

C++

基本プログラミング

| 講 | 座 |

安心して安全なC++プログラム開発の手引き

日向俊二◎著



■ サンプルファイルのダウンロードについて

本書掲載のサンプルファイルは、下記 URL からダウンロードできます。

<https://----->

- 本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送または E-mail）でお送りください。
- 電話によるお問い合わせはお受けできません。
- 本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
- 匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

本書で取り上げられているシステム名／製品名は、一般に開発各社の登録商標／商品名です。本書では、™ および ® マークは明記していません。本書に掲載されている団体／商品に対して、その商標権を侵害する意図は一切ありません。本書で紹介している URL や各サイトの内容は変更される場合があります。

はじめに

C++ は、オブジェクト指向のプログラミングに対応する強力なプログラミング言語です。現代の C++ は、機能がとても充実していますが、それだけにさまざまな種類の多数の要素から構成されています。本書を通して読むことで、C++ のプログラマとして知っておきたい基本的なことを幅広く学習できます。

C++ は C 言語の機能をほぼそのまま使えることもあって、独特の難しさがあるという問題もあります。たとえば、ポインタや配列の扱い方を間違ってしまうと致命的なエラーとなり、しかもその原因がわかりにくいことがあります。また、セキュリティ上のリスクを避けるためにも、ポインタや配列について特別な注意を払わなければならない場合があります。しかし、ポインタや配列を使う代わりに、C++ の string や STL (Standard Template Library) のコンテナを利用することなどで、C++ の特徴や機能を活用した、より安心して安全なプログラムを作成することができます。このように C++ の機能を積極的に使うことが、良い C++ プログラムを開発する際のキーポイントです。

本書では、より安心して安全なプログラミングを確実にできることに焦点を当てて、C++ の基本的なことから STL や例外処理まで、C++ プログラミングで必ず押さえておきたいことを具体的なコード例を示して解説します。本書にはコンパイルして実行可能なシンプルでわかりやすいサンプルプログラムを多数掲載してあります。そのため、それらをよく読んでコードの意味をつかみ、コンパイルして実行することで、C++ のプログラミングのさまざまな面をより深く理解できるようになるでしょう。

本書で C++ を学ぶことで、C++ のパワーを生かしながら、より安全で安心確実なプログラムを開発するための基礎力を身に付けてください。

2024 年 春 著者しるす

■ 本書の表記

- C++n** 使用できる C++ の規格を示します。たとえば C++11 は C++11 以降の規格でコンパイルするときに有効であることを示します。
- ()** ひとまとまりの実行可能なコードブロックである関数であることを示します。たとえば、main という関数を表すときに、「main という名前の関数」や「関数 main()」と表記しないで、単に「main()」と表記することがあります。
- abc** 斜体で示す文字列は、そこに具体的な文字や数値が入ることを表します。たとえば「if (*expr1* < *expr2*)」は、*expr1* と *expr2* に数値や変数、式などが入り、具体的には「if (x > 0)」や「if (a+b < 100)」などとなることを表します。
- 0xn** 0x で始まる表記は 16 進数表現の整数であることを表します。たとえば、0x41 は 10 進数で 65 であることを表します。
- On** 0 で始まる数値の表記は 8 進数表現の整数であることを表します。たとえば、041 は 10 進数で 33 であることを表します。
- ...** 書式の説明において任意の個数を記述できることを示します。
- []** 書式の説明において省略可能であることを示します。
- [X]** キーボードのキーを押すことを示します。たとえば、[F5] は F5 キーを押すことを意味します。
- [S] + [X]** キーボードの S キーを押したまま X キーを押すことを示します。[Ctrl] + [F5] は、Ctrl キーを押したまま F5 キーを押すことを意味します。
- >** Windows のコマンドプロンプトを表します。
- \$** Linux や WSL など UNIX 系 OS のコマンドプロンプトを表します。



.....
本文を補足するような説明や、知っておくとよい話題です。
.....

