第 1 章　　绪 言

随着计算机技术的不断发展以及软件设计规模的不断扩大，计算机软件的开发面临着两大难题：一是如何越过程序设计的复杂性障碍问题；另一个是如何利用软件来自然地表示客观世界，也就是对象模型问题。面向对象的程序设计技术很好地解决了上述问题，而 C＋＋语言正是面向对象程序设计技术的具体实现。

C＋＋语言是在 C 语言的基础上发展起来的，它既融合了面向对象程序设计技术，又保留了 C 语言的特征。C＋＋语言在提供了面向对象的设计能力的同时，又保持了与 C 语言的高度兼容性，使 C 程序设计人员能够比较容易地转向 C＋＋语言。

1.1　面向对象程序设计的特点

世界上的任何事物都可以被看做为对象，对象是对现实世界的抽象。一本书可以是一个对象，而一个图书馆也可以是一个对象。从软件设计的角度来讲，一个对象就是一个高度抽象的模块，该模块中既包含了相应的数据结构，又提供了对数据结构进行操作的方法。正是由于这种高度抽象的结果，使得面向对象程序设计具有许多面向过程程序设计所无法比拟的特点。

归纳起来，面向对象程序设计有如下一些主要特点。

1. 模块化

采用面向对象技术设计出来的程序都是由一个一个的对象组成的，在每一个对象中，既定义了相应的数据结构，同时又定义了操作这些数据结构的方法，这样，每一个对象都是一个完整的功能模块。因此，由面向对象程序设计语言所设计出来的程序，其模块化程度高，易于扩充和维护。

2. 数据隐藏

在面向过程的程序设计中，一般注重的是程序的代码，而把数据放在次要位置上。实际上，只有将数据和操作这些数据的方法结合起来，才能完整地描述一个程序。数据隐藏的目的是将对象的设计和对象的使用分开，使用者不必了解对象实现的细节，只要利用设计者所提供的接口来访问该对象即可。同时，由于为对象中的成员加上了访问权限控制信息，使得外部对对象中成员的访问是有限制的：对于那些私有成员来讲，外部就无法进行访问，从而起到了数据隐藏的作用。

面向对象程序设计通过定义类把数据和方法集成在一个对象中，外部程序只能通过类所规定的接口和类进行通信。

3. 继承

在面向对象程序设计中，对象是程序的基本单位，复杂的对象可由简单的对象来组成，而继承操作正是用于完成这一工作的。它允许从已经存在的类中继承相应的成员（数据和方

法），只要告诉编译器你的新类是由另一个类继承而来的，它就会把另一个类中除私有成员之外的所有成员都赋给你的新类，就如同你重新定义的一样。显然，继承操作能够减轻程序设计的工作量，提高程序的可靠性。

4. 多态性

所谓多态，是指一个名字可具有多种不同的语义，或者说多个函数具有相同的名字但具有不同的作用。

在继承操作中，如果父类和子类中的某个函数具有相同的名字，且被定义为虚函数，则利用指向父类的指针，根据所赋给的对象不同（父类对象或子类对象），就可以调用不同的函数。

面向对象程序设计中的多态性，给程序设计带来了很大的灵活性，使我们既能够重用已有的类，又能够满足新类的需要。

5. 重载

在面向过程的程序设计语言中，每个函数必须有一个惟一的名字，也就是说函数不能重名。而在像 C++这样的面向对象程序设计语言中，多个函数可以共用一个名字，只要它们的参数个数不同或参数类型不同即可，这就是函数的重载。在 C++语言中，不但函数可以重载，运算符也可以重载，可以根据需要为已有的运算符赋予不同的意义。显然，增加了重载功能以后，将提高程序的可理解性，使程序功能实现起来更自然、更流畅。

1.2 C++语言程序的开发过程

开发一个 C++语言程序的基本过程如图 1.1 所示。

1. 编辑

选择适当的编辑程序，将 C++语言源程序通过键盘输入到计算机中，并以文件的形式存入到磁盘中。经过编辑后得到的源程序文件是以 .CPP 为其文件扩展名的。

2. 编译

通过编辑程序将源程序输入到计算机后，需要经过 C++语言编译器将其编译成目标程序。在对源程序的编译过程中，可能会发现程序中的一些词法或语法错误，这时就需要重新利用编辑程序来修改源程序，然后再重新编译。经过编译后得到的目标文件是以 .OBJ 为其文件扩展名的。

