CHAPTER 3

第3章 数据与运算

3.1 引言

通过第2章的学习，读者已经能够编写简单的C++程序了，但是仍然有很多没学的内容、很多模糊的东西，当想更自由地编写程序或要编写的程序更加复杂、程序源代码更长时可能会遇到以下问题：没有很好的结构支持编写较大、较复杂的程序；自己取的变量名有时会有错；明明与书上的源程序相同，但是程序运行有错；有的程序对较小的数据运行正确，但对大一些的数据会计算出错误的结果，等等。

以下几个程序例子就反映了这些问题。

例B3.1 输入5个正整数，求它们的最大公约数。

```
/*************************************************************
*     MB3_1.cpp
*     输入5个正整数,求它们的最大公约数(程序结构臃肿,代码重复多)
*************************************************************/
#include <iostream>
using namespace std;
int main( )
{    int a,b,c,d,e,r;
     cin >> a >> b >> c >> d >> e;
     while(b!= 0)//求a、b的最大公约数,记录在a中
     {   r = a % b; a = b; b = r; }
     while(d!= 0)//求c、d的最大公约数,记录在c中
     {   r = c % d; c = d; d = r; }
     while(c!= 0) //求a、c的最大公约数,即原来a、b、c、d的最大公约数,记录在a中
     {   r = a % c; a = c; c = r; }
     while(e!= 0)//求a、e的最大公约数,即输入数的最大公约数,记录在a中
     { r = a % e; a = e; e = r; }
     cout <<"5个数的最大公约数为: "<< a << endl;
     return 0;
}
```

运行结果为：

```
12 27 63 33 75 ↵
5 个数的最大公约数为: 3
```

例B3.2　计算算术表达式“123×(456－356)＋73”(例A2.2)。

```
/*****************************************************************
*     MB3_2.cpp  错误程序
*     return 0 后面的分号是汉字字符
*****************************************************************/
#include <iostream>
using namespace std;
#include <iostream>
using namespace std;
int main( )
{
    cout << 123 * (456 - 356) + 73 << endl;
    return 0;
}
```

例B3.3　输入一个学生的数学、英语和 C++的成绩并输出平均成绩。

```
/*****************************************************************
*     MB3_3.cpp  错误程序
*     C++不能做变量名
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   int math,english,C++; double average;
    cin >> math >> english >> C++;
    average = (math + english + C++)/3.0;
    cout <<"平均成绩为: "<< average << endl;
    return 0;
}
```

例B3.4　输入非零数 n，计算阶乘 n!。

```
/*****************************************************************
*     MB3_4.cpp  运行有限制
*     计算的阶乘最大到 12!,再大的阶乘计算结果就是错误的
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   int f = 1,n,i;
    cin >> n; cout << n <<"!= ";
    for(i = 1;i <= n;i++) { f = f * i; }
    cout << f << endl;
    return 0;
}
```

运行结果为:

```
12↵
12!= 479001600
```

或

```
17↵
17!= -288522240
```

当输入为12时结果正确,当输入为17时输出的阶乘结果竟然是负数。

对于第1个问题,从第4章开始会将C++的语法进行详细讲解,而后面的内容非常丰富,完全可以支持用户编写更加复杂的程序。如果学过第6章"函数"再重新设计例B3.1,可如下编写。

```
#include <iostream>
using namespace std;
int gcd(int x,int y)     //函数 gcd 用于求 x 和 y 的最大公约数,函数值为 int 型数据
{    int r;
     while(y!=0) { r=x%y; x=y; y=r; }
     return x;
}
int main( )
{    int a,b,c,d,e,y;
     cin>>a>>b>>c>>d>>e;
     y= gcd(gcd(a,b),gcd(c,d)); y=gcd(y,e);     //调用函数 gcd 计算最大公约数
     cout<<"5 个数的最大公约数为: "<<y<<endl;
     return 0;
}
```

该程序使用函数(详见6.1节)减少了很多重复性的代码,而且逻辑结构清晰、简明。

对于第2~4个问题,则是与C++字符集、标识符和数据属性等C++语法有关,这也就是本章所要讲解的内容。

3.2 字符集与标识符

C++语言的字符是C++程序的最小单位,要构成合法的C++程序并不是什么字符都可以使用的,C++程序中可以出现的字符是有限制的,只能出现如表3-1所示的字符。

表3-1 C++字符集

分 类	字 符
26个小写字母	a、b、c、d、e、f、g、h、i、j、k、l、m、n、o、p、q、r、s、t、u、v、w、x、y、z
26个大写字母	A、B、C、D、E、F、G、H、I、J、K、L、M、N、O、P、Q、R、S、T、U、V、W、X、Y、Z
10个数字	0、1、2、3、4、5、6、7、8、9
其他字符	+、-、*、/、=、,、.、_、:、;、?、\、"、'、~、|、!、#、%、&、()、[]、{}、^、<、>、空格

(1) C++标识符:C++中使用的名字,如变量名、类型名、函数名等。C++标识符必须由字母、数字及下画线"_"构成,并且首字符不能是数字。若首字符是数字,则基本上是

C++的数值常数。例如，a、book、myPen、table2 等是标识符，而 12、032、0x7f、3.14f 等是数据常量。

(2) C++关键字：C++预定义的名字，在 C++程序的构成中具有特殊的含义。例如，if 表示条件的判断，int 表示数据的类型是整数类型等。C++关键字见表 3-2。

表 3-2　C++关键字

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

(3) C++标识符的设计：在程序中用户自己定义的标识符是可以任意取名的，但是不加限制地使用名字也会使程序阅读起来晦涩难懂，甚至使人对程序的功能发生误解。例如在一个要计算最大值和最小值的程序中，如果用标识符 minimum 命名的变量保存所求的最大值，用 maximum 命名的变量保存最小值将会严重误导阅读程序的用户，从而使程序的调试和维护难度与成本大幅度增加，尤其是设计大的程序时更是如此。因此，使用有实际含义的标识符就显得非常有必要了。另一方面，除了希望所取标识符能表达出实际含义外，有时候还需要知道标识符在程序中的与程序有关的一些属性信息，例如是类型名称、变量名还是常量名，变量是何种类型的变量等信息。

在命名标识符时，多个单词的组合名字可用骆驼法，即各个单词的首字母用大写，例如 myCar、sizeOfChar 表示 my car 和 size of char；也可用下画线间隔各个单词，例如 my_car、size_of_char。但是不能用空格间隔单词，否则就是多个标识符了。

为了在标识符中增加类型信息，可用匈牙利法，即用 n、d、p 等作为名字的前缀表示 int、double、指针类型。例如，nScore、dSalary、pIdentifier 分别表示 int 类型的存放成绩的变量、double 类型的存放工资的变量和指针类型的指向学号字符串的变量。

注 1：在 C++程序中，只有出现在双引号中的字符可以是汉字字符，程序其他部分不能出现汉字字符，尤其是标识符不能是汉字，分号、逗号等不能是汉字，特别要注意不能出现汉字空格。

3.3　基本数据类型

C++的数据类型很丰富，其分类如图 3-1 所示。

本章只讲解基本数据类型，对于复合数据类型将在第 5 章介绍，复合数据中的类是面向对象的数据类型，一直到第 10 章才会讲到。

在基本数据类型中整型和实型是数值类型，也就是人们通常所说的各种数。在计算机

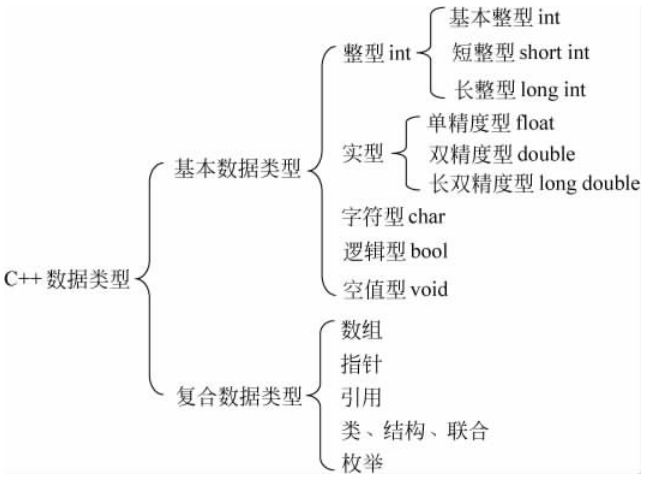

图 3-1　C++数据类型的分类

中整数和实数是两种存储和计算方式不同的数据,整数不带小数,实数带小数(小数部分可以为0)。整数的保存是精确的,实数的保存是非精确的,会有误差,但是实数表示的范围大得多。

由于数值型数据的计算非常重要,整型和实型还可以进一步分成各种不同类型的整型和实型。

整型可分成基本整型int、短整型short int和长整型long int。根据数据类型是否包含负数,整型还可分为有符号整型signed int和无符号整型unsigned int。修饰词short、long、signed、unsigned可以按不同的组合修饰int,从而得到整型的不同派生类型,具体见表3-3。

表 3-3　整数类型的属性

类　　型		占用的字节数	数的表示范围
基本类型(默认有符号)	int	2或4	$-2^{15}\sim2^{15}-1$ 或 $-2^{31}\sim2^{31}-1$
	short int	2	$-2^{15}\sim2^{15}-1$(−32768～32767)
	long int	4	$-2^{31}\sim2^{31}-1$(−2147483648～2147483647)
有符号类型	signed int	2或4	$-2^{15}\sim2^{15}-1$ 或 $-2^{31}\sim2^{31}-1$
	signed short int	2	$-2^{15}\sim2^{15}-1$
	signed long int	4	$-2^{31}\sim2^{31}-1$
无符号类型	unsigned int	2或4	$0\sim2^{16}-1$ 或 $0\sim2^{32}-1$
	unsigned short int	2	$0\sim2^{16}-1$(0～65535)
	unsigned long int	4	$0\sim2^{32}-1$(0～4294967295)