- 処理系** C++ のコンパイラの具体的な構成やコンパイルの方法、それが生成するファイルなどは、OS やコンパイラの種類とバージョンによって異なります。ある OS の特定の種類の特定のバージョン全体を表すときに、処理系という言葉を使います。たとえば、Linux 系の GCC と Windows で Microsoft のコンパイラを使うときはそれぞれ別の処理系であるとみなします。
- 標準出力** C++ のプログラムでは、出力にウィンドウを使うことがよくありますが、パイ

プヤリダイレクトという OS の機能を使って他のファイルやデバイスへと出力することもできます。これを標準出力と呼びます。

標準入力 C++ のプログラムでは、入力にキーボードを使うことがよくありますが、パイプヤリダイレクトという OS の機能を使って他のファイルやデバイスから入力することもできます。これを標準入力と呼びます。

ターミナル コマンドラインを入力してプログラムを起動したり、入力や表示を行う場所（典型的にはウィンドウ）を指します。Linux など UNIX 系システムでは一般にターミナル（Terminal）と呼ばれますが、Windows 10 の場合は「コマンドプロンプト」か「PowerShell」（切り替え可能）で、Windows 11 の場合は既定のターミナルは「Windows Terminal」または「Windows ターミナル」という名前を用意されています。ターミナルという言葉は初心者には理解しやすいように標準出力の代わりに使うことがあります。

キーボード 標準入力からの入力を、初心者にはわかりやすいようにキーボードからの入力と呼ぶ場合があります。

■ ご注意

- 本書の内容は本書執筆時の状態で記述しています。C++ の場合、コンパイラの種類や C++ のバージョンによって異なる点があり、本書の記述と実際とが異なる結果となる可能性があります。
- 本書の多くの記述は現在使われている一般的な C++ コンパイラの多くに対応していますが、サンプルコードをコンパイルして実行する場合は C++14 以降のコンパイラを使うことを推奨します。また、記述の一部は C++17 や C++20 で新たに導入されたことを含んでいます。
- 本書は C++ のすべてのことについて完全に解説するものではありません。必要に応じて C++ のドキュメントなどを参照してください。
- 本書のサンプルは、プログラミングを理解するために掲載するものです。実用的なアプリとして提供するものではありませんので、ユーザーのエラーへの対処やセキュリティ、その他の面で省略してあるところがあります。

■ 本書に関するお問い合わせについて

本書に関するお問い合わせは、sales@cutt.co.jp にメールでご連絡ください。

なお、お問い合わせは本書に記述されている範囲に限らせていただきます。特定の環境や特定の目的に対するお問い合わせ等にはお答えできませんので、あらかじめご了承ください。特に、特定の環境における特定のコンパイラや開発ツールのインストールや設定、使い方、読者固有の環境におけるエラーなどについてご質問いただいてもお答えできませんのでご了承ください。

お問い合わせの際には下記事項を明記してくださいますようお願いいたします。

- 氏名
- 連絡先メールアドレス
- 書名
- 記載ページ
- お問い合わせ内容
- 実行環境

目次

はじめに..... iii

第1章 C++の基礎知識..... 1

1.1 プログラミング言語 C++..... 1

1.2 プログラムの作成・実行手順..... 6

1.3 言語とライブラリ..... 12

1.4 C++ と日本語..... 17

第2章 C++の基本要素..... 21

2.1 プログラムの構成..... 21

2.2 基本要素..... 24

2.3 定数と変数..... 30

第3章 値と計算..... 37

3.1 整数..... 37

3.2 実数..... 51

3.3 その他の型..... 55

3.4 乱数..... 59

第4章 演算子..... 65

4.1 比較演算子..... 65

4.2 ビット関連演算子..... 68

4.3 論理演算子..... 72

4.4 さまざまな演算子..... 73

4.5 演算の優先順位..... 78

第5章 文字と文字列 81

- 5.1 文字 81
- 5.2 文字列 85
- 5.3 配列に保存する文字列 92
- 5.4 数値と文字列の変換 98
- 5.5 日本語の文字と文字列 101

第6章 制御構造 105

- 6.1 条件分岐 105
- 6.2 繰り返し 110
- 6.3 その他のステートメント 117

第7章 入出力 121

- 7.1 標準入力と出力 121
- 7.2 ストリーム 123
- 7.3 Cスタイルの書式付き入出力 128
- 7.4 Cスタイルのファイル入出力 134

