5 运算符和表达式

5.0 概述

5.0.0 运算符的种类

C语言的运算符大致分为 9 类:

- 算术运算符　用于各类数值运算，包括加(+)、减(-)、负号(-)、乘(*)、除(/)、求余(%)、自增(++)、自减(--) 8 种。
- 关系运算符　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(!=) 6 种。
- 逻辑运算符　用于逻辑运算，包括与(&&)、或(||)、非(!) 3 种。
- 位运算符　按二进制位进行运算，包括按位与(&)、按位或(|)、按位非(~)、按位亦或(^)、左移(<<)、右移(>>) 6 种。
- 赋值运算符　用于赋值运算，分为简单赋值(=)、复合算术赋值(+=, -=, *=, /=, %=)和复合位运算赋值(&=, |=, ^=, >>=, <<=) 3 类共 11 种。
- 条件运算符　用于条件求值(? :)。
- 逗号运算符　用于把若干表达式组合成一个表达式(,)。
- 指针运算符　用于下标[]、简介访问(*)、取地址(&)和成员访问(->或.) 5 种。
- 特殊运算符　括号()，所占内存字节数(sizeof)和强制类型转换运算符等 3 种。

5.0.1 运算符的特性

C语言的每个运算符都有其一些共有的性质

1. 运算符的操作数

   每个运算符所需的操作数（一般为表达式），一般称为“目”。根据操作数的个数，运算符可以分为单目运算符、双目运算符和三木运算符。大部分运算符都是双目，三目运算符只有条件运算符一个。对于双目运算符，两边的操作数必须类型相同，才能进行计算，不同类型的数据进行计算时才会进行自动类型转换，具体规则见 5.3 。

2. 运算符的优先级和结合性

   在由多个运算符组合而成的表达式中，存在一个计算顺序的问题。为了处理计算顺序，所有的运算符有其对应的优先级和结合性。C语言中运算符部分是数学中的运算符，其优先级和数学上的是相同的。

表 5 - 1 运算符的优先级与结合性

优先级	运算符	运算功能	目数	结合性
1	() [] . ->	提高优先级，函数调用 下标运算 直接成员访问 间接成员访问	 2 2 2	自左至右
2	! ~ ++ -- (类型名) * & sizeof -	逻辑非 按位求反 自增 自减 强制类型转换 间接访问 取地址 求类型或数据的长度 负号	1 1 1 1 1 1 1 1 1	自右至左
3	* / %	乘法 除法 求整数余数	2 2 2	自左至右
4	+ -	加法 减法	2 2	自左至右
5	<< >>	位左移 位右移	2 2	自左至右
6	< > <= >=	关系运算	2	自左至右
7	== !=	等于 不等于	2 2	自左至右
8	&	按位与	2	自左至右
9	^	按位亦或	2	自左至右
10	|	按位或	2	自左至右
11	&&	逻辑与	2	自左至右
12	||	逻辑或	2	自左至右
13	? :	条件运算	3	自右至左
14	= += -= *= /= %= >>= <<= &= ^= |=	赋值及复合赋值	2	自右至左
15	,	逗号运算	>=2	自左至右

3. 运算符的副作用

如果一个运算符运算后会改变操作数的值，就说这个运算符具有副作用（ sideeffect ）的，只有自加、自减和赋值运算符会有副作用。如果在一个表达式中存在副作用运算符，那么就要注意在同一个表达式中被修改的操作数的再次引用。这种表达式的计算结果是依赖于编译器，不同编译器可能会有不同的结果，比如说 (a++)+a 之类。

5.0.2 表达式的特性

表达式是由变量、常数、函数调用以及运算符组合而成的，特别地，单一的变量、常量和函数调用也可以是一个表达式。比如，变量 a 本身可以认为是一个表达式。所以的表达式有一些共有的性质。