注1:现在流行的C++编译软件(包括VS2008),int、signed int和unsigned int都占4个字节。

注2:对于各种不同的int型,可以省略int,如short int可以简单地用short表示、long int可以用long表示、unsigned int可以用unsigned表示。

实型可分单精度型float、双精度型double和长双精度型long double(VS2008版编译环境下long double与double属性相同,都占8个字节),不同实型的属性见表3-4。

表 3-4　实数类型的属性

类　　型	占用的字节数	绝对值大小的范围	有效数字个数
float	4	$10^{-38}\sim10^{38}$	7
double	8	$10^{-308}\sim10^{308}$	15
long double	8 或 10	$10^{-308}\sim10^{308}$ 或 $10^{-4932}\sim10^{4932}$	15 或 19

字符型 char 占一个字节，用 8 位二进制数码表示一个字符，主要表示 ASCII 码表中的 128 个字符(其中 8 位二进制数码就是字符的 ASCII 编码，详见附录 B)，也可以表示扩展的字符(主要是一些制表的符号)或表示汉字字符的一部分(常用的 GB 2312 汉字字符需要两个字节才能表示一个汉字)。字符也可以看成是单字节的整数，所以有符号的选择，不同字符类型的属性见表 3-5。

表 3-5　字符类型的属性

类　　型	占用的字节数	数的表示范围
char(默认有符号)	1	$-2^7\sim2^7-1$(−128～127)
signed char	1	$-2^7\sim2^7-1$
unsigned char	1	$0\sim2^8-1$(0～255)

逻辑型 bool 占一个字节，表示条件是否满足，只有两个值，即 true 和 false。其中，true 表示条件满足，false 表示条件不满足。在将逻辑型数据当成一位整数看时 true 表示 1，false 表示 0。若输出逻辑型数据，如执行语句“cout << true << “,”<< false << endl;”，则输出“1,0”。

空值型 void 表示没有具体的数据类型，故无具体属性。void 型主要用于函数返回值类型，表示函数没有函数值(见 6.1 节的例A6.2)。void 也用于存储单元的地址(见 8.3.5 小节)，表示忽略存储单元中数据的类型。

注 3：字符型(char、signed char、unsigned char)在计算机内部就是一个单字节的整数类型，可以进行整数的各种运算，但在输入和输出时则是以数值(即 ASCII 码)所对应的字符的形式出现的。

注 4：逻辑型(bool)在计算机内部就是 1 位二进制数的整数类型，只含 0(表示 false)或者 1(表示 true)，但是占用一个字节，同样可以进行整数的各种运算。

变量是用于存放数据的存储单元，所存放的数据是可以根据需要随时改变的。不同类型的数据必须存放在不同类型的变量中。

只要是已存在的类型，不管是 C++中预定义的类型还是用户自己定义的复合数据类型，都可以定义相应的变量。变量的定义如下：

```
int  a,b;
float  c,d;
double  x,y,z;
```

变量在定义时可以进行初始化，即给变量赋初值。见下面的示例：

```
int  a=3,b=4;
float  c= -0.75,d=8;
double  x,y=3.14,z=355.0/113;
```

注5：变量在使用前必须先定义，而且变量只能定义一次，不能重复定义。

在定义变量时计算机需要分配相应的存储字节给变量，初始化就是在分配存储字节的同时在变量中给出值，否则分配的字节内是不确定的值。但有时变量已经存在了，就不需要再分配字节了(变量不能重复定义)，此时可以使用变量说明，表示该名称是一个已存在的某个类型变量的名称，可以使用。其形式如下(详见7.1.2小节)：

```
extern int u,v;
```

变量在进行了定义或说明之后就可以随便使用了。变量可以重复说明。

3.4 常量

3.4.1 字面常量

所谓字面常量，就是直接给出的数据，例如123、－56、1.414等。字面常量根据数据类型的不同有不同的写法。各种字面常量见表3-6。

表3-6 各种形式的常量

整型常量	十进制	11、－135、＋78、0、12875、10357632
	八进制(前缀为0)	012、057613、－07311
	十六进制(前缀为0x或0X)	0x12ff、－0x12AA、0xFFFF、0x7fffffb0、－0x2372、0X10
	长整型(后缀为L或小写l)	12L、011L、－23L、－0x27Al(后缀为小写字母l)
	无符号型(后缀为U或u)	100u、0u、65535u、1298u、20U、0x12abU
实型常量	单精度(后缀为F或f)	12.75f、2.f、－3.14e2f、3E－9f、－11.67F
	双精度	12.75、－314.、.178、＋3.2、1e－6、7.18e5、2.13E＋5、3E－2
字符常量	可显字符	‘a’、‘Y’、‘$’、‘@’、‘=’、‘8’、‘?’
	转义序列(预定义)	‘\n’、‘\t’、‘\r’、‘\a’、‘\b’、‘\\’、‘\’’
	转义序列(ASCII)	‘\12’、‘\103’、‘\xA’、‘\x43’、‘\x63’、‘\x38’、‘\xff’
字符串常量		“Hello”、“This is C++\n”、“\“Books\””、“a”、“”
逻辑常量		true、false

注1：双精度常量后面跟后缀f或F即为单精度常量，双精度常量的小数点前面或后面可以没有数字，但不能都没有数字。常量1e－6表示实数1×10^{-6}，称为指数形式常量。

注2：在字符常量中只能表示一个字符，若字符常量写成‘ab’、‘>=’则是错误的。

注3：在字符常量的预定义转义序列中，‘\n’表示回车换行，‘\t’表示跳格，‘\r’表示回车，‘\a’表示警报声，‘\b’表示回退一个字符，‘\\’表示反斜杠字符，‘\’’表示单引号字符。

注4：字符常量中的ASCII转义序列是用最多3位八进制或最多两位十六进制ASCII码值表示字符，如‘\103’表示ASCII码为$(103)_8$的字符，即‘C’，‘\x63’表示ASCII码为$(63)_{16}$的字符，即‘c’。在八进制表示的字符常量中不能出现非八进制的数字，如‘\83’就是错误的常量。

整型常量通常使用十进制常量，但有时也要用到其他进制的常量(八进制数和十六进制数详见附录B)，它能给编程带来很大的方便。见下面的程序例子。

例A3.5　用十六进制表示最大整数和最小整数。

```
/*************************************************************
*     MA3_5.cpp
*     输出最大和最小的整数（使用十六进制整型常量）
*************************************************************/
#include <iostream>
using namespace std;
int main( )
{    int maxInt = 0x7fffffff,minInt = 0x80000000 ;
     cout <<"最大整数 = "<< maxInt << endl;
     cout <<"最小整数 = "<< minInt << endl;
     //输出八进制数、十六进制数
     cout << oct <<"八进制输出：maxInt = "<< maxInt <<"\tminInt = "<< minInt << endl;
     cout << hex <<"十六进制输出：maxInt = "<< maxInt <<"\tminInt = "<< minInt << endl;
     cout << dec <<"恢复十进制输出：maxInt = "<< maxInt <<"\tminInt = "<< minInt << endl;
     return 0;
}
```

运行结果为：

```
最大整数 = 2147483647
最小整数 = -2147483648
八进制输出：maxInt = 17777777777   minInt = 20000000000
十六进制输出：maxInt = 7fffffff   minInt = 80000000
恢复十进制输出：maxInt = 2147483647     minInt = -2147483648
```

目前 C++ 中 int 型数占 4 个字节，最大数为 $2^{31}-1$，十六进制补码表示为 0x7fffffff，最小数为 -2^{31}，十六进制补码表示为 0x80000000。在用 cout 输出时，输出 dec 则控制之后输出的整数都是有符号十进制数，输出 oct 则控制为无符号八进制数，输出 hex 则控制为无符号十六进制数。故实际输出的十六进制数和八进制数是作为无符号数 $(80000000)_{16}$ 来输出的，于是八进制输出的数为 $(20000000000)_8$。

对于实数常量的使用，一般的做法是当数值不是很大或很小时用小数的书写方式；如果数值非常大或非常小，就用指数形式表示，即哪种形式看起来清楚就用哪种，见下面的程序例子。

例A3.6　求最小 n 使得 n!＞10^{14}（使用指数形式常量）。

```
/*************************************************************
*     MA3_6.cpp
*     求最小 n 使得 n!> 1e14（使用指数形式常量）
*************************************************************/
#include <iostream>
using namespace std;
int main( )
{    double t = 1 ;
     int n = 1;
     while( t <= 1e14 )
     {   n++; t = t * n; }
     cout <<"n = "<< n << endl;
```

```
    return 0;
}
```

运行结果为：

```
n = 17
```

如果程序需要对字符进行比较判断，一般使用可显字符常量，这样的程序可读性较好，见下面的程序。

例A3.7 输入一个字符，判断是否为字母(使用可显字符常量)。

```
/*****************************************************************
*     MA3_7.cpp
*     输入一个字符,判断是否为字母(使用可显字符常量)
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   char ch;
    cout <<"输入一个字符: ";
    cin >> ch;   //用"cin >> ch;" 无法输入空格、跳格字符
    if( (ch >= 'A'&&ch <= 'Z') || (ch >= 'a'&&ch <= 'z') )//字符的 ASCII 编码进行比较
    {  cout <<"输入字符为字母字符"<< endl; }
    else
    {  cout <<"输入字符为非字母字符"<< endl; }
    return 0;
}
```

运行结果为：

```
输入一个字符: 8↵
输入字符为非字母字符
```

注5：判断ch中的字符是否为大写字母用“ch >='A'&&ch <='Z'”，而不用“ch >=65&&ch <=90”，因为在计算机内部'A'表示大写字母A的ASCII编码65。

3.4.2 符号常量

所谓符号常量，就是给字面常量取一个名称，于是在使用该常量的地方就可以直接用该名称代替。例如可以用pi给常量3.141592653589793取名，这样编写的程序可读性比较好。

