# 左值和右值

## （1）概念

**简单而言，左值就是能够出现在赋值符号左边或右边的东西，右值就是只能够出现在赋值符号右边的东西。**

更深一层来说，**左值就是L\_value**，L表示的是定位，地址；**右值就是R\_value**，R表示的是读取数据。

例如，a=b，这句话中a是一个左值，b是一个右值，那么可以说，将b地址内存里面的数据放到a地址内存中。

## （2）合法的表达式

**表达式是指由单个或多个操作数、运算符组成的符合C语言规则的式子。**

**左值可以是一个变量，或者是一个引用对象的特殊的表达式。**

**右值可以是一个常量，字面值常量，表达式。**

通过几个**例子**大家看一下：

a=3 正确的表达式

-i++错误的表达式（-i不是一个左值，没有办法寻址）

3=x错误的表达式

'a'+=5错误的表达式，原因在于，字符型字面值不可以作为左值。

# 运算符总览与运算符优先级

### 算术运算符

下表显示了 C 语言支持的所有算术运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| + | 把两个操作数相加 | A + B 将得到 30 |
| - | 从第一个操作数中减去第二个操作数 | A - B 将得到 -10 |
| \* | 把两个操作数相乘 | A \* B 将得到 200 |
| / | 分子除以分母 | B / A 将得到 2 |
| % | 取模运算符，整除后的余数 | B % A 将得到 0 |
| ++ | 自增运算符，整数值增加 1 | A++ 将得到 11 |
| -- | 自减运算符，整数值减少 1 | A-- 将得到 9 |

**对于++，--和其他运算符不同的区别就是他们的操作数是单操作数。**

### 关系运算符

下表显示了 C 语言支持的所有关系运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| == | 检查两个操作数的值是否相等，如果相等则条件为真。 | (A == B) 不为真。 |
| != | 检查两个操作数的值是否相等，如果不相等则条件为真。 | (A != B) 为真。 |
| > | 检查左操作数的值是否大于右操作数的值，如果是则条件为真。 | (A > B) 不为真。 |
| < | 检查左操作数的值是否小于右操作数的值，如果是则条件为真。 | (A < B) 为真。 |
| >= | 检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。 | (A >= B) 不为真。 |
| <= | 检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。 | (A <= B) 为真。 |

**如果结果值为真的话，输出的结果为1，输出的的值为假的话，输出的结果0.**

### 逻辑运算符

下表显示了 C 语言支持的所有关系逻辑运算符。假设变量 A 的值为 1，变量 B 的值为 0，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| && | 称为逻辑与运算符。如果两个操作数都非零，则条件为真。 | (A && B) 为假。 |
| || | 称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。 | (A || B) 为真。 |
| ! | 称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。 | !(A && B) 为真。 |

**在此运算符中，有一个短路特性：**

**（表达式1）&&（表达式2）如果表达式1为假，则表达式2不会进行运算，即表达式2“被短路”。**

**（表达式1）||（表达式2）如果表达式2位真，则表达式2不会进行运算，即表达式2“被短路”**。

### 位运算符

下表显示了 C 语言支持的位运算符。假设变量 A 的值为 60，变量 B 的值为 13，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| & | 如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。 | (A & B) 将得到 12，即为 0000 1100 |
| | | 如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。 | (A | B) 将得到 61，即为 0011 1101 |
| ^ | 如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。相异为1，相同为0 | (A ^ B) 将得到 49，即为 0011 0001 |
| ~ | 二进制补码运算符是一元运算符，具有"翻转"位效果。 | (~A ) 将得到 -61，即为 1100 0011，2 的补码形式，带符号的二进制数。 |
| << | 二进制左移运算符。左操作数的值向左移动右操作数指定的位数。 | A << 2 将得到 240，即为 1111 0000 |
| >> | 二进制右移运算符。左操作数的值向右移动右操作数指定的位数。 | A >> 2 将得到 15，即为 0000 1111 |

