

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事用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にご使用ください。お客様にかかる法令を遵守しないことにより生じた損害に関し、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。

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ (<http://www.renesas.com>)

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

ご注意

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、正確を期するため、慎重に制作したものです。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を負いません。
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。

SuperH RISC engine EC++クラスライブラリ

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

注意事項

- 1.本資料に記載された製品および製品の仕様は、信頼性、機能、設計の改良の理由により予告なく変更されることがあります。
- 2.本資料の一部または全部を当社に無断で転載または複製することを堅くお断りします。
- 3.本資料に記載された情報は、正確かつ信頼し得るものであります。ただし、これら記載された情報、製品または回路の使用に起因する損害または特許権その他権利の侵害に関しては、(株)日立製作所は一切その責任を負いません。
- 4.本資料によって第三者または(株)日立製作所の特許権その他権利の実施権を許諾するものではありません。
- 5.日立製半導体は、特に医療機器用として推奨できる製品を用意しておりません。しかしながら、医療機器用として日立製半導体をお考えのお客様は、当社営業窓口へ是非ご相談頂きますようお願いいたします。

はじめに

本ライブラリを使用するためには、以下に示すプログラム開発ツールが必要になります。

- SuperH RISC engine C/C++コンパイラパッケージ Ver.5.1

本ライブラリを使用する前に本マニュアルをよく読んで理解してください。また、関連マニュアルもよく読んで理解してください。

目次

第 1 章 EC++クラスライブラリ

1.1	ライブラリの概要.....	1
1.2	ストリーム入出力用クラスライブラリ	2
1.2.1	ios_base::Init クラス	3
1.2.2	ios_base クラス.....	4
1.2.3	ios クラス.....	9
1.2.4	ios クラスマニピュレータ	13
1.2.5	streambuf クラス	18
1.2.6	istream::sentry クラス.....	28
1.2.7	istream クラス.....	29
1.2.8	istream クラスマニピュレータ	38
1.2.9	istream メンバ外関数.....	39
1.2.10	ostream::sentry クラス	40
1.2.11	ostream クラス.....	41
1.2.12	ostream クラスマニピュレータ	45
1.2.13	ostream メンバ外関数	46
1.2.14	smanip クラスマニピュレータ	47
1.2.15	EC++入出力ライブラリの使用例	49
1.3	メモリ管理用ライブラリ	51
1.4	複素数計算用クラスライブラリ	53
1.4.1	float_complex クラス	53
1.4.2	float_complex メンバ外関数.....	57
1.4.3	double_complex クラス	65
1.4.4	double_complex メンバ外関数	69
1.5	文字列操作用クラスライブラリ	77
1.5.1	string クラス	77
1.5.2	string クラスマニピュレータ	97
	索引.....	103

1. EC++クラスライブラリ

1.1 ライブラリの概要

本章では、C++プログラムから標準的に利用できる組み込み向け C++クラスライブラリの仕様について説明します。「1.1 ライブラリの概要」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。以下の節では、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。

(1) ライブラリの種類

表 1.1 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 1.1 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

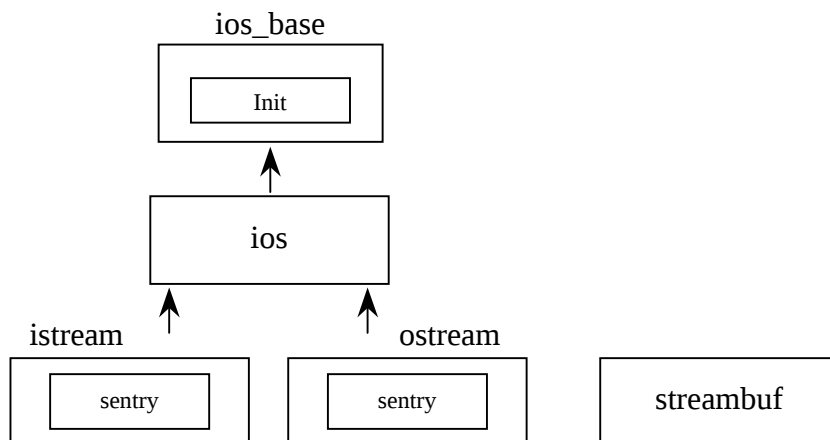
No	ライブラリの種類	内容	標準インクルードファイル
1	ストリーム入出力用 クラスライブラリ	入出力操作を行うライブラリです。	<ios>, <streambuf>, <istream>,<ostream>, <iostream>,<iomanip>
2	メモリ操作用 ライブラリ	メモリの確保・解放を行うライブラリです。	<new>
3	複素数計算用 クラスライブラリ	複素数データ演算を行うライブラリです。	<complex>
4	文字列操作用 クラスライブラリ	文字列操作を行うライブラリです。	<string>

1.2 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。

- (1) `<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。
- (2) `<streambuf>`
ストリームバッファに対する関数を定義します。
- (3) `<istream>`
入力ストリームからの入力関数を定義します。
- (4) `<ostream>`
出力ストリームへの出力関数を定義します。
- (5) `<iostream>`
入出力関数を定義します。
- (6) `<iomanip>`
引数を持つマニピュレータを定義します。

また、これらのクラスの階層図は次のようになります。



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マクロ名を示します。

■ 定義名一覧

定義名	種類	説明
streamoff	マクロ	long 型で定義された型です。
streamsize	マクロ	size_t 型で定義された型です。
int_type	マクロ	int 型で定義された型です。
pos_type	マクロ	long 型で定義された型です。
off_type	マクロ	long 型で定義された型です。

1.2.1 ios_base::Init クラス

■ 定義名一覧

定義名	種類	説明
init_cnt	データ	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。
Init ()	関数	コンストラクタです。
~ Init ()	関数	デストラクタです。

(1) ios_base:: Init::Init()

■ 機能

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。

■ リターン値

なし

(2) ios_base:: Init::~ Init ()

■ 機能

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。

■ リターン値

なし

1. EC++クラスライブラリ

1.2.2 ios_base クラス

■ 定義名一覧

定義名	種類	説明
fmtflags	マクロ	フォーマット制御情報を表す型です。
iostate	マクロ	ストリームバッファの入出力状態を表す型です。
openmode	マクロ	ファイルのオープンモードを表す型です。
seekdir	マクロ	ストリームバッファのシーク状態を表す型です。
fmtfl	データ	書式フラグです。
wide	データ	出力データの桁です。
sb	データ	streambuf オブジェクトへのポインタです。
prec	データ	出力時の精度(小数点以下の桁数)です。
fillch	データ	詰め文字です。
tiestr	データ	出力列へのポインタです。
void _ec2p_init_base()	関数	初期化します。
void _ec2p_copy_base(ios_base& ios_base_dt)	関数	ios_base_dt をコピーします。
ios_base()	関数	コンストラクタです。
~ios_base()	関数	デストラクタです。
fmtflags flags() const	関数	書式フラグ(fmtfl)を参照します。
fmtflags flags(fmtflags fmtflg)	関数	書式フラグ(fmtfl)と fmtflg の論理積を書式フラグ(fmtfl)に設定します。
fmtflags setf(fmtflags fmtflg)	関数	fmtflg を書式フラグ(fmtfl)に設定します。
fmtflags setf(fmtflags fmtflg, fmtflags mask)	関数	mask&fmtflg を書式フラグ(fmtfl)に設定します。
void unsetf(fmtflags mask)	関数	~mask&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。
char fill() const	関数	詰め文字(fillch)を参照します。
char fill(char ch)	関数	ch を詰め文字(fillch)に設定します。
int precision() const	関数	精度(prec)を参照します。
streamsize precision(streamsize preci)	関数	preci を精度(prec)に設定します。
streamsize width() const	関数	幅値(wide)を参照します。
streamsize width(streamsize wd)	関数	wd を幅値(wide)に設定します。

(1) 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;
```

(2) ios_base::iostate

ストリームバッファの入出力状態を定義します。

state 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;
```

(3) 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	= 0x1;	入力用のファイルを open します。
const ios_base::openmode ios_base::out	= 0x2;	出力用のファイルを open します。
const ios_base::openmode ios_base::ate	= 0x4;	オープン後一度だけ eof に seek します。
const ios_base::openmode ios_base::app	= 0x8;	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。

(4) ios_base::seekdir

ストリームバッファの seek 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列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;

(5) void _ec2p_init_base()

■機能

以下の値で初期設定します。

```
fmtfl = skipws | dec;  
wide = 0;  
prec = 6;  
fillch = ' ';
```

■リターン値

なし

(6) void _ec2p_copy_base(ios_base & ios_base_dt)

■機能

ios_base_dt をコピーします。

■リターン値

なし

(7) ios_base::ios_base()

■機能

クラス ios_base のコンストラクタです。

Init::Init()を呼び出します。

■リターン値

なし

(8) `ios_base::~~ios_base()`

■機能

クラス `ios_base` のデストラクタです。

■リターン値

なし

(9) `ios_base::fmtflags ios_base::flags() const`

■機能

書式フラグ(`fmtfl`)を参照します。

■リターン値

書式フラグ(`fmtfl`)

(10) `ios_base::fmtflags ios_base::flags(fmtflags fmtflg)`

■機能

`fmtflg`と書式フラグ(`fmtfl`)を書式フラグ(`fmtfl`)に設定します。

■リターン値

設定前の書式フラグ(`fmtfl`)

(11) `ios_base::fmtflags ios_base::setf(fmtflags fmtflg)`

■機能

`fmtflg` を書式フラグ(`fmtfl`)に設定します。

■リターン値

設定前の書式フラグ(`fmtfl`)

(12) `ios_base::fmtflags ios_base::setf(fmtflags fmtflg, fmtflags mask)`

■機能

`mask&fmtflg` の値を書式フラグ(`fmtfl`)に設定します。

■リターン値

設定前の書式フラグ(`fmtfl`)

(13) `void ios_base::unsetf(fmtflags mask)`

■機能

`~mask`と書式フラグ(`fmtfl`)を書式フラグ(`fmtfl`)に設定します。

■リターン値

なし

(14) `char ios_base::fill() const`

■ 機能

詰め文字(fillch)を参照します。

■ リターン値

詰め文字(fillch)

(15) `char ios_base::fill(char ch)`

■ 機能

ch を詰め文字として設定します。

■ リターン値

設定前の詰め文字(fillch)

(16) `int ios_base::precision() const`

■ 機能

精度(prec)を参照します。

■ リターン値

精度(prec)

(17) `streamsize ios_base::precision( streamsize preci)`

■ 機能

preci を精度(prec)に設定します。

■ リターン値

設定前の精度(prec)

(18) `streamsize ios_base::width() const`

■ 機能

フィールドの幅(wide)を参照します。

■ リターン値

幅(wide)

(19) `streamsize ios_base::width(streamsize wd)`

■ 機能

wd をフィールド幅(wide)に設定します。

■ リターン値

設定前の幅(wide)

