第 2 章　数据描述与基本运算

知识目标

◎ 了解C++程序语句的构成
◎ 了解C++标识符与其主要分类
◎ 了解C++数据类型与其声明(定义)的方法
◎ 了解C++常见表达式类型与基本运算方式

技能目标

◎ 熟练掌握C++中的数据类型及其转换
◎ 掌握C++中的运算符优先级、表达式及其运算
◎ 会利用C++中的函数实现一定的功能

本章将介绍C++语言的基础知识，主要包括数据的描述方式和与之相关的基本运算。这些数据类型是C++中预定义的内置的(built-in)或者说是基本的(primitive)数据类型，是构成其他复杂数据结构的基础。在C++语言中，数据的基本描述形式及运算方式与 C 语言基本相同。

2.1　标识符与关键字

标识符与关键字是C++语言中非常重要且容易混淆的两个概念。在形式上，它们都是具有独立含义的符号串，是构成C++语言的基本逻辑单位，这是它们最大的共同点。而通常情况下，标识符特指用户声明的符号串，而关键字是由系统保留的，具有特殊用途的符号串，这是它们的主要区别所在。

2.1.1　标识符

在C++语言中，标识符用以描述程序中使用的各种数据对象的名称。用户可以根据需要灵活地为各种数据对象命名，但是，在命名数据对象时也需要遵循一定的原则。下面是用户使用标识符时必须遵循的原则。

1.大小写区分原则

由于 C/C++语言编译器是大小写敏感的，因此，用户在命名标识符时必须区分大小写字母。在C++语言中，符号相同而大小写不同的标识符将会被编译器识别为两个不同的标

识符。例如，myData 与 mydata 即为两个不同的标识符。

2.标识符长度限制

通常情况下，绝大多数的C++编译器都对用户声明的标识符给出了长度限制，即用户声明标识符长度不能超出 255 个符号。而实际情况是，用户几乎不可能使用如此长的符号串作为标识符，因此，用户无须担心标识符长度越界。

3.起始符号约束

标识符通常由字母、数字与下划线构成，用户在声明标识符时，应以字母或下划线作为起始符号，而数字不能作为标识符的起始符号。

虽然用户可以使用下划线作为标识符的起始符号，如_myData，这在语法上没有任何错误，但是，C++编译器往往也会定义一些下划线起始的标识符。为了避免与编译器冲突，用户在程序设计过程中最好不要采用这样的形式。

4.不能与关键字同名

由于关键字是系统预先定义的具有特殊含义的符号串，用户在声明标识符时不能与关键字同名，否则会发生命名冲突，导致程序不能编译运行。

5.见名知义原则

为了提高用户程序的可读性，建议用户在为数据对象命名时，应遵循见名知义原则。即用户可以根据数据对象的名称直观地了解到该数据对象表示的具体含义。例如，在对学生年龄指定名称时，使用 stuAge 作为标识符比使用 abc 作为标识符更利于用户对程序语句的理解。

在上述 5 条原则中，前 4 条具有强制性约束，用户在使用标识符时必须遵循这 4 条原则，否则不能编译运行，而第 5 条原则不具有强制性约束。建议读者遵循第 5 条原则，这样可以大大提高程序的可读性与可维护性。

2.1.2　关键字

关键字是由系统预先定义的具有特殊含义的符号串。在C++语言中，共有 63 个关键字，如下所示。

asm	default	float	operator	static_cast	union
auto	delete	for	private	struct	unsigned
bool	do	friend	protected	switch	using
break	double	goto	public	template	virtual
case	dynamic_cast	if	register	this	void
catch	else	inline	reinterpret_cast	throw	volatile
char	enum	int	return	true	wchar_t
class	explicit	long	short	try	while
const	export	mutable	signed	typedef	
const_cast	extern	namespace	sizeof	typeid	
continue	false	new	static	typename	

用户在声明标识符时，不能与上述关键字重名。在程序设计中，这些关键字的使用频率有着显著不同。本书将在以后篇幅中对涉及的常用关键字进行相应的介绍。

2.2 常量与变量

无论何种数据，在C++语言中均可分为常量类型或者变量类型。顾名思义，常量是在整个程序运行过程中，取值不发生任何变化的数据；反之，在程序运行过程中，取值可以发生变化的数据即为变量。

2.2.1 常量

在C++语言中，通常有3种类型的常量：文字常量、宏常量和声明常量。这3种常量在使用时有着明显的区别。

1.文字常量

在C++语言中，那些直接以值的形式给出，并且在程序运行过程中，取值不可改变的数据称为文字常量(literal constant)。根据取值不同，文字常量也分为不同的数据类型，关于数据类型的内容将在以后篇幅中介绍。

例如，在C++语言中，存在如下语句：

```
totalNumber＝everyNum＊100;
```

在该语句中，100即为文字常量，表示取值为100的一个整数。在C++语言中，文字常量确实存在于内存的某个位置，但是用户不可以对其进行访问，也就是说，文字常量是不可寻址的。

2.宏常量

在C++语言中，用户可以使用宏来声明常量。通常情况下，用户应该在程序代码的起始处定义宏，其格式如下：

```
#define 宏名称 取值
```

例如，当用户需要使用圆周率常量时，可以采用如下的宏定义形式：

```
#define pi 3.1415926
```

当某个常量值需要在程序中多次使用时，采用宏常量形式比采用文字常量形式有着明显的优势：

(1)采用宏常量形式相当于为文字常量取了个名称，用户更加容易记忆和理解该常量代表的含义。

(2)如果在程序中使用文字常量，当该常量多次出现时，难免出现书写错误的情况，而且这些错误难以及时发现。

(3)如果需要指定新的常量取值时，使用宏常量情况下，只需要修改宏即可，而使用文字常量情况下，则需要对程序中所有出现的文字常量进行修改。

3.声明常量

事实上，宏常量也有其固有的缺陷。当用户使用宏常量时，不能指定该常量的数据类型，编译器也不能对该常量进行类型检测，成为一个重大隐患。因此，在C++语言中，要求使用声明常量来取代宏常量。声明常量格式如下：

```
const 数据类型 常量名称＝常量取值;
```

用户使用 const 关键字表示开始常量声明，然后给出常量的数据类型和常量名称，并在声明同时给出常量取值。例如，用户可以使用如下方式声明圆周率常量：

```
const float pi=3.1415926;
```

以上介绍了C++语言中常见的常量类型，并比较了这 3 种类型之间的异同点。这里再次强调，在C++语言中，用户应使用 const 声明方式来使用常量。

2.2.2　变量

与常量相比，在程序运行过程中，变量取值可以发生变化。用户在使用变量前，应该对该变量进行定义和初始化。

1.定义变量

用户在使用变量之前，应先定义该变量。在程序中，一个变量只能定义一次。用户可以使用如下结构来定义变量：

```
数据类型 变量 1,变量 2,…,变量 n;
```

在C++程序中，用户定义变量时先指定变量类型，然后给出需要定义的变量名称即可。如果用户需要定义 3 个整数 x、y、z 时，可以使用下面的语句：

```
int x,y,z;
```

当变量定义后，编译器即在内存中为该变量开辟了相应的存储空间，变量值即保存在该空间中，空间大小由变量的类型决定。在程序中，用户可以使用变量名称来访问变量，也可以通过变量存放地址来访问该变量。

2.变量初始化

当变量定义完成后，其存储空间中存放的数据即为初始值。由于存储空间是编译器随机指定的，因而其初始值也是随机的。为此，用户需要在定义变量的同时对该变量进行初始化操作，即用户为其指定初始值，从而提高程序的健壮性。

下面通过一个实例给出变量初始化赋值的过程。

【例 2-1】 定义 3 个整型变量并对比其初始化前后的取值。

打开 Visual C++ 6.0(以后如不特别说明，均是在该环境中创建)，创建一个空的控制台应用工程，在工程中添加一个C++文件，打开文件并加入如下代码。

```
#include <iostream>
using namespace std;
int main(int argc,char** argv)
{
    int x,y,z;
    int a=0,b=1,c=10;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
    cout<<"z="<<z<<endl;
    cout<<endl;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
```

```
    cout<<"c = "<<c<<endl;
    system("PAUSE");
}
```

程序运行结果如图 2-1 所示。

图 2-1 【例 2-1】程序运行结果

在上面程序中，x、y、z 这 3 个变量没有赋值，系统显示，它们的取值为随机产生的；而 a、b、c 这 3 个变量在定义的同时进行了赋值，其值为指定数据。

用户在初始化变量时，需要注意以下两点：

(1)建议用户在定义变量的位置立即进行初始化赋值，而不是在程序中利用赋值语句进行初始化赋值。

(2)变量初始化应逐一进行，不能采用如下方式：

```
int a=b=c=0;
```

2.3 基本数据类型

在C++语言中，为了方便用户使用，定义了多种数据类型。根据其不同特点，可以分为基本数据类型和构造数据类型，而构造数据类型是由基本数据类型按照一定规则生成的。因此，掌握基本数据类型是熟练使用C++语言的基础。在本节中，将详细介绍数值类型、字符类型、布尔类型。

2.3.1 数值类型

在程序设计语言中，用于完成数学运算的数据类型均称为数值类型，数值类型的数据主要分为整型数据和浮点型数据。

1.整型数据

在C++语言中，根据整数的取值范围和存储空间大小可以将整型数据进一步进行划分，而每一个整型数据都可以用不同数制形式来表示。

1)integer 类型

在C++语言中,integer 类型(简写为 int)是最常用的数据类型,每个 integer 类型占用 4 字节来存储数据,表示－2 147 483 648～2 147 483 647 的整数。如果用户不需要表示负数,可以将整型数据声明为无符号整型,用以表示 0～4 294 967 295 的整数。

```
int x,y;
unsigned int a,b;
```

上面的语句分别定义了整型变量 x 和 y;无符号整型变量 a 和 b。用户在定义同一类型的多个变量时,变量名之间用逗号分开,定义语句以分号结束。

2)short 类型

在实际应用中,为了减少数据的存储空间,可以使用短整型(short int,简写为 short)数据。每个短整型数据占用 2 字节,表示－32 768～32 767 的整数,如果将其定义为无符号短整型,则可以表示 0～65 535 之间的整数。下面给出它们的定义形式:

```
short int x,y;
unsigned short int a,b;
```

3)long 类型

为了表示更大范围内的整数,用户可以使用长整型(long int,简写为 long),此时,每个数据占用 8 字节存储空间。在C++语言中,长整型定义方式十分特殊,根据不同的系统平台,可以使用 long 或者 long long 来定义长整型。

一般情况下,当用户使用计算机为 64 位机时,long 表示长整型,而在 32 位计算机上,long 与 int 类型是等价的,即一个 long 类型数据也是占用 4 字节而不是 8 字节来表示整数。为此,在 C99 标准中,规定用户使用 long long 来表示长整型,每个 long long 类型数据占用 8 字节存储空间,表示－2^{63}～2^{63}－1 的整数或者 0～2^{64}的正整数。下面给出它们的定义形式:

```
long long x;
unsigned long long x;
```

Visual C++ 6.0 及以前版本不支持 long long 类型。

4)整数表示

在C++语言中,整数可以采用多种表示形式,分别是十进制表示、八进制表示和十六进制表示。默认情况下,整数都是十进制表示,为了表示八进制整数,需要在数字前加 0,为了表示十六进制整数,则需要在数字前加 0x。

例如,16 表示十进制整数;016 表示八进制整数,即十进制中的 14;0x16 表示十六进制整数,即十进制中的 22。

2.浮点型数据

在C++语言中,使用浮点型数据来表示非整型数据,浮点型数据可以划分为两类:单精度浮点型数据与双精度浮点型数据。而每个浮点数都可以使用小数表示法和科学计数法两种形式来描述。

1)float 类型

在C++语言中,每个 float 类型数据占用 4 字节空间,用以表示 1.18×10^{-38}～3.40×10^{38}的浮点数。可以按如下方式定义 float 类型数据:

```
float x,y;
```

2)double 类型

为了表示更大范围的浮点数,C++语言定义了 double 类型,每个 double 类型数据占用 8 字节的存储空间,可以表示 2.23×10^{-308}～1.79×10^{308}的浮点数。如果还需要在更大范围内表示数据,用户可以使用扩展的 double 类型,即 long double 类型,在C++语言中,每个 long double 类型占用 10 字节存储空间,可以表示 $3.37\times10^{-4\,932}$～$1.18\times10^{4\,932}$的浮点数。

