# C语言当中的运算符（重点）

## **概念**

-  表达式：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：左值是一种状态，表示一块内存是否可以进行写  
           入操作（赋值），如果一块内存允许被赋值，我  
 们称这块内存是左值。  
 允许出现在赋值号左侧的，我们就称为左值。

**可以改变的**  
-  右值：表示内存可进行读取操作的一种状态（取值）  
  
-  学习运算符的时候，学习什么  
 1、运算规则  
 2、操作数个数  
 3、结合性  
 4、优先级

## **算术运算符（+，-，\*，/，%，++，--）**

-  [+]  :  表示加法，二元运算符、从左往右  
 -  [-]  :  表示减法，二元运算符、从左往右  
 -  [\*]  :  表示乘法，二元运算符、从左往右  
 -  [/]  :  表示除法，二元运算符、从左往右  
 -  两个整数相除，结果为整型（取整）  
 **-  一个整数与一个浮点数进行（+-\*/）运算，结果为浮点数** -  除数不能为0  
 -  *Pyhton与C语言对于负数整除的处理是不一样的  
 Python:  -3  //  2      =>      -2  
 C          :  -3  /  2        =>      -1  
 原因是因为Python向负无穷取整、C向0取整* -  [%]  :  表示取余（取模）,二元运算符、从左往右  
 -  **C语言中取余运算的结果  =  两个操作数绝对值进行取余，符号  
    与左操作数相同** -  *Pyhton与C语言对于取余的处理是不一样的  
 Python:  -5  %  3      =>      1  
 C          :  -5  %  3      =>    -2  
  
 a  %  b  =  a  -  a整除b  \*  b  
 -5  %  3  =  -5  -  -2  \*  3  =  -5  +  6  =  1  
 -5  %  3  =  -5  -  -1  \*  3  =  -5  +  3  =  -2* **-  实际开发中，主要对正整数进行取余，尽量减少负数**    取余运算 ，如果涉及到负数取余，需要自己先明确负  
 数取余的规则，再选择如何去取余  
 -  只能对整数进行取余运算，小数不能进行取余  
 -  取余运算符右操作数不能为0

-  [++]  :  自增运算符，一元运算符  
**-  前自增运算符  （前++）  ++操作数  
-  先对操作数进行+1操作，再参与表达式运算。**

**-  后自增运算符  （后++）  操作数++  
-  先参与表达式运算，在对操作数进行+1操作。**

      int  i=1,  j=1;      
 printf("++i  =  %d\n",  ++i);  //  2  
 printf("j++  =  %d\n",  j++);  //  1  
  
 printf("i  =  %d\n",  i);  //  2  
 printf("j  =  %d\n",  j);  //  2  
  
 -  自增运算符主要使用在循环中  
 -  在实际开发中，不允许在**一个表达式**中对**一个变量**多次进行  
    自增、自减操作

-  [--]  :  自减运算符，一元运算符，同自增

## 赋值运算符（=，op=）

- 赋值运算符 ：=

- **不是判断是否相等**

- 语法形式： 操作数1 = 操作数2

- 把操作数2的值赋值给操作数1

- 操作数1必须是可被修改的左值

- 二元运算符，从右到左，**优先级倒数第二低**

**- 运算结果为右值**

- 复合赋值运算符

- 算术复合赋值运算符：+=、-=、\*=、/=、%=

- += ：二元运算符，从右到左

int a = 10; a += 10; // a=20

在变量a本身的基础上 +10，并把结果赋值给a

a += 10 <==> a = a + 10

- 位复合赋值运算符：&= |= ~= <<= >>= ^=

## 关系运算符（>、 <、 >=、 <=、 ==、 !=）

- 在运算时： 0：表示假 非0：表示真

- 在结果中： 0：表示假 1： 表示真

- >、 <、 >=、 <=、 ==、 !=

- 二元运算符、从左到右

- **结果为0或1如果关系表达式成立，结果为1，不成立，结果为0**

- int a = 200; 0 <= a <= 100 // 结果恒定为1

## 逻辑运算符（判断真假，&&，||，!）

- 逻辑与 ： &&

真 && 真 => 真

真 && 假 => 假

假 && 真 => 假

假 && 假 => 假

逻辑与的短路问题：当左操作数结果为假时，右操作数不执行

- 逻辑或 ： ||

真 || 真 => 真

真 || 假 => 真

假 || 真 => 真

假 || 假 => 假

逻辑或的短路问题：当左操作数结果为真时，右操作数不执行

- 逻辑非 ： !

!真 => 假

!假 => 真

## 条件运算符(操作数1 ？操作数2 ：操作数3)

- 操作数1 ? 操作数2 : 操作数3

- 判断[操作数1]的结果，如果结果为真（1），执行[操作数2]，

并把[操作数2]的结果作为整个表达式的结果，如果结果

为假（0），执行[操作数3]，并把[操作数3]的结果作为

整个表达式的结果，

- **C语言中唯一一个三元运算符**

int a;

scanf("%d", &a);

a>=0&&a<=100 ? printf("合法\n") : printf("不合法\n");

printf("%s\n", a>=0&&a<=100 ? "合法" : "不合法");

printf(a>=0&&a<=100 ? "合法\n" : "不合法\n");

## 位运算符（&，|，~，^, <<,>>，）

- 对内存中每一位进行操作的运算符

- 位与(&) : 对两个操作数，按位进行与运算**（同1为1）**

3 & 4 => 0011 & 0100 => 0000 => 0

- 位或(|) : 对两个操作数，按位进行或运算**（同0为0）**

- 位非(~) : 对操作数，按位进行非运算 0->1 1->0

- 位异或(^) : 对两个操作数，按位进行异或运算**（相同为0，不同为1）**

**0 异或 0 => 0**

**1 异或 0 => 1**

**0 异或 1 => 1**

**1 异或 1 => 0**

- 左移(<<)

- 操作数1 << 操作数2

- 将[操作数1]整体往左移动[操作数2]位。

- **x =** 2 << 1 => x =4

- 对于正整数而言，相当于 [操作数1] 乘以 2的[操作数2]次方

- ***对于负整数而言，C语言没有给出定义，结果是未定义的***

- 所有位整体左移，低位补0，**如果是有符号数，符号位会被数据位填充（丢弃最高位，低位补0）**

***问题：char x = ‘a’； int x = 0x7fffffff, y = 1;***

***int y； long int z;***

***z = x << y;***

***y = x <<= 4 printf("%ld\n",z);***

***结果是 -2***

***当 x = 2647483647时结果为9999998 ？？？***

- 右移(>>)

- 操作数1 >> 操作数2

- 将[操作数1]整体往右移动[操作数2]位。

- **x =** 4 >> 1 => x =2

- 对于正整数而言，相当于 [操作数1] 整除 2的[操作数2]次方

- ***对于负整数而言，C语言没有给出定义，结果是未定义的***

- 大部分编译器，对于负数右移运算，符号位补1，来实现

负整数整除2的效果

## 逗号运算符 (,)

- **\*\*优先级最低\*\*的运算符，二元运算符，从左到右**

- 操作数1 , 操作数2

- 先执行[操作数1]，再执行[操作数2]，然后把[操作数2]作为

整个表达式的结果。

常用形式（x，y）

## () 括号运算符

- 可以通过括号运算符更改运算符的优先级

## sizeof() 运算符

- 求[操作数]数据类型占用的字节数

- sizeof(100)

- sizeof(int)