1.2.3 ios クラス

■ 定義名一覧

定義名	種類	説明
sb	データ	streambuf へのポインタです。
tiestr	データ	ostream へのポインタです。
state	データ	streambuf の状態フラグです。
ios()	関数	コンストラクタです。
ios(streambuf *sbptr)		
void init(streambuf *sbptr)	関数	初期設定を行います。
virtual ~ios()	関数	デストラクタです。
operator void*() const	関数	エラー有無(!state&(badbit failbit))を判定します。
bool operator !() const	関数	エラー有無(state&(badbit failbit))を判定します。
iosstate rdstate() const	関数	状態フラグ(state)を参照します。
void clear(iosstate st=goodbit)	関数	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(iosstate st)	関数	st を状態フラグ(state)に設定します。
bool good() const	関数	エラー有無(state==goodbit)を判定します。
bool eof() const	関数	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	関数	エラー有無(state&badbit)を判定します。
bool fail() const	関数	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	関数	ostream オブジェクトへのポインタ(tiestr)を参照します。
ostream* tie(ostream* tstrptr)	関数	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	関数	ストリームバッファへのポインタ(sb)を参照します。
streambuf* rdbuf(streambuf* sbptr)	関数	sbptr をストリームバッファへのポインタ(sb)に設定します。
ios & copyfmt(const ios & rhs)	関数	rhs の状態フラグ(state)をコピーします。

(1) ios::ios()

■ 機能

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。

■ リターン値

なし

(2) `ios::ios(streambuf *sbptr)`

■機能

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。

■リターン値

なし

(3) `void ios::init(streambuf *sbptr)`

■機能

`sbptr` を `sb` に設定します。

`state`、`tiestr` を 0 に設定します。

■リターン値

なし

(4) `virtual ios::~~ios()`

■機能

クラス `ios` のデストラクタです。

■リターン値

なし

(5) `bool ios::operator void*() const`

■機能

エラー有無(`!state & (badbit|failbit)`)を判定します。

■リターン値

エラー有の場合：FALSE

エラー無の場合：TRUE

(6) `bool ios::operator!() const`

■機能

エラー有無(`state & (badbit|failbit)`)を判定します。

■リターン値

エラー有の場合：TRUE

エラー無の場合：FALSE

(7) `iosstate ios::rdstate() const`

■機能

状態フラグ(`state`) を参照します。

■ リターン値

状態フラグ(state)

(8) void ios::clear(iostate st=goodbit)

■ 機能

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 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。

■ リターン値

なし

(9) void ios::setstate(iostate st)

■ 機能

st を状態フラグ(state)に設定します。

■ リターン値

なし

(10) bool ios::good() const

■ 機能

エラー有無(state==goodbit)を判定します。

■ リターン値

エラー有の場合 : FALSE

エラー無の場合 : TRUE

(11) bool ios::eof() const

■ 機能

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

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

(12) bool ios::bad() const

■ 機能

エラー有無(state&badbit)を判定します。

■ リターン値

エラー有の場合 : TRUE

エラー無の場合 : FALSE

(13) `bool ios::fail() const`

■ 機能

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

不一致の場合 : `TRUE`

一致の場合 : `FALSE`

(14) `ostream* ios::tie() const`

■ 機能

`ostream` オブジェクトポインタ(`tiestr`)を参照します。

■ リターン値

オブジェクトポインタ(`tiestr`)

(15) `ostream* ios::tie(ostream* tstrptr)`

■ 機能

`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

設定前のオブジェクトポインタ(`tiestr`)

(16) `streambuf* ios::rdbuf() const`

■ 機能

`streambuf` へのポインタ(`sb`) を参照します。

■ リターン値

`streambuf` へのポインタ(`sb`)

(17) `streambuf* ios::rdbuf( streambuf* sbptr)`

■ 機能

`sbptr` をストリームバッファへのポインタ(`sb`)に設定します。

■ リターン値

設定前の `streambuf` へのポインタ(`sb`)

(18) `ios & ios::copyfmt( const ios & rhs)`

■ 機能

`rhs` の状態フラグ(`state`)をコピーします。

■ リターン値

`*this`

1.2.4 ios クラスマニピュレータ

■ 定義名一覧

マニピュレータ	説明
<code>ios_base& boolalpha(ios_base& str)</code>	bool 型の書式に設定します。
<code>ios_base& noboolalpha(ios_base& str)</code>	bool 型の書式をクリアします。
<code>ios_base& showbase(ios_base& str)</code>	基数表示接頭辞モードに設定します。
<code>ios_base& noshowbase(ios_base& str)</code>	基数表示接頭辞モードをクリアします。
<code>ios_base& showpoint(ios_base& str)</code>	小数点生成モードに設定します。
<code>ios_base& noshowpoint(ios_base& str)</code>	小数点生成モードをクリアします。
<code>ios_base& showpos(ios_base& str)</code>	+記号生成モードに設定します。
<code>ios_base& noshowpos(ios_base& str)</code>	+記号生成モードをクリアします。
<code>ios_base& skipws(ios_base& str)</code>	空白読み飛ばしモードに設定します。
<code>ios_base& noskipws(ios_base& str)</code>	空白読み飛ばしモードをクリアします。
<code>ios_base& uppercase(ios_base& str)</code>	大文字変換モードに設定します。
<code>ios_base& nouppercase(ios_base& str)</code>	大文字変換モードをクリアします。
<code>ios_base& internal(ios_base& str)</code>	内部補充モードに設定します。
<code>ios_base& left(ios_base& str)</code>	左側補充モードに設定します。
<code>ios_base& right(ios_base& str)</code>	右側補充モードに設定します。
<code>ios_base& dec(ios_base& str)</code>	10 進モードに設定します。
<code>ios_base& hex(ios_base& str)</code>	16 進モードに設定します。
<code>ios_base& oct(ios_base& str)</code>	8 進モードに設定します。
<code>ios_base& fixed(ios_base& str)</code>	固定小数点モードに設定します。
<code>ios_base& scientific(ios_base& str)</code>	科学表記法モードに設定します。

(1) `ios_base& boolalpha(ios_base& str)`

■ 機能

`bool` 型の書式に設定します。

■ リターン値

`str`

(2) `ios_base& noboolalpha(ios_base& str)`

■ 機能

`bool` 型の書式をクリアします。

■ リターン値

`str`

(3) `ios_base& showbase(ios_base& str)`

■ 機能

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

`str`

(4) `ios_base& noshowbase(ios_base &str)`

■ 機能

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

`str`

(5) `ios_base& showpoint( ios_base & str)`

■ 機能

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

`str`

(6) `ios_base& noshowpoint( ios_base& str)`

■ 機能

小数点を出力するモードをクリアします。

■ リターン値

`str`

(7) `ios_base& showpos( ios_base& str)`

■ 機能

+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

`str`

(8) `ios_base& noshowpos( ios_base & str)`

■ 機能

+記号生成出力モードをクリアします。

■ リターン値

`str`

(9) `ios_base& skipws( ios_base& str)`

■ 機能

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

`str`

(10) `ios_base& noskipws( ios_base& str)`

■ 機能

空白読み飛ばし入力モードをクリアします。

■ リターン値

`str`

(11) `ios_base& uppercase( ios_base& str)`

■ 機能

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

`str`

(12) `ios_base nouppercase( ios_base & str)`

■ 機能

大文字変換出力モードをクリアします。

■ リターン値

`str`

(13) `ios_base& internal( ios_base & str)`

■機能

フィールド幅(wide)の範囲で出力時に

- 1 . 符号、基数
- 2 . 詰め文字(fill)
- 3 . 数値

の順で出力します。

■リターン値

`str`

(14) `ios_base& left( ios_base & str)`

■機能

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

`str`

(15) `ios_base& right( ios_base & str)`

■機能

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

`str`

(16) `ios_base& dec( ios_base & str)`

■機能

変換基数を 10 進モードに設定します。

■リターン値

`str`

(17) `ios_base& hex( ios_base & str)`

■機能

変換基数を 16 進モードに設定します。

■リターン値

`str`

(18) `ios_base& oct( ios_base & str)`

■機能

変換基数を 8 進モードに設定します。

■ リターン値

str

(19) ios_base& fixed(ios_base & str)

■ 機能

固定小数点出力モードに設定します。

■ リターン値

str

(20) ios_base& scientific(ios_base & str)

■ 機能

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

str

1.2.5 streambuf クラス

■ 定義名一覧

定義名	種類	説明
_B_cnt_ptr	データ	バッファの有効データ長へのポインタです。
B_beg_ptr	データ	バッファのベースポインタへのポインタです。
_B_len_ptr	データ	バッファの長さへのポインタです。
B_next_ptr	データ	バッファの次の読み出し位置へのポインタです。
B_end_ptr	データ	バッファの終端位置へのポインタです。
B_beg_pptr	データ	制御バッファの先頭位置へのポインタです。
B_next_pptr	データ	バッファの次の読み出し位置へのポインタです。
C_flg_ptr	データ	ファイルの入出力制御フラグへのポインタです。
char* _ec2p_getflag() const	関数	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()	関数	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()	関数	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()	関数	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()	関数	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)	関数	streambuf のポインタをセットします。
streambuf()	関数	コンストラクタです。
virtual ~streambuf()	関数	デストラクタです。
streambuf* pubsetbuf(char* s, streamsize n)	関数	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)	関数	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)	関数	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()	関数	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。

定義名	種類	説明
streamsize in_avail()	関数	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。
int_type snextc()	関数	次の一文字を読み込みます。
int_type sbumpc()	関数	一文字読み込みポインタを次に設定します。
int_type sgetc()	関数	一文字読み込みます。
int sgetn(char* s, streamsize n)	関数	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)	関数	読み込み位置をブットバックします。
int sungetc()	関数	読み込み位置をブットバックします。
int sputc(char c)	関数	文字 c を挿入します。
int_type sputn(const char* s, streamsize n)	関数	s の指す n 個の文字を挿入します。
char* eback() const	関数	入力列の先頭ポインタを求めます。
char* gptr() const	関数	入力列の次ポインタを求めます。
char* egptr() const	関数	入力列の最後尾ポインタを求めます。
void gbump(int n)	関数	入力列の次ポインタを n 進めます。
void setg(char* gbeg, char* gnext, char* gend)	関数	入力列の各ポインタを代入します。
char* pbase() const	関数	出力列の先頭ポインタを求めます。
char* pptr() const	関数	出力列の次ポインタを求めます。
char* epptr() const	関数	出力列の最後尾ポインタを求めます。
void pbump(int n)	関数	出力列の次ポインタを n 進めます。
void setp(char* pbeg, char* pend)	関数	出力列の各ポインタを設定します。
virtual streambuf *setbuf(char* s, streamsize n)	関数 ^{*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))	関数 ^{*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))	関数 ^{*1}	ストリーム位置を変更します。

1. EC++クラスライブラリ

定義名	種類	説明
virtual int sync()	関数 ^{*1}	出力ストリームをフラッシュします。
virtual int showmanyc()	関数 ^{*1}	入力列の有効な文字数を求めます。
virtual streamsize xsgetn(char* s, streamsize n)	関数	s の指す記憶領域に n 個の文字を設定します。
virtual int_type underflow()	関数 ^{*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()	関数 ^{*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)	関数 ^{*1}	c によって示される文字をプットバックします。
virtual streamsize xspn(const char* s, streamsize n)	関数	s の指す n 個の文字を挿入します。
virtual int_type overflow(int_type c=eof)	関数 ^{*1}	c を出力列に挿入します。

注 *1 このクラスでは処理を定義していません。

(1) streambuf::streambuf()

