第 2 章　C++程序的组成部分

C++程序由类、函数、变量及其他元素组成。本书的大部分内容对这些组成部分进行深入解释，但为了解程序是如何组合在一起的，必须分析一个完整的工作程序。

本章将学习：

- C++程序的组成部分，
- 各部分如何协同工作？
- 函数及其用途。

2.1　一个简单程序

即使是第1章的简单程序HELLO.cpp也有许多令人关注的组成部分。本节将对该程序进行更详细的分析。为方便起见，程序清单2.1重新列出了HELLO.cpp的最初的版本。

程序清单2.1　用HELLO.cpp说明C++程序的组成部分

```
1:  #include <iostream>
2:
3:  int main()
4:  {
5:      std::cout << "Hello World!\n";
6:      return 0;
7:  }
```

输出：

Hello World!

分析：

在第1行，文件iostream被包含到当前文件中。

下面详细分析第1行：第1个字符是符号#，这是预处理器标记。每次启动编译器时，将首先运行预处理器。预处理器浏览源代码，查找以#开头的行，在编译器运行之前处理这些行。有关预处理器的详细内容请参看第21章。

include是一条预处理器指令，指出接下来是一个文件名，请找到该文件，读取它并将其放到这里。文件名两边的尖括号告诉预处理器，在通常的所有位置查找该文件。如果编译器安装正确，尖括号将使预处理器在保存编译器的所有包含文件的目录下查找iostream文件。文件iostream（输入输出流）由cout使用，用来向屏幕输出。第1行的作用就是将iostream.h文件包含到该程序中，就好像是你自己亲自将其输入一样。

注意：每次调用编译器时预处理器总是在编译器之前运行。预处理器把任何以#开头的行都翻译成一条特殊命令，为编译器准备好代码文件。

注意：并非所有的编译器都支持在#include 语句中省略文件扩展名。如果出现错误消息，则可能需要修

改编译器的包含搜索路径，或在#include 语句中加上文件扩展名。

实际程序从第 3 行开始，它包含有一个名为 main( )的函数。每个 C++程序都有一个 main( )函数。函数是执行一项或多项操作的代码块。通常函数是由其他函数调用，但 main( )特殊。程序在开始时会自动调用 main( )。

和所有函数一样，main( )函数也必须指出其返回值类型。在 HELLO.cpp 中，main( )的返回值类型为 int，这意味着在其运行结束后将向操作系统返回一个整数。在这个程序中，它返回整数 0，如第 6 行所示。向操作系统返回一个值相对来说是一个并不重要并且很少使用的功能，但 C++标准要求必须像上面那样声明 main( )函数。

警告：有些编译器允许将 main( )的返回类型声明为 void，这已不再是合法的 C++，请不要养成这种坏习惯。让 main( )返回 int，只需返回 0 即可，就像上面 main( )中最后一行那样。

注意：有些操作系统允许用户测试程序的返回值。习惯做法是返回 0，表示程序正常结束。

所有函数都以左大括号{开始，以右大括号}结束。main( )函数的两个大括号分别在第 4 行和第 7 行。在两个大括号之间的所有代码均被看作是函数的一部分。

第 5 行是本程序的灵魂。

这行使用对象 cout 将一条消息打印到屏幕上。第 6 章将简要介绍对象，而 cout 及其相关对象 cin 将在第 17 章详细介绍。这两个对象（cin 和 cout）在 C++中分别用来处理输入（如从键盘输入）和输出（如输出到控制台）。

cout 是标准库提供的一个对象。库是一组类。标准库是每个与遵循 ANSI 标准的 C++编译器都提供的一组标准类。

你告诉编译器，你要使用的 cout 对象位于标准库中，这是使用名称空间说明符 std 指出的。由于可能有来自多个厂商同名对象，因此 C++将世界划分成名称空间。名称空间是一种分类方法，例如，如果说到 cout 时，指的是标准名称空间而不是其他名称空间中的 cout，可以使用 std::cout 向编译器指出这一点。接下来的几章将详细地讨论名称空间。

cout 的用法如下：输入 cout 和输出重定向运算符（<<）。输出重定向运算符后面的所有内容都将被输出到屏幕上。要输出一个字符串，请务必使用双引号（" "）将其括起，如第 5 行所示。