第8章 関数とマクロ 149

- 8.1 関数 149
- 8.2 関数の定義 153
- 8.3 マクロ 163

第9章 配列とポインタ 169

- 9.1 配列 169
- 9.2 ポインタ 177
- 9.3 関数のポインタ 187
- 9.4 コマンドライン引数 189

第 10 章	構造体、共用体、列挙型	193
	10.1 構造体.....	193
	10.2 共用体と列挙型.....	204
第 11 章	クラス	209
	11.1 クラス.....	209
	11.2 継承.....	218
第 12 章	オーバーロードとオーバーライド	229
	12.1 オーバーロード.....	229
	12.2 オーバーライド.....	233
第 13 章	テンプレート	239
	13.1 テンプレートの概要.....	239
	13.2 関数テンプレート.....	241
	13.3 クラステンプレート.....	244
第 14 章	STL コンテナとアルゴリズム	247
	14.1 STL.....	247
	14.2 コンテナとイテレーター	249
	14.3 さまざまな STL コンテナ.....	252
	14.4 アルゴリズム	266
第 15 章	名前空間	295
	15.1 名前空間	295
	15.2 名前空間の作成と利用	299

第 16 章	マルチスレッド	305
	16.1 スレッドと並列処理.....	305
	16.2 排他制御	312
第 17 章	エラー処理とデバッグ	317
	17.1 例外処理	317
	17.2 アサート	321
	17.3 実行時型情報	323
第 18 章	ファイル構成	327
	18.1 ソースファイル.....	327
	18.2 ヘッダーファイル.....	328
	18.3 複数のソースモジュール	332
第 19 章	C 言語との連携	339
	19.1 C 言語コードの利用.....	339
	19.2 C ソースのリンク	341
	19.3 他言語の移植	347
付 録	351
	付録 A 開発環境	351
	付録 B トラブル対策	359
	付録 C 練習問題解答例	369
	付録 D 参考リソース	393
	索 引	394

1

C++ の基礎知識

ここでは C++ について最も基本的なことがらを解説します。C++ のさまざまなことについてこの章では概要を紹介し、各項目の具体的な例は第 2 章以降で説明します。

1.1 プログラミング言語 C++

ここでは、C++ とそのプログラミングについて最も基本的なことを説明します。

■ C++

プログラミング言語 C++ は、オブジェクト指向プログラミングのために C 言語から派生して生まれたプログラミング言語です。

オブジェクト指向ではクラスというものを定義してオブジェクトというものを作って利用します。クラスにはデータと、操作や処理を行うためのコードが含まれます。C++ ではデータとコードをクラスにまとめて扱うことができるので、比較的複雑で大規模なプログラムに適しています。

C++ には機能を実現するさまざまなクラスライブラリが提供されていて、これを活用することで複雑高度なことも実現できます。また、文字列操作のような単純なことも C 言語より容易かつ安全にできます。

■ C++ と C 言語

C++ はゼロから設計されたプログラミング言語ではなく、C 言語を拡張して作成されたオブジェクト指向プログラミング言語です。初期の C++ は C 言語のプリプロセッサ（前処理プログラム）を使って C++ のコードを C 言語のコードに書き換えることで C++ に導入された機能を実現する言語として作られました。

C++ は C 言語から生まれたため、C 言語と C++ のシンタックスの大半は同じです。実際、C++ のプログラムの中では C 言語のコードをほとんどそのまま使うことができます。そのため、C++ 固有のいくつかの点を除くと、C 言語と C++ はほとんど同じものとも考えることも可能です。しかし、それぞれの言語を適切に利用するためには、それぞれの言語の特性をきちんと把握しておく必要があります。

ほとんどの場合、C++ のプログラムの中に C 言語のコードを埋め込むことが可能で、C++ のプログラムの中に C 言語の関数を記述したり、C 言語で記述したモジュールと容易にリンクすることができます。しかし、C++ のコードと C 言語のコードをむやみに混在させるとトラブルのもとになる場合があります。

たとえば、入出力において、C++ の `>>` や `<<` を使う入出力と、`scanf()` や `printf()` を使う入出力をむやみに混用すると、意図した通りに入出力が行われない場合があります。

■ C++ のプログラミングスタイル

C++ のプログラミングの方法は、本来、C 言語のプログラミングの方法とは異なります。端的に言えば、C 言語のプログラミングとは関数を作って利用することであるのに対して、C++ はクラスを定義して利用することです。

