(double, double, long *) の第３引数の値を (int *)型とする

ただし、第１または第２引数が float 型で、-dbl_size=8 オプション指定時を除く

発生例：

[file.cpp]

```
// C++ソースとしてコンパイルした場合に無限ループになる例
#include <math.h>
double d1,d2;
int  i;
void func(void)
{
   d2 = frexp(d1, &i);
}
```

[コマンドライン例]

```
Command Line:
ccrx -cpu=rx600 -output=src file.cpp
```

[file.src] ソース出力例

```
_func:
   ; ...(中略)
   BSR  __$frexp__tm__2_f__FZ1ZPi_Q2_21_Real_type__tm__4_Z1Z5_Type ; frexp
```

```
の代替関数を呼ぶ
   ; ...(中略)

__$frexp__tm__2_f__FZ1ZPi_Q2_21_Real_type__tm__4_Z1Z5_Type:
L11:
   BRA   L11   ; 再帰呼び出しとなってしまう
```

回避策：

次のいずれかの方法で回避できます。

(1) -lang=c または -lang=c99 を指定し、C 言語としてコンパイルする。

(2) 引数の int および int*を long および long*に変更する。

(3) math.h の後に、使用する関数ごとに定義を追加する。

```
/* frexp 関数の場合 */
static double frexp(double x, int *y)
{ long v = *y; double d = frexp(x,&v); *y = v; return (d); }
/* ldexp 関数の場合 */
static double ldexp(double x, int y)
{ long v = y; double d = ldexp(x,v); return (d); }
/* scalbn 関数の場合 */
static double scalbn(double x, int y)
{ long v = y; double d = scalbn(x,v); return (d); }
/* remquo 関数の場合 */
static double remquo(double x, double y, int *z)
{ long v = *z; double d = remquo(x,y,&v); *z = v; return (d); }
```

回避策(2)の例

```
[file.cpp] の変更例
#include <math.h>
double d1,d2;
int  i;
void func(void)
{
    long x = i;         /* 一旦 long 型変数で受ける */
    d2 = frexp(d1, &x);  /* long 型変数で呼び出し */
    i = x;             /* i に値を設定 */
}
```

回避策(3)の例

```
[file.cpp] の変更例
#include <math.h>
/* 宣言を追加 */
static inline double frexp(double x, int *y)
{ long v = *y; double d = frexp(x,&v); *y = v; return (d); }
double d1,d2;
int  i;
void func(void)
{
    d2 = frexp(d1, &i);
}
```

1.11　旧バージョン（V.1.00）から移行したプロジェクトのビルドについて -- （L セクションの警告 L1120 およびエラーL3100）

V.1.01 Release 00 よりも古いコンパイラパッケージで作成した High-performance Embedded Workshop のプロジェクトを、V.1.01 Release 00 またはそれ以降のプロジェクトに変換して使用すると、次の警告およびエラーを発生することがあります。

```
L1120 (W) Section address is not assigned to "L"
L3100 (F) Section address overflow out of range : "L"
```

この場合は、本書「2.マニュアル修正項目」の「■(p.823) 15.4.1 V.1.00 との互換性 / (2) L セクション追加について(section オプション、Start オプション)」の内容に基づき、(a)または(b)の方法で対策してください。

【備考】

・本パッケージに含まれる High-performance Embedded Workshop で、V.1.01 Release 00 よりも古いプロジェクト（RX Toolchain 1.0.0.0 ～ 1.0.0.2）を V.1.01 Release 00 またはそれ以降（RX Toolchain 1.1.0.0 ～）に変換した場合は、自動的に(b)を行いますので、この現象は発生しません。

2. マニュアル修正項目

■(p.10) 2.1 ソースオプション / preinclude / 説明

＜修正内容＞

[変更前]　include オプション指定フォルダが複数…

[変更後]　preinclude オプション指定フォルダが複数…

■(p.12) 2.1 ソースオプション / change_message / 備考

＜追加内容＞

[変更後]

misra2004 オプション指定時に表示する、MISRA 検出メッセージ(記号(M)を表示)は本オプション制御対象外です。"

■(p.20) 2.2 オブジェクトオプション stuff,nostuff / 説明

＜修正内容＞

[変更前]

各セクション内のデータは常に定義順に出力されます。

[変更後]

C セクション内で初期値がない変数が初期値のある変数の後に出力されることを除き、各セクション内のデータは定義順に出力されます。

■(p.28) 2.4 最適化オプション / 表 2.7 / No.20 float_order / ダイアログメニュー

＜修正内容＞

[変更前]

