第0章

さあ、はじめよう

まず、はじめに小さなC++プログラムを考えましょう。

```
// 小さなC++プログラム
#include <iostream>
int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

これはしばしば「Hello, world!プログラム」とよばれるものです。ご覧のように小さいプログラムですが、この本を読み進める前に、ご自身のコンピュータ上でコンパイルし実行してみてください。実行結果は、ディスプレイ上のウインドウなどに

```
Hello, world!
```

と表示されるはずです。もし、うまく行かなければ、近くにいるC++経験者に助けを求めるか、http://www.acceleratedcpp.comのアドバイスを読んでください。[*1]

このプログラムはとても小さいものなので、コンパイル時にエラーメッセージが出て、それがタイプミスによるものでなければ、コンパイラなどの使い方に問題があるとすぐわかります。さらに、このプログラムを徹底的に理解することで、C++の基本について、驚くほど多くのことを学ぶことができます。それでは、このプログラムを1行1行ゆっくり見ていきましょう。

0.1 コメント

最初の行は

```
// 小さなC++プログラム
```

とあります。この// ではじまる部分は、行末までコメント（commnet）とよばれるものになります。コンパイラはコメントを読みません。したがって、その目的は、プログラムを読む人間の理解を助けるというものです。

[*1] 訳注：原著者によるものですので、英語です。

0.2 #include

入力や出力といったC++の基本的なツール（道具、仕組み）は、**言語そのもの**（core language）の一部ではなく、**標準ライブラリ**（standard library）というものに含まれています。この点は重要です。というのは、あるものが言語の一部であればすべてのC++プログラムで自由に使えますが、標準ライブラリの一部であればそれを使えるようにする作業が必要になるからです。

プログラム中で標準ライブラリのツールを使いたい場合は、**#include**ディレクティブ（directive）、というものを使わなければなりません。これは通常プログラムのはじめに書かれます。今のプログラムで使いたい標準ライブラリのツールは、入出力だけなので、プログラムのはじめに

```
#include <iostream>
```

とあるわけです。iostreamという名前は、ランダムアクセスとかグラフィカルではない、シーケンシャルな、あるいはストリーム型ともよばれる、入出力を意味しています。#includeの後で**角カッコ**（<と>、angle bracket）に挟まれていますが、これは、角カッコの中身が標準ライブラリの一部で、**標準ヘッダ**（standard header）とよばれるものであることを示しています。

標準ヘッダの詳細な内容は定められていませんが、ヘッダ名とその機能は決まっています。標準ヘッダを読み込む（include）と標準ライブラリの機能が使えるようになる、ということがポイントです。標準ライブラリが具体的にどのように作られているかを知る必要はありません。

0.3 main

プログラムの中で名前がついていて、プログラムの他のところで使えるものを**関数**（function）と言います。関数を使うことを、**呼び出す**（call）とも言います。すべてのC++プログラムはmainという関数を持つ必要があります。私たちがC++プログラムを実行しようとしたときに、実行環境がこの関数を呼び出し、実行してくれるのです。

main関数は、整数を戻す（返す）ことになっています。これはこのプログラムを実行した環境にプログラムの実行が成功であったか失敗であったかを知らせるためです。実行が成功であった場合0を戻し、そうでないときは0以外の整数を戻すことになっています。そのため、

```
int main()
```

と書いて、int型の値を戻すmainという名前の関数の定義を始めているのです。C++では、整数（integer）をintで表します。後の丸カッコの中は、プログラム実行時に外部から受け取るデータをパラメータとして書くところですが、この例では外からデータを受け取らないので、丸カッコの中には何も書いてありません。ここにパラメータを記入する例は§10.4で見ます。

0.4 中カッコ

mainの定義は、続く**中カッコ**（{と}、curly brace）の中に書かれます。また、中カッコの中に書かれるものをステートメント（文、statement）とよびます。

```
int main()
{                       // 中カッコ開く
                        // ここにステートメントを書く
}                       // 中カッコ閉じる
```

C++では、中カッコの中身は、何であれひとつのブロック（機能のまとまり）として扱われます。今のプログラムでは、{はmain関数の定義の始まりを意味し、}がその終わりを意味しています。言い換えれば、{と}の間にあるものは、すべて同じ関数内にあるということです。

今の例のように、中カッコの中にふたつ以上のステートメントがある場合は、それらは書かれている順番に実行されます。

0.5 出力に標準ライブラリを使う

中カッコ内の最初のステートメントが、このプログラムの本質的な仕事をする部分です。