1. 表达式的结果值

   C语言中所有的表达式都有结果值，而且这个结果值有其对应的数据类型（结果值的类型依赖于运算符和操作数的类型），不过有时不会使用这个结果值。比如“printf(“Hello World”);”语句中本身语句是有结果值的，且 printf 函数的返回值为 int 型，但很少使用。

2. 表达式的计算顺序

   当表达式中存在多个运算符时，需要知道那个运算符先，哪个后，不同的计算顺序会导致表达式的结果值不同。C语言中规定的计算顺序的规则如下：

   - 优先级高的运算符先算。
   - 相同优先级由结合性决定。所谓结合性就是，左结合就是先和左边的运算符运算，右结合就是先与右边的运算符运算。具体的运算符结合性见表 5 - 1 。
   - 部分运算符会强制计算顺序。比如逻辑与和或操作具有“短路”特性；逗号运算符会强制从左到右计算；选择运算符会优先计算逻辑判断表达式，然后再根据判断结果计算对应的表达式。

   但，最常用的还是：“括号的优先级是最高的”。要避免写因为计算顺序不同导致计算结果不同的表达式。

3. 左值和右值表达式

   表达式如果具有对应的可以由用户访问的存储单元，称之为左值表达式（简称左值），否则称之为右值表达式（简称右值）。C语言中，只有变量名、指针的简介访问、下标、成员访问表达式是左值，其他都是右值。

   具有副作用的运算符会限定操作数是左值的（赋值运算符只限定左边的操作数必须是左值的），比如说声明一个变量 a ，a++ 是可以的，因为 a 是左值的；但是 2++ 是不可以的，因为常量表达式不是左值的。又如 a=a+1 ，这个表达式中，等号左边的 a 是变量名表达式，是左值的，而等号右边的 a+1 这个加法表达式是右值，所以不能书写 a+1=2 这样的表达式。

5.1 赋值运算

在C语言中，符号“ = ”表示赋值的含义，不是数学里面的相等的含义，相等使用“ == ”表示，详见 5.4 关系运算 。赋值运算是一个双目运算符，它将右操作数的值赋给左操作数，使用时需要注意：

- 具有副作用，所以左操作数必须是左值的。
- 赋值表达式的结果值及其类型为赋值后左操作数的值及类型。
- 赋值是右结合的，所以 a=b=1 的计算顺序为先将 1 赋值给b，然后将 b=1 表达式的值赋值给 a。
- 赋值的左右操作数如果类型不相同，系统会进行自动类型转换，赋值操作的自动类型转换规则是以左操作数的类型为准。不同类型之间转换的规则比较复杂，总结如下：
  - 浮点数赋值给整数，舍弃掉小数部分。整数赋值给浮点数，数值不变，类型为左操作数类型。注意：浮点数赋值给整数时，由于浮点数表示可能存在精度误差，比如125.0可能表示为124.99…9，这时截断会导致精度问题。解决的方法为在对浮点对整型赋值时，加上一个很小的量（比如1e-6）再做赋值。
  - 有符号和无符号的整型之间赋值时，直接按内存中的补码进行赋值。例如

    ```
    unsigned int a=-1;
    ```

    此时a = 2^32-1 。
  - 长整型赋值给短整型时，会截断长整型补码的低位（低位的长度为短整型的长度）给短整型；短整型赋值给长整型时，短整型的补码会填充长整型的低位，高位部分补符号位（这样可以保持数值不变）。例如 int 型变量 a=(1LL<<32) ，啊的值为0,；long long型变量 a=-1 ，a的值为-1 。

- 字符型可以被认为是 1 个字节长的整型，按以上规则进行运算。但不同的编译器在处理时，有些（大部分）把字符型认为是有符号的，有些认为是无符号的。

5.2 算术运算

5.2.0 运算符

算术运算符中加、减、负号、乘法运算和数学中的含义相同。其中，减和负号都使用“ - ”，它们的区别在于减是双目的，负号是单目的；乘法运算使用的运算符是“ * ”（键盘上没有乘号” x “），千万不要按数学习惯不写乘号，那样会被编译器识别为一个标识符。

