# 2.1 进入C++

首先是一个显示消息的简单C++程序，使用C++工具cout生成字符操作。

**程序清单2.1**

// myfirst.cpp—显示一条信息displays a message

#include <iostream> // 预处理编译指令a PREPROCESSOR directive

int main() // 函数头function header

{ // 函数体开头start of function body

using namespace std; // 编译指令make definitions visible

cout << "Come up and C++ me some time."; // 输出一条信息message

cout << endl; // 另起一行start a new line

cout << "You won't regret it!" << endl; // 输出下一条信息more output

// 如果在显示信息前窗口关闭If the output window closes before you can read it,

// 则添加以下代码add the following code:

cout << "Press any key to continue." <<endl;

cin.get();

return 0; // 结束main()函数terminate main()

} // 函数体结尾end of function body

要让窗口一直打开，可在retum语句前添加如下语句：

cin.get();

示例非常简单，只包含一个名为main()的函数。myfirst.cpp示例包含下述元素。

* 注释，由前缀//标识。
* 预处理器编译指令#include。
* 函数头：int main()。
* 编译指令using namespace。
* 函数体，用{和}括起。
* 使用C++的cout工具显示消息的语句。
* 结束main()函数的return语句。

## 2.1.1 main()函数

去掉修饰后，程序清单2.1中的示例程序的基本结构如下：

int main()

{

statements

return 0;

}

这几行表明有一个名为main()的函数，并描述了该函数的行为。这几行代码构成了函数定义(function dcfnition)，该定义由两部分组成：第一行int main()叫函数头（function heading），花括号（{和}）中包括的部分叫函数体。

main()中最后一条语句叫做返回语句(return statement)，它结束该函数。

### 1．作为接口的函数头

就目前而言，C++句法要求main()函数的定义以函数头int main()开始。

通常，C++函数可被其他函数激活或调用，函数头描述了函数与调用它的函数之间的接口。位于函数名前面的部分叫做函数返回类型，它描述的是从函数返回给调用它的函数的信息。函数名后括号中的部分叫做形参列表（argument list）或参数列表（parameter list）；它描述的是从调用函数传递给被调用的函数的信息。这种通用格式用于main()时让人感到有些迷惑，因为通常并不从程序的其他部分调用main()。

然而，通常main()被启动代码调用，而启动代码是由编译器添加到程序中的，是程序和操作系统（UNIX、Windows 7或其他操作系统）之间的桥梁。事实上，该函数头描述的是main()和操作系统之间的接口。

看一下main()的接口描述，该接口从int开始。C++函数可以给调用函数返回一个值，这个值叫做返回值（retum value）。从关键字int可知，main()返叫一个整数值。接下来，是空括号。通常，C++函数在调用另一个函数时，可以将信息传递给该函数。括号中的函数头部分描述的就是这种信息。在这里，空括号意味着main()函数不接受任何信息，或者main()不接受任何参数。（main()不接受任何参数并不意味着main()是不讲道理的、发号施令的函数。相反，术语参数（argument）只是计算机人员用来表示从一个函数传递给另一个函数的信息）。

简而言之，下面的函数头表明main()函数可以给调用它的函数返回一个整数值，且不从调用它的函数那里获得任何信息：

int main()

很多现有的程序都使用经典C函数头：

main() // 经典的C风格函数头original C style

在C语言中，省略返回类型相当于说函数的类型为int。然而，C++逐步淘汰了这种用法。

也可以使用下面的变体：

int main(void) // very explicit style

在括号中使用关键字void明确地指出，函数不接受任何参数。在C++（不是C）中，让括号空着与在括号中使用void等效（在C中，让括号空着意味着对是否接受参数保持沉默）。

ANSI/ISO C++标准对那些抱怨必须在main()函数最后包含一条返叫语句过于繁琐的人做出了让步。如果编译器到达main()函数末尾时没有遇到返回语句，则认为main()函数以如下语句结尾：

return 0;

