お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

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。

SuperH RISC engine C/C++ コンパイラパッケージ

アプリケーションノート：＜コンパイラ活用ガイド＞C++言語活用 編

本ドキュメントでは、SuperH RISC engine C/C++ コンパイラ V.9 における C++言語の使用方法、注意点を説明します。

目次

1. C++言語の使用方法 2
1.1 グローバルクラスオブジェクトの初期処理と後処理 2
1.2 C言語オブジェクトの参照方法 4
1.3 new, delete の実装方法 5
1.4 スタティックメンバ変数 7

2. オプション活用法 9
2.1 組み込み向けC++言語 (Embedded C++：EC++) 9
2.2 実行時型情報 10
2.3 例外処理機能 13
2.4 プレリンカの起動抑止 13

3. C++言語記述のメリット・デメリット 14
3.1 コンストラクタ (1) 15
3.2 コンストラクタ (2) 17
3.3 デフォルトパラメータ 19
3.4 インライン展開 20
3.5 クラスメンバ関数 21
3.6 operator 演算子 23
3.7 関数のオーバロード 25
3.8 リファレンス型 27
3.9 スタティック関数 28
3.10 スタティックメンバ変数 31
3.11 匿名 union 33
3.12 仮想関数 34

4. よくあるお問い合わせ 38
4.1 C++言語仕様に関する機能 38
4.2 C++言語の例外処理を記述するとL2310 リンカエラーが発生する。 39
4.3 C++言語を使用している時、#pragma section の指定が正しく行えない。 40
4.4 EC++クラスライブラリのリエントラント性は？ 41

ホームページとサポート窓口<website and support,ws> 49

1. C++言語の使用方法

1.1 グローバルクラスオブジェクトの初期処理と後処理

■ポイント

C++言語でグローバルクラスオブジェクトを使用する場合、初期処理関数(_CALL_INIT)と後処理関数(_CALL_END)をmain 関数の前後で呼び出す必要があります。

■グローバルクラスオブジェクトとは？

下記のようにクラスオブジェクトの宣言を、関数の外で宣言しているものです。

（関数内クラスオブジェクト

```
void main(void)
{
  X XSample(10);
  X* P = &XSample;
  P->Sample2();
}
```

（グローバルクラスオブジェクト宣言）

```
X XSample(10);
void main(void)
{
     X* P = &XSample;
     P->Sample2();
}
```

関数の外で宣言

■なぜ初期処理／後処理が必要か？

上記のように、関数内でクラスオブジェクト宣言している場合、関数 main を実行しているときに、クラス X のコンストラクタを呼び出します。

それに対し、グローバルなクラスオブジェクト宣言は、関数を実行しても宣言が実行されることはありません。

そのため、main 関数の呼び出し前に_CALL_INIT を呼び出して、明示的にクラス X のコンストラクタを呼び出す必要があります。また同様に_CALL_END を main 関数後に呼び出しクラス X のデストラクタを呼び出すようにします。

■_CALL_INIT/_CALL_END の使用時と未使用時の動作

クラス X のメンバ変数 x の値を、参照した場合の値を下記に示します。

未使用の場合は以下のように、正しい値が得られず while 文内の式を実行しません。

（メンバ変数 x の値）

_CALL_INIT 使用時　→10

_CALL_INIT 未使用時→0

```
class X{
  int x;
public:
   X(int n){x = n}; // constructor
   ~X(){}           // destructor
   void Sample2(void);
};
X XSample(10); // global class object
void X::Sample2(void)
{
  while(x == 10)
  {
  }
}
void main(void)
{
  X* P = &XSample;
```

←参照場所

■_CALL_INIT/_CALL_END の呼び出し方法

main 関数の呼び出し前後で以下のように記述します。

```
void INIT(void)
{
    _INITSCT();
    _CALL_INIT();
    main();
    _CALL_END();
}
```

また、HEW をご使用の場合は、resetprg.c の_CALL_NIT/_CALL_END 呼び出し箇所のコメントを削除します。

（resetprg.c の PowerON_Reset 関数）

```
__entry(vect=0) void PowerON_Reset(void)
{
    set_imask_ccr(1);
    _INITSCT();

   _CALL_INIT();      // Remove the comment when you use global class object

  // _INIT_IOLIB();       // Remove the comment when you use SIM I/O

  // errno=0;           // Remove the comment when you use errno
  // srand(1);          // Remove the comment when you use rand()
  // _s1ptr=NULL;       // Remove the comment when you use strtok()

    HardwareSetup();  // Use Hardware Setup
    set_imask_ccr(0);

    main();

  // _CLOSEALL();       // Remove the comment when you use SIM I/O

   _CALL_END();      // Remove the comment when you use global class object

    sleep();
}
```

1.2 C 言語オブジェクトの参照方法

■ポイント

「extern "C"」宣言を用いることにより、既存の C オブジェクトプログラムの財産を、直接 C++プログラムから利用することができます。
また、C++オブジェクトの財産を、C プログラムから利用することができます。

■使用例

1.「extern "C"」宣言を用いることによりCオブジェクトプログラムの関数を参照できます。

（C++プログラム）

```
extern "C" void CFUNC();
void main(void)
{
    X XCLASS;
    XCLASS.SetValue(10);

    CFUNC();
}
```

（Cプログラム）

```
extern void CFUNC();
void CFUNC()
{
    while(1)
    {
        a++;
    }
}
```

2.「extern "C"」宣言を用いることによりC++オブジェクトプログラムの関数を参照できます。

（Cプログラム）

```
void CFUNC()
{
    CPPFUNC();
}
```

（C++プログラム）

```
extern "C" void CPPFUNC();
void CPPFUNC(void)
{
    while(1)
    {
        a++;
    }
}
```

■注意事項

1. エンコード方式、実行方式を変更したため、旧バージョン(Ver.5)のコンパイラが生成したC++のオブジェクトはリンクできません。
   必ずリコンパイルしてから使用してください。

2. 上記方法で呼び出した関数は、オーバロードすることはできません。

1.3 new, delete の実装方法

■ポイント

new を使用する場合は、低水準関数を実装する必要があります。

■説明

組み込みシステムにおいて new を使用する場合、実際のヒープメモリの動的な確保は malloc を使用することによって実現しています。

よって、malloc 使用時と同様に低水準インタフェースルーチン(sbrk)を実装し、割り付けるヒープメモリの容量を指定する必要があります。

■実装方法

HEW を使用する場合､ワークスペース作成時に｢ヒープメモリ使用｣がチェックされていることを確認してください。
チェックすることにより、次紙に示す sbrk.c と sbrk.h が自動的に生成されます。
確保するヒープメモリの容量は Heap Size で指定してください。
ワークスペース作成後に容量を変更する場合は sbrk.h で HEAPSIZE に定義する値を変更してください。

また、HEW を使用しない場合､次紙に示すファイルを作成しプロジェクトに実装してください。

```
 (sbrk.c)

#include <stdio.h>
#include "sbrk.h"

//const size_t _sbrk_size=    /* Specifies the minimum unit of      */
                            /* the defined heap area                         */

static  union  {
    long  dummy ;          /* Dummy for 4-byte boundary         */
    char heap[HEAPSIZE]; /* Declaration of the area managed   */
                         /*                        by sbrk */
 }heap_area ;

static  char  *brk=(char *)&heap_area;/* End address of area assigned    */

/***************************************************************************/
/*              sbrk:Data write
                                                                        */
/*              Return value:Start address of the assigned area (Pass)
                       */
/*                                          -1
        (Failure)                                                           */
/***************************************************************************/
char  *sbrk(size_t size)              /* Assigned area size    */
{
    char  *p;

    if(brk+size>heap_area.heap+HEAPSIZE) /* Empty area size            */
      return (char *)-1 ;

    p=brk ;                                          /* Area assignment
          */
    brk += size ;                           /* End address update    */
    return p ;
}
```

```
(sbrk.h)

/* size of area managed by sbrk */
#define HEAPSIZE 0x420
```

1.4 スタティックメンバ変数

■説明

C++では、クラスのメンバ変数を static 属性にすると、そのメンバ変数はクラス型の複数のオブジェクト間で共有することができます。
これにより、同じクラス型の複数オブジェクト間で、共通なフラグなどに利用することができるので便利です。

■使用例

main 関数内で、クラス A 型のオブジェクトを 5 つ作成します。
static なメンバ変数 num の初期値は 0 です。この値がオブジェクトの作成毎に、コンストラクタでインクリメントされます。
static なメンバ変数 num は各オブジェクト間で共有されるので、変数 num の値は 5 まで上昇します。

■FAQ

スタティックメンバ変数使用時のよくある質問を以下に示します。

【L2310 エラー発生】
static メンバ変数を使用したとき、リンク時に「** L2310 (E) Undefined external symbol "クラス名::static メンバ変数名" referenced in "ファイル名"」が出力される。