3)浮点数表示

对于每个浮点数,用户可以使用小数点表示,也可以使用科学计数法表示。小数点表示浮点数比较简单,即浮点数包括整数部分、小数点和小数部分。下面主要介绍科学计数法表示,其形式如下:

数字部分E[符号位]指数部分

例如,浮点数 127.34,可以用科学计数法表示为 1.2734E2(1.2734×10^{2})或 12734E－2(12734×10^{-2})。

在C++语言中,浮点数与整数相比有两个主要不同:

(1)整数可以分为有符号整数与无符号整数,而浮点数都是有符号的。

(2)整数数据都是精确表示的,而浮点数数据却有精度限制,float 类型可以提供最高 7 位的有效数字,而 double 与 long double 可以提供最高 16 位的有效数字。

默认情况下,C++语言认为所有浮点数均为 double 类型,用户可以通过在浮点数后方加上字母 f,用以告诉系统该数据为 float 类型。例如:

```
float x,y;
x=12.34;
y=12.34f;
```

在上面表达式中,12.34 为 double 类型,而 12.34f 为 float 类型,因此,系统在编译程序时会对 x=12.34 语句给出警告信息,并将 double 类型数据 12.34 自动转换为 float 类型。

在科学计数法表示浮点数时,指数部分数字只能为整数。

2.3.2 字符类型

在C++语言中,字符类型用来表示单一的符号,一个字符占用 1 字节存储空间。字符类型是C++语言的基本数据类型,是构成字符串类型的基础。

1.字符类型定义

用户可以使用如下方式定义一个字符变量。

```
char asc;
```

在理论上,char 类型可以表示 256(2^{8})种符号,而实际上,用户主使用字符类型来表示

ASCII 码表上的各种字母、数字、标点和控制字符。

2.字符类型赋值

在 ASCII 码表中，字符包括可显示字符（字母、数字和标点）和不可显示字符（控制字符）两类，其赋值方式也有所不同。

1）可显示字符赋值

对于 ASCII 码表中绝大多数可显示字符用户可以直接对其赋值，所赋的值要用单引号界定，如下所示。

```
asc='A';
asc='9';
asc=',';
```

在可显示字符中，有 3 个字符具有特殊应用，其赋值也与众不同。这 3 个字符的含义与赋值方式见表 2-1。

表 2-1　特殊可显示字符赋值

符　　号	特殊含义	赋　　值
反斜杠（\）	转义字符	\\
单引号（'）	字符数据定界符	\'
双引号（"）	字符串数据定界符	\"

用户可以通过如下语句将反斜杠、单引号与双引号赋值给字符变量：

```
asc='\\';
asc='\'';
asc='\"';
```

2）不可显示字符赋值

对于 ASCII 码表中不可显示字符（主要是控制字符），用户需要使用转义字符进行赋值，例如，用户需要将变量 asc 赋值为换行符，可以采用如下语句：

```
asc='\n';
```

常见的控制字符与对应的转义字符关系见表 2-2。

表 2-2　控制字符与转义字符对照表

控制字符	转义字符
\a	蜂鸣
\b	向后退格
\f	换页
\n	换行，光标至下行行首
\r	回车，光标至本行行首
\t	水平制表符
\v	垂直制表符
\0	空字符（NULL），不是空格符

3)ASCII 码值赋值

此外,ASCII 码表中的任何符号,用户都可以使用该符号的 ASCII 码值进行赋值。其赋值语句如下所示:

字符变量=整数;

即用户将整型值赋给字符变量,系统自动将该整数转换为与其数值对应的 ASCII 编码,因此,整数取值应为 0～127,例如:

```
asc=65;
asc=32;
```

这两条语句分别将字母 A 与空格赋值给字符变量 asc。

2.3.3 布尔类型

布尔类型是在 C99 标准中添加的数据类型。在早期 C 语言中没有布尔类型,用户使用整数 1 或 0 来表示真或假的不同状态。为了改变这种混乱状况,人们在C++语言中引入了布尔类型。

用户可以使用如下方式定义布尔型变量:

```
bool flag;
```

每个布尔型数据只有两个取值 true 和 false,分别表示真与假这两种状态。而有趣的是,当用户使用 cout 语句显示布尔型数据时,不会显示 true 和 false,而是将 true 显示为 1,false 显示为 0。

在实际情况中,C++语言允许(但不建议)用户将整型数据赋值给布尔型变量,系统会自动将整型数据转换为布尔型数据,整数 0 表示 false,而所有非 0 整数(无论正负)均表示 true。

2.3.4 数据类型转换

当不同类型的数据进行运算时,需要进行类型转换。在C++语言中,数据类型转换主要有两种形式:隐式数据转换和显式数据转换,其中显式数据转换又包括 C 风格转换和C++标准转换。

1.隐式数据转换

在C++语言中,字符数据、布尔数据、整数数据和浮点数据进行相互运算时,彼此之间将自动进行转换,转换原则是低精度数据(即占用存储空间较少的数据)向高精度数据(即占用存储空间较多的数据)转换,如图 2-2 所示。

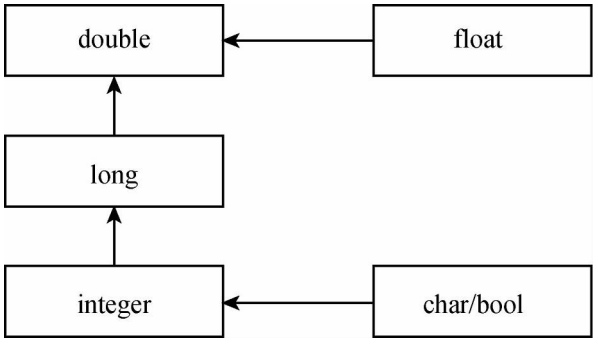

图 2-2 数据类型转换示意图

在图 2-2 中，有水平转换箭头和垂直转换箭头。水平转换箭头表示无论是否存在不同数据类型之间的运算，都要进行转换，即 float 类型数据一旦赋值，将自动转换为 double 类型数据，char 类型数据与 bool 类型数据一旦赋值，将自动转换为 integer 类型数据。

垂直转换箭头则表示，如果有 integer、long 和 double 这 3 种类型数据同时进行运算，数据类型转换的方向为：低精度数据将转换为高精度数据，整数转换为浮点数。

反之，如果按照箭头相反方向进行数据转换，C++语言允许进行这种转换，但是，在转换过程中会造成数据精度丢失。下面给出一个C++语言中基本数据类型转换的例子。

【例 2-2】　C++中数据的隐式转换。

创建一个空的控制台应用工程，在工程中添加一个C++文件，打开文件加入如下代码。

```
#include <iostream>
using namespace std;
int main(int argc,char** argv)
{
    float a;
    int b;
    char c;
    bool d;
    a=123.45;
    b=154;
    c='A';
    d=false;
    cout<<a<<endl;
    cout<<b<<endl;
    cout<<c<<endl;
    cout<<d<<endl;
    a=a+b+c+d;
    cout<<"a = "<<a<<endl;
    b=a;
    cout<<"b = "<<b<<endl;
    c=a;
    cout<<"c = "<<c<<endl;
    d=a;
    cout<<"d = "<<d<<endl;
    system("PAUSE");
}
```

程序运行结果如图 2-3 所示。

图 2-3 【例 2-2】程序运行结果

在【例 2-2】中，a、b、c、d 是 4 个不同类型的变量，其中，c 是 char 类型，d 是 bool 类型，它们在赋值同时即转换为 int 类型；同样，a 是 float 类型，在赋值时将自动转换为 double 类型存放。这 4 个数据相加时，将自动转换为精度最高的 double 类型进行加法运算，然后将运算结果分别以 double 类型、int 类型、char 类型和 bool 类型输出。

这里，特别强调以下几点：

(1)浮点型数据转换为整型数据时，小数点以后部分直接丢弃。

(2)浮点型数据转换为字符型数据时，浮点数先转换为整数 X，然后将该整数转换为 char 类型对应的－128～127 之间的整数 Y，方法如下：

X＝256×n＋Y；

其中，n 是非负整数；Y 取值为－128～127。如果 Y＜0，则不显示字符，否则，显示 ASCII 码表中对应的符号。

(3)浮点型数据转换为布尔型数据时，浮点数先转换为整数，0 表示 false(0)，而所有非 0 数均表示 true(1)。

2.显式数据转换

使用隐式数据转换固然方便，但在进行数据隐式转换时，编译器默认数据类型的转换是合法与安全的，因而不作任何类型检测，从而增加了程序中的不安全性。为此，在C++语言中，提倡使用显式数据类型转换。

1)C 风格转换

传统的显式数据类型转换是对 C 语言中显式数据类型转换的继承，用户在进行数据转换时应遵循以下语法规则：

(type)表达式；

或

type(表达式)；

这两种语法结构功能相同，均是将表达式的数据类型强制转换为指定数据类型。例如，用户将【例 2-2】中变量表达式 a＋b＋c＋d 的运算结果转换为布尔类型，则可以使用下面的语句：

(bool)(a＋b＋c＋d)；

或

```
bool(a+b+c+d);
```

2)C++标准转换

传统类型转换形式可以对任何类型进行转换,但是在转换过程中却忽略了不同类型数据进行转换时存在的巨大不同。为此,C++语言提供了更加精确的标准转换形式,并推荐使用标准转换形式来替换传统转换形式。

使用最多的标准转换形式如下所示:

```
static_cast<type>表达式;
```

该式表示将表达式类型转换为 type 类型。因此,用户如果需要将【例 2-2】中的变量表达式 a+b+c+d 的类型转换为 char 类型,可以使用如下方式:

```
static_cast<char>(a+b+c+d);
```

2.4　表达式与运算

在C++语言中有着丰富的表达式和运算模式,它们是构成C++语句的基础。本节将对这些表达式与相关运算进行详细的介绍。

2.4.1　表达式

在C++语言中,表达式是由一个或者多个操作数构成的。操作数包括程序中的常量、变量、函数名称和其他标识符。每个表达式可以是操作数单独构成,也可以是操作数和作用在其上的操作符共同构成。

1.简单表达式

由单个操作数构成的表达式是简单表达式,下面是简单表达式的例子。

```
3.1415926
"china"
upperBound
```

以上例子分别表示一个常数,取值为 3.141 59,类型为 double;一个字符串常数,取值为字符串首字符地址,类型为 char;一个变量,取值与类型由用户定义决定。

更多情况下,表达式是一个或者多个操作数和作用在其上的操作符共同构成的。不同类型的操作符构成不同类型的表达式,例如:

```
salary-tax
firstName+"."+lastName
```

通常情况下,表达式的运行结果是个右值,即表达式不能出现在赋值语句的左边,也就是说,用户不能对表达式赋值。

2.复杂表达式

当表达式中包括两个或者两个以上操作符时,这样的表达式称复杂表达式。下面是一个复杂表达式的例子:

```
x>0&&y>0
```

在这个表达式中,有两个子表达式:x>0 和 y>0,这两个关系表达式的运算结果进行与

运算，最终得到 x>0&&y>0 的值。

对于表达式，特别是复杂表达式而言，最重要的问题是按照正确的运算次序来求取表达式的值。因此，用户在求值时要遵循优先性原则和结合性原则，这些原则将在后续章节中介绍。

2.4.2 赋值运算

赋值运算是C++语言中的基本运算之一，通过赋值运算，可以将一个表达式的值传递给变量。赋值运算分为基本赋值运算、嵌套赋值运算和组合赋值运算3类。

1.基本赋值运算

在C++语言中，基本赋值语句由3部分构成，如下所示：

```
leftValue=rightValue;
```

在上面语句中，“=”并不表示等于，而是表示赋值；“=”左边部分称左值，右边部分称右值。上面语句应读做：将 rightValue 的值赋予 leftValue。因此，在C++语言中，左值通常是变量(严格说，是变量地址)，而右值是表达式。下面是赋值语句的例子：

```
pi=3.1415926;
```