这条隐含的返回语句只适用于main()函数，而不适用于其他函数。

### 2．为什么main()不能使用其他名称

之所以将myfirst.cpp程序中的函数命名为main()，原因是必须这样做。通常，C++程序序必须包含一个名为main()的函数。由于myfirst.cpp程序只有一个函数，因此该函数必须担负起main()的责任，在运行C++程序时，通常从main()函数开始执行。因此，如果没有main()，程序将不完整，编译器将指出未定义main()函数。

## 2.1.2 C++注释

C++注释以双斜杠（//）打头，到行尾结束。注释可以位于单独的一行上，也可以和代码位于同一行。

应使用注释来说明程序。程序越复杂，注释的价值越犬。注释不仅有助于他人理解这些代码，也有助于程序员自己理解代码，特别是隔了一段时间没有接触该程序的情况下。

C++也能够识别C注释，C注释包括在符号/\*和\*/之间：

#include <iostream> /\* a C-style comment \*/

由于C-风格注释以\*/结束，而不是到行尾结束，因此可以跨越多行。可以在程序中使用C或C++风格的注释，也可以同时使用这两种注释。但应尽量使用C++注释，因为这不涉及到结尾符号与起始符号的正确配对，所以它产生问题的可能性很小。

## 2.1.3 C++预处理器和iostream文件

如果程序要使用C++输入或输出工具，需要提供这样两行代码：

#include <iostream>

using namespace std;

C++和C一样，也使用一个预处理器，该程序在进行主编译之前对源文件进行处理。不必执行任何特殊的操作来调用该预处理器，它会在编译程序时自动运行。

程序清单2.1使用了#include编译指令：

#include <iostream> // 预处理器编译指令a PREPROCESSOR directive

该编译指令导致预处理器将iostream文件的内容添加到程序中。这是一种典型的预处理器操作：在源代码被编译之前，替换或添加文本。

为什么要将iostream文件的内容添加到程序中呢？答案涉及程序与外部世界之间的通信．iostream中的io指的是输入（进入程序的信息）和输出（从程序中发送出去的信息）。C++的输入/输出方案涉及iostream文件中的多个定义。为了使用cout来显示消息，第一个程序需要这些定义。#include便以指令导致iostream文件的内容随源代码文件的内容一起被发送给编译器。实际上，iostream文件的内容将取代程序中的代码行#include <iostream>。原始文件没有被修改，而是将源代码文件和iostream组合成一个复合文件。编译的下一阶段将使用该文件。

## 2.1.4头文件名

像iostream这样的文件叫做包含文件（include file）——由于它们被包含在其他文件中；也叫头文件（header file）——由于它们被包含在文件起始处。C++编译器自带了很多头文件，每个头文件都支持一组特定的工具。C语言的传统是，头文件使用扩展名h，将其作为一种通过名称标识文件类型的简单方式。例如，头文件math.h支持各种C语言数学函数，但C++的用法变了。现在，对老式C的头文件保留了扩展名h（C++程序仍可以使用这种文件），而C++头文件则没有扩展名。有些C头文件被转换C++头文件，这些文件被重新命名，去掉了扩展名h（使之成为C++风格的名称），并在文件名称前面加上前缀c（表明来自C语言）。例如C++版本的math.h为cmath。有时C头文件的C版本和C++版本相同，而有时候新版本做了一些修改。表2.1对头文件的命名约定进行了总结。

表2.1 头文件命名约定

|  |  |  |  |
| --- | --- | --- | --- |
| **头文件类型** | **约定** | **示例** | **说明** |
| C++旧式风格 | 以.h结尾 | iostream.h | C++程序可以使用 |
| C旧式风格 | 以.h结尾 | math.h | C、C++程序可以使用 |
| C++新式风格 | 没有扩展名 | iostream | C++程序可以使用，使用namespace std |
| 转化后的C | 加上前缀c，没有扩展名 | cmath | C++程序可以使用，可以使用不是C的特性，如namespace std |

## 2.1.5名称空间

如果使用iostream，而不是iostream.h，则应使用下面的名称空间编译指令来使iostream中的定义对程序可用：

using namespace std;