### 赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| = | 简单的赋值运算符，把右边操作数的值赋给左边操作数 | C = A + B 将把 A + B 的值赋给 C |
| += | 加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数 | C += A 相当于 C = C + A |
| -= | 减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数 | C -= A 相当于 C = C - A |
| \*= | 乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数 | C \*= A 相当于 C = C \* A |
| /= | 除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数 | C /= A 相当于 C = C / A |
| %= | 求模且赋值运算符，求两个操作数的模赋值给左边操作数 | C %= A 相当于 C = C % A |
| <<= | 左移且赋值运算符 | C <<= 2 等同于 C = C << 2 |
| >>= | 右移且赋值运算符 | C >>= 2 等同于 C = C >> 2 |
| &= | 按位与且赋值运算符 | C &= 2 等同于 C = C & 2 |
| ^= | 按位异或且赋值运算符 | C ^= 2 等同于 C = C ^ 2 |
| |= | 按位或且赋值运算符 | C |= 2 等同于 C = C | 2 |

### 杂项运算符 sizeof & 三元

下表列出了 C 语言支持的其他一些重要的运算符，包括 sizeof 和 ? :。

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| sizeof() | 返回变量的大小。 | sizeof(a) 将返回 4，其中 a 是整数。 |
| & | 返回变量的地址。 | &a; 将给出变量的实际地址。 |
| \* | 指向一个变量。 | \*a; 将指向一个变量。 |
| ? : | 条件表达式 | 如果条件为真 ? 则值为 X : 否则值为 Y |

### C 中的运算符优先级

**运算符的优先级**确定表达式中项的组合。这会影响到一个表达式如何计算。某些运算符比其他运算符有更高的优先级，例如，乘除运算符具有比加减运算符更高的优先级。

下表将按运算符优先级从高到低列出各个运算符，具有较高优先级的运算符出现在表格的上面，具有较低优先级的运算符出现在表格的下面。在表达式中，较高优先级的运算符会优先被计算。

|  |  |  |
| --- | --- | --- |
| 类别 | 运算符 | 结合性 |
| 后缀 | () [] -> . ++ - - | 从左到右 |
| 一元 | + - ! ~ ++ - - (type)\* & sizeof | 从右到左 |
| 乘除 | \* / % | 从左到右 |
| 加减 | + - | 从左到右 |
| 移位 | << >> | 从左到右 |
| 关系 | < <= > >= | 从左到右 |
| 相等 | == != | 从左到右 |
| 位与 AND | & | 从左到右 |
| 位异或 XOR | ^ | 从左到右 |
| 位或 OR | | | 从左到右 |
| 逻辑与 AND | && | 从左到右 |
| 逻辑或 OR | || | 从左到右 |
| 条件 | ?: | 从右到左 |
| 赋值 | = += -= \*= /= %=>>= <<= &= ^= |= | 从右到左 |
| 逗号 | , | 从左到右 |

# 运算符详解

## 条件运算符

1. /\*
2. 条件运算符为?和：，它是一个三元运算符，即有三个参与运算的操作数。由条件运算符组成条件表达式的一般形式为：
3. 表达式1 ?  表达式2 : 表达式3
4. 其求值规则为：如果表达式1的值为真，则以表达式2 的值作为条件表达式的值，否则以表达式3的值作为整个条件表达式的值。
5. 在这个过程中，如果表达式1为真，那么表达式3的结果不在进行运算，即发生忽略，反之，表达2的结果进行忽略。
6. \*/
7. #include<stdio.h>
8. #include<stdlib.h>
9. **int** main(**void**)
10. {
11. **int** a=3,b=4,c=5,d=6,max;
12. max=a>b?a:b;//结果就是为了得到a和b中较大的那个数
13. printf("%d",max);//条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值符。
14. max=a>d?a:c>d?c:d;//条件运算符的结合方向是自右至左。
15. printf("%d",max);
16. system("pause");
17. **return** 0;
18. }

