# 第四章 运算符和表达式

一、运算符和表达式简介

二、运算符分类

三、运算符优先级

四、类型转换

# 一、运算符和表达式简介

## 1.1 运算符

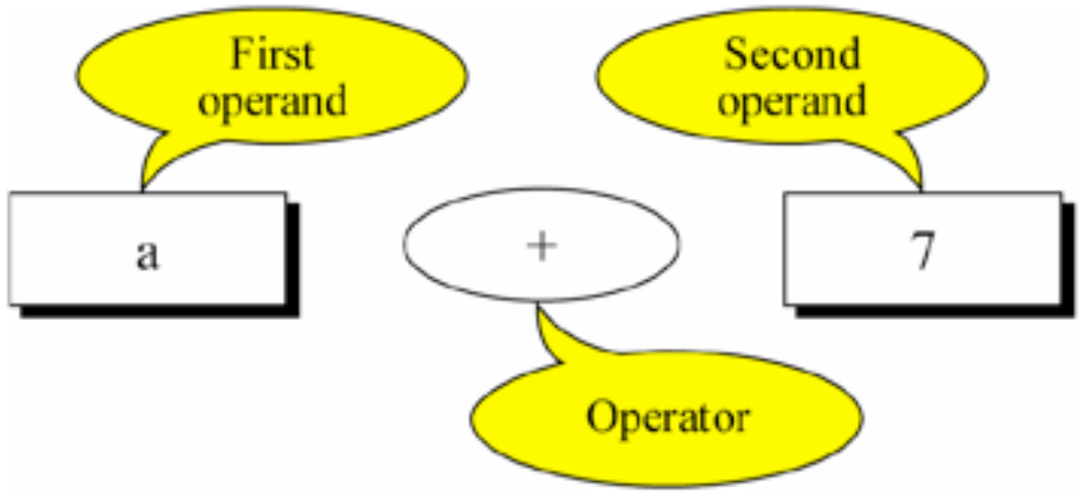
运算符也称操作符，C语言中的运算符主要用于构造表达式，C语言的运算异常丰富，除了控制语句和输入输出以外的几乎所有的基本操作都作为运算符处理。简单点来说，操作符的作用就是为了构造表达式，操作操作数，计算表达式的值，最终得到结果。如2 +5，其中的+运算符用于操作两个数的和。

## 1.2 表达式

表达式是由操作符和操作数组成，甚至可以是一个单一的值。

# 二、运算符分类

## 2.1 算术运算符




算术运算符主要有:

- +（加号） 加法运算(3+3)
- -（减号） 减法运算(3 - 1)
- *（星号） 乘法运算(3*3)
- /（正斜线） 除法运算(3/3)
- %（百分号） 求余运算10%3=1 (10/3=3……1)

算术运算符示例：

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int i = 5;
    int j = 2;
    printf("%d,%d,%d,%d,%d\n",i+j,i-j,j*j,i/j,i%j);
    return 0;
}
```

程序运行结果如下：
7,3,4,2,1

注：1、对于“/”运算，如果操作数中有浮点型则结果一定为浮点型

```
printf("%f\n",5/2.0);//结果为2.500000
```

2、对于“%”运算，操作数不可以为浮点型

```
printf(“%f\n",5%2.0);
```

Invalid operands to binary expression ('double' and 'double')

# 2.2 自增自减运算符

- 自增运算符记为“++”,其功能是使变量的值自增1。
- 自减 运算符记为“--”,其功能是使变量值自减1。

而自增和自减运算符根据前缀还是后缀又分为前置和后置的自增自减运算。

1、后置自增自减运算符即++或--在变量名后，如i++、i--

后置自增自减特点：先取变量本身的值，再做自增自减运算

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int i = 1;
    int j = i++;
    printf("i = %d,j = %d\n",i,j);
    return 0;
}
```