C++ は、オブジェクト指向プログラミングのためのプログラミング言語です。オブジェクト指向のアプローチでは、一連のものの共通する特性を突き止めて分類し（抽象化）、クラスを定義します。たとえば、住所録アプリケーションを作る場合、住所と氏名などからなる個人の情報のクラスと、そのクラスのインスタンス（オブジェクト）に氏名や住所などを保存するための C++ のクラスのメンバー関数（メソッド）を記述します。

C++ ではものごとを抽象化してオブジェクトとして捉えますが、このような作業は特別なことではありません。日常生活の中でも、我々は意識せずにものの特性を見極めて分類し、クラスとしてモデル化します。たとえば、近所の野良犬から山田さんの家のハチ公まで、犬とみなせる動物をすべて犬として扱って、他の動物と区別します。そして、犬は哺乳動物に属するということを知ると、犬を、他の哺乳動物が持つ特性と同じ特性を持つものとして認識します。

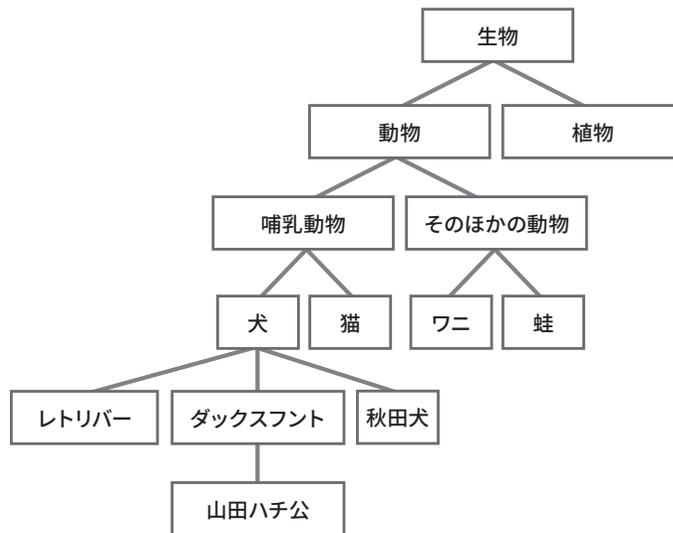


図1.1 ●オブジェクト指向のアプローチ

プログラミングにおけるオブジェクト指向のアプローチは、このような考え方をプログラミングに導入したものと捉えることができます。

オブジェクト指向のプログラミングでは、あらゆることを想定し、よく分析し検討して可能な限り現実の事象を論理的に矛盾なく扱いやすいように抽象化してモデルを考えます。オブジェクトのモデル化の方法や結果は、目的や対象によって異なります。また、複雑な問題を扱う場合には最初の段階ですべてを完全に捉えることが事実上不可能なため、試行錯誤を伴う作業になることがあるのがオブジェクト指向プログラミングの特徴です。たとえば、図 1.1 では動物を哺乳動物とそれ以外の動物に分けました。実生活においてはこの程度の分類で十分な場合が多いでしょうが、動物を学術的に扱うときには、哺乳類、甲殻類、両生類などのように類で分ける必要があるでしょう。また、目的によっては、「ワンと吠える」、「ニャオと鳴く」、「吠えも鳴きもしない」という基準で分類するほうが合理的である場合もあります。分類のしかたは目的によって異なり、いずれが正解でどれが間違いということはありません。プログラミングでも同じことで、目的に合わせて最適なモデル作りを目指します。

オブジェクトはクラスから生成します。クラスは、データとコードを持ちます。たとえば、犬クラスの場合、たとえば「名前」や「年齢」というデータと、「吠える」や「寝る」などのコードを定義します。

■ C++の規格

C言語の拡張として誕生したC++も当初はさまざまな種類がありました。その後、C++は、C++11 (ISO/IEC 14882:2011)、C++14 (ISO/IEC 14882:2014)、C++17 (ISO/IEC 14882:2017)、C++20 (ISO/IEC 14882:2020)などの規格が制定されましたが、サポートされる機能がそれぞれ多少異なります。



C++20やC++17などの規格を、慣用的に、仕様やバージョンなどと呼ぶことがあり、また言語標準と呼ぶことがあります。

コンパイラがサポートするC++のバージョンは、次のようにして調べたり設定することができます。

コンパイラがg++やclang++の場合、グローバル変数__cplusplusの値を調べることでサポートするC++言語のバージョンがわかります。たとえば、次のようなプログラムをコンパイルして実行します。