3. 连接

经过编译后生成的目标文件是不能直接执行的，它需要经过连接之后才能生成可执行的代码。连接后所得到的可执行文件是以 .EXE 为其文件扩展名。

4. 执行

经过编译、连接之后，源程序文件就可以生成可执行的文件，这时就可以执行了。在 DOS 系统下，只要键入可执行的文件名，并按“回车”键后，就可执行文件了。在 Windows 系统下，

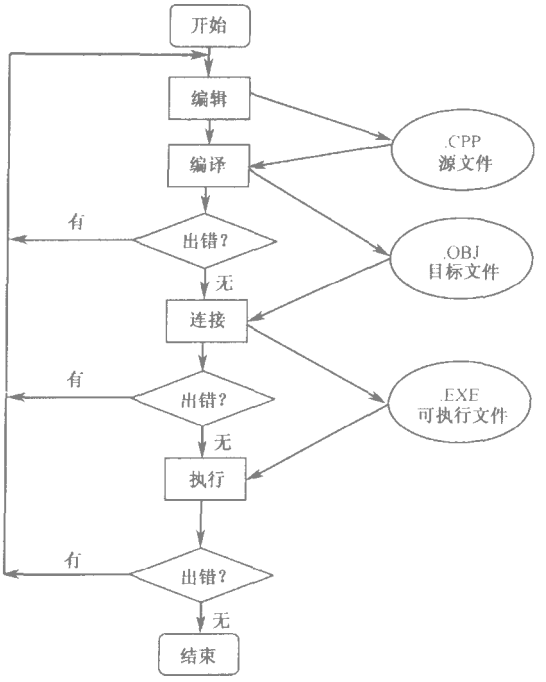

图 1.1 C++ 语言程序的开发过程

只要双击可执行文件名即可执行。

现在有许多厂家都推出一种集成环境来处理 C++语言程序，如 Turbo C++、Visual C++等，在这种集成环境下，对程序的编辑、编译和连接等操作，都可以在一个窗口下进行，使用起来非常方便。

1.3 C++语言程序的结构

从程序结构的角度来讲，C++语言同 C 语言基本上是相同的。在这一节中，我们通过编写几个简单的 C++语言程序，来阐述 C++语言的程序结构，同时，也对 C++语言的基本语法成分进行相应的说明，以便使读者对 C++语言程序有一个概括的了解，为以后的学习打下基础。

【例 1.1】编写一个 C++程序，用于显示字符串"Hello, World !"。

```
#include <iostream.h>
void main( )
{
    cout << "Hello, World! \n";
}
```

这是一个简单而完整的 C++语言程序，经过编辑、编译和连接后，其执行结果是在屏幕的当前光标位置上显示如下字符串：

Hello, World!

说明：

(1)一个 C++程序可以由多个函数组成，但任何一个完整的 C++程序，都必须包含一

个且只能包含一个名为 main( )的函数,程序总是从 main( ) 函数开始执行的。

(2)由左、右花括号括起来的部分是函数体，函数体中的语句将实现程序的预定功能。在本例中,main( ) 函数的函数体中只有一个语句，其功能是将字符串‘ Hello, World !’显示到屏幕上。

(3)cout 是标准输出流，它意味着进行标准输出，运算符＜＜表示将数据送入 cout 的意思，为了使用 cout 进行标准输出，必须在程序的开始处包含进 iostream.h 头文件。

【例 1.2】 从键盘输入两个数，并将这两个数的差显示出来。

```
#include <iostream.h>
int SUBab(int a, int b)      // 子函数
{
    int c;
    c=a-b;                   //求 a 和 b 的差
    return c;
}
void main( )                 /*主函数*/
{
    int x, y;
    cin >> x;                /* 输入 x*/
    cin >> y;                /* 输入 y*/
    int z;
    z=SUBab(x, y);
    cout << z;
}
```

说明：

(1)在 C++语言程序中，既可以使用 C语言中使用的/*和*/来进行注释，也可以使用//符号来进行注释。/*和*/表示由其括起来的部分是注释部分，而//表示从//符号开始到此行末的所有内容都是注释。