程序运行结果如下：

```
i = 2,j = 1
```

2、前置自增自减运算符即++或--在变量名前，如 ++i、--i

前置自增自减特点：先对变量自增自减运算，再取值

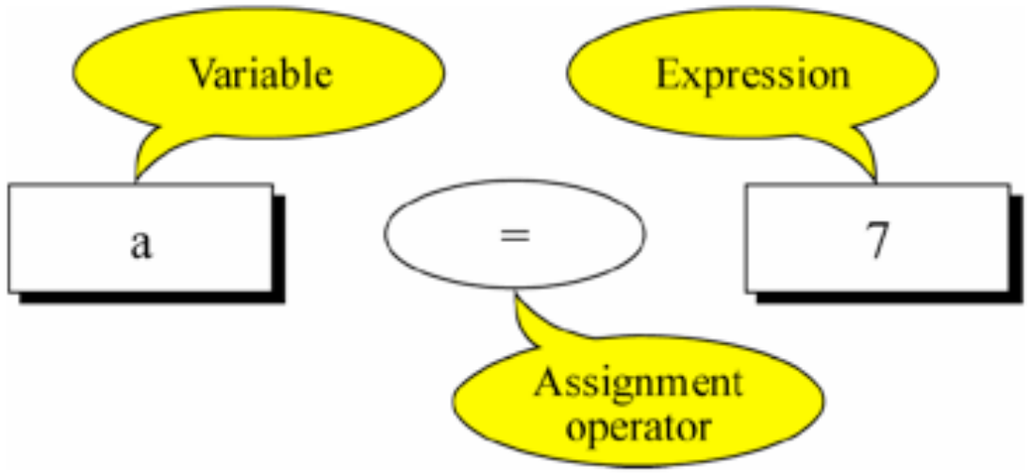
```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int i = 1;
    int j = ++i;
    printf("i = %d,j = %d\n",i,j);
    return 0;
}
```

程序运行结果如下：

```
i = 2,j = 2
```

注：对于自增或自减运算可采用“就近”原则，即谁在前先做什么，如果++或--在前，则先++或--运算，如果变量在前，则先取变量的值

# 2.3 赋值运算符和表达式




基本的赋值运算符是“=”，它的作用是将等号右边的表达式的值赋给等号左边的变量（左值）。

```c
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int a = 7;
    a = a+3;//先计算等号右边的值再通过赋值语句赋值给等号左边的变量
    printf("%d\n",a);//结果为10
    return 0;
}
```

注意：赋值运算是将原变量的值拷贝到新变量中（传值赋值），原变量的值改变时，新变量的值并不会随之改变，如：

```c
    int a = 5;
    int b = a;//是将a的值拷贝给b;
    a = 10;
    printf("b=%d\n",b);//结果仍为5;
```

# 2.4 二元复合运算符

二元复合运算符:在赋值符“=”之前加上其它双目运算符可构成复合赋值运算符。如+=、-=、*=、/ =、%=、<<=、>>=、&=、^=、|= 等，如: int i = 5; i+=3; 它等价于 i = i+3; 最终的i的值为 8;

```c
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int a = 5;
    a += 3;//等价于 a = a+3; 先计算a + 3的值，再赋值给a
    printf("a = %d\n",a);
    return 0;
}
```

程序运行结果如下:

```
a = 8
```

说明: 复合赋值运算符这种写法,对初学者可能不习惯,但十分有利于编译处理,能提高编译效率并产生质量较高的目标代码。

二元复合运算符练习：

| Contents of Variable x | Contents of Variable y | Expression | Value of Expression | Result of Expression |
|---|---|---|---|---|
| 10 | 5 | x *= y | 50 | x = 50 |
| 10 | 5 | x /= y | 2 | x = 2 |
| 10 | 5 | x %= y | 0 | x = 0 |
| 10 | 5 | x += y | 15 | x = 15 |
| 10 | 5 | x -= y | 5 | x = 5 |

