第 3 章　C 语言数据运算

在解决实际问题时不仅要考虑需要哪些数据，还要考虑对数据的操作，以达到求解问题的目的，因此运算符和表达式也是程序设计中首要考虑的基本问题。C 语言具有丰富的运算符和表达式，这为编写程序带来很大的方便。本章主要讲述运算符的功能和用法以及表达式的求值和确定类型的方法。

3.1　数据运算

数据处理是程序的核心部分。在数据处理中，各种运算又是最主要的部分。C 语言不仅数据类型丰富，运算符也十分丰富，几乎所有的操作都可以用运算符来处理。由运算符加适当的运算对象（常量、变量、函数等）可构成表达式，而表达式是 C 语言的重要要素之一，因此掌握好运算符的使用对编写程序是十分重要的。

对于每一个运算符，要注意从两个方面去把握：运算符的优先级和运算符的结合性。运算符的优先级指多个运算符在同一个表达式中时先进行什么运算，后进行什么运算；而运算符的结合性是指运算符所需要的数据是从左边开始取还是从右边开始取，因而有所谓“左结合性”和“右结合性”之说。

3.1.1　运算符

C 语言的运算符十分丰富，且应用非常广泛，可以按功能和运算对象的个数来对运算符分类。

1. 按照功能分类

运算符可按其功能大致分为 5 类：算术运算符、关系运算符、逻辑运算符、位运算符和特殊运算符。

（1）算术运算符　　+　-　*　/　%　++　--

（2）关系运算符　　>　>=　<　<=　==　!=

（3）逻辑运算符　　!　&&　||

（4）位运算符　　<<　>>　~　|　&　^

（5）赋值运算符　　=　复合赋值运算符（+=　-=　*=　/=　%=）

（6）条件运算符　　?:

（7）逗号运算符　　,

（8）指针运算符　　*　&

（9）求字节运算符　　sizeof

（10）强制类型转换运算符（类型标识符）

（11）分量运算符　　->

（12）下标运算符　　[ ]

（13）其他　　如函数调用运算符()

2. 按照连接运算对象的个数分类

运算符可按其运算对象的多少分为单目运算符、双目运算符和三目运算符。单目运算符的运算对象有 1 个，双目运算符要求运算符两侧各有 1 个运算对象。

（1）单目运算符（仅对 1 个运算对象进行操作）。

！　～　++　--　-（取负号）　（类型标记符）*　&　sizeof

例如，求负数单目运算符（-）：

-5

（2）双目运算符（对 2 个运算对象进行操作）。

+　-　*　%　<　<=　>　>=　==　!=　&&　||　<<　>>　&　|　^　=

复合赋值运算符（+=　−=　*=　/=　%=）

例如，加法双目运算符（+）：

2+3

获得两个操作数相除得整数商后的余数的求余双目运算符（%）：3%9 所得余数为 3。

（3）三目运算符（对 3 个运算符对象进行操作）。

? :

例如：

a>b? a:b

如果 a>b，则 a>b? a:b 的值为 a 的值，否则 a>b? a:b 的值为 b 的值。

（4）其他。

()　[]　.　->

3.1.2　优先级及结合性

学习 C 语言的运算符，不仅要掌握各种运算符的功能，以及它们各自可连接的运算对象个数，而且还要了解各种运算符彼此之间的优先级及结合性。运算符可按优先级从高到低分为 15 个等级，如表 3.1 所示。

表 3.1　运算符的优先级与结合性

优先级	运算符	含义	运算量个数	结合性
1	() [] -> .	括号运算符 下标运算符 指向结构体成员运算符 成员运算符		自左至右
2	! ~ ++、-- - (类型) *　& sizeof	逻辑非运算符 按位取反运算符 自加、自减运算符 负号运算符 强制类型转换运算符 指针和取地址运算符 取长度运算符	单目运算符	右结合
3	*、/、%	乘、除、求余运算符	双目运算符	自左至右
4	+、-	算术加、减运算符	双目运算符	自左至右