【解決策】
static メンバ変数の実体が定義されていないため、エラーが発生しています。
次紙に示すように、以下の定義を追加してください。

初期値がある場合：`int A::num = 0;`
初期値がない場合：`int A::a;`

【初期値代入不可能】
初期値を持っている static メンバ変数に初期値が代入されていない。

【解決策】
初期値を持つ static メンバ変数は、初期値付き変数として扱われるためデフォルトで D セクションに生成されます。よって、最適化リンケージエディタの ROM 化支援オプションの指定、およびイニシャルルーチンで_INITSCT 関数による ROM から RAM への D セクションコピー*が必要です。

【注】* HEW でイニシャルルーチンを自動生成している場合は、本対策は不要です。

（C++プログラム）

```
class A
{
private:
        static int num;
public:
        A(void);
        ~A(void);
};

int A::num = 0;          ← スタティックメンバ変数の実体定義

void main(void)
{
        A a1;
        A a2;
        A a3;            ← クラス A 型クラスオブジェクト作成
        A a4;
        A a5;
}

A::A(void)
{
        ++num;           ← static メンバ変数をインクリメント
}

A::~A(void)
{
        --num;
}
```

2. オプション活用法

2.1 組み込み向け C++言語 (Embedded C++：EC++)

■説明

組み込みシステムでは、ROM/RAM サイズおよび実行速度が重要です。
組み込み向け C++言語(EC++)は C++言語のサブセットで、組み込みシステムに向かない機能を排除した言語仕様になっています。
よって、組み込みシステムに適したオブジェクトを生成できます。

■指定方法

ダイアログメニュー：C/C++タブ Category:[Other]の Check against EC++ language specification
コマンドライン　　：*eccp*

■未サポートキーワード

以下のキーワードを記述するとエラーメッセージを出力します。

catch、const_cast、dynamic_cast、explicit、mutable、namespace、reinterpret_cast、static_cast、template、throw、try、typeid、typename、using

■未サポート言語仕様

以下の言語仕様を記述するとウォーニングメッセージを出力します。

多重継承、仮想基底クラス

2.2　実行時型情報

■説明

C++では仮想関数を持つクラスオブジェクトの場合、実行時でなければ判明しない型が存在します。
このような状況を支援する機能として、実行時識別の機能をサポートしています。
C++でこの機能を使うには、type_info クラス,typeid 演算子,dynamic_cast 演算子を使います。
本コンパイラでは下記のオプションを指定することにより、実行時型情報が使えるようになります。
また、リンク時に以下のオプションで、プレリンカを起動する必要があります。

■指定方法

ダイアログメニュー：　CPUタブEnable/disable runtime type information
コマンドライン　　：　*rtti=on | off*

ダイアログメニュー：　Link/LibraryタブCategory:[Input]のPrelinker controlをAutoまたはRun prelinker
コマンドライン　　：　*noprelinkを指定しない（デフォルト）*

■type_info クラス,typeid 演算子の使用例

type_info クラスは、オブジェクトの実行時の型に関する識別操作のためのクラスです。
type_info クラスを使うと、プログラム実行時の型比較判定、クラスの型名取得などができるようになります。
type_info クラスを使うには、typeid 演算子で仮想関数を持つクラスオブジェクトを指定します。

```
#include <typeinfo.h>
#include <string>
class Base{
protected:
        string *pname1;
public:
        Base() {
                pname1 = new string;
                if (pname1)
                        *pname1 = "Base";
        }
        virtual string Show() {return *pname1;}
        virtual ~Base() {
                if (pname1)
                        delete pname1;
        }
};
class Derived : public Base{
        string *pname2;
public:
        Derived() {
                pname2 = new string;
                if (pname2)
                        *pname2 = "Derived";
        }
        string Show() {return *pname2;}
        ~Derived() {
                if (pname2)
                        delete pname2;
        }
};
void main(void)
{
        Base* pb = new Base;
    Derived* pd = new Derived;

        const type_info& t = typeid(pb);
        const type_info& t1 = typeid(pd);
        t.name();
        t1.name();
}
```

■dynamic_cast 演算子の使用例

dynamic_cast 演算子を使うと、たとえば仮想関数を含むクラスとその派生クラス間では、実行時に dynamic_cast 演算子を使って、派生クラス型のポインタや参照を、基本クラス型のポインタまたは参照にキャストすることができます。

```
#include <string>
class Base{                                   // 基本クラス
protected:
        string *pname1;
public:
        Base() {
                pname1 = new string;
                if (pname1)
                        *pname1 = "Base";
        }
        virtual string Show() {return *pname1;}   // 仮想関数
        virtual ~Base() {                         // 仮想デストラクタ
                if (pname1)
                        delete pname1;
        }
};
class Derived : public Base{                  // 派生クラス
        string *pname2;
public:
        Derived() {
                pname2 = new string;
                if (pname2)
                        *pname2 = "Derived";
        }
        string Show() {return *pname2;}           // 仮想関数
        ~Derived() {                              // 仮想デストラクタ
                if (pname2)
                        delete pname2;
        }
};
void main(void)
{
        Derived *pderived = new Derived;
        Base *pbase = dynamic_cast<Base *> (pderived);   // 実行時、Base *にキャスト

        string ddd;
        ddd = pbase-> Show();                     // クラス名 Base を取得

        delete pbase;
}
```

2.3 例外処理機能

■説明

C++には、C にはない例外というエラー処理のメカニズムがあります。
例外とは、プログラム内部のエラー箇所とエラー対処コードを結び付けるための仕組みです。
例外のメカニズムを使ってエラー対処コードを一箇所にまとめることができます。
本コンパイラでは以下のオプションを指定することにより使用できます。

■指定方法

ダイアログメニュー：CPUタブUse try,throw and catch of C++
コマンドライン　　：*exception*

■使用例

ファイル”INPUT.DAT”のオープンに失敗したとき、例外処理を発生させて標準エラー出力に、エラーを表示します。

```
（例外処理発生C++プログラム例）

void main(void)
{
        try
        {
                if ((fopen("INPUT.DAT","r"))==NULL){
                        char * cp = "cannot open input file¥n";
                        throw cp;
                }
        }
        catch(char *pstrError)
        {
                fprintf(stderr,pstrError);
                abort();
        }
        return;
}
```

■注意事項

コード性能が低下する場合があります。

2.4 プレリンカの起動抑止

■説明

プレリンカを起動するとリンク速度が遅くなりますが、C++のテンプレート機能、実行時型変換を使用していないときは動作させる必要はありません。
リンカをコマンドラインでご使用の場合は、以下の noprelink オプションを指定してください。
Hew をご使用の場合は、Prelinker control リストボックスが Auto であれば、自動で noprelink オプションの出力を制御します。

■指定方法

ダイアログメニュー：Link/LibraryタブCategory:[Input]のPrelinker control
コマンドライン　　：*noprelink*

3. C++言語記述のメリット・デメリット

C++を組み込みシステムで適用する場合、注意を払いながらプログラミングをしないと、予想以上に大きなオブジェクトサイズになる、またはスピードが低下する等のトラブルを招く可能性があります。本章では、C 言語と比較して性能劣化を招く事例と招かない事例をそれぞれ紹介します。

コンパイラはC++プログラムをコンパイルする際、内部的にC++プログラムをC プログラムに変換してオブジェクトを生成します。C++プログラムと変換後のC プログラムを比較し、各機能のコード効率への影響を記述します。

No.	機能	開発・保守	サイズ	処理速度	参照
1	コンストラクタ (1)	◎	△	△	3.1
2	コンストラクタ (2)	◎	△	△	3.2
3	デフォルトパラメータ	◎	○	○	3.3
4	インライン展開	○	△	○	3.4
5	クラスメンバ関数	◎	△	△	3.5
6	operator 演算子	◎	△	△	3.6
7	関数のオーバロード	◎	○	○	3.7
8	リファレンス型	◎	○	○	3.8
9	スタティック関数	◎	○	○	3.9
10	スタティックメンバ変数	◎	○	○	3.10
11	匿名 union	◎	○	○	3.11
12	仮想関数	◎	△	△	3.12

◎ ：性能低下なし ○: 使用上注意要 △: 性能

3.1 コンストラクタ (1)

開発・保守	◎	サイズ	△	処理速度	△

■ポイント

コンストラクタを使うとクラスオブジェクトを自動的に初期化できますが、以下のようにオブジェクトサイズや処理速度に影響するため、注意が必要です。

■使用例

クラス A のコンストラクタとデストラクタを作成しコンパイルします。クラス宣言箇所でコンストラクタ/デストラクタの呼び出しが入り、コンストラクタ／デストラクタ本体では判定が加わるためサイズ／処理速度に影響がでます。

```
（C++プログラム）

class A
{
private:
        int a;
public:
        A(void);
        ~A(void);
        int getValue(void){ return a; }
};

void main(void)
{
        A a;
        b = a.getValue();
}

A::A(void)
{
        a = 1234;
}

A::~A(void)
{
}
```