```
std::cout << "Hello, world!" << std::endl;
```

ここでは、標準出力（standard output）にHello, world!と書き出すために、標準ライブラリの**出力演算子**（output operator）<<が使われ、それからstd::endlも出力されています。

std::がついたものは、それがstdという**名前空間**（namespace）に属していることを示しています。ここで名前空間とは、関連したものの集まりのことで、標準ライブラリに属するツールには、すべてstd::ではじまる名前がついているのです。詳しくいうと、iostreamという標準ヘッダにはcoutやendlというものが定義されているのですが、それらはstdという名前空間に属していて、使われるときにはstd::cout、std::endlという名前でよばれるのです。

C++プログラムがどのようなツールを使おうとも、std::coutという名前は**標準出力ストリーム**(standard ouput stream)を意味します。たとえば、ウインドウタイプのシステム上では、std::coutは、そのプログラムが実行結果を出力するために割り当てられたウインドウを意味します。したがって、std::coutに書かれた値が、そのウインドウに出力されることになります。

std::endlを出力すると、出力行はそこで終わり、次に何かを出力すると、それは次の行に出るようになります。

0.6 return ステートメント

このプログラムにある、

```
return 0;
```

のようなreturnステートメントは、その関数の実行を終了し、returnと;の間にある値（今は0）を、関数が呼ばれた場所に戻す働きをします。戻す値の型は、関数の前に書いた型と同じでなければなりません。mainが戻す型はintであり、それを戻されるのは、実行環境です。それゆえ、一般にmainは、整数値か整数を意味する式を戻すように書かなければなりません。

もちろん、プログラムを終了させる場所がひとつとは限りません。その場合はreturnステートメントが複数になります。関数の定義で戻り値を決めたら、正しい型の値を戻す必要があります。

0.7 少しだけ深く

このプログラムには、C++で重要な概念が、2つ使われています。それはエクスプレッションとスコープ（有効範囲）です。これらについては、これから多くのことを説明していきますが、ここではまず基本的なことを述べておくことにします。

エクスプレッション（表現、式、expression）は、コンピュータに何かを実行させる命令（要請）です。何かが実行されると、**結果**（result）が戻されますが、**副作用**（side effect）も起こります。[*2]この副作用は、結果とは別にプログラムや環境に何らかの影響を与えます。たとえば、3+4はエクスプレッションで、その結果は7が戻されます。そしてこれに副作用はありません。また、

```
std::cout << "Hello, world!" << std::endl
```

はエクスプレッションですが、画面にHello, world!と出力され、続いて改行されるのは副作用の働きです。

エクスプレッションの中には、いろいろな形の演算子（作用を表すもの、operator）とオペランド（作用されるもの、被演算子、operand）というものがあります。上のエクスプレッションでは2つの<<が演算子でstd::cout、"Hello, world!"、std::endlがオペランドです。

すべてのオペランドには型（type）があります。型に関する詳しい説明は後にまわしますが、基本的に、型はデータ構造とそのデータ構造に働く作用を決めているということができます。

型にはたいてい名前があります。たとえば、C++ではintが整数の型の名前です。また、ライブラリにはstd::ostreamがストリーム型の出力を意味する型として定義されています。今の例のstd::coutの型がstd::ostreamなのです。

演算子<<は2つのオペランドを持つのですが、上の例では<<が2つなのに対して、オペランドは3つしかありません。どうしてでしょう。それは、<<が**左結合**（left-associative）だからです。その意味は、同じエクスプレッションに複数回<<が出てきた場合、なるべく左側のものを使い右側のものは後回しで実行されていくということです。つまり、上の例では、最初の<<が右側のオペランドとしては"Hello, world!"だけを使って、左側のオペランドstd::coutと共に実行され、その後に、左の<<がstd::cout << "Hello, world!"を左側のオペランドとして使うということです。カッコを使って演算子とオペランドの関係をはっきりさせると、上のエクスプレッションは