续表

优先级	运算符	含义	运算量个数	结合性
5	<<、>>	位左移、位右移运算符	双目运算符	自左至右
6	<、<=、>、>=	关系运算符	双目运算符	自左至右
7	= =、!=	关系运算符	双目运算符	自左至右
8	&	按位与运算符	双目运算符	自左至右
9	^	按位异或运算符	双目运算符	自左至右
10	|	按位或运算符	双目运算符	自左至右
11	&&	逻辑与运算符	双目运算符	自左至右
12	||	逻辑或运算符	双目运算符	自左至右
13	?:	条件运算符	三目运算符	右结合
14	=、+=、-=、*=、/=、%= <<=、>>=、&=、|=、^=	复合算术运算符 复合位运算符	双目运算符	右结合
15	,	逗号运算符		自左至右

1. 优先级

求解表达式时，总是先按运算符的优先次序由高到低进行操作。优先级是用来标志运算符在表达式中的运算顺序的，相当于加括号，运算时进行脱括号运算。

2. 结合性

当一个运算对象两侧的运算符优先级相同时，则按运算符的结合性确定表达式的运算顺序。它分为两类：一类运算符的结合性为“从左到右”（大多数运算符），另一类运算符的结合性为“从右到左”。例如：3-5*2，按运算符的优先次序先乘后减，表达式的值为-7；3*5/2，5的两侧“*”和“/”优先级相同，则按结合性处理，算术运算符的结合性为“从左到右”，则先乘后除，表达式的值为 7。

C 语言规定了各种运算符的结合方向（结合性），关于结合性的概念在其他高级语言中是没有的，这是 C 语言的特点之一。

3.2 算术运算

3.2.1 算术运算符

1. 基本算术运算符

算术运算符有：-（取负）、+（取正）、+（加法）、-（减法）、*（乘法）、/（除法）、%（求余数）。

在算术运算符的使用中有以下几点需要注意：

（1）除法运算符“/”的运算结果和其运算对象有关。如果两个运算对象都是符号相同的整数，则运算符“/”的功能是整除，结果为整数。整除的含义是舍去小数部分，只保留整数部分。例如，5/3 的结果为 1，2/3 的结果为 0。两个运算对象中只要有一个是实数，则运算结果就是实数。例如 5.0/2 的结果为 2.5。如果两个运算对象都是整数，但其中有一个是负数，

另一个是正数，则一般采用“向零取整”的原则，即按其绝对值相除后再加上负号。还应注意，在做除法运算时，除数不能为 0。如果除数为 0，则会导致致命的错误而使程序以失败终止。

（2）求余数运算符“%”要求其运算对象都必须是整数，但可正可负。对有负数情况的处理有以下一般性原则：先按其绝对值求余数（|r|%|s|），然后取被除数 r 的符号作为余数的符号。例如，3%9 的结果为 3，5%-3 的结果为 2，-5%3 的结果为-2。

（3）没有乘方运算。例如，平方使用 2*2，立方使用 2*2*2，高次方使用 pow()函数。

2. 算术运算符的优先级

算术运算符的优先级从高到低为：

-（取负）→ *（乘法）、/（除法）、%（求余数）→+（加法）、-（减法）

乘法、除法和求余数的运算优先级相同，加法、减法的运算优先级相同。

例 3.1　算术运算的结果。

```
#include<stdio.h>
void main()
{
    printf("5/3=%d,-5/+3=%d\n",5/3,-5/+3);
    printf("3/5=%d,3.0/5=%f\n",3/5,3.0/5);
    printf("10%%3=%d,-10%%3=%d,10%%-3=%d,-10%%-3=%d\n",10%3,-10%3,10%-3,-10%-3);
}
```

程序运行结果：

```
5/3=1,-5/+3= -1
3/5=0,3.0/5=0.600000
10%3=1,-10%3=-1,10%-3=1,-10%-3=-1
```

由例 3.1 可知，要输出 1 个“%”，在字符串中必须有 2 个“%”，即“%%”。

3. 算术运算符的结合性

算术运算符的结合性为自左至右。

例如，对于以下式子：

```
z = b * e - r % - f + ((a + b ) * c) / p - q
      ① ④③② ⑧   ⑤    ⑥ ⑦ ⑨
```

运算时将按①、②、…、⑧、⑨所示的顺序进行。其中，f 前的“-”号为单目运算符，优先级最高，而 r 前的“-”是代表减运算。如何加以区分呢？凡是+、-号前没有数字或只有*、/、%等级别较高的运算符时，“+”、“-”即为单目运算符，否则为加、减运算符。