コンパイラ <最適化> / [詳細…] / [その他] / [浮動小数点式の演算順序変更]

[変更後]

コンパイラ <その他> / [その他のオプション] / [浮動小数点演算式の演算順序変更を積極的に行なう]

■(p.47) 2.5 マイコンオプション / 表 2.9 / No.19 pid / ダイアログメニュー

＜修正内容＞

[変更前]　[コードセクションを位置独立コードとして生成]

[変更後]　[データセクションを位置独立コードとして生成]"

■(p.56) 2.5 マイコンオプション base / 説明

＜修正内容＞

[変更前]

<アドレス値>=<レジスタ C>を指定した場合は、アドレス値から 64KB～256KB 以内の領域のアクセスは、指定したレジスタ C 相対で行います。

[変更後]

<アドレス値>=<レジスタ C>を指定した場合は、コンパイル時に割り付けアドレスが確定している領域のうち、アドレス値から 64KB～256KB 以内の領域のアクセスを、指定したレジスタ C 相対で行います。

■(p.135) 表 5.13 その他カテゴリオプション一覧 / No.12 / Total_size / ダイアログメニュー

＜修正内容＞

[変更前]

リンカ <その他>

[ユーザ指定オプション :]

[変更後]

リンカ <その他>

[その他オプション :]

[合計セクションサイズの画面表示 :]

■(p.223) 表 8.8 マスタ内の PIC/PID 機能オプションの指定規則 / No.3 nouse_pid_register オプション / コンパイル時

＜修正内容＞

[変更前] ○ 指定可

[変更後] △ 標準ライブラリ、スタートアップ内の PID レジスタ設定箇所以外は指定可

■(p.223) 表 8.8 マスタ内の PIC/PID 機能オプションの指定規則 / No.3 nouse_pid_register オプション / リンク可能オブジェクトのオプション指定の条件

＜修正内容＞

[変更前] nouse_pid_register の指定が必須

[変更後] 制限なし

■(p.224) 表 8.10 マスタとアプリケーション間の PIC/PID 機能オプションの組み合わせ規則 / マスタのオプション

＜修正内容＞

[変更前] nouse_pid_register が必須

[変更後] アプリケーションからマスタ上の関数を呼び出す場合は、nouse_pid_register が必須

■(p.258) 表 9.20 #pragma、キーワード一覧

＜追加内容＞

No.15

対象 / C99 標準

#pragma 拡張子 /

`#pragma STDC CX_LIMITED_RANGE` フラグ

`#pragma STDC FENV_ACCESS` フラグ

`#pragma STDC FP_CONTRACT` フラグ

機能 / システムの状態変更(*3)

[脚注] / *3) C99 言語を有効にしてコンパイルする場合、これらに対しては C99 言語の文法確認のみ行い、内容は無視します。

■(p.262) 9.2.1 #pragma、キーワード / #pragma stacksize / 備考

＜追加内容＞

<定数>に記述できる値の範囲は、4 から 2147483644(0x7ffffffc)までです。

■(p.275) 9.2.1 #pragma、キーワード / #pragma pack, unpack, packoption / 備考

＜削除内容＞

#pragma pack を指定した構造体メンバおよびクラスのメンバはポインタを用いてアクセスすることはできません(ポインタを使用したメンバ関数内でのアクセスを含みます)。

例

```
#pragma pack
struct st {
   char x;
   int y;
} ST;
int *p=&ST.y;   /* ST.y のアドレスが奇数になる場合があります */
void func(void) {
   ST.y=1;    /* 正しくアクセスできます */
   *p=1;    /* 正しくアクセスできない場合があります */
}
```

■(p.349) 9.3.1 標準 C ライブラリ / (8)math.h / ldexp 関数 / 例
＜修正内容＞
[変更前] int f;
[変更後] long f

■(p.360) 9.3.1 標準 C ライブラリ / (8)math.h / scalbn 関数 / 例
＜修正内容＞
[変更前] int e;
[変更後] long e

■(p.440) 9.3.1 標準 C ライブラリ / .(13) stdlib.h / mbstowcs 関数
＜追加内容＞
定義名 / mbstowcs
内容 / 多バイト文字列をワイド文字列に変換（詳細は Page 534 を参照）