（変換後のＣプログラム）

```
struct A {
        int a;
};

void *_nw__FUl(unsigned long);
void __dl__FPv(void *);
void main(void);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);

void main(void)
{
        struct A a;
        __ct__A(&a);
        _ b = ((a.a));
        __dt__A(&a, 2);
}
```

コンストラクタ呼び出し

デストラクタ呼び出し

```
struct A * __ct__A( struct A *this)
{
        if ( this != (struct A *)0
    ||  (  this = (struct A *)_nw__FUl(4))
           != (struct A *)0 )
        {
                (this->a) = 1234;
        }
        return this;
}
```

コンストラクタ本体

```
void __dt__A( struct A *const this,
  int flag)
{
        if (this != (struct A *)0){
                if (flag & 1) {
                        dl__FPv((void
*)this);
                }
        }
        return;
}
```

デストラクタ本体

3.2　　コンストラクタ (2)

開発・保守	◎	サイズ	△	処理速度	△

■ポイント

クラスを配列で宣言する場合に、コンストラクタを使うとクラスオブジェクトを自動的に初期化できますが、以下のようにオブジェクトサイズや処理速度に影響するため注意が必要です。

■使用例

クラス A のコンストラクタとデストラクタを作成しコンパイルします。クラス宣言箇所でコンストラクタ／デストラクタの呼び出しが入りますが配列で宣言しているため、動的なメモリの割り当て／解放が必要になります。

動的なメモリの割り当て／解放のために、new,delete を使用します。

そのため、低水準関数を実装する必要があります。（実装方法は SuperH RISC engine C/C++**コンパイラ、アセンブラ、最適化リンケージエディタ　ユーザーズマニュアルの「**9.2.2　**実行環境の設定」**参照）

コンストラクタ／デストラクタ本体では判定と低水準関数の処理が加わるためサイズ／処理速度に影響が出ます。

```
（C＋＋プログラム）
class A
{
private:
        int a;
public:
        A(void);
        ~A(void);
        int getValue(void){ return a; }
};

void main(void)
{
        A a[5];
        b = a[0].getValue();
}

A::A(void)
{
        a = 1234;
}

A::~A(void)
{
}
```

（変換後のCプログラム）

```
struct A {
        int a;
};

void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
void main(void);
void *__vec_new();
void __vec_delete();
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);

void main(void)
{
        struct A a[5];
        __vec_new( (struct A *)a, 5, 4, __ct__A);
        _ b = ((a.a));
        __vec_delete( &a, 5, 4, __dt__A, 0, 0);
}
```

コンストラクタ呼び出し

デストラクタ呼び出し

```
struct A *__ct__A( struct A *this)
{
        if((this != (struct A *)0)
         || ( (this = (struct A
*)__nw__FUl(4)) != (struct A *)0) )
        {
                (this->a) = 1234;
        }
        return this;
}
```

コンストラクタ本体

```
void __dt__A( struct A *const this,
 int flag)
{
        if (this != (struct A *)0){
                if (flag & 1){

        __dl__FPv((void *)this);
                }
        }
        return;
}
```

デストラクタ本体

3.3 デフォルトパラメータ

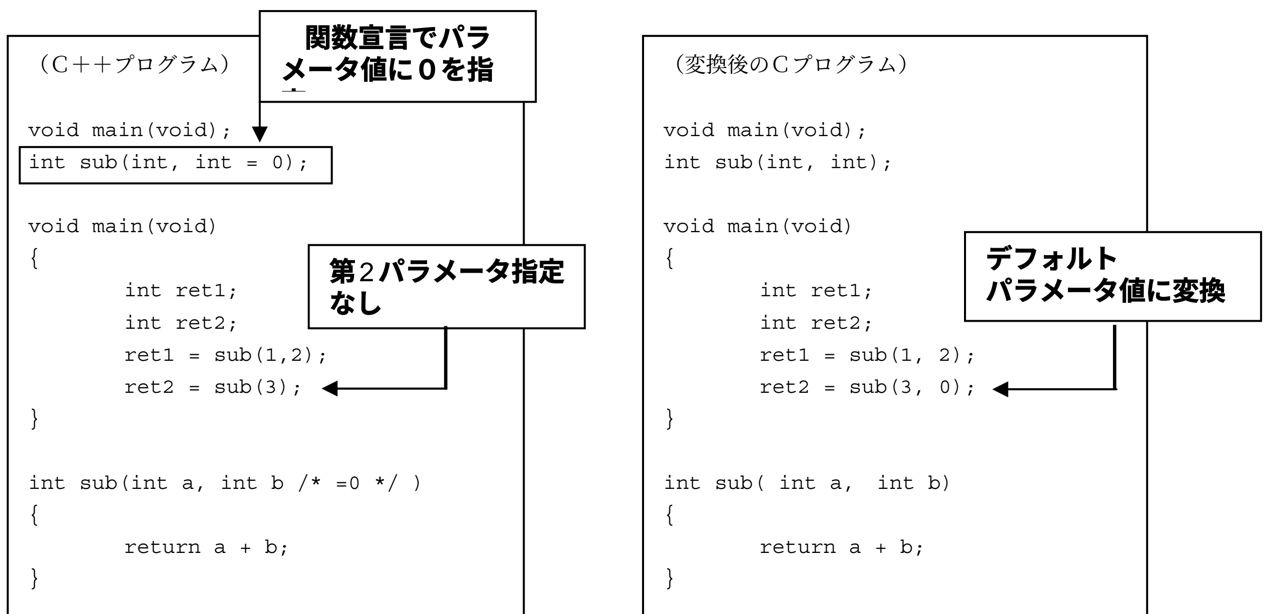
開発・保守	◎	サイズ	○	処理速度	○

■ポイント

C++では、デフォルトパラメータという、関数呼び出し時のデフォルト用法が使えます。
これは関数の宣言時に関数のパラメータにデフォルト値指定をすることにより使うことができます。
これにより多くの関数呼び出しではパラメータ指定の必要がなく、デフォルトのパラメータ値を使用することができ、開発効率が上がります。
また、パラメータを指定するとパラメータの値を変更することができます。

■使用例

関数 sub の宣言にデフォルトパラメータ値として、０を指定した場合の関数 sub 呼び出し例を以下に示します。
以下のように、関数 sub 呼び出しの際に、デフォルトパラメータの値で良い場合はパラメータを記述する必要がありません。しかも、C に変換してもプログラムの効率は悪化しておりません。
このようにデフォルトパラメータは開発･保守効率が良いことに加え、C 言語と比較してもデメリットがありません。



3.4　インライン展開

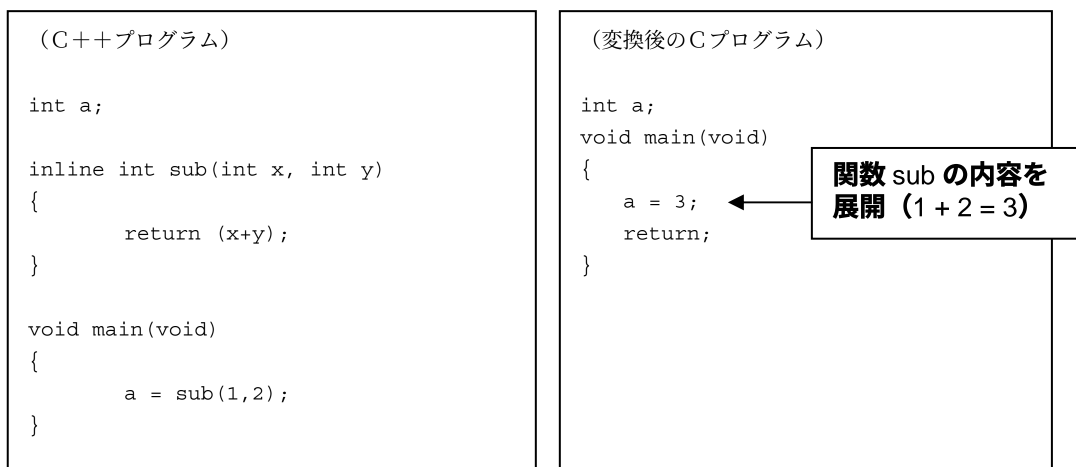
開発・保守	○	サイズ	△	処理速度	○

■ポイント

関数の本体定義を記述する際、先頭に inline を指定すると、その関数をインライン展開するので関数呼び出しのオーバヘッドがなくなり処理速度を向上させることができます。

■使用例

関数 sub を inline 指定し、メイン関数内にインライン展開します。その後に関数 sub 本体を削除します。
ただし、他のファイルから関数 sub を参照することはできません。
インライン展開は処理速度が確実に向上しますが、小さい関数に限定しないと、サイズが大きくなるので注意が必要です。