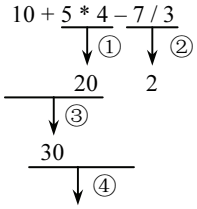
3.2.2　算术表达式

1. 算术表达式运算

算术表达式是用算术运算符将运算对象（常量、变量、表达式）连接起来的表达式。例如 3+6*9、(x+y)/2-1 等，都是算术表达式。算术表达式的结果是一个算术值。

当一个算术表达式中存在多个算术运算符时，各个运算符的优先级与常规算术运算相同，即先乘、除和取余，再计算加、减。同级运算符的计算顺序是从左到右，即先计算左边的算术表达式，再计算右边的表达式。当然也可以用圆括号改变表达式计算的先后顺序。

例如，对于以下算术表达式：



运算时将按①、②、③、④所示的顺序进行计算。

2. 算术表达式书写规则

（1）所有字符必须大小一样写在同一条水平线上。

（2）凡是相乘的地方必须写上“*”，不能省略也不能用小数点代替。

（3）表达式中出现的括号一律使用小括号，并且一定要成对。

（4）函数的自变量（即函数的参数）必须写在括号内。

（5）书写表达式时应注意数据类型、运算符的优先级及结合性。

例 3.2 计算圆锥的体积，圆锥的体积公式为 $v=\frac{1}{3}\pi r^2 h$ 。

如果使用宏定义#define PI 3.14159，则表达式可以写成：

```
V=PI*r*r*h/3
```

或

```
V=1.0/3*PI*r*r*h
```

但不能写成：

```
V=1/3*PI*r*r*h
```

这样写，1/3 的值为 0，因此，导致最后的计算结果为 0。

3.2.3 自增自减运算符

C 语言中有两个特殊的算术运算符，即自增、自减运算符（++和--）。这两个运算符都是单目运算符，它们既可以放在运算对象之前，也可以放在运算对象之后，形成前置形式和后置形式，而运算对象也只能是整型变量。不管前置还是后置，其运算结果都是一样的，都是把运算对象的值增 1 或减 1。设有整型变量 i，则++i、i++都使 i 值增 1，--i、i--都使 i 值减 1。

既然效果相同，那么区分前置和后置形式又有何意义呢？其意义不在于它所作用的变量本身，而在于对变量值的使用。

前置形式：++i、--i，它的功能是在使用 i 之前，i 值先加（减）1（即先执行 i+1 或 i-1，然后再使用 i 值）。

后置形式：i++、i--，它的功能是在使用 i 之后，i 值再加（减）1（即先使用 i 值，然后再执行 i+1 或 i-1）。

例如，j＝3 时：

```
k=++j;          /*赋值时，j 先增 1，再将 j 值赋给 k，结果 k=4，j=4   */
k=j++;          /*赋值时，j 值先赋给 k，然后 j 再增 1，结果 k=3，j=4 */
k=--j;          /*赋值时，j 先减 1，再将 j 值赋给 k，结果 k=2，j=2   */
k=j--;          /*赋值时，j 值先赋给 k，然后 j 再减 1，结果 k=3，j=2 */
```

使用自增、自减运算符应注意以下几点：

（1）自增、自减运算符的对象只能是简单变量，不能是常数或带有运算符的表达式。例如，6--、++(a=2)、++(-i)等都是错误的。

（2）自增、自减运算符的结合性为：前置时是自右向左的，后置时是自左向右的。例如，x=-k++等价于 x= -(k++)，这个表达式的意思是先用 k 变量的当前值加负号赋给 x，然后 k 再加 1。该表达式并不等于 x=(-k)++，因为这样就使++作用在表达式 k 上了，这是（1）中指出的错误。

（3）如果两个运算对象之间连续出现多个运算符，则 C 语言采用“最长匹配”原则。即在保证有意义的前提下，从左到右尽可能多地将字符组成一个运算符。因此，i+++j 就被解释成(i++)+j，而不是 i+(++j)。同样，i++++j 被解释成(i++)++j；i+++++j 被解释成((i++)++)+j，而这两种情况都是错误的。正确的写法应该是在连续的运算符中间的适当位置增加空格分隔符或括号，使分开的部分为有意义的运算对象。比如应把 i+++++j 写成 i++ + ++j 或(i++)+(++j)。

