第3章 运算符和表达式

C 语言中运算符和表达式数量之多，在高级语言中是少见的。正是丰富的运算符和表达式使 C 语言的功能非常强大，这也是 C 语言的主要特点之一。

C 语言的运算符可分为算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符以及特殊运算符等。

表达式由运算符和操作数构成，操作数可以是常量、变量、函数调用或是它们的组合。由不同运算符构成对应的表达式，例如由算术运算符和操作数构成算术表达式，由关系运算符和操作数构成关系表达式等。不同的表达式进行不同的运算，有各自不同的规则和作用。

在表达式中，根据运算符的优先级和结合性确定运算顺序。优先级高的先运算，优先级低的后运算。对于优先级相同的运算符要看其结合性，以确定自左向右进行运算还是自右向左进行运算。

3.1 算术运算符和算术表达式

3.1.1 算术运算符

算术运算符用来进行算术运算，包括+、−、*、/、%，分别执行加、减、乘、除和求余操作。

（1）加法运算符+。加法运算符为双目运算符（这里的目指操作数，单目就是一个操作数，双目就是两个操作数，三目就是三个操作数），如 a+b、4+8 等。它具有左结合性即自左向右运算。+号也可以作为正值运算符，代表一个数是正数，此时为单目运算符，如+2、+9 等，当+号为单目运算符时具有右结合性即自右向左运算。

（2）减法运算符−。减法运算符为双目运算符，可以用来计算两个值的差，此时具有左结合性。－也可作负值运算符，此时为单目运算符，如−x、−5 等具有右结合性。

（3）乘法运算符*。乘法运算符是双目运算符，具有左结合性。

（4）除法运算符/。除法运算符是双目运算符，具有左结合性。

注意，相同类型的数据进行四则运算时，结果与运算数的类型相同。不同类型的数据进行四则运算时，按照 2.7 节中所讲的不同类型的数据进行运算的转换规则进行计算。

【例 3-1】 算术运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   printf("%d,%d\n",20/7,-20/7);        //整数常量相除的结果为整数
   printf("%f,%f\n",20.0/7,-20.0/7);    //整数与实数相除结果为double类型的实数
   return 0;
}
```

运行结果如下：

```
2, －2
2.857143, －2.857143
```

本例中，20/7 和－20/7 的结果均为整型数据，小数部分全部舍去。20.0/7 和−20.0/7 由于有实数参与运算，因此结果也为实型。

（5）求余运算符（模运算符）%。求余运算符是双目运算符，具有左结合性。该运算符要求参与运算的操作数均为整型，其结果等于两个整数相除后剩余的余数。

【例 3-2】 求余运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   printf("%d,%d\n",3/2,3%2);
   printf("%d,%d\n",2/2,2%2);
   printf("%d,%d\n",1/2,1%2);
   printf("%d,%d\n",0/2,0%2);
   printf("%d,%d\n",-1/2,-1%2);
   printf("%d,%d\n",-2/2,-2%2);
   printf("%d,%d\n",-3/2,-3%2);
   return 0;
}
```

运行结果如下：

```
1,1
1,0
0,1
0,0
0,-1
-1,0
-1,-1
```

由于被除数=商×除数+余数，因此余数=被除数－商×除数，同时余数的符号由被除数的符号决定。

3.1.2 算术运算符的优先级和结合性

1．优先级

*、/、%的优先级别相同，高于+、−；+、−优先级相同；同一优先级的运算符同时出现时按从左到右顺序计算（左结合性）。

当+、−为单目运算符时（正号、负号）按从右到左的顺序进行计算（右结合性，即先计算表达式的值，再给表达式加上正号或者负号），优先级要高于*、/、%。

因为括号的优先级更高，因此要改变运算顺序只要加括号就可以了。括号全部为圆括号，必须注意括号的配对。算术运算符同括号结合的优先级如图 3-1 所示。

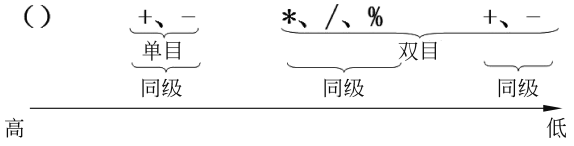

图 3-1 算术运算符同括号结合的优先级

例如，计算 3+8%−3−2，注意−3 中的−为单目运算符，表示 3 是负的；然后计算 8%−3 即 8 除以−3 取余，结果为 2；再计算 3+2−2，按从左到右的顺序计算结果为 3。

