第 1 章　C++ 编程基础

本 章 要 点

(1) C++ 是一种面向对象的高级程序设计语言，具有良好的数据、功能封装性，可以提高软件的可复用性。

(2) Visual C++ Studio 是 C++ 语言的一种集成开发环境，提供了高效的编辑、编译、连接和调试功能。

(3) 软件书写应当遵循良好的风格，这包括注释、变量命名、编排、简单性和一致性等方面。

1.1　程序语言的发展

1.1.1　机器语言

计算机是人们的好帮手，它有能进行快速处理的 CPU，有能快速存取信息的内存，有能存放大量信息的硬盘。但是，计算机毕竟是一台机器，除非我们告诉它做什么，怎么去做，否则它什么也不会做。

要让计算机做事，必须给它“指令”。可是计算机很笨，它只认识 0 和 1，它的所有操作和数都用 0 和 1 来表示。如计算机中要表示数 5，6，7，8，9，必须分别用四位二进制数来表示：0101，0110，0111，1000，1001。而让计算机进行加、减、乘、除等操作，也必须用二进制数来表示。如果让计算机计算 AL＝4＋5，则需要向计算机发出以下指令：

```
1011 0000 0000 0100
0000 0100 0000 0101
```

这是计算机可以理解的语言，我们称它为机器语言。

1.1.2　汇编语言

计算机用数来“思考”，而人不行。于是聪明的程序员想出了一种办法，用一些人易于理解的符号来代替上面的 0、1 序列，如求 AL＝4＋5 可以写成下面两句话：

```
MOV AL , 4
ADD AL , 5
```

上面的每一句话都代表一个指令。这样的语言很容易被翻译成机器语言，因为翻译是一件很机械的事情，非常适合计算机去完成，所以人们编写了**汇编程序**，专门负责将上

面这种符号语言(称为汇编语言)翻译成机器语言。程序员再也不用与繁琐的 0 和 1 打交道了。

1.1.3 高级语言

随着计算机硬件技术的迅速发展,计算机的处理能力不断提高。人们发现,可以用更自然的方式书写程序,如可以直接在程序中写"AL=4+5"来进行计算。然后将这些程序翻译成为机器能理解的机器语言。于是产生了高级语言,翻译这些高级语言的程序称为"编译程序"。C、FORTRAN、PASCAL、COBOL 等都是高级语言。

为什么不用人类的自然语言来作为计算机语言呢?因为人类语言有一个很大的缺陷:它是不精确的。即使是像法律、合同这样精心书写的条文,也还是可能会存在二义性。而计算机需要的是准确而详尽的指示,自然语言显然无法胜任。计算机高级语言的发展趋势是向自然化发展,但是自然语言无法作为计算机语言。

1.1.4 C 语言

1970 年,两位程序员 Brian Kernighan 和 Dennis Ritchie 在 B 语言的基础上首创了一种新的程序设计语言,取名为 C 语言。设计 C 语言的最初目的是编写操作系统,它是 UNIX 操作系统的开发语言。C 语言有很多优点:与硬件无关,移植方便;语言简洁,使用方便;丰富的运算符和数据类型;可以直接访问内存地址,能进行位操作;生成的目标代码质量高,程序运行效率高。由于这些优点,C 语言很快就被用于编写各种不同类型的程序,从而成为世界上最流行的语言之一。

1.1.5 C++ 语言

C++ 源于 C 语言。随着 C 语言的应用,它的缺点也逐渐显示了出来:C 语言的类型检查机制弱,使得程序开发过程中的错误不能在编译时被发现;C 语言本身是面向过程的语言,没有支持代码复用的机制,因此所有的程序都需要从头开始编制,而且当程序规模达到一定程度时,程序员很难控制程序的复杂性。