符号常量可如下定义：

```
const int maxInt = 0x7fffffff ;
const double pi = 3.141592653589793;
```

定义后，maxInt表示0x7fffffff，pi表示3.141592653589793。

例A3.8 输入一个角度度数(角度制)，输出相应的弧度数(使用符号常量)。

```
/*****************************************************************
*     MA3_8.cpp
```

```
*      输入一个角度度数(角度制),输出相应的弧度数(使用符号常量)
*************************************************************/
#include <iostream>
using namespace std;
const double pi = 3.141592653589793;      //定义符号常量 pi,通常在 main 前定义符号常量
int main( )
{    double degree,arc;
     cin >> degree;
     arc = degree * pi/180;                //使用符号常量 pi
     cout << degree <<"度的角度等于"<< arc << "弧度"<< endl;
     return 0;
}
```

运行结果为：

```
90 ↵
90 度的角度等于 1.5708 弧度
```

3.5　基本运算及表达式

C++的运算符非常丰富，除了大部分运算符是作用于两个操作数(二元运算符)以外，还有只作用于单个操作数的运算符(一元运算符)和作用于 3 个操作数的运算符(三元运算符)。运算符与操作数构成的式子称为表达式。

二元运算符有算术运算符＋、－、＊、/等，一元运算符有取负运算符－、＋＋、－－等，还有唯一的三元运算符?:，用于按条件(第一操作数)选择不同的操作数(第二、第三操作数)。当多个运算符组合在一起进行运算时需要考虑不同运算符的优先级。

常见的优先级关系有算术运算符的先乘除后加减。当相同优先级的运算合在一起时，如果没有括号，通常遵循从左往右的次序依次进行运算。但并不是所有的 C++运算符都是优先级相同就按先左后右的次序运算，也有先右后左的运算次序，通常将优先级相同的运算符的运算次序称为结合方向，大部分运算符的结合方向是从左往右，少数是从右往左(一元运算符、三元运算符和赋值运算符)。表 3-7 给出了 C++中所有运算符的优先级及其结合方向。

表 3-7　C++运算符的优先级和结合方向

优先级	运　算　符	结合方向
1	( )、.、->、[ ]、::、.*、->*、&(引用)	→
2	*(指向)、&(取地址)、new、delete、!、～、＋＋、－－、－(取负)、sizeof( )、显式转换	←
3	*、/、%	→
4	＋、－	→
5	<<、>>	→
6	<、<=、>、>=	→
7	==、!=	→
8	&(按位与)	→
9	^	→

续表

优先级	运　算　符	结合方向
10	|	→
11	&&	→
12	||	→
13	?:(三元运算符)	←
14	=、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=	←
15	,	→

3.5.1　算术运算、比较运算和逻辑运算

最常见的运算符有算术运算符、比较运算符和逻辑运算符。

算术运算符有+、-、*、/、% 。其中,+、-、*、/均可作用于整型、实型数据,而%只能作用于整型数据(包括字符型),结果为相除后的余数。

对于操作数都是整型的除法运算/,运算的结果还是整型,若数据除不尽,结果为整数部分,即商数,如27/8结果为3,-39/7结果为-5。

对于取模运算%,结果的符号与被除数相同或结果为0,结果的绝对值小于除数绝对值或为0。如27%8为3,27%(-8)也为3,-39%7结果为-4。

比较运算符有==、!=、<、<=、>、>=。比较运算的结果为逻辑值true或false。

逻辑运算符有 &&、|| 、!,表示与、或、非。逻辑运算符用于将多个简单的条件组合成复杂的复合条件。

例如,闰年的条件是年份是4的倍数但不是世纪年,或者是400倍数的世纪年。其C++实现如下:

```
(year % 4 == 0&&year % 100!= 0) || year % 400 == 0
```

对于表达式的值的数据类型来说,算术表达式的值的类型为操作数类型,如操作数类型不同,则按操作数的复杂性(int最简单,其次为float,最复杂为double)的不同,表达式值取最复杂的数值类型(详见3.6.4小节)。比较表达式和逻辑表达式的结果均为逻辑类型。

注1:逻辑表达式的操作数也要求是逻辑型,若是数值类型,则非零数会转换为逻辑量true,零会转换为逻辑量false。

3.5.2　增量/减量运算、赋值运算和逗号运算

C++的增量/减量运算符是++和--。如果n是一个变量,则不管n是整数类型还是实数类型,n++就是n=n+1,而n--就是n=n-1。该运算符还可以前置,如++n和--n,此处前置增量和减量与后置运算有相同的效果,但在某些情况下前置与后置有不同的效果(详见3.6.3小节)。

赋值运算符就是=,赋值表达式的值就是赋上新值的左边变量的值,即如果有赋值表达式y=3*x+1,则该表达式的值是赋值后的y的值。

注2:只有表示存储单元的表达式可以作为赋值运算的左操作数,这样的操作数称为左

值。所有不是左值的表达式称为右值。常量及计算值显然是右值。

赋值运算后的结果仍然是左值，即可有以下合法表达式：

$$(y = 3 + 5) = 7$$

注 3：用户可以对变量连续赋值，如“a＝b＝c＝1”表示 a、b、c 都赋上值 1，其本质是用赋值表达式给变量赋值，即“a＝(b＝(c＝1))”。

赋值号在某些特定情况下可以与右边的二元运算符构成以下复合的赋值运算符

+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=

表达式 x＋＝5 等价于表达式 x＝x＋5，表示右边表达式的值加到 x 上去。表达式 y＊＝3－4 等价于 y＝y＊(3－4)，表示右边表达式的值乘到 y 上去。

注 4：在 C++ 中，前置增量/减量运算、赋值运算和复合赋值运算的结果都是左值。

增量运算和复合赋值运算在程序设计中非常常见，见下列程序。

例C3.9　输入一批数据，输出数据的个数及平均值(增量与复合赋值)。

```
/*****************************************************************
*     MC3_9.cpp
*     输入一批数据,输出数据的个数及平均值(增量与复合赋值)
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   double x , sum ;int n ;
    sum = n = 0;                //连续赋值,0 连续赋值给 n 和 sum,等价于 sum = (n = 0)
    while(cin >> x)             //反复输入 x,直到单独一行输入^Z 终止输入
    {  n++; sum += x; }         //++与 += 的使用,等价于"n = n + 1;sum = sum + x;"
    cout <<"共"<< n <<"个数据,平均值为"<< sum/n << endl;
    return 0;
}
```

运行结果为：

```
12   -57.8   8.5↵
6   -102   37.88↵
5.5   20↵
^Z↵
共 8 个数据,平均值为 -8.74
```

上述程序反复输入数据直到单独一行输入文件结束符^Z，输入的循环才结束，然后输出统计的结果和平均值。

注 5：^Z 表示文件结束符，可在键盘上按住 Ctrl 键后再按 Z 键输入该结束符。

逗号运算符就是逗号“,”，表示先计算逗号左边的表达式，再计算逗号右边的表达式，将右边表达式的值作为逗号表达式的值。下列计算第 20 项斐波那契数的程序片段就使用了逗号运算。

```
int F1,F2,Fn,i;
for(F1 = F2 = Fn = 1,i = 3;i <= 20;F1 = F2,F2 = Fn,i++) { Fn = F1 + F2; }
```

其中,for 的第一个式子就是一个逗号表达式,由连续赋值、单个赋值和逗号运算符组成,第3个式子也是一个逗号表达式,由两个逗号运算符组成。执行逗号表达式就是依次执行逗号间隔开的各个表达式。例如第3个式子的执行为依次执行 F1=F2、F2=Fn、i++。

3.5.3 字符数据的处理

在 C++中字符型变量存放一个字符,实际上是在变量中存放该字符的 ASCII 编码,即一个整数值。字符类型本质上就是单字节的整数类型。

字符类型作为单字节整数类型可参与算术运算和比较运算,但字符类型的运算更多的是加减运算和比较运算,因为这些运算与字符的操作有关系。字符类型可以参与这些运算,给用户使用字符带来了方便。

在判断变量 ch 中的字符是否为某一类的字符时需要用到字符的比较,如判断 ch 的字符是否为小写字母可用"ch >= 'a'&&ch <= 'z'"。

当要改变 ch 中的字符时则要使用字符的算术运算。若 ch 中的字符是'b',则执行"ch=ch+2;"后 ch 中的值由原来'b'的 ASCII 码值 98(见附录 B)变为 100,即'b'后面第二个字符'd'的 ASCII 码,从而变成了'd'。由于小写字母的 ASCII 码比相应大写字母的 ASCII 码大 32,故 ch 中的字符如果是'b',则执行"ch=ch-32;"后 ch 的值比'b'的 ASCII 码 98 小 32,即'B'的 ASCII 码 66,这样就将 ch 中的字符变成了大写字母。

下列程序使用了字符比较和字符算术运算。

例C3.10 输入一行字符,将其中的小写字母转换成大写字母(字符算术运算)。

```
/*****************************************************************
*     MC3_10.cpp
*     输入一行字符,将其中的小写字母转换成大写字母(字符算术运算)
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   char ch;
    while( (ch=cin.get( ))!= '\n')   //反复从键盘输入字符,直到回车('\n')
    {  if( ch>= 'a'&&ch<= 'z' )      //判断 ch 为小写字母
        {  ch= ch+('A'-'a'); }       //字符加上大写到小写的差值即转为大写
        cout<<ch;
    }
    cout<<endl;
    return 0;
}
```

运行结果为:

```
this is C++↵
THIS IS C++
```

在上述程序中,"while((ch=cin.get( ))!= '\n')"表示表达式"(ch=cin.get( ))!='\n'"为 true 时反复执行循环体,其中,"ch=cin.get( )"表示用 cin.get( )从键盘输入一个字符赋值给 ch。故作用是反复从键盘输入字符,当输入回车时循环终止。

