第 2 章　C++程序结构

学习目标 Objective

本章将带领读者了解 C++程序的开发过程，剖析 C++程序结构，掌握 C++代码编写规范，熟练使用 C++的输入/输出对象。

内容导航 Navigation

- 程序分析
- 输入/输出对象
- 标识符
- 预处理
- 命名空间

2.1　简单程序

从这章开始神奇的 C++学习之旅。与学习其他所有程序语言一样，首先从一个简单的程序开始。

【实例 2-1】gushi（代码 2-1.txt）

新建名为“gushi”的【C++ Source File】源程序，源代码如下所示：

```
#include<iostream>
using namespace std;
void  main()
{
    cout<<"红豆生南国，春来发几枝。"<<endl;
    cout<<"愿君多采撷，此物最相思。"<<endl;
    system("pause");
}
```

【代码详解】

在程序中，定义了 main 函数，输出字符串“红豆生南国，春来发几枝。”和“愿君多采撷，此物最相思。”。

运行结果如图 2-1 所示。

图 2-1　代码运行结果

【实例分析】

本实例中使用 cout 函数实现输出字符串的效果。在本例中定义了主函数 main，每一个 C++程序都必须包含一个且只有一个 main 函数，main 函数是每个程序的起点。

2.2　C++程序分析

在上例中，可能有很多关键字是初学者不太理解的，本节将详细分析该例中用到的关键字。

2.2.1　#include 指令及头文件

在上例中，使用了 include 这个关键字，但是这个关键字起了什么作用呢？下面就来详细介绍 include 这个关键字。

```
#include<iostream>
```

include 是 C++的预处理指令，表示包含 C/C++标准输入头文件。C++编译系统会根据头文件名把该文件的内容包含进来。包含指令不仅仅限于.h 头文件，可以包含任何编译器能识别的 C/C++代码文件，包括.c、.hpp、.cpp、.hxx、.cxx 等，甚至.txt、.abc 等都可以。

提　示
C++虽然主要是在 C 的基础上发展起来的一门新语言，但它不是 C 的替代品，也不是 C 的升级，不要用""代替<>来包含系统头文件，虽然有些编译器允许你这样做，但它不符合 C/C++标准。错误的示例：#include "stdio.h"、#include "iostream"。

那么，在 C++中头文件是怎么定义的呢？

在语句“#include<iostream>”中，iostream.h 就是头文件。C++程序的头文件是以“.h”为后缀的，用于保存程序的声明。

一个头文件由 3 部分内容组成：

- 头文件开头处的版权和版本声明。
- 预处理块。
- 函数和类结构声明等。

在 C++中，头文件的作用主要包含以下两点。

（1）可以通过头文件来调用已有的程序功能。为了保护源代码的安全性，通过头文件的形式来调用该代码的功能。用户只需要按照头文件中的接口声明来调用该头文件中的功能，而不必关心具体功能是怎么实现的。编译器会从库中读取相应的代码。

（2）头文件可以加强安全性检查。在调用接口功能时，如果调用方式和头文件中的声明不一致，编译器就会报错，从而减少程序员的调试负担。

提　示

不要使用#include <iostream.h>，不要使用#include <string.h>，因为它们已经被 C++标准明确地废弃了，请改为 #include <iostream> 和 #include <cstring>，规则如下。

（1）如果这个头文件是旧 C++特有的，那么去掉.h 后缀，并放入 std 名字空间，如 iostream.h 变为 iostream。

（2）如果这个头文件是 C 也有的，那么去掉.h 后缀，增加一个 c 前缀，如 string.h 变为 cstring、stdio.h 变为 cstdio 等。

2.2.2 main 函数

在上例中，使用了 main()函数，那么这个 main()函数代表什么呢？C++程序必须有且只能有一个 main()函数，main()函数是程序的入口点，无论 main 函数在程序中处于什么样的位置。但是，并非所有 C++程序都有传统的 main()函数。用 C 或 C++写成的 Windows 程序入口点函数称为 WinMain()，而不是传统的 main()函数。

main()函数和其他函数一样也是函数，有相同的构成部分。在 32 位控制台应用程序中，C++ Builder 生成具有下列原型的默认 main()函数：int main(int argc,char** argv);。这个 main()函数形式取两个参数并返回一个整型值。

提　示

不要将 main 函数的返回类型定义为 void，虽然有些编译器允许你这样做，但它不符合 C/C++标准。不要将函数的 int 返回类型省略不写，在 C++中要求编译器至少给一个警告。错误的示例：void main() {}、main() {}。

第一个参数 argc，argc 代表参数的数量，指明有多少个参数将被传递给主函数 main()。真正的参数以字符串数组（即第 2 个参数 argv[]）的形式来传递。

main()函数本身以索引 0 为第一参数，所以 argc 至少为 1。它的总数是 argv 列阵的元素数目。这意味着 argv[0]的值是至关重要的，如果用户在控制台环境中程序名称后输入含参数的指令，那么随后的参数将传递给 argv[1]。

下面用一个实例来说明 main 如何调用参数。

【实例 2-2】main 函数调用参数（代码 2-2.txt）

新建名为“Maintest”的【C++Source File】源程序，源代码如下所示：

```
#include<iostream>
using namespace std;
int main(int argc,char* argv[])
{
    double Operand1,Operand2,Addition;
    Operand1=atoi(argv[1]);
    Operand2=atoi(argv[2]);
```

```
    Addition=Operand1+Operand2;
    cout<<"/nFirst Number:"<<Operand1<<endl;
    cout<<"/nSecond Number:"<<Operand2<<endl;
    cout<<Operand1<<"+"<<Operand2<<"="<<Addition<<endl;
    return 0;
}
```

【代码详解】

首先，在主程序中定义了三个 double 类型的变量。第一个变量 Operand1 对传入参数数组的第一个数取整，第二个变量 Operand2 对传入参数数组的第二个数取整，定义第三个变量 Addition 为前两个变量的和。

将第一个变量 Operand1 和第二个变量 Operand2 输出，然后将第三个变量 Addition 也输出。

在 Visual Studio 2019 主界面，选择【生成】|【生成 Maintest】菜单选项，即可生成可执行文件 Maintest.exe。在【DOS】窗口中执行 Maintest.exe 文件，输入两个参数 12.5 和 36.8，运行结果如图 2-2 所示。

图 2-2　代码运行结果

【实例分析】

在本例中，首先编译了该程序，生成 Maintest.exe 可执行文件。在调用可执行文件时，输入两个参数，分别是 12.5 和 36.8，最后输出的结果是按照程序设计输出的。该例通过调用 main 函数实现上述功能。

2.2.3　变量声明和定义

在 C++中，不仅变量有名字，枚举（enumeration）、函数（function）、类（class）、模板（template）等事务都有名字。在使用任何一个名字之前，必须先对该名字表示的事务进行声明（declaration）或者定义（definition）。在程序使用中，离不开变量。变量的定义可以为变量分配存储空间，还可以为变量指定初始值。在程序中，变量有且仅有一个定义。

声明是为了说明变量的类型和名字。定义也是声明，当定义变量的时候声明了它的类型和名字。可以通过使用 extern 关键字声明变量名而不定义它。不定义变量的声明包括对象名、对象类型和对象类型前的关键字 extern。extern 声明不是定义，也不分配存储空间。它只是说明变量定义在程序的其他地方，程序中变量可以声明多次，但只能定义一次。

例如：

```
int i;          //定义也可以说是声明
Extern int i;   //这就是单纯的声明
```

在上例中，就是一个单纯的声明，而不是定义。这条语句只是告诉程序有一个 int 型变量 i，

而没有为 i 分配空间，也没有给 i 赋值。

任何在多文件中使用的变量都需要有与定义分离的声明。在这种情况下，一个文件含有变量的定义，使用该变量的其他文件则包含该变量的声明（而不是定义）。

可以用下面的语法来定义（也是声明）一个变量。

```
变量类型说明符 变量名 1,变量名 2,...,变量名 3;
```

其中，变量类型说明符的作用是告诉编译器该变量的类型。表 2-1 列出了 C++中的一些基本数据类型。

表2-1 C++中的一些基本数据类型

基本数据类型	变量类型说明符
char	char
unsigned char	unsigned char
signed char	signed char
char16_t	char16_t
char32_t	char32_t
wchar_t	wchar_t
unsigned short int	unsigned short, unsigned short int
short int	signed short, signed short int, short, short int
unsigned int	unsigned int, unsigned
int	signedint, signed, int
unsigned long int	unsigned long int, unsigned long
long int	signed longint, signed long, long int, long
unsigned long long int	unsigned long long int, unsigned long long
long long int	signed long long int, signed long long, long long int, long long
bool	bool
float	float
double	double
long double	long double