用户在使用赋值语句时，必须保证右值的类型与左值类型相同，如果不同，必须将右值类型转换为左值类型。

2.嵌套赋值运算

在C++语言中，进行嵌套赋值操作，即赋值表达式的右值依然是一个赋值表达式，例如：

```
a=b=c=5;
```

为合法语句，相当于依次进行以下赋值运算：

```
c=5;
b=c;
a=b;
```

这里，特别指出，在变量定义时，不能使用嵌套赋值，例如：

```
int a=b=c=5;
```

为非法语句，应当修改为下面的形式：

```
int a=5;
int b=5;
int c=5;
```

3.组合赋值运算

在C++语言中，用户可以将算术运算与赋值运算组合在一起使用，构成组合赋值方式，以简化程序结构。例如：

```
a=a+5;
```

是赋值语句，其右值为算术表达式 a+5，表示将变量 a 的当前值加5后，赋值给变量 a。为此，用户可以将该语句用组合赋值方式给出，如下所示：

```
a+=5;
```

C++语言中的常见组合赋值表达式及其含义见表2-3。

表 2-3　C++ 语言中常见组合赋值表达式及其含义

组合赋值表达式	含　义
a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b

2.4.3　算术运算

在C++语言中，数值数据可以进行算术运算，用户可以使用运算符或C++语言提供的数学函数完成任意的算术运算。

1.四则运算

四则运算是最基本的算术运算，用户可以使用+、-、*（×）、/（÷）和括号来完成各种四则运算。C++语言中的四则运算表达式与数学四则运算表达式基本相同，只是在C++语言中，只有小括号，没有中括号和大括号，并且“×”用“*”表示，“÷”用“/”表示，这一点，在使用时应注意。例如，数学表达式[(15×a+6)-(b÷5)]×(a+2)，在C++语言中应使用如下表达式表示：

```
((15*a+6)-(b/5))*(a+2)
```

2.整数运算

当运算数全为整数时，用户可以进行整数运算。在C++语言中，整数运算有两个，整除和取余。整除运算用于获取两个整数的整数商，其运算符与四则运算中的除法符号相同；取余运算用于获取两个整数相除而得的余数，其运算符号为%。例如：

```
7/3=2
7%3=1
```

这里，特别强调指出，由于“/”既可以表示一般除，也可以表示整数除，因此，C++语言规定，当“/”两边的数据全为整型数据时，“/”表示整数除，否则为一般除。下面给出一段示例程序：

```
int a,b;
double x,y;
a=7;
b=2;
x=7;
y=2;
cout<<a/b<<endl;
cout<<x/y<<endl;
cout<<a/y<<endl;
```

在上面程序中，虽然a,b均赋值为整数，但是，由于x,y是double类型数据，因此程序运行结果依次为3,3.5和3.5。

3.数学函数

为了进行更复杂的数学运算，用户需要使用C++语言提供的数学函数。C++语言中的数学函数与C语言中的数学函数基本兼容，但是，用户在C++语言中使用数学函数时，应在程序首部导入相应函数库文件，如下所示：

```
#include <cmath>
#include <cstdlib>
```

常用的数学函数使用格式与功能描述见表2-4，用户可以查阅执行。

表2-4 C++语言中常用数学函数表

函　数	功　能	格　式
abs(int)	返回整数绝对值	abs(−5)=5
fabs(double)	返回浮点数绝对值	fabs(−0.5)=0.5
sqrt(double)	返回浮点数正平方根	sqrt(4)=2
pow(double,double)	返回任意浮点数的幂	pow(2.3,3.5)=$2.3^{3.5}$
exp(double)	返回以e为底的幂	exp(10)=e^{10}
log(double)	返回自然对数	log(10)=ln 10
log10(double)	返回以10为底的对数	log10(100)=lg100
sin(double)	返回正弦函数值	sin(30/180 * 3.14)=sin30°
cos(double)	返回余弦函数值	cos(30/180 * 3.14)=cos30°
tan(double)	返回正切函数值	tan(30/180 * 3.14)=tan30°
asin(double)	返回反正弦函数值	asin(0.5)=arcsin0.5
acos(double)	返回反余弦函数值	acos(0.5)=arccos0.5
atan(double)	返回反正切函数值	atan(2)=arctan2
rand()	返回[0,RAND_MAX−1]之间随机整数	

用户在使用上述函数时，特别要注意以下几点：

(1)所有的三角函数均使用弧度值而不是角度值。

(2)RAND_MAX是系统定义的大整数，根据系统不同可能有所不同，一般系统定义其为32 767。

(3)在使用rand函数前，用户应使用如下语句进行环境初始化，否则每次生成的随机数相同：

```
srand(time(NULL));
```

【例2-3】 生成10个−1 000～1 000的随机整数。

分析：rand函数用于生成[0,RAND_MAX−1]之间的随机数，为了生成指定范围[min,max]之间的随机数，用户可以使用如下公式进行处理：

```
min+(rand()/(RAND_MAX+1))*(max−min)
```

由于这里生成的随机数包括max，所以在公式中应使用RAND_MAX+1，而不是RAND_MAX。

步骤：创建一个空的控制台应用工程，在工程中添加一个C++文件，打开文件并加入如

下代码。

```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;
int main(int argc,char** argv)
{
    int i=0;
    int ret;
    srand(time(NULL));
    for(i=0;i<10;++i)
    {
        ret=((double)rand()/(RAND_MAX+1))*2000-1000;
        cout<<ret<<endl;
    }
    system("PAUSE");
}
```

程序运行结果如图 2-4 所示。

图 2-4　【例 2-3】程序运行结果

在【例 2-3】中，为了能初始化随机环境，需要使用 time(NULL)函数获得当前系统时间参数，为此需要在程序首部导入头文件 time.h。此外，在生成指定范围内的随机数时，程序将 rand 函数值转换为 double 类型，然后再进行运算，这一点是十分重要的。用户可以尝试一下，如果没有这步转换会有怎样的结果。

2.4.4　逻辑运算

逻辑运算是C++语言中重要而特殊的一类运算。在逻辑运算中，参与运算的运算数或表达式必须是布尔类型的，而逻辑运算结果也是布尔类型的。

在C++语言中，一共有 3 种逻辑运算：与、或、非。这 3 种运算的运算符、功能、用法和含

义见表 2-5。

表 2-5 C++语言中的逻辑运算

运算符	功能	用法	含义
!	逻辑非	! express	表达式逻辑值取反
&&	逻辑与	express1 && express2	两表达式全为 true 时取 true,其余为 false
||	逻辑或	express1 || express2	两表达式全为 false 时取 false,其余为 true

在逻辑与运算中,如果第一个表达式值为 false,根据逻辑与运算特点,无论第二个表达式取何值,整个与运算结果都为 false,因此,C++程序将不再计算第二个表达式的值;同理,当进行逻辑或运算时,如果第一个表达式的值为 true,则无论第二个表达式取值如何,整个或运算结果都是 true,因此,C++程序也不再计算第二个表达式的值。表 2-6 给出了常用逻辑运算取值关系。

表 2-6 逻辑运算取值关系表

表达式 A	表达式 B	A && B	A || B	! A	! B
true	true	true	true	false	false
true	false	false	true	false	true
false	true	false	true	true	false
false	false	false	false	true	true

2.4.5 关系运算

关系运算又称比较运算,用以判断两个表达式是否满足一定的关系。因此,参加关系运算的表达式取值必须是可以比较的,而关系表达式的取值一定是布尔值。

在C++语言中,一共有 6 种关系运算,这些运算的运算符与含义见表 2-7。

表 2-7 C++语言中关系运算符及其含义

运算符	表达式	含义
==	express1==express2	两表达式相等取 true,否则取 false
!=	express1!=express2	两表达式不等取 true,否则取 false
>	experss1>express2	表达式 1 值大于表达式 2 值取 true,否则取 false
>=	express1>=express2	表达式 1 值大于等于表达式 2 值取 true,否则取 false
<	express1<express2	表达式 1 值小于表达式 2 值取 true,否则取 false
<=	express1<=express2	表达式 1 值小于等于表达式 2 值取 true,否则取 false

不同类型数据在进行关系比较时遵循不同规则:当数值数据比较时,同数学上的数字比较;字符数据在比较时,根据其在 ASCII 码表中的取值进行比较;而布尔数据比较时,true 大于 false。

此外,C++语言允许一些特殊的关系运算存在。例如,下面的表达式即为合法表达式:

5>4<3

该表达式的取值为 true。由于在关系运算中，>与<是同级别运算符，此时应对表达式自左向右进行运算。首先计算 5>4，这是一个关系表达式，取值为 true，这样，原表达式就转化为 true<3，因为在C++语言中，true 相当于 1，true<3 即为 1<3，取值为 true。

同理，表达式'A'<'a'>=1 的取值也为 true，用户可以根据上面的说明进行验证。

2.4.6　位运算

位运算是C++语言中学习起来比较困难的内容。所谓位运算就是以计算机的字位(bit)为单位进行的运算，用户可以对整型数据和字符型数据进行位运算。在进行运算时，计算机自动将运算对象值按二进制位串进行处理，得到新的二进制位串。通过使用位运算，用户可以直接向某个字位进行取值与赋值，这对于底层操作而言是极为重要的。

1.位逻辑运算

在C++语言中，常用的位逻辑运算有按位与(&)、按位或(|)、按位非(~)和按位异或(^)4 种。这 4 种运算的定义与逻辑上的与、或、非、异或相同，只是运算对象由 bool 类型转换为字位，运算结果也为二进制位。当用户进行异或操作时，两操作数相异得 1，相同得 0。表 2-8 为C++位逻辑运算功能表。

表 2-8　C++位逻辑运算功能表

A	B	A & B	A | B	A^B	~A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

位逻辑运算通常以字节为单位进行，因此，用户通常定义 short 类型数据或 char 类型数据进行位逻辑运算。下面以 short 类型变量 a 为例，给出一些常见位逻辑运算的例子。

- 将 a 清空：a=a^a。
- 取 a 中左起第 4 位的值：a & 0x10。
- 将 a 中左起第 4 位赋值为 1：a | 0x10。

2.位移运算

与位逻辑运算相比，位移运算是以整型数据或字符型数据对应的二进制位串整体作为处理对象的。在C++语言中，位移运算分为左移运算和右移运算两种。

1)左移运算

左移运算将一个二进制串向左移动指定位数，当位串左移后，右端空出的位置补 0，而左端移出的数位丢弃，左移运算的格式如下：

操作数<<移动位数

例如，5<<2，当 5 为 short 类型整数时，对应的二进制位串为 00000101，左移两位后，位串为 00010100，即十进制的 20。

2)右移运算

右移运算将一个二进制串向右移动指定位数，当进行右移时，右端移出的数位丢弃，左端空出的位置需要填补数位。当数据为无符号类型时，左端补 0；当数据为有符号类型时，如

果左端第一位(即数字符号位)为0则补0,为1则补1。右移运算格式如下:

操作数>>移动位数

当用户执行5>>2时,即对00000101右移两位,得00000001,即十进制的1。

位运算是C++语言中比较复杂也比较难以理解的运算模式,灵活掌握位运算对于底层程序设计是极为重要的。

2.4.7 其他运算

在C++语言中,还存在其他类型的运算,主要是逗号运算与选择运算。选择运算可以替换简单的选择语句,故选择运算将在分支程序设计中介绍,这里主要介绍逗号运算。

在逗号表达式中,用户可以使用逗号将多个表达式串联起来,其格式如下:

a=表达式1,表达式2

在上面逗号表达式中,用户从左向右计算每个表达式,并将最后一个表达式的值作为整个逗号表达式的取值。例如,下面表达式:

a=3*5,6-2

该表达式的值为4。

当逗号表达式与赋值表达式写在一起时,用户应特别小心,例如下面的语句:

```
a=3;
a=a*3,a+2;
```

此时,该语句执行完毕后,a为9,不是5也不是11。这是因为,逗号运算的优先级最低,第2条语句是逗号语句,先执行赋值表达式a=a*3,此时a为9,然后执行逗号语句的后一部分a+2,此时,逗号表达式的值为11,但是a的值为9。

如果用户对语句进行如下修改,则运行结果完全不同:

```
a=3;
a=(a*3,a+2);
```

上述语句执行完毕后,a的值为5。这是因为,在赋值语句中,先计算括号中的逗号语句,并将该语句值赋给a,而在逗号表达式中,先计算表达式a*3,然后计算a+2,并将a+2的值5作为表达式的值赋值给a。