2．结合性

结合性是指当一个操作数两侧的运算符具有相同的优先级时，该操作数是先与左边的运算符结合，还是先与右边的运算符结合。

自左至右的结合方向称为左结合性；反之，称为右结合性。

除单目运算符、赋值运算符和条件运算符（三目运算符）采用右结合性外，其他运算符均采用左结合性。

在算术运算符中，只有单目运算符+和−的结合性是右结合（从右到左），其他运算符的结合性都是左结合（从左到右）。

3.1.3 算术表达式

用算术运算符和一对圆括号将操作数连接起来的、符合 C 语言语法规范的表达式称为算术表达式。操作数可以是变量、常量或函数等。

例如：

```
3 + pow(7, 3) / d, 3 + 6 * 9,(x + y) / 2-1
```

在表达式中，在双目运算符的左右两侧各加一个空格，可以增强程序的可读性。

在计算机语言中，对于表达式求值，就是按照表达式中各运算符的运算规则和优先级来获取运算结果的过程。算术表达式的运算规则和要求如下：

（1）在算术表达式中，可以使用多层圆括号，但左右括号必须配对。运算时从内层圆括号开始，由内向外计算表达式的值。

（2）按运算符的优先级次序执行。即先乘除后加减，如果有圆括号，则先计算括号。

（3）如果一个运算对象两侧运算符的优先级相同，则按 C 语言规定的结合性进行。

【例 3-3】 求表达式 15 / (8 % (2 + 1))* 6 +8 – 5 的值。

求值顺序如下：

```
15  / (8  % (2 + 1)) *  6 + 8 - 5
15  / (8  %  3 ) *  6 + 8 - 5
15  /  2  *  6 + 8 - 5
7  * 6  + 8 - 5
42 + 8  - 5
50 - 5
45
```

3.2 关系运算符和关系表达式

3.2.1 关系运算符

在程序中经常需要比较两个量的大小关系，以决定程序的下一步工作。比较两个量的运算符称为关系运算符。在 C 语言中有以下关系运算符：

< 小于

<= 小于或等于

> 大于

>= 大于或等于

== 等于

!= 不等于

关系运算符都是双目运算符，因此都采用左结合性。关系运算符的优先级低于算术运算符，高于赋值运算符。在 6 个关系运算符中，<、<=、>、>=的优先级相同，高于==和!=，==和!=的优先级相同。