## 逻辑运算符

1. /\*
2. 与运算符(&&)和或运算符(||)均为二元运算符。具有左结合性。非运算符(!)为一元运算符，具有右结合性。
3. “&&”和“||”低于关系运算符，高于赋值运算符，且&&的优先级高于||的优先级，“!”高于算术运算符。
4. \*/
5. #include<stdio.h>
6. #include<stdlib.h>
7. **int** main(**void**)
8. {
9. **int** a=3,b=2,c=1,d=5,e=6,f;
10. f=a<b||b<c&&c<d||d<e;
11. printf("%d",f);
12. system("pause");
13. **return** 0;
14. }

## 关系运算符

1. /\*
2. 关系运算符都是二元运算符，其结合性均从左至右。关系运算符的优先级低于算术运算符，
3. 高于赋值运算符。在六个关系运算符中，<、<=、>、>=的优先级相同，高于==和!=，==和!=的优先级相同。
4. \*/
5. #include<stdio.h>
6. #include<stdlib.h>
8. **int** main(**void**)
9. {
10. **int** x=3,y;
11. y=2<x>=8;//从左至右运算
12. printf("%d",y);
13. system("pause");
14. **return** 0;
15. }

## 赋值运算符

1. /\*
2. 赋值表达式的功能是计算表达式的值再赋予左边的变量。赋值运算符具有右结合性，因此a=b=c=5可理解为a=(b=(c=5))。
3. \*/
4. #include<stdio.h>
5. #include<stdlib.h>
7. **int** main(**void**)
8. {
9. **int** x=3,y=4;
10. x+=y;
11. printf("%d",x);
12. system("pause");
13. **return** 0;
14. }

**在赋值运算符中体现了类型转换，就是说，如果赋值运算符两边的数据类型不相同的话，系统将自动进行类型转换，即把赋值号右边的类型转换为左边的类型。**

**具体规定如下：**

**实型赋予整型，舍去小数部分。**

**整型赋予实型，数值不变，但将以浮点形式存放，即增加小数部分（小数部分的值为0）**

**字符型赋予整型，由于字符型为一个字节，而整型为二个字节，故将字符的ASCII码的值放到整型量的低八位中，高八位为0.整型赋予字符型，只把低八位赋予字符量**

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **int** a,c,b=97;
7. **float** x,y=8.88;
8. **char** c1='a',c2;
9. a=y;
10. x=b;
11. c=c1;
12. c2=b;
13. printf("a=%d, x=%f, c=%d, c2=%c \n", a, x, c, c2);
14. printf("%d",x);
15. system("pause");
16. **return** 0;
17. }

## 算术运算符

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **int** x=4,y=3,z;
7. printf("%d,%d\n",20/7,-20/7);//当两个操作数均为整型或者是char型时，除法运算实现了整型的运算，结果与被除数的符号相等
8. printf("%f,%f\n",20.0/7,-20.0/7);//只要有一个操作数是实型，则结果的类型为实型
9. printf("%d\n",100%3);//求余运算要求两个操作数均为整型
10. z=x+++y;//++i：i自增1后再参与其它运算。--i：i自减1后再参与其它运算。i++：i参与运算后，i的值再自增1。i--：i参与运算后，i的值再自减1。
11. printf("%d,%d",x,z);
12. z=x+++--y;
13. printf("%d,%d,%d",x,y,z);
14. system("pause");
15. **return** 0;
16. }