2.4.8 运算优先级

C++语言中运算类型众多,表达式构成复杂,为了能正确地书写和理解各类表达式,用户必须了解运算的优先级。下面是C++语言中各种运算优先级的总结,居于前者优先级高。

(1)括号与下标。括号优先级最高,先括号内后括号外,如果有多重括号,先内部括号,后外部括号;对于数组、结构体、指针的下标运算应优先进行。

(2)先函数后运算。表达式中有函数存在,应先计算函数值,然后再进行运算。

(3)单目运算。一般情况下,单目运算优先于多目运算,主要单目运算有++、--、!、~等。

(4)算术运算优先于位运算,位运算优先于关系运算,关系运算优先于逻辑运算。在算术运算中,乘除优先于加减;在关系运算中,不等关系比较优先于等于比较。

(5)赋值运算优先于逗号运算。当表达式中两种运算优先级相同时,一般应按顺序自左

向右进行运算。如果用户不能确定运算的优先级时，应使用括号来说明优先级。

在表达式中使用括号来表示优先级是良好编程习惯的体现。

2.5　标准输入/输出

数据输入/输出功能是应用程序中必不可少的基本功能。在C++程序中，可以通过标准输入/输出和文件输入/输出两种方式来实现人机数据交换。所谓标准输入/输出方式，就是用户通过标准输入设备——键盘和标准输出设备——显示屏来实现人机间数据交换。

2.5.1　标准输入函数

在C++语言中，用户通过使用 cin 函数来实现标准数据输入。为了能正确使用 cin 函数，用户需要在程序头部导入基本输入/输出函数库 iostream，并将当前程序的命名空间设置为标准空间，即用户需要在程序头部加入如下语句：

```
#include <iostream>
using namespace std;
```

当 cin 函数工作时，C++程序将自动在存储空间中开辟一个缓冲区用于暂存从键盘获取的输入数据，当用户输入完毕后，再将缓冲区中的数据传递给 cin 函数指定的变量。

1. 单一数据输入

使用 cin 函数为单一变量赋值十分简单，用户可以通过使用下面的方式来调用 cin 函数。

```
cin>>变量名称;
```

例如，下面的语句：

```
cin>>x;
```

表示用户通过键盘获得变量 x 的取值。当程序运行时，用户输入 x 的取值信息后，存入输入缓冲区，最后，以 Enter 键作为输入结束标志。此时，cin 函数自动将缓冲区中信息传递给变量 x。

cin 函数在执行过程中，并不对数据合法性进行检测，因此，用户在键入变量值时应保证变量取值的合法性。

2. 多个数据输入

如果用户需要对多个变量进行赋值，可以采用两种不同的方式。一种方式是，当程序需要输入多个数据值时，可以多次使用 cin 函数，每次完成一个数据的赋值工作，如下所示：

```
cin>>变量 1;
cin>>变量 2;
```

…

cin>>变量 n;

另一种方式是,用户使用 cin 函数,一次完成对多个变量的赋值工作,如下所示:

cin>>变量 1>>变量 2>>…>>变量 n;

例如,下面的语句:

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

表示对 a、b、c 和 d 这 4 个变量进行赋值。

3.cin 函数中特殊符号

用户在使用 cin 函数输入数据时,空格键、Tab 键和 Enter 键十分特殊,用它们来表示数据与数据之间的分隔。例如,上面的语句,用户需要一次对 a、b、c、d 这 4 个变量赋值,当程序运行时,用户首先对变量 a 赋值,赋值完成后,可以使用空格键、Tab 键和 Enter 键中任何一个表示当前输入完毕,可以开始下一个输入。当所有数据输入完成后,按 Enter 键完成数据输入。

因此,对于"cin>>a>>b>>c>>d;"语句而言,用户在输入界面中输入:

1 空格 2 空格 3 空格 4 回车

则完成将 1、2、3 和 4 这 4 个数字依次赋值给 a、b、c 和 d 的功能。

这里特别指出,空格键、Tab 键和 Enter 键在 cin 函数中用做数据输入的分隔符号,因此,cin 函数不会将这 3 个符号存放在输入缓冲区中,并且用户不能使用 cin 函数完成空格键、Tab 键和 Enter 键的读取工作。

4.cin 函数特殊应用

为了完善 cin 函数的功能,C++语言允许 cin 函数对其功能进行扩充,允许 cin 函数读取任意符号和字符串。

1)cin.get()方法

在C++语言中,用户可以使用 cin.get()方法获取任意的字符信息。用户可以通过如下方式使用 cin 函数为字符变量 ch 赋值。

```
cin.get(ch);
```

或

```
ch=cin.get(ch);
```

这两种使用方法效果相同。

在 cin.get()方法中,用户在键盘上的任意输入均保存到缓冲区中,并传递给程序变量。例如,用户需要输入一个空格符给变量 a,应使用如下语句:

```
char a,b;
a=cin.get();
b=cin.get();
```

当程序执行时,用户在键盘上输入空格符,然后输入回车符表示输入结束。此时,第一次输入的空格符和第二次输入的回车符均进入缓冲区,并分别被赋值给 a 和 b。

2)cin.getline()方法

在C++语言中,用户可以使用 cin.getline()方法一次读入一行数据,该行数据必须以回车符作为结束标志,在数据中可以存在空格符和 Tab 键。其语法格式如下:

cin.getline(A,n);

在这里,A 是一个字符数组的名称,n 表示读入的字符长度限制。

用户在使用 cin.getline()方法时应注意以下几点:

(1)字符数组长度应大于输入字符串的长度。

(2)计算输入字符串长度时,表示输入结束的回车符不统计。

(3)如果 n 小于输入字符串长度,则读入该字符串前 n 个字符,如果 n 大于等于输入字符串长度,则读入整个字符串。

2.5.2　标准输出函数

在C++语言中,用户可以使用 cout 函数将一个或多个数据输出到屏幕上。与 cin 函数相似,cout 函数首先将数据存放在输出缓冲区中,然后将输出缓冲区中数据按要求显示在屏幕上。

1.数据输出

在C++语言中,用户可以按照下面的方式使用 cout 函数:

cout<<数据;

该语句功能是将数据在当前位置上输出。如果用户需要输出多个数据,可以按下面方式使用 cout 函数:

cout<<数据 1<<数据 2<<…<<数据 n;

这里,数据可以是任何类型的常量、变量和控制符号。

当 cout 函数将多个数据输出到屏幕时,数据与数据之间连续输出,没有分隔符号,因此用户不能够区分。为此,C++语言提供了 ends 与 endl 两个控制符供用户使用。ends 表示在当前位置插入空格符,并刷新输出缓冲;而 endl 表示在当前位置输出换行符,并刷新输出缓冲。

例如,下面的语句:

```
cout<<'a'<<'b'<<'c'<<endl;
cout<<'a'<<ends<<'b'<<ends<<'c'<<endl;
cout<<'a'<<endl<<'b'<<endl<<'c'<<endl;
```

这 3 条语句执行后结果如图 2-5 所示。

图 2-5　输出效果图

2.格式控制

在 cout 函数中,用户可以对输出的数据进行格式设置。这些设置绝大多数是对数据进行的,用户需要在输出数据前添加C++语言定义好的控制符,如下所示:

cout<<控制符<<数据;

此时,数据按控制符的要求显示。常用的控制符见表 2-9。

表 2-9 cout 函数中格式控制符

控 制 符	说 明
hex	以十六进制显示数据
oct	以八进制显示数据
showbase/noshowbase	产生数值前缀/不产生数值前缀
showpoint/noshowpoint	总显示小数点/仅小数存在显示小数点
showpos/noshowpos	非负数前显示+/不显示+
left/right	数据左对齐/右对齐
scientific/fixed	以科学计数法/定点小数显示浮点数
setprecision(n)	设置浮点数精度
setw(w)	设置数字显示宽度

这里,特别指出,用户如果需要使用 setprecision 与 setw 这两个控制符,需要在程序首部导入头文件 iomanip。下面给出格式化输出的例子。

【例 2-4】 求 1～10 的平方根,并以定点小数形式显示,要求达到 10 位精度,每个数字显示宽度为 15 位,右对齐。

创建一个空的控制台应用工程,在工程中添加一个C++文件,打开文件并加入如下代码。

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main(int argc,char** argv)
{
    int i;
    double t;
    for(i=1;i<=10;++i)
    {
        t=sqrt(double(i));
        cout<<setw(15)<<setprecision(10)<<fixed<<right<<t<<endl;
    }
    system("PAUSE");
}
```

程序运行结果如图 2-6 所示。

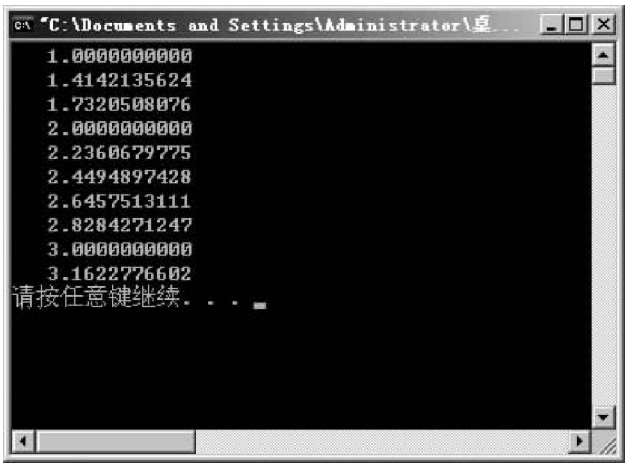

图 2-6　【例 2-4】程序运行结果

习　题　2

简答题

1. 请指出下面标识符中，哪些是合法的变量名称。

abc	3x	－ac	int	$
_3x	_b	next day	OK?	X *

2. 请指出下面哪些是合法的常量取值。

2^3	－0x2aL	0x7g	3e－5	'\n'
7ff	lg3	12.56E	'105'	"abc"

3. 请给出下面表达式的值。

(1)－3＋4 * 5－6

(2)－3＋4%5－6

(3)－3 * 4%－6/5

(4)(7＋6)% 5/2

4. 编写程序实现下列表达式的计算。

(1)$\sin 30^\circ - \tan 20^\circ$

(2)$23^2 - \sqrt{a^2+b^2}$

5. 求下列表达式的值。

(1)1<4 &&. 4>10

(2)!(5 >＝15)||(2<4－3)

第7章　类与对象

知识目标

◎ 熟练地创建类

◎ 熟练地建立类中的成员函数，特别是构造函数与析构函数

◎ 了解利用指针访问类的方法

技能目标

◎ 能使用 Visual C++ 6.0 开发环境创建类及其成员变量

◎ 掌握构造函数的重载、复制，能使用构造函数为类中的成员函数赋初始值，能区分析构函数与构造函数

◎ 掌握成员函数内联与外联的区别、函数重载以及函数缺省参数等

C++语言源于C语言，但是C++语言并不仅仅是“更好的C语言”，而是一门与C语言有本质区别的语言。在C++语言中，添加了面向对象机制，增加了软件的可扩展性与可重用性，大大提升了软件开发效率，也降低了软件维护成本。C++语言的精髓在于面向对象的使用，如果没有面向对象思想的指导，C++语言的优点将难以发挥。

7.1　类与对象概述

面向对象程序设计思想是当今主流的软件开发思想。在面向对象思想中，最重要的概念无非是对象与类。用户深刻认识这两个重要概念将有利于在程序设计过程中贯彻面向对象的思想，真正完成面向对象的程序设计。

7.1.1　对象及其特征

对象是面向对象程序设计的核心与关键所在。所谓对象指的是现实生活中存在的一个客观事物，可以是物质的，也可以是无形的，例如，一个人，一份计划，一次旅行活动等。

在面向对象程序设计中，对象被定义为系统中描述客观事物的一个实体，是面向对象系统中的一个基本单位。通常情况下，对象可以分为两个部分：属性与操作。属性是一个对象的静态特征，用来表示对象的状态，例如，一个人作为一个对象，其属性可以包括身高、体重、性别、年龄等内容，这些内容都是用来描述人的特征，在程序设计中，属性表现为若干数据类型组成的集合。与属性相对应的是对象的操作，操作也称服务，用以描述一个对象的一个动

作序列。必须特别指出的是,操作的主要功能是对对象的属性进行修改,修改对象自身的属性称自操作,修改其他对象的属性称它操作。

一个对象可以包括若干属性与服务,对象由自身属性与操作结合组成。在面向对象程序设计中,一个对象具有以下几个特征:

(1)唯一性:一个对象有一个唯一的名称,用以与其他对象区分。

(2)状态性:每个对象的属性构成一个状态集,用以表示对象的特定状态,用户可以对对象的状态进行修改。

(3)嵌入性:一个对象的成员也可以是另一个对象。

(4)独立性:对象与对象之间具有高度独立性,互相之间几乎没有干扰,依赖性也降至最低。

在C++语言中,用户在实现一个对象时,须将对象分成3个部分,分别是:

- 数据:用于描述对象内部属性与状态,这些数据只有对象自身可以访问,称私有数据。
- 方法:也称操作,是对象中定义的函数,可以对私有数据进行处理。
- 接口:是对象与其他程序的接口部分,系统通过接口可以调用对象中相应的操作。

7.1.2 类的定义

在面向对象概念中,每个对象用以描述一个具体的实体,而事实上,对每一个实体进行描述与处理是不需要的,也是不可行的。通常情况下,用户可将对象中非本质的属性与操作剔除,将众多对象中共有的、本质的属性与操作提取出来,将其定义为一个对象类,这样的过程即为系统抽象过程,产生的结果即为类。

在程序设计中,每个类的名字用一个标识符表示,显然,这个标识符在程序中应该是唯一的。类是对象集合的抽象,在对象集中的任何一个对象均视为类的一个实例。例如,人是一个类,表示所有人的共性特征,而张三是所有人这个集合中的一员,因而,张三即为人这个类的一个实例。

在面向对象程序设计中,类概念涉及数据结构(类的属性)、功能实现(类函数)与访问控制(类接口),通过它们可以完整地解决特定的问题。

在C++语言中,用户可以采用下面的方式来定义一个类:

```
class 类名称
{
    private:
        //私有数据与私有成员函数说明
    public:
        //公共数据与公共成员函数说明
    protected:
        //受保护数据与成员函数说明
};
```

用户在声明一个类的时候,需要进行3方面的操作:首先,利用关键字class来指定一个类的名称,显然,用户在程序中要按名称来访问类,因而类的名称必须具有唯一性;其次,用户在类当中必须说明相应的数据,这些数据用来描述类的种种属性;最后,用户应在类当中声明成员函数,这些函数可以对类的属性进行处理,也称方法。当整个类声明完毕后,应在

最后添加分号，也就是说，类的声明在C++程序中是一个复合语句，该语句要有结束标志，即分号。

用户在声明变量与成员函数时，可以将其声明为私有(private)、公共(public)和保护(protected)中的一种。这 3 种模式有明显区别：私有类型是只有类内部定义的成员函数才能访问；公共类型是除了类自身成员以外，任何其他类的成员也可以访问；保护类型指的是除了类自身以外，通过友元和类继承方式由其他特定类进行访问。通常情况下，在类中，须将数据声明为私有的，函数声明为公共的。下面给出一个类声明的实例。

【例 7-1】 声明工作人员类。在类中，可以说明工作人员的姓名、年龄、家庭住址信息；并且用户可以显示工作人员信息，并能修改其姓名、年龄和家庭住址信息。

分析：在本例中，3 个用以表示类属性的数据(姓名、年龄与住址)应声明为私有类型；此外，要求可以显示类中数据信息，修改姓名、年龄和家庭住址信息，因此，每个操作可以声明为一个成员函数，这些成员函数应声明为公共的。

步骤：在 Visual C++ 6.0 开发系统中，用户使用类时，应采用 Visual C++ 6.0 提供的类向导来生成类结构，不建议用户直接手写代码来生成类。

利用类向导生成类的具体步骤如下：

(1)在 Visual C++ 6.0 中创建一个空的控制台应用工程。

(2)在控制台中选择类视图，右击工程名，在弹出的快捷菜单中选择 New Class 选项，如图 7-1 所示。

(3)在弹出的 New Class 对话框中输入所建类的名称，其他选项保持默认值，并单击 OK 按钮，如图 7-2 所示。

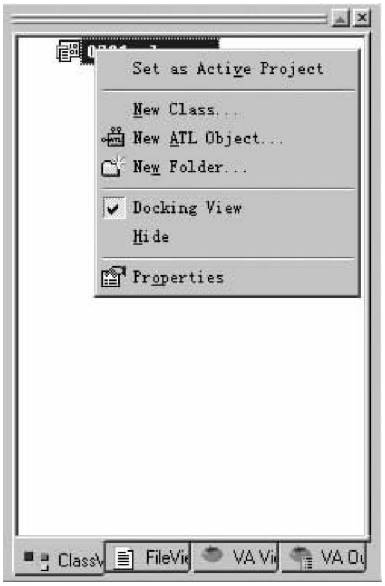

图 7-1　类视图

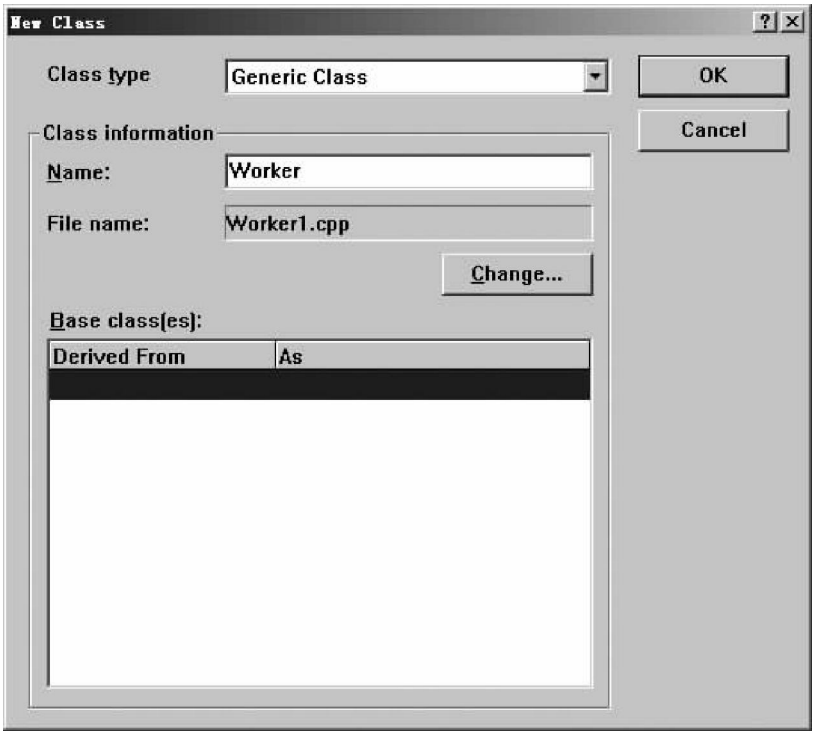

图 7-2　New Class 对话框

此时，对话框关闭，用户完成一个空的名为 Worker 类的创建工作。

(4)回到类视图，右击新生成的 Worker 类，在弹出的快捷菜单中选择 Add Member Variable 选项，如图 7-3 所示。

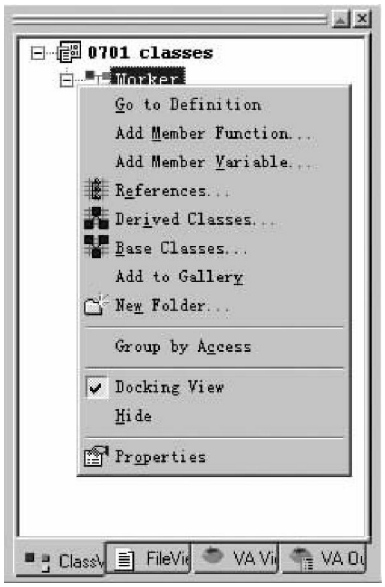

图 7-3 类视图添加成员变量

(5)在弹出的 Add Member Variable 对话框中选择成员变量的数据类型与变量名称和访问变量的权限说明,如图 7-4 所示。

图 7-4 Add Member Variable 对话框

单击 OK 按钮,即向类中添加了一个私有变量 char name[16],重复(5)操作,用户可以向类中添加其他变量。

(6)成员变量添加完毕后,用户再次进入类视图,进行添加成员函数的操作,右击 Worker 类,在弹出的快捷菜单中选择 Add Member Function 选项,进入 Add Member Function 对话框,在该对话框中,用户设定函数返回值类型,无返回值设定为空,然后设定函数参数类型,无参数设定为空,其他选项均取默认值,如图 7-5 所示,单击 OK 按钮,完成函数添加工作。用户可以利用这样的方式将其他成员函数添加完毕。

图 7-5 Add Member Function 对话框

当所有的成员变量与成员函数添加完毕后，类向导就完成了添加类的工作。当类添加完成后，在项目中自动生成两个文件：一个是类的头文件，在文件中是类的声明；另一个是类的实现文件，这是一个C++文件，在该文件中是类的成员函数的定义，用以实现成员函数的具体功能。用户可以直接在这两个文件中对类声明进行修改，但必须保证这两个文件中成员变量与成员函数在名称、类型与参数上的一致性。【例7-1】中类头文件如下所示：

```
class Worker
{
    public:
        void setAddress(char * inputAddr);
        void setName(char * inputName);
        void setWorkerAge(int inputAge);
        void listInfo(void);
        Worker();
        virtual ~Worker();
    private:
        char address[32];
        int age;
        char name[16];
};
```

以后，当用户需要建立对象时，均可参照本例执行。此外，在类声明代码中，有两个函数是自动生成的，即Worker()与~Worker()，分别称构造函数与析构函数，其作用以后再进行介绍。至于C++中的成员函数，在定义时，其格式与普通函数有一点不同，即用户须说明成员函数与所在类的关系，具体格式如下：

```
函数返回值 类名称::成员函数名(函数参数);
```

例如，下面的语句：

```
void Worker::setAddress(char * inputAddr);
```

在C++中，用户可以创建一个只有成员变量而没有成员函数的类。这在语法上是可行的，但是，这样的类的用途仅相当于一个结构体。

7.1.3　类与对象的关系

在面向对象技术中，类与对象是紧密相连又相互区别的概念。类是一组对象中具有共性的特征的抽象，反过来说，每个对象都包含了对象类中的所有特征，类是类对象的一个模板。在C++语言中，一个对象可以称为是一个对象类的一次应用，或者说是一个对象实例。在程序设计中，类相当于一个说明，用户不能利用类来进行程序设计，而是应当利用类来获得一个实例，也就是一个对象，并对这个具体的对象进行各种操作。从某种意义上来说，类很像一个标识符，而类的实例就是利用类定义的一个变量，也就是一个类变量。

7.1.4 利用类访问对象

在C++语言中，一个具体对象对应一个类的应用实例，也就是一个类变量。当用户声明完成一个类后，这个类并没有存在，只有当用户定义一个类变量后，才在内存空间开辟出一段位置用以存放类变量。这一点与结构体类型有相似的地方。用户可以使用下面的方式来定义一个类变量：

类名称 类变量名称；

例如：

```
Worker myWorker;
```

当类变量定义完毕后，就可以访问类变量了。在一个类中，既包括成员函数，也包括成员变量。用户可以使用“.”符号来访问类变量中的成员，具体格式如下：

类变量名.类成员函数；

例如，下面的语句：

```
myWorker.setWorkerAge(18);
```

该语句的功能是调用 myWorker 变量中的 setWorkerAge 成员函数，该函数的实际参数为 18。当该函数执行完毕后，myWorker 变量中的 age 分量值即设定为 18。

用户可以直接访问类变量中的成员函数，因为成员函数是 public 属性的。通常情况下，不允许用户直接访问类变量中的成员变量，因为这些成员变量往往都是 private 属性的。例如，下面的语句：

```
myWorker.age=18;
```

该语句试图直接将类变量中的成员变量 age 赋值为 18。但是，由于 age 是类 Worker 中的私有变量，只有 Worker 内成员函数才能访问，其他任何程序代码不能直接访问，故上面语句在编译时会出错。

下面给出一个利用类进行程序设计的例子。

【例 7-2】 编写代码，完成【例 7-1】中所提出的要求。

分析：在【例 7-1】中，已经声明了类 Worker，因此，首先须将 Worker 类中成员函数代码添加完毕。类中函数代码通常添加在对应类的实现文件——Worker.cpp 文件中。

步骤：由于类 Worker 已经声明完毕，用户只需打开 Worker.cpp 文件，在其中添加成员函数的实现代码即可。代码如下所示：

