第 2 章

C++程序结构

学习指引

C 语言是结构化和模块化的面向过程的语言，C++语言是面向对象的程序设计语言，两者的区别主要在于编程思想。因为 C 是基于过程的，强调的是程序的功能，以功能为中心。而 C++是面向对象的，强调程序的分层、分类，以抽象为基础，进行对象的定义与展示，即程序设计。

本章将详细介绍 C++程序结构，主要内容包括：C++的程序组成结构、命名空间、输入与输出等。

重点导读

- 熟悉并掌握函数的主体以及预编译指令。
- 熟悉代码注释。
- 熟悉命名空间。
- 熟悉并掌握输入输出。

2.1 Hello C++程序的组成结构

学习编程是一个由易到难的过程，先编写一个最简单的程序，了解 C++语言的基本组成结构。本节将以第 1 章中“ Hello C++ ”为例。

【例 2-1】编写程序，输出“ Hello C++! ”。

（1）在 Visual Studio 2017 中，新建名称为“ 2-1.cpp ”的 Project1 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>                    /*包含标准输入输出库*/
using namespace std;                   /*使用命名空间 std*/
int main()                             /*定义主函数*/
{
   cout << "Hello C++!" << endl;       /*向标准输入输出设备输出字符串*/
   return 0;                           /*返回值*/
}
```

【程序分析】在运行程序时会在屏幕上输出一行信息：“ Hello C++ ! ”。而组成这段程序可以分为三个部分：

（1）在程序开头用 main 代表“ 主函数 ”的名字。每一个 C++程序都必须有一个 main()函数。main 前面的 int 的作用是声明函数的类型为整型。程序第 6 行的作用是向操作系统返回一个零值。如果程序不能正常执行，则会自动向操作系统返回一个非零值，一般为-1。函数体是由大括号{}括起来的。本例中主函数内只有一个以 cout 开头的语句。

注意：C++所有语句最后都应当有一个分号。

（2）在程序的第 1 行有“ #include <iostream> ”，这不是 C++的语句，而是 C++的一个预处理指令，它以“ # ”开头来与 C++语句相区别，行的末尾没有分号。

#include <iostream>是一个“ 包含命令 ”，它的作用是将文件 iostream 的内容包含到该命令所在的程序文件中，代替该命令行。文件 iostream 的作用是向程序提供输入或输出时所需要的一些信息。iostream 是 i-o-stream 三个词的组合，从它的形式就可以知道它代表“ 输入输出流 ”的意思，由于这类文件都放在程序单元的开头，所以称为“ 头文件 ”（head file）。在程序进行编译时，先对所有的预处理命令进行处理，将头文件的具体内容代替#include 命令行，然后再对该程序单元进行整体编译。

（3）程序的第 2 行“ using namespace std; ”的意思是“ 使用命名空间 std ”。C++标准库中的类和函数是在命名空间 std 中声明的，因此程序中如果需要用到 C++标准库（通过#include 命令行来调用），就需要用“ using namespace std; ”作声明，表示要用到命名空间 std 中的内容。

在 Visual Studio 2017 中的运行结果如图 2-1 所示。

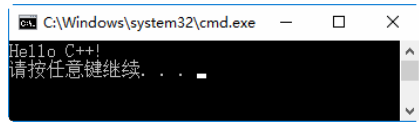

图 2-1　Hello C++的组成结构

在初学 C++时，对本程序中的第 1，2 行可以不必深究，只需知道：如果程序有输入或输出时，必须使用“ #include <iostream> ”命令以提供必要的信息，同时要用“ using namespace std; ”，使程序能够使用这些信息，否则程序编译时将出错。

2.2　预处理器编译指令#include

预处理编译是 C++组织程序的工具。在程序运行前，预编译器将编译好指令发送给预处理器，为了识别这条指令，会在前面加上字符“ # ”。预编译处理指令不是 C++语言中的语句。在上述程序中，“ #include<iostream> ”的作用是在编译之前将文件 iostream 的内容增加（包含）到程序中，以作为其中一部分。

iostream 是系统定义的一个头文件，在该文件里设置了 C++的 I/O 相关环境，定义输入输出流对象 cout 与 cin 等。采用预处理指令的目的，在于增强和扩展语言编程的环境，为编程设计人员提供更为方便的编程手段。

2.3　程序的主体——main()

预处理器编译指令的后面是程序的主体 main()函数。单词 main 代表主函数的意思，main()函数是程序执行的入口，程序从 main()函数的第一条指令开始执行，直到 main()函数结束，同时整个程序也将执行结束。注意函数的格式，在 main 后面有个小括号“ () ”，小括号内是放参数的地方。

main()函数是组成程序最基本的部分，在声明 main()函数时，总是要在它前面加上返回类型，这是一种

标准化约定，表示 main()函数是没有返回类型的函数。函数的类型就是它的返回值，函数处理结果的返回值由 return 语句给出。

任一函数的描述都是包括在一对“ { ”和“ } ”中的语句序列，每个语句以“ ; ”结束。C++中严格区分大小写，但不严格限制程序的书写格式，不过从可读性角度出发，程序书写应采用内缩格式，一般一个语句占一行。

注意：在很多 C++应用程序中，都使用了类似于下面的 main()函数变种。

例如：