从下面的两行语句中，可以看出由/*和*/以及//所表示的注释之间的差异：

```
a=1; /* a=a+10; */     a=a+100;     //  结果为 a=101
a=1; // a=a+10;       a=a+100;       //结果为 a=1
```

(2)C++语言程序中的所有变量都必须定义为某种数据类型，同时必须遵循“先定义、后使用”的原则，但并不要求所有变量一定在函数体的开始处（即可执行语句的前面）进行定义，只要在变量的使用之前进行定义即可。例如，此程序中的变量 z 的定义位置就是位于此函数体的可执行语句的中间部分。

(3)一个 C++程序可以由多个函数组成，通过函数之间的调用来实现相应的功能。程序中所使用的函数，既可以是系统提供的库函数，也可以是用户根据需要自己定义的函数。例如，此程序中的 SUBab() 就是用户自己定义的用于求两个数的差的函数。

(4)cin 是标准输入流，它意味着进行标准输入，运算符＞＞表示从 cin 读数据的意思，同 cout 一样,为了使用 cin 进行标准输入，必须在程序的开始处包含进 iostream.h 头文件。

从表面上来看,C++语言只是在 C语言的基础上增加了对类的处理功能，而实际上，除了类之外,C++语言还在很多细小的方面进行了扩充和完善，同时，即便是类本身也包含了很多复杂的内容。因此，要想学好 C++语言，最根本的方法就是多练习、多上机。由于C++

语言是在C语言的基础发展起来的，因此，几乎C语言中的所有功能在C++语言中都可以直接使用,这为C++语言的初学者提供了方便。

习题

1.1 根据所学的知识，请总结一下什么是面向对象程序设计以及面向对象程序设计的特点。

1.2 编一程序，利用标准输出流对象cout,将“C++ Programming Language”字符串显示到屏幕上。

1.3 编一程序，利用标准输入流对象cin和标准输出流对象cout，从键盘输入一字符串，并将其显示到屏幕上。

1.4 试分析下列程序，并给出其运行结果。

```
#include <iostream.h>
int Mulab(int a, int b)
{
    int Mvaluc;
    Mvalue=a * b;
    return Mvalue;
}
void main( )
{
    int x, y;
    cin >> x;
    cin >> y;
    int result;
    result=Mulab(x, y);
    cout << result;
}
```

1.5 请指出下列程序的错误。

```
# include <stdio.h>
void main( )
{
    int x, y, z;          //变量定义
    cin >> x
    cin >> y
    z=x+y;
    printf("Result=%d", z);
    }
```

第 2 章　数据类型和运算符

C++语言程序中所用到的每一个常量、变量以及函数等都是程序的基本操作对象，它们都隐式地或显式地与一种数据类型相联系，每种数据类型都表示了它的可能取值范围以及能在其上所进行的运算。C++ 语言中提供了丰富的数据类型和运算符，运用这些数据类型和运算符能够进行复杂的数学运算。

本章主要讨论 C++语言中的一些基本概念，如基本数据类型、指针和引用、运算符以及利用这些运算符来构成相应表达式的一些规则等。

2.1　基本概念

2.1.1　标识符

在计算机语言中，标识符的概念经常被用到。所谓标识符，是指用来标识程序中所用到的变量名、函数名、类型名、数组名、文件名以及符号常量名等的有效字符序列。

在 C++语言中，标识符的命名规则是：由字母(大、小写皆可)、数字及下划线组成，且第一个字符必须是字母或下划线。

由上述标识符的命名规则可知，下面的标识符名是合法的：

year, Day, ATOK.x1, _CWS, _change_to

而下面的标识符名是不合法的：

#123, .COM, $100, 2002Y, 1_2_3

在 C++语言中，大写字母和小写字母是有区别的，即作为不同的字母来看待。如标识符 RAN、Ran 和 ran 分别表示 3 个不同的标识符，这一点同有的程序设计语言是有区别的，应引起注意。

2.1.2　常量

常量又称常数，是指在程序运行过程中其值不能被改变的量，如：100, 3.14 等。常量也分为不同的类型，这是由常量本身隐含决定的（将在下面详细介绍）。为了增加程序的可读性，可以用一个名字（字符序列）来代表一个常量，此时的常量被称为符号常量。有关符号常量的使用，将在“编译预处理”一章中详细介绍。