■(p.440) 9.3.1 標準 C ライブラリ / .(13) stdlib.h / wcstombs 関数
＜追加内容＞
定義名 / wcstombs
内容 / ワイド文字列を多バイト文字列に変換（詳細は Page 535 を参照）

■(p.534) 9.3.1 標準 C ライブラリ / .mbstowcs 関数 / 関数説明文
＜修正内容＞
[変更前] 多バイト文字文字列をワイド文字列に変換
[変更後] 多バイト文字列をワイド文字列に変換

■(p.642) 10.3.1 アドレス制御命令 / .BLKB / 備考
＜追加内容＞
オペランドに指定できる値の最大値は 7FFFFFFFH です。

■(p.643) 10.3.1 アドレス制御命令 / .BLKW / 備考
＜追加内容＞
オペランドに指定できる値の最大値は 3FFFFFFFH です。

■(p.644) 10.3.1 アドレス制御命令 / .BLKL / 備考
＜追加内容＞

オペランドに指定できる値の最大値は 1FFFFFFFH です。

■(p.645) 10.3.1 アドレス制御命令 / .BLKD / 備考

＜追加内容＞

オペランドに指定できる値の最大値は 0FFFFFFFH です。

■(p.690) 11.2 メッセージ一覧 / C5014

＜修正内容＞

[変更前]

C5014 (E) Extra text after expected end of preprocessing directive

プリプロセッサ文の後にさらにテキストが記述されています。

[変更後]

C5014 (E)(W) Extra text after expected end of preprocessing directive

プリプロセッサ文の後にさらにテキストが記述されています。

■(p.694) 11.2 メッセージ一覧 / C5062

＜修正内容＞

[変更前]

C5062 (W) Shift count is negative

シフトカウントが負の値です。指定された通りに演算します。

[変更後]

C5062 (W) Shift count is negative

シフトカウントが負の値です。この演算の結果を確認してください。

■(p.694) 11.2 メッセージ一覧 / C5063

＜修正内容＞

[変更前]

C5063 (W) Shift count is too large

シフトカウントが有効ビット数を超えています。指定された通りに演算します。

[変更後]

C5063 (W) Shift count is too large

シフトカウントが型のビット数以上の値です。演算結果を確認してください。

■(p.767) 11.2 メッセージ一覧 / C6373, C6374, C6375

＜削除内容＞

C6373 (W) Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)

64 ビット整数型がより小さい整数型へと暗黙的に変換されています。移植性の問題になる可能性があります。

C6374 (W) Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)

64 ビット整数型がより小さい整数型へと明示的に変換されています。移植性の問題になる可能性があります。

C6375 (W) Conversion from pointer to same-sized integral type (potential portability problem)

ポインタから同サイズの整数型へと変換しています。移植性の問題になる可能性があります。

■(p.814) 14.2 アセンブラの翻訳限界 / 表 14.2 / No.1 / 1 行文字数

＜修正内容＞

[変更前]　8190 文字

[変更後]　32760 文字

■(p.823) 15.4.1 V.1.00 との互換性 / (2) L セクション追加について(section オプション、Start オプション)

＜修正内容＞ ※変更後の内容のみ示します。

[変更後]

(2) L セクション追加について(section オプション、Start オプション)

V.1.01 では、C セクションに対する map や base オプション適用時のコード効率向上を図るため、文字列リテラルなどのリテラル領域の出力先として L セクションを新規に追加しました。

L セクションの割り付けアドレスを指定しない V.1.00 からプロジェクト変換したプロジェクトは、リンク時に L セクションに対して、最適化リンケージエディタがアドレスエラー L3100(F)を出力する場合があります。

```
L3100 (F) Section address overflow out of range : "L"
```

これを回避するためには、次のいずれかの方法を実施してください。

なお、コード効率の面から、通常は(a)の方法を推奨します。(b)の方法は、セクション構成を変えたくない場合に使用ください。

(a) リンク時の最適化リンケージエディタの Start オプションに指定するセクション列に L を追加する

[コマンドラインでの指定方法]

例)

V.1.00 での指定例

-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D*,P,PIntPRG,W*/0FFFF8100,FIXEDVECT/0FFFFFFD0

変更例(C の後に L を追加する)

-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000,C_1,C_2,C,L,C$*,D*,P,PIntPRG,W*/0FFFF8100,FIXEDVECT/0FFFFFFD0