（4）自增、自减运算可以提高程序的执行效率，速度快。因为++n 运算只需要一条机器指令就可完成，而 n=n+1 则要对应三条机器指令。

（5）当 n 为基本数据类型的变量时，++或--表示 n 加 1 或减 1，但当 n 为指针变量或数组下标变量时，其加 1 与减 1 的概念与单纯的加 1 与减 1 是不一样的，请注意后续章节的这一概念。

例 3.3　自增运算应用举例。

```
#include<stdio.h>
void mian()
{
    int a,b,c,d;
    a=5;b=5;a=a++;b=++b;printf("a=%d,b=%d\n",a,b);
    c=a++;d=++b;
    printf("c=%d,d=%d\n",c,d);
}
```

程序运行结果：

```
a=6,b=6
c=6,d=7
```

从运行结果可知，第 1 次 a 的运算是后置运算，其值都相同，等于 6，因为 a 和 b 都是单独使用；第 2 次 a 的运算仍是后置运算，b 的运算仍是前置运算，但 c 和 d 的值不相同。

例 3.4　分析下面程序的运行结果。

```
#include<stdio.h>
void main()
{
    int m=3,n=3,p=3,a,b,c;
    a=(m++)+(m++);
    b=(n++)+(++n);
    c=(++p)+(++p);
    printf("%d,%d\n",m,a);
    printf("%d,%d\n",n,b);
    printf("%d,%d\n",p,c);
}
```

程序运行结果：

```
5,6
5,8
5,10
```

对于 a=(m++)+(m++);语句，两处 m 都是先使用后自加，两个 3 相加赋值给 a，a=6，然后

m 自加两次得到 5。对于 b=(n++)+(++n);语句，有一处 n 是先自加得到 4，另一处 n 是先使用后自加，因此两个 4 相加赋值给 b，b=8，然后 n 再自加得到 5。对于 c=(++p)+(++p);语句，两处 p 都是先自加后使用，因此 p 先自加两次得到 5，然后两个 5 相加赋值给 c，c=10。

像这样能够产生副作用的代码在实际编程中极少用到，这样使用大大降低了程序的易读性，建议大家尽量不要使用。

例 3.5 分析下面程序的运行结果。

```
#include<stdio.h>
void main()
{
    char c1,c2,c3,c4;
    c1=c2=c3=c4='B';
    printf("%c,%c\n",c1,--c1);
    printf("%c,%c\n",--c2,c2);
    printf("%c,%c\n",c3++,c3);
}
```

程序运行结果：

```
A,A
A,B
B,B
```

注意：在 Visual C++中，printf()函数中参数求解的顺序是从右到左。

3.3 混合运算与类型转换

1. 表达式中数据间的混合运算

整型、实型都是数值型的数据，毫无疑问可以进行混合运算。C 语言中字符型和整型可以通用，因此，在 C 语言中整型、实型和字符型的数据可以混合运算。例如：

```
11.2*'b'-100/5+'0'
```

在 C 语言中是合法的表达式。字符型数据以它对应的 ASCII 码值参加计算，上面的表达式相当于：

```
11.2*98-100/5+48
```

2. 数据类型的转换

表达式中运算符所处理对象的数据类型不可能都是同一类型。当表达式中出现类型不同的数据时要对数据进行类型转换，转换的方法有两种：一种是自动转换，另一种是强制类型转换。