2.1.3　变量

变量是指在程序运行过程中其值可以被改变的量。变量被区分为不同的类型，不同类型的变量在内存中占用不同的存储单元，以便用来存放相应变量的值。

组成变量名(标识符)的有效字符数随 C++语言的编译系统而定。有的编译系统允许使用长达 31 个字符的变量名，而有的编译系统只取变量名的前 8 个字符作为有效字符，后面的字符无效，不被识别，这样，只要变量名的前 8 个字符相同，就被认为是同一个变量。因此，在

进行程序设计之前，应首先了解所使用的编译系统中对变量名长度的规定，以免造成变量使用上的混乱。

2.1.4 关键字

关键字，又被称为保留字或保留关键字，也是 C++语言中的一种标识符，它用来命名 C++语言程序中的语句、数据类型和变量属性等。每个关键字都有固定的含义，不能另作其他用途，C++语言中的所有关键字都是用小写字母来表示的。

2.2 基本数据类型

在 C++语言中，共有 5 种基本数据类型，它们分别由如下关键字来进行定义：

bool　布尔型
int　整型
char　字符型
float　单精度浮点型
- double　双精度浮点型
- void　void 型

C++语言规定，对程序中用到的所有变量，都必须先定义后使用，每个变量只能与一种数据类型相联系。在定义变量时，不能把 C++语言中具有固定含义的关键字（如 int、char 等）作为变量名，同时，同一个函数内所定义的变量不能同名。

2.2.1 整型变量及其常量

整型变量可用来存放整型数据（即不带小数点的数），其定义方式如下：

```
int i1, i2;
```

其中，i1 和 i2 即被定义为整型变量。

在 C++语言中，整型常量可以用 3 种数制来表示。

(1) 十进制整型常量：例如 319，－200 等，其每个数字位可以是 0～9。

(2) 十六进制整型常量：如果整型常量是以 0x 或 0X 开头，那么，这就是用十六进制形式表示的整型常量。例如，十进制数的 129，用十六进制形式表示为 0x81 或 0X81，其每位数字可以是 0～9，A～F。

(3) 八进制整型常量：如果整型常量的最高位为 0，那么它就是以八进制形式表示的整型常量。例如，十进制数的 128，用八进制表示为 0200。需注意的是，八进制表示中的每个数字位必须是 0～7。

2.2.2 浮点型变量及其常量

在 C++语言中，把带有小数点的数称为浮点数。浮点型变量又被称为实型变量，按其能够表示的数的精度，又被分为单精度浮点型变量和双精度浮点型变量。

单精度浮点型变量的定义方式如下：

```
float f1, f2;
```

其中，f1 和 f2 即被定义为单精度浮点型变量。

双精度浮点型变量的定义方式如下：

```
double  d1, d2;
```

其中，d1 和 d2 即被定义为双精度浮点型变量。

单精度浮点型变量和双精度浮点型变量之间的差异仅仅体现在所能表示的数的精度上。

浮点型常量一般不分 float 和 double 型，任何一个浮点型常量既可以赋给 float 型变量也可以赋给 double 型变量，但由于 float 型变量和 double 型变量所能表示的数的精度不同，所以，在赋值时，将根据变量的类型来截取相应的有效位数。

浮点型常量有如下 2 种表示形式。

(1) 十进制数形式：它是由数字和小数点来表示的。例如：3.14159，－7.2，9.8 等。

(2) 指数法形式：指数法又称为科学计数法，它是由尾数、指数及字母 e(或 E) 来表示的。例如：十进制数的 180000.0，用指数法可表示为 1.8e5。

2.2.3 字符型变量及其常量

字符型变量用于存放一个单个字符，其定义方式如下：

```
char  c1, c2;
```

其中，c1 和 c2 即被定义为字符型变量。

字符型常量是由单引号括起来的一个字符。如：'A'、'*'、'8' 等。在 C++ 语言中，还允许使用一些特殊形式的字符型常量，这些字符型常量都是以反斜线字符'\'开头的字符序列。例如：

'\n'：换行字符。

'\r'：回车字符。

'\b'：退格字符。

'\t'：制表字符，又被称为横向跳格字符。

'\''：单引号字符。

'\"'：双引号字符。

除了上述具有特殊意义的字符外，C++语言还允许在字符'\'后面紧跟 1 ～3 位八进制数或在'\x'后面紧跟 1 ～3 位十六进制数来表示相应系统中所使用的字符的编码值。如：'\7'(或'\007'、'\07' 可以用来表示响铃字符。