20 世纪 80 年代初,美国 AT&T 贝尔实验室的 Bjarne Stroustrup 设计并实现了 C 语言的扩充、改进版本,C++ 语言由此诞生。C++ 语言改进了 C 的不足之处,增加了对面向对象的程序设计的支持,在改进的同时,保持了 C 的简洁性和高效性。C++ 包含了 C 的所有语法,大多数 C 程序都可以简单地转化为 C++ 程序(只是里面不包含 C++ 新的特征),所以原来的一大批 C 程序员可以很容易地成为 C++ 语言的拥戴者。更重要的是,植根于 C 语言的 C++ 语言继承 C 的高效简洁的特点,可以使用比其他大多数语言更高效的方法组织信息。所以,C++ 语言获得了巨大的成功。

时至今日,C++ 越来越受到重视并且已经得到了广泛的应用,许多软件公司如 Microsoft,Inprise 公司都为 C++ 设计编译系统,提供不同应用级别的类库和越来越方便的开发环境。利用 C++ 设计并实现应用系统已是日益简单和快捷的事情了。

1.2 C++ 语言简介

1.2.1 程序＝数据＋操作

计算机语言是作为程序员和计算机之间的桥梁而存在的。计算机本身只是一台机器，它所能做的事情就是在指定的数据上执行指定的操作。因此，要让计算机帮人做事，就必须按计算机的思路来告诉它要做什么事情，也就是程序员需要将要做的事情分解成为“数据＋操作”。

1.2.2 数据类型

计算机中，数据被存储为一系列的字或字节，C++ 语言则将这些字或字节组织成为有用的数据。不同的组织方式得到不同的数据类型，C++ 中有简单的数据类型，如整型、浮点型、字符型等，也有由这些简单类型构造而成的复合的数据类型，如结构、联合。数据用变量或常量来存放。常量存放的是不变的数据，而变量存放的数据可以变化。

1.2.3 对数据的操作——表达式

类似于我们用“(1＋2)＊4＝12”来进行数学运算，C++ 中可以用**操作符**对数据进行操作。操作符和操作数一起构成了**表达式**，根据操作符的不同，表达式可以是算术表达式、逻辑表达式、条件表达式等。表达式可以构成表达式语句。

1.2.4 数据操作的流程——语句控制

语句是 C++ 最小的可执行单元。程序的运行过程就是对语句的执行过程。语句一般按顺序执行，但流程控制语句可以改变程序的执行流程。语句分为声明语句、表达式语句、流程控制语句等。声明语句定义变量或常量；表达式语句在指定数据上执行指定操作；流程控制语句控制程序的执行顺序。

1.2.5 操作的复用——函数

一组相关的语句可以组成**函数**。函数可以完成特定的功能，如计算一个数的平方根、求 N 个数的平均值。函数可以供程序员在程序中反复使用。C++ 提供丰富的标准函数，这些函数可以完成常用的操作，如查找、排序、输入输出、数学运算等。

1.2.6 数据和操作的封装——类

传统的程序设计语言都是将数据和操作分开，而 C++ 的重要突破是：它允许将数据和作用在这些数据上的操作(函数)组合在一起，形成一个整体——**类**，可以将类看成是一种特殊的数据类型，它的特殊性在于这种数据类型带有自己的操作，类的变量就是对象。

1.2.7 类的复用——派生和继承

可以在一个类的基础上派生出新的类，新的类**继承**了原来类的所有特征。继承允许

在较小的类的基础上建立复杂的类，也允许先建立一个抽象的类，再**派生**得到具体的类。

1.2.8 软件模块

在 C ++ 中，可以将一组函数组织成一个**模块**，也可以将一些相关类的定义组织成一个模块，而模块相连就构成了**程序**。

C ++ 语言的一个重要目的就是把指令组织成为可以重复使用的组件。这样，就可以在别人已有的基础上构造自己的程序。类和继承是 C ++ 强大的关键：类和对象符合人们的认识规律，继承使复用变得简单。