High-performance Embedded Workshop での設定手順は以下のとおりです。

1. メニュー「ビルド」を開き「RX Standard Toolchain」を選択する
2. 「最適化リンカ」タブをクリックし、カテゴリメニューから「セクション」を選択し、設定項目メニューから「セクション」を選択する
3. L セクションを配置する Address 領域をリスト表示から選択して「編集」または「追加」ボタンをクリックして、L セクションを配置するアドレスを設定する

(b) コンパイル時およびライブラリビルド時に-section=L=C を選択する

コンパイル時、およびライブラリビルド時に、ccrx および lbgrx コマンドのそれぞれに -section=L=C を指定することで、リテラル領域の出力先が C セクションに変更され、V.1.00 互換のセクション構成にすることができます。

High-performance Embedded Workshop での設定手順は以下のとおりです。

1. メニュー「ビルド」を開き「RX Standard Toolchain」を選択する
2. 「コンパイラ」タブをクリックする
3. カテゴリメニューから「オブジェクト」を選択する
4. 「詳細(D」ボタンをクリックして「オブジェクト詳細」ダイアログボックスを開く
5. プルダウンメニューから「リテラル領域(L)」を選びメニュー直下のテキストボックスに「C」と入力し、OK ボタンをクリックしてウィンドウを閉じる
6. 「標準ライブラリ」タブをクリックして、上記 3～5 の手順を繰り返す

3. 機能追加項目

V.1.02 Release 00 では次の項目の機能を追加しました。

3.1 ソースオプション【マニュアル 2.1 章】

コンパイラのソースオプションに追加したオプションを示します。

No	オプション	ダイアログメニュー	内容	
11	-misra2004={ all | apply=<ルール指定> | ignore=<ルール指定> | required | required_add=<ルール指定> | required_remove=<ルール指定> | <ファイル名> } <ルール指定>: <ルール番号>[,<ルール番号>,...]	コンパイラ <MISRA C ルール検査> [MISRA-C の検査を有効にする] [解析オプション:] [ルール番号:]	MISRA-C:2004 ルールによるソースチェックを行う	新規
12	-ignore_files_misra= <ファイル名>[,<ファイル名>,...]	コンパイラ <MISRA C ルール検査> [MISRA-C の検査を有効にする] [拡張設定:] [検査対象外ファイル]	MISRA-C:2004 チェック対象外のファイルを選択する	新規
13	-check_language_extension	コンパイラ <MISRA C ルール検査> [MISRA-C の検査を有効にする] [拡張設定:] [拡張機能を検査]	拡張機能の使用によって部分抑止された MISRA-C:2004 ルールを有効にする	新規

misra2004

＜書式＞

-misra2004 = {
all
| apply=<ルール番号>[,<ルール番号>,...]
| ignore=<ルール番号>[,<ルール番号>,...]
| required
| required_add=<ルール番号>[,<ルール番号>,...]
| required_remove=<ルール番号>[,<ルール番号>,...]
| <ファイル名> }

＜説明＞

MISRA-C:2004 のルールチェック機能を有効にし、そのチェック対象を指定します。

-misra2004=all オプション指定時は、本コンパイラでサポートしている全てのルールをチェック対象とします。

-misra2004=apply<ルール番号>[,<ルール番号>,...]オプション指定時は、サポートしているルールのうち、指定されたルール番号をチェック対象とします。

-misra2004=ignore=<ルール番号>[,<ルール番号>,...]オプション指定時は、サポートしているルールのうち、指定されたルール番号以外のルールをチェック対象とします。

-misra2004=required オプション指定時は、サポートしているルールのうち、ルールの分類が"required"になっているルールをチェック対象とします。”required”のルールについては、MISRA-C:2004 を参照してください。

-misra2004=required_add=<ルール番号>[,<ルール番号>,...]オプション指定時は、サポートしているルールのうち、ルールの分類が"required"になっているルールと指定されたルール番号をチェック対象とします。

-misra2004=required_remove=<ルール番号>[,<ルール番号>,...]オプション指定時は、サポートしているルールのうち、ルールの分類が”required”になっているルールから指定されたルール番号を除いたルール番号をチェック対象とします。

-misra2004 =<ファイル名>オプション指定時は、サポートしているルールのうち、指定されたファイル名に記載されたルール番号をチェック対象とします。ファイル名で指定されたファイル内の記述は、1 ルールを 1 行で記述します。