需要注意的是，上面介绍的由'\'开头的特殊字符，仅代表一个单个字符，而不代表多个字符，它仅代表相应系统中的一个编码值。

2.2.4 void 型数据

void型是不具有值的特殊的数据类型，它明确指出没有值的这一特性。 void 型主要用在函数值类型说明以及指针类型说明等。不存在 void型一般变量。

例如，不能以下述方式来定义变量：

```
void  dt;
```

void 型一般用在如下三个方面：

(1) 用于定义指向任何数据类型的指针。例如：

```
void  *p;
```

(2) 如果函数没有返回值，则在函数定义时，可将返回值类型定义为 void 型。例如：

```
void func1 (int n)
{
  ……
}
```

（3）如果函数没有参数，则在函数定义时，其参数的位置可以放上 void 说明，以明确表明此函数无参数。例如：

```
int func2 (void)
{
  ……
}
```

2.2.5 bool 型变量及其常量

只能保存真、假值的变量为 bool 型变量，其定义形式如下：

```
bool b1, b2;
```

bool 型常量只有两个：true 和 false。例如：

```
bool b1, b2;
……
b1=true;
b2=false;
```

bool 型变量一般用于条件判断语句中。

2.3 long, short, signed, unsigned 关键字

2.3.1 long 和 short 关键字

在 C++语言中，可以在 int 的前面加上 long 和 short，以定义长整型和短整型变量，同时，还可以在 double 的前面加上 long，以定义长浮点型变量。例如：

```
long int Li;
short int Si;
long double Ld;
```

这里：

```
long int Li;
short int Si;
```

可以分别简写为：

```
long Li;
short Si;
```

int 型变量、short int 型变量以及 long int 型变量所能表示的数据大小的关系如下所示：

```
short int <= int <= long int
```

float 型变量、double 型变量以及 long double 型变量所能表示的数据大小的关系如下所示：

```
float  <=  double  <=  long double
```

2.3.2 signed 和 unsigned 关键字

可以在 char 和 int 等定义整型变量的关键字前面加上 signed 和 unsigned 关键字，以决定该整型变量是有符号整型变量还是无符号整型变量。这里 signed 用于定义有符号整型变量，unsigned 用于定义无符号整型变量。如果在定义整型变量的前面没有加上 signed 或 unsigned，则其默认为有符号整型变量。例如：

```
signed int si;          //si 是有符号整型变量
unsigned int ui;        //ui 是无符号整型变量
signed char sc;         //sc 是有符号字符型变量
unsigned char uc;       //uc 是无符号字符型变量
```

由于有符号变量的最高位用于表示符号（0 :正数，1 负数),所以无符号变量所能够表示的正数的范围将大于同一种类型的有符号变量所能表示的正数的范围。例如：

```
char c;              //c  能表示的数的范围为 -128 ~127
signed char sc;      //sc 能表示的数的范围为 -128 ~127
unsigned char uc;    //uc 能表示的数的范围为 0 ~255
```

这里：

```
signed int si;
unsigned int ui;
```

可以分别简写为：

```
signed si;
unsigned ui;
```

2.4 指针和引用

2.4.1 指针

指针在 C++语言中占有重要的地位，同时指针也是 C++ 语言的一大特点。C++ 语言的高度灵活性及其特别强的表达能力，在很大程度上来自于巧妙而恰当地使用指针。 C++语言的指针既可以指向各种类型的变量、对象，也可以指向函数。正确地使用指针，能够有效地表示和处理复杂的数据结构，特别有利于对动态数据的管理。

用来存放某种对象首地址（指针值）的变量被称为指针变量。例如：

```
int a;
int *p;              //定义指针变量 p
p=&a;                //将 a 变量的地址放入 p 中
*p=100;              //将 100 存入由 p所指向的变量（即 a 变量)中
```

其中，* 为间接访问运算符，& 为取地址运算符。

指针的用法非常灵活，它可以指向各种类型的变量或对象，通过指针可以达到间接访问的目的。需要注意的是，在使用指针变量之前必须要为其初始化。

2.4.2 void 型指针

前面我们已经提到过指针的类型可以是 void 型，在 C 语言中，void 型指针同其他类型的指针可以互相赋值，但是在 C++ 语言中，要想将 void 型指针赋给其他类型的指针变量，必须