3.5 クラスメンバ関数

開発・保守	◎	サイズ	△	処理速度	△

■ポイント

クラスを定義すると、情報隠蔽が可能になり、開発・保守の効率が上がります。
しかし、サイズ／処理速度に影響が出るので注意が必要です。

■使用例

private クラスメンバ変数 a,b,c をアクセスするクラスメンバ関数 set,add を例とします。
クラスメンバ関数呼び出し時、C++プログラム上のパラメータ指定では、値のみもしくはパラメータなしです
しかし、変換後の C プログラムを見るとクラス A（struct A）のアドレスもパラメータとして渡されます。
また、クラスメンバ関数本体で private クラスメンバ変数 a,b,c をアクセスをしています。
しかし、this ポインタを使用してアクセスすることになります。
以上のことから、クラスメンバ関数を使用すると、サイズ／処理速度に影響が出るので注意が必要です。

```
（C++プログラム）

class A
{
private:
        int a;
        int b;
        int c;
public:
        void set(int, int, int);
        int add();
};

int main(void)
{
        A a;
        int ret;

        a.set(1,2,3);
        ret = a.add();

        return ret;
}
void A::set(int x, int y, int z)
{
        a = x;
        b = y;
        c = z;
}
int A::add()
{
        return (a += b + c);
}
```

（変換後のＣプログラム）

```
struct A {
        int a;
        int b;
        int c;
};
void set__A_int_int(struct A *const, int, int, int);
int add__A(struct A *const);

int main(void)
{
        struct A a;
        int ret;

        set__A_int_int(&a, 1, 2, 3);
        ret = add__A(&a);

        return ret;
}
void set__A_int_int(struct A *const this, int x, int y, int z)
{
        this->a = x;
        this->b = y;
        this->c = z;
        return;
}
int add__A(struct A *const this)
{
        return (this->a += this->b + this->c);
}
```

3.6 operator 演算子

開発・保守	○	サイズ	△	処理速度	△

■ポイント

C++では operator というキーワードにより、演算子を多重定義することができます。
これによりマトリックス演算やベクトル計算などのユーザの演算処理を、簡潔に記述することができます。
しかし、operator を使う場合はサイズ／処理速度に影響が出るので注意が必要です。

■使用例

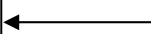
以下は単項演算子“＋”を operator キーワードで多重定義している例です。
これで下記 Vector クラスを宣言した場合、単項演算子“＋”がユーザの演算処理に変更できます。
しかし、変換後の C プログラムを見ると、this ポインタによる参照が行われているため、サイズ／処理速度に影響があります。

（C＋＋プログラム）

```
class Vector
{
private:
        int x;
        int y;
        int z;
public:
        Vector & operator+ (Vector &);
};

void main(void)
{
        Vector a,b,c;

        a = b + c;
}

Vector & Vector::operator+ (Vector & vec)
{
        static Vector ret;

        ret.x = x + vec.x;
        ret.y = y + vec.y;      ← ユーザの演算処理（加算）
        ret.z = z + vec.z;

        return ret;
}
```

（変換後のＣプログラム）

```
struct Vector {
        int x;
        int y;
        int z;
};

void main(void);
struct Vector *__plus__Vector_Vector(struct Vector *const, struct Vector *);

void main(void)
{
        struct Vector a;
        struct Vector b;
        struct Vector c;

        a = *__plus__Vector_Vector(&b, &c);
        return;
}

struct Vector *__plus__Vector_Vector( struct Vector *const this,  struct Vector *vec)
{
        static struct Vector ret;

        ret.x = this->x + vec->x;
        ret.y = this->y + vec->y;
        ret.z = this->z + vec->z;

        return &ret;
}
```



this **ポインタでの参照**

3.7 関数のオーバロード

開発・保守	◎	サイズ	○	処理速度	○

■ポイント

C++では異なる関数に同一名を与える“オーバロード”が可能です。
具体的には同じような処理をする関数で、引数の型だけが違う場合などに有効な機能です。
共通性のない関数に同じ名前をつけると、不具合のもとになるので注意が必要です。
本機能を使用しても、サイズ／処理速度に影響を与えません。

■使用例

第1､第2パラメータ同士を加算し、加算した値を戻り値とする例です。
関数名はすべてaddで、それぞれパラメータと戻り値の型が違います。
変換後のCプログラムのとおり、add関数呼び出し、add関数本体でコードサイズの増加はありません。
よって、サイズ／処理速度に影響を与えません。

```
（C＋＋プログラム）

void main(void);
int add(int,int);
float add(float,float);
double add(double,double);

void main(void)
{
        int    ret_i = add(1, 2);
        float  ret_f = add(1.0f, 2.0f);
        double ret_d = add(1.0, 2.0);
}

int add(int x,int y)
{
        return x+y;
}

float add(float x,float y)
{
        return x+y;
}

double add(double x,double y)
{
        return x+y;
}
```

（変換後のCプログラム）

```
void main(void);
int add__int_int(int, int);
float add__float_float(float, float);
double add__double_double(double, double);

void main(void)
{
        auto int ret_i;
        auto float ret_f;
        auto double ret_d;

        ret_i = add__int_int(1, 2);
        ret_f = add__float_float(1.0f, 2.0f);
        ret_d = add__double_double(1.0, 2.0);
}

int add__int_int( int x,  int y)
{
        return x + y;
}

float add__float_float( float x,  float y)
{
        return x + y;
}

double add__double_double( double x,  double y)
{
        return x + y;
}
```

3.8　リファレンス型

開発・保守	◎	サイズ	○	処理速度	○

■ポイント

パラメータをリファレンス型とすると、プログラムを簡潔に記述することができるので、開発・保守性が向上します。
また、リファレンス型を使用しても、サイズ／処理速度に影響を与えません。

■使用例

以下のようにポインタ渡しでなく、リファレンス型で渡すと、記述が簡潔になります。
また、リファレンス型はパラメータ a,b の値でなく、a,b のアドレスが渡されます。
リファレンス型を使用しても変換後のＣプログラムのとおり、サイズ／処理速度に影響を与えません。

（C++プログラム）

```
void main(void);
void swap(int&, int&);

void main(void)
{
        int a=100;
        int b=256;

        swap(a,b);
}

void swap(int &x, int &y)
{
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}
```

（変換後のＣプログラム）

```
void main(void);
void swap(int *, int *);

void main(void)
{
        int a=100;
        int b=256;

        swap(&a, &b);
}

void swap(int *x, int *y)
{
        int tmp;
        tmp = *x;
        *x = *y;
        *y = tmp;
}
```

3.9　スタティック関数

開発・保守	◎	サイズ	○	処理速度	○

■ポイント

クラスの構成が派生クラスなどにより、複雑になってくると、private 属性の static なクラスメンバ変数のアクセスが大変になり、public 属性にすることになってしまいます。

このようなときに、private 属性のまま static なクラスメンバ変数をアクセスするには、インタフェース用のメンバ関数を作成し、その関数に static を指定します。

このように、static なクラスメンバ変数のみにアクセスするのが、スタティック関数です。

■使用例

次紙のようにスタティック関数を使って、スタティックメンバ変数をアクセスします。

クラスを使用するのでクラスはコード効率に影響を与えますが、スタティック関数自体はサイズ／処理速度に影響を与えません。

■備考

スタティックメンバ変数については、「1.4　**スタティックメンバ変数**」を参照してください。

（C++プログラム）

```
class A
{
private:
        static int num;
public:
        static int getNum(void);
        A(void);
        ~A(void);
};

int A::num = 0;

void main(void)
{
        int num;

        num = A::getNum();

        A a1;
        num = a1.getNum();

        A a2;
        num = a2.getNum();
}

A::A(void)
{
        ++num;
}

A::~A(void)
{
        --num;
}

int A::getNum(void)
{
        return num;
}
```

（変換後のCプログラム）

```
struct A
{
        char __dummy;
};
void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
int getNum__A(void);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
int num__1A = 0;
void main(void)
{
        int num;
        struct A a1;
        struct A a2;

        num = getNum__A();

        __ct__A(&a1);
        num = getNum__A();

        __ct__A(&a2);
        num = getNum__A();

        __dt__A(&a2, 2);
        __dt__A(&a1, 2);
}
int getNum__A(void)
{
        return num__1A;
}
struct A *__ct__A( struct A *this)
{
        if ( (this != (struct A *)0)
        || ( (this = (struct A *)__nw__FUl(1)) != (struct A *)0) ){
                ++num__1A;
        }
        return this;
}
void __dt__A( struct A *const this, int flag)
{
        if (this != (struct A *)0){
                --num__1A;
                if(flag & 1){
                        __dl__FPv((void *)this);
                }
        }
        return;
}
```

スタティック関数（int getNum__A(void); を指す）

スタティックメンバ変数（int num__1A = 0; を指す）

スタティックメンバ変数にアクセス（return num__1A; を指す）

3.10　スタティックメンバ変数

開発・保守	◎	サイズ	○	処理速度	○

■ポイント

C++では、クラスのメンバ変数を static 属性にすると、そのメンバ変数はクラス型の複数のオブジェクト間で共有することができます。
これにより、同じクラス型の複数オブジェクト間で、共通なフラグなどに利用することができるので便利です。