（1）自动转换。

自动转换发生在不同数据类型量的混合运算时，由编译系统自动完成。自动转换遵循以下规则：

1）参与运算的数据类型不同时，可先将数据类型转换成同一类型，然后再进行运算。

2）转换按数据长度增加的方向进行，以保证精度不降低。如 int 型和 long 型进行运算，应先把 int 型转换成 long 型后再进行运算。

3）所有的实数运算都是以双精度进行的，即使仅含 float 单精度运算的表达式，也要先转换成 double 型，再进行运算。

4）char 型和 short 型参与运算时，必须先转换成 int 型。

图 3.1 表示了各数据类型自动转换的规则。

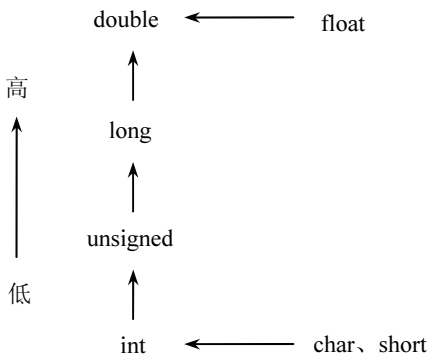


图 3.1　转换规则图

下面以一个实例来看看算术表达式中不同数据类型间的转换。

假设变量的定义为：

```
int i;
float f;
double d;
char ch;
```

现计算 ch/i+f*d-(f+d)的值，则其类型转换如图 3.2 所示。

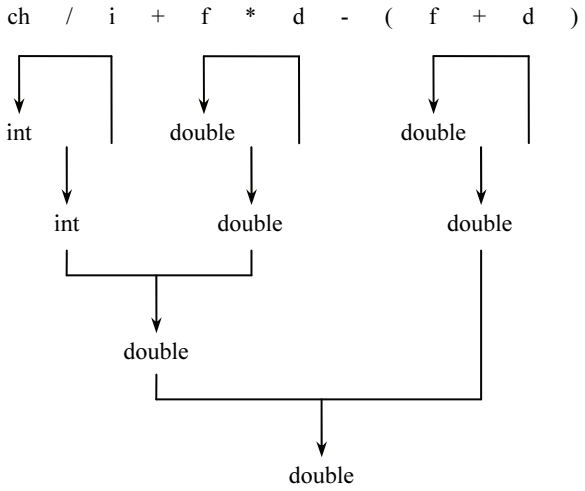


图 3.2　计算表达式 ch/i+f*d-(f+d)的数据类型转换示意图

例 3.6　不同类型数据间的算术运算。

```
#include<stdio.h>
void main()
{
    float a,b,c;
    a=7/2;      /*计算 7/2 得 int 型值 3，因此 a 的值为 3.0*/
    b=7/2*1.0; /*计算 7/2 得 int 型值 3，再与 1.0 相乘，因此 b 的值为 3.0*/
    c=1.0*7/2; /*先计算 1.0*7 得 double 型结果 7.0，再计算 7.0/2，得 c 的值为 3.5*/
```

```
    printf("a=%f,b=%f,c=%f\n",a,b,c);
}
```

程序运行结果：

```
a=3.000000,b=3.000000,c=3.500000
```

（2）强制类型转换。

有时根据需要有必要、有意识地改变某个表达式的数据类型，就需要强制类型转换。强制类型转换是通过类型转换运算来实现的。其一般形式为：

(类型标识符)(表达式)

其功能是把表达式的运算结果强制转换成类型标识符所表示的类型。

例如，设整型变量 a 的值为 36，求表达式(double)(a)的值。就是把变量 a 的值转换成 double 型，表达式的值为双精度实型数值 36.0。

设实型变量 x、y 的值分别为 12.2 和 32.6，求表达式(int)(x+y)的值。就是把 x+y 的计算结果 44.8 转换成 int 型，表达式的值为 44。

使用强制类型转换应注意以下几点：

1）强制类型转换形式中的表达式一定要用括号括起来，否则仅对紧跟强制类型转换运算符的量进行类型转换。而对单一数值或变量进行强制类型转换，则可不要括号。例如：

```
(int) a+b          /*将变量 a 转换成整型再与 b 相加*/
(int) (a+b)        /*将 a+b 的值转换成整型*/
(double) (a)/5     /*等价于(doubel)a/5 */
```

2）强制类型转换的结果是一个指定类型的中间值，而原来变量的类型并没有改变。例如：

```
float x=5.85;
int n;
n=(int) x;
```

其结果得到一个整型值 5，变量 x 的 float 类型并未改变，x 内存放的值 5.85 也未改变。

3）强制类型转换是一种不安全的数值转换，因为强制类型转换在将高类型的数据转换为低类型数据时可能造成数据精度的损失。

例如，上例中 x 的值为 5.85，强制类型转换后的值为 5，x 的小数部分被舍去，精度大大降低。

例 3.7 分析以下程序的运行结果。

```
#include<stdio.h>
void main()
{
    char a='A';
    int b=2,f=5;
    float c=1.2,d=6.8;
    double e=1.25;
    printf("%d\n",(int) (a+b*c-d/e)%f);
}
```

程序运行结果：

```
1
```

程序中表达式 a+b*c-d/e 的计算结果为 61.96，类型为 double 型，求余运算符要求两个运算的对象必须都是整型。因此，必须将表达式的计算结果 61.96，强制转换为整型值 61，否则将会出错。