3.2.2 关系表达式

关系表达式的一般形式如下：

表达式　关系运算符　表达式

例如，a+b>c−d，x>3/2，'a'+1<c，−i−5*j==k+1 都是合法的关系表达式。

关系表达式也允许出现嵌套的情况，如 a>(b>c)，a!=(c==d)等。

注意，关系表达式的结果是“真”和“假”，是布尔类型的值，但是 C 语言中没有布

尔类型数据，因此 C 语言使用 1 和 0 表示结果的“真”和“假”。而关系运算符两边的操作数可以是任意类型的合法数据，只要这两个数据可以比较即可。

例如，5>0 的结果为“真”，因此该表达式的值为 1；3>5 的结果为“假”，因此该表达式的值为 0。

【例 3-4】 关系运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   if('A'==65) printf("结果成立\n");        //判断相等用两个等号
   else printf("结果不成立\n");
   if(1=='B'>65)  printf("结果成立\n");     //>的优先级高于==, 因此先计算>
   else printf("结果不成立\n");
   if(3>2>1) printf("结果成立\n");          //两个>的优先级相同,因此从前向后算
   else printf("结果不成立\n");     //先计算 3>2, 结果为 1, 再计算 1>1,结果为 0
   return 0;
}
```

注意，不同的运算符在一起运算时一定要根据优先级和结合性的顺序进行计算。

3.3 逻辑运算符和逻辑表达式

3.3.1 逻辑运算符

C 语言中提供了三种逻辑运算符，优先级从高到低为

! 非运算　　&&与运算　　|| 或运算

与运算符&&和或运算符||均为双目运算符，具有左结合性。非运算符!为单目运算符，具有右结合性。例如，a>b&&c>d 等价于(a>b) && (c>d)；!b==c||d<a 等价于((!b)==c)||(d<a)；a+b>c &&x+y<b 等价于((a+b)>c) && ((x+y)<b)。

因此例 3-4 中若想表示 2<3 且 2>1 可以表示为（2<3）&&（2>1），同样表示成绩 score 在 [0,100]上可以表示为 score>=0 && score<=100。

与前面两种运算符的优先级关系如下：

!→算术运算符→关系运算符→&&→||

在关系运算符和逻辑运算符组成的表达式中，也可以用圆括号来改变运算的优先顺序。

逻辑运算的结果也为“真”和“假”两种，用 1 和 0 来表示。其求值规则如下：

（1）与运算&&参与运算的两个量都为真时，结果才为真，否则为假。例如，5>0 && 4>2，由于 5>0 为真，4>2 也为真，因此相与的结果也为真，其值为 1。

（2）或运算||参与运算的两个量只要有一个为真，结果就为真。两个量都为假时，结果为假。例如，5>0||5>8，由于 5>0 为真，相或的结果也就为真。

（3）非运算!后面的运算量为真时，结果为假；后面的运算量为假时，结果为真。例如，!(5>0)的结果为假，其值为 0。

逻辑运算符的操作数可以是任意类型的值，其中 0 表示“假”，所有非 0 的数值均表

示“真”。逻辑运算的结果以 1 代表“真”，以 0 代表“假”。例如，由于 5 和 3 均为非 0，因此 5&&3 的值为“真”，即为 1。又如，5||0 的值为“真”，即为 1。

3.3.2　逻辑表达式

逻辑运算符加上操作数就构成了逻辑表达式，使用时要注意以下几点：

（1）逻辑运算符的优先级都低于算术运算符和关系运算符（逻辑非!除外）。例如，10>1+12&&5>4 等价于(10>(1+12))&&(5>4)，其结果当然是假（即 0）。

（2）在逻辑运算符组成的表达式中，也可以像算术表达式一样，用圆括号来改变运算的优先次序。

（3）进行 a||b 运算时，当 a 为真时则不再求 b 的值，因为 a 为真则 a||b 的值一定为真，与 b 无关，因此不再求 b 的值。

（4）进行 a&&b 运算时，当 a 为假时则不再求 b 的值，因为 a 为假则 a&&b 的值一定为假，与 b 无关，因此不再求 b 的值。

3.4　赋值运算符和赋值表达式

3.4.1　赋值运算符和表达式

一个等号=构成赋值运算符，它的作用是将一个表达式的值赋给一个变量。

用赋值运算符连接起来的式子称为赋值表达式。

赋值运算符的一般形式如下：

```
变量 = 赋值表达式
```

功能：先计算赋值符右边表达式的值，然后将该值赋给赋值号左边的变量，即存入由赋值号左边的变量所代表的存储单元中。整个赋值表达式的值为右边表达式的值。

例如：

```
x = 8
y = (float)7 / 3
```

注意：

（1）赋值号的左边必须是变量，不能是常量或者表达式，这是因为右边表达式的结果要存储在左边变量所在的存储单元中，而常量或表达式并不对应存储单元，因此无法存储右边表达式的值。

例如：

```
a+b=9              不合法
9=a+b              不合法
a=b=7+8            合法，变量 a 和 b 的值都是 15
```

（2）在所有的 C 语言运算符中，赋值运算符的优先级只高于逗号运算符，结合性为自

右至左。

例如表达式 a=b=7+8，按照运算符的优先级，先计算 7+8 的值为 15；按照赋值运算符自右至左的结合性，先将 15 赋给变量 b，然后再把变量 b 的值赋给变量 a。

（3）赋值号 = 是一个运算符，x = 8 是一个表达式，而表达式应该有一个值，C 语言规定最左边变量所得到的值就是赋值表达式的值。

（4）在赋值表达式后面加上分号，就构成了赋值语句，赋值语句是 C 程序中常见的语句。

x＝8 是一个赋值表达式，而“x＝8;”是一条赋值语句。

【例 3-5】 ==与=的区别。