ルール番号は、10 進数値およびピリオド(".")で指定してください。

MISRA-C:2004 のチェック項目に該当した場合、次の形式でメッセージを表示します。

ファイル名(*行番号*) : C6700 (M) Rule *ルール番号*: *メッセージ*

＜備考＞

コマンドライン上で同一オプションを複数回指定した場合には、最後に記述したオプションが有効になります。

サポートしていないルール番号を、各オプションの<ルール番号>に指定した場合はエラーC6703(F)となり、そこで処理を中止します。

-misra2004=<ファイル名>で指定したルールファイルがオープンできない場合、エラーC6701(F)となります。また、ルールファイルからルール番号を読み込めない場合は、エラーC6702(F)となり、いずれの場合もそこで処理を中止します。

-lang オプションに cpp,c99 または ecpp を選択した場合、本オプションを無視します。
-output=prep と同時に指定した場合、本オプションを無視します。

本オプションでサポートしている MISRA-C:2004 のルール番号を次に示します。

```
   2.2   2.3
   4.1   4.2
   5.2   5.3   5.4   5.5   5.6
   6.1   6.2   6.3   6.4   6.5
   7.1
   8.1   8.2   8.3   8.5   8.6   8.7   8.11  8.12
   9.1   9.2   9.3
  10.1  10.2  10.3  10.4  10.5  10.6
  11.1  11.2  11.3  11.4  11.5
  12.1  12.3  12.4  12.5  12.6  12.7  12.8  12.9  12.10 12.11
  12.12 12.13
  13.1  13.2  13.3  13.4
  14.2  14.3  14.4  14.5  14.6  14.7  14.8  14.9  14.10
  15.1  15.2  15.3  15.4  15.5
  16.1  16.3  16.5  16.6  16.9
  17.5
  18.1  18.4
  19.3  19.6  19.7  19.8  19.11 19.13 19.14 19.15
  20.4  20.5  20.6  20.7  20.8  20.9  20.10 20.11 20.12
```

#pragma などの拡張機能を用いたソースでは、ルールチェックの一部が抑止されます。
抑止される内容は、check_language_extension オプションの項目を参照してください。

misra2004 オプションで表示される MISRA 診断メッセージは、change_message オプションで制御することはできません。

ignore_files_misra

＜書式＞

-ignore_files_misra=<ファイル名>[,<ファイル名>,...]

＜説明＞

MISRA-C:2004 のルールチェック対象外のソースファイルを指定します。

＜備考＞

コマンドライン上で同一オプションを複数回指定した場合には、それぞれのオプションが有効になります。

-misra2004 オプションの指定がない場合は、本オプションを無視します。

指定されたソースファイル名が、コンパイル対象でない場合は、指定されたソースファイル名を無視します。

check_language_extension

＜書式＞

-check_language_extension

＜説明＞

C 言語規格から本コンパイラが拡張した言語仕様対しても MISRA2004 ルールチェック対象にします。

本コンパイラでは、misra2004 オプションのデフォルトでは、次の場合はチェックが抑止されます。これをチェックしたい場合は、misra2004 オプション指定時に、check_language_extension オプションを指定してください。

●プロトタイプ宣言がない場合(ルール 8.1)で、該当関数に #pragma entry または #pragma interrupt のいずれかの指定があるとき。

[例]

```
#pragma interrupt vfunc
extern void service(void);
void vfunc(void)
{
    service();
}
```

関数 vfunc にはプロトタイプ宣言がありませんが、#pragma interrupt の対象なので -misra2004=all を指定してコンパイルしても、-check_language_extension の指定がないと、ルール 8.1 のチェックメッセージが表示されません。

＜備考＞
-misra2004 オプションの指定がない場合は、本オプションを無視します。

3.2 オブジェクトオプション【マニュアル 2.2 章】

コンパイラのオブジェクトオプションに追加したオプションを示します。

No	オプション	ダイアログメニュー	内容	
7	-nouse_div_inst	コンパイラ <オブジェクト> [DIVS,DIVU,FDIV 命令生成抑止]	除算、剰余算に DIV,DIVU,FDIV 命令を使用しない	新規

nouse_div_inst
＜書式＞
-nouse_div_inst

＜説明＞
プログラム中の全ての除算および剰余算を、DIV 命令、DIVU 命令および FDIV 命令を使わないコードを生成します。

＜備考＞
本オプション指定時は、DIV,DIVU および FDIV 命令を生成する代わりに、それぞれの命令に相当する処理を行うランタイム関数の呼び出しに置き換えます。
このため、ROM サイズやコード実行速度といったコード効率が悪くなる場合があります。
本オプションは、ライブラリジェネレータ(lbgrx)でも指定することができます。