在定义变量时，需要遵循以下规则。

（1）变量名的首字母必须为 26 个英文字母的大小写加下画线，其他字母必须为 26 个英文字母的大小写加下画线和数字。

（2）变量名不可以是 C++中预留的关键词。前面已经介绍过一些关键词，例如 signed、unsigned、int 和 double 等。

（3）C++标准规定，所有以两个下画线开头的名字，以及一个下画线加上一个大写字母开头的名字，例如__range、__Range 或者_Range，在程序中都不可以用，因为要为标准库预留；所有以一个下画线开头并且第二个字符并不是下画线，也不是大写字母的名字，例如_range，在程序中不可以用在全局名字空间（对变量来说，在全局名字空间的变量也是全局变量）。全局变量和全局名字空间稍后再提。如果你用了这些名字，编译器可能不会报错，但是你的程序的可移植性就变差了，因为换到另一个编译器，就可能和另一个编辑器的库实现存在名字冲突。

提 示
使用 C++变量作用域时一定要注意，一般是以一对花括号范围作为一个作用域的。

（4）在 C++中，名字的大小写是不同的，即大写字母的名字和小写字母的名字是不同的名字。例如，age、Age、AGE 是 3 个不同的名字。

2.2.4 函数的声明

在上面的实例中，定义了一个函数 main。其实在 C++中，函数声明不仅仅是 main 函数。在 C++程序中调用任何函数之前，首先要对函数进行定义。如果调用此函数在前，定义函数在后，就会产生编译错误。

为了使函数的调用不受函数定义位置的影响，可以在调用函数前进行函数的声明。这样，无论函数是在哪里定义的，只要在调用前进行函数的声明，就可以保证函数调用的合法性。

在 C++中，函数的定义格式为：

```
返回值类型 函数名(参数列表)
{
   函数体;
}
```

提 示
参数的书写要完整，不要图省事只写参数的类型而省略参数名字。如果函数没有参数，就用 void 填充。另外，参数命名要恰当，顺序要合理。

通常，函数名可以是任何合法的标识符。函数的参数列表是可选的，如果函数不需要参数，就可以省略参数列表，但是参数列表两边的括号不能省略。

函数体描述的是函数的功能，主要由一条或多条语句构成。函数也可以没有函数体，此时的函数称为空函数。空函数不执行任何动作。在开发程序时，当前可能不需要某个功能，但是将来可能需要，此时可以定义一个空函数，在需要时为空函数添加实现代码。

函数都有一个返回值，当函数结束时，将返回值返回给调用该函数的语句。但是，函数也可以没有返回值，即返回值类型为 void。如果函数有返回值，通常在函数体的末尾使用 return 语句返回一个值，其类型必须与函数定义时的返回值类型相同或兼容。

下面用一个简单的实例来说明函数如何声明和定义。

【实例 2-3】函数应用（代码 2-3.txt）

新建名为“maxtest”的【C++ Source File】源程序，源代码如下所示：

```
//从键盘输入两个数，调用 max()函数的方法，求这两个数中的较小值
#include<iostream>
using namespace std;
int main()
{
    //min()函数原型声明语句
    float min(float,float);
```

```
    float a,b,min;              //声明变量
    //输入参数并计算
    cout<<"从键盘输入：a=";
    cin>>a;
    cout<<"从键盘输入：b=";
    cin>>b;
    max=min(a,b);               //调用 min()函数
    cout<<"较大值是："<<min<<endl;
     system("pause");
    return 0;
}
//定义 min()函数
float min(float x,float y)  //min()返回值类型为浮点型
{
    float z;
    z=(x<y)?x:y;                //比较 x 和 y，如果 x<y，就用 x 初始化 z，否则用 y 初始化 z
    return(z);
}
```

【代码详解】

在这个例子中，首先定义了 main 函数，在 main 函数中声明了 min 函数，之后声明了 3 个变量 m、n 和 Min。然后，调用 cin，输入两个 float 类型的数值，分别复制给 a 和 b。最后调用 min 函数找到两个数中较小的数字，并将该值输出。

运行结果如图 2-3 所示。

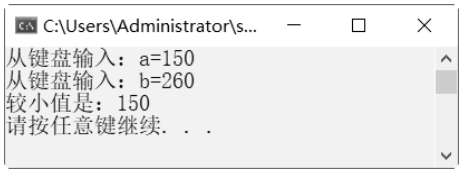

图 2-3　代码运行结果

【实例分析】

首先输入一个数为 150，接着输入 260。根据程序设计，取 150 和 260 中较小的数，将较小的数输出。

2.2.5　关于注释

在 C++中，注释是用来帮助程序员读程序的语言结构，是一种程序礼仪，可以用来概括程序的算法、表达变量的意义或者阐明一段比较难懂的程序代码。注释不会增加程序的可执行代码的长度。在代码生成以前，编译器会将注释从程序中剔除掉。

提　示
说明性文件（如头文件（.h 文件）、.inc 文件、.def 文件、编译说明文件（.cfg 文件）等）头部应进行注释，注释必须列出，如版权说明、版本号、生成日期、作者、内容、功能、与其他文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

C++中有两种注释符号，一种是注释对（/*,*/），与 C 语言中的一样。注释的开始用/*标记，

编译器会把/*与*/之间的代码当作注释。注释可以放在程序的任意位置，可以含有制表符（tab）、空格符或换行符，还可以跨越多行程序。

例如：

```
/*
*这是第一次看到 C++的类定义
*类可用于基于对象和
*面向对象编程中
*screen 类的实现代码在第 13 章中
*/
```

另一种注释符是双斜线（//），它可以用来注释一个单行，程序行中注释符右边的内容都将被当作注释而被编译器忽略。

例如：

```
//这部分称为类体
public:
voidhome();      //将光标移至 0,0
voidrefresh();  //重绘 screen
```

在 C++中，注释的种类分为以下几种。

- 重复性注释：只是用不同文字把代码的工作又描述一次。这种代码对程序本身并没有提供更多信息。
- 解释性注释：通常用于解释复杂、敏感的代码块。对于复杂的代码，需要写注释来说明该代码的功能，以增加程序的可读性。
- 标记性注释：用于告诉开发者某处的工作未做完。在实际工作中，经常会以这些注释作为程序骨架的占位符，或者已知 bug 的标记。
- 概述性注释：将若干行代码以一两句话说出来，程序员能够快速读取注释，了解程序的功能，增加可读性。
- 意图性注释：用来指出要解决的问题，而非解决的方法。意图性注释和概述性注释没有明显的界限，其差异也无足轻重，都是非常有效的注释。

有些信息不能通过代码来体现，比如版权声明、作者、日期、版本号等信息以及与代码设计有关的一些注意事项，但是这些信息都必须体现在源代码中，所以就用注释来记录这些信息。

写好一份注释，是写出完美程序的前提，写好注释并不比写好一段程序更容易。所以在写注释的过程中，必须遵循以下几个原则。

（1）站在读者的立场编写注释

编写的代码将会面对很多不同的读者。其中还包括代码的复审者。他们希望看到的是准确的注释。注释的内容应包含：源程序的特性（文件名、作用、创建时间等）和函数注释。

（2）及时编写注释

注释应该是在编程的过程中同时进行的，不能在程序开发完成之后再补写注释。这样会多花很多时间，并且在长时间之后，会慢慢读不懂自己的程序了。

（3）好注释能在更高抽象层次上解释想干什么

在编写注释这一问题上，经常犯的一个错误是将代码已经清楚说明的东西换种说法再写一次。如果为代码添加中文注释的时候，简单地等同于将英文译作中文，那么这样的注释能够给他人带来的好处微乎其微，更多时候是徒增阅读负担以及维护工作量。

2.3 输入输出对象

C++的输出和输入是用“流”（stream）的方式来实现的，所谓的“流”是从数据的传输抽象而来的，可以将其理解为文件。图 2-4 表示 C++通过流进行输入输出的过程。

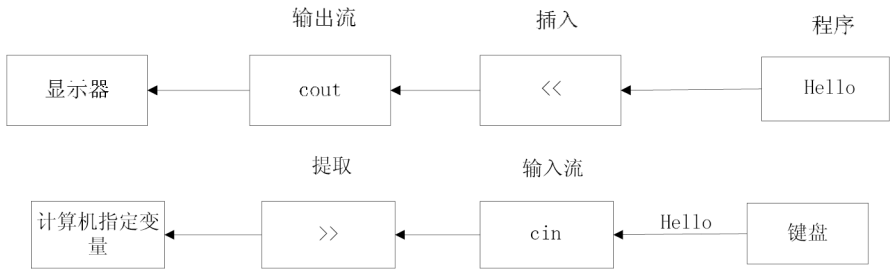

图 2-4　C++通过流进行输入输出的过程

有关流对象 cin、cout 和流运算符的定义等信息是预先定义好的流对象，存放在 C++的输入输出流库中，因此如果在程序中使用 cin、cout 和流运算符，就必须使用预处理命令把头文件 stream 包含到本文件中：

```
#include<iostream>
```