```
#include<stdio.h>
int main(int argc,char *argv[])
{   int a=0;
    if(a==0) printf("%d\n",a);  //0
    if(a=1) printf("%d\n",a);   //1
    return 0;
}
```

第一个 if 语句中的 a==0 中的两个等号是关系运算符，用来判断 a 是否等于 0，此时成立，因此运行结果为 0。

第二个 if 语句中的 a=1 中的一个等号是赋值运算符，用来将 1 赋值给 a，因此运行结果为 1。

注意：初学者经常将关系运算符==写成=，这是一个常见的错误。

3.4.2 复合赋值表达式

1. 复合赋值运算符

在赋值运算符之前加上其他运算符就可以构成复合赋值运算符。C 语言规定的 10 种复合赋值运算符如下：

+=，－=，*=， /=，%=，这 5 个是复合算术运算符。

&=，^=，|=，<<=，>>=，这 5 个是复合位运算符，详见 3.8 节。

2. 复合算术运算符的运算规则

复合算术运算符的运算规则为先计算复合赋值运算符右边表达式的值，然后再使用复合赋值号左边变量和表达式的值进行双目运算，最后得出的结果赋值给左边的变量。各运算符运算规则如表 3-1 所示。

3. 复合赋值表达式

用复合赋值运算符连接起来的式子称为复合赋值表达式。

复合赋值表达式的一般格式如下：

变量　复合赋值运算符　表达式

表 3-1　复合赋值运算符的运算规则

运　算　符	名　　称	例　　子	等　价　于	结　合　性
+=	加赋值	a+=b	a=a+(b)	自右至左
—=	减赋值	a—=b	a=a—(b)	
*=	乘赋值	a*=b	a=a*(b)	
/=	除赋值	a/=b	a=a/(b)	
%=	求余赋值	a%=b	a=a%(b)	

它等价于

变量 = 变量　运算符 (表达式)

注意：当表达式为简单表达式时，表达式外的一对圆括号才可缺省，否则可能出错。例如，x += 5 等价于 x = x+5，而 y *= x + 5 等价于 y = y*(x+5)，而不是 y = y*x+5。

3.5　自加、自减运算符

自加、自减运算符（增量运算符）的作用是使变量的值增加或减少 1。自加运算符记为++，其功能是使变量的值增 1。自减运算符记为--，其功能是使变量值减 1。自加、自减运算符均为单目运算符，都具有右结合性。自加、自减运算符可以放在运算对象的前面也可以放在后面，放在前面称为前置运算符，放在后面称为后置运算符。前置运算符先使变量的值增（或减）1，然后再以变化后的值参与其他运算，即先增减、后运算。后置运算规则为变量先参与其他运算，然后再使变量的值增（或减）1，即先运算、后增减。其表达式与变量变化的规则如表 3-2 所示。

表 3-2　自加、自减运算符的运算规则

原始变量 i 的值	运　　算	表达式的值	运算后变量 i 的值
3	i++;	i++的值为 3	4
3	++i;	++i 的值为 4	4
3	i--;	i--的值为 3	2
3	--i;	--i 的值为 2	2

在理解和使用自加、自减运算符的时候很容易出错。特别是当它们出现在较复杂的表达式或语句中时，常常难以弄清，因此应仔细分析。

【例 3-6】 增量运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
    int i=3;
    printf("%d,%d\n",++i,i);
    i=3;
    printf("%d,%d\n",i++,i);
```

```
    i=3;
    printf("%d,%d\n",--i,i);
    i=3;
    printf("%d,%d\n",i--,i);
    return 0;
}
```

运算结果如下：

```
4,4
3,4
2,2
3,2
```

【例 3-7】 增量运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   int i=3;
   printf("%d,%d\n",++i,i);
   printf("%d,%d\n",i++,i);
   printf("%d,%d\n",--i,i);
   printf("%d,%d\n",i--,i);
   return 0;
}
```

运算结果如下：