3.4　赋值运算

3.4.1　赋值运算符

赋值符号“=”就是赋值运算符。赋值运算符的一般形式为：

变量=表达式;

赋值运算符的作用是，首先计算表达式的值，然后将该值赋值给等号左边的变量，实际上是将特定的值写到变量所对应的内存单元中。例如：

```
int a,x;
a=5;                /*将 5 赋值给变量 a*/
x=3*a+55;           /*将表达式 3*a+55 的计算结果 70，赋值给变量 x*/
```

使用赋值运算符应注意以下几点：

（1）赋值运算符的优先级只高于逗号运算符，比其他任何运算符的优先级都低，且有自右向左的结合性。例如：

```
x=1+6/2;
```

由于所有其他运算符的优先级都比赋值运算符高，所以先计算赋值运算符右边的表达式的值，再把此值赋给变量 a。

（2）从形式上看，赋值运算符与数学上的等号是一样的，但含义不一样，因为赋值运算符的操作是将赋值号右边的表达式的值赋给左边的变量。它是单方向的传递操作，因此，假设变量 i 的值为 5，尽管代数中 i=i+1 不成立，但赋值表达式 i=i+1 却是允许的。其作用是首先计算表达式 i+1 的值 6，再将 6 送给赋值号左边的变量 i 中，i 就得到一个新值 6，并且永远保留这个值，直到又重新对 i 赋值为止。

（3）赋值运算符的左侧只能是变量，不能是常数或表达式，而右侧可以是常量、赋过值的变量或表达式。例如，以下是不合法的赋值表达式：

```
12=a                /*赋值运算符的左侧是常量*/
2*a=3*5+55          /*赋值运算符的左侧是表达式*/
x=b                 /*赋值运算符的右侧是没有赋过值的变量*/
```

（4）赋值运算可以连续进行。例如，赋值表达式 a=b=c=5；由于赋值运算符的结合方向是自右向左，因此，这个表达式等于 a=(b=(c=5))。即先将 5 赋值给 c，c 的值赋值给 b，b 的值赋值给 a，a、b、c 的值都是 5。

（5）赋值表达式有值，其值为所赋变量的值。赋值表达式中的“表达式”还可以是一个赋值表达式。例如：

```
a=(x=5)*(y=3);      /*x 的值为 5，y 的值为 3，表达式的值为 15，a 的值为 15*/
x=20-(y=7);         /*y 的值为 7，表达式的值为 13，x 的值为 13*/
```

（6）当赋值运算符两边的类型不一致时，要进行类型转换。

实型类型（float 或 double 型）赋值给整型变量时，舍去小数部分。例如，int k=5.78；则 k 的值为 5。整型数据赋值给实型变量时，数值不变，以实数形式存储到变量中。例如，float x=2；则 x=2.000000。

3.4.2 复合赋值运算符

在赋值运算符前面加上其他运算符，就构成了复合赋值运算符。

+= a+=b 等价于 a=a+b

-= a-=b 等价于 a=a-b

= a=b 等价于 a=a*b

/= a/=b 等价于 a=a/b

%= a%=b 等价于 a=a%b

例如，a-=b+4; 等价于 a=a-(b+4);

为了便于记忆，可以这样理解：

```
a + = b;
a + = b;          /*将等号左边的部分一并平移到等号的右边*/
```

```
= a + b           /*在等号左边补上原来等号左边的变量*/
a = a + b;
```

说明：

①如果等号右边是一个表达式，则相当于它有括号。例如，a%=b+5;等价于 a=a%(b+5);。

②凡二目运算符，都可以与赋值运算符一起组合成复合赋值运算符，共有以下 10 个：

+= -= *= /= %= >>= <<= &= ^= |=

③赋值表达式也可以包含复合赋值运算符。例如：

```
int a=12;
a+=a-=a*a;
```

复合运算符的结合方向是“自右向左”，先计算 a-=a*a，等价于 a=a-a*a，a=12-12*12=-132。再计算 a+=-132，等价于 a=a+(-132)=-264。

④复合运算符在书写时，两个运算符之间不能有空格，否则就是错误的。

例 3.8 分析以下程序的运行结果。

```
#include<stdio.h>
void main()
{
    int n=2;
    n+=n-=n*n;
    printf("n=%d\n",n);
}
```