■ 機能

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 = _Blen_ptr = 0

B_beg_pptr = &B_beg_ptr

B_next_pptr = &B_next_ptr

■ リターン値

なし

(2) virtual streambuf::~~streambuf()

■ 機能

デストラクタです。

■ リターン値

なし

(3) streambuf* streambuf::pubsetbuf(char* s, streamsize n)

■ 機能

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

setbuf(s,n)

- (4) `pos_type streambuf::pubseekoff(off_type off, ios_base::seekdir way, ios_base::openmode which=(ios_base::openmode)(ios_base::in|ios_base::out))`

■ 機能

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

新たに設定されたストリームの位置

- (5) `pos_type streambuf::pubseekpos(pos_type sp, ios_base::openmode which=(ios_base::openmode)(ios_base::in | ios_base::out))`

■ 機能

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

先頭からのオフセット

- (6) `int streambuf::pubsync()`

■ 機能

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

0

- (7) `streamsize streambuf::in_avail()`

■ 機能

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

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
読み込み位置が有効でない場合 : 0(showmanyc()を呼び出します)

- (8) `int_type streambuf::snextc()`

■ 機能

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

EOF でない場合 : 読み込んだ文字
EOF の場合 : EOF

(9) int_type streambuf::sbumpc()

■機能

一文字読み込みポインタを次に設定します。

■リターン値

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

(10) int_type streambuf::sgetc()

■機能

一文字読み込みます。

■リターン値

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

(11) int streambuf::sgetn(char* s, streamsize n)

■機能

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

設定した文字数

(12) int_type streambuf::sputback(char c);

■機能

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

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

(13) int streambuf::sungetc()

■機能

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

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

(14) int streambuf::sputc(char c)

■機能

文字 c を挿入します。

- リターン値

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

(15) `int_type streambuf::sputn( const char* s, streamsize n)`

- 機能

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

挿入された文字数

(16) `char* streambuf::eback() const`

- 機能

入力ストリームの先頭ポインタを求めます。

- リターン値

先頭ポインタ

(17) `char* streambuf::gptr() const`

- 機能

入力ストリームの次ポインタを求めます。

- リターン値

次ポインタ

(18) `char* streambuf::egptr() const`

- 機能

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

最後尾ポインタ

(19) `void streambuf::gbump(int n)`

- 機能

入力ストリームの次ポインタを n 進めます。

- リターン値

なし

(20) void streambuf::setg(char* gbeg, char* gnext, char* gend)

■ 機能

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;
```

■ リターン値

なし

(21) char* streambuf::pbase() const

■ 機能

出力ストリームの先頭ポインタを求めます。

■ リターン値

先頭ポインタ

(22) char* streambuf::pptr() const

■ 機能

出力ストリームの次ポインタを求めます。

■ リターン値

次ポインタ

(23) char* streambuf::epptr() const

■ 機能

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

最後尾ポインタ

(24) void streambuf::pbump(int n)

■ 機能

出力ストリームの次ポインタを *n* 進めます。

■ リターン値

なし

(25) void streambuf::setp(char* pbeg, char* pend)

■ 機能

出力列の各ポインタに、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;

■ リターン値

なし

(26) virtual streambuf* streambuf::setbuf(char* s, streamsize n)

■ 機能

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

*this (このクラスでは処理を定義していません)

(27) virtual pos_type streambuf::seekoff(off_type off, ios_base::seekdir way ,
ios_base::openmode=(ios_base::openmode)(ios_base::in | ios_base::out))

■ 機能

ストリーム位置を変更します。

■ リターン値

(-1) (このクラスでは処理を定義していません)

(28) virtual pos_type streambuf::seekpos(pos_type off,
ios_base::openmode=(ios_base::openmode)(ios_base::in | ios_base::out))

■ 機能

ストリーム位置を変更します。

■ リターン値

(-1) (このクラスでは処理を定義していません)

(29) virtual int streambuf::sync()

■ 機能

出力ストリームをフラッシュします。

■ リターン値

0 (このクラスでは処理を定義していません)

(30) virtual int streambuf::showmanyc()

■機能

入力ストリームの有効な文字数を求めます。

■リターン値

0 (このクラスでは処理を定義していません)

(31) virtual streamsize streambuf::xsgetn(char* s, streamsize n)

■機能

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

入力された文字数

(32) virtual int_type streambuf::underflow()

■機能

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

EOF (このクラスでは処理を定義していません)

(33) virtual int_type streambuf::uflow()

■機能

次ポインタの一文字を読み込みます。

■リターン値

EOF (このクラスでは処理を定義していません)

(34) virtual int_type streambuf::pbackfail(int_type c=eof)

■機能

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

EOF (このクラスでは処理を定義していません)

(35) virtual streamsize streambuf::xsputn(const char* s, streamsize n)

■機能

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

挿入された文字数

(36) virtual int_type streambuf::overflow(int_type c=eof)

■ 機能

c を出力ストリームに挿入します。

■ リターン値

EOF (このクラスでは処理を定義していません)

1.2.6 istream::sentry クラス

■ 定義名一覧

定義名	種類	説明
ok_	データ	入力可能状態か否かを意味します。
sentry(istream& is, bool noskipws= _FALSE)	関数	コンストラクタです。
~sentry()	関数	デストラクタです。
operator bool()	関数	ok_を参照します。

(1) istream::sentry::sentry(istream&is, bool noskipws= _FALSE)

■ 機能

内部クラス sentry のコンストラクタです。

■ リターン値

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、出力列と関連する外部 C ストリームの同期をあわせませす。

(2) istream::sentry::~sentry()

■ 機能

内部クラス sentry のデストラクタです。

■ リターン値

なし

(3) istream::sentry::operator bool()

■ 機能

ok_を参照します。

■ リターン値

ok_

1.2.7 istream クラス

■ 定義名一覧

定義名	種類	説明
chcount	データ	最後にコールされた入力関数が抽出した文字数です。
int::_ec2p_getistr(char* str, unsigned int dig, int mode)	関数	str を dig が示す基数で変換します。
istream(streambuf *sb)	関数	コンストラクタです。
~istream()	関数	デストラクタです。
istream& operator >>(bool &n)	関数	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 >>(float &n)		
istream& operator >>(double &n)		
istream& operator >>(long double &n)		
istream& operator >>(void* &p)	関数	void を指すポインタに変換して p に格納します。
istream& operator >>(streambuf *sb)	関数	文字を抽出し、sb の指す記憶領域へ格納します。
streamsize gcount() const	関数	chcount(抽出文字数)を求めます。
int_type get()	関数	文字を抽出します。
istream& get(char &c)	関数	文字を抽出し c に格納します。
istream& get(signed char &c)		
istream& get(unsigned char &c)		

1. EC++クラスライブラリ

定義名	種類	説明
istream& get(char* s, streamsize n)	関数	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)	関数	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)		
istream& get(streambuf &sb)	関数	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)	関数	文字列を抽出し、sb の指す記憶領域に格納します。途中文字'delim'を検出したら、入力を終了します。
istream& getline(char* s, streamsize n)	関数	サイズ n-1 の文字列を抽出し、sb 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)	関数	サイズ n-1 の文字列を抽出し、sb の指す記憶領域に格納します。 途中文字'delim'を検出したら、入力を終了します。
istream& getline(signed char* s, streamsize n, char delim)		

定義名	種類	説明
<code>istream& getline(unsigned char* s, streamsize n, char delim)</code>	関数	サイズ n-1 の文字列を抽出し、sb の指す記憶領域に格納します。 途中文字'delim'を検出したら、入力を終了します。
<code>istream& ignore(unsigned streamsize n=1, int_type delim=streambuf::eof)</code>	関数	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。
<code>int_type peek()</code>	関数	次の入手可能な入力文字を求めます。
<code>istream& read(char* s, streamsize n)</code>	関数	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。
<code>istream& read(signed char* s, streamsize n)</code>		
<code>istream& read(unsigned char* s, streamsize n)</code>		
<code>streamsize readsome(char* s, streamsize n)</code>	関数	n 個の文字列を抽出し、s の指す記憶領域に格納します。
<code>streamsize readsome(signed char* s, streamsize n)</code>		
<code>streamsize readsome(unsigned char* s, streamsize n)</code>		
<code>istream& putback(char c)</code>	関数	文字を入力ストリームに戻します。
<code>istream& unget()</code>	関数	入力ストリームの位置を戻します。
<code>int sync()</code>	関数	ストリームがあるかどうかを調べます。 この関数は <code>streambuf::pubsync()</code> を呼び出します。
<code>pos_type tellg()</code>	関数	入力ストリームの位置を調べます。 この関数は <code>streambuf::pubseekoff(0,cur,in)</code> を呼び出します。
<code>istream& seekg(pos_type & pos)</code>	関数	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 <code>streambuf::pubseekpos(pos)</code> を呼び出します。
<code>istream& seekg(off_type & off, ios_base::seekdir dir)</code>	関数	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 <code>stream::pubseekoff(off,dir)</code> を呼び出します。

(1) `int istream::_ec2p_getistr( char* str, unsigned int dig, int mode)`

■ 機能

`str` を `dig` が示す基数で変換します。

■ リターン値

変換した基数を返します。

(2) `istream::istream( streambuf *sb)`

■ 機能

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。

■ リターン値

なし

(3) `virtual istream::~istream()`

■ 機能

クラス `istream` のデストラクタです。

■ リターン値

なし

(4) `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 >> ( float &n)`

`istream& istream::operator >> ( double &n)`

`istream& istream::operator >> ( long double &n)`

■ 機能

抽出した文字を `n` に格納します。

■ リターン値

`*this`

(5) `istream& istream::operator >> ( void * &p)`

■ 機能

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

`*this`

(6) `istream& istream::operator >> ( streambuf *sb)`

■ 機能

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

`*this`

(7) `streamsize istream::gcount() const`

■ 機能

`chcount`(抽出文字数)を参照します。

■ リターン値

`chcount`

(8) `int_type istream::get()`

■ 機能

文字を抽出します。

■ リターン値

抽出可能の場合：抽出した文字

抽出不可の場合：`setstate(failbit)`呼び出し、EOF

(9) `istream& istream::get(char &c)`

l `stream& istream::get(signed char &c)`

`istream& istream::get(unsigned char &c)`

■ 機能

文字を抽出し `c` に格納します。抽出した文字が EOF の場合は、`failbit` を設定します。

■ リターン値

`*this`

(10) `istream& istream::get( char* s, streamsize n)`

`istream& istream::get( signed char* s, streamsize n)`

`istream& istream::get( unsigned char* s, streamsize n)`

■ 機能

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

`*this`

(11) `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)`

■ 機能

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

`*this`

(12) `istream& istream::get( streambuf&sb, char delim)`

■ 機能

文字列を抽出し、`sb` の指す記憶領域に格納します。

途中文字 '`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

`*this`

(13) `istream& istream::get( streambuf&sb)`

■ 機能

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

`*this`

(14) `istream& istream::getline( char* s, streamsize n)`

`istream& istream::getline( signed char* s, streamsize n)`

`istream& istream::getline( unsigned char* s, streamsize n)`

■機能

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

`*this`

(15) `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)`

■機能

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

`*this`

(16) `istream& istream::ignore( streamsize n = 1, int_type delim = streambuf::eof)`

■機能

n 個の文字を読み飛ばします。

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

`*this`

(17) `int_type istream::peek()`

■機能

次の入力可能な入力文字を求めます。

■リターン値

`ok_==FALSE` の場合、`EOF`

`ok_!=FALSE` の場合、`rdbuf()->sgetc()`

(18) `istream& istream::read( char* s, streamsize n)`

`istream& istream::read( signed char* s, streamsize n)`

`istream& istream::read( unsigned char* s, streamsize n)`

■機能

`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

*this

- (19) streamsize istream::readsome(char* s, streamsize n)
streamsize istream::readsome(signed char* s, streamsize n)
streamsize istream::readsome(unsigned char* s, streamsize n)

■ 機能

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

抽出した文字数

- (20) istream& istream::putback(char c)

■ 機能

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 EOF の場合は、badbit を設定します。

■ リターン値

*this

- (21) istream& istream::unget()

■ 機能

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 EOF の場合、badbit を設定します。

■ リターン値

*this

- (22) int istream::sync()

■ 機能

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

入力ストリームがない場合 : EOF
入力ストリームがある場合 : 0

- (23) pos_type istream::tellg()

■ 機能

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

ストリームの先頭からのオフセット
入力処理にエラーが発生した場合、-1

(24) `istream& istream::seekg( pos_type & pos)`

■ 機能

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

`*this`

(25) `istream& istream::seekg( off_type & off, ios_base::seekdir dir)`

■ 機能

`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

`*this`

1.2.8 istream クラスマニピュレータ

■ 定義名一覧

定義名	説明
istream& ws(istream &is)	空白文字を読み飛ばします。

(1) istream& ws(istream &is)

■ 機能

空白類を読み飛ばします。

■ リターン値

is

1.2.9 istream メンバ外関数

■ 定義名一覧

定義名	種類	説明
istream& operator >> (istream& in, char* s)	関数	文字を抽出し、s で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 &s)		
istream& operator >> (istream& in, signed char &s)		
istream& operator >> (istream& in, unsigned char &s)		

- (1) istream& operator >>(istream& in, char* s)
 istream& operator >>(istream& in, signed char* s)
 istream& operator >>(istream& in, unsigned char* s)
 istream& operator>> (istream& in, char &s)
 istream& operator>> (istream& in, signed char &s)
 istream& operator>> (istream& in, unsigned char &s)

■ 機能

文字を抽出し、s で指す記憶領域に格納します。

n-1 個の文字を格納したか、または入力列に 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

in

1.2.10 ostream::sentry クラス

■ 定義名一覧

定義名	種類	説明
ok_	データ	出力可能状態が否かを意味します。
__ec2p_os	データ	ostream オブジェクトへのポインタです。
sentry(ostream &os)	関数	コンストラクタです。
~sentry()	関数	デストラクタです。
operator bool()	関数	ok_ を参照します。

(1) ostream::sentry::sentry(ostream &os)

■ 機能

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。

■ リターン値

なし

(2) ostream::sentry::~sentry()

■ 機能

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。

■ リターン値

なし

(3) ostream::sentry::operator bool()

■ 機能

ok_ を参照します。

■ リターン値

ok_

1.2.11 ostream クラス

■ 定義名一覧

定義名	種類	説明
ostream(streambuf *sbptr)	関数	コンストラクタです
~ostream()	関数	デストラクタです
ostream & operator << (bool n)	関数	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 << (float n)		
ostream & operator << (double n)		
ostream & operator << (long double n)		
ostream & operator << (void *n)		
ostream & operator << (streambuf *sbptr)	関数	sbptr の出力列を出力ストリームに挿入します。
ostream & put(char c)	関数	文字 c を出力ストリームに挿入します。
ostream & write(const char* s, streamsize n)	関数	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()	関数	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。

1. EC++クラスライブラリ

定義名	種類	説明
pos_type tellp()	関数	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)	関数	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)	関数	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。

(1) ostream::ostream(streambuf *sbptr)

■機能

コンストラクタです。
ios (sbptr)を呼び出します。

■リターン値

なし

(2) virtual ostream::~~ostream()

■機能

デストラクタです。

■リターン値

なし

- (3) `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 << ( float n)`
`ostream & ostream::operator << ( double n)`
`ostream & ostream::operator << ( long double n)`
`ostream & ostream::operator << ( void *n)`

■ 機能

`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

`*this`

- (4) `ostream & ostream::operator << (streambuf *sbptr)`

■ 機能

`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

`*this`

- (5) `ostream & ostream::put( char c)`

■ 機能

`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

`*this`

(6) `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)`

■ 機能

`sentry::ok_==TRUE` かつ `rddbuf()->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

`*this`

(7) `ostream & ostream::flush()`

■ 機能

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

`*this`

(8) `pos_type ostream::tellp()`

■ 機能

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

現在のストリームの位置

処理中にエラーが発生した場合は -1

(9) `ostream & ostream::seekp( pos_type pos)`

■ 機能

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

`*this`

(10) `ostream & ostream::seekp( off_type off, seekdir dir)`

■ 機能

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(pos,dir)` を呼び出します。

■ リターン値

`*this`

1.2.12 ostream クラスマニピュレータ

■ 定義名一覧

定義名	説明
<code>ostream & endl(ostream &os)</code>	改行を付加し、出力ストリームをフラッシュします。
<code>ostream & ends(ostream &os)</code>	ヌルコードを付加します。
<code>ostream & flush(ostream &os)</code>	出力ストリームをフラッシュします。

(1) ostream & endl(ostream &os)

■ 機能

ストリームに改行文字を付加します。
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

os

(2) ostream & ends(ostream &os)

■ 機能

出力列にヌルコードを挿入します。

■ リターン値

os

(3) ostream & flush(ostream &os)

■ 機能

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

os

1.2.13 ostream メンバ外関数

■ 定義名一覧

定義名	種類	説明
ostream& operator << (ostream& os, char s)	関数	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 signed char* s)		
ostream& operator << (ostream& os, const unsigned char* s)		

- (1) 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 signed char* s)
ostream& operator<< (ostream& os, const unsigned char* s)

■ 機能

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

os

1.2.14 smanip クラスマニピュレータ

■ 定義名一覧

定義名	種類	説明
smanip resetiosflags(ios_base::fmtflags mask)	関数	mask 値で指定されたフラグをクリアします。
smanip setiosflags(ios_base::fmtflags mask)	関数	書式フラグ(fmtfl)の設定を行います。
smanip setbase(int base)	関数	出力時に用いる基数をセットします。
smanip setfill(char c)	関数	詰め文字(fillch)の設定を行います。
smanip setprecision(int n)	関数	精度(prec)の指定を行います。
smanip setw(int n)	関数	フィールド幅(wide)の設定を行います。

(1) smanip resetiosflags(ios_base::fmtflags mask)

■ 機能

mask 値で指定されたフラグをクリアします。

■ リターン値

入出力対象のオブジェクト

(2) smanip setiosflags(ios_base::fmtflags(0), mask)

■ 機能

書式フラグ(fmtfl)の設定を行います。

■ リターン値

入出力対象のオブジェクト

(3) smanip setbase(int base)

■ 機能

出力時に用いる基数をセットします。

■ リターン値

入出力対象のオブジェクト

(4) smanip setfill(char c)

■ 機能

詰め文字の設定を行います。

■ リターン値

入出力対象のオブジェクト

(5) `smanip setprecision( int n)`

- 機能

精度の指定を行います。

- リターン値

入出力対象のオブジェクト

(6) `smanip setw( int n)`

- 機能

フィールド幅の設定を行います。

- リターン値

入出力対象のオブジェクト

1.2.15 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。

■ 定義名一覧

定義名	種類	説明
_file_Ptr	データ	ファイルポインタです。
mystrbuf()	関数	コンストラクタです。
mystrbuf(void *ptr)		streambuf バッファの初期化を行います。
~mystrbuf()	関数	デストラクタです。
void *myfptr() const	関数	FILE 型構造体へのポインタを返します。
mystrbuf *open(const char* filename, int mode)	関数	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()	関数	ファイルのクローズを行います。
virtual streambuf *setbuf(char* s, streamsize n)	関数	-
virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode=(ios_base::openmode) (ios_base::in ios_base::out))	関数	-
virtual pos_type seekpos(pos_type sp, ios_base::openmode=(ios_base::openmode) (ios_base::in ios_base::out))	関数	ストリームポインタの位置を変えます。
virtual int sync()	関数	ストリームをフラッシュします。
virtual int showmanyc()	関数	入力列の有効な文字数を返します。
virtual int_type underflow()	関数	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)	関数	c によって示される文字をプットバックします。
virtual int_type overflow(int_type c=streambuf::eof)	関数	c によって示される文字を挿入します。
void _Init(_f_type *fp)	関数	初期処理です。

<例>