另外，程序中的字符进行比较和算术运算就是整数进行比较和算术运算，但在输入和输出时则是字符。

注 6：从键盘输入时是成批地输入数据直到回车，输入的数据暂时保存在内存的某个地方（即输入缓冲区），回车后才利用 cin 或 cin.get( )反复读取暂存的数据。

注 7：输入字符也可用 cin >> ch 的形式，但是这样输入字符会忽略输入的空白间隔符（空格、Tab 键（跳格）和回车）。

3.6　对表达式的进一步说明

3.6.1　整数除和算术溢出

算术运算中容易用错的有整数除和算术溢出，下面来看一个程序。

例B3.11　演示整数除和算术溢出。

```
/*****************************************************************
*     MB3_11.cpp
*     演示整数除和算术溢出
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   //计算 1 + 1/2 + 1/3 + … + 1/10
    int i ; double s1 = 0,s2 = 0;
    for(i = 1; i <= 10; i++) { s1 += 1/i ; }              //1/i 为整数除,i > 1 时结果为 0
    for(i = 1; i <= 10; i++) { s2 += 1.0/i ; }            //1.0/i 为实数除,i > 1 时结果为小数
    cout <<"s1 = "<< s1 <<", s2 = "<< s2 << endl;
    //两个变量 i、j 的乘积超出 int 型数据范围,则超出 4 字节的高位部分丢失
    int j,m,k1 , k2 ;
    i = 0x30000; j = 0x20000; m = 0x10000;
    cout << hex <<"i = "<< i <<", j = "<< j <<", m = "<< m << endl;     //按十六进制输出 i、j、m
    k1 = i * j/m;  //等价于"k1 = (i * j)/m;",计算 i * j 时超出 int 数据范围(算术溢出),残留值 0
    k2 = i/m * j;  //i/m 值为 3,再计算 3 * j 得 0x60000,此时不会产生算术溢出
    cout <<"k1 = "<< k1 <<" , k2 = "<< k2 << endl;          //按十六进制输出 k1、k2
    return 0;
}
```

运行结果为：

```
s1 = 1, s2 = 2.92897
i = 30000, j = 20000, m = 10000
k1 = 0 , k2 = 60000
```

从程序代码来看，s1 和 s2 都是计算 $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{10}\approx 2.928968254$，k1 和 k2 都是计算 $\frac{(30000)_{16}\times(20000)_{16}}{(10000)_{16}}=(60000)_{16}$ 但是运行结果大不相同，只有 s2 和 k2 的结果是正确的。

求和计算不同的原因在于,在从1到10的循环中,s1加上1/i的值,由于1和i均为整数类型,故1/i为整数除,结果为整数,于是只有当i=1时1/i值为1,当i>1时1/i值为0,故s1的结果为1。而s2每次加上的是1.0/i的值,由于1.0是双精度类型,故1.0/i为双精度类型,故得到了1/i的小数值,从而最终s2得到了正确的结果。

乘除式子的计算结果不同的原因则是k1在计算i*j/m时先计算i*j,其结果为$(600000000)_{16}$,已经超出了int型数据的表示范围——$(-80000000)_{16}$～$(7FFFFFFF)_{16}$,造成高位部分数据丢失,即算术溢出。溢出后,乘积只剩下低8位十六进制数$(00000000)_{16}$,即0,再除以$(10000)_{16}$结果仍然为0。而k2的计算式是i/m*j,先计算i/m,结果为3,再计算乘法(即3*$(20000)_{16}$)得到结果$(60000)_{16}$,该乘法没有产生算术溢出,故得到了正确的结果。

3.6.2 比较运算的特殊用法

对于比较运算的设计,用户要特别注意对实数是否相等的判断及连续不等式的C++实现。下面看一个程序运行的情况。

例B3.12 演示实数相等比较和连续不等式的表示。

```
/*****************************************************************
*     MB3_12.cpp
*     演示实数相等比较和连续不等式的表示
*****************************************************************/
#include <iostream>
#include <cmath>
using namespace std;
const double eps = 1e-7;                              //定义误差常量 eps
int main( )
{   //误差影响实数的相等比较
    double x,y;
    x = 3.0 * 0.1 ; y = 0.3 ;
    cout <<"x = "<< x <<"\ty = "<< y << endl;
    if(x == y) cout <<"(x == y)结果为 true"<< endl;    //if 后面若是单个语句可省略花括号
    else  cout <<"(x == y)结果为 false"<< endl;        //else 后面若是单个语句也可省略花括号
    if(fabs(x - y)< eps) cout <<"|x - y|< 1e - 7 结果为 true"<< endl;
    else  cout <<"|x - y|< 1e - 7 结果为 false"<< endl;
    //连续不等式的数学式与 C++表示式不同
    int m = 20;
    cout <<"m = "<< m << endl;
    if(0 <= m <= 10) cout <<"(0 <= m <= 10) 结果为 true"<< endl;
    else  cout <<"(0 <= m <= 10) 结果为 false"<< endl;
    if(0 <= m&&m <= 10) cout <<"(0 <= m&&m <= 10) 结果为 true"<< endl;
    else  cout <<"(0 <= m&&m <= 10) 结果为 false"<< endl;
    return 0;
}
```

运行结果为:

```
x = 0.3   y = 0.3
(x == y) 结果为 false
```

```
|x-y|<1e-7 结果为 true
m=20
(0<=m<=10) 结果为 true
(0<=m&&m<=10) 结果为 false
```

从运行结果来看，双精度变量 x 和 y 保存的都是 0.3，但是相等比较的结果确实是 false。这是怎么回事呢？

其实，在计算机内部变量 x 和 y 保存的是不同的数据，但是由于输出时的精度控制，结果都是输出了 0.3。当然，x 和 y 的数值虽然不同，但是相差很小，可以分析得到 x≈0.30000000000000004441、y≈0.29999999999999998890，两个变量中的数据是不同的，相差 5.551×10^{-17}。当 y 中直接保存 0.3 时，实际保存的数据其实是比 0.3 小一点点的数据而不是 0.3 的精确值。

产生上述结果的原因在于，实数类型在计算机中是不精确的，甚至在保存数据时就产生了误差，并且在进行各种算术运算时也会产生计算误差，因此进行比较时的实数都会带有累积误差，将误差的干扰考虑在内，实数的相等比较就要看两个数是否相差在误差范围内了，如果是就认为除去误差后两个数是相等的。因此，判断实数相等的式子应该是“fabs(x－y)<eps”，其中 eps 是一个很小的数，表示误差的界限。一般来说，相对误差通常取存储精度的开平方根，故 double 型实数的误差界限可取 10^{-7}、10^{-8}。本书不分绝对误差和相对误差(除非比较的数值都非常大)，误差界限一律取为 10^{-7}。

就上面的程序来讲，0.3 在计算机中保存时要转换成二进制的科学记数法，而该数的二进制数是一个无限小数，在实际存储时舍去了一部分小数，从而变量 y 实际保存的是比 0.3 略小的数值。0.1 在计算机中保存时舍入是进位的，乘以 3 后又一次舍入进位，故变量 y 实际得到的数值比 0.3 略大。

如果是两个整数比较，由于整数在计算机内是精确表示的，因此相等比较直接使用==进行比较，例如“a==b”。

进一步看后面连续不等式部分的运行结果。当 m 值为 20 时，“0<=m<=10”的结果为 true，而另一个表达式“0<=m&&m<=10”的结果却是 false。

从数学上说，当 m=20 时，不等式关系 0≤m≤10 是不成立的。但是在 C++中，表达式“0<=m<=10”却成立，结果为 true。原因是在 C++中如果按照结合方向加上括号，就有等价式“(0<=m)<=10”，而该式子的计算为先进行“0<=m”的计算，如果 m 的值是 20，则“0<=m”的结果为 true，再计算式子“true<=10”，true 与数值进行比较时相当于整数 1，即比较 1<=10，于是结果为 true。因此 C++的表达式“0<=m<=10”与数学关系式 0≤m≤10 的含义是不同的。

数学关系式 0≤m≤10 表示两个不等式 0≤m 和 m≤10 都满足，作为 C++的复合条件，表达式应该是“0<=m && m<=10”。上面程序的结果验证了这一点。

注 1：一般来讲，编程时对于>、>=、<、<=等大小比较不去考虑误差，对于经过一系列计算的实数值要进行相等比较时才会考虑误差，并使用 fabs(x－y)< eps 之类的式子进行比较。

注 2：两个 double 型实数进行相等比较时误差界限通常使用 10^{-7}。若两个比较的实数都相当大，需要比较相对误差界限，如 fabs((x－y)/x)< 1e－7。在使用 double 型的字面常

量时可以取存储精度,即取16位有效数字,比double型的有效位数多一位。

注3:在C++中,数学关系0≤m≤10要用"0 <= m&&m <= 10"表示,不能用"0 <= m <= 10"表示。

3.6.3 前置与后置的增量和减量

增量和减量运算是非常常用的一种运算,通常情况下单独作为一个语句或作为单独的公式,此时前置和后置运算没有区别。但当前置运算和后置运算作为更大的表达式的一部分时,两者的效果是不同的。首先看下面的程序,然后对运行结果进行说明。

例B3.13 演示前置与后置的增量和减量运算。

```
/*****************************************************************
*     MB3_13.cpp
*     演示前置与后置的增量和减量运算
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{
    int a , b ; double x;
    //独立的增量和减量运算
    a = 10; x = 3.14;cout <<"a = "<< a << "\t x = "<< x << endl;
     -- a; ++x;                  //-- 使变量值减1,++使变量值增1,不管对整数还是实数
    cout <<"after -- a; a = "<< a <<"\t after ++x; x = "<< x << endl;
    a = 10; x = 3.14;cout <<"a = "<< a << "\t x = "<< x << endl;
    a-- ; x++;                   //单独使用时,-- 、++的前置运算与后置运算效果一样
    cout <<"after a-- ; a = "<< a <<"\t after x++; x = "<< x << endl;
    //参与表达式的增量与减量前置和后置有不同效果
    a = 10; cout <<"begin a = "<< a << endl ;
    b = ++a;cout <<"after b = ++a; a = "<< a << " , b = "<< b << endl;
    a = 10; cout <<"begin a = "<< a << endl ;
    b = a++;cout <<"after b = a++; a = "<< a << " , b = "<< b << endl;
    return 0;
}
```