注意：需要注意的是，重定向运算符由两个小于号组成，它们之间没有空格。

文本字符串是一系列可打印的字符。

最后两个字符（\n）告诉 cout，在 Hello World!之后换行。这个特殊代码将在第 18 章讨论 cout 时详细介绍。

第 7 行的右大括号是 main( )函数结束的标记。

2.2　cout 简介

在第 17 章，你将学习如何使用 cout 将数据显示到屏幕上。就现在而言，你可直接使用它，而不必关心其工作原理。要将一个值输出到屏幕，只需写 cout，后面跟上插入运算符（<<）。插入运算符（<<）由两个小于号（<）组成，但 C++将其作为一个字符看待。

最后在插入运算符后输入要输出的数据。程序清单 2.2 演示了如何使用 cout。准确地按程序清单输入，并将其中的 Jesse Liberty 改成你的名字。当然，如果你也叫 Jesse Liberty，那就不用了。

程序清单 2.2　使用 cout

```
1: // Listing 2.2 using std::cout
```

```
2:  #include <iostream>
3:  int main()
4:  {
5:      std::cout << "Hello there.\n";
6:      std::cout << "Here is 5: " << 5 << "\n";
7:      std::cout << "The manipulator std::endl ";
8:      std::cout << "writes a new line to the screen.";
9:      std::cout <<  std::endl;
10:     std::cout << "Here is a very big number:\t" << 70000;
11:     std::cout << std::endl;
12:     std::cout << "Here is the sum of 8 and 5:\t";
13:     std::cout << 8+5 << std::endl;
14:     std::cout << "Here's a fraction:\t\t";
15:     std::cout << (float) 5/8 << std::endl;
16:     std::cout << "And a very very big number:\t";
17:     std::cout << (double) 7000 * 7000 << std::endl;
18:     std::cout << "Don't forget to replace Jesse Liberty ";
19:     std::cout << "with your name...\n";
20:     std::cout << "Jesse Liberty is a C++ programmer!\n";
21:     return 0;
22: }
```

输出：

```
Hello there.
Here is 5: 5
The manipulator endl writes a new line to the screen.
Here is a very big number:      70000
Here is the sum of 8 and 5:     13
Here's a fraction:              0.625
And a very very big number:     4.9e+007
Don't forget to replace Jesse Liberty with your name...
Jesse Liberty is a C++ programmer!
```

警告：有些编译器存在错误，要求在加法操作数两边加上圆括号，即将第13行改为：

```
13:      cout << (8+5) << std::endl;
```

分析：

在第2行中，#include<iostream>将iostream文件加入到源代码中。使用cout及其相关函数时必须这样做。

第5行是cout最简单的用法：打印字符串（一系列字符）。\n是一个特殊的格式符，它告诉cout另起行。

在第6行中，3个值被传递给cout，它们由插入运算符分开。第1个值是字符串"Here is 5: "。请注意冒号后的空格，它也是字符串的组成部分。接下来，将5传递给插入运算符，然后传递换行符（总是在双引号或单引号中）。这将导致将下述内容显示到控制台：

```
Here is 5: 5
```

由于在第1个字符串后没有换行符，因此第2个值紧接着第1个值显示。这被称为串接两个值。

第7行显示了一条提示性消息，然后程序使用了控制符std::endl。endl的作用是另起一行（endl的其他用途将在第16章讨论）。注意，endl也是由标准库提供的，因此在它前面加上了std::，就像在cout前面要加上std::一样。