■使用例

main 関数内で、クラス A 型のオブジェクトを 5 つ作成します。
static なメンバ変数 num の初期値は 0 です。この値がオブジェクトの作成ごとに、コンストラクタでインクリメントされます。
static なメンバ変数 num は各オブジェクト間で共有されるので、変数 num の値は 5 まで上昇します。
また、クラスを使用するのでクラス自体はコード効率に影響を与えます。
しかし、スタティックメンバ変数自体は、コンパイラの内部でメンバ変数 num は通常のグローバル変数のように扱われるので、サイズ／処理速度に影響を与えません。

■備考

スタティックメンバ変数の詳細は、「1.4　**スタティックメンバ変数**」を参照してください。

（C++プログラム）

```
class A
{
private:
        static int num;
public:
        A(void);
        ~A(void);
};

int A::num = 0;

void main(void)
{
        A a1;
        A a2;
        A a3;
        A a4;
        A a5;
}

A::A(void)
{
        ++num;
}

A::~A(void)
{
        --num;
}
```

クラス A 型クラスオブジェクト作成

static **メンバ変数をインクリメント**

（変換後のＣプログラム）

```
struct A
{
        char __dummy;
};
void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
int num__1A = 0;
void main(void)
{
        struct A a1;
        struct A a2;
        struct A a3;
        struct A a4;
        struct A a5;
        __ct__A(&a1);
        __ct__A(&a2);
        __ct__A(&a3);
        __ct__A(&a4);
        __ct__A(&a5);
        __dt__A(&a5, 2);
        __dt__A(&a4, 2);
        __dt__A(&a3, 2);
        __dt__A(&a2, 2);
        __dt__A(&a1, 2);
}
struct A *__ct__A( struct A *this)
{
        if( (this != (struct A *)0)
        || ( (this = (struct A *)__nw__FUl(1)) != (struct A *)0) ){
                ++num__1A;
        }
        return this;
}
void __dt__A( struct A *const this,  int flag)
{
        if(this != (struct A *)0){
                --num__1A;
                if (flag & 1){
                        __dl__FPv((void *)this);
                }
        }
        return;
}
```

3.11 匿名 union

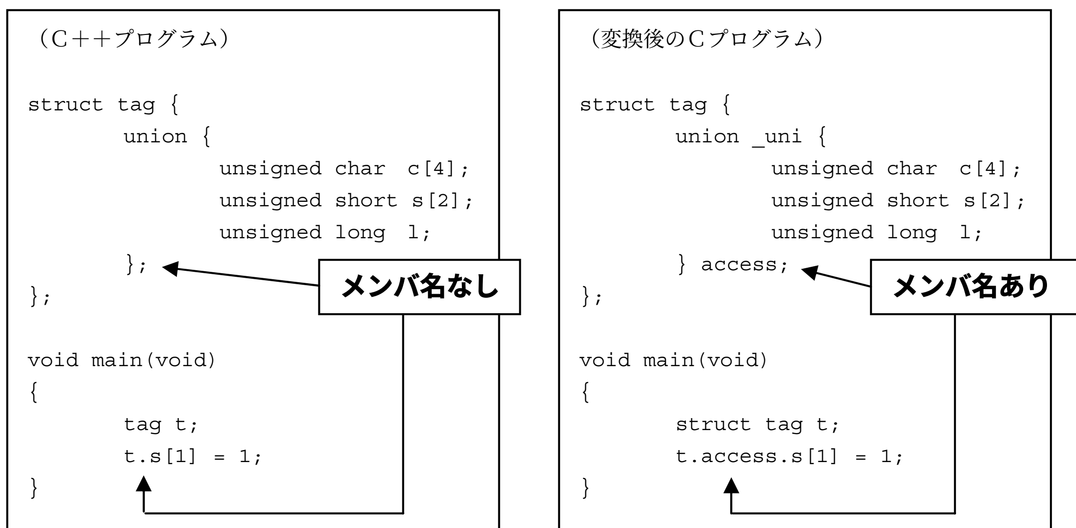
開発・保守	◎	サイズ	○	処理速度	○

■ポイント

C++で匿名 union を使うと、C 言語とは異なりメンバ名を指定せずに、メンバをダイレクトにアクセスすることができます。
これにより開発効率が向上します。また、サイズや処理速度に影響しません。

■使用例

以下のように関数 main で、union のメンバ変数 s をアクセスする場合を例とします。
C++プログラムではメンバ変数 s をダイレクトにアクセスしていますが、変換後の C プログラムではコンパイラが自動でメンバ名を作成し、このメンバ名を用いてアクセスしています。
このように簡潔な記述で、オブジェクト効率に影響を与えず、メンバ変数にアクセスできます。



3.12 仮想関数

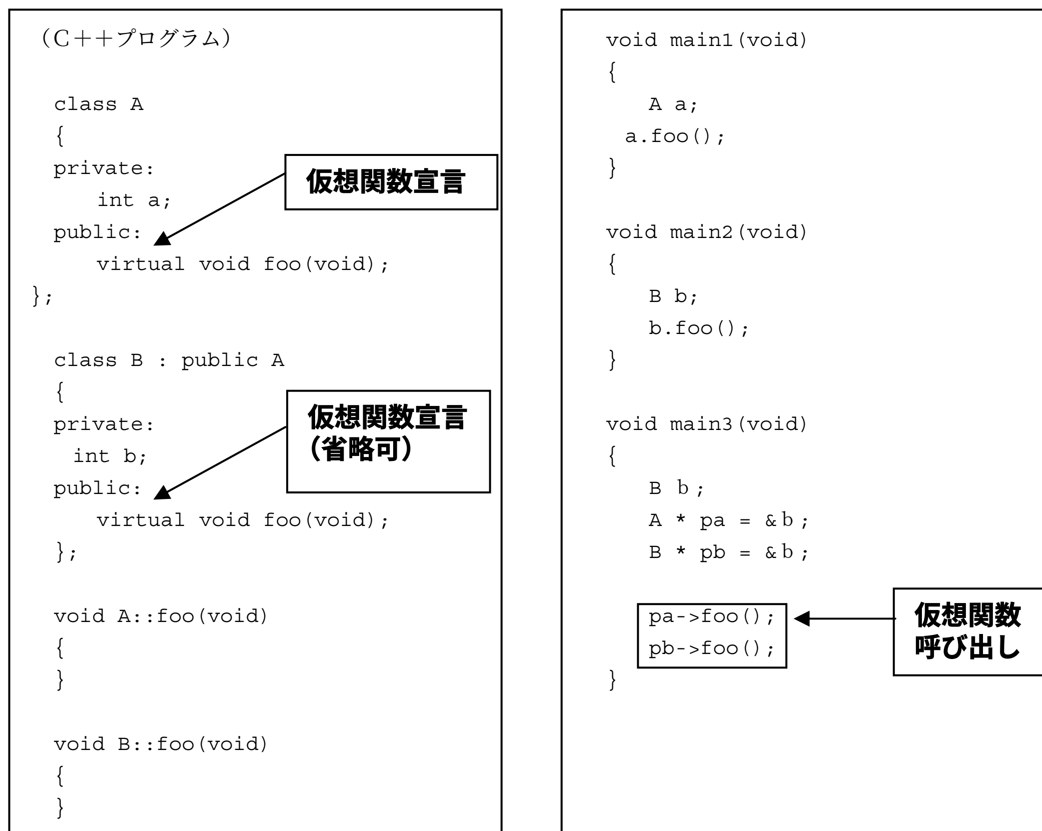
開発・保守	◎	サイズ	△	処理速度	△

■ポイント

仮想関数を使わない場合、以下のプログラムのような、基本クラス、派生クラスにそれぞれ存在する同名の関数があった場合、意図どおりに正しく関数呼び出しをすることができません。
仮想関数を宣言すると、上記の呼び出しが意図どおりに正しく行うことができます。
仮想関数を使うと、開発効率が向上しますが、サイズや処理速度に影響を与えるので注意が必要です。

■使用例

main3 関数の呼び出しでは、2 つのポインタにクラス B のアドレスを格納しています。
virtual 宣言をしているので、正しくクラス B の foo 関数を呼び出します。
virtual 宣言をしないと、一方はクラス A の foo 関数を呼び出します。
また、仮想関数を使うと次紙以降に示す、テーブルなどを作成するため、サイズや速度に影響を与えます。