运行结果为:

```
a = 10 x = 3.14
after -- a; a = 9       after ++x; x = 4.14
a = 10 x = 3.14
after a-- ; a = 9       after x++; x = 4.14
begin a = 10
after b = ++a; a = 11 , b = 11    //b得到了a增1后的值(前置为先增1)
begin a = 10
after b = a++; a = 11 , b = 10    //b得到了a增1之前的值(后置为之后增1)
```

从运行结果来看,不管是整型变量还是实型变量,单独使用增量运算时就是使变量的值增加1,前置增量和后置增量的运算效果一样。减量与增量运算类似,只是效果为使变量的值减少1。

考虑前置增量表达式,如++a,表达式表示a增加1,然后取变量a。故++a是指增加

1 之后的变量 a，是左值。

而后置增量表达式，如 a++，表达式表示 a 的值先暂存到 CPU 中，a 再增加 1，然后取 CPU 中的值。所以 a++表示 a 之前保存的值，但 a 现在已经增加 1 了。故 a++是指 a 增加 1 之前的数值，是右值。

对于上述程序的运行，变量 a 的值开始为 10，然后执行语句“b=++a;”，此时 a 先增加 1 成为 11，然后 a 赋值给 b，于是 a 和 b 的值都是 11。当变量 a 的值为 10 时执行语句“b=a++;”，此时 a 的值增加 1 变成 11，但 a 之前的值 10 赋给变量 b。于是 a 值为 11，b 值为 10。

注 4：前置增量和后置增量，前置减量和后置减量，其操作数必须是左值。例如 a++，则 a 必须是左值。

注 5：前置增量/减量表达式为左值，后置增量/减量表达式为右值。如++a 为左值，a++为右值。

3.6.4　类型的转换

在表达式的计算中会涉及不同类型的数据，此时就会有不同类型数据的转换。下面从 3 个方面论述数据的类型转换，即显式转换、赋值转换和算术转换。

(1) 显式转换：直接用类型标识符进行的转换，也称强制类型转换，为一元运算。

若有定义：

```
int a = 3; double x = -3.84; char ch = '0';      //'0'的 ASCII 编码为 48
```

则各种显式转换见表 3-8。

表 3-8　不同类型转换时的形式与结果值

int(x)	(int)x	static_cast<int>(x)	取整数部分，值为−3
double(a)	(double)a	static_cast<double>(a)	变成 double 形式，值为 3.0
int(ch)	(int)ch	static_cast<int>(ch)	变成 int 形式，值为 48

注 6：在表 3-8 中，int(x)、(int)x、static_cast<int>(x)都表示将 x 的值显式转换为 int 类型的值，是 3 种不同但是等价的显式转换的写法，效果一样。

字符型就是单字节整型，逻辑型是只含一位二进制数的整型。当包括字符型、逻辑型的各种整型和各种实型数值进行转换时，其变化情况见表 3-9。

表 3-9　不同类型转换时数值的变化

表　达　式	转换成的类型	数 值 变 化
整型	等字节的整型	字节内二进制数不变
较长字节整型	较短字节整型	截取低位字节的二进制数
较短字节无符号整型	较长字节整型	高位字节进行 0 扩展的二进制数
较短字节有符号整型	较长字节整型	高位字节进行符号扩展的二进制数
double	float	舍入到更短的有效位，降低精确度
float	double	取原数值，保留原数的精确度
整型或实型	逻辑型	非零取 true，零取 false

续表

表 达 式	转换成的类型	数 值 变 化
逻辑型	整型或实型	true为1,false为0
实型	整型	截断小数,取整数部分
整型	实型	整数转成实数,数值不变

注7：符号扩展是指如果原来字节的最高位是0,则高位字节进行0扩展(高位全部补0)；如果原来字节的最高位是1,则高位字节进行1扩展(高位全部补1)。符号扩展保证正数扩展后仍为正数,负数扩展后仍是负数。

注8：整数类型转换成整数类型时,内部的二进制(分析时常用十六进制)数值的变化规律比较简单。整数转换成等长字节类型整数时,内部二进制数值不变；转换成较短字节类型整数时,取低位字节二进制数值；转换成较长字节类型整数时,内部二进制数值进行扩展。若原来是无符号类型整数,则进行无符号扩展(高位字节0扩展)；若原来是有符号类型整数,则进行符号扩展。

不同类型整型转换时数值变化举例如下。

若有定义：

```
int a = 0x0A00CD39;          unsigned long b = 0xFFFFFFFF;
unsigned char ch = '\xF0';   short int c = 0xFF00;
```

则数值变化情况见表3-10。

表3-10　不同类型整型转换时数值变化的例子

	原类型	原值	结果类型	结果值	表示值
(int)b	unsigned long	$(FFFFFFFF)_{16}$	int	$(FFFFFFFF)_{16}$	−1
(char)a	int	$(0A00CD39)_{16}$	char	$(39)_{16}$	'9'
(long int)ch	unsigned char	$(F0)_{16}$	long int	$(000000F0)_{16}$	$240=+(F0)_{16}$
(unsigned int)c	short int	$(FF00)_{16}$	unsigned int	$(FFFFFF00)_{16}$	$4294967040=+(FFFFFF00)_{16}$

注9：若数据是无符号类型,则表示值就是内部二进制数(或十六进制数)转换成十进制数,如unsigned int型的值$(FFFFFF00)_{16}=4294967040$；若数据是有符号类型,则表示值就是以内部十六进制数作为补码的数,如int型的值$(FFFFFFFF)_{16}$,最高位为1,表示负数,相反数的值为$(00000001)_{16}=1$,故表示值为−1。进制转换与补码见附录B。

(2) 赋值转换：如果赋值号两边类型相同,直接进行赋值；如果类型不同,则赋值号右边的值的类型转换成赋值号左边的类型,然后进行赋值。

(3) 算术转换：这是一种自动转换或隐式转换。不同类型数据参与算术运算,运算的操作数会先自动转换成相同类型的操作数,然后再进行运算。不同类型数据参与算术运算时的转换规则见图3-2。

注10：short int、bool、char类型数据参与算术运算时会首先自动转换成int,然后再参与运算；不同类型数据进行算术运算时按向下箭头方向首先自动转换成相同类型；long int和unsigned int混合运算时都自动转换成unsigned long int类型。

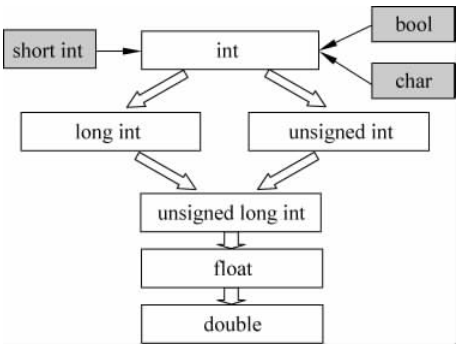

图 3-2　转换规则

若有定义：

```
int a = 3; long b = 4; float x = 4.0 f; double y = 5.2; char ch = 'a'; short int c = 10;
```

则各种混合表达式的类型转换情况见表 3-11。

表 3-11　几个混合类型算术表达式的自动类型转换

表　达　式	数 据 类 型	转换后的统一类型	表达式类型
a/x	int / float	float	float
b＋y	long ＋ double	double	double
c＊c	short ＊ short	int	int
ch－32	char －int	int	int
ch－'A'	char －char	int	int

关于各种类型的转换的演示可见下列程序的运行结果。

例B3. 14　演示转换规则。

```
/*************************************************************
*     MB3_14.cpp
*     演示转换规则：显式转换、赋值转换、算术转换
*************************************************************/
#include <iostream>
using namespace std;
void type(bool) { cout <<"bool"<< endl; }
void type(char) { cout <<"char"<< endl; }
void type(int) { cout <<"int"<< endl; }
void type(unsigned int) { cout <<"unsigned int"<< endl; }
void type(float) { cout <<"float"<< endl; }
void type(double) { cout <<"double"<< endl; }
int main( )
{
    //显式转换
    int a = 10,b = 20; float x = 1.0f; double y = 1.0;
    type((int)x);        //float 转换为 int
    type(char(x + y));   //double 转换为 char
    type(double(a < b)); //bool 转换为 double
```

```
    //赋值转换
    a = 3;
    x = a;                  //等价于"x = (float)a;",int 转换为 x 类型 float,x 得到转换后的值 3.0f
    b = x;                  //等价于"b = (int)x;",float 转换为 b 类型 int,b 得到转换后的值 3
    cout <<"a = "<< a << " , x = "<< x <<" , b = "<< b << endl;
    y = a = x =  - 3.1416f;//等价于"y = (a = (x = - 3.1416f));",第一次赋值 x 得到 - 3.1416f
                           //第二次赋值 x 的 float 型值转换为 a 类型 int 值 - 3,赋值给 a
                           //第三次赋值 a 的 int 型值转换为 y 类型 double 值 - 3.0,赋值给 y
    cout <<"x = "<< x <<" , a = "<< a <<" , y = "<< y << endl;
    //算术转换
    type( 'a' + 0);        //char + int 转换后 int + int,结果 int
    type( 'a' - 'A');      //char - char 转换后 int - int,结果 int
    type(true + true);     //bool + bool 转换后 int + int,结果 int
    type(3/7);             //int/int,类型一致,不转换,结果 int
    type(0x80000000u + 1234);     //unsigned + int 转换后 unsigned + unsigned,结果 unsigned
    type(1.0f/3);          //float/int 转换后 float/float,结果 float
    type(1.2f + 3 * 4.0); //float + (int * double): 乘法 int * double 转换成 double * double
                    //中间结果 double; 加法 float + double 转换成 double + double,结果 double
    return 0;
}
```