```
#include <istream>
#include <ostream>
#include <mystrbuf>
#include <string>
#include <new>

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;
}
```

1.3 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：

- ・ 割当可能な領域を作り出して返します。
- ・ そうできない場合の動作は規定されていません。

■ 定義名一覧

定義名	種類	説明
<code>new_handler</code>	マクロ	<code>void</code> 型を返す関数へのポインタ型です。
<code>_ec2p_new_handler</code>	データ	例外処理関数へのポインタです。
<code>void* operator new(size_t size)</code>	関数	<code>size</code> 分の領域を確保します。
<code>void* operator new[](size_t size)</code>	関数	<code>size</code> 分の配列領域を確保します。
<code>void* operator new(size_t size, void *ptr)</code>	関数	<code>ptr</code> の指している領域を記憶領域として割り当てます。
<code>void* operator new[](size_t size, void *ptr)</code>	関数	<code>ptr</code> の指している領域を記憶領域として割り当てます。
<code>void operator delete(void* ptr)</code>	関数	領域を解放します。
<code>void operator delete[](void* ptr)</code>	関数	配列領域を解放します。
<code>new_handler set_new_handler(new_handler new_P)</code>	関数	<code>_ec2p_new_handler</code> に例外処理関数アドレス (<code>new_P</code>) をセットします。

(1) `void* operator new( size_t size)`

■ 機能

`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` がセットされていれば、`new_handler` を呼び出します。

■ リターン値

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

(2) `void* operator new[]( size_t size)`

■ 機能

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` がセットされていれば、`new_handler` を呼び出します。

■ リターン値

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

(3) `void* operator new( size_t size, void *ptr)`

■ 機能

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

ptr

(4) `void* operator new[]( size_t size, void *ptr)`

■ 機能

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

ptr

(5) `void operator delete( void* ptr)`

■ 機能

ptr が指す記憶領域を解放します。ptr が NULL のときはなにもしません。

■ リターン値

なし

(6) `void operator delete[]( void* ptr)`

■ 機能

ptr が指す配列領域を解放します。ptr が NULL のときはなにもしません。

■ リターン値

なし

(7) `set_new_handler(new_handler new_P)`

■ 機能

_ec2p_new_handler に new_P をセットします

■ リターン値

_ec2p_new_handler の値

1.4 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通りです。

(1) <complex>

float_complex クラス、double_complex クラスを定義します。

また、これらのクラスの階層図は次のようになります。

float_complex

double_complex

1.4.1 float_complex クラス

■ 定義名一覧

定義名	種類	説明
value_type	マクロ	float 型です。
_re	データ	float 精度の実数部を定義します。
_im	データ	float 精度の虚数部を定義します。
float_complex(float re=0.0f, float im=0.0f)	関数	コンストラクタです。
float_complex(const double_complex& rhs)		
float real() const	関数	実数部(_re)を求めます。
float imag() const	関数	虚数部(_im)を求めます。
float_complex& operator=(float rhs)	関数	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)	関数	rhs を実数部に加算し、和を*this に格納します。
float_complex& operator-=(float rhs)	関数	rhs を実数部から減算し、差を*this に格納します。
float_complex& operator*=(float rhs)	関数	rhs を乗算し、積を*this に格納します。
float_complex& operator/=(float rhs)	関数	rhs で除算し、商を*this に格納します。
float_complex& operator=(const float_complex& rhs)	関数	rhs をコピーします。
float_complex& operator+=(const float_complex& rhs)	関数	rhs を加算し、和を*this に格納します。
float_complex& operator-=(const float_complex& rhs)	関数	rhs を減算し、差を*this に格納します。
float_complex& operator*=(const float_complex& rhs)	関数	rhs を乗算し、積を*this に格納します。
float_complex& operator/=(const float_complex& rhs)	関数	rhs で除算し、商を*this に格納します。

(1) `float_complex::float_complex( float re=0.0f, float im=0.0f)`

■ 機能

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;
```