■変換後のCプログラム（仮想関数のためのテーブルなど）

```
  struct __T5585724;
  struct __type_info;
  struct __T5584740;
  struct __T5579436;
struct A;
  struct B;
  extern void main1__Fv(void);
  extern void main2__Fv(void);
  extern void main3__Fv(void);
  extern void foo__1AFv(struct A *const);
  extern void foo__1BFv(struct B *const);
  struct __T5585724
  {
     struct __T5584740 *tinfo;
     long offset;
     unsigned char flags;
  };
  struct __type_info
  {
     struct __T5579436 *__vptr;
  };
  struct __T5584740
  {
     struct __type_info tinfo;
     const char *name;
     char *id;
     struct __T5585724 *bc;
  };
  struct __T5579436
  {
     long d;                        // this ポインタオフセット
     long i;                        // 未使用
     void (*f)();                    // 仮想関数コール用
  };
  struct A {                         // クラス A 宣言
     int a;
     struct __T5579436 *__vptr;        // 仮想関数テーブルへのポインタ
  };
  struct B {                         // クラス B 宣言
     struct A __b_A;
     int b;
  };
  static struct __T5585724 __T5591360[1];
  #pragma section $VTBL
  extern const struct __T5579436 __vtbl__1A[2];
  extern const struct __T5579436 __vtbl__1B[2];
  extern const struct __T5579436 __vtbl__Q2_3std9type_info[];
  #pragma section
  extern struct __T5584740 __T_1A;
  extern struct __T5584740 __T_1B;
```

```
static char __TID_1A;                    // 未使用
static char __TID_1B;                    // 未使用
static struct __T5585724 __T5591360[1] =   // 未使用
{
    {
        &__T_1A,
        0L,
        ((unsigned char)22U)
  }
};
#pragma section $VTBL
const struct __T5579436 __vtbl__1A[2] =     // クラスA用仮想関数テーブル
{
    {
        0L,                              // 未使用領域
        0L,                              // 未使用領域
        ((void (*)())&__T_1A)               // 未使用領域
    },
    {
        0L,                              // this ポインタオフセット
        0L,                              // 未使用領域
        ((void (*)())foo__1AFv)             // A::foo()へのポインタ
    }
};
const struct __T5579436 __vtbl__1B[2] =     // クラスB用仮想関数テーブル
{
    {
        0L,                              // 未使用領域
        0L,                              // 未使用領域
        ((void (*)())&__T_1B)               // 未使用領域
    },
    {
        0L,                              // this ポインタオフセット
        0L,                              // 未使用領域
        ((void (*)())foo__1BFv)             // B::foo()へのポインタ
    }
};
#pragma section
struct __T5584740 __T_1A =                 // クラスA用型情報(未使用)
{
    {
        (struct __T5579436 *)__vtbl__Q2_3std9type_info
    },
    (const char *)"A",
    &__TID_1A,
    (struct __T5585724 *)0
};
struct __T5584740 __T_1B =                 // クラスB用型情報(未使用)
{
    {
        (struct __T5579436 *)__vtbl__Q2_3std9type_info
    },
  (const char *)"B",
    &__TID_1B,
    __T5591360
};
```

■変換後のCプログラム（仮想関数の呼び出し）

```
  void main1__Fv(void)
  {
     struct A _a;
     _a.__vptr = __vtbl__1A;
     foo__1AFv( &_a );                           // A::foo()のコール
     return;
}
  void main2__Fv(void)
  {
     struct B _b;
     _b.__b_A.__vptr = __vtbl__1A;
     _b.__b_A.__vptr = __vtbl__1B;
     foo__1BFv( &_b );                           // B::foo()のコール
     return;
  }
  void main3__Fv(void)
  {
     struct __T5579436 *_tmp;
     struct B _b;
     struct A *_pa;
     struct B *_pb;

     (*((struct A*)(&_b))).__vptr = __vtbl__1A;
     (*((struct A*)(&_b))).__vptr = __vtbl__1B;
     _pa = (struct A *)&_b;
     _pb = &_b;

     _tmp = _pa->__vptr + 1;
     (  (void (*)(struct A *const))  _tmp->f  )  (  (struct A *)_pa + _tmp->d);
     // B::foo()をコール(_pa の指すオブジェクトが B のため)

     _tmp = _pb->__b_A.__vptr + 1;
     (  (void (*)(struct B *const))  _tmp->f  )  (  (struct B *)_pb + _tmp->d );
     // B::foo()をコール

     return;
  }
```

4. よくあるお問い合わせ

4.1 C++言語仕様に関する機能

■質問

C++言語で開発していく上で、用意されている機能はありますか？

■回答

SuperH RISC engine C/C++コンパイラでは C++言語開発用に以下の機能をサポートしています。

（1） EC++クラスライブラリサポート
EC++クラスライブラリをサポートすることにより、C++プログラムから組み込み向けC++クラスライブラリを標準的に利用することができます。
このライブラリには次の4種類があります。
・ストリーム入出力用クラスライブラリ
・メモリ操作用ライブラリ
・複素数計算用クラスライブラリ
・文字列操作用クラスライブラリ

それぞれの内容の詳細は SuperH RISC engine C/C++**コンパイラ、アセンブラ、最適化リンケージエディタ ユーザーズマニュアル**の**「10.4.2 EC++クラスライブラリ」**を参照してください。

（2） EC++言語仕様シンタックスチェック機能
コンパイラのオプション指定すると、EC++言語仕様に基づいてC++プログラムをシンタックスチェックします。

【指定方法】
ダイアログメニュー：コンパイラタブ カテゴリ:[その他] その他のオプション:EC++言語に基づいたチェック
コマンドライン　　：*ecpp*

（3） その他の機能
その他にC++プログラムを効率よく記述するために次のC++機能を理解して使ってください。
＜Better C機能＞
・関数インライン展開
・＋,－,<<などの演算子カスタマイズ
・多重定義関数による名称の単純化
・コメント記述容易
＜オブジェクト指向機能＞
・クラス
・コンストラクタ
・仮想関数

C++プログラムでライブラリ関数を使用する場合の実行環境の設定については SuperH RISC engine C/C++**コンパイラ、アセンブラ、最適化リンケージエディタ ユーザーズマニュアル**の**「9.2.2(4) C/C++ライブラリ関数の初期設定（_INTLIB）」**を参照してください。

4.2　C++言語の例外処理を記述すると L2310 リンカエラーが発生する。

■質問

L2310 (E) Undefined external symbol __curr_eh_stack_entry が発生するが、対処方法が判りません。

■回答

__curr_eh_stack_entry は C++の例外処理(try・catch)を用いたとき、コンパイラ内部で使用される外部変数です。EC++(-ecpp)と C++の例外処理(try・catch)を両方選択した場合、EC++の言語仕様では例外処理はサポート外のため、本エラーが発生します。EC++ と 例外処理を併用しないようにしてください。尚、標準ライブラリ構築時に全ての項目を選択すると、EC++が使用されますので、ご注意下さい。

4.3 C++言語を使用している時、#pragma section の指定が正しく行えない。

■質問

C++言語を使用している時、#pragma section の指定が正しく行えない時があります。対処方法を教えてください。

■回答

1．C++言語を使用している時は、コンパイル単位の最初の宣言に対して#pragma section の指定が必要となります。

```
(例 1)
#pragma section _A
extern int N;
#pragma section

int N

/* 変数 N は B_A セクションに配置されます */


(例 2)
extern int N;

#pragma section _A
int N
#pragma section

/* 変数 N は B セクション(デフォルト)に配置されます */
```

2．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 の指定が必要となります。

```
（例）
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
 };

// 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
```

4.4 EC++クラスライブラリのリエントラント性は？

■質問

EC++クラスライブラリのリエントラント性を教えてください。

■回答

EC++クラスライブラリのリエントラント性は以下の通りです。 new は、標準ライブラリ構築ツールで reent オプションを指定しており、且つ exception 指定が無い場合はリエントラントに実行できます。

○:リエントラント　△:errno を設定　＊:exception 指定時に非リエントラント

	FunctionName	Reentrant	
ios	ios_base & boolalpha(ios_base&);	○	
	ios_base & noboolalpha(ios_base&);	○	
	ios_base & showbase(ios_base&);	○	
	ios_base & noshowbase(ios_base&);	○	
	ios_base & showpoint(ios_base&);	○	
	ios_base & noshowpoint(ios_base&);	○	
	ios_base & showpos(ios_base&);	○	
	ios_base & noshowpos(ios_base&);	○	
	ios_base & skipws(ios_base&);	○	
	ios_base & noskipws(ios_base&);	○	
	ios_base & uppercase(ios_base&);	○	
	ios_base & nouppercase(ios_base&);	○	
	ios_base & internal(ios_base&);	○	
	ios_base & left(ios_base&);	○	
	ios_base & right(ios_base&);	○	
	ios_base & dec(ios_base&);	○	
	ios_base & hex(ios_base&);	○	
	ios_base & oct(ios_base&);	○	
	ios_base & fixed(ios_base&);	○	
	ios_base & scientific(ios_base&);	○	
istream	istream& operator>>(istream&, char&);	○	＊
	istream& operator>>(istream&, unsigned char&);	○	＊
	istream& operator>>(istream&, signed char&);	○	＊
	istream& operator>>(istream&, char*);	○	＊
	istream& operator>>(istream&, unsigned char*);	○	＊
	istream& operator>>(istream&, signed char*);	○	＊
	istream& ws(istream& is);	○	＊
ostream	ostream& operator<<(ostream&, char);	○	＊
	ostream& operator<<(ostream&, const char*);	○	＊
	ostream& operator<<(ostream&, signed char);	○	＊
	ostream& operator<<(ostream&, unsigned char);	○	＊
	ostream& operator<<(ostream&, const signed char*);	○	＊
	ostream& operator<<(ostream&, const unsigned char*);	○	＊
	ostream & endl(ostream &);	○	＊
	ostream & ends(ostream &);	○	＊
	ostream & flush(ostream &);	○	＊
iostream	extern istream cin;	○	
	extern ostream cout;	○	