运行结果为:

```
int                             //显式转换成 int
char                            //显式转换成 char
double                          //显式转换成 double
a = 3 , x = 3 , b = 3           //"a = 3; x = a; b = x;"的结果
x = - 3.1416 , a = - 3 , y = - 3 //"y = a = x =  - 3.1416f;"的结果
int                             //char + int 的结果类型
int                             //char - char 的结果类型
int                             //bool + bool 的结果类型
int                             //int/int 的结果类型
unsigned int                    //unsigned + int 的结果类型
float                           //float/int 的结果类型
double                          //float + (int * double)的结果类型
```

首先对该例程序做一些说明。在程序开始处有一些行,如第一行:

```
void type(bool) { cout <<"bool"<< endl; }
```

表示函数 type。在执行 type(表达式)时,若表达式的值是 bool 类型,则输出字符串"bool"。再看后面的几行,圆括号内有 char、int、unsigned int、float、double,于是执行 type(表达式)时表达式的值是 char、int 或 unsigned int 等,会输出相应类型的字符串。

3.6.5 短路表达式

对于逻辑运算符构成的逻辑表达式在计算时要注意,逻辑表达式进行的计算并不是所有的式子都要进行计算,而是采用捷径方式,一旦有了最终的结果就不再计算下去,因此逻辑表达式也称为短路表达式。

在逻辑表达式中,短路表达式的作用可以用逻辑表达式"(a > 0) && (b++ > 0) &&

(——c>0)"加以说明,其中 a、b、c 为 int 型变量,该表达式可以用图 3-3 所示的流程框图说明。

在该逻辑表达式中,a>0、b++>0、——c>0 是 3 个逻辑型的量,表示 3 个条件,每一个条件满足逻辑量为 true,条件不满足逻辑量为 false。这 3 个逻辑量中只要有一个是 false,结果必然是 false。因此在计算判断过程中一旦发现有一个量为 false,表达式计算结束,否则一直计算判断下去,直到最终得到逻辑结果或 true 或 false。图 3-3 所示的流程框图显示了 C++内部对逻辑表达式的实际的实现方式。从该框图可以看出,一旦某一步的逻辑量为 false,则直接跳出表达式的计算过程,得到结果 false,否则,每个逻辑量都是 true 结果才为 true。下列程序演示了逻辑表达式的这一特性。

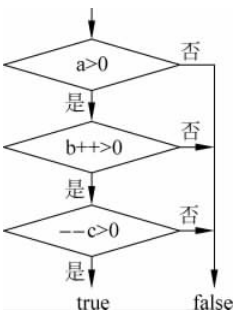

图 3-3　流程图

例B3.15　演示逻辑运算的短路表达式。

```
/*****************************************************************
*     MB3_15.cpp
*     演示逻辑运算的短路表达式
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{    int a , b , c ; bool k;                    //bool 类型变量 k 用于记录条件表达式的值
     a = 10 ; b = 20 ; c = 30 ;
     cout <<"a = "<< a << ",b = "<< b << ",c = "<< c << endl;
     k = (a > 0)&&(b++> 0)&&( -- c > 0) ;       //即"k = ((a > 0)&&(b++> 0))&&( -- c > 0) ;"
     cout <<"a = "<< a << ",b = "<< b << ",c = "<< c <<",k = "<< k << endl;
     k = (a < 0)&&(b++> 0)&&( -- c > 0) ;       //即"k = ((a < 0)&&(b++> 0)) &&( -- c > 0) ;"
     cout <<"a = "<< a << ",b = "<< b << ",c = "<< c <<",k = "<< k << endl;
     return 0;
}
```

运行结果为:

```
a = 10,b = 20,c = 30
a = 10,b = 21,c = 29,k = 1     //执行"k = ((a > 0)&&(b++> 0)); &&( -- c > 0) ;" 之后 k 为 true
a = 10,b = 21,c = 29,k = 0     //执行"k = ((a < 0)&&(b++> 0)); &&( -- c > 0) ;" 之后 k 为 false
```

上述程序在执行语句"k= ((a<0)&&(b++>0));"之后,由于第一个条件 a<0 不满足,故第一个逻辑量为 false,故整个表达式的结果为 false,计算终止。后面的 b++>0 和——c>0 都没来得及计算,特别是 b++和——c 没有执行,故 a、b、c 的值都不变。

注 11:当数值型量参与逻辑运算时,非 0 为 true,0 为 false,故数值型量可以构成逻辑表达式,并且同样会提前终止。如果"double a=0.01; int b=0,c=10;",则表达式"a&&b++&&——c"的结果为 false,a、c 值不变,b 值为 1,因为 b++值为 0,表示 false,计算终止,——c 没有运行。类似分析,表达式"!a&&b++&&——c"的结果为 false,a、b、c 值都不变。

3.7 位运算

在C++中可以对整数的二进制数的各个位进行操作,这就是位运算。

位运算有&(按位与)、|(按位或)、~(按位取反)、^(按位异或)、<<(左移)、>>(右移)。

位运算可分成两类,一类是二进制位的按位运算与、或、非和异或,即&、|、~、^,各个二进制位之间互不干扰;另一类是二进制位的移位运算,如左移和右移,即<<和>>。

若1表示true、0表示false,则逻辑运算规则见表3-12。

表3-12 逻辑运算规则

逻辑运算	运算规则	特点
与 ×	1×1=1　1×0=0 0×1=0　0×0=0	操作数都为1,结果才为1,否则为0
或 +	1+1=1　1+0=1 0+1=1　0+0=0	操作数都为0,结果才为0,否则为1
非 '	1'=0　0'=1	非1即0,非0即1
异或⊕	1⊕1=0　1⊕0=1 0⊕1=1　0⊕0=0	操作数不同,结果为1,否则为0,即不进位相加

在位运算中,&、|、~、^为按位进行逻辑运算。而<<和>>则是字节各位进行左移和右移,左移时右边移进0,右移时左边移进的数字是最高位的数字(0表示正数,1表示负数)或0(无符号类型)。为了简单起见,下面以单字节数据为例对位运算进行说明。

若有定义"signed char a=0xCF , b=0x6C ;",则各种位运算的结果见表3-13。

表3-13 位运算举例

a = $(11001111)_2$ b = $(01101100)_2$ a&b = $(01001100)_2$	a = $(11001111)_2$ b = $(01101100)_2$ a | b = $(11101111)_2$	a = $(11001111)_2$ b = $(01101100)_2$ a^b = $(10100011)_2$
a = $(11001111)_2$ ~a = $(00110000)_2$	a = $(11001111)_2$ a<<3 = $(01111000)_2$	a = $(11001111)_2$ a>>3 = $(11111001)_2$

这里说明一下"a >> 3"的结果。变量a是有符号的,故扩展是符号扩展,即最高位扩展,而a的值是$(11001111)_2$,最高位是1,于是扩展到高位的数字都是1,即左边移进的都是1。

注1:对于二进制数来说,左移一位相当于乘以2,右移一位相当于除以2(结果下取整)。变量在计算机内是以二进制数保存的,所以左移和右移相当于乘以或除以2或2的乘幂。于是不管是否有符号,左移运算时右边都应该补0。而右移运算时,无符号数表示正数,故左边移进0。如果是有符号数右移运算,则右移后正数应该仍是正数,负数仍是负数。故左边应该补最高位数字。

下列程序演示了位运算的结果。

例B3.16 演示位运算的结果。

```
/*****************************************************************
*     MB3_16.cpp
```

```
*     演示位运算的结果
*************************************************************/
#include <iostream>
#include <iomanip>                    //输出时用了格式控制 setfill('0')和 setw(2),需加上此行
using namespace std;
int main( )
{    short int a = 0xFF, b = 0xAAAA , cAnd, cOr, cXor , cNotb;
     //按位运算
     cAnd = a&b; cOr = a|b; cXor = a^b; cNotb = ~b;
     cout << setfill( '0')<< hex ;       //输出时填充字符设为'0',并用十六进制输出整数
     cout <<"a = "<< setw(4)<< a         //setw(4)表示输出至少 4 位,右对齐,左补填充字符'0'
          <<"\t b = "<< setw(4)<< b << endl;
     cout <<"a&b = "<< setw(4)<< cAnd <<"\t a|b = "<< setw(4)<< cOr << endl;
     cout <<"a^b = "<< setw(4)<< cXor <<"\t ~b = "<< setw(4)<< cNotb << endl;
     //移位运算
     short int bL,bR;
     unsigned short int u = 0xAAAA,uR;
     bL = b << 3; bR = b >> 4; uR = u >> 4;
     cout <<"b << 3 = "<< setw(4)<< bL <<"\t b >> 4 = "<< setw(4)<< bR << endl;
     cout <<"unsigned u = "<< setw(4)<< u <<"\t u >> 4 = "<< setw(4)<< uR << endl;
     return 0;
}
```

运行结果为：

```
a = 00ff   b = aaaa
a&b = 00aa          a|b = aaff
a^b = aa55          ~b = 5555
b << 3 = 5550       b >> 4 = faaa
unsigned u = aaaa   u >> 4 = 0aaa
```

在该程序中对输出的格式有要求，各个 short int 型变量的值统一用 4 位十六进制数(正好表示两个字节的数据)输出。此时需要用输出控制 setw(4)，表示后面的一项数据输出一定占 4 位，由于 setw(4)每次只作用于一项数据，故每个数据的输出都要在前面加上 setw(4)。另外，若输出的十六进制数不足 4 位，要求前面填充 0，故需要用输出控制 setfill('0')将填充字符由默认的空格改为'0'。