■ リターン値

なし

(2) `float_complex::float_complex( const double_complex& rhs)`

■ 機能

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();
```

■ リターン値

なし

(3) `float float_complex::real() const`

■ 機能

実数部を求めます。

■ リターン値

`this->_re`

(4) `float float_complex::imag() const`

■ 機能

虚数部を求めます。

■ リターン値

`this->_im`

(5) `float_complex& float_complex::operator=(float rhs)`

■ 機能

`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

`*this`

(6) `float_complex& float_complex::operator+=(float rhs)`

■ 機能

`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

`*this`

(7) `float_complex& float_complex::operator-=(float rhs)`

■ 機能

`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

`*this`

(8) `float_complex& float_complex::operator*=(float rhs)`

■ 機能

`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

`*this`

(9) `float_complex& float_complex::operator/=(float rhs)`

■ 機能

`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

`*this`

(10) `float_complex& float_complex::operator=(const float_complex& rhs)`

■ 機能

`rhs` をコピーします。

■ リターン値

`*this`

(11) `float_complex& float_complex::operator+=(const float_complex& rhs)`

■ 機能

`rhs` を加算し、結果を`*this` に格納します。

■ リターン値

`*this`

(12) `float_complex& float_complex::operator-=( const float_complex& rhs)`

■ 機能

rhs を減算し、結果を*this に格納します。

■ リターン値

*this

(13) `float_complex& float_complex::operator*=( const float_complex& rhs)`

■ 機能

rhs と乗算し、結果を*this に格納します。

■ リターン値

*this

(14) `float_complex& float_complex::operator/=( const float_complex& rhs)`

■ 機能

rhs で除算し、結果を*this に格納します。

■ リターン値

*this

1.4.2 float_complex メンバ外関数

■ 定義名一覧

定義名	種類	説明
float_complex operator+(const float_complex& lhs)	関数	lhs の単項 + 演算を行います。
float_complex operator+(const float_complex& lhs, const float_complex& rhs)	関数	lhs と rhs を加算し、和を lhs に格納します。
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)	関数	lhs の単項 - 演算を行います。
float_complex operator-(const float_complex& lhs, const float_complex& rhs)	関数	lhs から rhs を減算し、差を lhs に格納します。
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)	関数	lhs と rhs を乗算し、積を lhs に格納します。
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)	関数	lhs を rhs で除算し、商を lhs に格納します。
float_complex operator/ (const float_complex& lhs, const float& rhs)		
float_complex operator/ (const float& lhs, const float_complex& rhs)		

1. EC++クラスライブラリ

定義名	種類	説明
bool operator==(const float_complex& lhs, const float_complex& rhs)	関数	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)	関数	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)	関数	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 float_complex& x)	関数	u,(u)または x を(u,v) (u:実数部、v:虚数部) 形式で出力します。
float real(const float_complex& x)	関数	実数部を求めます。
float imag(const float_complex& x)	関数	虚数部を求めます。
float abs(const float_complex& x)	関数	絶対値を求めます。
float arg(const float_complex& x)	関数	位相角度を求めます。
float norm(const float_complex& x)	関数	2 乗の絶対値を求めます。
float_complex conj(const float_complex& x)	関数	共役複素数を求めます。
float_complex polar(const float& rho, const float& theta)	関数	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)	関数	複素余弦を求めます。
float_complex cosh(const float_complex& x)	関数	複素双曲余弦を求めます。
float_complex exp(const float_complex& x)	関数	指数関数を求めます。

定義名	種類	説明
<code>float_complex log(const float_complex& x)</code>	関数	自然対数を求めます。
<code>float_complex log10(const float_complex& x)</code>	関数	常用対数を求めます。
<code>float_complex pow(const float_complex& x, int y)</code>	関数	x の y 乗を求めます。
<code>float_complex pow(const float_complex& x, const float& y)</code>		
<code>float_complex pow(const float_complex& x, const float_complex& y)</code>		
<code>float_complex pow(const float& x, const float_complex& y)</code>		
<code>float_complex sin(const float_complex& x)</code>	関数	複素正弦を求めます。
<code>float_complex sinh(const float_complex& x)</code>	関数	複素双曲正弦を求めます。
<code>float_complex sqrt(const float_complex& x)</code>	関数	右半空間における範囲での平方根を求めます。
<code>float_complex tan(const float_complex& x)</code>	関数	複素正接を求めます。
<code>float_complex tanh(const float_complex& x)</code>	関数	複素双曲正接を求めます。

(1) `float_complex operator+( const float_complex& lhs)`

■ 機能

lhs の単項 + 演算を行います。

■ リターン値

lhs

(2) `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)`

■ 機能

lhs と rhs を加算し、結果を lhs に格納します。

■ リターン値

`float_complex(lhs)+=rhs`

(3) `float_complex operator - ( const float_complex& lhs)`

■ 機能

lhs の単項 - 演算を行います。

■ リターン値

`float_complex( - lhs.real(), - lhs.imag())`

(4) `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)`

■ 機能

lhs から rhs を減算し、結果を lhs に格納します。

■ リターン値

`float_complex(lhs) - =rhs`

(5) `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)`

■ 機能

lhs と rhs を乗算し、結果を lhs に格納します。

■ リターン値

`float_complex(lhs)*=rhs`

(6) `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)`

■ 機能

lhs を rhs で除算し、結果を lhs に格納します。

■ リターン値

`float_complex(lhs)/=rhs`

(7) `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)`

■ 機能

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

`lhs.real()==rhs.real() && lhs.imag()==rhs.imag()`

(8) `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)`

■ 機能

`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

`lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()`

(9) `istream& operator>>( istream& is, float_complex& x)`

■ 機能

`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

`is`

(10) `ostream& operator<<( ostream& os, const float_complex& x)`

■ 機能

`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

`os`

(11) `float real( const float_complex& x)`

■ 機能

実数部を求めます。

■ リターン値

`x.real()`

(12) `float imag( const float_complex& x)`

■ 機能

虚数部を求めます。

■ リターン値

`x.imag()`

(13) `float abs( const float_complex& x)`

■ 機能

絶対値を求めます。

■ リターン値

$|x.\text{real()}| + |x.\text{imag()}|$

(14) `float arg( const float_complex& x)`

■ 機能

位相角度を求めます。

■ リターン値

`atan2f( x.imag() , x.real())`

(15) `float norm( const float_complex& x)`

■ 機能

2乗の絶対値を求めます。

■ リターン値

$x.\text{real()}^2 + x.\text{imag()}^2$

(16) `float_complex conj( const float_complex& x)`

■ 機能

共役複素数を求めます。

■ リターン値

`float_complex(x.real(), (-1)*x.imag())`

(17) `float_complex polar( const float& rho, const float& theta)`

■ 機能

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

`float_complex(rho*cosf(theta), rho*sinf(theta))`

(18) `float_complex cos( const float_complex& x)`

■ 機能

複素余弦を求めます。

■ リターン値

`float_complex( cosf(x.real())*coshf(x.imag()), (-1)*sinf(x.real())*sinhf(x.imag()))`

(19) `float_complex cosh( const float_complex& x)`

■ 機能

複素双曲余弦を求めます。

■ リターン値

`cos(float_complex((-1)*x.imag(), x.real()))`

(20) `float_complex exp( const float_complex& x)`

■ 機能

指数関数を求めます。

■ リターン値

`expf(x.real())*cosf(x.imag()),expf(x.real())*sinf(x.imag())`

(21) `float_complex log( const float_complex& x)`

■ 機能

(e を底とする)自然対数を求めます。

■ リターン値

`float_complex( logf(abs(x)), arg(x))`

(22) `float_complex log10( const float_complex& x)`

■ 機能

(10 を底とする)常用対数を求めます。

■ リターン値

`float_complex( log10f(abs(x)), arg(x)/logf(10))`

(23) `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)`

■ 機能

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

`exp(y* logf(x))` (`float_complex pow (const float_complex& x,const float_complex& y)`のとき)

`exp( y*log(x))` (上記以外)

(24) `float_complex sin( const float_complex& x)`

■ 機能

複素正弦を求めます。

■ リターン値

`float_complex( sinf(x.real())*coshf(x.imag()), cosf(x.real())*sinhf(x.imag()))`

(25) `float_complex sinh(const float_complex& x)`

■ 機能

複素双曲正弦を求めます。

■ リターン値

`float_complex(0,-1)*sin(float_complex((-1)*x.imag(),x.real()))`

(26) `float_complex sqrt( const float_complex& x)`

■ 機能

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

`float_complex(sqrtf(abs(x))*cosf(arg(x)/2), sqrtf(abs(x))*sinf(arg(x)/2))`

(27) `float_complex tan( const float_complex& x)`

■ 機能

複素正接を求めます。

■ リターン値

$\sin(x) / \cos(x)$

(28) `float_complex tanh( const float_complex& x)`

■ 機能

複素双曲正接を求めます。

■ リターン値

$\sinh(x) / \cosh(x)$

1.4.3 double_complex クラス

■ 定義名一覧

定義名	種類	説明
value_type	マクロ	double 型です。
_re	関数	double 精度の実数部を定義します。
_im	関数	double 精度の虚数部を定義します。
double_complex(double re=0.0, double im=0.0)	関数	コンストラクタです。
double_complex(const float_complex&)		
double real() const	関数	実数部を求めます。
double imag() const	関数	虚数部を求めます。
double_complex& operator=(double rhs)	関数	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)	関数	rhs を実数部に加算し、和を*this に格納します。
double_complex& operator-=(double rhs)	関数	rhs を実数部から減算し、差を*this に格納します。
double_complex& operator*=(double rhs)	関数	rhs を乗算し、積を*this に格納します。
double_complex& operator/=(double rhs)	関数	rhs で除算し、商を*this に格納します。
double_complex& operator=(const double_complex& rhs)	関数	rhs をコピーします。
double_complex& operator+=(const double_complex& rhs)	関数	rhs を加算し、和を*this に格納します。
double_complex& operator-=(const double_complex& rhs)	関数	rhs を減算し、差を*this に格納します。
double_complex& operator*=(const double_complex& rhs)	関数	rhs を乗算し、積を*this に格納します。
double_complex& operator/=(const double_complex& rhs)	関数	rhs で除算し、商を*this に格納します。

(1) double_complex(double re=0.0, double im=0.0)

■ 機能

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;
```