2.3.1 cout 输出数据

cout 语句的一般格式为：

```
cout<<表达式 1<<表达式 2<<……<<表达式 n;
```

在定义流对象时，系统会在内存中开辟一段缓冲区，用来暂存输入输出流的数据。在执行 cout 语句时，先把插入的数据顺序存放在输出缓冲区中，直到输出缓冲区满或遇到 cout 语句中的 endl（或'\n'、ends、flush）为止，此时将缓冲区中已有的数据一起输出，并清空缓冲区。输出流中的数据在系统默认的设备（一般为显示器）输出。

cout 可以输出整数、实数、字符以及字符串，cout 中插入符“<<”后面可以跟变量、常量、转义字符、对象等表达式。

一个 cout 语句可以分成若干行。

```
cout<<"This is a simple C++ program."<<endl;
```

可以写成：

```
cout<<"This is"     //注意行末尾无分号
<<"a C++"
```

```
    <<"program."
    <<endl;              //语句最后有分号
```

也可写成多个 cout 语句：

```
cout<<"This is";    //语句末尾有分号
cout<<"a C++";
cout<<"program.";
cout<<endl;
```

以上 3 种情况的输出均为：

```
This is a simple C++ program
```

下面通过一个具体例子来展示 cout 输出的用法。

【实例 2-4】cout 的用法（代码 2-4.txt）

新建名为“couttest”的【C++ Source File】源程序，源代码如下所示：

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=6;i>=1;i--)
    {
        cout<<"count="<<i<<endl;
    }
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，在主程序中使用了一个 for 循环，将从 6 到 1 的 int 型变量全部输出一遍。

运行结果如图 2-5 所示。

图 2-5　代码运行结果

【实例分析】

从整个示例来看，分别调用 cout 将 6~1 输出到屏幕上。

前面介绍了 cout 的默认格式，但是在实际应用中，输入输出有一些特殊的要求，如在输出实数时规定字段宽度、只保留两位小数、数据向左或向右对齐等。

如果使用了控制符，在程序单位的开头除了要加 iostream 头文件外，还要加 iomanip 头文件。

下面通过一个具体的例子来说明如何使用控制符。

【实例 2-5】cout 控制符（代码 2-5.txt）

新建名为“couttest”的【C++ Source File】源程序，源代码如下所示：

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    double a=123.456789012345; //对 a 赋初值
    cout<<a<<endl;              //输出:123.457
    cout<<setprecision(9)<<a<<endl;    //输出:123.456789
    cout<<setprecision(6)<< a <<endl;  //恢复默认格式
    cout<<setiosflags(ios::fixed)<< a <<endl;      //输出:123.456789
    cout<<setiosflags(ios::fixed)<<setprecision(8)<<a<<endl;
    //输出:123.45678901
    cout<<setiosflags(ios::scientific)<<a<<endl;   //输出:12345679
    cout<<setiosflags(ios::scientific)<<setprecision(4)<<a<<endl;
    //输出:123.5
    int b=123456;           //对 b 赋初值
    cout<<b<<endl;          //输出:123456
    cout<<hex<<b<<endl;     //输出:1e240
    cout<<setiosflags(ios::uppercase)<<b<<endl;    //输出:1E240
    cout<<setw(10)<<b<<','<<b<<endl;               //输出:1E240,1E240
    cout<<setfill('*')<<setw(10)<<b<<endl;         //输出:*****1E240
    cout<<setiosflags(ios::showpos)<<b<<endl;      //输出:1E240
    system("pause");
    return 0;
}
```

【代码详解】

在本例中，首先定义了一个 double 型变量 a，再调用 cout 各种标识符，按照需要将 double 型变量 a 输出。接下来，定义了 int 型变量 b，再调用 cout 各种类型标识符将 int 型变量 b 输出。

运行结果如图 2-6 所示。

图 2-6　代码运行结果

【实例分析】

从运行结果来看，利用 cout 标识符的控制符实现了各类数据的输出。

2.3.2　cin 读取输入数据

cin 可以从键盘获得多个输入值，语句的一般格式为：

```
cin>>变量 1>>变量 2>>……>>变量 n;
```

与 cout 类似，一个 cin 语句可以分成若干行。

```
cin>>a>>b>>c>>d;
```

可以写成：

```
cin>>a      //注意行末尾无分号
>>b         //这样写可能看起来清晰些
>>c
>>d;
```

也可以写成：

```
cin>>a;
cin>>b;
cin>>c;
cin>>d;
```

以上 3 种情况均可以从键盘输入：1234↙。

也可以分多行输入数据：

```
1↙
23↙
4↙
```

在用 cin 输入时，系统会根据变量的类型从输入流中折取相应长度的字节。

下面通过一个例子来说明如何使用 cin 来输入。

【实例 2-6】cin 的用法（代码 2-6.txt）

新建名为“cintest”的【C++ Source File】源程序，源代码如下所示：

```
#include<iostream>
using namespace std;
void main()
{
    cout << "请输入您的名字和年龄:" << endl;
    char name[10];
    int age;
    cin >> name;
    cin >> age;
    cout << "您的名字是: " << name << endl;
    cout << "您的年龄是: " << age << endl;
    system("pause");
```

```
}
```

【代码详解】

在该例中，首先定义了一个 char 类型的字符串 name，又定义了一个 int 类型的变量 age。再使用 cin 从键盘输入 name 和 age，最后将赋值的变量输出。

运行结果如图 2-7 所示。

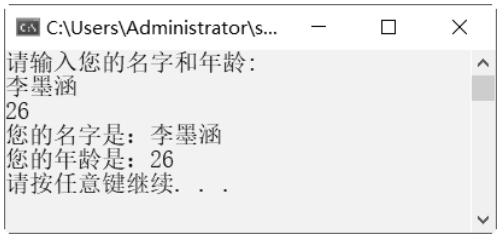

图 2-7　代码运行结果

【实例分析】

从运行结果来看，利用 cin 实现了 name 和 age 的输入。

提　示
在默认方式下，标准输入设备是键盘，标准输出设备是显示器，而不论何种情况，标准输出设备总是显示器。

2.4　标识符

在 C++中，标识符是用来定义资源的，当用户创建一个新自由对象的时候，系统会为其提供一个默认标识符或者用户自己定义一个标识符。

标识符用于字符序列，表示下列操作之一：

- 对象或变量名称。
- 类、结构或联合名。
- 枚举类型名称。
- 类、结构、联合或枚举的成员。
- 函数或类成员函数。
- typedef 名称。
- 标签名称。
- 宏名。
- 宏参数。

以下字符用作标识符的第一个字符或者所有后续字符是合法的。

```
_abcdefghijklm
nopqrstuvwxyz
ABCDEFGHIJKLM
```

```
NOPQRSTUVWXYZ
```

以下字符可以作为标识符中除第一个字符外的所有字符。

```
0123456789
```

提　示
标识符只能在说明它或定义它的范围内是可见的，而在该范围之外是不可见的。

2.4.1　保留字

保留字也叫关键字，它是 C++系统预定义的，由小写英文字母组成的单词、词头或词组。每个保留字都被系统赋予了一定的含义，具有相应的功能，所以用户不能把它们作为非保留字使用。

在 C++中，保留字分为表 2-2 所示的几类。

表2-2　保留字类型

保留字类型	保留字
类型说明保留字	int,long,short,float,double,char,unsigned,signed, const,void,volatile,enum,struct,union
语句定义保留字	if,else,goto,switch,case,do,while,for,continue,break,return,default,typedef
存储类说明保留字	auto,register,extern,static
长度运算符保留字	sizeof

2.4.2　标识符命名

在 C++中，各种数据对象都需要用标识符来区分，即它的名字。

标识符的命名规则如下。

（1）以非数字字符开头，如字母或下画线“_”。

（2）只能由字母、数字和下画线 3 类字符组成。

（3）区分大小写。

（4）有穷字符序列，只有前 32 个字符有效，超过 32 个字符，以后的字符忽略不计。

（5）不能与 C++关键字相同。

2.5　预处理

C++的预处理（preprocess）是指在 C++程序源代码被编译之前，由预处理器（preprocessor）对 C++程序源代码进行的处理。虽然预处理命令不是 C++语言的一部分，但是它有扩展 C++程序设计环境的作用。

提　示
预处理命令是 C++统一规定的，但是它不是 C++语言本身的组成部分，不能直接对它们进行编译（因为编译程序不能识别它们）。

在预处理中，包含以下常用的预处理。

- #include：包含头文件。
- #if：条件。
- #else：否则。
- #elif：否则如果。
- #endif：结束条件。
- #ifdef 或#ifdefined：如果定义了一个符号，就执行操作。
- #ifndef 或#if!defined：如果没有定义一个符号，就不执行操作。
- #define：定义一个符号。
- #undef：删除一个符号。
- #line：重新定义当前行号和文件名。
- #error：输出编译错误，停止编译。
- #pragma：提供专用的特性，同时保证与 C++的完全兼容。

下面对几个常用的预处理进行详细的说明。

1. #include 在程序中包含头文件

#include 的作用是将另一个源文件的内容合并到当前程序中，被合并的源文件称为“头文件”。头文件通常以.h 结尾，其内容可使用#include 预处理器指令包含到程序中。使用 include 命令可以减少程序员的重复劳动。

头文件中一般包含函数原型与全局变量。

包含头文件的形式常有下面两种。