リスト 1.1 ● cppver.cpp

```
#include <iostream>

int main()
{
    std::cout << __cplusplus << std::endl;
    return 0;
}
```

このプログラムを実行すると、「201703」や「201402」などの文字列が出力されるでしょう。「201703」はC++17を、「201402」はC++14を表しています。



特定の規格の他に、コンパイラベンダーが独自の仕様を追加している場合があります。たとえば、MicrosoftのC++コンパイラにはMicrosoft固有の仕様があります。

■ バージョンの指定

特定の C++ のバージョン（言語標準）を指定してコンパイルする場合は、g++ と clang++ の場合は次のように -std オプションを指定します。

```
>g++ -std=c++17 -o cppver cppver.cpp
```

```
>cppver
201703
```

Visual Studio の場合は、メニューから [プロジェクト] → [○○のプロパティ] で「○○プロパティページ」ダイアログボックスを開き、[C/C++] → [言語] で「C++ 言語標準」から指定したい C++ のバージョンを選択します。

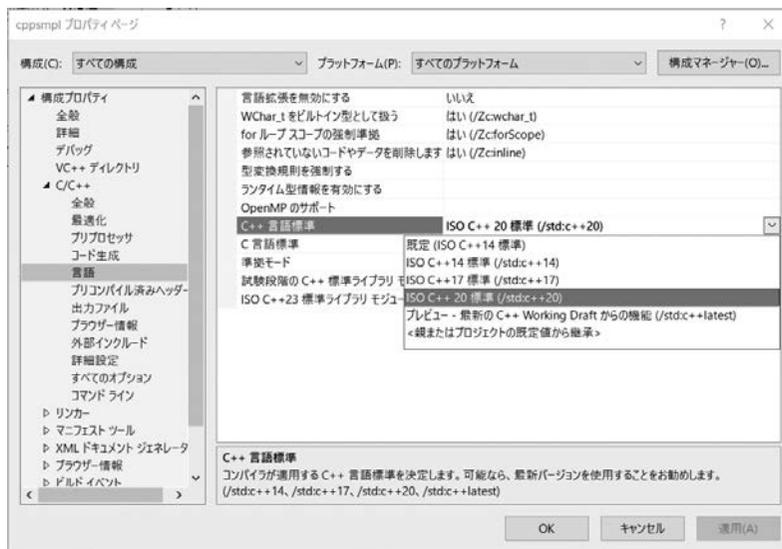


図1.2 ● Visual Studioの「○○プロパティページ」

Microsoft のコンパイラを使う場合にコマンドラインから特定の言語のバージョンを指定して cl でコンパイルする場合は、オプション /std を指定します。次の例は C++17 を指定する例です。

```
>cl /std:c++17 /EHsc sample.cpp
```

「/EHsc」は出力（ostreamで例外処理）を使っているような場合の警告を抑止するためのオプションです。

Eclipse (Pleiades) の場合は、メニューから [プロジェクト] → [プロパティ] で「○○のプロパティ」ダイアログボックスを開き、[C/C++ ビルド] → [設定] で「GCC C++ Compiler」の「ダイレクト」で「言語標準」から指定したいC++のバージョンを選択します。

