第1章

C++ 语言概述

C 语言和 C++ 语言是当今世界上最流行的编程语言。它们功能强大，应用面广，譬如我们正在使用的 Windows 操作系统的很大一部分，就是使用 C 语言编写成的。

C++ 和 C 的关系如图 1-1 所示。C 是 C++ 的一个子集，不包含 C++ 中的面向对象(OOP)部分。C++ 具有面向对象的优点，成为继 C 语言之后另一个令人不敢忽视的“风云人物”。

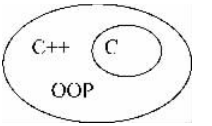

图 1-1　C++ 与 C 的关系图

1.1　认识 C++ 语言

C++ 语言是由 AT&T Bell 实验室的 Bjarne Stroustrup 博士在 20 世纪 80 年代初期提出的。最初的版本叫“C with Classes”(带类的 C)。这表明 C++ 语言在语法上对 C 的主要扩展是对类结构的实现。C++ 的第一个版本于 1979 年 8 月开始在 AT&T 内部使用。同一年晚些时候，开始使用“C++”这个名字。第一个商业 C++ 编译器在 1985 年 10 月发布，同时出版了《C++ Programming Language》的第一版。模板和异常处理是在 20 世纪 80 年代后期被加入的，它们在《The Annotated C++ Reference Manual》和《The C++ Programming Language (2rd Edition)》中被详细描述。

目前的 C++ 是由 ISO C++ 标准定义的，该 C++ 标准(ISO/IEC 14882)在 1998 年以 22 比 0 的投票结果获得批准，并且在《The C++ Programming Language (3rd Edition)》中描述。

但是，标准并不是一个教程，即使是专家级程序员也最好从教科书开始学习 C++ 以及了解C++ 的新特性。

C++ 的新特性如下：

以类定义的形式实现了“对象”，类中不仅包括 C 结构中的数据定义，而且还包括对数据进行操作的函数的声明和定义，这种把数据和函数封装在一个对象中的技术是 C++ 的一个主要革新。

类的实例可以用构造函数和析构函数自动进行初始化和释放，这就消除了程序的初始化错误。

C++ 中类的定义方式增强了数据隐藏性，在默认情况下，类中定义的数据只能被类中的成员函数引用。外部（客户）程序在使用类时不会改变类的内部实现，它们只能通过调用成员函数来访问类。

C++ 允许对操作符和函数进行重载。对一个函数的多个定义可以采用相同的名字，编译器可以在函数调用时识别出合适的定义形式。像“++”和“->”这样的普通操作符也可以被重载为加法的含义。

C++ 允许“类”类型的特征——数据和函数——被子类所继承。子类也称为派生类，子类反过来可以添加更多的数据和函数定义。这就鼓励了用共享类库的方式重用已有代码，从而节省了软件开发过程中的费用。多继承允许派生类从多个基类中继承特征。

C++ 允许类定义虚函数：一个函数有多个定义，在程序进行时决定使用哪个定义。这叫做多态性，它的意思就是在运行时才从函数定义中进行选择，这称为后期绑定（late binding）或动态聚束（dynamic binding）。

可定义模板类，它允许在代码不变的情况下，用不同类型的数据定义同一个类的不同实例。这更进一步提高了代码的重用率。

C++ 中面向对象程序设计的工具是类、继承和虚函数。这些工具使 C++ 语言非常适合编写处理大量相关对象的软件。

和 C 语言相比，C++ 有许多较小的语法改进。它提供了一种新的引用机制，从而对 C 语言中的指针间接引用进行了补充。C++ 简化了动态分配和释放内存的过程，并且实现了一种新 I/O 流库，它用层次分明的类对输入和输出流进行了定义。

C++ 是 C 的扩展，它通过简化设计，使软件能够更好地反映真实世界，降低代码的长度和复杂性，从而增加代码的可靠性。

以上已经对 C++ 的内容做了介绍，让我们开始写第一个程序吧！

1.2 第一个 C++ 程序

C++ 程序的文件扩展名没有统一的标准。在 UNIX 操作系统下，C++ 源代码文件名可以用.c 或者.C 结尾。对许多个人计算机系统和工作站来说，其扩展名是.cpp。一些基于个人计算机的 C++ 编译器要求为.cxx。本书我们只使用常用的.cpp。