注意：endl表示end line，是end-ell而不是end-one。通常它念作“end-ell”。

endl比'\n'更好，因为endl将适应当前使用的操作系统，而在某些OS或平台上，'\n'可能不是完整的换

行符。

第 10 行使用了一个新的格式化字符“\t”，它插入一个制表符。第 10 至 16 行使用它来将输出对齐。第 10 行表明，不仅可显示整型数据，也可显示长整型数据。第 13 和 14 行表明，cout 还可进行简单加法。在第 14 行中，将 8+5 的值传递给 cout，而输出是 13。

第 15 行将 5/8 的值插入到 cout 中。(float)告诉 cout，应计算小数，因此在屏幕上显示出一个小数。第 17 行将 7000*7000 的值传递给 cout，(double)告诉 cout 这是个浮点数。第 3 章讨论数据类型时，将介绍这些数据类型。

在第 18 行和第 20 行，你应用自己的名字代替 Jesse Liberty，这样，输出结果将确认你是一个真正的 C++ 程序员。这肯定是真的，因为计算机是这么说的。

2.3　使用标准名称空间

你将发现，在 count 和 endl 前面使用 std::非常烦人。尽管使用名称空间指示是一种很好的方式，但大量的输入很讨厌。ANSI 标准提供了两种方法来解决这个小问题。

第一种方法是，在代码清单的开头告诉编译器，你将使用标准库 count 和 endl，如程序清单 2.3 的第 5 和 6 行所示。

程序清单 2.3　使用关键字 using

```
1:  // Listing 2.3 - using the using keyword
2:  #include <iostream>
3:  int main()
4:  {
5:      using std::cout;
6:      using std::endl;
7:
8:      cout << "Hello there.\n";
9:      cout << "Here is 5: " << 5 << "\n";
10:     cout << "The manipulator endl ";
11:     cout << "writes a new line to the screen.";
12:     cout << endl;
13:     cout << "Here is a very big number:\t" << 70000;
14:     cout << endl;
15:     cout << "Here is the sum of 8 and 5:\t";
16:     cout << 8+5 << endl;
17:     cout << "Here's a fraction:\t\t";
18:     cout << (float) 5/8 << endl;
19:     cout << "And a very very big number:\t";
20:     cout << (double) 7000 * 7000 << endl;
21:     cout << "Don't forget to replace Jesse Liberty ";
22:     cout << "with your name...\n";
23:     cout << "Jesse Liberty is a C++ programmer!\n";
24:     return 0;
25: }
```

输出：

```
Hello there.
Here is 5: 5
The manipulator endl writes a new line to the screen.
```

```
Here is a very big number:      70000
Here is the sum of 8 and 5:     13
Here's a fraction:              0.625
And a very very big number:     4.9e+007
Don't forget to replace Jesse Liberty with your name...
Jesse Liberty is a C++ programmer!
```

分析：

输出与前一个程序清单相同。程序清单 2.3 和 2.2 的惟一差别是：在第 5 和第 6 行，添加了告诉编译器将使用标准库中两个对象的语句，这是使用关键字 using 来完成的。这样，就不再需要限定 count 和 endl 对象了。

第二种解决办法是，告诉编译器程序将使用整个标准名称空间；也就是说，除非特别说明，否则任何对象都来自标准名称空间。在这种情况下，应使用 using namespace std;而不是 using std::count;，如程序清单 2.4 所示。

程序清单 2.4　使用关键字 namespace

```
1:  // Listing 2.4 - using namespace std
2:  #include <iostream>
3:  int main()
4:  {
5:      using namespace std;
6:
7:      cout << "Hello there.\n";
8:      cout << "Here is 5: " << 5 << "\n";
9:      cout << "The manipulator endl ";
10:     cout << "writes a new line to the screen.";
11:     cout << endl;
12:     cout << "Here is a very big number:\t" << 70000;
13:     cout << endl;
14:     cout << "Here is the sum of 8 and 5:\t";
15:     cout << 8+5 << endl;
16:     cout << "Here's a fraction:\t\t";
17:     cout << (float) 5/8 << endl;
18:     cout << "And a very very big number:\t";
19:     cout << (double) 7000 * 7000 << endl;
20:     cout << "Don't forget to replace Jesse Liberty ";
21:     cout << "with your name...\n";
22:     cout << "Jesse Liberty is a C++ programmer!\n";
23:     return 0;
24: }
```