除法使用运算符“ / ”。需要注意，如果除法的两个操作数都是整型，那么它进行的是整数除，并且结果也是整数，所以3/2=1，而不是1.5 。如果需要得到浮点结果，要将其中一个操作数强制转换成浮点数类型，(double) 3/2 ，或者使用1.0*3/2 。当然，除法的除数是不能为0的，要检验除数，否则会产生错误。

模运算（即取余）使用运算符“ % ”，这个运算只能进行整数的模运算，浮点数的模运算要使用库函数 fmod() 。模运算的实质是除法，所以模数也不能为0 。另外，负数取模的结果仍然是非正的，-4%3 的结果为 -1 ；如果想得到其非负余数的话，可以使用 ((a%m)+m)%m 的方法。

自加运算使用运算符“ ++ ”，两个加号连写。这是一个有副作用的单目运算符，所以操作数必须是左值。自加运算符的含义是将操作数的值加上1，自加运算有两种写法，前缀（++a）与后缀（a++），其区别在于表达式的值不同，前缀表达式的值是自加完以后操作数的值，而后缀表达式的值是自加之前操作数的值。例如：

```
int a=1, b;
b = a++;         // b 的值为 a 自加之前的值，所以为 1
b = ++a;         // b 的值为 a 自加之后的值，所以为 3
```

自减运算和自加运算规则相同，只是加1变为减1 。注意，不要滥用自加或自减运算，尽量不要使用于编译器相关的表达式 。

5.2.1 溢出问题

算术运算需要注意数据溢出的问题。因为所有的数据类型的表示范围是有限的，所以算术运算中，运算结果可能会超出数据类型表示的范围，产生溢出（Overflow）。溢出后的结果对于程序来说毫无意义。

要避免溢出只能使用表示范围更大（或者精度更高）的数据类型，如果结果仍然超出 C 语言中所有的数据类型的表示范围（或精度）的话，只能使用数组模拟进行算术运算的方法（也被称为大数运算或高精度计算）来确保计算结果的正确（或精度要求）。

5.2.2 算术表达式值的类型

算术表达式值的类型为其操作数的类型，对于双目运算符而言，如果两个操作数的类型不同时，系统会自动把其中一个进行类型转换，使得两个操作数类型一致后再进行运算。自动转换的规则是按编译系统自动类型转换规则进行的，具体见下节。

5.3 类型转换运算

转换的方法有两种，自动类型转换与强制类型转换。

5.3.0 自动类型转换

自动类型转换，也叫隐式类型转换，由编译系统自动完成，无需使用者介入。一般的转换规则遵循两条：

- 等浮点数和整数混合计算时，整型转换成浮点型。
- 当低精度数据和高精度数据混合计算时，低精度数据转换成高精度数据类型。

  其具体转换规则如下所示：

图 5 - 1 自动转换规则示意图（以表代图）

$\leftarrow$ 为必然进行的转换		**$\Leftarrow$ 为必要时进行的转换**
double	$\leftarrow$	float
$\Uparrow$		
unsigned long long		
$\Uparrow$		
long long		
$\Uparrow$		
unsigned int	$\leftarrow$	unsigned short
$\Uparrow$		
int	$\leftarrow$	char，unsigned char，short

5.3.1 强制类型转换

强制类型转换，也叫显示类型转换，由使用者完成转换，强制类型运算符为“(类型名)”，圆括号不能少，其后接一个表达式，表达式值的类型会被强制转换成指定的类型。即

(类型名) 表达式

例如，(int)3.2 的值为 3 ，类型为整型；(double)3 的值为 3.0 ，类型为浮点型。

5.4 关系运算

关系运算是计算操作数之间的大小关系，所以的关系运算符都是双目的，一个6个，分别为 == (等于) 、 != (不等于) 、 > (大于) 、 < (小于) 、 >= (大于等于) 、 <= (小于等于) 。注意，等于的运算符与数学上的等于是不同的，由两个等号链接而成，而不是一个。