利用 Visual Studio 可以开发许多各式各样的程序，从数学计算机辅助程序、图像处理程序、特效处理甚至是开发游戏程序等，确实用处很多。以下的例子将通过 Visual Studio 内建的应用程序向导生成一个 C++ 的程序。

步骤 1：打开工程

双击 Microsoft Visual C++ 6.0 软件图标，执行“文件”→“新建”命令，打开 Visual Studio 工程向导。

步骤 2：选择 C++ Source File 选项

在“新建”窗口中打开“文件”选项卡，在清单中选择 C++ Source File 选项，如图 1-2 所示。

图 1-2　新建 C++ 源文件

步骤 3：生成应用程序

单击“确定”按钮后，生成了应用程序，包含文件、程序代码、注释等。如果不需要加上额外的程序，即可按下键盘上的 F7 键，Visual Studio 将为程序进行编译连接，如图 1-3 所示。

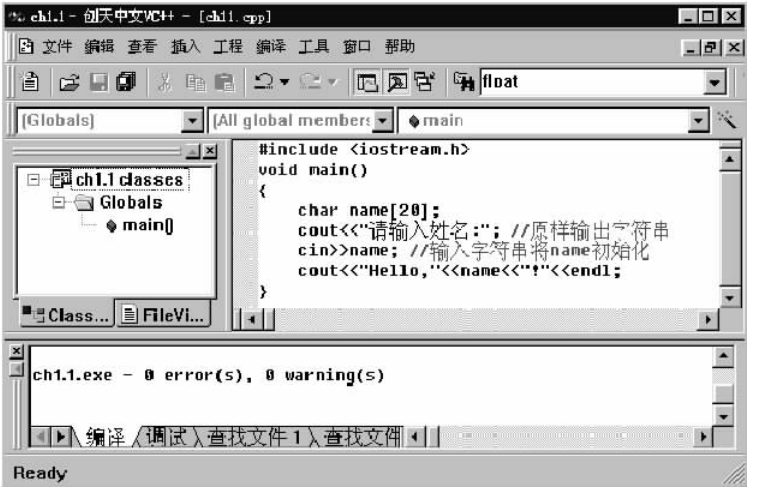

图 1-3　编译和连接结果与 C++ 源文件

现在可以按下 Ctrl＋F5 键执行程序了，执行结果如图 1-4 所示。

图 1-4　ch1.1 执行结果图

说明：在传统的 C 语言中提供了块注释方式，其形式为：/ * … * /。而在 C++ 中，除了保留了块注释方式外，还增加了一种更为方便的单行注释方式，其形式为：//…。

iostream.h 是一个标准头文件，它包含了应用程序在进行编译和执行时所需要的声明。iostream.h 和 C 语言中

的 stdio.h 头文件等同,但不可以代替它。iostream.h 声明了 C++ 库函数和工具; stdio.h声明了标准 C 库函数,比如 printf 等。而这些声明对 C++ 流 I/O 库的使用是必要的。

cout 工具代表标准输出流对象。假如终端屏幕是标准输出设备,操作符"<<"右面的字符被送到 cout,然后 cout 将这些字符显示在用户终端上。

cin 工具用于读取用户由键盘输入的数据,直到用户按 Enter 键为止; 其中">>"表示将会依次将数据移入对应的变量中,如果输入的数据包含两个以上的数值,则以空格作为间隔符号。

main()前面的 int(integer)定义当程序执行时向操作系统返回一个数值(一般为 0)。\n 和 endl 为换行字符。

1.3 C++ 对 C 的扩展

C++ 除了继承了 C 语言中的精华和增加了许多面向对象的特征之外,同时又将 C 语言的不足和问题做了很多改进。下面将总结从 ISO C 到 C++ 中最重要的语法变化,而不讨论 C++ 所增加的面向对象的程序设计工具和 I/O 流、模板以及其他库等。