C ++ 语言提供了简单而强大的语法，本书不仅讲解 C ++ 的语法，还介绍如何编制“好”的程序。好的编程风格加上强大的语法，就可以创建能实现复杂功能的程序了。

1.3 如何学习 C ++ 语言

1.3.1 勤能生巧

“纸上谈兵”的故事我们都知道。光看兵书是无法真正带兵打仗的，同样，光靠看书也是无法真正学会 C ++ 语言的。学习程序设计语言的惟一方法就是编写程序。在编写程序并上机调试的过程中所学到的知识会比从书本上学到的多得多。书本只能引读者入门，介绍基本的知识、基本的技能以及学习更多知识和技能的方法，大部分知识是需要练习才能掌握的。

因此，学习本书的时候，建议读者坐在计算机旁，一边看书，一边练习。书中的例子可以上机试一试，书后的练习也要自己动手完成。

1.3.2 风格与规范

即使编写的程序只有自己看，最好也要加上注释。在编制非常简单的程序时，可能还体会不到注释的好处，一旦程序复杂，就会发现注释的好处。即使是自己编写的程序，过一段时间以后，也可能发现不可理解。注释可以帮助我们理清思路，提高程序的可读性。

1.4 用 Visual C ++ 开发程序

1.4.1 程序——从概念到运行

用高级语言编写的源程序需要经历编辑、编译、连接才能成为可执行程序。

C ++ 语言是一种高级的程序设计语言，语言中利用字母、数字以及其他各种符号来表达程序员的思想。程序员首先利用**编辑器**将程序输入到计算机中，并以源程序文件的形式存放，组成程序的源程序文件可能有多个。这些源程序文件分别用**编译器**翻译成目标代码，目标代码以汇编语言的形式存放。然后，**连接器**将组成程序的各个目标代码文件和一些库函数连接在一起，组成一个完整的可执行程序。

这里的编辑器、编译器和连接器本身也是程序，它们也是程序员以某种语言书写并翻

译成的可执行程序。目前，大多数开发系统都将这些工具性程序集成起来，形成一个完整的开发环境，称为集成开发环境(Integrated Development Environment，简称 IDE)。程序开发人员不需要脱离该环境，就可以编辑、编译、连接、调试和运行所开发的程序。这些开发环境有 Borland C ++ ，Visual C ++ 等等。

1.4.2 Visual C ++ 简介

Visual C ++ 是 Microsoft 公司的 Visual Studio 开发工具箱中的一个 C ++ 程序开发包。Visual Studio 提供了一整套开发 Internet 和 Windows 应用程序的工具，包括 Visual C ++ ，Visual Basic，Visual FoxPro，Visual InterDev，Visual J ++ 以及其他辅助工具。Visual C ++ 包中除包括开发程序所必需的编辑器、C ++ 编译器、连接程序、调试程序外，还包括所有的库、例子和为创建 Windows 应用程序所需要的文档。Visual C ++ 中的集成开发环境称为 Developer Studio。

从最早期的 1.0 版本，发展到最新的 6.0 版本，Visual C ++ 已经有了很大的变化，在界面、功能、库支持方面都有许多的增强。最新的 6.0 版本在编译器、MFC 类库、编辑器以及联机帮助系统等方面都比以前的版本做了较大改进。

Visual C ++ 一般分为三个版本：学习版、专业版和企业版，不同的版本适合不同类型的应用开发。

1.4.3 建立应用程序

遵循以下步骤试着建立一个应用程序：

第一步：启动集成开发环境 Developer Studio(见图 1.1)