```
#include "Worker.h"
#include <iostream>
using namespace std;
Worker::Worker()
{
}
Worker::~Worker()
{
}
void Worker::listInfo()
{
```

```
    cout<<"Name:"<<name<<";Age:"<<age<<";Address:"<<address<<endl;
}
void Worker::setName(char * inputName)
{
    strcpy(name,inputName);
}
void Worker::setWorkerAge(int inputAge)
{
    age=inputAge;
}
void Worker::setAddress(char * inputAddr)
{
    strcpy(address,inputAddr);
}
```

在这里，构造函数 Worker()没有添加代码，析构函数～Worker()也没有添加代码，它们的功能在下节进行介绍。此外，在成员函数 listInfo()中使用了 cout 函数，因此用户需要添加相应头文件。

通过上面的步骤，实现了类 Worker 的定义工作，并可以在代码中调用对应类了。为此，用户需要在工程中添加一个空的.cpp 文件，可以命名为 main.cpp，用来添加主调函数。主调函数内容如下所示：

```
#include <iostream>
#include "Worker.h"
using namespace std;
int main(int argc,char** argv)
{
    Worker myWorker;
    char tmpstr[32];
    int tmpInt=-1;
    cout<<"请输入名称:";
    memset(tmpstr,0,sizeof(tmpstr));
    fflush(stdin);
    cin.getline(tmpstr,sizeof(tmpstr));
    myWorker.setName(tmpstr);
    cout<<"请输入年龄:";
    cin>>tmpInt;
    myWorker.setAge(tmpInt);
    cout<<"请输入地址:";
    memset(tmpstr,0,sizeof(tmpstr));
    fflush(stdin);
    cin.getline(tmpstr,sizeof(tmpstr));
```

```
    myWorker.setAddress(tmpstr);
    myWorker.listInfo();
    system("PAUSE");
    return 0;
}
```

在主调函数中,为了能使用类,必须预编译相关头文件 Worker.h。此外,在主调函数中,切记对类成员变量赋值必须通过成员函数来实现,不能直接赋值。程序运行结果如图 7-6 所示。

图 7-6 【例 7-2】程序运行结果

通过【例 7-2】程序的编写可以感受到:当程序规模很小时,难以看出面向对象程序设计相对于传统的面向过程程序设计的优势。但是当程序规模较大,复杂性较高时,面向对象程序设计则具有极大的优越性。

7.2 构造函数与析构函数

在 Visual C++ 6.0 开发环境中,当用户利用类向导创建一个类时,将自动生成两个成员函数,其中一个成员函数与类名称相同,另一个成员函数则是通过“~”加类名来命名的。这两个函数分别称构造函数与析构函数,它们在类中的主要作用是完成类的初始化处理与运行后类环境清理的工作。

7.2.1 构造函数

构造函数是类中的一个特殊函数,与普通成员函数相比,它有许多特别之处,主要表现在以下几个方面:

(1)构造函数名称是确定的,构造函数名称与类名称必须一致,不能任意命名。

(2)构造函数没有返回值,这里特别强调,是没有返回值,而不是返回值为空。

(3)用户无法获取构造函数地址。

(4)用户不能直接调用构造函数,构造函数是在定义类对象时自动运行的。

在C++语言中,构造函数的主要作用是对类中的成员变量进行初始化处理,为其指定缺

省值。用户在使用构造函数时,可以设定其参数,并对其进行重载与复制,下面详细介绍这些方法。

1.构造函数的参数

构造函数在通常情况下是没有参数的,如果需要通过构造函数对类成员变量进行初始化赋值操作,可以在构造函数中添加代码,直接对相关成员变量赋值,从而完成初始化功能。以【例7-2】为例,用户可以编写如下所示构造函数代码,完成数据初始化的工作。

```
Worker::Worker()
{
    memset(name,0,sizeof(name));
    age=-1;
    memset(address,0,sizeof(address));
}
```

没有参数的构造函数最为简单,但是无参数的构造函数有一个重要缺陷,即所有的类变量在定义时,其成员变量均初始化为同样的值,而很多情况下,需要将类变量的成员变量在定义时设定为指定值。因此,需要为构造函数设定参数,其方法如下:

```
Worker::Worker(char * cNameParam,int iAgeParam,char * cAddrParam)
{
    strcpy(name,cNameParam);
    age=iAgeParam;
    strcpy(address,cAddrParam);
}
```

这里,类的构造函数使用了参数。用户可以编写代码,将输入参数的值传递给对应的成员变量,从而完成类成员变量初始化的工作。在Visual C++ 6.0中,添加完有参数的构造函数后,不能忘记在头文件(Worker.h)中加入对该构造函数的声明,如下所示:

```
Worker(char * cNameParam,int iAgeParam,char * cAddrParam);
```

2.构造函数的重载

通过上面的说明,可以得到结论:构造函数可以有参数也可以没有参数。显然,构造函数可以是不唯一的,在C++语言中,一个类可以同时存在多个构造函数。那么,问题就转换为,如果类中同时存在多个构造函数,则定义类变量时,应如何选择唯一的构造函数执行。在C++语言中,可以使用重载的方式来解决这个问题。

所谓构造函数的重载,指的是根据构造函数中参数的格式与类型的不同,编译器自动选择相应的构造函数来执行的方式。在类中可以存在多个构造函数,但是,不允许有两个构造函数的参数格式与参数类型完全一致。

以【例7-2】为例,用户定义下面类变量时,其调用的构造函数是不同的,其调用代码如下所示:

```
Worker myWorker1;
Worker myWorker2("Mary",34,"New York");
```

显然,类变量myWorker1在定义时,没有给出参数,故编译器在执行时,调用的是没有参数的构造函数;而对于类变量myWorker2,在定义同时给出了3个输入参数,因而,编译器

在执行时，调用的是有 3 个参数的构造函数。最后强调一点，除了这两个构造函数之外，用户还可以定义新的构造函数，只需参数的格式与类型不同即可。

3.构造函数的复制

构造函数的复制，其实也是一个构造函数的调用方法。当用户定义一个类变量 A 时，用一个已经存在的类变量 B 来完成对 A 的初始化工作，即 A 在初始化时复制了 B 的构造函数。通过这样的方式，用户可以完成同类对象之间的成员变量值的传递。

在C++语言中，最常用的构造函数复制有两种形式：一种是将类变量作为参数传递给构造函数；另一种则是直接对类变量进行复制。具体形式如下：

```
Worker myWorker1("Tom",29,"Washington D.C.");
Worker myWorker2(myWorker1);
```

或

```
Worker myWorker2＝myWorker1;
```

在上面的语句中，myWorker2 即完成了对 myWorker1 的构造函数的复制，因而，在类变量 myWorker2 中，其成员变量的值与 myWorker1 的成员变量值相同。

最后，以【例 7-2】中的类为例，介绍一下构造函数的使用方法。

【例 7-3】 以【例 7-2】为例，使用构造函数为类中的成员函数赋初始值。

分析：在C++语言中，用户可以使用重载的方式来执行类的构造函数，还可以通过构造函数赋值的方式，利用一个已有的类变量的构造函数来为新的类变量赋值。

步骤：如【例 7-2】所示，创建一个 Worker 类，并在工程中添加一个空的 main.cpp 文件，打开该文件，加入以下代码。

```
#include ＜iostream＞
#include "Worker.h"
using namespace std;
int main(int argc,char** argv)
{
    Worker myWork;
    cout＜＜"无参数构造函数:";
    myWork.listInfo();
    cout＜＜endl;
    Worker myWork1("Catherine",29,"London");
    cout＜＜"全参数构造函数:" ;
    myWork1.listInfo();
    cout＜＜endl;
    Worker  myWork2(28);
    cout＜＜"单参数构造函数:" ;
    myWork2.listInfo();
    cout＜＜endl;
    Worker myWork3(myWork2);
    cout＜＜"构造函数复制:";
    myWork3.listInfo();
```

```
    cout<<endl;
    system("PAUSE");
    return 0;
}
```

与【例 7-2】相比，在 Worker 类中增加了两个构造函数。因此，用户应在 Worker.h 中增加相应内容，具体代码如下：

```
class Worker
{
    public:
        void setAddress(char * inputAddr);
        void setWorkerAge(int inputAge);
        void setName(char * inputName);
        void listInfo(void);
        Worker();
        Worker(int iAgeParam);  //增加的构造函数，下同
        Worker(char * cNameParam,int iAgeParam,char * cAddrParam);
        virtual ~Worker();
    private:
        char address[32];
        int age;
        char name[16];
};
```

除此之外，用户还应在 Worker.cpp 文件中添加相应的实现代码，具体如下所示：

```
Worker::Worker()                //无参数构造函数
{
    memset(name,0,sizeof(name));
    age=-1;
    memset(address,0,sizeof(address));
}
Worker::Worker(int iAgeParam)  //单参数构造函数
{
    memset(name,0,sizeof(name));
    age=iAgeParam;
    memset(address,0,sizeof(address));
}
Worker::Worker(char * cNameParam,int iAgeParam,char * cAddrParam)
//全参数构造函数
{
    strcpy(name,cNameParam);
    age=iAgeParam;
```

```
    strcpy(address,cAddrParam);
}
```

程序运行结果如图 7-7 所示。

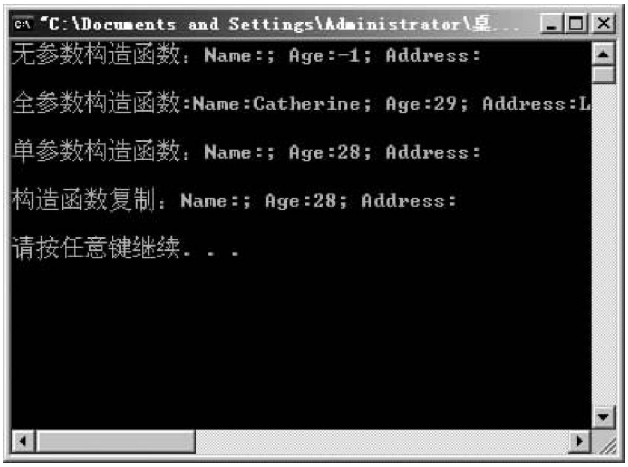

图 7-7 【例 7-3】程序运行结果

7.2.2 析构函数

与构造函数功能相对应的是析构函数，在C++语言中，当一个类变量使用完毕后，程序会自动调用析构函数，对类变量中的成员变量进行清理，从而在最大程度上避免程序出现异常。特别是类变量中，如果出现指针分量，务必将其设置为空。

析构函数是程序自动调用的成员函数，在C++程序设计中，当一个对象被删除或程序运行结束时，也就是说当类变量不再使用时，系统自动调用析构函数。虽然析构函数与构造函数有很多相似之处，但是它们之间依然有很多区别，主要表现在以下几个方面：

(1)在一个类中，可以有多个构造函数，但是只能有一个析构函数。

(2)在程序执行时，一定是先执行构造函数，后执行析构函数。

(3)构造函数必须是系统调用，析构函数通常也是系统调用，但是在必要时，用户也可以编写代码来调用析构函数，调用方式如下：

类变量名::析构函数名();

(4)构造函数可以有参数，析构函数没有参数。

相对而言，析构函数的使用比较简单。以【例 7-2】为例，可以在析构函数中加入如下代码，完成相应的操作：

```
Worker::~Worker()
{
    memset(name,0,sizeof(name));
    age=-1;
    memset(address,0,sizeof(address));
}
```

上述代码的主要功能是在清除成员变量前，将其取值设定为默认值。

7.3 类成员函数特征

类的成员函数是根据某个类需要解决问题的需求而定义的函数。通常情况下,用户只能通过调用成员函数来访问类中封装的数据。在C++语言中,构造函数、析构函数等都是成员函数。对于成员函数,用户主要了解内联与外联的区别、函数重载以及函数缺省参数等方面的内容。

7.3.1 内联与外联

在C++语言中,所有的成员函数可以分为两类:内联函数与外联函数。这两种函数在定义与使用时有一定区别,了解这些不同之处有助于读者编写出高效的程序。

1.内联函数与外联函数

所谓内联函数,指的是用户在声明一个类时,其成员函数以定义的形式给出而不是以声明的形式给出。也就是说,用户在类声明中,对于成员函数直接给出其实现代码;反之,如果成员函数的实现代码不是在类声明中给出,而是在类声明外给出的,则称为外联函数。下面给出一个内联函数与外联函数的代码:

```
class NUMBER
{
    private:
        int x;
    public:
        void ADD();
        void SUB(){x=x-1;}
}
NUMBER::ADD()
{
    x=x+1;
}
```

在上面代码中,声明了一个类 NUMBER,该类中有一个成员变量 x 与两个成员函数 ADD()与 SUB()。其中函数 ADD 在类中声明,但是其实现代码在类声明之外,因而,成员函数 ADD 是外联函数;成员函数 SUB 的声明与实现在类声明中,即 SUB 函数是定义在类声明之中的,因而是内联函数。

通过前面的例子可知,Visual C++ 6.0 的成员函数在默认情况下是外联函数。

2.内联函数与外联函数的区别

一般情况下,认为内联函数与外联函数的主要区别在于执行效率上。在C++语言中,外联函数的编译执行与普通函数编译执行是一致的:用户程序运行到外联函数时,需要保存当前运行状态,然后转而执行外联函数,当函数执行完毕后,再恢复保存的状态继续执行。而对于内联函数而言,系统在内联函数开始处直接执行其代码,而不需要保存状态与恢复状态

等工作，因而内联函数的运行效率高于外联函数。

通常情况下，用户可以使用内联函数来代替C++语言中的带参数的宏声明。内联函数可以实现宏的功能，而在安全性与稳定性上要优于宏。

在C++语言中，如果用户需要将成员函数功能实现部分放在类声明之外，而又希望将函数声明为内联函数，方法非常简单，只需在相应函数前添加关键字 inline 即可。例如，在上面提到的代码中，用户可以将 ADD 函数以如下方式声明，即可以以内联的方式使用 ADD 函数：

```
inline ADD();
```

3.内联函数局限性

虽然内联函数效率较高，但是，用户不能将所有的成员函数都声明为内联函数。通常情况下，在C++语言中，内联函数都是与存取相关的函数或使用频率较高的一些函数。此外，内联函数的功能实现代码不能太长，如果内联函数的函数体体积过大，编译器会自动将内联函数按外联的方式进行编译，因而就达不到提高运行效率的目的。

7.3.2 成员函数重载

在介绍构造函数时，已经涉及了函数重载的内容，事实上，除了构造函数，在类中的所有成员函数（不包括析构函数）均可以实现重载。所谓成员函数重载，就是在一个类中定义了若干同名的成员函数，但是这些函数的参数类型、个数是不同的，系统根据调用成员函数时提供的参数类型与参数个数，自动地选择合适的成员函数执行。

通过函数重载功能，用户可以更加方便地使用函数，从而使得程序设计更加灵活，稳定性更强，程序的可读性也更高。在C++语言中，函数重载无须特殊说明，用户只要在定义两个同名的成员函数的同时，使这两个成员函数的参数表不同即可。编译器会自动地将这两个成员函数视为函数重载。

通常情况下，函数重载有两种主要模式：一种是参数表中，形式参数的个数不同；另一种是参数个数相同，而参数类型不同。但无论何种模式，当程序编译时，系统会根据调用参数时提供的实际参数的个数与类型来选择正确的成员函数。下面给出一个成员函数重载的例子。

【例 7-4】 定义一个类，要求可以比较数字的大小，并返回最大值。要求使用函数重载实现。

分析：比较数字大小并返回最大值，可以比较两个数字，也可以比较 3 个数字，可以比较整型数据、浮点型数据，也可以比较字符型数据。因此，用户可以使用函数重载来实现相应功能。这里以两个整数比较、3 个整数比较、两个浮点数比较、两个字符比较来实现成员函数重载。

步骤：参照【例 7-2】在 Visual C++ 6.0 中创建一个类 MAXDATA，在类中加入 4 个同名的成员函数 GetMaxNum，具体声明形式如下：

```
int GetMaxNum(int inA,int inB);
int GetMaxNum(int inA,int inB,int inC);
double GetMaxNum(double inA,double inB);
char GetMaxNum(char inA,char inB);
```

并在 MAXDATA.cpp 文件中完成函数功能设计。最后，需要添加一个 main.cpp 文件

来调用 MAXDATA 类。

其中,类声明代码 MAXDATA.cpp 如下:

```
class MAXDATA
{
    public:
        char GetMaxNum(char inA,char inB);
        double GetMaxNum(double inA,double inB);
        int GetMaxNum(int inA,int inB,int inC);
        int GetMaxNum(int inA,int inB);
        MAXDATA();
        virtual ~MAXDATA();
};
```

类成员函数功能代码 MAXDATA.cpp 如下:

```
#include "MAXDATA.h"
#include <iostream>
using namespace std;
MAXDATA::MAXDATA()
{
}
MAXDATA::~MAXDATA()
{
}
int MAXDATA::GetMaxNum(int inA,int inB)
{
    cout<<"成员函数:"<<"int MAXDATA::GetMaxNum(int inA,int inB)"<<endl;
    if(inA>inB)
        return inA;
    else
        return inB;
}
int MAXDATA::GetMaxNum(int inA,int inB,int inC)
{
    cout<<"成员函数:"<<"int MAXDATA::GetMaxNum(int inA,int inB,int
inC)"<<endl;
    if(inA>inB)
        if(inA>inC)
            return inA;
        else
            return inC;
    else
```

```
        if(inB>inC)
            return inB;
        else
            return inC;
    }
    double MAXDATA::GetMaxNum(double inA,double inB)
    {
        cout<<"成员函数:"<<"double MAXDATA::GetMaxNum(double inA,double
inB)"<<endl;
        if(inA>inB)
            return inA;
        else
            return inB;
    }
    char MAXDATA::GetMaxNum(char inA,char inB)
    {
        cout<<"成员函数:"<<"char MAXDATA::GetMaxNum(char inA,char inB)"<<endl;
        if(strcmp(inA,inB)> 0)
            return inA;
        else
            return inB;
    }
```

主控程序 main.cpp 代码如下：

```
#include <iostream>
#include "MAXDATA.h"
using namespace std;
int main(int argc,char** argv)
{
    MAXDATA maxData;
    int iRet;
    double dRet;
    char* cRet;
    iRet=maxData.GetMaxNum(34,56);
    cout<<iRet<<endl;
    iRet=maxData.GetMaxNum(23,45,67);
    cout<<iRet<<endl;
    dRet=maxData.GetMaxNum(34.5,-27.6);
    cout<<dRet<<endl;
    cRet=maxData.GetMaxNum("China","china");
    cout<<cRet<<endl;
```

```
    system("PAUSE");
    return 0;
}
```

程序运行结果如图 7-8 所示。

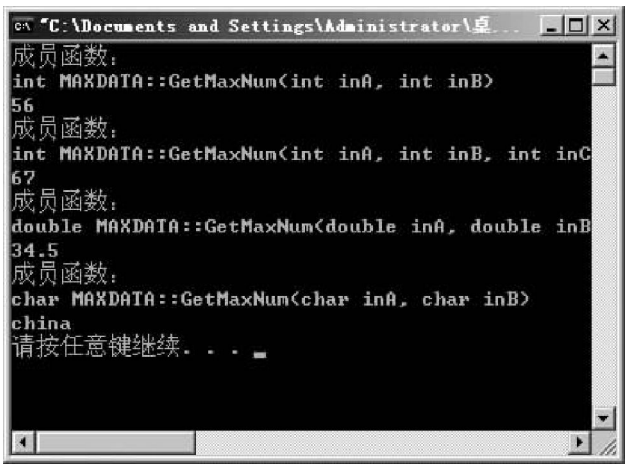

图 7-8　【例 7-4】程序运行结果

在【例 7-4】中的类 MAXDATA 中，没有类成员变量，因而类的构造函数与析构函数均为空函数。

7.3.3　成员函数缺省值

对于类的成员函数，用户在声明时，不仅可以指定参数，还可以为参数设定缺省值，而制定了缺省值的参数称默认参数。在C++语言中，指定了默认参数，其本质是为该参数设定了一个缺省的实际参数。当函数调用时，如果指定了实际参数，则按实际参数来调用函数，如果没有指定实际参数，则按默认参数来调用函数。例如，下面的函数声明：

```
void MYCLASS::initData(int inA=-1);
```

如果在调用函数时，使用下面的语句：

```
MYCLASS myClass;
myClass.initData(10);
```

则函数调用时，其实际参数为 10。如果按下面的语句来调用函数：

```
myClass.initData();
```

即用户在调用函数时，没有为其设定实际参数值，则系统将自动地以默认值-1 作为实参的值来调用函数。

在C++语言中，用户可以为成员函数中的多个参数设定缺省值，但是，用户在设定默认参数时应遵循下面的原则：

(1)若有多个默认参数，则默认参数之间必须是连续的。

(2)如果函数有默认参数，则默认参数必须居于参数表右方，也就是说，当成员函数中存在默认参数，则参数表中最右一个参数必须是默认参数。

下面给出几个函数默认参数的说明，首先是正确的默认参数形式：

```
void CLASSNAME::funcname(double dA,double dB=0.5,double dC=-1.7);
```

而错误的默认参数形式如下所示：

```
void CLASSNAME::funcname(double dA,double dB=0.5,double dC);
void CLASSNAME::funcname(double dA=-1.2,double dB,double dC=0.0);
```

对于错误的默认参数形式,其错误原因,用户可以自行判断。

默认参数虽然可以使得程序设计更加方便,但是也降低了程序的健壮性,一般情况下,不建议用户使用成员函数的缺省值形式。

7.4 静态成员与友元

在C++语言中,静态成员与友元是两个重要的概念。用户使用静态成员变量或静态成员函数可以达到“有限制”地使用全局变量或全局函数的效果;而通过使用友元类型,可以有条件地扩大类成员的使用范围。

7.4.1 静态成员

静态成员是C++类中的重要应用,使用静态成员可以使不同的类变量共享一个类成员,从而达到数据与函数共享的目的。

1.静态成员的作用

当用户在程序中声明完类结构后,可以使用类来定义类变量,一个类变量称该类的一个实例,也就是一个类对象。在程序中,用户每定义一个类变量,则相当于创建了一个类的副本,也就是说在内存中划分出一块空间来存放类中的成员数据。显然,不同的类变量需要在不同的内存空间中存放相应的成员数据,这些数据是完全独立的。

但是,在很多情况下,用户需要不同的类变量对同一个成员数据进行处理。例如,将键盘视做一个类,无论定义多少个类变量,键盘只有一个,所有的类变量都共享一个键盘单击事件返回的 ASCII 码值。针对这样的情况,用户可以将需要多个类变量共享的数据声明为静态数据,其方法是在类的成员变量或成员函数前,增加一个保留字 static。当类中的某个成员类型被声明为静态类型,则该类型数据只会在内存中存放一份,而所有该类的对象均共享这个数据。

通过上面的介绍,静态成员在使用时很像一个全局变量,但是,与真正的全局变量相比较,静态成员有其显著优点,主要表现在:

(1)限定了全局访问的范围,只有同一个类的类变量才能访问,其他类成员均不能访问。

(2)将全局访问的数据与操作集中在一起,提高了程序的可读性与可维护性。

2.静态成员变量

在一个类中,任何成员变量均可以声明为静态类型。当用户定义第一个类变量时,需要对该静态变量进行初始化操作。静态变量的初始化非常特殊,用户不能在静态变量声明时直接进行初始化赋值,例如,下面的语句:

```
static int a=10;
```

是错误的。通常情况下,用户需要在类的外部声明,在 Visual C++ 6.0 中就是在实现类功

能的cpp文件中加入赋值语句，该语句不作为任何成员函数内部的语句。其语法格式如下：

数据类型 类名称::静态变量名=初始值;

当该语句执行时，系统为静态变量划分空间，并完成赋值工作。

3.静态成员函数

与静态成员变量类似，用户也可以将成员函数声明为静态成员函数。静态成员函数的功能与静态成员变量功能相似，用户可以将其近似地看做全局函数，同一个类的所有类实例均共享该函数。用户可以通过下面的方式在类声明中说明一个静态成员函数：

static 函数类型 函数名(参数表);

在C++语言中，静态成员函数的特点如下：

(1)静态成员函数是在类中定义的，而与类的具体实例无关，类的所有实例共享静态成员函数。

(2)静态成员函数不能与非静态成员函数同名且同参数。

(3)静态成员函数主要用于访问静态成员数据，不能访问非静态成员数据。

(4)在C++语言中，用户无须定义类实例，即可以使用静态成员函数。

下面，给出一个在类中使用静态数据与静态函数的例子。

【例7-5】 定义一个学生类，统计学生类的类实例个数。

分析：本例要求统计类实例的个数，也就是在程序中定义了多少类变量。因此，用户需要定义一个静态变量，当定义一个类实例时，将该变量自加1；同时，用户还需要定义一个静态函数，用以获得当前静态变量的取值。

步骤：如【例7-2】所示，在空工程中添加一个类CSTUDENT。用户为该类添加一个静态成员变量和一个静态成员函数，然后在工程中添加一个main.cpp文件来调用CSTUDENT类。

其中，类声明的代码在STUDENT.h中，具体代码如下：

```
class CSTUDENT
{
    public:
        static int GetStuCount();
        CSTUDENT();
        virtual ~CSTUDENT();
    private:
        static int iStuCount;
};
```

可以在STUDENT.cpp中加入成员函数实现代码，同时，应在这里加入对类中静态成员变量的初始化代码。具体代码如下：

```
#include "STUDENT.h"
int CSTUDENT::iStuCount=0;
CSTUDENT::CSTUDENT()
{
    iStuCount++;
}
```

```
CSTUDENT::~CSTUDENT()
{
}
int CSTUDENT::GetStuCount()
{
    return iStuCount;
}
```

最后,在添加的 main.cpp 文件中加入下面的代码,完成类的调用工作。在代码中,用户每定义一个类变量,均执行静态函数,求取当前静态成员变量值。

```
#include <iostream>
#include "STUDENT.h"
using namespace std;
int main(int argc,char** argv)
{
    CSTUDENT stuA;
    cout<<"当前有学生:"<<stuA.GetStuCount()<<endl;
    CSTUDENT  stuB;
    cout<<"当前有学生:"<<stuB.GetStuCount()<<endl;
    CSTUDENT  stuC;
    cout<<"当前有学生:"<<stuC.GetStuCount()<<endl;
    CSTUDENT  stuD;
    cout<<"当前有学生:"<<stuD.GetStuCount()<<endl;
    system("PAUSE");
    return 0;
}
```

程序运行结果如图 7-9 所示。

图 7-9 【例 7-5】程序运行结果

7.4.2　友元

友元是C++程序设计中的重要部分，通过友元，用户可以将面向对象程序设计与面向过程的程序设计结合起来。虽然友元破坏了程序的封装性与数据隐蔽性，但是，合理使用友元可以大大提升程序设计的效率。友元可以分为友元函数、友元成员与友元类。

1.友元函数

在面向对象程序设计中，只有类中的成员函数才可以访问在类中定义的数据。如果非本类的成员函数要访问类中数据，必须将其声明为友元函数。友元函数是在类中声明的，可以访问类中成员数据的非成员函数。换句话说，如果用户需要在类中调用一个外部函数，并需要该函数访问类中的成员数据，可以将那个函数声明为类的友元函数。用户可以使用下面的方式来声明一个友元函数：

```
friend 函数类型 函数名(参数表);
```

通常情况下，友元函数具有以下几个特点：

(1)友元函数是外界函数，不是任何类的成员函数。

(2)用户在一个类中加入友元函数，则该友元函数可以访问这个类的成员数据，但是，在访问成员数据时，必须使用“类实例名＋成员数据名”的形式。

(3)友元函数可以直接在类中定义，也可以在类中声明，在类外定义实现。在类外定义友元函数时，不需要 friend 关键字。

2.友元成员

友元成员是友元的另一种类型，其作用与友元函数非常类似。如果用户需要在类 A 中调用类 B 的某个成员函数，则需要在 A 中将希望调用的 B 中函数声明为友元成员。友元成员与友元函数的最大区别在于：友元函数不属于任何类；而友元成员是某个类中的成员函数。用户可以使用下面的方式来声明一个友元成员：

```
friend 函数类型 类名称::成员函数(参数表);
```

3.友元类

友元的另一种形式是友元类。友元类与友元成员很相似，当用户希望在类 B 中的所有成员函数都可以访问类 A 中的成员数据时，可以将整个类 B 声明为类 A 的友元类。用户可以使用下面的方式来声明一个友元类：

```
friend class 类名;
```

综上所述，友元是提高程序运行效率的一种手段。用户通过在类中声明友元，可以达到非类成员函数访问类中成员数据的目的。如果声明的函数不属于任何类，则称友元函数；如果声明的函数是某个类的成员函数，则称友元成员；最后，用户可以声明友元类，从而可以使用另一个类的所有成员函数来访问本类的成员数据。通常情况下，在不是非常追求效率的环境中，不建议使用友元。

下面，给出一个使用友元函数的程序实例。

【例 7-6】 定义一个学生成绩类，要求记录学生姓名，语文、数学与外语成绩，并利用友元函数计算该学生的总分与平均分。

分析：按【例 7-2】所示步骤建立一个 CSTUSCORE 类，在类中加入成员数据：姓名、3 门

课成绩,并加入两个友元函数,分别计算总分与平均分。

步骤:创建类 CSTUSCORE,添加成员数据与友元函数。类声明在 STUSCORE.h 中,类实现代码在 STUSCORE.cpp 中,用户添加 main.cpp 文件,添加代码调用 CSTUSCORE 类。

其中类声明代码 STUSCORE.h 如下:

```
class CSTUSCORE
{
public:
    CSTUSCORE(char * cInName,int iInChinese,int iInMath,int iInEnglish);
    virtual ~CSTUSCORE();
    friend int GetTotalScore(CSTUSCORE cs);
    friend double GetAvgScore(CSTUSCORE cs);
    friend void ListStuScore(CSTUSCORE cs);
private:
    int mathScore;
    int chineseScore;
    int englishScore;
    char name[16];
};
```

在函数实现文件中(STUSCORE.cpp),用户应加入如下代码:

```
#include "STUSCORE.h"
#include <iostream>
using namespace std;
CSTUSCORE::CSTUSCORE(char * cInName,int iInChinese,int iInMath,int iInEnglish)
{
    strcpy(name,cInName);
    chineseScore=iInChinese;
    mathScore=iInMath;
    englishScore=iInEnglish;
}
CSTUSCORE::~CSTUSCORE()
{
    memset(name,0,sizeof(name));
    chineseScore=-1;
    mathScore=-1;
    englishScore=-1;
}
int GetTotalScore(CSTUSCORE cs)
{
    return cs.chineseScore+cs.mathScore+cs.englishScore;
```

```
}
double GetAvgScore(CSTUSCORE cs)
{
    return(cs.chineseScore+cs.mathScore+cs.englishScore)/ 3.0;
}
void ListStuScore(CSTUSCORE cs)
{
    cout<<"姓名:"<<cs.name<<endl;
    cout<<"语文成绩:"<<cs.chineseScore<<endl;
    cout<<"数学成绩:"<<cs.mathScore<<endl;
    cout<<"英语成绩:"<<cs.englishScore<<endl;
}
```

最后,用户在添加的 main.cpp 文件中,加入如下代码:

```
#include <iostream>
#include "STUSCORE.h"
using namespace std;
int main(int argc,char** argv)
{
    CSTUSCORE stuscore1("Mary",89,90,78);
    ListStuScore(stuscore1);
    cout<<"总分:"<<GetTotalScore(stuscore1)<<endl;
    cout<<"平均分:"<<GetAvgScore(stuscore1)<<endl;
    system("PAUSE");
    return 0;
}
```

程序运行结果如图 7-10 所示。

图 7-10　【例 7-6】程序运行结果

7.5 对象指针与对象应用

在C++语言中,无论是类中的成员数据、成员函数还是类本身都可以使用指针表示。类成员数据、成员函数的指针表示与传统的利用指针表示数据以及函数的方式相似。这里主要介绍利用指针与引用来处理对象。

7.5.1 对象指针

利用指针来表示类以及类的成员是非常简单的事情,其语法结构与利用指针表示普通数据与函数的方式十分相似。用户首先声明一个类指针,并将类的一个实例(对象)的地址赋值给类指针即可。假设有一个类 MYCLASS,如下所示:

```
class MYCLASS
{
    public:
        int myfunc(int b){return a * b+c;}
        MYCLASS(int in){a=in;}
        ~MYCLASS();
    private:
        int a;
        int c;
}
```

用户可以使用下面的指针方式表示类中的成员数据:

```
数据类型 类名称::*指针名称;
```

例如,定义一个指向 MYCLASS 类中的一个整型变量的指针,可以使用下面的语句:

```
int MYCLASS::*pa;
```

如果需要为该指针赋值,将该类的一个实例 A 中的成员变量的地址传递给指针,可以使用下面的语句:

```
pa=A.c;
```

这里,特别强调一点,A 是类 MYCLASS 的一个实例。也就是说,指针指向的成员数据一定是类实例的成员数据。

同样,用户可以使用相似的方式定义指向类成员函数的指针,其语法格式如下:

```
函数返回值(类名称::*函数指针名)(参数表);
```

如用户需要定义一个指针指向 MYCLASS 类中的 myfunc 函数,可以使用下面的语句:

```
int(MYCLASS::*pmyfunc)(int)=A::myfunc;
```

同样,A 是类 MYCLASS 的一个实例。

最后,介绍一下利用指针来表示一个类实例的方法。事实上,这才是程序设计中最常见的一种利用指针表示类的方式,语法格式如下:

```
类名称  *类指针名;
```

例如,下面的语句:

```
MYCLASS *pmyclass=&A;
```

即将 MYCLASS 类的一个实例 A 的地址传递给类指针 pmyclass。

7.5.2　对象引用

在C++语言中，对象引用与对象指针的功能基本相似，通常情况下，用户使用指针或引用方式来表示对象的目的是将对象地址作为函数的参数进行函数调用。这样做的优点主要有以下两个方面：

(1)实现按地址调用函数，函数可以修改类中的成员。

(2)传递指针而不是类实例，可以大大提高程序运行效率。

在程序设计中，由于引用方式更为简洁，所以得到了更广泛的应用。利用类的引用方式来调用函数，其语法格式如下：

```
返回值=函数名称(类名称 & 类实例);
```

在具体使用时，可以参考按地址调用函数的内容。

7.5.3　this 指针

在C++语言中，用户可以使用指针来表示类和类中的成员，其中，有一个指针十分特殊，这个指针就是 this 指针。

this 指针是C++系统中内部定义的一个指针，它隐含在类中的每个成员函数中，用以表示当前正在执行该成员函数的类实例，也就是当前使用该成员函数处理数据的对象。

当对一个对象调用成员函数时，编译程序先将对象的地址赋给 this 指针，然后调用成员函数，每次成员函数存取数据成员时，通常不显式地使用 this 指针来引用数据成员。当然，用户同样也可以使用 *this 来标识调用该成员函数的对象。

习　题　7

一、选择题

1. 关于类定义格式的描述中，(　　)是错误的。

A. 一般类的定义格式分为说明部分和实现部分

B. 一般类中包含有数据成员和成员函数

C. 类中成员有 3 种访问权限：公有、私有和保护

D. 成员函数都应是公有的，数据成员都应是私有的

2. 下列关键字中，(　　)不是类中定义数据成员使用的关键字。

A. static　　B. float　　C. extern　　D. double

3. (　　)不是构造函数的特征。

A. 构造函数的函数名与类名相同

B. 构造函数可以重载

C. 构造函数可以设置缺省参数

D. 构造函数必须指定类型说明

4.(　　)是析构函数的特征。

A. 一个类中只能定义一个析构函数

B. 析构函数名与类名不同

C. 析构函数的定义只能在类体内

D. 析构函数可以有一个或多个参数

5. 在(　　)情况下适宜采用 inline 定义内联函数。

A. 函数体含有循环语句　　B. 函数体含有递归语句

C. 函数代码少,频繁调用　　D. 函数代码多,不常调用

二、程序设计题

创建一个 employee 类,该类中有字符数组,表示姓名、街道地址、市、省和邮政编码。把表示构造函数、changename()、display()的函数的原型放在类定义中,构造函数初始化每个成员,display()函数把完整的对象数据打印出来。其中的数据成员是保护的,函数是公共的。