1. C++ 提供了单行注释方式

在传统的 C 语言中提供了块注释方式,其形式如下:

```
/* explanation sentence */
```

而在 C++ 中,除了保留了块注释方式外,还增加了一种更为方便的单行注释方式,其形式如下:

```
// explanation sentence
```

2. 变量作用域

在传统的 C 语言中,局部变量的说明必须放在可执行代码的前面,数据说明语句和可执行语句的混合将引起编译错误。C++ 提供了(几乎是)在一个函数的任意地方声明并使用一个变量的功能,所说明变量的作用域是从对该变量进行说明的地方开始到该变量所在的最小分程序的末尾。如下面的代码段:

```
#include <iostream.h>
int main()
{
    for(int x = 5; x<10; x++ )
    cout<<"x 的值: "<<x<<endl;
    int y = 65;
    cout<<"x 和 y 的值分别为: "<<x<<y<<endl;
    return 0;
}
```

说明：在C++中，声明语句不一定在函数的开始部分，而是可以跟在其他语句的后面。本例中把变量和初始化当成for循环中控制语句的一部分。

```
#include "iostream.h"
// 首先定义一个全局变量
int temp;
int main(int argc,char * argv[])
{
// 现在再定义一个同名的内部变量
int temp;
// 下面可以使用这个变量
temp = 10;
// 使用全局变量
::temp = 6;
// 在屏幕上显示结果
cout << temp <<"\n"<<::temp <<"\n";
return 0;
}
```

运行后，将在屏幕上输出10，换行，6，换行。

说明：::是变量域运算符，::temp表示引用全局变量中的temp。

3. 关键字

C++引入了许多新关键字，它们包括：class、delete、friend、inline、new、operator、private、protected、public、template、virtual。

这些关键字将在后面进行解释，任何使用这些关键字做变量的程序都是不合法的C++程序。

4. 函数重载

函数重载提供了一个定义和使用变量的方便、强大的功能。在C语言中，如果我们需要一个求两个整数最大值的函数，可以这样写："int Max_Int(int,int)；"。如果我们又需要一个求两个浮点数最大值的函数，我们又得定义："float Max_Float(float,float)；"。如果又要求其他类型的呢？必须定义不同的其他函数。而C++允许同名函数，只要定义的参数类型或个数不同，编译器会自动进行连接，如int Max(int,int)、float Max(float,float)等。

5. 操作符重载

C++允许对诸如＋、＝、<<、>>、＋＝等运算符进行重载，以适应不同的需要。这一特性对于面向对象程序设计来说是非常必要的。在前面的例子中使用了cout << temp << "\n"<<::temp <<"\n"；其中cout是一个在iostream.h中定义的屏幕对象，这个头文件中已经对"<<"进行了重载，使它可以支持诸如整数、浮点数、字符、字符串等常用数据类

型的输出，所以我们可以简单地使用 cout << temp 来输出一个变量而不用指明类型。

6. 默认参数

在 C 语言中，函数调用必须按照参数表全部显式传送。C++ 允许使用默认参数。如已有函数声明 int AFunction(int num=10)，在调用时可以直接写 AFunction()，编译器会自动认为我们接受了 num=10 这个参数。这一特性对于参数很多，而大多数一般又无须特殊设置的函数调用显得很方便。

注意：对于多个参数默认的情况，如果某个参数缺省的话，则其后边的参数也只能取默认值。对于设计函数来说，诸如 void function(int a=0,int b,int c=2)这样的声明就不那么聪明了。事实上，在这里 a 是不能缺省的。但若改为 void function(int b,int c=2,int a=0)就好了。修改后，对于调用函数来说，如果想接受 a 的默认值，那么必须同时接受 c 的默认值。

看以下例子：

```
void function(int a = 0,int b = 1,int c = 2);
{
  ⋮
}
```

以下的函数调用都是合法的。

```
function(); //equal to function(0,1,2)
function(12); //equal to function(12,1,2)
function(12,13); //equal to function(12,13,2)
function(12,13,14);
```