```
int main(int argc,char* argv[])
```

这符合标准且可以接受，因为其返回类型为 int 型。括号里的内容是提供给程序的参数。该程序允许用户执行时提供命令行参数。

2.4 返回值 return

return 表示从被调函数返回到主调函数继续执行，返回时可附带一个返回值，由 return 后面的参数指定。函数可以有返回值也可以没有返回值，当没有返回值时，函数类型声明为 void 型。每个函数都有类型，如果在定义中没有给出类型则默认为 int 型。main()也是函数，并且其返回值总是一个整数。

return 通常是必要的，因为函数调用的时候计算结果通常是通过返回值带出的。如果函数执行不需要返回计算结果，也经常需要返回一个状态码来表示函数执行的顺利与否（-1 和 0 就是最常用的状态码），主调函数可以通过返回值判断被调函数的执行情况。根据约定，编程人员在程序运行成功时返回 0，并在出现错误时返回-1。然而，返回值若是整数，则编程人员可利用整个整数范围，指出众多不同的成功和失败状态。

return 的语法格式如下：

```
return 表达式;
```

函数的计算结果通过该语句传递回主调函数。函数体内可以没有 return 语句，当需要在程序指定位置退出时，可以在该处放置一个“ return ; ”。

2.5 命名空间

所谓命名空间，就是一个由程序设计者命名的内存区域，程序设计者可以根据需要指定一些有名字的空间域，把一些全局实体分别放在各个命名空间中，从而与其他全局实体分隔开来。

2.5.1 命名空间的意义

假设这样一种情况，当一个班上有两个名叫张三的学生时，为了明确区分他们，我们在使用名字之外，不得不使用一些额外的信息，比如他们的家庭住址，或者他们父母的名字等。

同样的情况也出现在 C++应用程序中。例如，可能会写一个名为 fun()的函数，在另一个可用的库中也存在一个相同的函数 fun()。这样，编译器就无法判断用户所使用的是哪一个 fun()函数。

因此，引入命名空间这个概念，其实是为了避免变量或函数重名的问题。因为一个项目组内多个工程

师进行开发，有可能会出现全局变量或函数重名的现象，而如果每个人都定义了自己的命名空间，就可以解决这个问题，即使重名，只要分属不同的命名空间就不会出现问题。

所以，从本质上讲，命名空间就是定义了一个范围，将多个变量和函数等包含在内，使其不会与命名空间以外的任何变量和函数等发生重名的冲突。

【例 2-2】编写程序，在命名空间里定义函数。

（1）在 Visual Studio 2017 中，新建名称为“ 2-2.cpp ”的 Project2 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
namespace first_name       /*第一个命名空间*/
{
   void fun()
   {
       cout << "第一个命名空间所包含的内容" << endl;
   }
}
namespace second_name      /*第二个命名空间*/
{
   void fun()
   {
       cout << "第二个命名空间所包含的内容" << endl;
   }
}
int main()
{
   first_name::fun();      /*调用第一个命名空间中的函数*/
   second_name::fun();     /*调用第二个命名空间中的函数*/
   return 0;
}
```

【程序分析】第一个命名空间 first_name 所包含的函数名为 fun；第二个命名空间 second_name 所包含的函数名也为 fun。但在主函数中调用时，编译器是许可的。