■ リターン値

なし

(2) `double_complex( const float_complex&)`

■ 機能

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();
```

■ リターン値

なし

(3) `double double_complex::real() const`

■ 機能

実数部を求めます。

■ リターン値

`this->_re`

(4) `double double_complex::imag() const`

■ 機能

虚数部を求めます。

■ リターン値

`this->_im`

(5) `double_complex& double_complex::operator=(double rhs)`

■ 機能

`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

`*this`

(6) `double_complex& double_complex::operator+=(double rhs)`

■ 機能

`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

`*this`

(7) `double_complex& double_complex::operator-=(double rhs)`

■ 機能

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

*this

(8) `double_complex& double_complex::operator*=(double rhs)`

■ 機能

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

*this

(9) `double_complex& double_complex::operator/=(double rhs)`

■ 機能

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

*this

(10) `double_complex& double_complex::operator=( const double_complex& rhs)`

■ 機能

rhs をコピーします。

■ リターン値

*this

(11) `double_complex& double_complex::operator+=( const double_complex& rhs)`

■ 機能

rhs を加算し、結果を*this に格納します。

■ リターン値

*this

(12) `double_complex& double_complex::operator-=( const double_complex& rhs)`

■ 機能

rhs を減算し、結果を*this に格納します。

■ リターン値

*this

(13) `double_complex& double_complex::operator*=( const double_complex& rhs)`

■ 機能

rhs と乗算し、結果を*this に格納します。

■ リターン値

*this

(14) `double_complex& double_complex::operator/=( const double_complex& rhs)`

■ 機能

rhs で除算し、結果を*this に格納します。

■ リターン値

*this

1.4.4 double_complex メンバ外関数

■ 定義名一覧

定義名	種類	説明
double_complex operator+(const double_complex& lhs)	関数	lhs の単項 + 演算を行います。
double_complex operator+(const double_complex& lhs, const double_complex& rhs)	関数	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)	関数	lhs の単項 - 演算を行います。
double_complex operator-(const double_complex& lhs, const double_complex& rhs)	関数	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)	関数	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)	関数	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)		

1. EC++クラスライブラリ

定義名	種類	説明
bool operator==(const double_complex& lhs, const double_complex& rhs)	関数	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)	関数	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)	関数	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, double_complex& x)	関数	u,(u)または x を (u,v) (u:実数部、v:虚数部) 形式で出力します。
double real(const double_complex& x)	関数	実数部を求めます。
double imag(const double_complex& x)	関数	虚数部を求めます。
double abs(const double_complex& x)	関数	絶対値を求めます。
double arg(const double_complex& x)	関数	位相角度を求めます。
double norm(const double_complex& x)	関数	2 乗の絶対値を求めます。
double_complex conj(const double_complex& x)	関数	共役複素数を求めます。
double_complex polar(const double& rho, const double& theta)	関数	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)	関数	複素余弦を求めます。
double_complex cosh(const double_complex& x)	関数	複素双曲余弦を求めます。
double_complex exp(const double_complex& x)	関数	指数関数を求めます。

定義名	種類	説明
<code>double_complex log(const double_complex& x)</code>	関数	自然対数を求めます。
<code>double_complex log10(const double_complex& x)</code>	関数	常用対数を求めます。
<code>double_complex pow(const double_complex& x, int y)</code>	関数	x の y 乗を求めます。
<code>double_complex pow(const double_complex& x, const double& y)</code>		
<code>double_complex pow(const double_complex& x, const double_complex& y)</code>		
<code>double_complex pow(const double& x, const double_complex& y)</code>		
<code>double_complex sin(const double_complex& x)</code>	関数	複素正弦を求めます。
<code>double_complex sinh(const double_complex& x)</code>	関数	複素双曲正弦を求めます。
<code>double_complex sqrt(const double_complex& x)</code>	関数	右半空間における範囲での平方根を求めます。
<code>double_complex tan(const double_complex& x)</code>	関数	複素正接を求めます。
<code>double_complex tanh(const double_complex& x)</code>	関数	複素双曲正接を求めます。

(1) `double_complex operator+( const double_complex& lhs)`

■ 機能

lhs の単項 + 演算を行います。

■ リターン値

lhs

(2) `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)`

■ 機能

lhs と rhs を加算し、結果を lhs に格納します。

■ リターン値

`double_complex(lhs)+=rhs`

(3) `double_complex operator - ( const double_complex& lhs)`

■ 機能

lhs の単項 - 演算を行います。

■ リターン値

`double_complex( - lhs.real(), - lhs.imag())`

(4) `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)`

■ 機能

lhs から rhs を減算し、結果を lhs に格納します。

■ リターン値

`double_complex(lhs) - =rhs`

(5) `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)`

■ 機能

lhs と rhs を乗算し、結果を lhs に格納します。

■ リターン値

`double_complex(lhs)*=rhs`

(6) `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)`

■ 機能

lhs を rhs で除算し、結果を lhs に格納します。

■ リターン値

`double_complex(lhs)/=rhs`

(7) `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)`

■ 機能

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

`lhs.real()==rhs.real() && lhs.imag()==rhs.imag()`

(8) `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)`

■ 機能

`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

`lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()`

(9) `istream& operator>>( istream& is, double_complex& x)`

■ 機能

`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

`is`

(10) `ostream& operator<<( ostream& os, const double_complex& x)`

■ 機能

`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

`os`

(11) `double real( const double_complex& x)`

■ 機能

実数部を求めます。

■ リターン値

`x.real()`

(12) `double imag( const double_complex& x)`

■ 機能

虚数部を求めます。

■ リターン値

`x.imag()`

(13) `double abs( const double_complex& x)`

■ 機能

絶対値を求めます。

■ リターン値

$|x.\text{real()}| + |x.\text{imag()}|$

(14) `double arg( const double_complex& x)`

■ 機能

位相角度を求めます。

■ リターン値

`atan2( x.imag(), x.real())`

(15) `double norm( const double_complex& x)`

■ 機能

2乗の絶対値を求めます。

■ リターン値

$x.\text{real()}^2 + x.\text{imag()}^2$

(16) `double_complex conj( const double_complex& x)`

■ 機能

共役複素数を求めます。

■ リターン値

`double_complex(x.real(), (-1)*x.imag())`

(17) `double_complex polar( const double& rho, const double& theta)`

■ 機能

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

`double_complex(rho*cos(theta), rho*sin(theta))`

(18) `double_complex cos( const double_complex& x)`

■ 機能

複素余弦を求めます。

■ リターン値

`double_complex( cos(x.real())*cosh(x.imag()), (-1)*sin(x.real())*sinh(x.imag()))`

(19) `double_complex cosh( const double_complex& x)`

■ 機能

複素双曲余弦を求めます。

■ リターン値

`cos(double_complex((-1)*x.imag(), x.real()))`

(20) `double_complex exp( const double_complex& x)`

■ 機能

指数関数を求めます。

■ リターン値

`exp(x.real())*cos(x.imag()),exp(x.real())*sin(x.imag())`

(21) `double_complex log( const double_complex& x)`

■ 機能

(e を底とする)自然対数を求めます。

■ リターン値

`double_complex( log(abs(x)), arg(x))`

(22) `double_complex log10( const double_complex& x)`

■ 機能

(10 を底とする)常用対数を求めます。

■ リターン値

`double_complex( log10(abs(x)), arg(x)/log(10))`

(23) `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)`

■ 機能

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

`exp( y*log(x))`

(24) `double_complex sin( const double_complex& x)`

■ 機能

複素正弦を求めます。

■ リターン値

`double_complex( sin(x.real())*cosh(x.imag()), cos(x.real())*sinh(x.imag()))`

(25) `double_complex sinh(const double_complex& x)`

■ 機能

複素双曲正弦を求めます。

■ リターン値

`double_complex(0,-1)*sin(double_complex((-1)*x.imag(),x.real()))`

(26) `double_complex sqrt( const double_complex& x)`

■ 機能

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

`double_complex(sqrt(abs(x))*cos(arg(x)/2), sqrt(abs(x))*sin(arg(x)/2))`

(27) `double_complex tan( const double_complex& x)`

■ 機能

複素正接を求めます。

■ リターン値

$\sin(x) / \cos(x)$

(28) `double_complex tanh( const double_complex& x)`

■ 機能

複素双曲正接を求めます。

■ リターン値

$\sinh(x) / \cosh(x)$

1.5 文字列操作クラスライブラリ

文字列操作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。

(1) <string>

string クラスを定義します。

また、クラスの階層図は次のようになります。