```
(std::cout << "Hello, world!") << std::endl
```

[*2] 訳注：普通の日本語では「副作用」は好ましくない「マイナス効果」のように考えられがちですが、ここでいう「副作用」は、一般にはプログラマが望むものです。戻される「結果」より「副作用」の方が重要であることも多々あります。

となるのです。ここで、それぞれの<<はオペランドの種類にしたがって振る舞いが決められます。左の<<の左側のオペランドstd::coutの型はstd::ostreamでした。右側のオペランドは文字列リテラルです。文字列リテラルとは"で囲った文字列のことです。これは少し難しいので詳しくは§10.2で説明しましょう。演算子<<の左右にこのようなオペランドがある場合は、右のオペランドの文字列が左のオペランドの表す出力画面に書かれるようになっているのです。

そして、左の<<が戻した結果が、右の<<の左オペランドになります。実は、左の<<ではstd::ostreamという型のstd::coutそのものが戻されるようになっています。また、今度の右オペランドはstd::endlですが、これは**操作子**（マニピュレータ、manipulator）とよばれるものです。操作子を使うと、ストリームが操作でき、単に文字を書き出す以上の処理が行えるのです。演算子<<の左オペランドがstd::ostreamの型を持ち、右オペランドが操作子であった場合、ストリームには操作子が影響を与え、その結果が戻されます。今の例のstd::endlは、出力行を改行させるという影響を与えます。

結局、エクスプレッション全体はstd::coutを戻し、その副作用が、Hello, world!と標準の出力場所に出力し、さらに改行するのです。エクスプレッションにセミコロン;がついている場合は、戻り値は捨て去るという意味になります。これは、副作用が望んだ仕事をしてくれるからです。

名前の**スコープ**（scope）とは、プログラム中でその名前が意味を持つ範囲のことです。C++にはいくつかの種類のスコープがありますが、そのうち2つがこのプログラムにあります。

最初のものは名前空間です。これは関連する名前の集まりです。標準ライブラリはすべての名前を名前空間std内で定義しています。そのため、自分でstdという名前空間を定義しようなどとしなければ、名前を二重に定義してしまうこともないのです。標準ライブラリのものを使う場合には、それが標準ライブラリのものであることを明確にする必要があります。たとえば、std::coutは名前空間std内で定義されたcoutという意味になります。

std::coutのように::を使った名前は、**修飾された名前**（qualified name）と言います。演算子::は**スコープ演算子**（scope operator）とも言われます。::の左側に（修飾されているかもしれない）スコープの名前がきます。たとえば、今の例ではstdが名前空間であることを示しており、::の右側は、そのスコープでの名前です。結局、std::coutは「stdというスコープ（今の場合は名前空間）内のcoutという名前」という意味になるのです。

中カッコは別の種類のスコープです。mainなど、関数の中身はスコープです。この事実は、今のように小さなプログラムではたいしたことに思えないかもしれませんが、関数を書くときにはいつも重要になります。

0.8 詳細

この章で例にみたプログラムは単純なものでしたが、多くの基礎的事項を含んでいました。以後は、この章の内容をもとに進めていくので、読み進める前にこの章は完全に理解しておく必要があります。

理解を助けるため、第16章を除く他の章でもそうしますが、「詳細」という節と練習を付ける事にします。この「詳細」は、まとめと、ときに補足も含みます。各章で説明したことを忘れないように、「詳細」を見直すことをお勧めします。

プログラムの構造： C++プログラムは、基本的に**自由形式**（free form）になっている。これは、空白が、独立したものを分けるだけにあるという意味である。特に、改行（ある行と次の行をわけるもの）も、空白の一種で

あって、一般には特別な意味はないことに注意すること。空白をどういれるかで、プログラムが読みやすくなったり読みにくくなったりする。普通プログラムは、なるべく読みやすく書くべきである。

自由形式の例外は3つある。

文字列リテラル　ダブルクォーテーション（"）で囲まれた文字列。これは複数行にできない。

`#include` 名前　これは1行で書く。（コメントは別。）

`//` コメント　この形のコメントはその行の行末まで続く。

`/*`ではじまるコメントは自由形式で、複数行になってもかまわない。このコメントは`*/`が出てきたところで終わる。

型： データ構造とそのデータ構造に作用するものを定義する。C++には、2種類の型がある。1つは、`int`のようにC++言語に最初からある型（本書では「組み込み型」とよぶ）で、もう1つは`std::ostream`のように言語とは別に定義されるもの。