iomanip	smanip resetiosflags(ios_base::fmtflags);	○	*
	smanip setiosflags(ios_base::fmtflags);	○	*
	smanip setbase(int);	○	*
	smanip setfill(char);	○	*
	smanip setprecision(int);	○	*
	smanip setw(int);	○	*
complex	float_complex operator+(const float_complex&);	○	*
	float_complex operator-(const float_complex&);	○	*
	float_complex operator+(const float_complex&, const float_complex&);	○	*
	float_complex operator+(const float_complex&, const float&);	○	*
	float_complex operator+(const float&, const float_complex&);	○	*
	float_complex operator-(const float_complex&, const float_complex&);	○	*
	float_complex operator-(const float_complex&, const float&);	○	*
	float_complex operator-(const float&, const float_complex&);	○	*
	float_complex operator*(const float_complex&, const float_complex&);	○	*
	float_complex operator*(const float_complex&, const float&);	○	*
	float_complex operator*(const float&, const float_complex&);	○	*
	float_complex operator/(const float_complex&, const float_complex&);	○	*
	float_complex operator/(const float_complex&, const float&);	○	*
	float_complex operator/(const float&, const float_complex&);	○	*
	bool operator==(const float_complex&, const float_complex&);	○	
	bool operator==(const float_complex&, const float&);	○	
	bool operator==(const float&, const float_complex&);	○	
	bool operator!=(const float_complex&, const float_complex&);	○	
	bool operator!=(const float_complex&, const float&);	○	
	bool operator!=(const float&, const float_complex&);	○	
	istream& operator>>(istream&, float_complex&);	○	*
	ostream& operator<<(ostream&, const float_complex&);	○	
	double_complex operator+(const double_complex&);	○	*
	double_complex operator-(const double_complex&);	○	*
	double_complex operator+(const double_complex&, const double_complex&);	○	*
	double_complex operator+(const double_complex&, const double&);	○	*
	double_complex operator+(const double&, const double_complex&);	○	*
	double_complex operator-(const double_complex&, const double_complex&);	○	*
	double_complex operator-(const double_complex&, const double&);	○	*
	double_complex operator-(const double&, const double_complex&);	○	*
	double_complex operator*(const double_complex&, const double_complex&);	○	*
	double_complex operator*(const double_complex&, const double&);	○	*
	double_complex operator*(const double&, const double_complex&);	○	*
	double_complex operator/(const double_complex&, const double_complex&);	○	*
	double_complex operator/(const double_complex&, const double&);	○	*
	double_complex operator/(const double&, const double_complex&);	○	*
	bool operator==(const double_complex&, const double_complex&);	○	
	bool operator==(const double_complex&, const double&);	○	
	bool operator==(const double&, const double_complex&);	○	
	bool operator!=(const double_complex&, const double_complex&);	○	
	bool operator!=(const double_complex&, const double&);	○	
	bool operator!=(const double&, const double_complex&);	○	
	istream& operator>>(istream&, double_complex&);	○	*
	ostream& operator<<(ostream&, const double_complex&);	○	
	float real(const float_complex&);	○	*

	float imag(const float_complex&);	○	*
	float abs(const float_complex&);	△	
	float arg(const float_complex&);	△	
	float norm(const float_complex&);	○	*
	float_complex conj(const float_complex&);	○	*
	float_complex polar(const float&, const float&);	△	*
	double real(const double_complex&);	○	*
	double imag(const double_complex&);	○	*
	double abs(const double_complex&);	△	
	double arg(const double_complex&);	△	
	double norm(const double_complex&);	○	*
	double_complex conj(const double_complex&);	○	*
	double_complex polar(const double&, const double&);	○	*
	float_complex cos (const float_complex&);	△	*
	float_complex cosh (const float_complex&);	△	*
	float_complex exp (const float_complex&);	△	*
	float_complex log (const float_complex&);	△	*
	float_complex log10(const float_complex&);	△	*
	float_complex pow(const float_complex&, int);	△	
	float_complex pow(const float_complex&, const float&);	△	
	float_complex pow(const float_complex&, const float_complex&);	△	
	float_complex pow(const float&, const float_complex&);	△	
	float_complex sin (const float_complex&);	△	*
	float_complex sinh (const float_complex&);	△	*
	float_complex sqrt (const float_complex&);	△	*
	float_complex tan (const float_complex&);	○	
	float_complex tanh (const float_complex&);	○	
	double_complex cos (const double_complex&);	△	*
	double_complex cosh (const double_complex&);	△	*
	double_complex exp (const double_complex&);	△	*
	double_complex log (const double_complex&);	△	*
	double_complex log10(const double_complex&);	△	*
	double_complex pow(const double_complex&, int);	△	
	double_complex pow(const double_complex&, const double&);	△	
	double_complex pow(const double_complex&, const double_complex&);	△	
	double_complex pow(const double&, const double_complex&);	△	
	double_complex sin (const double_complex&);	△	*
	double_complex sinh (const double_complex&);	△	*
	double_complex sqrt (const double_complex&);	△	*
	double_complex tan (const double_complex&);	○	
	double_complex tanh (const double_complex&);	○	
string	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);	○	*
	bool operator == (const string &lhs,const string &rhs);	○	*
	bool operator == (const char *lhs,const string &rhs);	○	*
	bool operator == (const string &lhs,const char *rhs);	○	*
	bool operator != (const string &lhs,const string &rhs);	○	*
	bool operator != (const char *lhs,const string &rhs);	○	*

	bool operator != (const string &lhs,const char *rhs);	○	*
	bool operator < (const string &lhs,const string &rhs);	○	*
	bool operator < (const char *lhs,const string &rhs);	○	*
	bool operator < (const string &lhs,const char *rhs);	○	*
	bool operator > (const string &lhs,const string &rhs);	○	*
	bool operator > (const char *lhs,const string &rhs);	○	*
	bool operator > (const string &lhs,const char *rhs);	○	*
	bool operator <= (const string &lhs,const string &rhs);	○	*
	bool operator <= (const char *lhs,const string &rhs);	○	*
	bool operator <= (const string &lhs,const char *rhs);	○	*
	bool operator >= (const string &lhs,const string &rhs);	○	*
	bool operator >= (const char *lhs,const string &rhs);	○	*
	bool operator >= (const string &lhs,const char *rhs);	○	*
	void swap(string &lhs, string &rhs);	○	*
	istream & operator >> (istream &is,string &str);	○	*
	ostream & operator << (ostream &os,const string &str);	○	*
	istream & getline (istream &is,string &str,char delim);	○	*
	istream & getline (istream &is,string &str);	○	*
	double_complex::operator /=(const double_complex &)	△	*
	float_complex::operator /=(const float_complex &)	△	
	double_complex::operator *=(const double_complex &)	○	*
	float_complex::operator /=(const float_complex &)	○	*
	float_complex::operator *=(const float_complex &)	○	*
	float_complex::float_complex(const double_complex &)	○	*
	istream::operator >>(float &)	○	*
	istream::operator >>(double &)	○	*
	istream::operator >>(long double &)	○	*
	istream::_ec2p_getfstr(char *, unsigned int)	○	*
	istream::operator >>(bool &)	○	*
	istream::operator >>(short &)	○	*
	istream::operator >>(unsigned short &)	○	*
	istream::operator >>(int &)	○	*
	istream::operator >>(unsigned int &)	○	*
	istream::operator >>(long &)	○	*
	istream::operator >>(unsigned long &)	○	*
	istream::operator >>(void *&)	○	*
	istream::_ec2p_getistr(char *, unsigned int, int)	○	*
	istream::_ec2p_strtoul(char *, char **, int)	○	*
	istream::operator >>(streambuf *)	○	*
	istream::get()	○	*
	istream::get(char &)	○	*
	istream::get(signed char &)	○	*
	istream::get(unsigned char &)	○	*
	istream::get(char *, long)	○	*
	istream::get(char *, long, char)	○	*
	istream::get(signed char *, long)	○	*
	istream::get(signed char *, long, char)	○	*
	istream::get(unsigned char *, long)	○	*
	istream::get(unsigned char *, long, char)	○	*
	istream::get(streambuf &)	○	*
	istream::get(streambuf &, char)	○	*