程序运行结果：

```
n=-4
```

在程序中，对于 n+=n-=n*n 表达式，复合运算符自右向左结合，先执行 n-=n*n，即 n=n-n*n=-2，再执行 n+=-2，即 n=n-2=-2-2=-4，所以输出为 n=-4。

3.5 关系运算

关系运算符用于进行运算对象的比较。用关系运算符将两个运算对象连接起来的式子称

为关系表达式。其中运算对象可以是 C 语言中各种类型的合法表达式。

关系表达式的一般形式为：

<表达式><关系运算符><表达式>

功能：

（1）首先计算关系运算符两边表达式的值；

（2）然后进行两个值的比较。如果是数值型数据，就直接比较值的大小；如果是字符型数据，则比较字符的 ASCII 值的大小；

（3）比较的结果为逻辑值“真”或“假”。由于 C 语言中没有逻辑型数据，因此用数值 1 代表逻辑真，用数值 0 代表逻辑假。即关系表达式的运算结果不是 0 就是 1。

注意：

①关系运算符中>，>=，<，<=的优选级相同，高于==，!=。与算术运算符和赋值运算符相比，关系运算符的优先级低于算术运算符，而高于赋值运算符。

②关系运算符是双目运算，具有左结合性，按照从左至右的顺序运算。一个关系表达式中含有多个关系运算符时，要特别注意它与数学式的区别。

③关系运算符两边的运算对象的数据类型不同时，系统会自动将它们转换成相同的数据类型，之后再进行关系运算。

④一般而言，在 C 程序中可以对实型数据进行大于或小于的比较，但通常不进行==或!=的关系运算。因为实型数据在内存中存放时有一定的误差，很难比较它们是否相等。如果一定要进行比较，则可以用它们的差的绝对值去与一个很小的数相比，如果小于此数，就认为它们是相等的。

例 3.9　计算各关系表达式的值。

（1）10>10

（2）10>=10

（3）0>10!=10

（4）3*8-9!=5+7%3

（5）int a=6,b=8,c=14;　c>b>a

上述关系表达式的计算过程及结果如下：

（1）10 与 10 是两个相等的常量，因此 10>10 的结果为假，该表达式的值为 0。

（2）同上题，表达式 10>=10 的结果为真，返回值为 1。

（3）根据关系表达式的优先级，该表达式等价于(0>10)!=10，0>10 的结果为假返回值为 0，0!=10 的结果为 1。

（4）根据运算符的优先级规则，该表达式等价于(3*8-9)!=(5+7%3)，进一步运算后表达式转化为 15!=6，结果为 1。

（5）先算 c>b，即 14>8 值为 1，1>a 即 1>6 结果为 0。若是数学中的表达式这很显然是正确的，所以要注意这样的问题，正确表达此语义的 C 语句应为 c>b&&b>a。

3.6　逻辑运算

逻辑运算符用于进行逻辑运算。C 语言中有 3 个逻辑运算符，分别是逻辑与（&&）、逻辑或（||）和逻辑非（!）。

其中&&和||是双目运算符，需要两个运算对象，而逻辑非！是单目运算符，只需要一个运算对象。无论是哪一种逻辑运算，都只能对逻辑类型的运算对象进行操作。由于 C 语言中没有逻辑型数据，因此只要运算对象的值不是 0（即包括非 0 的所有正数和负数），就作为“逻辑真”处理，而运算对象的值是 0 则作为“逻辑假”处理。

用逻辑运算符将运算对象连接成的式子称为逻辑表达式。其中运算对象可以是 C 语言中各种类型的合法表达式。

逻辑表达式的一般形式为：

<表达式><逻辑运算符><表达式><逻辑运算符>…

功能：

（1）从左到右依次计算表达式的值，如果是 0 值就作为逻辑假，如果是非 0 值，就作为逻辑真。

（2）按照逻辑运算符的运算规则依次进行逻辑运算，一旦能够确定逻辑表达式的值时，就立即结束运算，不再进行后面表达式的计算。逻辑运算的结果为 0 或 1。

注意：

① 3 个逻辑运算符的优先级顺序依次是逻辑非“!”、逻辑与“&&”、逻辑或“||”。其中“!”是单目运算符，它比算术运算符的优先级高，而“&&”和“||”的优先级则低于关系运算符，高于赋值运算符。