在位运算中 &、|和^各有妙用。

- & 用于将某些位设置成 0 或取某些位。

将某些位设置成 0：

```
a    =  000011 11
t    =  111111 01
a&t  =  000011 01
```

取某位的数字：

```
a    =  000011 11
t    =  000000 10
a&t  =  000000 10
```

- | 用于将某些位设置成 1。

将某些位设置成 1：

```
a    =  0 0001111
t    =  0 1000000
a|t  =  0 1001111
```

• ^用于将某些位取反。

将某些位取反：a = 00 001111

t = 00 111100

a ^ t = 00 110011

下列程序利用 & 取位的作用输出字节的各二进制位。

例C3.17 输入一个 int 型的数,输出该数的 32 位二进制数。

```
/*****************************************************************
*     MC3_17.cpp
*     输入一个 int 型的数,输出该数的 32 位二进制数
*****************************************************************/
#include <iostream>
using namespace std;
int main( )
{   int a;     unsigned int t;                  //t 右移运算移进的数字是 0
    cout <<"输入一个整数: ";
    cin >> a;
    cout << hex <<"a 的十六进制数是: "<< a <<",32 位二进制数是: ";
    for(t = 0x80000000 ; t != 0 ; t = t >> 1 )    //不断右移测试位,从(100…0)2 到(00…01)2
    {   if( (a & t) != 0 ) cout << 1;             //取测试位后非零,表示该位数字为 1,否则为 0
        else cout << 0 ;
    }
    cout << endl;
    return 0;
}
```

运行结果为：

```
输入一个带符号长整型的数: -12345↵
a 的十六进制数是: ffffcfc7,32 位二进制数是: 11111111111111111100111111000111
```

该程序的方法是用$(100\cdots0)_2$和变量 a 进行按位与,若 a 的最高位为 1,则按位与的结果为$(100\cdots0)_2$,非零。若 a 的最高位为 0,则按位与的结果为 0。由按位与的结果非零与否可知 a 的相应二进位是 1 还是 0。然后用$(0100\cdots0)_2$测试变量 a 的第二位数字是 1 还是 0,再用$(0010\cdots0)_2$测试,一直到$(00\cdots01)_2$。将测试数$(00\cdots010\cdots0)_2$保存在 unsigned int 类型的变量 t 中,每次用右移运算将测试位移动一位,直到测试位移出变量 a。

注 2：$(100\cdots0)_2$ *右移后要成为*$(010\cdots0)_2$,*则数据必须是无符号的,否则会成为*$(110\cdots0)_2$。

*3.8 数据的输出格式控制

在用 cout 进行数据的输出时,输出格式是可以按照用户的要求进行控制的。本节介绍一些简单的数据输出格式控制,利用这些控制,用户可以编写出有较好输出风格的程序。

进行输出格式控制很简单,只要在 cout 的输出数据中插入相关的格式控制即可,在 cout 中插入的格式控制称为格式操纵符。例如“cout << hex << 32 << endl;”执行后输出为 20。其中,hex 即为格式操纵符,控制后面的整数都以无符号十六进制数输出,故上述语句输出 32 的十六进制值 20。表 3-14 列出了一些简单的格式操纵符。

表 3-14　常用格式操纵符及作用

格式操纵符	作　　用
dec	控制后面的所有整数都以十进制输出，它是默认进制
hex	控制后面的所有整数都以无符号十六进制输出
oct	控制后面的所有整数都以无符号八进制输出
endl	控制光标移动到下一行的第一列，既输出内容换一行
setw(n)	设置输出项的最小宽度为 n，若输出项的宽度较大，以实际宽度输出；若输出项的宽度较小，则输出项右对齐，默认宽度为 1
setfill(ch)	设置填充字符 ch。当设置了较宽的输出宽度而数据项宽度较小时，输出项右对齐，左边补充填充字符，默认是空格
setprecision(n)	设置实型数据输出的精度为 n，默认为 6
setiosflags(x)	设置指定的格式标志 x
resetiosflags(x)	取消指定的格式标志 x

注 1：对于上述表格中的格式操纵符，setw 只作用于后面第一个输出数据，然后恢复默认宽度 1（一次性操纵符）。其他的格式操纵符都是一直作用下去的，直到遇到新的格式操纵符相应格式才会发生改变。

注 2：格式操纵符 dec、hex、oct 只对整数类型（除字符类型外）起作用。

注 3：使用带参数的格式操纵符需要在程序前面加上“#include < iomanip >”。

setiosflags(x)和 resetiosflags(x)所使用的格式标志 x 有很多，可以用于控制实数输出是否用科学记数法（指数形式）、控制数据左对齐、控制十六进制输出数据是否用大写字母等。本节只介绍小数形式和指数形式的输出控制的格式标志，实际上还有省略格式标志的默认格式，详见表 3-15。

表 3-15　实数输出形式的格式标志

格 式 标 志	作　　用
ios_base::fixed	输出小数形式的实数（精度为小数点后的位数）
ios_base::scientific	输出指数形式的实数（精度为尾数小数点后的位数）
默认格式	由数值大小输出小数形式（$10^{-4} \leqslant$ 绝对值 $< 10^{\text{精度位数}}$）或指数形式的实数（精度为有效数字位数，且不输出最后的有效数字 0）

下面的程序例子演示了简单的输出格式控制。

例B3.18　演示数据的常用输出格式控制。

```
/*****************************************************************
*     MB3_18.cpp
*     演示数据的常用输出格式控制
*****************************************************************/
#include < iostream >
#include < iomanip >
using namespace std;
int main( )
{    int a =  - 256 ;
     //整数的格式输出
     cout <<"a = "<< a <<"(十进制)\t"                    //输出十进制数
```

```
        <<oct<<a<<"(八进制)\t"                    //输出无符号八进制数
        <<hex<<a<<"(十六进制)"<<dec<<endl;        //输出无符号十六进制数,再恢复十进制
    cout<<setw(8)<<a<<endl;                       //按字段长8输出a,右对齐,左边填充空格
    cout<<setfill('0')<<setw(8)<<a<<endl;         //填充字符'0',字段长8进行输出
    cout<<setw(8)<<-a<<endl;
    //实数的格式输出
    double x=1./3,y=12.3; float z=-0.000057;
    cout<<"x="<<x<<"\ty="<<y<<"\tz="<<z<<endl;    //默认精度为6(6位有效数字)
    cout<<setprecision(20);                       //设精度为20位(20位有效数字)
    cout<<"x="<<x<<"\ny="<<y<<"\nz="<<z<<endl;    //\n表示换行控制字符
    cout<<setiosflags(ios_base::fixed);           //设输出小数形式(小数点后20位)
    cout<<"x="<<x<<"\ny="<<y<<"\nz="<<z<<endl;
    cout<<setprecision(5);                        //设精度为5位(小数形式下为小数点后5位)
    cout<<"x="<<x<<"\ty="<<y<<"\tz="<<z<<endl;
    cout<<resetiosflags(ios_base::fixed)          //取消小数形式
        <<setiosflags(ios_base::scientific);      //设输出指数形式
    cout<<"x="<<x<<"\ty="<<y<<"\tz="<<z<<endl;    //指数形式输出(小数点后5位)
    return 0;
}
```

运行结果为:

```
a=-256(十进制) 37777777400(八进制) ffffff00(十六进制)
    -256                                  //字段长8输出-256,右对齐,左边填充空格
0000-256                                  //字段长8输出-256,右对齐,左边填充0
00000256
x=0.333333 y=12.3 z=-5.7e-005             //默认格式,6位有效数字,后面的0不输出
x=0.33333333333333331                     //17位有效数字,精度为20位,但实际最多17位
y=12.300000000000001                      //17位有效数字
z=-5.7000001106644049e-005                //太大或太小的数值用指数形式,17位有效数字
x=0.33333333333333331000                  //小数形式,小数点后20位小数,因为精度为20
y=12.30000000000000100000
z=-0.00005700000110664405
x=0.33333  y=12.30000  z=-0.00006         //小数形式,精度为5,5位小数
x=3.33333e-001 y=1.23000e+001 z=-5.70000e-005    //指数形式,精度为5,5位小数
```

下面对运行结果做一下简单说明。

首先int型变量a存储的值是-256,a所占的4个字节实际保存的是-256的补码,即$(11111111\ \ 11111111\ \ 11111111\ \ 00000000)_2=(37777777400)_8=(FFFFFF00)_{16}$,故输出无符号八进制数和无符号十六进制数为37777777400和ffffff00。

在输出整数时,若字段长比数据位数长,右对齐,左边补充填充字符,默认为空格。当cout后插入setfill('0')时,将填充字符变成'0'。若后面再用长字段输出短数据,左边补充的是新的填充字符'0'。

在输出实数时,开始是默认格式,即太大或太小的数值用指数形式,其他数值用小数形式,但是小数后面的0不输出。当用小数格式和指数格式输出时,小数后面的0也要输出。

实数的精度对于默认格式是指有效位数,对于小数格式和指数格式则是小数点后的小数位数。默认格式最多输出17位有效位数,但是小数格式和指数格式则按照精度的大小输出小数位数。

另外，由于 double 型数或 float 型数存放时舍入的原因，实际存放的数据与精确值有微小的误差，在 17 位输出精度下会将误差显示在输出结果中。

用户最后要注意，不要混淆数值的精度与输出精度。数值精度表示数值实际的精确性，表示与精确数的差异；而输出精度表示输出多少位数，这些输出的位数可能是精确数字，也可能是不精确的数字。如变量 a 中数值精确度为 15 位有效数字、变量 b 中数值精度为 4 位有效数字、用精度 3(用 setprecision(3)控制)输出 a、用精度 10 输出 b，则 a 只输出 3 位精确数字，还有很多精确数字没有输出来，b 输出的 10 位有效数字中只有前 4 位是精确数字，其余 6 位数字是不精确数字。

注 4：输出精度表示输出的数字个数，不是数据本身的精度。

练习题

3.1　下列哪些是标识符，哪些是 C++关键字？

cpp　Main　do　try　goto　single　double　a[i]　cout.get　sqrt(2.0)　N100　M-10　2L　0xAA　_　A_B　sin(x)　exam＊1　prime&even　math,score　mathScore　Mr.　man-made　FF00　￥100　Ch3　§3.1　△　π　λ　α　长度　成绩