```
graph TD
    string
```

1.5.1 string クラス

定義名	種別	説明
iterator	マクロ	char*型です。
const_iterator	マクロ	const char*型です。
npos	データ	文字列の最大長(UINT_MAX 文字)です。
s_ptr	データ	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	データ	オブジェクトが格納している文字列長です。
s_res	データ	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。
string(void)	関数	コンストラクタです。
string(const string& str, size_t pos=0, size_t n=npos)		
string(const char*str, size_t n)		
string(const char* str)		
string(size_t n, char c)		
~string()	関数	デストラクタです。
string& operator=(const string& str)	関数	str を代入します。
string& operator=(const char* str)	関数	str を代入します。
string& operator=(char c)	関数	c を代入します。
iterator begin()	関数	文字列の先頭ポインタを求めます。
const_iterator begin() const		
iterator end()	関数	文字列の最後尾ポインタを求めます。
const_iterator end() const		

1. EC++クラスライブラリ

定義	種別	説明
size_t size() const	関数	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	関数	確保している領域のサイズを求めます。
void resize(size_t n, char c)	関数	格納可能な文字列の長さを n に変更します。
void resize(size_t n)	関数	格納可能な文字列の長さを n に変更します。
size_t capacity() const	関数	確保している領域のサイズを求めます。
void reserve(size_t res_arg = 0)	関数	領域の再割り当てを行います。
void clear()	関数	格納されている文字列を clear します。
bool empty() const	関数	格納している文字列の長さが 0 かチェックします。
const char & operator[](size_t pos) const	関数	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)	関数	str の文字列を追加します。
string& operator+=(const char* str)	関数	str の文字列を追加します。
string& operator+=(char c)	関数	c の文字を追加します。
string& append(const string& str)	関数	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)	関数	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)	関数	文字列 str の n 文字分を追加します。
string& append(size_t n, char c)	関数	n 個の文字 c を追加します。

定義	種別	説明
string& assign(const string& str)	関数	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)	関数	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)	関数	文字列 str の n 文字分を代入します。
string& assign(size_t n, char c)	関数	n 個の文字 c を代入します。
string& insert(size_t pos1, const string& str)	関数	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)	関数	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* s, size_t n)	関数	pos の位置に文字列 s を n 文字分挿入します。
string& insert(size_t pos, const char* str)	関数	pos の位置に文字列 str を挿入します。
string& insert(size_t pos, size_t n, char c)	関数	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())	関数	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)	関数	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)	関数	位置 pos から n 個分取り除きます。

1. EC++クラスライブラリ

定義	種別	説明
iterator erase(iterator position)	関数	position により参照された文字を取り除きます。
iterator erase(iterator first, iterator last)	関数	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)	関数	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)	関数	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)	関数	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)	関数	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)	関数	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)		
string& replace(iterator i1, iterator i2, const char* str, size_t n)	関数	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。

定義	種別	説明
<code>string& replace(iterator i1, iterator i2, size_t n, char c)</code>	関数	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。
<code>size_t copy(char* str, size_t n, size_t pos=0) const</code>	関数	位置 pos に文字列 str の n 文字分の文字列をコピーします。
<code>void swap(string& str)</code>	関数	str の文字列と交換します。
<code>const char* c_str() const</code>	関数	文字列を格納している領域へのポインタを参照します。
<code>const char* data() const</code>		
<code>size_t find(const string& str, size_t pos=0) const</code>	関数	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。
<code>size_t find(const char* str, size_t pos=0) const</code>		
<code>size_t find(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t find(char c, size_t pos=0) const</code>	関数	位置 pos 以降で文字 c が最初に現れる位置を検索します。
<code>size_t rfind(const string& str, size_t pos=npos) const</code>	関数	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。
<code>size_t rfind(const char* str, size_t pos=npos) const</code>		
<code>size_t rfind(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t rfind(char c, size_t pos=npos) const</code>	関数	位置 pos 以前で文字 c が最後に現れる位置を検索します。
<code>size_t find_first_of(const string& str, size_t pos=0) const</code>	関数	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。
<code>size_t find_first_of(const char* str, size_t pos=0)</code>		

1. EC++クラスライブラリ

定義	種別	説明
<code>size_t find_first_of(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t find_first_of(char c, size_t pos=0) const</code>	関数	位置 pos 以降で文字 c が最初に現れる位置を検索します。
<code>size_t find_last_of(const string& str, size_t pos=npos) const</code>	関数	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。
<code>size_t find_last_of(const char* str, size_t pos=npos) const</code>		
<code>size_t find_last_of(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t find_last_of(char c, size_t pos=npos) const</code>	関数	位置 pos 以前で文字 c が最後に現れる位置を検索します。
<code>size_t find_first_not_of(const string& str, size_t pos=0) const</code>	関数	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。
<code>size_t find_first_not_of(const char* str, size_t pos=0) const</code>		
<code>size_t find_first_not_of(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t find_first_not_of(char c, size_t pos=0) const</code>	関数	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。
<code>size_t find_last_not_of(const string& str, size_t pos=npos) const</code>	関数	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。
<code>size_t find_last_not_of(const char* str, size_t pos=npos) const</code>		
<code>size_t find_last_not_of(const char* str, size_t pos, size_t n) const</code>	関数	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。
<code>size_t find_last_not_of(char c, size_t pos=npos) const</code>	関数	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。

定義	種別	説明
string substr(size_t pos=0, size_t n=npos) const	関数	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	関数	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	関数	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	関数	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	関数	str と比較します。
int compare(size_t pos1, size_t n1, const char* str, size_t n2=npow) const	関数	位置 pos1 から n1 文字分の文字列と str の sn2 文字分の文字列を比較します。

(1) string::string(void)

■ 機能

以下のように設定します。

```
s_ptr = 0;
s_len = 0;
s_res = 1;
```

■ リターン値

なし

(2) string::string(const string& str, size_t pos=0, size_t n=npow)

■ 機能

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。

■ リターン値

なし

(3) string::string(const char* str, size_t n)

■機能

以下に設定します。

```
s_ptr = str;  
s_len = n;  
s_res = n+1;
```

■リターン値

なし

(4) string::string(const char* str)

■機能

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
```

■リターン値

なし

(5) string::string(size_t n, char c)