而以下的函数调用则是非法的。

```
function(,13,14);
function(12,,14);
```

7. 按引用传送

C 语言的一些特性有时候不太理想。例如，它在传递参数到函数时，要先把变量复制一份，然后把副本送给函数。一方面，对于较大类型的变量（如结构或对象），这样传送参数显然效率很低。另一方面，在函数内部只可以修改副本，而无法改变变量本身。在 C 语言中，解决这一问题的方法是使用指针，即按地址传送。C++ 继承了这一特性，而且还引入了“引用”的概念。例如：

```
void Fn(int &nAnotherVar)
{
nAnotherVar = 10;
}
int main()
{
```

```
int nAVar = 5;
// 调用函数
Fn(nAVar);
// 现在,nAVar 的值变为 10 了
}
```

C++ 使用“&”表示引用变量(传送变量的地址而不是变量值),这样就完全避开了指针变量。不过由于指针的重要性,诸如 Windows、UNIX、Linux 这样的操作系统的编程接口(API)都是基于 C 语言而写的,只能使用指针。“引用”这一概念并不能像其他特性一样引人注目。

8. 内联函数

C 语言中一个比较麻烦的特性是 # define 宏指令。宏指令与其他的语句不同,在编译之前宏指令会被预处理程序翻译解释。而预处理程序的规则显然与编译程序不同,且预处理程序不像编译程序那么灵活。

有两种方法可以解决这一问题。方法一是在 C++ 中禁止使用 # define 宏指令,而要求写成函数的形式。C++ 的设计人员没有这么做,原因是宏指令在预处理时展开为代码,而函数则放在不同的段内,调用时需额外的开销。在这一方面,宏指令具有函数调用所没有的优点。方法二是 C++ 的设计人员引入了 inline 关键字。在一个函数前面加上 inline 表示这是个内联函数,例如:

```
// MIN——宏指令
#define MIN(x,y) (((x)>(y))?(y):(x))
// MIN——内联函数
inline int Min(int nx,int ny)
{
if(nx>ny)
{
return ny;
}
return nx;
}
void SomeFunction(int nx,int ny)
{
// 调用内联函数
nx = Min(nx,ny);
}
```

内联函数与普通函数具有相同的语法,只是它们被调用的时候会像宏调用那样被扩展。因此它们更像一个 # define 宏指令而非一个普通函数。

9. void 类型

在 C++ 中,void 类型指针是可以与任何类型指针相匹配的指针,void 类型指针可以

赋给任何类型的指针，从而可以避免许多无用的类型转换。而 void 类型函数是无返回值的函数，它可以限制函数返回一个无用的 int 型数值。

10. 关键字 const

在 C++ 中，关键字 const 用于将一个标识符说明为常量，即其值在程序的运行过程中不变的量，程序不能以任何方式对其进行修改。而 C++ 中的 const 与 C 语言中的 #define 是有区别的：const 所说明的常量是有内存单元与之相对应的量，在程序编译时其值还不能确定；而 #define 所说明的是在编译时便能确定其数值的常量。

关键字 const 也可以用来修饰函数参数表中的某些参数，用以保证被修饰的参数对应的实参在该函数内部不被改动。

当关键字 const 用来修饰指针时，根据使用的形式不同，它可以冻结指针所指向的变量、冻结指针本身或同时冻结指针及其所指向的变量。如下所示：

```
const char *name = "Mike"; //冻结指针所指向的变量
char *const name = "Mike"; //冻结指针本身
const char *const name ="Mike"; //同时冻结指针及其所指向的变量
```

11. new 和 delete

C++ 为了提高内存管理上的灵活性，提供了动态内存分配和释放的 new 和 delete 操作符，用来作为 C 语言中 malloc()和 free()函数的替换。它们以下面的方式进行。

```
#include <iostream.h>
int main()
{
    int *p;
    p = new int;
    *p = 67;
    cout<<"p 的值: "<<p<<" *p 的值: "<<*p<<endl;
    delete p;
    return 0;
}
```

new 为一个整数分配内存堆空间，并向 p 返回一个指向这个空间的指针，从此以后，指针的间接引用将被用来访问 p 所存储的内容。

1.4 编写并运行 C++ 程序

1.4.1 类

C++ 语言引入了类，使它成为面向对象的程序设计语言，这是 C 语言功能的一个重要扩展。类是 C 语言中结构(struct)概念的推广。

类(class)定义中默认情况下的成员是 private 的，而结构(struct)定义中默认情况下

的成员是 public 的。

在 C 中，结构中不允许有成员函数，只允许有数据成员。和结构相同的是，类中包含了许多成员(members)；和结构不同的是，这些成员可以是成员函数，也可以是数据成员。可以利用类来描述现实世界中的对象，比如长方体。