```
4,4
4,5
4,4
4,3
```

注意：

（1）自增运算符和自减运算符只能作用于变量，而不能作用于常量和表达式，如 6++、(x-y)--都是不合法的。这是因为增量运算符本质上也是赋值运算，因此同赋值运算符左边必须是变量的原理相同。

（2）++和--结合方向是“自右至左”的，而其他算术运算符的结合方向是“自左至右”的。

（3）自增运算符（++）和自减运算符（--）常用于数组下标改变和循环次数的控制。

（4）不要在一个表达式中对同一个变量进行多次自加、自减运算。例如，写成 a++*--a+a--/++a，这种表达式不仅可读性差，而且不同的编译系统对这样的表达式将有不同的解释，进行不同的处理，因此所得到的结果也是各不相同的。又如表达式 i+++++j 在编译时是无法通过的，应写成（i++）+（++j）。

C 语言中有的运算符是由一个字符构成的，有的是由两个字符组成的，C 编译器处理这种表达式时会尽可能地自左向右进行处理（在处理标识符、关键字时也按照同样的原则处理），例如 i+++j 会被解释为（i++）+j。

3.6　逗号运算符和逗号表达式

C 语言中逗号“,”也是一种运算符，称为逗号运算符。其功能是把两个表达式连接起来组成一个表达式，称为逗号表达式。

其一般形式如下：

表达式 1，表达式 2,…，表达式 n

其求值过程是从前向后求这 n 个表达式的值，然后以表达式 n 的值作为整个逗号表达式的值。

【例 3-8】 逗号运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   int a=2,b=4,c=6,x,y;
   y=(x=a+b, b+c);
   printf("y=%d,x=%d\n",y,x);
   return 0;
}
```

运行结果为

```
y=10, x=6
```

在此例中 y 等于整个逗号表达式的值，也就是等于表达式 b+c 的值。若改为

```
y=(x=a+b),(b+c);
```

则运行结果为

```
y=6, x=6
```

程序先计算 x=a+b，结果为 6，然后 y=x 即 y 的值为 6。本例中，x 是第一个表达式的值；由于赋值符号“=”优先级别高于逗号运算符，因此 y 等于 x 的值。

对于逗号表达式还要说明几点：

（1）逗号表达式一般形式中的表达式 1、表达式 2 等也可以又是一个逗号表达式。例如，表达式 1，（表达式 2，表达式 3），这就形成了逗号表达式的嵌套。因此可以把逗号表达式扩展为以下形式：表达式 1，表达式 2，…，表达式 n。整个逗号表达式的值等于表达式 n 的值。

（2）程序中使用逗号表达式，通常是要分别求逗号表达式内各表达式的值，而并不一定求解整个逗号表达式的值。

（3）并不是在所有出现逗号的地方都组成逗号表达式，如在变量说明中，函数参数表中逗号只是用作各变量之间的间隔符。

（4）逗号运算符的结合性是从左到右，因此逗号表达式将从左到右进行运算。

（5）在所有运算符中，逗号运算符的优先级最低。

3.7 条件运算符和条件表达式

3.7.1 条件运算符和表达式

条件运算符是 C 语言中唯一一个具有三个操作数的运算符（三目运算符），它的一般形式为

表达式 1 ?表达式 2：表达式 3

其运算规则如下：先求解表达式 1 的值，如果值为真（非零），则求解表达式 2 的值，并把它作为整个表达式的结果；如表达式 1 的值为假（零），则求解表达式 3 的值，并把它作为整个表达式的结果。

例如：

```
x>y?x:y
```

在这个表达式中有两个运算符，一个是大于运算符，另一个是条件运算符。因为大于运算符的优先级高于条件运算符，因此先计算 x>y 是否成立，若成立则结果是 x，若不成立则结果是 y。