进行强制类型转换。例如：

```
char   * pc;
void   * pv;
    ...
pv=pc;                    //C 语言和 C++ 语言皆可
pc=pv;                    //C 语言可以，但 C++ 语言不行
pc=(char *)pv;            //C   语言和 C++ 语言皆可
```

又例如，malloc( ) 函数是用于内存申请的，此函数的返回值是指向申请到的内存的 void 型指针。此时：

```
char * pc

pc=(char * ) malloc (100);//C 语言和 C++ 语言皆可
pc=malloc(100);           //C   语言可以，但 C++ 语言不行
```

2.4.3 引用

引用是一个变量的别名，它自动地适用于间接访问运算符 *，但引用变量中的值(地址值)是不能被改变的。

引用的定义方式如下：

```
int   x;
int   &y=x;
```

此时，y 即是引用变量，它是 x 变量的别名，也就是说，y 和 x 都代表着同一个变量。

【例 2.1】 编一程序，用于说明引用变量的定义和使用。

```
#include <stdio.h>
int main( )
{
    int x;
    int &y=x;      //y 是引用变量，它是 x 的别名
    y=12;
    printf("x=%d\n", x);
    printf("y=%d\n", y);
    return(0);
}
```

程序的输出结果如下：

```
x=12
y=12
```

说明：

(1)注意引用变量和指针变量在定义上的差异。

(2)由于引用变量在初始化时被设定了某一变量的地址之后就不能再改变了，因此在程序中定义引用变量时必须要初始化。

上述程序如果利用指针来编写，则程序如下：

```
#include <stdio.h>
int main( void)
```

```
    int  x;
    int *p=&x ;          //p指向 x,或说 *p是 x 的别名
    *p=12;               //将 12 存入由 p所指向的对象 x 中
    printf ("x=%d\n",x) ;
    printf ("*p=%d\n", *p);
    return(0) ;
}
```

程序的输出结果如下：

```
x=12
*p=12
```

由上述程序可知，使用引用编写的程序比利用指针编写的程序看起来更清晰一些。从表面上看指针和引用有许多相似的地方，但指针和引用还是有重要差别的，其主要差别如下：

(1)指针变量的值是可以改变的，但引用变量在初始化时被设定了某一变量的地址之后，就不能再改变了。

(2)指针既可以对值也可以对地址进行操作，但引用只能对值进行操作。

例如：

```
int a ;
int b;
int &r=a;
int *p=&b;
r=100;               //正确
r=&a;                //错误，不能对地址进行操作
*p=200;              // 正确
p=&a;                //正确，指针可以对地址进行操作
```

引用主要用在如下两个方面：

(1）引用可以用做函数参数。

(2)函数的返回值可以是引用。

我们将在第 5 章中对引用的使用进行详细介绍。

2.5 数组

数组是一些具有相同类型的数据的集合，它是由某种类型的数据按照一定的顺序组成的。同一个数组中的每个元素都具有相同的变量名，但具有不同的序号（下标），只有一个下标的数组被称为一维数组，有两个下标的数组被称为二维数组，依次类推，C++语言允许使用任意维数的数组。当处理大量的、同类型的数据时，利用数组是很方便的。数组同其他类型的变量一样，也必须先定义后使用。

2.5.1 数组的定义和使用

数组的定义方式如下：

```
int x[10] ;
```

它表示 x 是具有 10 个元素的一维整型数组变量， x 中的每个元素可用于存放一个整型数据，

每个元素分别用 x[0], x[1],…, x[9]来表示。

例如：

```
int y[3][4];
```

它表示 y 是具有 12 个元素的二维整型数组变量。 y 中的每个元素分别用 y[0][0], y[0][1],…, y[2][3] 来表示。

在定义数组的同时，可以为其初始化。例如：

```
int dt[5]={10,20,30,40,50};
```

它表示 dt[0]=10,dt[1]=20,dt[2]=30,dt[3]=40,dt[4]=50。

数组元素是通过下标来使用的。例如：

```
int  x;
int  a[ ]={1,2,3,4,5};
     …
x=a[3];
     …
```

2.5.2 字符串

由双引号括起来的字符序列称为字符串。例如：

"C++ Programming Language" 就是一个字符串。

对字符数组的初始化可以利用字符串来进行。例如：

```
char st[80]="This is a string.";
```

由于字符串都是以'\0'结尾的，因此，上述字符串的实际长度为 18。也可以利用下述形式来对字符数组进行初始化：

```
char st[80]={'T','h','i','s',' ','i','s',' ','a',' ','s','t','r','i','n','g','.','\0'};
```