## 逗号运算符

1. /\*
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，并不一定要求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. \*/
5. #include<stdio.h>
6. #include<stdlib.h>
8. **int** main(**void**)
9. {
10. **int** a=3,b=5,z;
11. z=a>b,a+b;//在这个运算过程中，逗号的运算优先级比赋值运算的优先级要低，因此应该先算赋值运算，再算逗号运算
12. printf("%d",z);
13. z=(a>b,a+b);//在这个运算过程中，赋值的运算优先级比特殊运算符的优先级要低，因此应该先算特殊运算，再算赋值运算运算，在特殊运算中，逗号运算的结果是输出逗号后面的数。
14. printf("%d",z);
15. system("pause");
16. **return** 0;
17. }

## 位运算符

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **int** a=9,b=5,c;
7. c=a&b;
8. printf("%d\n",c);//按位与运算符"&"是二元运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的两个二进位均为1时，结果位才为1 ，否则为0。参与运算的数以补码方式出现。
9. c=a|b;
10. printf("%d\n",c);//按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只要对应的二个二进位有一个为1时，结果位就为1。参与运算的两个数均以补码出现。
11. c=a^b;
12. printf("%d\n",c);//按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为1.相同为0.
13. c=~a;
14. printf("%d\n",c);//求反运算符～为单目运算符，具有**右结合**性。 其功能是对参与运算的数的各二进位按位求反。
15. c=a<<4;
16. printf("%d\n",c);//左移运算符“<<”是双目运算符。其功能把“<< ”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数
17. c=b>>4;
18. printf("%d\n",c);//右移运算符“>>”是双目运算符。其功能是把“>> ”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。
19. // 应该说明的是，对于无符号数，在移动的时候前面补0，对于有符号数，在右移时，符号位将随同移动。当为正数时， 最高位补0，而为负数时，符号位为1
20. system("pause");
21. **return** 0;
22. }

**在求反运算中需要注意：**

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **short** a=0xf45a,b;
7. b=~a;
8. printf("%x\t%x",a,b);//这个的结果说明在运算中如果%x表明了将结果以int的形式给输出，如果结果前面为1则前面的几位都为1，如果为0，则前面的也为0.
9. system("pause");
10. **return** 0;
11. }

# 类型转换

**类型转换分为自动类型转换和强制类型转换**

## 自动类型转换

自动类型转换遵循下面的规则：

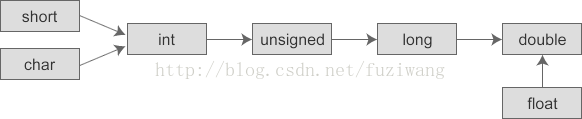
1、若参与运算的数据类型不同，则先转换为同一类型，然后在进行运算。

2、转换按数据长度增加的方向进行，以保证精度不降低。例如int型和long型运算时，先把int量转成long型后再进行运算

3、所有的浮点数运算都是以双精度进行的，即使仅含有float单精度量运算的表达式，也要先转换成double型后再进行运算。

4、char型和short型参与运算的时候，必须先转换成int型。

5、在赋值运算中，赋值号两边的数据类型不同时，需要把左边的表达式的类型转换成左边变量的类型。如果右边表达式的数据类型长度比左边长时，将丢失一部分数据，这样会降低精度。



#include<stdio.h>

int main(){

    float PI=3.14159;

    int s1, r=5;

    double s2;

    s1 = r\*r\*PI;

    s2 = r\*r\*PI;

    printf("s1=%d, s2=%f\n", s1, s2);

    return 0;

}

在这个代码中运算结果为：s1=78, s2=78.539753在计算表达式r\*r\*PI表达式中的r和pI都转换为double型，表达式的结果也为double型，但是由于s1为整型，所以赋值运算的结果仍为整型，舍去了小数部分。

## 强制类型转换

**人为的改变类型，已达到人们想要的效果**

# 实例

#include<stdio.h>

#include<stdlib.h>

int main(void)

{

int a=1,b=0,c=1,d;

d=0\*a\*a>b?a:b\*(++b)\*c;//考虑优先级，注意逗号的优先级是最低的，其次就是赋值运符，然后就是条件运算符

printf("%d",d);

system("pause");

return 0;

}