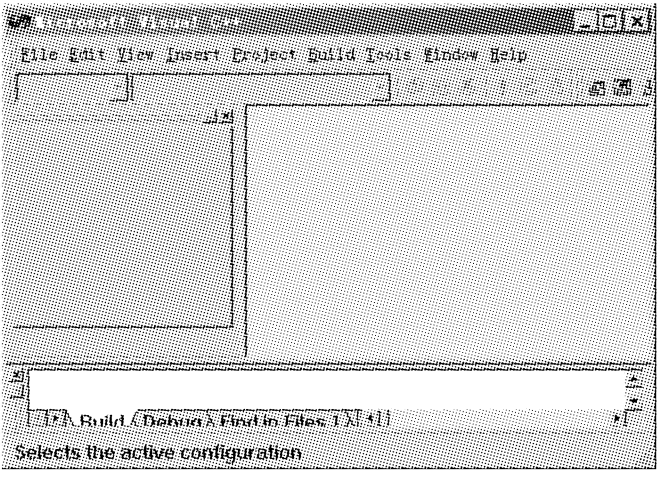

图 1.1 启动集成开发环境 Developer Studio

单击任务栏中“开始”后选择“程序”，找到 Microsoft Visual Studio 6.0 文件夹后，单击其中的 Microsoft Visual C ++ 6.0 图标，则可以启动 Developer Studio。

Developer Studio 用户界面是一个由窗口、工具条、菜单、工具及其他部分组成的一

个集成界面。通过这个界面，用户可以在同一环境下创建、测试、调试应用程序。

第二步：创建一个新的程序(项目)

开发的程序在 IDE 中称为项目。一个新的程序的创建过程为：

(1) 从主菜单中选择 File | New，将显示出 New 对话框；

(2) 选择 Projects 标签，并从列表中单击 Win32 Console Application；

(3) 在“Location”编辑框中输入想要存放程序文件的目录如 c:\testVC；

(4) 在对话框的右上角的“project name”编辑框内键入项目的名字，如“Hello”，然后单击 OK 继续；

(5) 系统将显示一个询问项目类型的程序向导，选择“an empty project”，然后单击 Finish；

(6) 在下一个对话框中选 OK 结束配置，此时系统创建新程序所用到的各种文件。

第三步：在刚刚创建的项目中增加一个 C ++ 源程序文件

用下面的方法在所创建的项目中添加一个文件：

(1) 在主菜单上选择 File | New；

(2) 在 New 对话框中选择 File 标签，单击“C ++ Source File”；

(3) 选中 Add to Project 复选框；

(4) 在右边的 File name 编辑框中为文件指定一个名字，如 Hello，系统将自动加上后缀. cpp。此时，新的空白文件将自动打开。

第四步：输入程序

(1) 在上面打开的文件中输入以下内容：

```
#include <iostream.h>
int main() {
  cout << "hello!" << endl ;
  return 0;
}
```

(2) 输入结束后，单击工具栏中的“save”图标，或者选择 File | Save 来保存文件。

第五步：编译、连接得到可执行程序

选择主菜单的 Build | Compile Hello. cpp 来编译这个文件。如果输入的内容没有错误，那么在屏幕下方的输出窗口将会显示：

```
hello.obj -0 error(s), 0 warning(s)
```

如果在编译时得到错误或警告，可能是源文件出现错误，再次检查源文件是否有错误，有则改正它。

因为输入的程序很简单，只有一个文件，所以这一个文件编译通过以后，就可以进行连接来得到可执行程序了。选择主菜单的 Build|Build Hello. exe 来进行连接。如果连接正确，则在屏幕下方的输出窗口将会显示：

```
hello.exe -0 error(s), 0 warning(s)
```

这样就得到了一个可执行程序 hello.exe 了。

第六步：运行程序

选择 Build|Execute hello.exe(或者 Ctrl+F5)，在开发环境中执行步骤五所得到的可执行程序。程序运行以后将显示一个类似于 DOS 的窗口，在窗口中显示一行“hello”，紧接着在下面显示“Press any key to continue”，这句话是系统提示按任意键退出当前运行的程序，回到开发环境中。按任意键，窗口关闭，退回到 Visual C++ 开发环境。