```
#include<iostream>
#include"myheader.h"
```

前者<>用来引用标准库头文件，后者""常用来引用自定义的头文件。

对于前者，编译器只搜索包含标准库头文件的默认目录；对于后者，编译器首先搜索正在编译的源文件所在的目录，找不到时再搜索包含标准库头文件的默认目录。

如果把头文件放在其他目录下，为了查找到它，就必须在双引号中指定从源文件到头文件的完整路径。

2. #define 定义符号、宏

（1）符号

```
#define PI 3.14159265
```

定义符号 PI 为 3.14159265。

```
#undef PI
```

取消 PI 的值。

这里 PI 看起来像一个变量，但它与变量没有任何关系，它只是一个符号或标志，在代码编译

前，此符号会用一组指定的字符来代替 3.14159265，不是一个数值，只是一个字符串，不会进行类型检查。

在编译前，预处理器会遍历代码，在它认为置换有意义的地方，用字符串 PI 的定义值（3.14159265）来代替在注释或字符串中的 PI，不进行替换。

在 C 中常以#define 来定义符号常量，如上面的“#define PI 3.14159265”，但在 C++中最好使用 const 来定义常量。

```
const long double PI=3.14159265;
```

两者相比较，前者没有指定类型，容易引起不必要的麻烦，而后者定义得很清楚，所以在 C++中推荐使用 const 来定义常量。

#define 有以下缺点：

① 不支持类型检查。

② 不考虑作用域。

③ 符号名不能限制在一个命名空间中。

（2）#undef 删除#define 定义的符号

```
#define PI 3.14159265
//之间所有的 PI 都可以被替换为 3.14159265
#undef PI
```

之后不再有 PI 这个标识符。

（3）定义宏

```
#define Print(Var) count<<(Var)<<endl
```

将宏名中的参数代入语句中的参数。var 是带入参数表。带参数的宏相当于一个函数的功能，但是比函数更加便捷。宏后面没有分号。

Print(Var)中的 Print 和“(”之间不能有空格，否则“(”就会被解释为置换字符串的一部分。

```
#define Print(Var,digits) count<<setw(digits)<<(Var)<<endl
```

调用：

```
Print(ival,15)
```

预处理器就会把它换成：

```
cout<<setw(15)<<(ival)<<endl;
```

所有的情况下都可以使用内联函数来代替宏，这样可以增强类型的检查：

```
template<classT>inline void Print(const T &var,const int &digits)
{
count<<setw(digits)<<var<<endl;
}
```

调用：

```
Print(ival,15);
```

使用宏时应注意易引起的错误：

```
#define max(x,y) x>y?x:y;+
```

result=max(myval,99);替换成 result=myval>99?myval:99;，这个没有问题，是正确的。调用 result=max(myval++,99);替换成 result=myval++>99?myval++:99;，这样如果 myval>99，那么 myval 就会递增两次，这种情况下()是没什么用的，如 result=max((x),y)替换成 result=(myval++)>99?(myval++):99;。

再如：

```
#define  product(m,n) m*n
```

result=product(5+1,6);替换为 result=5+1*6;，所以产生了错误的结果，此时应使用()把参数括起来。

```
#define product(m,n) (m)*(n)
```

这样 result=product(5+1,6);，结果正确。

2.6 命名空间

在书写程序的时候，很多时候都要用到 namespace，那么这个 namespace 是什么呢？

2.6.1 命名空间的定义

命名空间（namespace）是一种描述逻辑分组的机制，可以将按某些标准在逻辑上属于同一个集团的声明放在同一个命名空间中。

在 C++中，名称（name）可以是符号常量、变量、宏、函数、结构、枚举、类和对象等。在大规模程序的设计中，以及在程序员使用各种各样的 C++库时，为了避免这些标识符的命名发生冲突，标准 C++引入了关键字 namespace（命名空间/名字空间/名称空间/名域），可以更好地控制标识符的作用域。

原来 C++标识符的作用域分成三级：如复合语句和函数体、类和全局。现在，在其中的类和全局之间，标准 C++又添加了命名空间这个作用域级别。

命名空间可以是全局的，也可以位于另一个命名空间中，但是不能位于类和代码块中。所以，在命名空间中声明的标识符默认具有外部链接特性（除非它引用了常量）。

在所有命名空间之外，还存在一个全局命名空间，它对应文件级的声明域。因此，在命名空间机制中，原来的全局变量，现在被认为位于全局命名空间中。

有两种形式的命名空间——有名的和无名的。

```
named-namespace-definition:
namespaceidentifier{namespace-body}
unnamed-namespace-definition:
namespace{namespace-body}
```

```
namespace-body:
declaration-seqopt
```

下面通过一个例子来说明如何定义命名空间。

【实例 2-7】定义命名空间（代码 2-7.txt）

新建名为“mmkjtest”的【C++Source File】源程序，源代码如下所示：

```
#include<iostream>
#include<string>
using namespace std;
//using namespace 编译指示，使在 C++标准类库中定义的名字在本程序中可以使用
//否则，iostream、string 等 C++标准类就不可见了，编译就会出错
//两个在不同命名空间中定义的名字相同的变量
namespace myown1{
    string user name="myown1";
}
namespacem yown2{
    string user_name="myown2";
}
int main()
{
    cout<<""
        <<"Hello,  "
        <<myown1::user_name     //用命名空间限制符 myown1 访问变量 user_name
        <<"  andgoodbye!"<<endl;
    cout<<""
        <<"Hello,  "
        <<myown2::user_name     //用命名空间限制符 myown2 访问变量 user_name
        <<"  andgoodbye!"<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在本例中，定义了两个命名空间，分别是 myown1 和 myown2，每个命名空间都定义了一个变量。

在主程序中，将各个命名空间的内容输出。

运行结果如图 2-8 所示。

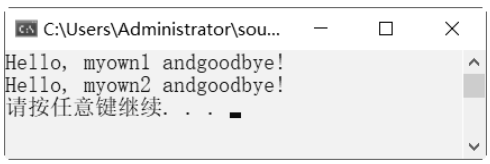

图 2-8　代码运行结果

【实例分析】

从运行结果可以看出，每个命名空间的内容都被很好地调用了。

2.6.2 using 关键字

在 C++的命名空间中，为了方便使用，又引入了关键字 using。利用 using 声明可以在引用命名空间成员时不必使用命名空间限定符“::”。

使用方法如下：

```
Using namespace std;
```

下面通过一个例子来说明如何使用 using 关键字。

【实例 2-8】using 关键字（代码 2-8.txt）

新建名为“usingtest”的【C++ Source File】源程序，源代码如下所示：

```
#include<iostream>
#include<string>
using namespace std;
//using namespace 编译指示，使在 C++标准类库中定义的名字在本程序中可以使用
//否则，iostream、string 等 C++标准类就不可见了，编译就会出错
//两个在不同命名空间中定义的名字相同的变量
namespace myown1{
    string user_name="myown1";
}
namespace myown2{
    string user_name="myown2";
}
int main()
{
    using namespace myown1;
    cout<<""
        <<"Hello, "
        <<user_name
        <<" and goodbye!"<<endl;
    //using namespace myown2;
    cout<<""
        <<"Hello, "
        <<myown2::user_name     //用命名空间限制符 myown2 访问变量 user_name
        <<" and goodbye!"<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在本例中，定义了两个命名空间，分别是 myown1 和 myown2，每个命名空间都定义了一个变量。

在主程序中，使用 using 关键字调用了 myown1 的命名空间，在调用第二个命名空间时，没有使用 using 调用。

运行结果如图 2-9 所示。

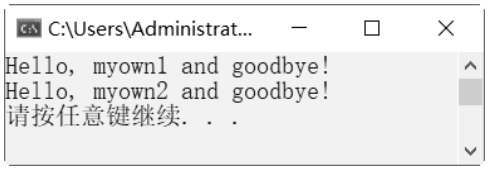

图 2-9　代码运行结果

【实例分析】

从运行结果可以看出，每个命名空间的内容都被很好地调用了。

2.6.3　命名空间 std

C++标准中引入命名空间的概念是为了解决不同模块或者函数库中相同标识符冲突的问题。有了命名空间的概念，标识符就被限制在特定的范围内，不会引起命名冲突。典型的例子是 std 命名空间，C++标准库中所有标识符都包含在该命名空间中。

如果确信在程序中引用某个或者某些程序库不会引起命名冲突，那么可以通过 using 操作符来简化对程序库中标识符（通常是函数）的使用，例如：

```
using namespace std;
```

那么就可以不用在标识符加前缀 std::来使用 C++标准库中的函数了。

C++标准引入的 std 命名空间并不向后兼容目的 C++标准库。旧的 C++标准库的头文件中声明的标识符是全局范围的，不需要使用 std 命名空间限定就可以使用。为了区分在标准化进程中的这种变化，C++新的标准库启用了新的头文件命名格式。这样就允许程序员通过包含不同格式的头文件来使用不同的 C++标准库。

新的 C++标准库的头文件不再包含扩展名（.h、hpp、hxx 等），形式如下：