在 Visual Studio 2017 中的运行结果如图 2-2 所示。

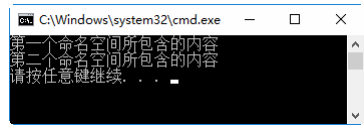

图 2-2　命名空间的调用

为了避免同名混淆，使用命名空间可以起到相互分隔的作用，把一些全局实体分隔开来。C++可以根据需要设置多个命名空间，每个命名空间名代表一个不同的命名空间域，但是不同的命名空间不能同名。

这样，可以把不同的库中的实体放到不同的命名空间中，或者说，用不同的命名空间把不同的实体隐蔽起来。过去我们用的全局变量可以理解为全局命名空间，独立于所有有名的命名空间之外，它是不需要用 namespace 声明的，实际上是由系统隐式声明的，存在于每个程序之中。

2.5.2　命名空间的用法

1. namespace 的声明

C++语言引入命名空间这一概念主要是为了避免命名冲突，其关键字为 namespace。

```
namespace first_name       /* first_name 表示命名空间的名称*/
{
   void fun();             //声明函数 fun()
}
```

为了调用带有命名空间的函数或变量，需要在前面加上命名空间的名称。

例如：

```
first_name::fun();   //也可以是变量
```

注意：指定所使用的变量时需要用到“ : : ”操作符，“ : : ”操作符是域解析操作符。

2. std 标准命名空间

标准 C++库的所有的标识符都是在一个名为 std 的命名空间中定义的。std 是一个类（输入输出标准），它包括了 cin 成员和 cout 成员，执行“ using namespace std ; ”语句后才能使用它的成员。

而#include<iostream>包含了 std 这个类。在类的使用之前需要预处理一下，代码才可以使用 cin,cout 这两个成员函数。如果不使用预处理“ using namespace std; ”，需要加上 std::cin 或者 std::cout 再去使用它的成员函数。

std 是 standard（标准）的缩写，表示这是存放标准库的有关内容的命名空间，含义清楚，不必死记。这样，在程序中用到 C++标准库时，需要使用 std 作为限定。

（1）使用“ using namespace std; ”

```
#include <iostream>
using namespace std;
int main()
{
   cout<<"This is a C++ program"<<endl;
   return 0;
}
```

（2）不使用“ using namespace std; ”

```
#include <iostream>
int main()
{
   std::cout<<" This is a C++ program"<<std::endl;
   return 0;
}
```

2.6　C++代码中的注释

程序的注释是解释性语句，C++代码中允许包含注释，这将提高源代码的可读性。所有的编程语言都允许某种形式的注释。C++支持单行注释和多行注释。注释中的所有字符会被 C++编译器忽略。

用“ // ”作注释时，有效范围只有一行，即本行有效，不能跨行。而用“ /*……*/ ”作注释时有效范围为多行。只要在开始处有一个“ /* ”，在最后一行结束处有一个“ */ ”即可。因此，一般习惯是内容较少的简单注释常用“ // ”，内容较长的常用“ /*……*/ ”。

例如：

```
//单行注释
/*多行注释 1
多行注释 2*/
```

例如：

```
#include <iostream>
using namespace std;
int main()
{
```

```
    cout << "Hello C++!" << endl;   //输出 Hello C++!
    return 0;
}
```

在“ /* ”和“ */ ”注释内部，“ // ”字符没有特殊的含义。在“ // ”注释内，“ /* ”和“ */ ”字符也没有特殊的含义。因此，可以在一种注释内嵌套另一种注释。

例如：

```
/* 用于输出 Hello C++!的注释
cout << "Hello C++!" << endl;    //输出 Hello C++!
*/
```

2.7　C++函数

函数能够将应用程序划分成多个功能单元，并且通过选择实现调用。在函数被调用时，通常会有一个值返回给调用它的函数。

【例 2-3】编写程序，完成一个函数的调用。

（1）在 Visual Studio 2017 中，新建名称为“ 2-3.cpp ”的 Project3 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
int fun();    //声明函数
int fun()     //定义函数