分析：

同样，输出也与该程序的以前版本相同。使用 using namespace std;的优点是，不再需要指定实际使用的对象（如 count 和 endl）；缺点是可能不小心地使用了错误库中的对象。

纯正论者喜欢在 count 和 endl 的前面使用 std::；而懒人喜欢使用 using namespace std;。在本书中，大部分情况下将使用 using 来声明要使用的名称，但处于好玩，也偶尔会使用一下另一种风格。

2.4　对程序进行注释

当你编写程序时，你的意图对你来说是清晰和不言而喻的。有趣的是，一个月后，你再来看你的程序，

将发现令人迷惑。没有人知道为何会这样，但这种事情经常发生。

为避免自己迷惑，同时帮助他人理解你的程序，应使用注释。注释是被编译器忽略的文本，但可以告诉读者，程序的某个地方想做什么。

2.4.1　注释的类型

C++注释有两种：单行注释和多行注释。

单行注释使用双斜杠（//）来表示。双斜杠告诉编译器，忽略之后到行尾的所有内容。

多行注释以斜杠和星（/*）打头。这种注释标记告诉编译器，忽略之后到星和斜杠（*/）之间的所有内容。这两种注释标记可以位于同一行，它们之间也可以有一行或多行，但每个/*必须有与之匹配的*/。

很多 C++程序员在大多数时候使用双斜杠型单行注释，将多行注释标记用于将程序中的大型代码块注释掉。在使用多行注释标记注释掉的代码块中，可以包含单行注释。多行注释标记之间的所有内容（包括双斜杠型注释）都将被忽略。

注意：多行注释被称为 C-风格注释，因为它是 C 编程语言引入和使用的。单行注释最初是 C++而不是 C 语言的组成部分，因此被称为 C++-风格注释。最新的 C 和 C++标准都支持这两种注释风格。

2.4.2　使用注释

有些人建议在函数的开头编写注释，说明函数的功能和返回值。

函数应该有一个明确说明其功能的名称，那些令人迷惑的代码都应重新设计和重新编写，使其含义是不言而喻的。不应将注释用作编写晦涩代码的借口。

这并不是说绝不要使用注释，只是不应依赖注释来阐明晦涩的代码，而应修改这样的代码。总之，应编写良好的代码，将注释作为解释代码的辅助工具。

程序清单 2.5 演示注释的用法，表明了注释对程序的运行和结果没有任何影响。

程序清单 2.5　演示注释用法的 HELP.cpp

```
1:  #include <iostream>
2:
3:  int main()
4:  {
5:      using std::cout;
6:
7:      /* this is a comment
8:      and it extends until the closing
9:      star-slash comment mark */
10:     cout << "Hello World!\n";
11:     // this comment ends at the end of the line
12:     cout << "That comment ended!\n";
13:
14:     // double-slash comments can be alone on a line
15:     /* as can slash-star comments */
16:     return 0;
17: }
```

输出：

```
Hello World!
That comment ended!
```

分析：

第 7～9 行以及第 11、14、15 行的注释都被编译器忽略。第 11 行的注释到行尾结束，而第 7 行和第 15

行的注释需要注释结束标记。

注意：有些编译器支持第三种注释方式，这种注释被称为文档注释，用 3 个斜杠（///）进行标记。支持这种注释的编译器让你能够使用这些注释生成有关程序的文档。由于这种注释当前不是 C++标准的组成部分，因此这里不介绍它。

2.4.3 有关注释的警告

对明显的事情进行注释往往没有多大意义。事实上，注释可能降低效率，因为代码可能被修改，而程序员往往忘记修改注释。然而，一个人觉得很明显的事情，在另一个人看来也许是晦涩的，因此，在什么情况下添加注释需要一定的判断力。