■機能

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;
```

■リターン値

なし

(6) string::~~string()

■機能

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。

■リターン値

なし

(7) string& string::operator=(const string& str)

■機能

str のデータを代入します。

■リターン値

*this

(8) `string& string::operator=( const char* str)`

■ 機能

`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

`*this`

(9) `string& string::operator=( char c)`

■ 機能

`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

`*this`

(10) `string::iterator string::begin()`

`string::const_iterator string::begin() const`

■ 機能

文字列の先頭ポインタを求めます。

■ リターン値

文字列の先頭ポインタ

(11) `string::iterator string::end()`

`string::const_iterator string::end() const`

■ 機能

文字列の最後尾ポインタを求めます。

■ リターン値

文字列の最後尾ポインタ

(12) `size_t string::size() const`

`size_t string::length() const`

■ 機能

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

格納されている文字列の文字列長

(13) `size_t string::max_size() const`

■ 機能

確保している領域のサイズを求めます。

■ リターン値

確保している領域のサイズ

(14) `void string::resize( size_t n, char c)`

■ 機能

オブジェクトが格納可能な文字列の長さを `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>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。

■ リターン値

なし

(15) `void string::resize( size_t n)`

■ 機能

オブジェクトが格納している文字列の長さを `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。

■ リターン値

なし

(16) `size_t string::capacity() const`

■ 機能

確保している領域のサイズを求めます。

■ リターン値

確保している領域のサイズ

(17) `void string::reserve( size_t res_arg = 0)`

■ 機能

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。

■ リターン値

なし

(18) `void string::clear()`

■ 機能

格納されている文字列を `clear` します。

■ リターン値

なし

(19) `bool string::empty() const`

■ 機能

格納している文字列の長さが 0 かチェックします。

■ リターン値

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

(20) `const char & 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)`

■ 機能

`s_ptr[pos]`を参照します。

■ リターン値

`n < s_len` の場合 : `s_ptr[pos]`

`n >= s_len` の場合 : `'\0'`

(21) `string& string::operator+=(const string& str)`

■ 機能

`str` が格納している文字列を追加します。

■ リターン値

`*this`

(22) `string& string::operator+=(const char* str)`

■ 機能

`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

`*this`

(23) `string& string::operator+=(char c)`

■ 機能

`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

`*this`

(24) `string& string::append( const string& str)`

`string& string::append( const char* str)`

■ 機能

`str` の文字列をオブジェクトに追加します。

■ リターン値

`*this`

(25) `string& string::append( const string& str, size_t pos, size_t n);`

■ 機能

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

`*this`

(26) `string& string::append( const char* str, size_t n)`

■ 機能

文字列 `str` の `n` 文字分を追加します。

■ リターン値

`*this`

(27) `string& string::append( size_t n, char c)`

■ 機能

`n` 個の文字 `c` を追加します。

■ リターン値

`*this`

(28) `string& string::assign( const string& str)`

`string& string::assign( const char* str)`

■ 機能

`str` の文字列を代入します。

■ リターン値

`*this`

(29) `string& string::assign( const string& str, size_t pos, size_t n)`

■ 機能

位置 `pos` に文字列 `str` の `n` 文字分を代入します。

■ リターン値

`*this`

(30) `string& string::assign( const char* str, size_t n)`

■ 機能

文字列 `str` の `n` 文字分を代入します。

■ リターン値

`*this`

(31) `string& string::assign( size_t n, char c)`

■ 機能

`n` 個の文字 `c` を代入します。

■ リターン値

`*this`

(32) `string& string::insert( size_t pos1, const string& str)`

■ 機能

位置 `pos1` に `str` の文字列を挿入します。

■ リターン値

`*this`

(33) `string& string::insert( size_t pos1, const string& str, size_t pos2, size_t n)`

■ 機能

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

`*this`

(34) `string& string::insert( size_t pos, const char* str, size_t n)`

■ 機能

`pos` の位置に文字列 `str` を `n` 文字分挿入します。

■ リターン値

`*this`

(35) `string& string::insert( size_t pos, const char* str)`

■ 機能

`pos` の位置に文字列 `str` を挿入します。

■ リターン値

`*this`

(36) `string& string::insert( size_t pos, size_t n, char c)`

■ 機能

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

`*this`

(37) `string::iterator string::insert( iterator p, char c=char())`

■ 機能

`p` が指す文字列の前に、文字 `c` を挿入します。

■ リターン値

挿入された文字

(38) `void string::insert( iterator p, size_t n, char c)`

■ 機能

`p` で指す文字の前に、`n` 個の文字 `c` を挿入します。

■ リターン値

なし

(39) `string& string::erase( size_t pos=0, size_t n=npos)`

■ 機能

位置 `pos` から `n` 個分取り除きます。

■ リターン値

`*this`

(40) `iterator string::erase( iterator position)`

■ 機能

`position` により参照された文字を取り除きます。

■ リターン値

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()`

(41) `iterator string::erase( iterator first, iterator last)`

■ 機能

範囲`[first, last]`において文字を取り除きます。

■ リターン値

`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()`

(42) `string& string::replace( size_t pos1, size_t n1, const string& str)`

`string& string::replace( size_t pos1, size_t n1, const char* str)`

■機能

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

`*this`

(43) `string& string::replace( size_t pos, size_t n1, const string& str, size_t pos2, size_t n2)`

■機能

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

`*this`

(44) `string& string::replace( size_t pos, size_t n1, const char* str, size_t n2)`

■機能

位置 `pos` から `n1` 文字分の文字列を、`str` の `n2` 文字分の文字列で置き換えます。

■リターン値

`*this`

(45) `string& string::replace( size_t pos, size_t n1, size_t n2, char c)`

■機能

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

`*this`

(46) `string& string::replace( iterator i1, iterator i2, const string& str)`

`string& string::replace( iterator i1, iterator i2, const char* str)`

■機能

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

`*this`

(47) `string& string::replace( iterator i1, iterator i2, const char* str, size_t n)`

■機能

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

`*this`

(48) `string& string::replace( iterator i1, iterator i2, size_t n, char c)`

■ 機能

位置 `i1` から `i2` までの文字を、`n` 個の文字 `c` で置き換えます。

■ リターン値

`*this`

(49) `size_t string::copy( char* str, size_t n, size_t pos=0) const`

■ 機能

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

`rlen`

(50) `void string::swap( string& str)`

■ 機能

`str` の文字列と交換します。

■ リターン値

なし

(51) `const char* string::c_str() const`

`const char* string::data() const`

■ 機能

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

`s_ptr`

(52) `size_t string::find( const string& str, size_t pos=0) const`

`size_t string::find( const char* str, size_t pos=0) const`

■ 機能

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

文字列のオフセット

(53) `size_t string::find( const char* str, size_t pos, size_t n) const`

■ 機能

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

文字列のオフセット

(54) `size_t string::find( char c, size_t pos=0) const`

■機能

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

文字列のオフセット

(55) `size_t string::rfind( const string& str, size_t pos=npos) const`

`size_t string::rfind( char *str, size_t pos=npos) const`

■機能

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

文字列のオフセット

(56) `size_t string::rfind( const char* str,size_t pos,size_t n) const`

■機能

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

文字列のオフセット

(57) `size_t string::rfind( char c,size_t pos=npos) const`

■機能

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

文字列のオフセット

(58) `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`

■機能

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

文字列のオフセット

(59) `size_t string::find_first_of( const char* str, size_t pos, size_t n) const`

■機能

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

文字列のオフセット

(60) `size_t string::find_first_of( char c, size_t pos=0) const`

■ 機能

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

文字列のオフセット

(61) `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`

■ 機能

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

文字列のオフセット

(62) `size_t string::find_last_of( const char* str, size_t pos, size_t n) const`

■ 機能

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

文字列のオフセット

(63) `size_t string::find_last_of( char c, size_t pos=npos) const`

■ 機能

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

文字列のオフセット

(64) `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`

■ 機能

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

文字列のオフセット

(65) `size_t string::find_first_not_of( const char* str, size_t pos, size_t n) const`

■ 機能

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

文字列のオフセット

(66) `size_t string::find_first_not_of( char c, size_t pos=0) const`

- 機能

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

文字列のオフセット

(67) `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`

- 機能

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

文字列のオフセット

(68) `size_t string::find_last_not_of( const char* str, size_t pos, size_t n) const`

- 機能

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

文字列のオフセット

(69) `size_t string::find_last_not_of( char c, size_t pos=npos) const`

- 機能

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

文字列のオフセット

(70) `string string::substr( size_t pos=0, size_t n=npos) const`

- 機能

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

範囲`[pos,n]`の文字列を持つオブジェクト

(71) `int string::compare( const string& str) const`

- 機能

文字列と `str` の文字列を比較します。

■ リターン値

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

(72) int string::compare(size_t pos1, size_t n1, const string& str) const

■ 機能

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

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

(73) int string::compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const

■ 機能

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

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

(74) int string::compare(const char* str) const

■ 機能

str と比較します。

■ リターン値

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

(75) int string::compare(size_t pos1, size_t n1, const char* str, size_t n2=npos) const

■ 機能

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

文字列が同一の場合は：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

1.5.2 string クラスマニピュレータ

■ 定義名一覧

定義名	説明
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)	

1. EC++クラスライブラリ

定義名	説明
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。
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。
istream & getline(istream &is, string &str)	

(1) `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)`

■機能

lhs の文字列（または文字）に rhs の文字列（または文字）を結合し、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

結合した文字列を格納するオブジェクト

(2) `bool operator ==( const string &lhs, const string &rhs)`

`bool operator ==( const char *lhs, const string &rhs)`

`bool operator ==( const string &lhs, const char *rhs)`

■機能

lhs の文字列と rhs の文字列を比較します。

■リターン値

文字列が同一の場合：TRUE

文字列が異なる場合：FALSE

(3) `bool operator !=( const string &lhs, const string &rhs)`

`bool operator !=( const char *lhs, const string &rhs)`

`bool operator !=( const string &lhs, const char *rhs)`

■機能

lhs の文字列と rhs の文字列を比較します。

■リターン値

文字列が同一の場合：FALSE

文字列が異なる場合：TRUE

(4) `bool operator <( const string &lhs, const string &rhs)`

`bool operator <( const char *lhs, const string &rhs)`

`bool operator <( const string &lhs, const char *rhs)`

■機能

lhs の文字列長と rhs の文字列長を比較します。

■ リターン値

lhs.s_len < rhs.s_len の場合 : TRUE
lhs.s_len >= rhs.s_len の場合 : FALSE

- (5) bool operator >(const string &lhs, const string &rhs)
bool operator >(const char *lhs, const string &rhs)
bool operator >(const string &lhs, const char *rhs)

■ 機能

lhs の文字列長と rhs の文字列長を比較します。

■ リターン値

lhs.s_len > rhs.s_len の場合 : TRUE
lhs.s_len <= rhs.s_len の場合 : FALSE

- (6) bool operator <=(const string &lhs, const string &rhs)
bool operator <=(const char *lhs, const string &rhs)
bool operator <=(const string &lhs, const char *rhs)

■ 機能

lhs の文字列長と rhs の文字列長を比較します。

■ リターン値

lhs.s_len <= rhs.s_len の場合 : TRUE
lhs.s_len > rhs.s_len の場合 : FALSE

- (7) bool operator >=(const string &lhs, const string &rhs)
bool operator >=(const char *lhs, const string &rhs)
bool operator >=(const string &lhs, const char *rhs)

■ 機能

lhs に格納された文字列長と rhs に格納された文字列長を比較します。

■ リターン値

lhs.s_len >= rhs.s_len の場合 : TRUE
lhs.s_len < rhs.s_len の場合 : FALSE

- (8) void swap(string &lhs, string &rhs)

■ 機能

lhs の文字列と rhs の文字列を交換します。

■ リターン値

なし

(9) `istream & operator >> ( istream &is, string &str)`

■ 機能

文字列を `str` に抽出します。

■ リターン値

`is`

(10) `ostream & operator << ( ostream &os, const string &str)`

■ 機能

文字列を挿入します。

■ リターン値

`os`

(11) `istream & getline( istream &is, string & str, char delim)`

`istream & getline( istream &is, string &str)`

■ 機能

`is` から文字列を抽出し , `str` に付加します。

■ リターン値

`is`

索引

—	
_ec2p_copy_base	6
_ec2p_init_base	6
_ec2p_new_handler	51
A	
abs	62, 74
arg	62, 74
B	
boolalpha	14
C	
conj	62, 74
cos	62, 74
cosh	63, 75
D	
dec	16
double_complex	65, 66
double_complex::imag	66
double_complex::operator*=	67, 68
double_complex::operator/=	67, 68
double_complex::operator+=	66, 67
double_complex::operator=	66, 67
double_complex::operator - =	67
double_complex::real	66
double_complexクラス	65
double_complexメンバ外関数	69
E	
EC++入出力ライブラリの使用例	49
endl	45
ends	45
exp	63, 75
F	
fixed	17
float_complex::float_complex	54
float_complex::imag	54

float_complex::operator*=	55, 56
float_complex::operator/=	55, 56
float_complex::operator+=	55
float_complex::operator=	54, 55
float_complex::operator - =	55, 56
float_complex::real	54
float_complexクラス	53
float_complexメンバ外関数	57
flush	45

G

getline	101
---------	-----

H

hex	16
-----	----

I

imag	61, 73
Init::~Init	3
Init::Init	3
int_type	2
internal	16
ios::~ios	10
ios::bad	11
ios::clear	11
ios::copyfmt	12
ios::eof	11
ios::fail	12
ios::good	11
ios::init	10
ios::ios	9, 10
ios::operator void*	10
ios::operator!	10
ios::rdbuf	12
ios::rdstate	10
ios::setstate	11
ios::tie	12
ios_base::~ios_base	7
ios_base::fill	8
ios_base::flags	7
ios_base::fmtflags	5
ios_base::Initクラス	3
ios_base::ios_base	6
ios_base::iostate	5
ios_base::openmode	6
ios_base::precision	8
ios_base::seekdir	6

ios_base::setf	7
ios_base::unsetf	7
ios_base::width	8
ios_baseクラス	4
iosクラス	9
iosクラスマニピュレータ	13
istream::_ec2p_getistr	32
istream::~istream	32
istream::gcount	33
istream::get	33, 34
istream::getline	35
istream::ignore	35
istream::istream	32
istream::operator>>	32, 33
istream::peek	35
istream::putback	36
istream::read	35
istream::readsome	36
istream::seekg	37
istream::sentryクラス	28
istream::sync	36
istream::tellg	36
istream::unget	36
istreamクラス	29
istreamクラスマニピュレータ	38
istreamメンバ外関数	39

L

left	16
log	63, 75
log10	63, 75

M

mystdbufクラス	49
-------------------	----

N

noboolalpha	14
norm	62, 74
noshowbase	14
noshowpoint	14
noshowpos	15
noskipws	15
nouppercase	15

O

oct	16
off_type	2

operator -	60, 72
operator delete	52
operator delete[]	52
operator new	51, 52
operator new[]	52
operator!=	61, 73, 99
operator*	60, 72
operator/	60, 72
operator[]	87
operator+	59, 71, 99
operator<	99
operator<<	46, 61, 73, 101
operator<=	100
operator==	60, 72, 99
operator>	100
operator>=	100
operator>>	39, 61, 73, 101
ostream::~ostream	42
ostream::flush	44
ostream::operator<<	43
ostream::ostream	42
ostream::put	43
ostream::seekp	44
ostream::sentryクラス	40
ostream::tellp	44
ostream::write	43
ostreamクラス	41
ostreamクラスマニピュレータ	45
ostreamメンバ外関数	46

P

polar	62, 74
pos_type	2
pow	63, 75

R

real	61, 73
resetiosflags	47
right	16

S

scientific	17
sentry::~sentry	28, 40
sentry::operator bool	28, 40
sentry::sentry	28, 40
set_new_handler	52
setbase	47

setfill	47
setiosflags	47
setprecision	48
setw	48
showbase	14
showpoint	14
showpos	15
sin	64, 76
sinh	64, 76
skipws	15
smanip クラス マニピュレータ	47
sqrt	64, 76
streambuf::~~streambuf	20
streambuf::eback	23
streambuf::egptr	23
streambuf::epptr	24
streambuf::gbump	23
streambuf::gptr	23
streambuf::in_avail	21
streambuf::overflow	27
streambuf::pbackfail	26
streambuf::pbase	24
streambuf::pbump	24
streambuf::pptr	24
streambuf::pubseekoff	21
streambuf::pubseekpos	21
streambuf::pubsetbuf	20
streambuf::pubsync	21
streambuf::sbumpc	22
streambuf::seekoff	25
streambuf::seekpos	25
streambuf::setbuf	25
streambuf::setg	24
streambuf::setp	25
streambuf::sgetc	22
streambuf::sgetn	22
streambuf::showmanyc	26
streambuf::snextc	21
streambuf::sputbackc	22
streambuf::putc	22
streambuf::putn	23
streambuf::streambuf	20
streambuf::sungetc	22
streambuf::sync	25
streambuf::uflow	26
streambuf::underflow	26
streambuf::xsgetn	26

streambuf::xsputn.....	26
streambufクラス	18
streamoff	2
streamsize	2
string::~~string	84
string::append	88
string::assign.....	88, 89
string::at	87
string::begin	85
string::c_str	92
string::capacity.....	86
string::clear	86
string::compare	95, 96
string::copy.....	92
string::data.....	92
string::empty	87
string::end	85
string::erase	90
string::find.....	92, 93
string::find_first_not_of.....	94, 95
string::find_first_of.....	93, 94
string::find_last_not_of.....	95
string::find_last_of.....	94
string::insert.....	89, 90
string::max_size	85
string::operator[].....	87
string::operator+=.....	87
string::operator=	84, 85
string::replace	91, 92
string::reserve	86
string::resize.....	86
string::rfind	93
string::size	85
string::string	83, 84
string::substr	95
string::swap.....	92
stringクラス	77
stringクラスマニピュレータ	97
swap.....	100

T

tan.....	64, 76
tanh.....	64, 76

U

uppercase	15
-----------------	----

W

WS.....	38
---------	----

す

ストリーム入出力用クラスライブラリ.....	2
------------------------	---

め

メモリ管理用ライブラリ	51
-------------------	----

ら

ライブラリの概要	1
----------------	---

漢字

複素数計算用クラスライブラリ.....	53
文字列操作用クラスライブラリ.....	77

SuperH RISC engine EC++ クラスライブラリ 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
神奈川県川崎市中原区下沼部1753 〒211-8668

ADJ-702-271