	istream::getline(char *, long)	○	*
	istream::getline(char *, long, char)	○	*
	istream::getline(signed char *, long)	○	*
	istream::getline(signed char *, long, char)	○	*
	istream::getline(unsigned char *, long)	○	*
	istream::getline(unsigned char *, long, char)	○	*
	istream::_ec2p_gets(char *, long, char, int)	○	*
	istream::ignore(long, long)	○	*
	istream::peek()	○	*
	istream::putback(char)	○	*
	istream::unget()	○	*
	istream::sync()	○	*
	istream::tellg()	○	*
	istream::seekg(long)	○	*
	istream::seekg(long, int)	○	*
	streambuf::_$gptr() const	○	*
	streambuf::_$egptr() const	○	*
	istream::read(char *, long)	○	*
	istream::read(signed char *, long)	○	*
	istream::read(unsigned char *, long)	○	*
	istream::readsome(char *, long)	○	*
	istream::readsome(signed char *, long)	○	*
	istream::readsome(unsigned char *, long)	○	*
	istream::sentry::sentry(istream &, bool)	○	*
	ostream::operator <<(void *)	○	*
	ostream::operator <<(streambuf *)	○	*
	ostream::operator <<(float)	○	*
	ostream::operator <<(double)	○	*
	ostream::operator <<(long double)	○	*
	ostream::operator <<(short)	○	*
	ostream::operator <<(unsigned short)	○	*
	ostream::operator <<(int)	○	*
	ostream::operator <<(unsigned int)	○	*
	ostream::operator <<(long)	○	*
	ostream::operator <<(unsigned long)	○	*
	ostream::sentry::sentry(ostream &)	○	*
	ostream::sentry::~sentry()	○	*
	ostream::put(char)	○	*
	ostream::write(const char *, long)	○	*
	ostream::write(const signed char *, long)	○	*
	ostream::write(const unsigned char *, long)	○	*
	ostream::flush()	○	*
	ostream::tellp()	○	*
	ostream::seekp(long)	○	*
	ostream::seekp(long, int)	○	*
	string::_$size() const	○	*
	string::operator +=(const string &)	○	*
	string::operator +=(const char *)	○	*
	string::operator +=(char)	○	*
	string::append(const string &)	○	*
	string::append(const string &, unsigned long, unsigned long)	○	*

	string::append(const char *, unsigned long)	○	*
	string::append(const char *)	○	*
	string::append(unsigned long, char)	○	*
	string::_$c_str() const	○	*
	string::operator =(const string &)	○	*
	string::operator =(const char *)	○	*
	string::operator =(char)	○	*
	string::assign(const string &)	○	*
	string::assign(const string &, unsigned long, unsigned long)	○	*
	string::assign(const char *, unsigned long)	○	*
	string::assign(const char *)	○	*
	string::assign(unsigned long, char)	○	*
	string::swap(string &)	○	*
	string::_$size() const	○	*
	string::_$c_str() const	○	*
	string::compare(const string &) const	○	*
	string::compare(unsigned long, unsigned long, const string &) const	○	*
	string::compare(unsigned long, unsigned long, const string &, unsigned long, unsigned long) const	○	*
	string::compare(const char *) const	○	*
	string::compare(unsigned long, unsigned long, const char *, unsigned long) const	○	*
	string::find(const string &, unsigned long) const	○	*
	string::find(const char *, unsigned long, unsigned long) const	○	*
	string::find(const char *, unsigned long) const	○	*
	string::find(char, unsigned long) const	○	*
	string::find_first_of(const string &, unsigned long) const	○	*
	string::find_first_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_first_of(const char *, unsigned long) const	○	*
	string::find_first_of(char, unsigned long) const	○	*
	string::find_first_not_of(const string &, unsigned long) const	○	*
	string::find_first_not_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_first_not_of(const char *, unsigned long) const	○	*
	string::find_first_not_of(char, unsigned long) const	○	*
	string::find_last_of(const string &, unsigned long) const	○	*
	string::find_last_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_last_of(const char *, unsigned long) const	○	*
	string::find_last_of(char, unsigned long) const	○	*
	string::find_last_not_of(const string &, unsigned long) const	○	*
	string::find_last_not_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_last_not_of(const char *, unsigned long) const	○	*
	string::find_last_not_of(char, unsigned long) const	○	*
	string::rfind(const string &, unsigned long) const	○	*
	string::rfind(const char *, unsigned long, unsigned long) const	○	*
	string::rfind(const char *, unsigned long) const	○	*
	string::rfind(char, unsigned long) const	○	*
	string::insert(unsigned long, const string &)	○	*
	string::insert(unsigned long, const string &, unsigned long, unsigned long)	○	*
	string::insert(unsigned long, const char *, unsigned long)	○	*
	string::insert(unsigned long, const char *)	○	*
	string::insert(unsigned long, unsigned long, char)	○	*
	string::insert(char *, char)	○	*
	string::insert(char *, unsigned long, char)	○	*

	string::replace(unsigned long, unsigned long, const string &)	○	*
	string::replace(unsigned long, unsigned long, const string &, unsigned long, unsigned long)	○	*
	string::replace(unsigned long, unsigned long, const char *, unsigned long)	○	*
	string::replace(unsigned long, unsigned long, const char *)	○	*
	string::replace(unsigned long, unsigned long, unsigned long, char)	○	*
	string::replace(char *, char *, const string &)	○	*
	string::replace(char *, char *, const char *, unsigned long)	○	*
	string::replace(char *, char *, const char *)	○	*
	string::replace(char *, char *, unsigned long, char)	○	*
	string::substr(unsigned long, unsigned long) const	○	*
	ios::init(streambuf *)	○	*
	ios_base::Init::init_cnt	○	*
	string::npos	○	*
	string::_ec2p_at	○	*
	mystrbuf::open(const char *, int)	○	*
	mystrbuf::close()	○	*
	mystrbuf::setbuf(char *, long)	○	*
	mystrbuf::seekoff(long, int, int)	○	*
	mystrbuf::seekpos(long, int)	○	*
	mystrbuf::sync()	○	*
	mystrbuf::_$showmanyc()	○	*
	mystrbuf::underflow()	○	*
	mystrbuf::pbackfail(long)	○	*
	mystrbuf::overflow(long)	○	*
	mystrbuf::_Init(_f_type *)	○	*
	mystrbuf::open(const char *, int)::_$wkmode	○	*
	mystrbuf::open(const char *, int)::_$mddat	○	*
	streambuf::sbumpc()	○	*
	streambuf::sgetc()	○	*
	streambuf::sputbackc(char)	○	*
	streambuf::sungetc()	○	*
	streambuf::sputc(char)	○	*
	streambuf::streambuf()	○	*
	streambuf::gbump(int)	○	*
	streambuf::pbump(int)	○	*
	streambuf::_$setbuf(char *, long)	○	*
	streambuf::_$seekoff(long, int, int)	○	*
	streambuf::_$seekpos(long, int)	○	*
	streambuf::_$sync()	○	*
	streambuf::_$showmanyc()	○	*
	streambuf::xsgetn(char *, long)	○	*
	streambuf::_$underflow()	○	*
	streambuf::uflow()	○	*
	streambuf::_$pbackfail(long)	○	*
	streambuf::xsputn(const char *, long)	○	*
	streambuf::_$overflow(long)	○	*
	streambuf::_ec2p_setbPtr(char **, char **, long *, long *, char *)	○	*
	ios_base::boolalpha	○	*
	ios_base::dec	○	*
	ios_base::fixed	○	*
	ios_base::hex	○	*

	ios_base::internal	○	*
	ios_base::left	○	*
	ios_base::oct	○	*
	ios_base::right	○	*
	ios_base::scientific	○	*
	ios_base::showbase	○	*
	ios_base::showpoint	○	*
	ios_base::showpos	○	*
	ios_base::skipws	○	*
	ios_base::unitbuf	○	*
	ios_base::uppercase	○	*
	ios_base::adjustfield	○	*
	ios_base::basefield	○	*
	ios_base::floatfield	○	*
	ios_base::badbit	○	*
	ios_base::eofbit	○	*
	ios_base::failbit	○	*
	ios_base::goodbit	○	*
	ios_base::app	○	*
	ios_base::ate	○	*
	ios_base::binary	○	*
	ios_base::in	○	*
	ios_base::out	○	*
	ios_base::trunc	○	*
	ios_base::beg	○	*
	ios_base::cur	○	*
	ios_base::end	○	*
	ios_base::_fmtmask	○	*
	ios_base::_statemask	○	*
new	void* operator new(size_t size)		*
	void* operator new[ ](size_t size)		*
	void* operator new(size_t size, void* ptr)	○	*
	void* operator new[ ](size_t size, void* ptr)	○	*
	void operator delete(void* ptr)		*
	void operator delete[ ](void* ptr)		*
	new_handler set_new_handler(new_handler new_P)		*

ホームページとサポート窓口<website and support,ws>

ルネサステクノロジホームページ

http://japan.renesas.com/

お問合せ先

http://japan.renesas.com/inquiry

csc@renesas.com

改訂記録<revision history,rh>

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2007.9.1	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

© 2007 Renesas Technology Corp., All rights reserved.