3.3 コンパイラのエラーレベル【マニュアル 11.1 章】

コンパイラのエラーレベルに追加した種類を示します。

	エラーレベル	動作	
(M)	MISRA2004 検出	処理を継続します。	新規

3.4 コンパイラメッセージ追加【マニュアル 11.2 章】

コンパイラのメッセージ一覧に追加したメッセージを示します。

C6700 (M) Rule <ルール番号>: <内容>
MISRA2004 の<ルール番号>と<内容>の該当箇所を検出しました。

C6701 (F) Cannot open rule file <ファイル名>
MISRA2004 ルールファイル<ファイル名>をオープンできませんでした。

C6702 (F) Incorrect description "<内容>" in rule file
MISRA2004 ルールファイルの記述 <内容> が正しくありません。

C6703 (F) Rule <ルール番号> is unsupported

指定された MISRA2004 の<ルール番号>はサポートされていません。

3.5 コンパイラの翻訳限界【マニュアル 14.1 章】

コンパイラの翻訳限界に追加した項目を示します。

No	分類	項目	翻訳限界	
32	出力ファイル	アセンブリソースとして出力できる 1 行の最大文字数	8190 文字	新規

以上

RX ファミリ C/C++コンパイラパッケージ V.1.00 Release 01
添付標準ライブラリについて

本製品では、RX600 用に標準ライブラリファイル(*.lib)を 4 種類添付しています。
添付の標準ライブラリファイルを使用することにより、ビルドに要する時間を短縮することができます。

1. 添付ライブラリー覧

本製品に添付される、標準ライブラリファイルの一覧を表 1 に示します。

【ご注意】

ライブラリで選択している「マイコンオプション」は、ご使用のコンパイラオプションと一致させる必要があります。いずれとも一致しない場合は、これらの標準ライブラリは使用できませんので、ご利用のコンパイラオプション(アセンブラオプション)をライブラリジェネレータに指定して、作成されるライブラリをご使用ください。

ライブラリ名	用途	最適化*2 オプション	マイコンオプション *1 *2		
			-endian	-cpu -rtti -exception -noexception	その他 *3
rx600lq.lib	RX600、速度優先 リトルエンディアン	-speed -goptimize	-endian=little	-cpu=rx600 -rtti=on -exception	-round=nearest -denormalize=off -dbl_size=4 -unsigned_char -unsigned_bitfield -bit_order=right -unpack -fint_register=0 -branch=24
rx600ls.lib	RX600、サイズ優先 リトルエンディアン	-size -goptimize			
rx600bq.lib	RX600、速度優先 ビッグエンディアン	-speed -goptimize	-endian=big		
rx600bs.lib	RX600、サイズ優先 ビッグエンディアン	-size -goptimize			

表 1 ライブラリー覧

*1 マイコンオプションは、コンパイラマニュアルの「2.5 マイコンオプション」を参照ください。

*2 High-performance Embedded Workshop のビルド設定により、これらのオプションがどのように設定されているかを確認するには、コンパイラマニュアルの「表 2.7 最適化オプション一覧」「表 2.9 マイコンオプション一覧」の「ダイアログメニュー」欄を参照ください。

*3 これらのオプション選択は省略時設定と同じです。

2. ライブラリ指定の方法

添付の標準ライブラリファイルを使用する場合は、2.2 または 2.3 の方法でリンクしてください。

2.1 ライブラリの格納場所

統合開発環境のインストール先が C:¥Program Files¥Renesas¥Hew の場合:

`C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥RX¥1_0_1¥lib`

(1_0_1 は、コンパイラパッケージのリビジョンにより異なります。)