1.5 程序风格

1.5.1 效率与风格

下面的程序代码能读懂吗？

```
while ( '\n'!=(*p++=*q++));
```

再来看另一段程序：

```
// 将由 source_ptr 所指的字符串拷贝到 target_ptr 所指的字符串位置
// 源字符串以换行符'\n'结束
while(1) {
   *target_ptr= *source_ptr;
  if (*source_ptr=='\n')
    break;
  ++source_ptr;      // 源指针后移
  ++target_ptr;      // 目标指针后移
}
```

其实，上面两个程序的功能是完全一样的。前一个程序只占一行代码，而后一种写法却用了九行，似乎前者效率更高，但是对于一个维护人员更愿意看第二个程序。

程序并不只是写给计算机的，更多的时候，我们需要阅读别人的程序或者别人需要阅读我们的程序。理由有三：①应用系统越来越复杂，很多系统都是建立在其他人员已有努力的基础上，阅读别人的程序我们才能更好地利用它；②复杂系统的开发不能只靠一个人进行，需要许多程序员相互协作，我们需要经常阅读别人的程序来了解详细的实现，别人同样也需要看我们的程序；③程序并不是一成不变的，随着环境的变化或需求的变化，程序经常需要改变，而维护人员需要阅读以前的程序以便进行修改。事实上，程序员并没有花费大量时间去编写程序，而在维护、升级和调试已有代码上所花的时间远比编写新代码的时间多得多。我们写的程序更多的时候是由人来阅读，而不是计算机来运行。因此，在编写程序时保持良好的风格，不仅有利于自己对程序的调试，而且会大大增加程序的可复用机会；表面上看可能因此花费更多时间，但是这些将在未来带来更高的效率。

1.5.2 注释——整理编程思路、增加程序的可读性

尽管高级语言已经不再像机器语言、汇编语言那样低级，已经尽量贴近人类的自然语

言，但它毕竟是为计算机而设计的。人们在阅读程序时，依然会感到难以理解，为解决这一问题，高级语言中允许在程序中加入注释。注释是为增加程序的可读性而在程序中附加的说明性文字。计算机并不懂注释的内容，它“看”程序的时候，会把注释略过，注释也丝毫不会影响程序的运行。因此，注释是为人而写的。

在编程的时候，通过写注释，可以帮助我们理清思路，而且注释又能帮助别人理解我们的思路。编写比较复杂的算法程序的时候，一般先将算法的步骤用注释的形式写下来，然后在每一句注释下面用具体的代码实现。例如编写一个模拟人们到银行存取款时排队的程序，我们先写下如下算法思路：

(1) 银行开门，所有队伍为空；

(2) 处理新顾客到达事件，并为每个队的第一位顾客服务，直到银行到达下班时刻(这是一个循环)；

① 如果有新顾客到达，则

- 选择一个长度最短的队伍
- 将该顾客加入到队伍的最后

② 如果没有新顾客到达，则处理每个队伍的第一个顾客

(3) 不让新的顾客进来；

(4) 依次处理还在银行里的顾客，直到所有的顾客都被处理完；

(5) 银行关门。

然后，我们将这一描述变成程序代码。因为有很详细的注释，如果别人来看我们的程序也非常容易看懂。

1.5.3 注释的形式

C++ 的注释有两种形式：单行注释和多行注释。单行注释，它以符号‘//’引导，程序行中从符号‘//’到行尾的所有内容都是注释。如下面就是单行注释：

```
int average; // 学生的平均成绩
```

其中，‘int average;’是真正的程序代码，表示定义了一个整型的变量 average，而后面跟的内容从‘//’开始一直到行尾都是注释，这一注释解释了 average 这一变量表示的含义。

多行注释，用符号串‘/*’表示注释开始，用符号串‘*/’表示注释结束。注释可以跨越多行。下面是一个多行注释的例子：

```
/*************************************
 ********本程序在屏幕上输出一句问候语：hello! *******
 *************************************/
#include <iostream.h>
void main()
{
```

```
    cout << "hello!" << endl;
}
```