名前空間： 関連する名前をグループにする仕組み。標準ライブラリにあるものの名前はすべて名前空間の`std`で定義されている。

文字列リテラル： ダブルクォーテーション（"）で囲まれた文字の列。これは1行で書く。\マーク（バックスラッシュ）[*3]付きのある文字は文字列リテラルの中で特別な意味を持つ。

`\n`　改行
`\t`　タブ
`\b`　バックスペース
`\"`　文字列リテラルのおわり（terminator）ではなく、文字としての（"）
`\'`　文字列リテラルや文字リテラル（§1.2を参照）の中での文字としての（'）
`\\`　文字列の中での文字としての\マーク。この後の文字は普通の文字として扱われる。

§10.2と§A.2.1.3で、文字列リテラルについてより詳しく説明する。

定義とヘッダ： C++プログラム中のすべての名前には定義がある。標準ライブラリ内の名前はヘッダとよばれるものの中にあり、`#include`によってアクセスできるようになる。名前は使われる前に定義しなければならない。したがって、`#include`文は、ヘッダ内の名前が使われる前に書いておかなければならない。特に`<iostream>`ヘッダは入出力のツールを定義している。

main関数： C++プログラムにはいつも、`main`という名の`int`を戻す関数が1つ必要。この`main`を呼び出すことでプログラムが実行されるからである。実行が成功した場合は0を戻し、それ以外の戻り値はなんらかのエラーを意味する。一般に、すべての関数は最低1つはある`return`文で終了する。しかし、`main`関数は特別で、`return`を省略することもできる。その場合、`main`は0を戻すものとして扱われる。ただ、きちんと`return`文を書くのがよい習慣である。

[*3] 訳注：日本工業規格「情報交換符号」で制定されたときに\の代わりに円記号¥が割り当てられた（JIS X 0201）。そのためパソコンなどShift-JISコードを使っている場合も、\は円記号¥が表示される。

中カッコとセミコロン： あまり目立たない記号だが、C++では重要。小さい記号なので見落としやすいが、1つでもつけ忘れると、コンパイラが理解不可能なエラーメッセージを出すことがよくある。

一般に、中カッコの中に何もない場合、ステートメントが1つある場合、複数のステートメントがある場合、どの場合も中カッコ全体を1つのステートメントと見る。複数のステートメントは順番に実行されることになっている。関数の中身は、たとえ1行だけであっても、中カッコで囲まれていなければならない。中カッコのペアはスコープを意味する。

セミコロンのついたエクスプレッションは、ステートメントである。これは、このステートメントを副作用のために実行し、戻り値は捨ててしまうことを意味する。副作用のないステートメントで戻り値を捨て、結果としてそれを空のステートメント（null statement）としてもかまわない。

出力： `std::cout << e`は、`e`を標準の出力ストリームに書き出す。`std::cout`の型は`ostream`で、上のステートメントは`std::cout`自身を戻す。これによって、出力の命令をつなげて書ける。

課題

0-0 `Hello, world!`プログラムをコンパイルし、実行してみてください。

0-1 下のステートメントは何をするものでしょうか。

```
3 + 4;
```

0-2 以下の文を出力するプログラムを書いてください。

```
This (") is a quote, and this (\) is a backslash.
```

0-3 `\t`はタブを表しますが、環境によってタブの意味は違います。自分の環境ではどのようになるか確かめてください。

0-4 「`Hello, world!`プログラム」全体を出力するプログラムを書いてください。

0-5 以下は正しいプログラムでしょうか。その理由も説明してください。

```
#include <iostream>
int main() std::cout << "Hello, world!" << std::endl;
```

0-6 以下は正しいプログラムでしょうか。その理由も説明してください。

```
#include <iostream>
int main() {{{{{{ std::cout << "Hello, world!" << std::endl; }}}}}}
```

0-7 以下のプログラムはどうでしょうか。

```
#include <iostream>
int main()
{
    /*これは複数行にまたがるコメントです。
    このコメントは/*ではじまり、 */でおわるタイプのものです。*/
    std::cout << "うまくいくかな？" << std::endl;
```