2.6 枚举

在程序设计过程中，如果一个变量仅在很小的范围内取值，则可以把它定义为枚举类型，使用枚举类型的变量能够提高程序的可读性。

定义枚举变量有以下几种方式：

(1)先定义枚举类型，然后定义枚举变量。例如：

```
enum color {red, yellow, blue, green, white, black}    ;
```

这里：

enum 为定义枚举类型变量的关键字， color 为枚举类型名，red、yellow 和 black 等为枚举元素或枚举常量。

定义了枚举类型 color 以后，就可以利用 color 类型来定义相应的枚举变量。例如：

```
color c;
```

(2)定义枚举类型的同时，定义枚举变量。例如：

```
enum color { red, yellow } c, b;
```

这里的 c、b 就是 color 类型的枚举变量。

(3)直接定义枚举变量。例如：

```
enum { red, yellow } c;
```

枚举变量的取值范围仅限于枚举元素表中的值（即枚举常量）。

使用枚举类型数据时，应注意如下几个问题：

(1)枚举元素都是常量，即枚举常量，而不是变量，因此，不能为枚举元素赋值。例如：

```
red=4;
```

是错误的。

(2)每个枚举元素都有一个确定的整数值，如果在枚举类型定义时没有显式地给出枚举元素的值，则这些元素的值按顺序依次为 0,1,2,…。

我们也可以在枚举类型定义时显式地给出枚举元素的值。例如：

```
enum color{ red=6,yellow=1.blue,green,white};
```

它定义了 red 的值为 6,yellow 的值为 1，以后顺序加 1,即 blue 的值为 2,green 的值为 3.white 的值为 4。

(3)可以将一个整数经强制类型转换后赋给枚举变量。例如：

```
enum color{red,yellow,blue,green,white,black} c;
c=(color) 2;
```

它相当于下面的赋值语句：

```
c=blue;
```

(4)枚举常量可以直接赋给整型变量。例如：

```
int  a;
…
a=red;//即 a=0
```

使用枚举类型数据的示意性例子如下：

```
enum color {red, blue, black, white};
void main( )
{
    int a;
    color mycolor;
    mycolor=white;
    …
    if (mycolor != white)
        a=100;
    else
        a=200;
    …
}
```

2.7 内存的申请与释放

在 C 语言中，一般是通过使用 malloc( )和 free( ) 两个函数来进行内存申请和释放的。尽管 C++ 语言中也可以使用这两个函数，但使用 C++语言中提供的 new 和 delete 两个运算符来进行内存申请和释放将更为方便和有效，因为在使用 new 和 delete 时，不需要任何头文件。

new 和 delete 的用法如下：

(1)单个变量的内存申请与释放。

```
int *dt
dt=new int;        // 申请一个整型内存空间，并由 dt 指向。
……
delete dt;         // 释放由 dt 指向的内存空间
```

上面申请的内存空间中的初值是不确定的，为了在申请内存空间的同时为其赋初值，可采用如下方式：

```
dt=new int(5);   // 表示 dt 所指向的内存空间中的初值为 5
……
delete dt;       //释放由 dt 指向的内存空间
```

(2)数组的内存申请与释放。

```
int *dt;
dt=new int[20];  //dt 指向具有 20 个整型元素的内存空间
……
delete [ ]dt;    //释放由 dt 指向的内存空间
```

(3)在定义变量的同时申请内存。

```
char *ss=new char[80];     //ss 指向具有 80 个字符元素的内存空间
int *dt=new int[20];       //dt 指向具有 20 个整型元素的内存空间
```

在进行二维数组的内存空间申请与释放时，需要注意其写法。例如：

```
char *ss;
ss=new char[10][80];
```

将出现错误。需改为如下形式：

```
char (*ss)[80];          //这里的括号必须有，它表示 ss 指向一个 char[80]类型的对象
ss=new char[10][80];
……
delete [ ]ss;
```