```
#include<iostream>
#include<string>
```

这种新标准同样涵盖 C 标准库，C 标准库的头文件现在可以这样引用：

```
#include<cstdlib>//was:<stdlib.h>
#include<cstring>//was:<string.h>
```

而这种新格式的头文件中定义的标识符全部涵盖在 std 命名空间下。

这种新的命名方式的便利之处就在于可以方便地区分旧的 C 标准库中的头文件和新的 C++标准库中的头文件。比如 C 标准库和 C++标准库中原先都有一个<string.h>的头文件，如果同时使用，就会很不方便。现在不存在这样的问题了，形式如下：

```
#include<string>//C++ string class
#include<cstring>//C char * functions
```

2.7 小试身手——入门经典程序

1. 求一元二次方程 $ax^2+bx+c=0$ 的根

```
#include <iostream>
#include <string>
#include<cmath>
#include<iomanip>
using namespace std;
int main()
{
    float a,b,c;
    float x1,x2;
    cout<<"请输入 a,b,c 的值：";
    cin>>a>>b>>c;
    float t=b*b-4*a*c;
    if(t<0)
        cout<<"此方程无实根."<<endl;
    else
    {
        x1=(-b+sqrt(t))/(2*a);
        x2=(-b-sqrt(t))/(2*a);
        cout<<setiosflags(ios::fixed)<<setiosflags(ios::right);
        cout<<setprecision(4);
        cout<<"x1="<<x1<<endl;
        cout<<"x2="<<x2<<endl;
    }
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，首先定义了 float 变量 a、b、c 和 x1、x2，输入 a、b、c 三个数作为一元二次方程的系数。定义 float 型变量 t 为 b*b-4*a*c，判断 t 的值，若 t<0，则该方程无解；若 t>0，则解出方程的两个值 x1 和 x2，并且打印出来。

运行结果如图 2-10 所示。

图 2-10　代码运行结果

【实例分析】

从运行结果来看，本例的目的是求解一元二次方程。输入一元二次方程的三个系数 a、b、c 分别是 1、2、3，以这三个系数组成的方程的解是-1 和-2。在本例中，使用 cin 实现了系数的输入，使用 cout 实现了结果的输出。

2. 求两个数中的最小值

输入 int 型变量 x 和 y，比较 x 和 y 的大小，将 x 和 y 中较小的输出。

```
#include <iostream>
using namespace std;
int main()
{
    int x, y, min;
    cout << "Enter x: ";
    cin >> x;
    cout << "Enter y: ";
    cin >> y;
    min = (x < y ? x : y);
    cout << x << (x > y ? " > " : " <= ") << y << endl;
    cout << min << " is the smaller number." << endl;
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，定义了三个 int 型变量 x、y、min，输入 x 和 y，使用比较运算符比较 x 和 y 的大小，把其中较小的值赋给 min，在输出时，仍然使用比较运算符，判断输出大于号还是小于号，最后将 min 输出。

运行结果如图 2-11 所示。

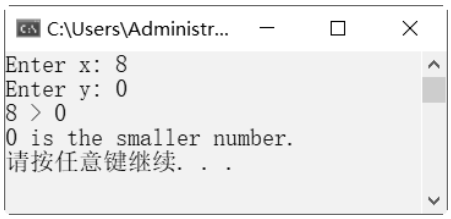

图 2-11　代码运行结果

【实例分析】

从运行结果来看，比较了 x 和 y 的大小，并输出结果。在该程序中，灵活地使用了比较运算符，首先比较两个数的大小，返回其中较小的；然后，使用比较运算符比较两个数的大小，返回的是比较结果。

2.8　疑难解惑

疑难 1　下列标识符哪些是合法的?

```
Program, -page, _lock, test2, 3in1, @mail, A_B_C_D
```

Program、_lock、test2、A_B_C_D 是合法的标识符，其他的不是。

疑难 2　下面一段程序的含义是什么?

```
①#include <iostream.h>
```

```
②void main(void)
{
    ③cout<<"Hello!\n";
    ④cout<<"Welcome to c++!\n"
}
```

①指示编译器将文件 iostream.h 中的代码嵌入该程序中该指令所在的地方。

②主函数名，void 表示函数没有返回值。

③输出字符串“Hello!”到标准输出设备（显示器）上。

④输出字符串“Welcome to c++!”。

在屏幕输出如下：

```
Hello!
Welcome to c++!
```

疑难 3　注释有什么作用？C++中有哪几种注释的方法？它们之间有什么区别？

注释在程序中的作用是对程序进行注解和说明，以便于阅读。编译系统在对源程序进行编译时不理会注释部分，因此注释对于程序的功能实现不起任何作用。而且由于编译时忽略注释部分，因此注释内容不会增加最终产生的可执行程序的大小。适当地使用注释能够提高程序的可读性。在C++中，有两种注释的方法：一种是沿用 C 语言的方法，使用“/*”和“*/”括起注释文字；另一种是使用“//”，从“//”开始，直到它所在行的行尾，所有字符都被作为注释处理。

2.9　经典习题

（1）判别某一年是否为闰年，满足闰年的条件是：

① 能被 4 整除而不能被 100 整除。

② 能同时被 100 和 400 整除。

（2）判断输入的一个字符是否为大写：

① 若是，则将其转换成小写。

② 若不是，则原样输出。

第 3 章　基本数据类型

学习目标 Objective

本章将带领读者认识 C++的常量和变量，了解常量和变量的性质，掌握如何声明和定义一个常量和变量。熟练使用整型、字符型、布尔型等基本数据类型，理解 typedef 的含义，以及如何使用 typedef。

内容导航 Navigation

- 常量
- 变量
- Bool

3.1　变量与常量

常量和变量是在 C++程序中使用频繁的元素，代表了数据的可变性。常量是在定义了之后值不能被改变的量，而变量在定义了之后还可以再赋值，即值可以被改变。

3.1.1　变量

变量指的是一个有名字的对象，即内存里一段有名字的连续的存储空间，变量的名字就叫作变量名，变量的值就是这段内存空间里存储的值。

每个变量都有自己的类型。变量的类型就是该变量所表示的内存空间所存储的数据类型。变量的类型可以是任何一种基本数据类型（当然也可以是非基本数据类型），变量占用的内存空间的大小在绝大多数情况下就是该变量类型的大小。

变量的作用是存储程序中需要处理的数据，它可以在程序中的任何位置使用。

1. 变量的定义

语法：

```
数据类型 变量名;
```

例如：

```
int age;
```

其中，int 是数据类型（整型），而 age 是变量名，更多的时候，就说是变量 age。在上例中，最后的分号表示变量定义已经完成，因为 C++语句总是以分号结束。

提　示
在 C++中，变量命名不能取名为 C 和 C++的保留字，不能超过 250 个字符，不能在同一作用范围内有同名变量，不能夹有空格。

如果要声明一个字符类型变量：

```
char letter;
```

声明一个 bool 类型的变量：

```
bool tagp;
```

其他类型，除了 void 不能直接定义一个变量以外，格式都是一样的。

有时同一数据类型有多个变量，此时可以分别定义，也可以一起定义：

```
int a;
int b;
int c;
```

或

```
int a,b,c;
```

一起定义多个同类型变量的方法：在不同变量之间以逗号（,）分隔，最后仍以分号（;）结束。

2. 变量的赋值和输入

变量的赋值是通过赋值操作符（=）将其右边的值赋值给左边的变量。当定义一个变量的时候，编译器会在内存中分配该变量的存储空间，变量的赋值相当于将赋值操作符右边的值写到左边的变量所代表的内存存储空间中。

【实例 3-1】变量赋值（代码 3-1.txt）

新建名为“fztest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
int main()
{
    int num_apples, num_oranges, num_fruits;
    num_apples = 32;      //1
    num_oranges = 27;     //2
    num_fruits = num_apples + num_oranges; //3
    std::cout << "There are totally " << num_fruits << " fruits." << std::endl;
    num_apples = num_apples - 1;    //4
    num_fruits = num_apples + num_oranges; //5
    std::cout << "If you eat one apple, there will be "<< num_fruits << " fruits
left."<< std::endl;
    system("pause");
    return 0;
```

```
}
```

【代码详解】

在程序中，定义了 3 个 int 型变量，分别是 num_apples、num_oranges 和 num_fruits。接下来给 num_apples 赋值为 32，给 num_oranges 赋值为 27，给 num_fruits 赋值为前两个数的和，然后将 num_fruits 值输出。num_apples 减 1，num_fruits 重新赋值为前两个数的和。最后将结果输出。