关系运算符与操作数构成关系表达式，表达式的值为 int 型，值只有两个，0和1 。如果操作数符合关系运算符的判断，那么表达式的值为 1 ，否则为 0 。

关系运算符应该避免在不同类型数据之间进行运算，如果左右操作数类型不同，会进行自动类型转换后比较，所以有时会出现一些错误的结果，例如：

```c
#include <stdio.h>
int main()
{
    int x = -1;
    unsigned int y = 1;
    printf("%d\n", x>y);
    // 输出结果为 1，但这个结果是错误的，因为 x 是 int 型，而 y 是 unsigned int 型，两边数据类型不一致，进行自动类型转换，x 转换为 2^32-1 ，所以才会大于 1 。
    return 0;
}
```

对于浮点数的关系运算，由于浮点数存在精度表示误差的问题，所以应该避免直接比较运算。一般的做法是使用一个精度控制的常量，误差在一定精度范围内认为两个浮点数是相等的。例如：

```c
#include <stdio.h>
#define EPS (1e-6)
int main()
{
    double x, y;
    scanf("%lf%lf", &x, &y);
    if (x>y+EPS) {puts("greater");}
    else if (x<y-EPS) {puts("less");}
    else {puts("equal");}
    return 0;
}
```

5.5 逻辑运算

逻辑运算是判断命题是否成立的运算，一个有 3 种，即逻辑非(!)、逻辑与(&&)、逻辑或(||)。逻辑表达式的结果值为 int 型，命题成立为值为 1 ，否则为 0 。注意，C语言中，在逻辑判断时，逻辑运算的操作数是 0 ，被认为是假，操作数非 0 都认为是真，而不仅仅是 1 。

表 5 - 2 逻辑非运算的真值表

a	!a
非0	0
0	1

表 5 - 3 逻辑与运算的真值表

a	b	a&&b
0	0	0
0	非0	0
非0	0	0
非0	非0	1

表 5 - 4 逻辑或运算的真值表

a	b	a || b
0	0	0
0	非0	1
非0	0	1
非0	非0	1

逻辑运算经常与关系运算、算术运算组合使用，用于分支语句和循环语句中逻辑判断表达式。例如判断一个字符变量 c 是否为英文字母，可以使用这样的表达式：

```
c>='A' && c<='Z' || c>='a' && c<='z'
```

判断一个整型变量 year 代表的年份是否为闰年可以使用这样的表达式：

```
year%400==0 || year%100!=0 && year%4==0
```

由于非 0 表示真的关系，所以也可以写成 `!(year%400) || year%100 && !(year%4)`。

注意，逻辑与和或运算时会强制先计算右操作数的值。因为逻辑与中，如果左操作数为0，那么表达式一定为0，这样就不会计算右操作数了；同理，逻辑或中，如果左操作数为非0，那么表达式一定为 1，同样也不会计算右操作数。称之为"短路"。例如：

```
int a = 2, b = 1, c = 3;
a < b && c++;
```

因为 a<b 的值为 0，所以在左操作数 a<b 计算完以后，逻辑与表达式的值就为 0 l（虽然并没有用整个表达式的值），而右操作数 c++ 是不会被计算的，所以 c 的值仍然是 3。

5.6 位运算

数据再内存中都是以2进制的形式存在的，C 语言提供位运算，可以对数据的位（bit）进行操作。位运算一般用于整型数据，但是也可以用于浮点数据（只要可以理解浮点数据的二进制运算）。整型数据再内存中是以补码形式存在的，位运算是按位对数据的补码进行计算。由于二进制过于冗长，一般用十六进制（或八进制）来表示，使用位运算需要熟悉十六进制下数据的表达方式，如下表所示。