通用规则是，注释不应说明发生了什么事情，而应说明为什么会发生这种事情。

2.5 函　　数

尽管 main( )是一个函数，但它并不是通常意义上的函数。函数只有在程序执行时被调用才能起作用，而 main( )函数由操作系统调用。

通常，如果没有遇到函数，程序将依次逐行地执行源代码。遇到函数后，程序将执行该函数。函数结束后，程序又返回到调用该函数代码行的下一行继续执行。

函数的作用可用削铅笔来比喻。画图时，如果铅笔突然断了，你可能先停止画图去削铅笔，削好铅笔后再继续画图。程序需要某项服务时，可以调用一个函数来执行这项服务，待函数执行完后，再返回到原来的地方继续执行。程序清单 2.6 说明了这一点。

注意：第 5 章将更详细地介绍函数；函数可返回的数据类型将在第 3 章详细介绍。本章只简要地介绍函数，因为几乎所有的 C++程序都需要用到函数。

程序清单 2.6　演示函数调用

```
 1: #include <iostream>
 2:
 3: // function Demonstration Function
 4: // prints out a useful message
 5: void DemonstrationFunction()
 6: {
 7:     std::cout << "In Demonstration Function\n";
 8: }
 9:
10: // function main - prints out a message, then
11: // calls DemonstrationFunction, then prints out
12: // a second message.
13: int main()
14: {
15:     std::cout << "In main\n" ;
16:     DemonstrationFunction();
17:     std::cout << "Back in main\n";
18:     return 0;
19: }
```

输出：

```
In main
```

```
In Demonstration Function
Back in main
```

分析：

第 6～8 行定义了函数 DemostrationFunction( )。被调用时，它将一条消息打印到屏幕然后返回。

实际程序开始于第 13 行。在第 15 行中，main( )函数打印了一条消息，指出当前位于 main( )函数中。打印这条消息后，第 16 行调用 DemostrationFunction( )函数。该调用导致程序流程跳转到第 5 行的函数 DemostrationFunction( )，进而执行该函数中的所有命令。在这个程序中，整个函数由第 7 行的代码组成，它打印另一条消息。DemostrationFunction( )执行完毕后（第 8 行），程序返回到函数调用的下一行，这里为第 17 行。在这一行，main( )打印最后一条消息。

2.5.1　使用函数

函数要么返回一个值，要么返回 void，后者意味着什么也不返回。将两个整数相加的函数可能返回它们的和，因此需要将其定义为返回一个整型值。如果函数只打印一条消息而不返回任何东西，应将其定义为返回 void。

函数由函数头和函数体组成，而函数头又由返回类型、函数名和参数组成。函数参数让你能够将值传递给函数。因此，如果函数将两个数相加，则这两个数都作为函数的参数。下面是一个典型的函数头，它声明了一个名为 Sum 的函数，该函数接受两个整数值（first 和 second），并返回一个整数值：

```
int Sum(int first, int second)
```

参数用于声明要传入的值的类型；调用函数时实际传入的值被称为实参。很多程序员都将参数和实参看做同义词；另一些人则将它们分开。对于 C++编程而言，参数和实参之间的区别并不重要，因此看到这两个词被交替使用时，读者不用担心。

函数主体由左大括号、零条或更多的语句以及右大括号组成。函数的功能由语句实现。

函数可能使用 return 语句来返回一个值。返回的值必须是函数头中声明的类型。另外，该语句导致函数结束。如果函数中不包括 return 语句，它将在结束时自动返回 void。如果函数被声明为返回一个值，但没有 return 语句，有些编译器将发出警告或错误消息。

程序清单 2.7 演示了一个接受两个整型参数并返回一个整型值的函数。读者不必考虑其中语法以及如何使用整型值（如 int first）的细节，这些内容将在第 3 章详细介绍。