注意：

（1）在整个条件表达式中，表达式 1 一般是关系表达式，表达式 2 和表达式 3 可以是任意表达式，当然也可以是条件表达式。

例如：

```
x >0?1:x<0?-1:0
```

由于条件表达式的结合性是从右至左，因此上述表达式相当于 x >0?1: (x<0? − 1:0)。

（2）在程序中，常将条件表达式的结果赋值给一个变量。

例如：

```
int a,b,max;
scanf("%d%d",&a,&b);
max=(a>b)?a:b;
```

该程序用来将 a 和 b 的最大值赋值给变量 max。

3.7.2 条件运算符的优先级和结合性

条件运算符的结合性为“从右到左”（即右结合性）。条件运算符的优先级高于赋值运算符，但是低于逻辑运算符、关系运算符、算术运算符。

例如：

```
m = n >15 ? 1:0
```

在这个表达式中，先计算 n>15 是否成立，若成立则条件表达式的值为 1，若不成立则条件表达式的值为 0，最后将条件表达式的结果赋值给变量 m。

3.8 位运算符和位运算表达式

3.8.1 位运算符

位运算符的作用是按位（二进制位）对变量进行运算，但是并不改变参与运算的变量的值。如果要求按位改变变量的值，则要利用相应的赋值运算。另外，位运算符不能用来对浮点型数据进行操作。C 语言中共有 6 种位运算符，分别是~、<<、>>、&、^、|。位运算一般的表达形式如下：

变量 1　位运算符　变量 2

位运算符有不同的优先级，从高到低依次是：~（按位取反）→<<（左移）、>>（右移）→&（按位与）→^（按位异或）→|（按位或）。位运算是指按照二进制位进行的运算。

表 3-3 是位运算符按位取反、与、或和异或的逻辑真值表，X、Y 分别表示两个变量。

表 3-3　位运算符真值表

X	Y	～X	～Y	X&Y	X^Y	X|Y
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	1

~运算符是单目运算符，结合性为自右至左；其他位运算符都是双目运算符，结合性为自左至右。位运算符也可以与赋值运算符一起构成复合位运算符。就是在赋值运算符=的前面加上其他运算符。

以下是 C 语言中的复合位运算符：

（1）<<=（左移位赋值）；

（2）>>=（右移位赋值）；

（3）&=（逻辑与赋值）；

（4）^=（逻辑异或赋值）；

（5）|=（逻辑或赋值）。

复合位运算符的运算规则与复合算术运算符的运算规则相似。复合位运算符的运算规则如表 3-4 所示。

表 3-4 复合位运算符的运算规则

运 算 符	名 称	例 子	等 价 于	结 合 性
<<=	左移赋值	a<<=3	a=a<<(3)	自右至左
>>=	右移赋值	a>>=n	a=a>>(n)	
&=	按位与赋值	a&=b	a=a&(b)	
^=	按位异或赋值	a^=b	a=a^(b)	
|=	按位或赋值	a|=b	a=a|(b)	

很明显采用复合赋值运算符会降低程序的可读性，但这样却可以使程序代码简单化，并能提高编译的效率。对于 C 语言的初学者在编程时最好还是根据自己的理解和习惯去使用程序表达的方式，不要一味追求程序代码的短小。

3.8.2 位运算符的运算功能

1．按位与——&

按位与是指参加运算的两个数据，按二进制位进行“与”运算。如果两个相应的二进制位都为 1，则该位的结果值为 1；否则为 0。

例如：3&5

```
   00000011
&  00000101
   00000001
```

由此可知 3&5=1。

按位与的用途如下。

（1）清零。若想对一个存储单元清零，即使其全部二进制位置 0，只要找一个二进制数，其中各个位符合以下条件：

原来的数中为 1 的位，新数中相应位为 0。然后使二者进行&运算，即可达到清零的目的。

例如，原数为 43，即（00101011）$_2$，另找一个数，设它为 148，即（10010100）$_2$，将两者按位与运算：

```
   00101011
&  10010100
   00000000
```

当然，最简单就是将原数据与 0 进行按位与运算，即可达到将原数据清零的目的。

（2）取一个数中某些指定位。若有一个整数 a（假设占 2 字节），想要取其中的低字节，只需要将 a 与 8 个 1 按位与即可。

例如：

```
   00101100 10101100
&  00000000 11111111
   00000000 10101100
```

（3）保留指定位。

与一个数进行“按位与”运算，此数在该位取 1。

例如，有一个数 84，即（01010100）$_2$，想把其中从左边算起的第 3、4、5、7、8 位保留下来，运算如下：

```
  01010100
& 00111011
  00010000
```

2．按位或——|

两个相应的二进制位中只要有一个为 1，该位的结果就为 1。借用逻辑学中或运算的定义来说就是，一真即真。

例如，48|15，将 48 与 15 进行按位或运算。

```
  00110000
| 00001111
  00111111
```

按位或运算常用来对一个数据的某些位取值为 1。例如，如果想使一个数 a 的低 4 位改为 1，则只需要将 a 与（00001111）$_2$ 进行按位或运算即可。