对于多维数组的内存申请与释放的写法如下：

```
int a[n][n1]…[n8];
```

应为：

```
int (*a)[n1]…[n8];
a=new int[n][n1]…[n8];
……
delete [ ]a;
```

2.8 const 关键字

const 关键字主要用来定义其数值不能改变的变量。例如：

```
const  double  pai=3.14159;
```

这样，变量 pai 在以后就不能赋以新的值了。例如：

```
pai=0.0;    // 出错
```

需要注意的是，当 const 用于指针变量定义时，const 所处的位置不同，其所代表的意义也不同。例如：

```
int dt1=10 , dt2=20 , dt3=30;
int * const p2=&dt2;        //p2 的地址是 const 的指针 p2
const int * p3=&dt3;        // * p3 的值是 const 的指针 p3
* p2=1000;                  //正确
p2=&dt1;                    //出错,p2 的指针值不能改变
p3=&dt1;                    //正确
* p3=1000;                  //出错,由 p3 所指向的值不能改变
```

2.9 volatile 关键字

volatile 关键字主要用于向系统传递一种信息，这就是告诉系统不要进行最优化处理。例如：

```
int trip;

void waittrip(void)
 {
    trip=0;
    while(trip==0)

}
```

从语法上来讲，由于 trip 的值固定为 0，因此经过编译优化处理后，while 语句可能就变成了如下形式：

```
while(1)
```

这就成为无限循环了。尽管 trip 的值在表面看来固定为 0，但是通过硬件设备的中断处理、I/O端口处理或内部时钟处理等，就可以在程序感觉不到的情况下，使 trip 的值发生变化。这样，若进行优化处理，就出现了问题。为了避免这一情况的发生，可以在定义相应的变量时使用 volatile 关键字。

下面的程序将通知系统“不要将 trip 变量进行优化处理”。

```
volatile int trip;
  ……
void waittrip(void)
{
    trip=0;
    while (trip==0)        //在 trip=0 时循环

}
```

在 trip 变量定义的前面加上 volatile 关键字以后，编译系统将不对 trip 变量进行优化处理，这样，就避免了由于优化处理而可能带来的问题。

2.10 typedef 关键字

C++ 语言提供了许多标准类型名，如 int、char 和 double 等，用户可以直接使用这些类型

名来定义所需要的变量。同时，为了增加程序的可读性以及处理上的方便，C++语言还允许使用 typedef 关键字来定义已有类型的别名。例如：

```
typedef int word;
typedef unsigned char byte;
```

它表示可以用 word 来定义 int 型变量，可以用 byte 来定义 unsigned char 型变量。

例如：

```
word a;          / 相当于 int a;
byte b;          //相当于 unsigned char b;
```

使用 typedef 关键字需要注意如下几个问题：

(1)typedef 关键字不能创造新的类型，只能为已有的类型增加一个类型名。

(2)typedef 关键字只能用来定义类型名，而不能用来定义变量。

2.11 变量的存储类

在 C++ 语言中，变量的定义包含三个方面的内容：一是变量的数据类型，如 int、char 和 float 等；二是变量的作用域，所谓变量的作用域，是指一个变量能够起作用的程序范围，也就是说，一个变量定义好之后，在何处能够使用该变量，在 C++语言中，变量的作用域是由变量的定义位置决定的，不同位置的变量，其作用域也是有差异的；三是变量的存储类（或称存储类别），即变量（数据）在内存中的存储方法，不同的存储方法，将影响变量值的存在时间（即生存期）。

在 C++语言中，变量的存储类共有如下 4 种：

- auto 存储类
- static 存储类
- extern 存储类
- register 存储类

2.11.1 auto 存储类

auto存储类，即自动存储类。在函数内部定义的变量，如果不指定其存储类，那么它就是 auto 类变量。例如：

```
void func( )
{
    int a;
    auto int b;
    ……
}
```

a 和 b 都是 auto 存储类变量。

自动类变量是在动态存储区中分配存储单元的，当函数返回时，自动类变量中存放的数据也就消失了。

在变量初始化方面，自动类变量在每调用一次函数时都要赋一次初值，且自动类变量的默认初值不确定。

【例 2.2】 编写一程序，每调用一次函数，显示一自动类变量中的内容，然后为其值加 1。