程序清单 2.7　一个简单函数

```
 1: #include <iostream>
 2: int Add (int first, int second)
 3: {
 4:     std::cout << "In Add(), received " << first << " and
        ↪ " << second << "\n";
 5:     return (first + second);
 6: }
 7:
 8: int main()
 9: {
10:     using std::cout;
11:     using std::cin;
12:
13:
14:     cout << "I'm in main()!\n";
15:     int a, b, c;
16:     cout << "Enter two numbers: ";
17:     cin >> a;
```

```
18:     cin >> b;
19:     cout << "\nCalling Add()\n";
20:     c=Add(a,b);
21:     cout << "\nBack in main().\n";
22:     cout << "c was set to " << c;
23:     cout << "\nExiting...\n\n";
24:     return 0;
25: }
```

输出：

```
I'm in main()!
Enter two numbers: 3 5

Calling Add()
In Add(), received 3 and 5

Back in main().
c was set to 8
Exiting...
```

分析：

第 2 行定义了 Add( )函数。它接受两个整型参数并返回一个整型值。程序本身从第 8 行开始。在第 16 行，程序提示用户输入两个数。用户输入两个数（用空格将两数分开），然后按回车键。用户输入的数字被存储在第 17 和 18 行的变量 a 和 b 中。在第 20 行，main( )将用户输入的两个数作为参数传递给 Add( )函数。

程序跳到从第 2 行开始的 Add( )函数处执行。该函数通过参数 first 和 second 收到存储在 a 和 b 中的值，然后将它们打印并相加。第 5 行返回结果，同时返回到调用它的函数——这里为 main( )。

在第 17 和 18 行，对象 cin 被用来获取变量 a、b 的值；在程序余下的地方，cout 被用来输出到控制台。变量及该程序的其他方面将在接下来的几章深入介绍。

2.5.2 方法和函数

无论使用何种称呼，函数始终是函数。需要注意的是，不同的语言和编程方法学可能使用不同的术语来称呼函数。一个较常用的术语是方法。方法是类中函数的另一种称呼。

2.6 小　　结

学习复杂课题（如编程）的困难在于，正在学习的东西又依赖于其他有待学的东西。本章介绍了 C++程序的基本组成部分。

2.7 问 与 答

问：#include 的作用是什么？

答：这是一个预处理器编译指令。预处理器在你调用编译器时运行。该指令使得预处理器将#include 后面的<>中的文件读入程序，其效果如同将这个文件输入到源代码中的这个位置。

问：//注释和/*注释之间有何不同？

答：//注释到行尾结束；/*注释到*/结束。//注释也被称为单行注释，/*注释通常被称为多行注释。请记住，即使是函数的结尾也不能作为/*注释的结尾，必须加上注释结尾标记*/，否则将出现编译阶段错误。

问：好注释与坏注释的区别在哪里？

答：好注释告诉读者某段代码是如何工作的或它将要做什么；而坏注释只说明某行代码做什么。代码行的含义应不言而喻，良好的代码行应该不用注释就能让读者明白其功能。

2.8 作　　业

作业包括测验和练习，前者帮助加深读者对所学知识的理解，后者提供了使用新学知识的机会。请尽量先完成测验和练习题，然后再对照附录 D 中的答案，继续学习下一章之前，请务必弄懂这些答案。

2.8.1 测验

1. 编译器和预处理器之间有何不同？
2. 为什么函数 main( )是特殊的？
3. 有哪两种注释？它们之间有何不同？
4. 注释能否嵌套？
5. 注释可以长于一行吗？

2.8.2 练习

1. 编写一个程序，将“I love C++”打印到控制台。
2. 编写一个可以编译、链接和运行的最短程序。
3. 查错：输入下面的程序并编译它。它为什么不能通过编译？如何修复？

```
1: #include <iostream>
2: main()
3: {
4:     std::cout << Is there a bug here?";
5: }
```

4. 修复练习 3 中的错误，然后重新编译、链接并运行它。
5. 修改程序清单 2.7，以包含一个减法函数。将该函数命名为 Subtract( )，并像使用 Add( )函数那样使用它。另外，应将传递给函数 Add( )的值传递给该函数。