3．按位异或——^

异或运算符^，也称 XOR 运算符。它的运算规则是若参加运算的两个二进制位相同，则结果为 0，不同时则结果为 1。

例如，3^5=6。

```
  00000011
^ 00000101
  00000110
```

按位异或有如下 3 个特点：

（1）0^0=0，0^1=1，即 0 异或任何数其值不变；

（2）1^0=1，1^1=0，即 1 异或任何数该数取反；

（3）任何数异或自己相当于把自己置 0。

因此异或运算符的作用如下：

（1）使某些特定的位翻转。例如，对数 10100001 的第 6 位和第 7 位翻转，则可以将该数与 00000110 进行按位异或运算。

```
10100001^00000110 = 10100111
```

（2）实现两个值的交换，而不必使用临时变量。例如，交换两个整数 a=10100001，b=00000110 的值，可通过下列语句实现：

```
a = a^b;   //a=10100111
b = b^a;   //b=10100001
a = a^b;   //a=00000110
```

4．按位取反—— ~

按位取反运算符是单目运算符，用于求整数的二进制反码，即分别将操作数各二进制位上的 1 变为 0，0 变为 1。

例如：

```
~63=-64
```

~ 00111111

11000000

按位取反主要用于间接地构造一个数，以增强程序的可移植性。

5．按位左移—— <<

左移运算符是双目运算符，用来将一个数的各二进制位左移若干位，移动的位数由右操作数指定（右操作数必须是非负值），其右边空出的位用 0 填补，高位左移溢出则舍弃该高位。

例如：

```
a=a<<2
```

此示例的目的是将 a 的二进制数左移 2 位，右边空出的位补 0，左边溢出的位舍弃。若 a=15，即（00001111）$_2$，则左移 2 位得（00111100）$_2$。

左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 2×2=4，15＜＜2=60，即将 15 乘以 4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含 1 的情况。

假设以 1 字节（8 位）存一个整数，若 a 为无符号整型变量，则当 a 取值为十进制数 64 即二进制数（01000000）$_2$ 时，左移一位时溢出的是 0，而左移 2 位时，溢出的高位中包含 1，从而使变量 a 的值变为 0。

6．按位右移——>>

右移运算符为双目运算符，用来将一个数的各二进制位右移若干位，移动的位数由右操作数指定（右操作数必须是非负值），移到右端的低位被舍弃，对于无符号数，高位补 0。和左移相对应，右移时，若右端移出的部分不包含有效数字 1，则每右移一位相当于移位对象除以 2。

注意：对于无符号数，右移时左边高位移入 0；对于有符号数，如果原来符号位为 0（该数为正），则左边也是移入 0。如果符号位原来为 1（即负数），则左边移入 0 还是 1 取决于所用的计算机系统。有的系统移入 0，有的系统移入 1。移入 0 的称为“逻辑移位”，即简单移位；移入 1 的称为“算术移位”。

例如：

```
5>>2=1
5      00000101
5>>2   00000001
```

3.8.3　不同长度的数据进行位运算

如果两个数据长度不同（例如 short 型和 int 型）进行位运算时（如 a&b，而 a 为 short 型，b 为 int 型），系统会将两者按右端对齐。如果 b 为正数，则左侧补满 0。若 b 为负，则左端应补满 1。如果 b 为无符号整数型，则左端补满 0。

3.8.4　位运算举例

【例 3-9】 将两个整数进行交换。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
    int  a,b;
    scanf("%d%d",&a,&b);
    a=a^b;
    b=a^b;
    a=a^b;
    printf("%d,%d\n",a,b);
    return 0;
}
```

输入：3 5↙

输出：5，3

具体过程请读者自行展开，进行验证。

位运算有很多具体的应用，这里不再进行过多讨论。

3.9　强制类型转换运算符

不同类型的数据在进行计算时会按照从短到长的格式进行自动转换，但是有时候需要的结果却是短的格式的数据，这时候就需要对数据进行强制转换。

强制类型转换是通过类型转换运算来实现的，其一般形式如下：

（类型说明符）（表达式）

其功能是把表达式的运算结果强制转换成类型说明符所表示的类型。例如，(float) a 把 a 转换为实型，(int)(x+y) 把 x+y 的结果转换为整型。

在使用强制转换时应注意以下问题：

类型说明符和表达式都必须加括号（单个变量可以不加括号），如把(int)(x+y)写成(int)x+y 则成了把 x 转换成 int 型之后再与 y 相加了。

无论是强制转换或是自动转换，都只是为了本次运算的需要而对变量的数据长度进行的临时性转换，这并不改变数据说明时对该变量定义的类型。

【例 3-10】 强制转换运算符的使用。