2.2 High-performance Embedded Workshop から選択する場合

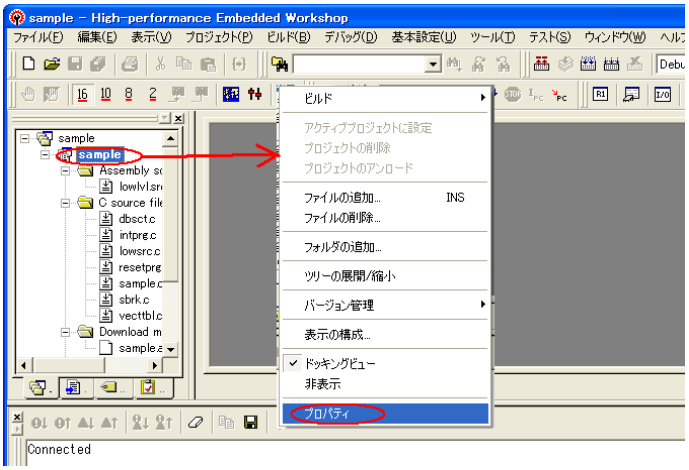
ご使用のプロジェクトに対して、ライブラリを選択する手順を説明します。

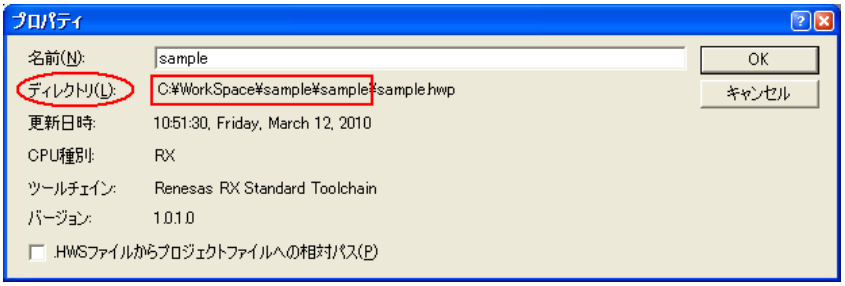
(1) プロジェクトを開きます。

(2) ご使用のプロジェクトの設定を確認し、表 1 からライブラリを選択してください。

(3) プロジェクトディレクトリを確認します。

ワークスペースウィンドウのプロジェクトを選択して、右クリックメニューの「プロパティ」を選択して表示される、「ディレクトリ(L)」で確認してください。





この表示で、拡張子が.hwp であるファイルの格納場所がプロジェクトディレクトリです。

(4) (2)で選択したライブラリファイルを、上記 2.1 の格納場所から、プロジェクトディレク

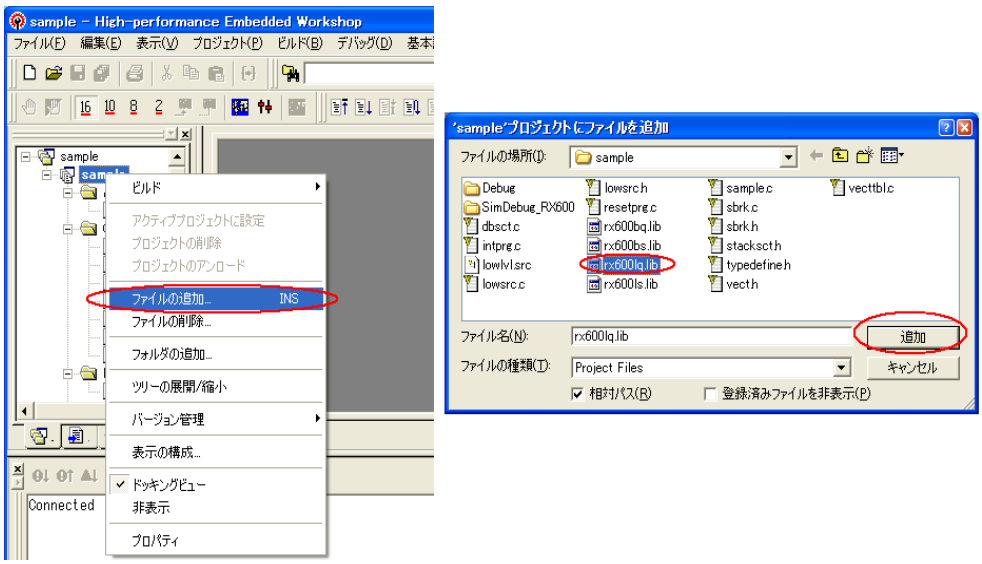
トリにコピーしてください。

[コマンドプロンプトによる手順(4)の実行例]