运行结果如图 3-1 所示。

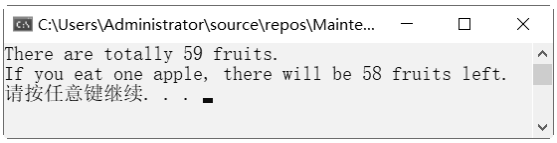

图 3-1　代码运行结果

【实例分析】

在本例中，定义了 int 型变量，通过“=”实现了对 int 型变量的赋值操作。

3. 变量初始化

在给一个变量赋值之前，这段存储空间里保存的是随机值，它甚至可能是别的程序运行完毕后在这段存储空间留下的值。

未初始化的变量是危险的，因为当你不小心使用了一个未初始化或未赋值的变量时，程序的运行结果是未知的。程序中通常需要对一些变量预先设置初始值，这样的过程称为初始化。

在什么时候对变量进行初始化呢？

（1）在定义时初始化变量。

```
int a=0;
```

通过一个等号，让 a 的值等于 0。

同时定义多个变量也一样：

```
int a = 0, b= 1;
```

（2）在定义以后赋值。

```
int a;
a = 100;
```

如果想在程序中知道每个变量名的类型，变量类型所占内存空间的大小和内存空间的首地址可以通过 sizeof 表达式、typeid 表达式和地址操作符来完成。

下面通过一个例子来说明如何使用这两个表达式来显示变量的类型和内存空间大小。

【实例 3-2】siezeof 和 typeid（代码 3-2.txt）

新建名为“sizetest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
int main()
{
```

```
    unsigned long ul_var = 0;
    float f_var = 0.0F;
    std::cout << typeid(ul var).name() << " " << sizeof(ul var)<< std::endl;
    std::cout << typeid(f var).name() << " " << sizeof(f var) << std::endl;
    system("pause");
    return 0;
}
```

【代码详解】

首先，在主程序中定义了一个 unsigned long 类型的变量 ul_val，初始化为 0。下面又定义了一个 float 类型的变量 f_val，该变量初始化为 0.0F。调用 typeid 和 sizeof 将两个变量的类型名和空间大小输出。

运行结果如图 3-2 所示。

图 3-2　代码运行结果图

【实例分析】

在本例中，使用 typeid 实现了取得变量数据类型的作用，使用 sizeof 实现了求得变量大小的效果。

3.1.2　常量

前面介绍了 C++中变量的用法，这里介绍关于常量的用法。常量是指常数或在程序运行过程中值始终不变的数据，其值不能被改变。

常量的定义有以下两种方式。

1. 宏表示常数

宏的语法为：

```
#define 宏名称 宏值
```

下面通过一个实例来说明#define 的用法。

【实例 3-3】宏定义常数（代码 3-3.txt）

新建名为“Htest”的【C++Source File】源程序，源代码如下所示：

```
#include <iostream>
using namespace std;
#define PI 3.14159
int main(int argc, char* argv[])
```

```
{
    double square = 0,volume =0, radius=0;
    cout<<"请输入半径长度"<<endl;
    cin>>radius;
    square = PI * radius * radius;
    cout<<"半径长度为："<<radius<<"的圆面积是："<<square<<endl;
    volume = 4 * PI * radius * radius * radius /3;
    cout<<"半径长度为："<<radius<<"的球体积是："<<volume<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在这个例子中，首先使用宏定义了一个 PI 常量，初始化为 3.14159。在主程序中，使用 cin 输入一个圆的半径。根据输入的半径计算出圆的面积和球的体积，将算得的面积和体积输出。

运行结果如图 3-3 所示。

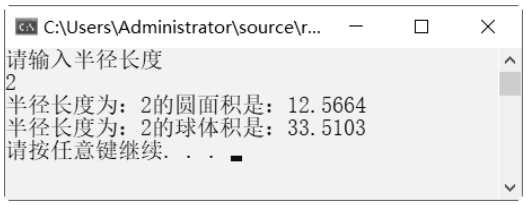

图 3-3　代码运行结果图

【实例分析】

从结果来看，使用#define 实现了 PI 的宏定义，在程序编译时，只要有 PI 的地方全部替换成 3.14159。

2. const 定义

```
const 数据类型 常量名 = 常量值;
```

相比变量定义的格式，常量定义必须以 const 开始。另外，常量必须在定义的同时完成赋值，而不能先定义后赋值。

```
const float PI = 3.1415926;
```

【实例 3-4】宏定义常数（代码 3-4.txt）

新建名为“HCtest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
using namespace std;
const double PI = 3.14159;
int main(int argc, char* argv[])
{
    double square = 0,volume =0, radius=0;
    cout<<"请输入半径长度"<<endl;
    cin>>radius;
    square = PI * radius * radius;
```

```
    cout<<"半径长度为："<<radius<<"的圆面积是："<<square<<endl;
    volume = 4 * PI * radius * radius * radius /3;
    cout<<"半径长度为："<<radius<<"的球体积是："<<volume<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在这个例子中，首先使用 const 定义了一个 PI 常量，初始化为 3.14159。在主程序中，使用 cin 输入一个圆的半径。根据输入的半径计算出圆的面积和球的体积，将计算出的面积和体积输出。

运行结果如图 3-4 所示。

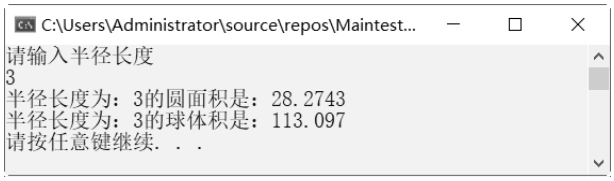

图 3-4　代码运行结果图

【实例分析】

从结果来看，使用 const 实现了对常量 PI 的定义。与宏定义不同，使用 const 定义常量不是在编译时就起作用，而是在运行时才发生作用。

下面介绍几种常见的数据类型常量的表达方式。

1. 整型常量

C++提供 3 种表示整型常量的形式。

- 十进制表示，即十进制整数，例如 132、-345。
- 八进制表示。以 0 开头的整数常量，例如 010、-0536。
- 十六进制表示。以 0x 开头的整数常量，例如 0x7A、-0x3de。

2. 实型常量

C++ 提供两种实型常量的表示形式。

- 定点数形式：它由数字和小数点组成，如 0.123、.234、0.0 等。
- 指数形式：数字+E（或 e）+整数。E 前必须有数字，E 后必须是整数。例如，123e5 或 123E5 都表示 123×10^5。

要注意，E 或 e 的前面必须要有数字，且 E 后面的指数必须为整数。

3. 字符常量

字符常量是由一对单引号引起来的字符，其值为引起来的字符在 ASCII 表中的编码，所以字符和整数可以互相赋值，例如：

```
char c=98;
```

提 示
将常量尽量局部化，如果本模块使用，甚至只有本文件（.cpp）使用，就限制在其中。很多常量不是全局都会使用，本模块内部的，一定不要对外部公开，除非是关键的全局常量。

3.2 基本变量类型

前面讲了变量的基本定义以及如何操作。本节将介绍 C++中常用的几个基本的变量类型如何使用。

提 示
变量是存放在内存中的，程序根据内存地址来访问变量。

3.2.1 整数类型

在现实生活中，整数是大家常用的描述方式，在 C++中则用整型来描述整数。整型规定了整数的表示形式、运算和表示范围。

在 C++中，整型数据类型是用关键字 int 声明的常量或变量，其值只能为整数。根据 unsigned、singed、short 和 long 等修饰符，整型数据类型可分为 4 种，分别对应无符号整型、有符号整型、短整型和长整型。在 C++中，整型变量的声明方式如下：

```
[修饰符] <int>  <变量名>
```

每种整型变量都有不同的表示方式，表 3-1 说明了每种整型变量的取值范围。

表3-1 整形变量的取值范围

类型	长度	取值范围
int	32	-2 147 483 648~2 147 483 648
short int	16	-32 768~32 768
long int	32	-2 147 483 648~2 147 483 648
unsigned int	32	0~4 294 967 295
unsigned short	16	0~65 535
unsigned long	32	0~4 294 967 295

下面通过一个实例来说明 int 类型的使用方法。

【实例 3-5】int 类型的用法（代码 3-5.txt）

新建名为“inttest”的【C++ Source File】源程序，源代码如下所示：

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

```
    int a;
    //定义整型变量
    a=100;
    //变量赋初值
    cout<<"a="<<a<<endl;
    cout<<"size a  "<<sizeof(a)<<endl;
    //输出 a 的值
    short int b=100.01;
    //变量赋值
    cout<<"b="<<a<<endl;
    cout<<"size b  "<<sizeof(b)<<endl;
    //输出 a 的值
    system("pause");
    return 0;
}
```

【代码详解】

在该例的主程序中，首先定义了一个整型变量 a，给该变量赋值 100。接下来，输出该变量的值和该变量所占内存空间的大小。下面定义了 short int 型变量 b，对该变量赋值 100.01，然后输出该变量的值和该变量占用的内存大小。

运行结果如图 3-5 所示。

图 3-5　代码运行结果图

【实例分析】

从整个示例来看，使用 int 实现了定义整型变量的操作，使用 short int 实现了定义短整型变量的操作。整型变量与短整型变量的区别在于它们的取值范围不同。

提　示
整型数据在溢出后不会报错，达到最大值后又从最小值开始记。在编程时，注意定义变量的最大取值范围，一定不要超过这个取值范围。

3.2.2　字符类型

在 C++中，字符型数据类型只占据 1 字节，其声明关键字为 char。同样，可以给其加上 unsigned、singed 修饰符，分别表示无符号字符型和有符号字符型。在 C++中，字符型变量的声明方式如下：

```
[修饰符] <char> <变量名>
```