单行注释一般用于简短的说明，如说明它前面一句程序或下面一句（或几句）程序的含义。而多行注释一般出现在函数或文件的开头，用于说明该函数或文件的相关信息，如功能、算法、作者、编制日期等。我们可以按自己喜欢的方式使用单行注释和多行注释。

写注释也是有规律可循的，下面介绍一些非常实用的注释方法。

1. 程序头

在每个程序的开头写上注释，提供该程序的有关信息，一般包含以下内容：

（1）程序的名字；

（2）程序功能的简短描述；

（3）作者的名字（以便今后联系、交流）；

（4）编写程序的目的、用途；

（5）程序的用法（程序是在什么环境下运行，要不要给运行参数，要不要进行输入等。如果程序的用法不能简单地用几句话说出来，就需要有更详细的用户手册或联机帮助来详细说明用法）；

（6）程序中用到的别人的成果（如果合法拷贝了别人的程序代码，应该将那位作者的名字也列出来）；

（7）文件格式（程序所要读或写的文件的格式，如名称的要求、内容的格式要求、位置要求等）；

（8）程序的限制（如程序所处理的最大数目、能接受的输入内容、不能适用的情形、程序不能检查的错误输入等）；

（9）程序的修改史（该程序曾被哪些人修改过，修改过的内容、修改的目的、修改的日期等，每一个参与修改程序的程序员都应该留下信息）；

（10）程序中处理错误的方法（程序中如何处理如文件找不到、内存不够、除数为零等错误）。

当然，并不一定要为每个文件加上所有这些注释，应该根据实际情况加以取舍。

2. 在函数的前面加上注释，提供有关函数的信息

函数的注释信息一般包含以下内容：

（1）函数的功能（说明该函数的作用）；

（2）函数的参数（函数的各个参数的含义、参数允许的取值范围等）；

（3）返回值（函数返回值的含义，如 0 表示什么，1 表示什么）；

（4）函数的副作用（函数运行产生的影响，如改变了系统的状态、创建了一个文件、或者申请了一些内存空间等）。

下面是一个为函数写注释的例子：

```
/*************************************************
 * 函数功能:计算正整数 n 的阶乘 n!。n!=n*(n-1)*(n-2)*…*1
 * 函数参数:n,表示所要计算阶乘的正整数,要求 n 满足:
 *      0 < n < 100
 * 函数的返回值:如果输入参数不正确,返回-1,
 *          否则返回计算 n! 得到的结果
 *************************************************/
```

3. 说明所定义的变量的含义

为变量所写的注释说明该变量所表示的含义。变量的命名非常重要,一个好的变量名字本身就能说明很多,但是名字的表达能力毕竟是有限的,不能将所有信息都通过变量名字来传达出来,因此还需要借助于注释来进一步解释变量。如下面的变量定义:

```
int number;
```

看到 number 这个单词,人们会猜想它可能是用来表示某个数字的。然而是表示人的个数,还是表示书的数目,或是表示其他什么,从名字上就很难知道了。这时就需要注释来进一步解释:

```
int number; // 已经到学校报到的本科学生人数
```

这样就非常清楚了。想一想,如果要用变量名来传达这么多信息,该是多么长的一个变量名:

```
number_of_registered_undergradate_students
```

建议在编制程序时,要为变量加上合适的注释。

4. 在函数定义体中,用注释解释算法思路

函数的实现包括很多语句,这些语句所包含的逻辑不会总是一目了然,在合适的地方加上合适的注释,会使程序清晰易懂。下面是一个这样的例子:

```
/*************************************************
 * 函数功能:计算正整数 n 的阶乘 n!。
 * n!=n*(n-1)*(n-2)*…*1
 * 函数参数:n,表示所要计算阶乘的正整数,要求 n 满足:
 *      0 < n < 100
 * 函数的返回值:如果输入参数不正确,返回-1,
 *          否则返回计算 n! 得到的结果
 *************************************************/
double factorial(int n)
{
  double result; // 临时存放计算阶乘的结果

  /****检查参数的合法性 ****/
```