3.2　画出 C++数据类型分类图。

3.3　写出 VS2008 版编译环境下 int 型数据、unsigned int 型数据和 short int 型数据的最大值、最小值和最大值的位数(十进制)。

3.4　实数中 float 型和 double 型数据的大小范围各是多少？各有几位有效数字？

3.5　字符型数据占几个内存字节？数据的数值范围是多少？

3.6　bool 类型数据占几个内存字节？bool 类型的值有哪些，表示什么含义？

3.7　下列定义是否有错？若有错，说明是什么错误。

(a) Long int　a,b;　double　c,b=3.0;

(b) float　f1,f2,f_100=3.0f;　short int c1=3,d;

(c) char　ch='$',letter.1='a',letter.2='A';　double π=3.1415926536;

(d) int　c=d=5;　unsigned int studentID="111110217";

3.8　下列语句的输出结果是什么(十进制输出格式)？

(a) cout<<100<<endl;　　(b) cout<<0x100<<endl;

(c) cout<<0100<<endl;　　(d) cout<<-0xFF<<endl;

(e) cout<<-0775<<endl;　　(f) cout<<0xffff<<endl;

(g) cout<<-0x8000<<endl;

3.9　下列哪些是合法的 C++字面常量，哪些不是？若不是 C++字面常量，说明理由。

0108　-0111　0x111　0X123　-0xEFG　+16.81　-1035763　0　3711L　1126u　2.11f　.715　-12.　.0　+e15　0.0314e+2　-.1e-10　.e7　'π'　'常'　'<='　'\\'　'\'　'\''　'\28'　'\x28'　'$'　'@'　'\n'　"\""　"a,b,c"　"常量"　"π"　True　false　pi　1K　10L　1.2M

3.10　定义如下的符号常量(不考虑常量的单位)。

(a) 2 的平方根：$\sqrt{2} \approx 1.4142135623730950488$

(b) 圆周率：π ≈3.141592653589793

(c) 欧拉常数：e ≈ 2.7182818284590452354

(d) 真空中的光速：c ≈ 299792458 m/s

(e) 阿伏伽德罗常数：NA ≈ 6.0221367×10^{23}/mol

(f) 重力加速度：g ≈ 9.80665 m/s^2

3.11 将下列表达式按运算符的优先级和结合方向加上括号，得到完全由括号决定运算次序的等价式。

(a) a+3*b-c

(b) x*3+y%4*7+(3/y-5*x)

(c) x<=3&&x>-5‖y>0

(d) y!=0&&x<=y‖x<=1/y

(e) ch=(ch-'A'+3)%26+'A'

(f) x=3,y=4,i=j=0

(g) cout<<3*x+5<<endl

(h) y=*p->s

3.12 写出下列公式的C++表达式。

(a) $\frac{a}{bc}$

(b) $\frac{a}{b}\times c$

c) $\cfrac{1}{a+\cfrac{1}{b+\cfrac{1}{c}}}$

(d) $2+\frac{a+x}{b+x^2}$

(e) 3sin x+4sin 2x

(f) $\sqrt{x^2+5}+3.2-x^3+\frac{2}{3}x^5$

3.13 写出下列判别式(ch为字符变量)。

(a) ch是大写字母

(b) ch是数字

(c) ch是空格

(d) ch是右边括号中的标点符号(，、。、！、？)

3.14 写出下列判别式(x、y是double型变量，a是int型变量，要考虑a为负数)。

(a) $0<2x+3\leqslant10$

(b) $3x^2-2x-7=0$ (误差 10^{-7})

(c) $\sqrt{x^2+3}=x-2$(误差 10^{-7})

(d) a是奇数

(e) a是一个五位数

(f) a是3的倍数但不是4的倍数

(g) a的十位数字是5

(h) a不是1和3

(i) a为1、3、7中的一个

3.15 若有"int a=3,b=7; double x=2.7,y= -7.5;"，写出计算下列表达式后相关变量a、b和x、y的值(只需写出有变化的变量的值)。

(a) ((a= -8)+=7)-=5

(b) a=b/= -2

(c) a*=a*=7

(d) b+=a*=b+a

(e) a=2*a*(a/4)-a

(f) a='\20'+20,b=a*7%2

(g) x*=(y+=5,x+=2,x*y)

(h) x*=y+=5,x+=(2,x*y)

(i) y=2/5*x*y

(j) x=127/4+11/2, a=127/4.0+11.0/2

3.16 若有"char ch; ch=cin.get();"，写出满足下列要求的程序片段。

(a) ch中保存的字符若是大写字母，则改成小写字母；若是小写字母，则改成大写。

(b) ch中保存的字符若是数字，则将数字字符转换成对应数字保存在int型变量a中('0'、'1'、…、'9'转换成0、1、…、9)。

(c) ch中保存的字符若是数字，则改成数字后面第3个数字字符('0'、'1'、…、'9'改成'3'、'4'、…、'2')。

3.17　若有“int a=3,b=0;”,判断下列表达式是否正确,若正确则写出计算后变量 a、b 的值;若不正确说明理由。

(a) (++a)++　　(b) ++(a++)　　(c) ++(++a)

(d) (a++)++　　(e) b=++a　　(f) b=a++

3.18　若有“int a=3; double x=2.7; short int s= -12; unsigned short u=12;”,写出计算下列表达式后相关变量 a、x、s、u 的值(只需写出有变化的变量的值,本题需要用到附录 B 的补码知识)。

(a) x=a=x*2　　(b) a=s=120000　　(c) a=u=120000

(d) s=a=0x10FF00　　(e) u=a=0x10FF00　　(f) u=s= -1

(提示:用十六进制数表示内部数据,再进行类型转换,最后换算成十进制数,有符号类型则用补码)

3.19　若有“double x=2.7;”,写出下列表达式的值及其类型。

(a) int(sqrt(8.0)+0.7)*2　　(b) int((sqrt(8.0)+0.7)*2)

(c) (int)x%2　　(d) double(1*2*3*4)/80

(e) (int)'8'　　(f) (char)('3'+3)

(g) x+(int)(x*2)　　(h) 'a'+0

(i) (int)((unsigned short)(-1))　　(j) (x*x+1)/2

(k) 1/2*(x*x+1)　　(l) 7*9/2-7/2*9

3.20　若有“int a=1,b=4,c=7;”,计算下列表达式的值,并写出计算表达式后变量 a、b、c 的值。

(a) a>5&&c<a+b　　(b) --a>0 || --b>0 || ++c>0

(c) a-- || --b || c++　　(d) ! a&&(b=3)&&c<10

(e) ! (a&&(b=3)&&c<10)　　(f) (a*=2)&&(b%=2)&&c++

3.21　编一程序,计算最小的 n,使得 $1^2+2^2+3^2+\cdots+n^2>32767$。

3.22　编一程序,对输入的 int 型数据做以下改动,将该数原来的百位、十位和个位数字改为新数的个位、百位和十位数字并输出,若输入的数不足 3 位则输出信息“数据太小”。数据改动如 12345 改成 12453,-2375 改成-2753 (提示:取百位数可用 n/100%10)。

3.23　编一程序,将输入的一行字符做加密处理后输出,加密规则(恺撒密码)是对每一个字母在字母表中循环后移 3 位,其余字符不变。

例如:　　I am a student.

加密后为:　　L dp d vwxghqw.

3.24　若有“short int a=0x5A3D,b=0xB5FF,c;”,写出执行下列表达式后 c 的值(用十六进制数表示)。

(a) c=a&b　　(b) c=a | b

(c) c=a^b　　(d) c= ~a

(e) c=a<<3　　(f) c=b>>3

(g) c=(unsigned short)b>>3　　(h) c=a&0xF0F0

(i) c=a^0xF2　　(j) c=a^3

(k) c=(a&0xFF80)|(b&0x7F)　　(l) c=(a<<2) | ((a>>6)&0x3F9)

3.25　编一程序,将输入的 int 型数的补码形式改成原码形式。所谓原码,就是二进制的最高位为符号位(0 表示正数,1 表示负数),其他位是数值位,数值位的值等于该数绝对值的数值位。例如－3 的原码为 10000000 00000000 00000000 00000011(提示:若为负数则取负,并置最高位为 1)。

3.26　编一程序,对输入的 int 型数的二进制奇偶位进行交换处理,并用十六进制数输出处理前后的数据。

例如交换前数据:11011010 01010011 10000110 11010000

交换后数据:11100101 10100011 01001001 11100000

(提示:用 0xAAAAAAAA 取得奇数位的数字,用 0x55555555 取得偶数位的数字,再移位合并)

*3.27　编一程序,对输入的 int 型数进行二进制各位前后倒置,并用十六进制数输出倒置前后的数据。

例如倒置前数据:11001010 01010011 10000000 11000000

倒置后数据:00000011 00000001 11001010 01010011

(提示:原数从高位到低位依次测试,新数从低位到高位设置数字)

*3.28　若有"int a,b,c; char ch;　cin >> a >> b >> c >> ch;",写出满足下列要求的输出语句。

(a) 输出 a 的十六进制数和 b 的八进制数,再输出 c 的十进制数,用跳格'\t'间隔

(b) 按照 8 位十六进制无符号数的形式输出 a 的值,高位数字补 0

(c) 按照两位十六进制无符号数的形式输出 ch 的 ASCII 码值,高位数字补 0(提示:转换成 unsigned char,再转换成 int 按格式输出)

(d) 按照 10 位精度输出$\sqrt{2}$的值

(e) 按照 12 位精度以指数形式输出$\left(\sqrt{2}-\frac{17}{12}\right)^2$的值(提示:避免整数除)

(f) 按照 8 位精度以小数形式输出$\frac{1}{8}$的值