# 2.5 一元运算符 sizeof

sizeof的作用就是返回某一数据类型在内存中占用的字节数，它是C/C++中的操作符而不是一个函数。

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int a = 5;
    printf("%d,%d,%d\n",
           sizeof(int),sizeof(a),sizeof(3.14));//3.14这个数值默认为double类型
    return 0;
}
```

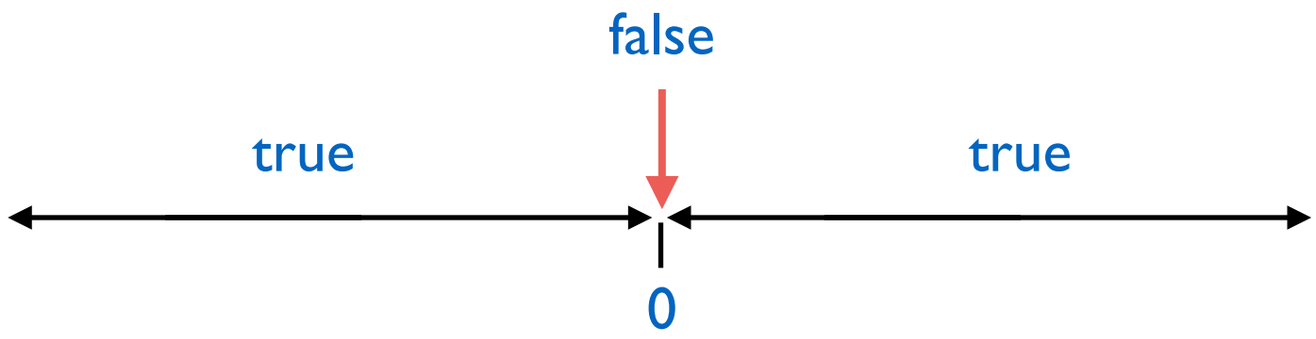
程序运行结果如下：

```
4,4,8
```

# 2.6 关系运算符和逻辑运算符

1、**逻辑数据：**表达式的值为true或false的数据

- 0 is false —— 0为假
- Everything else is true —— 非0为真

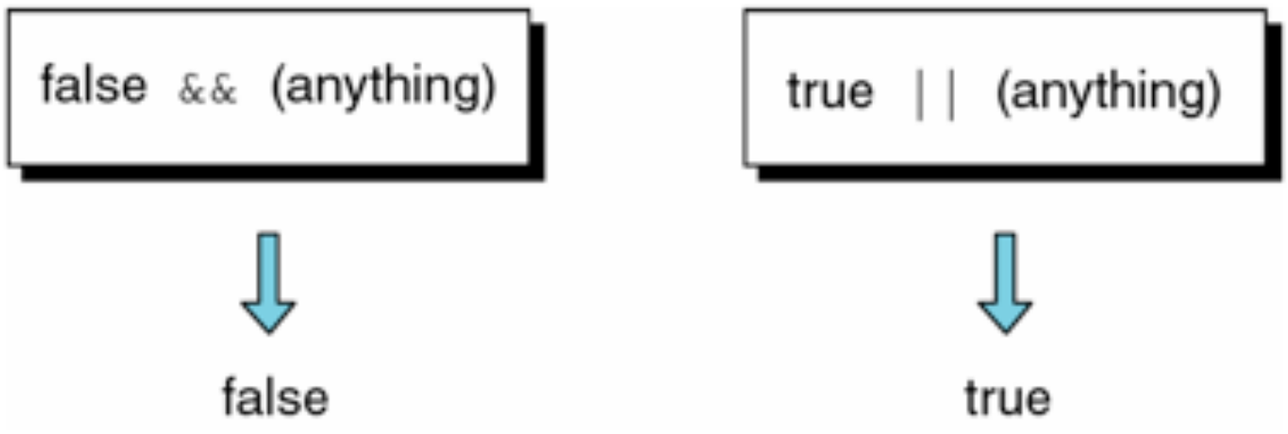


2、逻辑运算符：常见的逻辑运算符有“&&”和“||”，以及“!”。

(1) && 与：相当于数学中的且，如 a && b;

(2) || 或：如 a || b;

(3) ! 非：取反，如 ! a ;



逻辑运算符示例：