② 逻辑非“!”具有右结合性，要写在运算对象的左边，并且仅对紧跟其后的运算对象进行逻辑非运算。

逻辑与“&&”和逻辑或“||”具有左结合性，要严格按照从左至右的顺序依次运算。

例 3.10 假设 x=10，y=20，z=0 分别计算各逻辑表达式的值。

（1）!x

（2）x||y||z

（3）!x+5||10%20

上述逻辑表达式的计算过程及结果如下：

（1）由于 x=10，非 0 为真，因此!x 结果为 0。

（2）操作数 x 和 y 的值都是非 0 值，根据真值表返回 1，不会再判断 z 值。

（3）根据各运算符的优先级该表达式相当于((!x)+5)||(10%20)，!x 的结果是 0，0+5 是 5 非 0，10%20 结果为 10 非 0，非 0||非 0 返回值为 1。

3.7 逗号运算

C 语言提供了一种特殊的运算符——逗号运算符，又称为顺序求值运算符。用它将两个表达式连接起来。例如：

55+8,7+9

称为逗号表达式。逗号运算符的优先级在所有运算符中最低，并且具有左结合性。逗号表达式的一般格式如下：

表达式 1,表达式 2

逗号表达式的求解过程是：先求表达式 1 的值，再求表达式 2 的值，整个逗号表达式的值是表达式 2 的值。例如，上面的逗号表达式“55+8,7+9”的值为 16。又如，逗号表达式

“a=3*5,4*a”的值为 60，把“a=3*5”作为一个表达式，先求解，即计算和赋值后得到 a 的值为 15，然后求解 4*a，得 60，因此整个逗号表达式的值为 60。

说明：

①逗号表达式又可以和另一个表达式组成一个新的逗号表达式。例如：

```
(a=6,3*a),a+10
```

表达式 1 是(a=6,3*a)，表达式 2 是 a+10，先将 6 赋值给 a，再计算 3*a 得 18，a 的值不变，最后计算 a+10，得 16，整个表达式的值是 16。

②并不是所有出现的逗号都是逗号运算符，函数的参数之间也是用逗号分隔的。例如：

```
c=max(a,b);
```

“a,b”不是逗号表达式，而是 max()函数的两个参数。又如：

```
printf("%d,%d,%d\n",(a,b,c),a,b);
```

“(a,b,c)”是一个逗号表达式，它的值是变量 c 的值，后面的两个逗号是参数的分隔符。

逗号表达式的一般形式可以扩展为：

表达式 1,表达式 2,表达式 3,…,表达式 n

整个表达式的值是表达式 n 的值。例如，若 a、b、c 的初值分别为 2、7、10，则语句“x=(a=a+b,b=c*4,c=c-a);”执行后，变量 x 的值为 1。

例 3.11　假设变量 x 为 int 型，计算各逗号表达式的值。

（1）x=10,x+9

（2）(x=1+2,x+3),x+3

上述逗号表达式的计算过程及结果如下：

（1）先计算表达式“x=10”的值得到结果为 10，变量 x 的值也是 10；然后计算第二个表达式“x+9”的结果是 19，因此整个表达式的值为 19。

（2）先计算出 x 的值等于 3，再进行 x+3 的运算得到结果为 6，此时表达式(x=1+2,x+3)的结果为 6，变量 x 的值为 3；然后进行 x+3 的运算得到结果为 6，因此，整个逗号表达式的结果为 6。

习题 3

3.1　求下面算术表达式的值。

（1）x+a%3*(int)(x+y)%2/4　　设 x=2.5，a=7，y=4.7

（2）(float)(a+b)/2+(int)x%(int)y　　设 a=2，b=3，x=3.5，y=2.5

3.2　写出程序运行结果。

```
#include<stdio.h>
void main()
{   int i,j,m,n;    i=8;  j=10;
    m=++i; n=j++;
    printf("%d,%d,%d,%d",i,j,m,n); }
```

3.3　写出下面表达式运算后 a 的值。设原来 a=12，且 a 已定义为整型变量。

（1）a+=a　　（2）a-=2

（3）a*=2+3　　（4）a/=a+a

（5）a%=(n%=2)，n 的值等于 5　　（6）a+=a-=a*=a