在 ASCII 中，共有 127 个字符。其中 1～31 和 127 为不可见字符，其余全部为可见字符。

提　示
字符是为针对处理 ASCII 码字符而设的，字符在表示方式和处理方式上与整数吻合，在表示范围上是整数的子集，其运算可以参与到整数中去，只要不超过整数的取值范围。

计算机不能直接存储字符，所以所有字符都是用数字编码来表示和处理的。例如，a 的 ASCII 码值是 97，A 的 ASCII 码值是 65，等等。如果一个字符被当作整数使用，其值就是对应的 ASCII 码值；如果一个整数被当作字符使用，该字符就是这个整数在 ASCII 码表中对应的字符。

通常在 C++中，单个字符使用单引号表示。

例如，字符 a 可以写为 'a'。

单引号只能表示一个字符，如果字符的个数大于 1，就变成了字符串，只能使用双引号来表示了。

在 C++中，还有一些比较特殊的字符，这些字符是以转义符（"\"）开头的，称为转义字符。表 3-2 列出了转义字符。

表3-2　转义字符

转义字符	含义	ASCII 值
\a	响铃（BEL）	007
\b	退格（BS）	008
\f	换页（FF）	012
\n	换行	010
\r	回车（CR）	013
\t	水平制表（HT）	009
\v	垂直制表（VT）	011
\\	反斜杠	092
\?	问号字符	063
\'	单引号字符	039
\"	双引号字符	034
\0	空字符（NULL）	000

下面通过一个实例来说明字符的使用方法。

【实例 3-6】字符类型（代码 3-6.txt）

新建名为“chartest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
using namespace std;
int main()
{
    char cch;
    //定义字符型变量
    cch='A';
    //变量赋值
    cout<<"cch="<<cch<<endl;
    int ich;
    //定义整型变量
```

```
    ich='A';
    //变量赋值
    cout<<"ich="<<ich<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在本例中，首先定义了一个 char 型变量 cch，其后给 cch 赋值为'A'，将字符变量 cch 输出。再定义一个 int 型变量 ich，给它赋值也是'A'，然后将该变量输出。运行结果如图 3-6 所示。

图 3-6　代码运行结果图

【实例分析】

从结果来看，定义了字符型数据 cch 和整型数据 ich，给它们赋值都为字符'A'，输出后其结果不同，整型变量 ich 的输出为 65。这是因为字符型数据在计算机内部是转换为整型数据来操作的，如上述代码中的字母 A，系统会自动将其转换为对应的 ASCII 码值 65。

3.2.3　浮点数类型

浮点数类型表示的是带有小数点的数据。在 C++中，浮点数类型分为以下 3 种。

提　示
浮点数内部表示特殊，操作不同于整数，能够表示的大小范围比同样大小的整数空间大很多，在两个连续的整数之间能够表示很多精细的数值。

（1）单精度浮点型（float）：专指占用 32 位存储空间的单精度值。当用户需要小数部分并且对精度的要求不高时，单精度浮点型的变量是有用的。下面是一个声明单精度浮点型变量的例子。

```
float a,b;
```

（2）双精度浮点型（double）：占用 64 位的存储空间。当用户需要保持多次反复迭代的计算的精确性时，或在操作值很大的数据时，双精度型是最好的选择。例如，前面计算圆周长，声明的常量和变量均为双精度型，代码如下：

```
double radius,area;
```

（3）扩展精度浮点型（long double）：占用 80、96 或者 128 位存储空间。

“精度”是指尾数中的位数。通常 float 类型提供 7 位精度，double 类型提供 15 位精度，long double 类型提供 19 位精度，但 double 类型和 long double 类型在几个编译器上的精度是相同的。除

了精度有所增加之外，double 类型和 long double 类型的取值范围也在扩大。

表 3-3 说明了浮点数的取值范围。

表3-3　浮点数的取值范围

类型	精度	取值范围
float	7	1.2×10^{-38}～3.4×10^{38}
double	15	2.2×10^{-308}～1.8×10^{308}

显然，这些类型都可以表示 0，但不能表示 0 和正负范围中下限之间的值，所以这些下限是非 0 值中最小的值。

下面通过一个实例来说明浮点数的应用。

【实例 3-7】浮点数应用（代码 3-7.txt）

新建名为“floattest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    float a;
    //定义浮点型变量
    double b;
    //定义浮点型变量
    a=3.1415926;
    //变量赋初值
    b=3.1415926;
    cout<<"a="<<a<<endl;
    //输出变量的值
    cout<<"b="<<b<<endl;
    cout<<"b="<<setprecision(9)<<b<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，首先定义了一个 float 类型的变量 a，又定义了一个 double 类型的变量 b。给 a 和 b 赋值为 3.1415926，将 a 和 b 先输出，再调用 setprecision 函数保留 9 位小数输出 b。

运行结果如图 3-7 所示。

图 3-7　代码运行结果

【实例分析】

从运行结果来看，无论定义的变量为单精度数据类型 float 还是双精度数据类型 double，其输出的小数位都相同，这是因为没有设置输出精度，系统默认输出 6 位小数（包括小数点）。若需要 double 型变量输出更多的小数位，则应设置精度。

3.2.4 布尔类型

布尔类型在 C++中表示真假，用 bool 表示，其直接常量只有两个：true 和 false，分别表示逻辑真和逻辑假。同样，如果要把一个整型变量转换成布尔型变量，其对应关系为：若整型值为 0，则其布尔型值为假（false）；若整型值为 1，则其布尔型值为真（true）。

提　示
bool 型输出形式可以选择，默认为整数 1 和 0，若要输出 true 和 false，则可以使用输出控制符 d。

下面通过一个实例来说明布尔类型的使用。

【实例 3-8】布尔应用（代码 3-8.txt）

新建名为“booltest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
using namespace std;
int main()
{
    bool bflag;
    //定义布尔型变量
    int iflag;
    //定义整型变量
    bflag=true;
    //变量赋值
    iflag=true;
    cout<<"bflag="<<bflag<<endl;
    //输出变量的值
    cout<<"iflag="<<iflag<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，首先定义了一个 bool 类型的变量 bflag，又定义了一个 int 型的变量 iflag。给 bflag 和 iflag 都赋值为 true，将 iflag 和 bflag 输出。

运行结果如图 3-8 所示。

图 3-8　代码运行结果

【实例分析】

从运行结果来看，上述程序定义了布尔型变量 bflag 和整型变量 iflag，给其赋值后输出。可以看到，其输出并不是 true，而都是整数值 1，这是使用布尔类型数据时需要注意的。

3.3 typedef

在现实生活中，信息的概念可能是长度、数量和面积等。在 C++语言中，信息被抽象为 int、float 和 double 等基本数据类型。从基本数据类型名称上不能够看出其所代表的物理属性，并且 int、float 和 double 为系统关键字，不可以修改。

为了解决用户自定义数据类型名称的需求，C++语言中引入了类型重定义语句 typedef，可以将已有的类型名用新的类型名代替，从而丰富数据类型所包含的属性信息。

typedef 的语法描述：

```
typedef 类型名称 类型标识符;
```

typedef 为系统保留字，“类型名称”为已知数据类型名称，包括基本数据类型和用户自定义的数据类型，“类型标识符”为新的类型名称。

例如：

```
typedef double LENGTH;
typedef unsigned int COUNT;
```

定义新的类型名称之后，可像基本数据类型那样定义变量，例如：

```
typedef unsigned int COUNT;
unsigned int b;
COUNT c;
```

typedef 的主要应用有如下几种形式：

（1）为基本数据类型定义新的类型名。

（2）为自定义数据类型（结构体、公用体和枚举类型）定义简洁的类型名称。

（3）为数组定义简洁的类型名称。

（4）为指针定义简洁的名称。

typedef 主要有以下用途。

（1）定义一种类型的别名，而不只是简单的宏替换，可以用来同时声明指针型的多个对象，例如：

```
char* pa, pb; //
```

这个声明只声明了一个字符指针 pa 和字符变量 pb，而不是声明了两个字符指针。

使用 typedef 可以声明两个字符指针：

```
typedef char* PCHAR;    // 一般用大写
PCHAR pa, pb;           // 可行，同时声明了两个指向字符变量的指针
```

虽然使用以下语句也可行：

```
char *pa, *pb;
```

但相对来说没有用 typedef 的形式直观，尤其在需要大量指针的地方，typedef 的方式更省事。

（2）用在旧的 C 代码中，声明 struct 新对象时，必须要带上 struct，即形式为“struct 结构名 对象名”，如：

```
struct tagPOINT1
{
int x;
int y;
};
struct tagPOINT1 p1;
```

而在 C++中，可以直接写“结构名 对象名”，即

```
tagPOINT1 p1;
```

为了简化 struct 的定义，在 C++中使用 typedef 来定义：

```
typedef struct tagPOINT
{
int x;
int y;
}POINT;
POINT p1; //
```