```
#include <stdio.h>
int main(int argc, const char * argv[]) {

    int a = 1;
    int b = 0;
    if (a && b) {//与：其中一个表达式为假则整个表达式为假
        printf(“Hello world!\n”);
    }else{
        printf(“Goodbye world!\n");
    }

    if (a || b) {//或：其中一个表达式为真则结果为真
        printf(“Hello Beijing!\n”);
    }else{
        printf(“Goodbye Beijing!\n”);
    }
    return 0;
}
```

程序运行结果如下：
Goodbye world!

程序运行结果如下：
Hello Beijing!

逻辑运算符真值表：

| 表达式a | 表达式b | !a | a && b | a || b |
| --- | --- | --- | --- | --- |
| 真 | 真 | 假 | 真 | 真 |
| 真 | 假 | 假 | 假 | 真 |
| 假 | 真 | 真 | 假 | 真 |
| 假 | 假 | 真 | 假 | 假 |

总结：1、逻辑与(&&)，全真为真，有假则假

2、逻辑或(||)，一真为真，全假为假

3、逻辑非(!)，取反，即表达式为真则取反为假，表达式为假，取反为真。

3、**关系运算符：**关系运算符和表达式:用于比较运算。包括大于(＞)、小于(＜)、等于(==)、大于等于(>=)、小于等于(<=)和不等于(!=)六种。

关系运算符，运算结果为布尔型（真**or**假）

| Operator | Meaning |
|---|---|
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| == | equal |
| != | not equal |

A＜B， A是否小于B?

当A = 5 ; B =10; 结果为true;

当A = 5 ; B =5; 结果为false;

当A = 15 ; B =10; 结果为false;

当A = 15 ; B =10; A!=B,结果为true。

# 2.6 条件表达式

示例代码：

```c
#include <stdio.h>
int main()
{
    int a = 5;
    int b = 10;
    int max = 0;

    if (a > b)      //if(条件表达式)
        max = a;    //如果条件表达式值为真，就执行if后面的语句
    else            //如果条件表达式值为假，就执行else后面的语句
        max = b;

    printf("max = %d\n", max );
    return 0;
}
```

程序运行结果如下：

```
max = 10
```

求两个数中的最小值或最大值改进:

三目运算符，计算机C语言的重要组成部分。条件运算符是唯一有3个操作数的运算符，所以有时又称为三元运算符。

如：对于条件表达式a>b ? x:y，先计算条件 a>b，然后进行判断。如果a > b的值为true，计算x的值，运算结果为x的值；否则，计算y的值，运算结果为y的值。

```c
#include <stdio.h>
int main()
{
    int a = 5;
    int b = 10;
    int max = 0;
    max = a > b ? a : b;
//三目运算符（? : ）,问号前面的表达式为真则取：前的值，否则取：后的值
    printf("max = %d\n", max );
    return 0;
}
```

程序运行结果如下:

```
max = 10
```

# 三、 运算符的优先级和结合性

- 运算符的优先级:

C语 中,运算符的运算优先级共分为15级。1级最高,15级最低。在表达式中,优先级较高的先于优先级较低的进行运算。而在一个运算量两侧的运算符优先级相同时,则按运算符的结合性所规定的结合方向处理。

- 运算符的结合性:

C语言中各运算符的结合性分为两种：左结合性和右结合性

例如：算术运算符的结合性是自左至右，即先左后右。
如有表达式x-y+z则y应先与“-”号结合，执行x-y运算，然后再执行+z的运算。这种自左至右的结合方向就称为“左结合性”。

而自右至左的结合方向称为“右结合性”。 最典型的右结合性运算符是赋值运算符。
如x=y=z，由于“=”的右结合性，应先执行y=z再执行x=(y=z)运算。
C语言运算符中有不少为右结合性，应注意区别，以避免理解错误。

# 运算符优先级和结合性一览表：

| 优先级 | 运算符 | 名称或含义 | 使用形式 | 结合方向 | 说明 |
|---|---|---|---|---|---|
| 1 | [] | 数组下标 | 数组名[常量表达式] | 左到右 | -- |
| | () | 圆括号 | (表达式）/函数名(形参表) | | -- |
| | . | 成员选择（对象） | 对象.成员名 | | -- |
| | -> | 成员选择（指针） | 对象指针->成员名 | | -- |
| | | | | | |
| 2 | - | 负号运算符 | -表达式 | 右到左 | 单目运算符 |
| | ~ | 按位取反运算符 | ~表达式 | | |
| | ++ | 自增运算符 | ++变量名/变量名++ | | |
| | -- | 自减运算符 | --变量名/变量名-- | | |
| | * | 取值运算符 | *指针变量 | | |
| | & | 取地址运算符 | &变量名 | | |
| | ! | 逻辑非运算符 | !表达式 | | |
| | (类型) | 强制类型转换 | (数据类型)表达式 | | -- |
| | sizeof | 长度运算符 | sizeof(表达式) | | -- |
| | | | | | |
| 3 | / | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| | * | 乘 | 表达式*表达式 | | |
| | % | 余数（取模） | 整型表达式%整型表达式 | | |
| 4 | + | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| | - | 减 | 表达式-表达式 | | |
| 5 | << | 左移 | 变量<<表达式 | 左到右 | 双目运算符 |
| | >> | 右移 | 变量>>表达式 | | |
| | | | | | |
| 6 | > | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| | >= | 大于等于 | 表达式>=表达式 | | |
| | < | 小于 | 表达式<表达式 | | |
| | <= | 小于等于 | 表达式<=表达式 | | |
| | | | | | |
| 7 | == | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| | ! = | 不等于 | 表达式!= 表达式 | | |
| | | | | | |
| 8 | & | 按位与 | 表达式&表达式 | 左到右 | 双目运算符 |
| 9 | ^ | 按位异或 | 表达式^表达式 | 左到右 | 双目运算符 |
| 10 | \| | 按位或 | 表达式\|表达式 | 左到右 | 双目运算符 |
| 11 | && | 逻辑与 | 表达式&&表达式 | 左到右 | 双目运算符 |
| 12 | \|\| | 逻辑或 | 表达式\|\|表达式 | 左到右 | 双目运算符 |
| | | | | | |
| 13 | ?: | 条件运算符 | 表达式1?<br>表达式2: 表达式3 | 右到左 | 三目运算符 |
| | | | | | |
| 14 | = | 赋值运算符 | 变量=表达式 | 右到左 | -- |
| | /= | 除后赋值 | 变量/=表达式 | | -- |
| | *= | 乘后赋值 | 变量*=表达式 | | -- |
| | %= | 取模后赋值 | 变量%=表达式 | | -- |
| | += | 加后赋值 | 变量+=表达式 | | -- |
| | -= | 减后赋值 | 变量-=表达式 | | -- |
| | <<= | 左移后赋值 | 变量<<=表达式 | | -- |
| | >>= | 右移后赋值 | 变量>>=表达式 | | -- |
| | &= | 按位与后赋值 | 变量&=表达式 | | -- |
| | ^= | 按位异或后赋值 | 变量^=表达式 | | -- |
| | \|= | 按位或后赋值 | 变量\|=表达式 | | -- |
| | | | | | |
| 15 | , | 逗号运算符 | 表达式,表达式,... | 左到右 | -- |

# 四、显式和隐式类型转换

**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。

一般情况下，数据的类型的转换通常是由编译系统自动进行的，不需要人工干预，所以被称为隐式类型转换。但如果程序要求一定要将某一类型的数据转换为另外一种类型，则可以利用强制类型转换运算符进行转换，这种强制转换过程称为显式类型转换。

（1）显式类型转换：

```c
#include <stdio.h>
int main()
{
    float pi = 3.14;
    int i = (int)pi;  //显式强制类型转换
    printf("i = %d\n",i);
    return 0;
}
```

程序运行结果如下：
i = 3

（2）隐式类型转换：

```c
#include <stdio.h>
int main()
{
    float pi = 3.14;
    int i = pi;  //隐式强制类型转换
    printf("i = %d\n",i);
    return 0;
}
```

程序运行结果如下：
i = 3

自动转换遵循以下规则:

(1) 若参与运算量的类型不同,则先转换成同等类型,然后进行运算。

(2) 在执行期间,低精度被自动转换为高精度。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 型运算时,先把 int 量转成 long 型后再进运算。

(3) char 型和 short 型参与运算时,必须先转换成 int 型。

(4) 在赋值运算中,赋值号两边的数据类型不同时,赋值号右边的类型将转换为左边的类型。如果右边的数据类型精度的长度长于左边长时,右边量将丢失一部分数据,这样会降低精度,类型转换不存在四舍五 。


# The End

渥瑞达·教学部