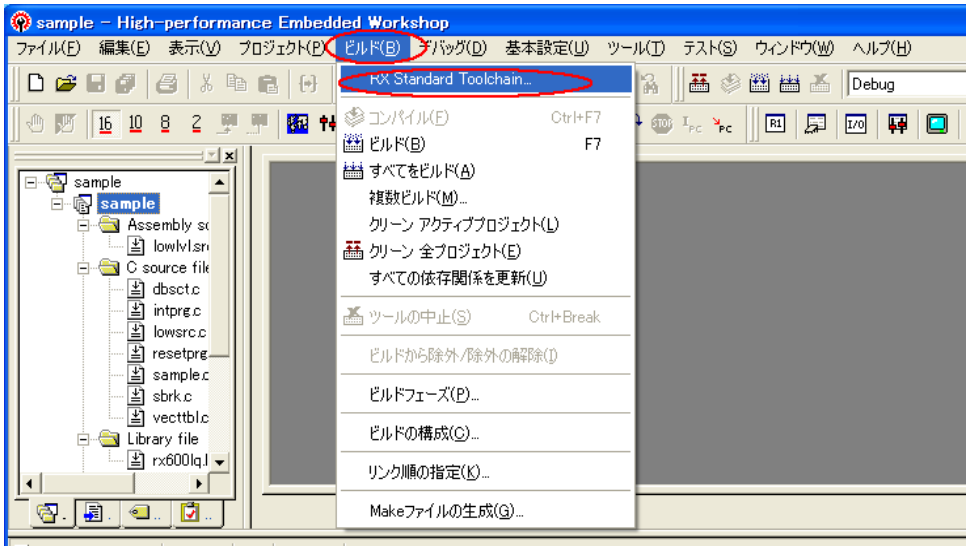
```
copy "C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥RX¥1_0_1¥lib¥rx600lq.lib" C:¥WorkSpace¥sample¥sample
```

(5) ワークスペースウィンドウのプロジェクトを選択して、右クリックメニューの「ファイルの追加 INS」を選択してください。

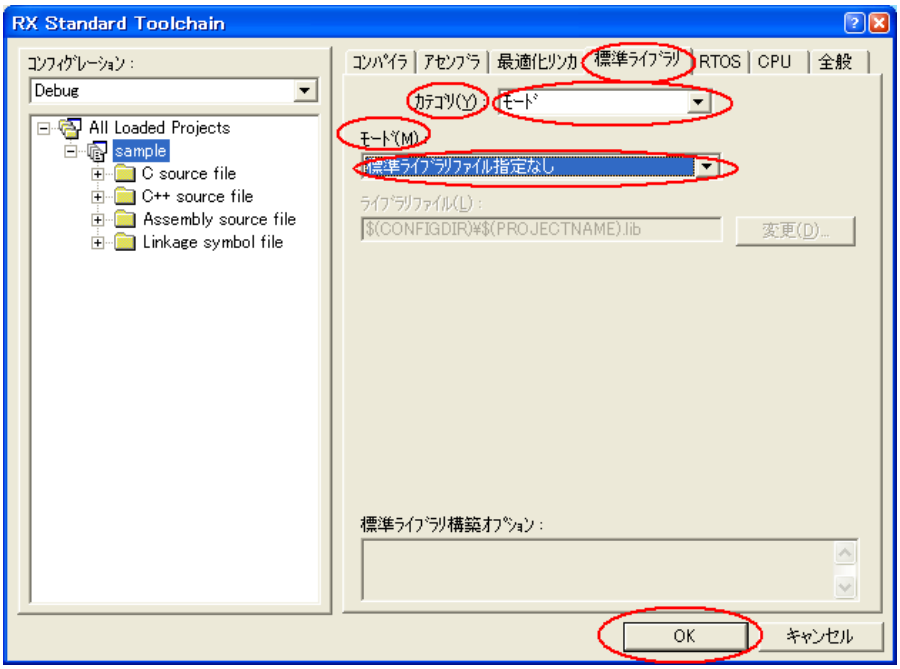
(6) (4)でコピーしたライブラリファイルを選択し、「追加」ボタンを押します。



(7)「ビルド(B)」→「RX Standard Toolchain...」 を選択します。



(8)「標準ライブラリ」タブを選択します。
(9)「カテゴリ(Y)」のプルダウンメニューで「モード」を選択します。
(10)「モード(M)」のプルダウンメニューで「既存標準ライブラリ指定なし」を選択します。
(11)「OK」ボタンを押し、以上の設定を保存します。



これでプロジェクトの設定は完了です。
ビルドすれば、(6)で選択したライブラリがリンクされます。

2.3 最適化リンケージエディタに直接指定する場合

製品に含まれているライブラリファイルを、上記 2.1 の格納場所から、任意のディレクトリにコピーしてください。
次に、最適化リンケージエディタの Library オプションにコピーしたライブラリファイルを指定して、リンクしてください。

以上