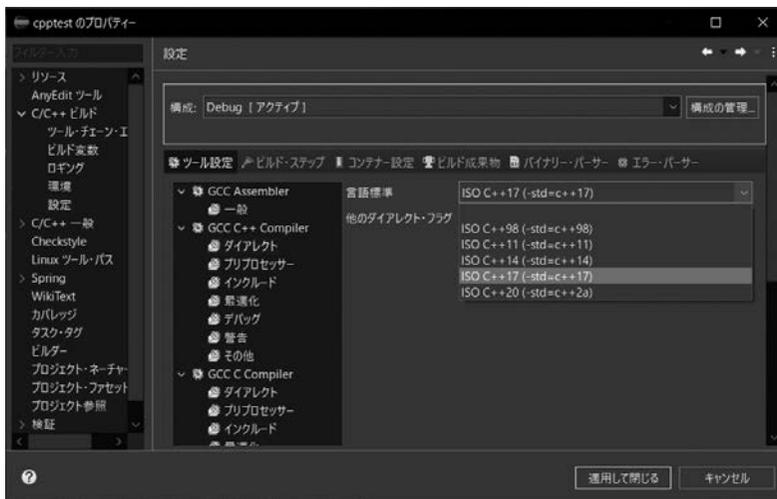


図1.3 ● Eclipse (Pleiades) の「○○のプロパティ」

ここで注意が必要なことは、特定の規格（言語標準）に準拠している場合でも、必ずしもそのC++の規格のすべての機能をサポートしていない場合があるという点です。たとえば、Visual Studioで「ISO C++20 標準」を選択しても（本書執筆時点では）実装されていないC++20の機能があります。規格にあるから必ずできると考えないほうが良いでしょう。

1.2 プログラムの作成・実行手順

ここでは、C++ 言語の単純なプログラムを作成して実行する手順を解説します。

■ プログラム開発の流れ

ここで説明するのは最も原始的なC++プログラム作成の流れです。プログラム開発でIDE

を使うのが当たり前になって久しいですが、Visual Studio や Pleiades、または Visual Studio Code のような高機能エディタを使う場合でも、効率よく作業するためにこの基本的な流れを把握しておきましょう。

(1) ソースファイルの作成

テキストエディタなどを使って、ソースコードと呼ばれる人間が読める形で記述されたプログラムを作成します。作成したプログラムはプレーンテキスト形式のファイルとして保存しますが、ファイルの拡張子は .txt ではなく .cpp としてください。

(2) コンパイル

コンパイラと呼ばれるプログラムを使って、ソースコードが保存されているファイルを読み込み、コンピューターが実行できる形にしたファイル（実行可能ファイル）を生成します。C++ プログラムのコンパイルに使うコマンドとして、g++、clang++、clang、cl.exe（Microsoft Windows）などがあります。単純なプログラムをコンパイルして実行する場合は、いずれを使っても操作方法や生成される見かけの結果はほぼ同じです。IDE を使った開発では、「ビルド」のようなメニュー項目を選択したりツールバーのボタンをクリックすれば、コンパイラプログラムが動いて実行可能ファイルを生成します。

なお、プログラムを正しくコンパイルできない場合は付録 B 「トラブル対策」を参照してください。

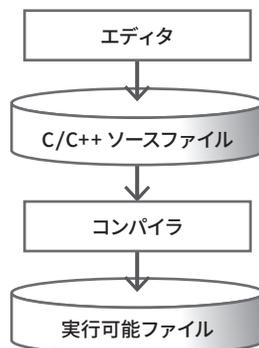


図1.4 ●C++プログラムのコンパイル手順の例

(3) 実行

実行可能ファイルは、特に指示をしなければ、Windowsではソースファイル名の拡張子が .exe に変更された名前、UNIX系OSでは a.out という名前で生成されます (WindowsでUNIX系コンパイラを使う場合は a.exe になります)。コマンドプロンプトに対して実行可能ファイル名を入力すると、プログラムが実行されます。

IDEを使った開発では、「実行」のようなメニュー項目を選択したりツールバーのボタンをクリックします。

以上が基本的なコンパイルの手順です。通常はプログラムの動作確認とデバッグで上記の手順を繰り返すことになります。

■ コンパイルの過程

普通、単にコンパイル (またはビルド) と呼んでいる、ソースファイルから実行可能ファイルを生成する処理は、図 1.5 に示すように、プリプロセス、コンパイル、リンクに分けることができます (図 1.5 は図 1.4 をさらに詳しく説明した図です)。