长方体的属性包括长、宽和高，在这些信息上至少可以进行两种操作：①计算体积；②计算表面积(本例子以求体积操作为例说明)。利用 C++ 的类，可以用下面的方法来记录这些数据和操作：

```
class rectangle
{
protected:
    int length;
    int width;
public:
    void area(int l,int w);
}; //定义类后要加分号
```

这是一个类声明；可以定义一个类的实例，它的实例被称为类对象或类变量，就像 C 语言中对结构(struct)实例进行定义一样：

```
class rectangle myrectangle;
```

或

```
rectangle myrectangle;
```

一般用第二种定义，因为 rectangle 本身就是一个类型，所以在这里就没有必要使用关键字 class 了。

当对类 rectangle 进行编码时，很可能会在程序中将类的声明存储在一个头文件中(比如 rectangle.h)，用 # include 引导。这样就必须编写代码把类成员函数放在一个程序文件中(比如 rectangle.cpp)。可能会这样定义成员函数 area。

```
void rectangle::area(int l,int w)
{
    int rectarea = l * w;
    cout<<"长方形的面积为："<<rectarea<<endl;
}
```

这个成员函数的函数头形式如下：

```
void rectangle::area(int l,int w)
```

作用域限定符::表明函数 area 是类 rectangle 的一个成员函数，返回值为空(void)，同时接收两个 int 类型的参数。

当声明了类并定义了它的所有成员函数后，类的整个定义就完成了。比如对类的一个实例 myrectangle，现在可以这样计算体积：

```
myrectangle.area(3,4);
```

类 rectangle 分为私有(private)和公有(public)两种。关键字 private 表明在它后面

声明的类成员只能由类 rectangle 的成员函数访问，比如 length 和 width。关键字 public 表明其他任何函数都可以调用这些函数。

类的私有部分增强了数据封装性，数据封装性意味着除了类 rectangle 的私有成员外，其他代码都不能访问类的私有成员。对外部程序所提供的只是类的函数调用接口，而类的内部仍然是一个“黑盒子”。这种原理使得程序高度模块化，程序员只要知道如何调用它就可以了，而不必去了解其中的代码。

使用派生类可以对一般的类 rectangle 做进一步改进。派生类不但具有 rectangle 的特征，同时也增加了自己的新特征。

1.4.2 类的对象的初始化

变量必须初始化。若其初始化过程被省略，程序往往就会发生错误。在 C++ 中，对象变量如何初始化？可以使用构造函数和析构函数自动进行初始化和释放。这使 C++ 程序比 C 程序更加可靠。

构造函数是类的一个成员函数，它对类的一个变量进行初始化。构造函数名一般和类的名字相同。析构函数也是类的一个成员函数，在类实例撤销之前它完成了一些内务管理操作。析构函数的名字和类的名字相同，只是它的前面加了一个“～”符号。

构造函数是类实例定义的一部分，当案例被撤销时，析构函数被自动（而不是显式）调用。下面是使用构造函数和析构函数重新编写的类 rectangle。

```
class rectangle
{
protected:
    int length;
    int width;
public:
    rectangle();
    ～rectangle();
    void area(int l,int w);
};
```

每次定义类 rectangle 的一个实例时，比如说 myrectangle，构造函数 rectangle()便会自动调用。在 myrectangle 创建之后，构造函数立即完成使其成为一个可用的对象所需要的一切操作。当被撤销时，通常在函数末自动调用析构函数～ rectangle()。析构函数在实例 myrectangle 撤销并退出之前立即完成初始化，清除所需要的一切操作。下面是轮流使用构造函数和析构函数的例子。

```
void main()
{
    rectangle myrectangle; //定义类的一个对象，调用构造函数初始化它
    ⋮                      //做其他的一些操作
                           //调用析构函数删除对象
}
```