{
    cout << "这是阳光明媚的一天!" << endl;
    cout << "5+5=" << 5+5 << endl;
    return 0;
}
int main()
{
    fun();     //调用函数
    return 0;
}
```

【程序分析】本例中定义了一个函数，其函数名为 fun()，返回类型为 int，展现了声明函数，调用函数，最后输出结果的过程。这个函数简单演示了 cout 的功能，既可以显示文本，还可以显示简单算术运算的结果。

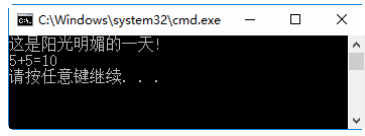

图 2-3　程序运行结果

在 Visual Studio 2017 中的运行结果如图 2-3 所示。

因为在定义函数 fun()的类型时是 int 整型，所以 fun()函数必须返回一个整数（这里返回的是 0）。同样，main()函数也返回 0。但是，由于 main()函数将其所有的任务都交给了函数 fun()去完成，所以更明智的做法是在 main()函数中返回该函数的返回值。

【例 2-4】编写程序，完成一个函数的调用。

（1）在 Visual Studio 2017 中，新建名称为“ 2-4.cpp ”的 Project4 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
int fun()
```

```
{
    cout << "这是阳光明媚的一天!" << endl;
    cout << "5+5=" << 5+5 << endl;
    return 0;
}
int main()
{
    return fun();     /*返回 fun()函数的返回值*/
}
```

【程序分析】该代码的输出与【例 2-1】相同，但编写方式存在细微差别。首先在 main()函数前定义了函数 fun()，因此无须声明该函数。另外，main()函数中直接调用 fun()函数，并将该函数的返回值作为 main()函数的返回值，使主函数更加简短，调用过程如图 2-4 所示。

在 Visual Studio 2017 中的运行结果如图 2-5 所示。

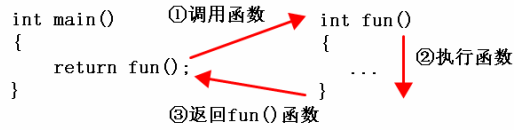

图 2-4　函数调用

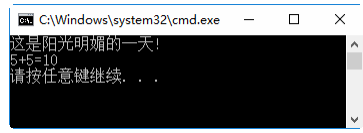

图 2-5　程序运行结果

注意：在函数无须做任何决策，也无须返回成功和失败状态时，可将其返回类型声明为 void 型，例如：“ void fun(); ”。

2.8　输入与输出

计算机与用户进行交互的过程中，数据的输入和输出是必不可少的操作过程。在 C 语言中，通常会使用函数 scanf()、printf()来对数据进行输入输出操作。而在 C++语言中，C 语言的这一套输入输出库仍然能使用，但是 C++又增加了一套新的、更容易使用的输入输出库。

由于输入和输出并不是 C++语言中的正式组成成分，并且 C 和 C++本身都没有为输入和输出提供专门的语句结构。所以 C++的输入输出发生在流中，流是字节序列。如果字节流是从设备（如键盘、磁盘驱动器、网络连接等）流向内存，这叫作输入操作。如果字节流是从内存流向设备（如显示屏、打印机、磁盘驱动器、网络连接等），这叫作输出操作。C++的输入输出流程图，如图 2-6 所示。

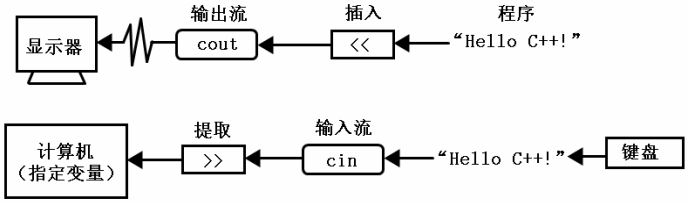

图 2-6　输入输出流程图

2.8.1　标准输出流 cout

预定义的对象 cout 是 iostream 类的一个实例。cout 对象连接到标准输出设备，通常是显示屏。cout 是与流插入运算符 “ << ” 结合使用的。

【例 2-5】编写程序，完成一个数据的输出。

（1）在 Visual Studio 2017 中，新建名称为 “ 2-5.cpp ” 的 Project5 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
int main()
{
   int a=5;
   float b=3.5;
   cout << "请输出一个整数: a=" << a << endl;
   cout << "请输出一个小数: b=" << b << endl;
   system("pause");
   return 0;
}
```

【程序分析】本程序定义了一个 int 型变量 a 并赋值为 5, 和一个 float 型变量 b，赋值为 3.5。

在 Visual Studio 2017 中的运行结果如图 2-7 所示。

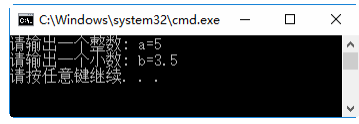

图 2-7　标准输出流 cout

2.8.2　标准输入流 cin

预定义的对象 cin 是 iostream 类的一个实例。cin 对象附属到标准输入设备，通常是键盘。cin 是与流提取运算符 “ >> ” 结合使用的，如下所示：

【例 2-6】编写程序，完成一个数据的输入。

（1）在 Visual Studio 2017 中，新建名称为 “ 2-6.cpp ” 的 Project6 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
int main()
{
   int a;
   float b;
   cout << "请输入一个整数：" << endl;
   cin >> a;
   cout << "整数 a= " << a << endl;
   cout << "请输入一个小数：" << endl;
   cin >> b;
   cout << "小数 b= " << b << endl;
   return 0;
}
```