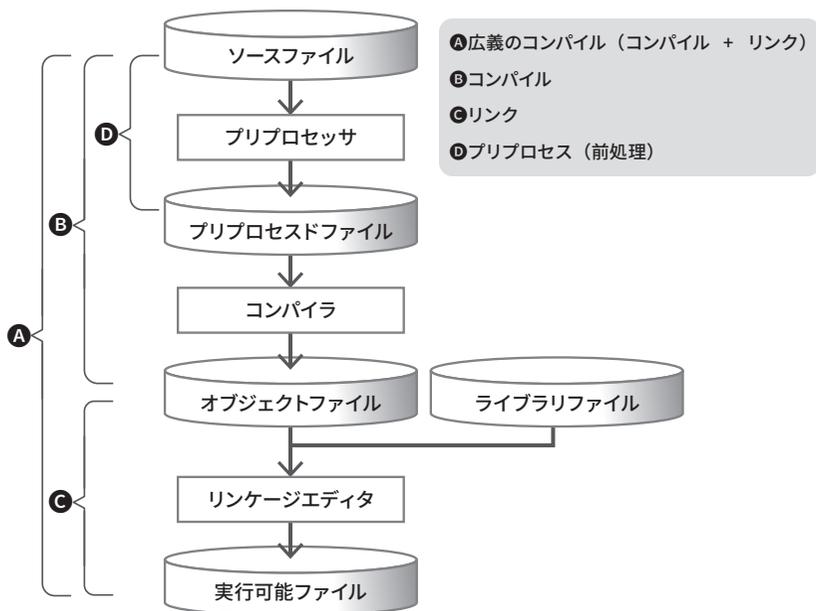


図1.5 ●C++プログラムのコンパイル手順

ソースファイルをコンパイルしてオブジェクトファイルを生成するプログラムを狭義のコンパイラといいます。ソースファイルを前処理（プリコンパイル）する部分だけを、特にプリプロセッサといいます。

オブジェクトファイルを他のオブジェクトファイルやライブラリとリンクして実行可能ファイルを生成するプログラムを、リンカーまたはリンケージエディタといいます。

前項「プログラム開発の流れ」で説明したコンパイルのコマンド `g++`、`clang`、`cl`などは、正確に言えば、コンパイルとリンクを制御するコマンドです。簡単なプログラムであれば、これらのコマンドにソースファイル名を引数として与えるだけで実行可能ファイルを作成することができます。しかし、複数のソースファイルをコンパイル・リンクするような場合や、プログラムをデバッグしたり効率的なコンパイラオプションを指定する場合などに、コンパイルの詳細な過程を知っておくと役立ちます。

■ C++ の Hello プログラム

ここでは C++ の最も基本的なプログラムを作成して実行する方法をとおして C++ プログラムのコンパイルに関することを説明します。

C++ の最も基本的なプログラムを次に示します。これは、「Hello, C++」という文字列を表示して終了する小さなプログラムです。

リスト 1.2 ● hello.cpp

```
#include <iostream>

int main()
{
    std::cout << "Hello, C++" << std::endl;

    return 0;
}
```

このプログラムを正しくコンパイルして実行できれば、C++ のプログラミングの基本的な環境が整っていることを確認できます。

■ ディレクトリの作成

通常、プログラムを作るときには、最初にそのプログラム専用のディレクトリを作成します。プログラムの規模が大きくてソースファイルが複数になるような場合には、そのプログラム

のプロジェクト用のディレクトリを作成します。

この場合、hello という名前の単一のソースコードだけの C++ プログラムなので、mkdir コマンドでサブディレクトリ hello を作成します。そして、その後で、cd コマンドでカレントディレクトリを hello にします。

UNIX 系 OS の場合は、次のようにします。

```
$ mkdir hello
$ cd hello
```

Windows の場合は、次のようにします。

```
>mkdir hello
>cd hello
```



Visual Studio や Eclipse (Pleiades) のような IDE でプログラムを作成して実行するためには、新しい C 言語プログラムのプロジェクトを作成します。IDE を使う場合には付録 A 「開発環境」も参照してください。

■ プログラムの編集と保存

プログラムまたはプロジェクトのためのディレクトリを作成したら、プログラムファイルを作成したり編集します。

この場合は、テキストエディタを起動してリスト 1.2 のプログラムを入力します。

プログラムを入力したら、hello.cpp という名前を付けてファイルに保存します。

C++ のプログラムファイルの拡張子は、現在では .cpp にするのが一般的です。拡張子として他に、.cxx、.cc、.C (大文字の C)、.CC (大文字の CC) などさまざまなものを使うことができますが、既存の C++ プログラムのソースファイルを使う場合と特別な理由がある場合を除いてこれらの拡張子は使わず、.cpp にすることを推奨します。特に .C (大文字の C) は C 言語のソースファイルの拡張子 .c (小文字の c) と勘違いしやすいので避けるべきです。