这样就比原来的方式少写了一个 struct，比较省事，尤其在大量使用的时候。

或许，在 C++中，typedef 的这种用途并不是很大，但是理解它对掌握以前的旧代码还是有帮助的。

（3）用 typedef 来定义与操作系统无关的类型。

比如定义一个叫 REAL 的浮点类型，在目标操作系统一上，让它表示最高精度的类型为：

```
typedef long double REAL;
```

在不支持 long double 的操作系统二上，改为：

```
typedef double REAL;
```

在连 double 都不支持的操作系统三上，改为：

```
typedef float REAL;
```

也就是说，当跨平台时，只要修改 typedef 本身就行，不用对其他源码做任何修改。

标准库就广泛使用了这个技巧，比如 size_t。

另外，因为 typedef 是定义了一种类型的新别名，不是简单的字符串替换，所以它比宏有更好的稳定性。

（4）为复杂的声明定义一个新的简单的别名。方法是：在原来的声明里逐步用别名替换一部

分复杂声明，如此循环，把带变量名的部分留到最后替换，得到的就是原声明的最简化版。例如：

①原声明：

```
int *(*a[5])(int, char*);
```

变量名为 a，直接用一个新别名 pFun 替换 a 就可以了：

```
typedef int *(*pFun)(int, char*);
```

原声明的最简化版：

```
pFun a[5];
```

②原声明：

```
void (*b[10]) (void (*)());
```

变量名为 b，先替换右边部分括号里的，pFunParam 为别名一：

```
typedef void (*pFunParam)();
```

再替换左边的变量 b，pFunx 为别名二：

```
typedef void (*pFunx)(pFunParam);
```

原声明的最简化版：

```
pFunx b[10];
```

③原声明：

```
doube(*)() (*e)[9];
```

变量名为 e，先替换左边部分，pFuny 为别名一：

```
typedef double(*pFuny)();
```

再替换右边的变量 e，pFunParamy 为别名二：

```
typedef pFuny (*pFunParamy)[9];
```

原声明的最简化版：

```
pFunParamy e;
```

在理解复杂声明时，可以使用“右左法则”：

从变量名看起，先往右，再往左，碰到一个圆括号就调转阅读的方向；括号内分析完就跳出括号，还是按先右后左的顺序，如此循环，直到整个声明分析完。

举例：

```
int (*func)(int *p);
```

首先找到 func，外面有一对圆括号，而且左边是一个*号，这说明 func 是一个指针；然后跳出这个圆括号，先看右边，又遇到圆括号，这说明(*func)是一个函数，所以 func 是一个指向这类函数的指针，即函数指针，这类函数具有 int*类型的形参，返回值类型是 int。

```
int (*func[5])(int *);
```

func 右边是一个[]运算符，说明 func 是具有 5 个元素的数组；func 的左边有一个*，说明 func 的元素是指针（注意这里的*不是修饰 func 的，而是修饰 func[5]的，原因是[]运算符的优先级比*高，func 先跟[]结合）。跳出这个括号，看右边，又遇到圆括号，说明 func 数组的元素是函数类型的指针，它指向的函数具有 int*类型的形参，返回值类型为 int。

【实例 3-9】typedef 应用（代码 3-9.txt）

新建名为“typetest”的【C++ Source File】源程序，源代码如下所示：

```
#include <iostream>
using namespace std;
typedef unsigned int UINT;
int main (int argc, char *argv[])
{
    unsigned int a;
    a=125;
    UINT b;
    b=222;
    cout<<"a="<<a<<endl;
    cout<<"sizeof a="<<sizeof(a)<<endl;
    cout<<"b="<<b<<endl;
    cout<<"sizeof b="<<sizeof(b)<<endl;
    system("pause");
    return 0;
}
```

【代码详解】

在该例中，使用 typedef 定义了一个 unit 类型，该类型等同于 int 型。在主程序中，定义了一个 int 型变量 a，给 a 赋值为 125；定义了一个 unit 型变量 b，给它赋值为 222。将 a 的值和 a 的大小输出，将 b 的值和 b 的大小输出。

运行结果如图 3-9 所示。

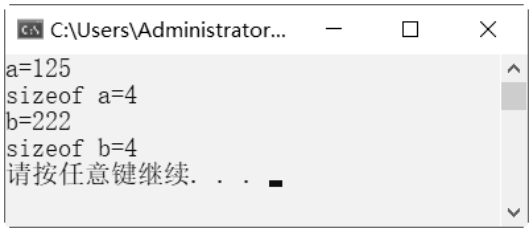

图 3-9　代码运行结果

【实例分析】

从运行结果来看，a 和 b 属于同一种数据类型（unsigned int 型），因为 UINT 标识符已经定义为 unsigned int 类型。

3.4　小试身手——测试基本数据类型的字节长度

本节通过一个综合实例来讲述如何测试计算机中数据类型的字节长度，程序源代码如下：

```
#include<iostream>
using namespace std;
void main()
{
    cout<<"The size of an int is:"<<sizeof(int)<<"bytes\n";
    cout<<"The size of a short int is: "<<sizeof(short)<<"bytes\n";
    cout<<"The size of a long int is: "<<sizeof(long)<<"bytes\n";
    cout<<"The size of a char is: "<<sizeof(char)<<"bytes\n";
    cout<<"The size of a float is: "<<sizeof(float)<<"bytes.\n";
    cout<<"The size of a double is:"<<sizeof(double)<<"bytes.\n";
    system("pause");
}
```

【代码详解】

在该例中，使用 sizeof 分别输出了 int、short、long、char、float、double 在计算机中所占的字节数。

运行结果如图 3-10 所示。

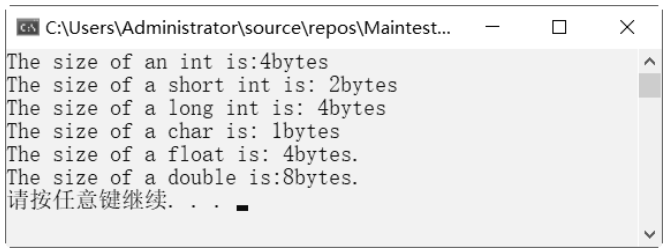

图 3-10　代码运行结果

【实例分析】

从运行结果来看，int、long、float 占 4 字节，double 占 8 字节，short 占 2 字节，char 占 1 字节。可见，不同数据类型所占用的字节数也不相同。

下面通过案例来理解变量的初始化。

```
#include <iostream>
using namespace std;
int main ()
{
    int a=5;    // 初始值为 5
    int b(2);   // 初始值为 2
    int result; // 不确定初始值
    a = a + 3;
    result = a - b;
    cout << result<<endl;
    system("pause");
    return 0;
```

```
}
```

【代码详解】

在该例中，首先定义了 int 型变量 a，赋值为 5，然后定义了 int 型变量 b，赋值为 2，接着定义了 int 型变量 result，给 a 赋值为 a+3，给 result 赋值为 a-b，最后将 result 的结果输出。

运行结果如图 3-11 所示。

图 3-11　代码运行结果

【实例分析】

从运行结果来看，定义了 int 型变量，并且对 int 型变量进行了简单的加减运算，在定义 a 和 b 时，分别使用了两种不同的定义方法。

3.5　疑难解惑

疑问 1　C++在代码移植中，使用整型时应注意什么?

（1）出于效率考虑，应该尽量使用 int 和 unsigned int。

（2）当需要指定容量的整型时，不应该直接使用 short、int、long 等，因为在不同的编译器上它们的容量不相同。此时应该定义它们相应的宏或类型。例如在 Visual C++6.0 中，可以如下定义：

```
typedef unsigned char UBYTE;
typedef signed char SBYTE;
typedef unsigned short int UWORD;
typedef signed short int SWORD;
typedef unsigned int UDWORD;
typedef signed int SDWORD;
typedef unsigned   int64 UQWORD;
typedef signed __int64 SQWORD;
```

在代码中使用 UBYTE、SBYTE、UWORD 等，这样当代码移植的时候只需要修改相应的类型即可。

疑问 2　在 C++中，0 所扮演的不同角色是什么？

（1）整型 0

这是最熟悉的一个角色。作为一个 int 类型，整型 0 占据 32 位的空间。

（2）空指针 NULL

NULL 是一个表示空指针常量的宏。

（3）字符串结束标志'\0'

'\0'与上述两种情形有所不同，它是一个字符。

（4）逻辑 FALSE/false

虽然将 FALSE/false 放在了一起，但是你必须清楚 FALSE 和 false 之间不只是大小写这么简单的差别。false/true 是标准 C++语言里新增的关键字，而 FALSE/TRUE 是通过#define 定义的宏，用来解决程序在 C 与 C++环境中的差异。

疑问 3　typedef 和 define 的区别是什么?

在某些情况下，使用它们会达到相同的效果，但是它们有实质性的区别，一个是 C/C++的关键字，一个是 C/C++的宏定义命令，typedef 用来为一个已有的数据类型起一个别名，而#define 用来定义一个宏定义常量。

3.6　经典习题

（1）编写一个程序，计算用户输入非 0 整数的倒数，该程序应把计算的结果存储在 double 类型的变量中，再输出它。

（2）编写一个程序，从键盘上读取 4 个字符，把它们放在一个 4 字节的整型变量中，把这个变量的值显示为一个十六进制；分解变量的 4 字节，以相反的顺序输出它们，先输出低位字节。