【程序分析】本程序定义了两个变量，一个是 int 型的 a，另一个是 float 型的 b。然后通过 cin 为这两个变量赋值并输出。

在 Visual Studio 2017 中的运行结果如图 2-8 所示。

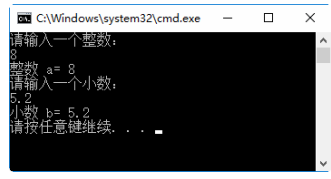

图 2-8　标准输入流 cin

C++中的输入与输出可以看作是一连串的数据流，输入即可视为从文件或键盘中输入程序中的一串数据流，而输出则可以视为从程序中输出一连串的数据流到显示屏或文件中。

在编写 C++程序时，如果需要使用输入输出时，则需要包含头文件 iostream，它包含了用于输入输出的对象，例如常见的 cin 表示标准输入、cout 表示标准输出、cerr 表示标准错误。

注意：iostream 是 Input Output Stream 的缩写，意思是“ 输入输出流 ”。

cout 和 cin 都是 C++的内置对象，而不是关键字。C++库定义了大量的类（Class），开发者可以使用它们来创建对象，cout 和 cin 就分别是 ostream 和 istream 类的对象，只不过它们是由标准库的开发者提前创建好的，可以直接拿来使用。这种在 C++中提前创建好的对象称为内置对象。

使用 cout 进行输出时需要紧跟“ << ”运算符，使 cin 进行输入时需要紧跟“ >> ”运算符，这两个运算符可以自行分析所处理的数据类型，因此无须像使用 scanf 和 printf 那样给出格式控制字符串。

注意：endl 最后一个字符是字母“ l ”，而非阿拉伯数字“ 1 ”，它是 end of line 的缩写。

【例 2-7】编写程序，同时输入一个整数和小数。

（1）在 Visual Studio 2017 中，新建名称为“ 2-7.cpp ”的 Project7 文件。

（2）在代码编辑区域输入以下代码。

```
#include <iostream>
using namespace std;
int main()
{
   int a;
   float b;
   cout << "请输入一个整数和小数:" << endl;
   cin >> a >> b;
   cout << "整数 a= " << a << endl;
   cout << "小数 b= " << b << endl;
   return 0;
}
```

【程序分析】在程序的第 8 行代码表示从标准输入（键盘）中读入一个 int 型的数据并存入到变量 a 中。如果此时用户输入的不是 int 型数据，则会被强制转化为 int 型数据。在第 9 行代码将输入的整型数据输出。从该语句中我们可以看出 cout 能够连续地输出。同样 cin 也是支持对多个变量连续输入的。

在 Visual Studio 2017 中的运行结果如图 2-9 所示。

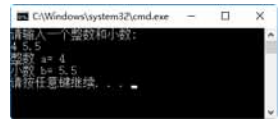

图 2-9　输入整数和小数

2.9　就业面试技巧与解析

2.9.1　面试技巧与解析（一）

面试官：#include 的作用是什么？

应聘者：这是一个预处理器编译指令，总是以字符#打头。预处理器在调用编译器时，该指令使得预处理器将 include 后面的<>中的文件读入程序。就是事先把后面需要使用的文件在开头处就定义了。

2.9.2　面试技巧与解析（二）

面试官：简述 C++语言程序的组成。

应聘者：C++程序结构由编译预处理、注释和程序等组成。也有人称程序为函数，因为程序是由一个主函数和若干个函数组成的。

面试官：单行注释和多行注释之间有何不同？

应聘者：单行注释到行尾就结束；而多行注释到“ */ ”才结束。即使是函数的结尾也不能作为多行注释的结尾，必须要加上注释结尾标记“ */ ”，否则将出现编译错误。