```
#include<stdio.h>
int main(int argc,char *argv[])
{
   float f=5.75;
   printf("(int)f=%d,f=%f\n",(int)f,f);
   return 0;
}
```

运行结果为

```
(int)f=5,f=5.750000
```

本例表明，f 虽强制转换为 int 型，但只在运算中起作用，是临时的，而 f 本身的类型并不改变。因此，(int)f 的值为 5（删去了小数），而 f 的值仍为 5.75。

3.10 优先级和结合性

C 语言中的单目运算符、赋值运算符（包括复合赋值运算符）和三目运算符都具有自右至左的结合性，其他的运算符都具有自左至右的结合性。各种运算符的优先级以及结合性如表 3-5 所示。

表 3-5 各种运算符的优先级以及结合性

优 先 级	运 算 符	含 义	运 算 对 象	结 合 性
最高 15	()	圆括号或函数参数表	—	自左至右
	[]	数组元素下标		
	→	指向结构体成员		
	.	结构体成员		
14	! ~	非、按位取反	单目	自右至左
	++ −−	自加、自减		
	+ −	正、负		
	*	间接运算符		
	&	取址运算符		
	（类型名）	强制类型转换		
	sizeof	求所占字节数		
13	* / %	乘、除、求余	双目	自左至右
12	+ −	加、减		
11	<< >>	左移、右移		
10	< <=	小于、小于或等于		
	> >=	大于、大于或等于		
9	== !=	等于、不等于		

续表

优　先　级	运　算　符	含　　义	运 算 对 象	结　合　性
8	&	按位与	双目	自左至右
7	^	按位异或		
6	|	按位或		
5	&&	与		
4	||	或		
3	?:	条件运算符	三目	自右至左
2	=	赋值运算符	—	自右至左
	+= −= *= /= %=	复合算术运算赋值		
	<<= >>= &= ^= !=	复合位运算赋值		
最低 1	,	逗号运算符	—	自左至右

3.11　小结

（1）C 语言的运算符非常丰富，使 C 语言的运算非常方便，这也是 C 语言的主要特点之一。

C 语言的运算符可分为算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符以及特殊运算符等。不同的运算符有不同的运算规则。

（2）由运算符和操作数构成表达式，其中操作数可以是常量、变量、函数调用或者是它们的组合。

（3）在表达式中，根据运算符的优先级和结合性确定运算顺序。优先级高的先运算，优先级低的后运算。对于优先级相同的运算符要看其结合性，以确定自左向右进行运算还是自右向左进行运算。

习题 3

1．给出下列程序的运行结果

（1）程序如下：

```
int main(int argc,char *argv[])
{
   int a=10,b=3;
   printf("%d\n",a%b);
   printf("%d\n",a/b*b);
   printf("%d\n",-a%b);
   printf("%d\n",a-=b+++1);
   return 0 ;
}
```

（2）程序如下：

```
int main(int argc,char *argv[])
{
   int i,j,m,n;
   i=8;j=10;
   m=++i;
   n=j++;
   printf("%d,%d,%d,%d\n",i,j,m,n);
   return 0 ;
}
```

（3）程序如下：

```
int main(int argc,char *argv[])
{
    int a=0,b=1;
    a++&&b++;
    printf("%d,%d\n",a,b);
    a=1,b=0;
    a++||b++;
    printf("%d,%d\n",a,b);
    return 0 ;
}
```

2．程序设计题

（1）设长方形的高为 1.5，宽为 2.3，编程求该长方形的周长和面积。

（2）编写一个程序，将大写字母 A 转换为小写字母 a。