■ コンパイル

C++ コンパイラ (g++ または clang や cl など) のコマンドを使ってソースファイルをコンパイルします。次のように、コンパイラの名前の後にオプション `-o` と実行可能ファイル名を指定し、さらにソースファイル名を指定する方法が一般的です。

```
g++ -o hello hello.cpp
または
clang++ -o hello hello.cpp
```

多くの C++ コンパイラで出力ファイル名の指定に `-o` オプションが使われます。ただし、`cl.exe` ではオプション `'o'` を使うことは推奨されていません。

```
cl hello.cpp
```

いずれの場合も、コンパイルが問題なく終了すると、`hello` という名前の実行可能ファイルが生成されます (Windows の場合は `hello.exe` という名前になります)。

■ 実行

エラーメッセージが表示されなければ、コンパイルが成功しているので実行します。UNIX 系 OS や Windows の新しいシェルの場合は次のようにします。

```
$ ./hello
Hello, C++
```

`./` はカレントディレクトリのファイルであることを示します。

Windows の従来のコマンドプロンプトの場合や環境変数 `PATH` にカレントディレクトリ (`.`) が含まれていれば、`hello [Enter]` で実行することができます。

```
> hello
Hello, C++
```

プログラムを正しくコンパイルしたり実行できない場合は、付録 B「トラブル対策」を参照してください。また、Visual C++ や Eclipse などの IDE を使う場合は、それぞれの製品のドキュ

メントを見て hello プログラムを作成してみてください。

問題 1-1

C++ の Hello プログラムを作成してコンパイルし、実行してください。

1.3 言語とライブラリ

ここでは C++ 言語におけるプログラミング言語とライブラリについて解説します。

■ コンパイラとライブラリ

プログラミング言語としての C++ には、「文字列を出力する」というような特定の目的のための機能を持ったものが含まれていません。

通常、C++ のプログラミングでは、こうした特定の機能を使いたいときには、既存の関数ライブラリやクラスライブラリを利用します。たとえば、「文字列を出力する」ためには、通常はあらかじめ作成されていてコンパイラに添付されているライブラリの機能を使います。また、平方根や立方根を計算するような場合や乱数を発生させるようなときには、ライブラリの関数を使います。

ライブラリはクラスと関数を集めてまとめたものです。ライブラリ関数は、特定のライブラリに含まれる関数を指します。クラスライブラリは、クラスを集めてまとめたもので、クラスには機能を実現するメンバー関数（メソッド）と値を保存するフィールドや定数が定義されています。

C++ では、C++ のプログラムにリンクするように作成されたどのようなライブラリでも使うことができますが、主に次のようなライブラリを使います。

■ C++ の標準ライブラリ

C++ 言語の機能を使うためのライブラリがコンパイラと共に提供されています。

C++ のプログラミングでは、たとえば、文字列を扱うために `string` というクラスを使うことができますが、これは C++ の標準ライブラリに含まれています。また、出力に `cout << s`

<< endl;」のようなコードを使いますが、この「<<」を実現するコードも標準ライブラリに含まれています。



.....
 C++ の標準ライブラリのうち主要なものいくつかは後の章で具体的な使用例を示します。しかし、すべてを説明することはできないので、詳しくは付録 D 「参考リソース」に URL を記載したリファレンスを参照してください。

標準 C++ ライブラリの要素は以下のとおりです。

- 標準テンプレートライブラリ (STL)
- IOStream 機能
- ロケール機能
- テンプレート化された文字列クラス (string クラス)
- 複素数を表現するテンプレート化された複素数クラス (complex クラス)
- 数値配列処理用に最適化された数値の配列クラス (valarray クラス)
- 基本データ型の値の範囲や符号などの情報を統一した方法で提供するクラス (numeric_limits クラス)
- メモリー管理機能
- 多言語文字セットのサポート
- 例外処理機能

標準ライブラリの内容はヘッダーファイルで整理されています。主なものを以下に示します。

表1.1 ●標準ライブラリの主なヘッダーファイルとその主な内容

ヘッダー	説明
<any>	あらゆる型の値を保持できるメモリー型を提供する (C++17)
<bitset>	ビットの固定サイズシーケンスを提供する
<chrono>	時間ユーティリティを提供する (C++11)
<memory>	メモリーを扱うための機能を提供する
<memory_resource>	ポリモルフィックなアロケータや実装を提供する (C++17)
<optional>	任意で値を持たせられるオブジェクトの型を提供する (C++17)