表 5 - 5 10,2,16进制的数据对照表

10进制	2进制	16进制	10进制	2进制	16进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

C语言一共提供6种位运算符，下表列举了这6种运算的符号及含义。

表 5 - 6 位运算符

运算符	含义	说明
~	按位取反	单目，操作数的二进制中0变1，1变0
$	按位与	双目，左右操作数的二进制对位进行与运算
|	按位或	双目，左右操作数的二进制对位进行或运算
^	按位异或	双目，左右操作数的二进制对位进行异或运算
<<	左移	双目，左操作数值向左移动右操作数值个位，右边补0
>>	右移	双目，左操作数值向左移动右操作数值个位，左边补0或1

按位取反举例如下：

表达式	第15~8位	第7~0位	16进制
1	0000 0000	0000 0001	0001
~1	1111 1111	1111 1110	FFFE

按位取与举例如下：

表达式	第15~8位	第7~0位	16进制
5	0000 0000	0000 0101	0005
6	0000 0000	0000 0110	0006
5&6	0000 0000	0000 0100	0004

按位取或举例如下：

表达式	第15~8位	第7~0位	16进制
5	0000 0000	0000 0101	0005
6	0000 0000	0000 0110	0006
5|6	0000 0000	0000 0111	0007

左移举例如下：

表达式	第15~8位	第7~0位	16进制
1	0000 0000	0000 0001	0001
1<<3	0000 0000	0000 1000	0008
-1	1111 1111	1111 1111	FFFF
-1<<3	1111 1111	1111 1000	FFF8

右移举例如下：

表达式	第15~8位	第7~0位	16进制
100	0000 0000	0110 0100	0064
1>>3	0000 0000	0000 1100	000C
-100	1111 1111	1001 1100	FF9C
-1>>3	1111 1111	1111 0011	FF9C

右移运算，如果操作数是有符号数，不同的编译器对于左边补0或1，处理可能不一样，大部分编译器会补符号位。

编程中常使用一个整数的位来表示一个状态，使用位运算来读取或改写对应位的值，其中常常使用另外一个整数作为掩码（需要处理位为1，其他位为0的整数）来屏蔽掉不需要处理的位。常用位运算举例如下：

操作	表达式	举例
flag 的第 n 位是否为1	flag & (1 << n)	0x88&(1<<4) 的值为 0x08
将 flag 的第 n 位清0	flag &= ~(1 << n)	flag 初为 0x88, n 为 4, flag 终为 0x80
将 flag 的第 n 位清1	flag |= (1 << n)	flag 初为 0x88, n 为 3, flag 终为 0x8C
将 flag 的第 n 位翻转	flag ^= (1 << n)	flag 初为 0x88, n 为 4, flag 终为 0x80
判断 n 是奇数	n & 1	n 为 3, n&1 为 1
判断 n 是偶数	~n & 1	n 为 2, ~n&1 为 1
整数 x 乘以2的 n 次幂	x << n	x 为 1, n 为 4, x<<n 的值为 16
整数 x 整除2的 n 次幂	x >> n	x 为 10, n 为 2, x>>n 的值为 2
整数 x 对2的 n 次幂取模	x&(1<<n)-1	x 为 10, n 为 3, 表达式的值为 2
交换值	a^ = b^ = a^ = b	a 和 b 也可以是浮点类型
整数 a 和 b 的均值	((a^b) >> 1) + (a&b)	保证计算中间结果不会溢出
计算 x 最后一个位为1的数	x & -x	x 为 6, 表达式结果为 2
32位有符号整数 x 的绝对值	(x^(x>>31)) - (x>>31)	

5.7 复合赋值运算

在程序设计中，诸如 x = x + y 之类的表达式是司空见惯的，这类运算的特点是：参与运算的量既是运算分量，又是存储对象。为避免对同一存储对象的地址重复计算，提高编译效率，C语言引入复合赋值运算符。凡是双目算术、位操作运算符都可以与赋值运算符组合成复合赋值运算符，C语言一个提供了 10 种复合赋值运算符： += 、 -= 、 *= 、 /= 、 %=、 <<= 、 >>= 、 &= 、 ^= 、 |= 。