名称空间的支持是一项C++特性，旨在让您编写大型程序以及将多个厂商现有的代码组合起来的程序时更容易，它还有助于组织程序。一个潜在的问题是，可能使用两个已封装好的产品，而它们都包含一个名为wanda()的函数。这样，使用wanda()函数时，编译器将不知道指的是哪个版本。名称空间让厂商能够将其产品封装在一个叫做名称空间的单元中，这样就可以用名称空间的名称来指出想使用哪个厂商的产品。因此，Microflop Industries可以将其定义放到一个名为Microflop的名称空间中。这样，其wanda()函数的全称为Microflop::wanda()；同样，Piscine公司的wanda()版本可以表示为Piscine::wanda()。这样，程序就可以使用名称空间来区分不同的版本了。

按照这种方式，类、函数和变量便是C++编译器的标准组件，它们现在都被放置在名称空间std中。仅当头文件没有扩展名h时，情况才是如此。这意味着在iostream中定义的用于输出的cout变量实际上是std::cout，而endl实际上是std::endl。因此，可以省略编译指令using，以下述方式进行编码：

std::cout << "Come up and C++ me some time.”;

std:: cout << std::endl;

然而，多数用户并不喜欢将引入名称空间之前的代码（使用iostream:h和cout）转换为名称空间代码（使iostream和std::cout），除非他们可以不费力地完成这种转换。于是，using编译指令应运而生。下面的代码表明，可以使用std名称空间中定义的名称，而不必使用std::前缀：

using namespace std;

这一using编译指令使得std名称空间中的所有名称都可用，这是一种偷懒的做法。在大型项目一个潜在的问题。更好的方法是，只使所需的名称可用，这可以通过使用using声明来实现：

using std:cout; //make cout available

using std::endl; //make endl available

using std::cin; //make cin available

用这些编译指令替换下述代码后，便可以使用cin和cout，而不必加std::前缀：

using namespace std; //lazy approach;all names avarlable

## 2.1.6使用cout进行C++输出

myfirst.cpp程序使用下面的C++语句：

cout << "Come up and C++ me some time.”;

双引号括起的部分是要打印的消息。在C++中，用双引号括起的一系列字符叫做字符串，因为它是由若干字符组合而成的。<<符号表示该语句将把这个字符串发送给cout；该符号指出了信息流动的路径。cout是一个预定义的对象，知道如何显示字符串、数字和单个字符等。

从概念上看，输出是一个流，即从程序流出的一系列字符。cout对象表示这种流，其属性是在iostream文件中定义的。cout的对象属性包括一个插入运算符（<<），它可以将其右侧的信息插入到流中。请看下面的语句（注意结尾的分号）：

Cout << “Come up and C++ me some time.”;

它将字符串“Come up and C++ me some time.”插入到输出流中。因此，与其说程序显示了一条消息，不如说它将一个字符串插入到了输出流中。

### 1．控制符endl

程序清单2.1中第二个输出流中看起来有些古怪的符号：

cout << endl;

endl是一个特殊的C++符号，表示一个重要的概念：重起一行。在输出流中插入endl将导致屏幕光标移到下一行开头。诸如endl等对于cout来说有特殊含义的特殊符号被称为控制符（manipulator）。和cout一样，endl也是在头文件iostream中定义的，且位于名称空间std中。

打印字符串时，cout不会自动移到下一行，因此在程序清单2.1中，第一条cout语句将光标留住输出字符串的后面。每条cout语句的输出从前一个输出的末尾开始，因此如果省略程序清单2.1中的endl。得到的输出将如下：

Come up and C++ me some time.You won’t regret it！

### 2．换行符

C++还提供了另一种在输出中指示换行的旧式方法：C语言符号\n：

cout << "What's next?\n"; // \n means starta new line

\n被视为一个字符，名为换行符。

显示字符串时，在字符串中包含换行符，而不是在末尾加上endl，可减少输入量：

cout << “Pluto is a dwarf planet.\n”; // show text,go to next line

cout << “PlutO is a dwarf planet.”<<endl; // show text,go to next line

另一方面，如果要生成一个空行，则两种方法的输入量相同，但对大多数人而言，输入endl更为方便。