一般记某个复合赋值预算符为“ θ =”，那么表达式 a θ = b 等价于 a = a θ (b) ，所以 a += b 等价于 a = a+b ，其他复合赋值运算符类似。

5.8 逗号运算

在 C 语言中，逗号表达式就是将若干个独立的表达式用逗号结合成一个表达式，即逗号表达式。逗号表达式会强制从左到右计算各个子表达式，最后一个子表达式的值为整个逗号表达式的值。注意，逗号运算符的优先级是最低的，所以表达式 `a = 1, 2;` 中第一个子表达式为 `a = 1` ，第二个表达式为2，所以运算完毕以后，a 的值为 1 ，整个逗号表达式的值为 2 。

5.9 sizeof运算

sizeof 运算符是一个单目运算符，它返回常量、变量、类型和数组的所占内存的字节数。它有以下三种形式：

- sizeof(类型名)
- sizeof(常量、变量名或数组名)
- sizeof 常量、变量名或数组名

很多人会认为 sizeof 是一个函数，但其实它是一个运算符，而且 sizeof 表达式的值在编译期间就已经确定，在运行期就是一个常量，不需要进行计算。

```c
#include <stdio.h>
int main()
{
    int ia[10], ib;
    char ca[10], cb;
    float fa[10], fb;
    double da[10], db;
    long long la[10], lb;
    printf("类型\t\t数组\t变量\t类型\t常量\n");
    printf("int\t\t%d\t%d\t%d\t%d\n", sizeof(ia), sizeof(ib), sizeof(int), sizeof(1));
    printf("char\t\t%d\t%d\t%d\t%d\n", sizeof(ca), sizeof(cb), sizeof(char), sizeof('a'));
    printf("float\t\t%d\t%d\t%d\t%d\n", sizeof(fa), sizeof(fb), sizeof(float), sizeof(1.0f));
    printf("double\t\t%d\t%d\t%d\t%d\n", sizeof(da), sizeof(db), sizeof(double), sizeof(1.0));
    printf("long long\t%d\t%d\t%d\t%d\n", sizeof(la), sizeof(lb), sizeof(long long), sizeof(1LL));
    /* 运行结果
类型            数组    变量    类型    常量
int             40      4       4       4
char            10      1       1       4
float           40      4       4       4
double          80      8       8       8
long long       80      8       8       8
```

```
    */
    // 这样就得到了不同类型的变量与常量所占内存的字节数，其中 char 型常量参入计算时会被隐式转换成 int 型，所以它所占内存与 char 型变量不同，而且对于不同位的编译器，这些数值会发生变化，所以如果程序存在移植的可能，就要注意这些变化。
    return 0;
}
```

5.10 其他运算符

取地址运算符 & ，是一个单目运算符（& 做双目运算符时是按位与计算，见 5.6），后面接一个左值表达式，比如说 &a ，表示取变量 a 的地址。例如在使用 scanf() 函数经常使用到如 scanf(“%d”, &a); 类似的语句。

间接访问运算符将在后面指针中讨论；成员访问运算符将在复合数据类型中讨论；条件运算符等价于一个 if~else 语句，将在分支结构中讨论。

5.11 表达式与语句

所有的表达式后面加上分号都会构成一个语句，称其为表达式语句。但不是所有的表达式语句都有意义，比如C语言不反对以下类似的语句行：

```
    a + 2;
    a >= 1;
    ++i;
    a = a + 10;
    // 这四行语句都符合C语言中的语法规定，但前两行是无意义的，它们对程序的状态没有任何的影响，它们的存在合法而不合理，后两行在特定的前后环境里会使程序发生改变，它们是合理的，它们的存在才有意义。
```