第 5 章

CHAPTER 5

运算符和表达式

运算符的优先级和结合性会决定表达式的运算次序，而表达式的值又是如何按照运算符的性质决定计算顺序的呢？此类问题的解决有什么既定的逻辑规律？

5.1 优先级和结合性

一个表达式中可能包含多个不同的运算符，它们可以连接不同的数据，从而构成表达式。表达式中当各个式子采用不同的运算顺序时，往往会使表达式得出不同的结果，甚至出现运算错误，所以当表达式中含有多种运算时，数据和运算符必须按一定顺序进行结合，这样才能保证运算的合理性和结果的唯一性。

每种运算符都有其相应的优先级，优先级决定哪一种运算先被执行。在 C 语言中运算符的优先级可使用一张表格表示，称为 C 语言运算符优先级表，优先级表中优先级从上到下依次递减。

表达式中的运算结合次序，主要取决于表达式中各种运算符的优先级，优先级高的运算符先结合，优先级低的运算符后结合。在 C 语言运算符优先级表中，由于同一行中的运算符的优先级相同，所以在运算符优先级相同的情况下，数据和运算符的结合顺序由运算符结合性决定。

5.1.1 优先级表

C 语言运算符优先级表，共有 15 个优先级，很多运算符的优先级是一样的，见表 5-1。

表 5-1 C 语言运算符优先级表

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		

续表

优先级	运算符	名称或含义	使用形式	结合方向	说明
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
	＞＝	大于或等于	表达式＞＝表达式		
	＜	小于	表达式＜表达式		
	＜＝	小于或等于	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? 表达式 2：表达式 3	右到左	三目运算符

续表

优先级	运算符	名称或含义	使用形式	结合方向	说明
14	=	赋值运算符	变量=表达式	右到左	赋值运算符
	/=	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	

以上表格就是C语言运算符优先级表，其详尽地列出了所有的C语言运算符优先级和结合性级别，并附有简要说明。此表非常重要，在实际的编程工作中可能要经常查阅，建议读者对其内容应有一定的熟练度。

5.1.2　左结合和右结合

优先级表描述了C语言运算符的优先级等级，还有很多的C语言运算符的优先级是相同的，如果它们都出现在同一表达式中，则究竟哪个运算应先被执行呢？为解决此类问题，便引申出了运算符结合性的概念。

有以下示例，代码如下：

```
int a = 0, b = 1, c = 2;
a = b = c;
```

这里“=”运算符的优先级相同，究竟是先计算哪一个等式呢？是先a=b，还是先b=c？如果是前者，则a的值为1，如果是后者，则a和b的值都为2。

稍微有点编程经验就很容易看出，a、b、c的值最后都是2。这是因为“=”运算符的右结合性，最右边的值会向左边传递，最后让a、b、c都为2。具体是b=c先被执行，然后a=b再被执行。

这种从右向左开始计算的运算符，其结合性称为右结合，反之就是左结合。

C语言中具有右结合性的运算符包括所有单目运算符(～取反、！取非、－负号、＋正号、＋＋自增、－－自减)、赋值运算符和三目条件运算符(?:)，其他的运算符都是左结合性质。

注意：C语言中唯一的三目运算符是“?:”，用法为A? B: C，其逻辑为如果A的逻辑值为真，就执行B，否则就执行C并返回所执行表达式的结果。其实用if-else结构也可以实现此类逻辑，但由于此运算符很简洁，并且类似的逻辑在代码中很常见，所以C语言给出了专用运算符。

5.1.3 表达式的值

综合以上内容，确定一个 C 语言表达式的运算顺序，其步骤如下。

(1) 先根据运算符的优先级，确定哪些运算先被执行。

(2) 在优先级相同的情况下，再根据结合性确定哪些运算先被执行。

有以下表达式示例，代码如下：

```
a = b = !c * f + a - b/(d - g);
```

在上式中，因为“()”的优先级最高，所以先计算(d－g)，然后因为“!”的优先级别比算术运算符的优先级高，所以再计算!c，最后因为乘除的优先级比较高，所以计算!c * f 和 b/(d－g)，最后计算!c * f 加 a 的值，再减去 b/(d－g)的值。至此“＝”表达式右边已经计算完成(算术运算符的优先级大于赋值运算符)，这个值先传给 b(“＝”运算符右结合)，再传给 a。

以上表达式的运算步骤如下：

(1) 计算(d－g)的值，因为括号运算符的优先级最高。

(2) 计算!c 的值，因为逻辑非运算符的优先级次高。

(3) 分别计算!c * f 和 b/(d－g)的值，先进行乘除运算。

(4) 计算!c * f＋a－ b/(d－g)的值，后进行加减运算。

(5) 赋值 a＝b＝!c * f＋a－b/(d－g)，运算完成。

在上述分析中，每个阶段的计算完后都已经有了计算结果，这个中间结果在程序中没有体现，是 C 语言编译器自动处理的，本书将其称为“临时变量”，类似小学时学四则运算，运算时要把中间结果记录下来，以便于进行下一步运算。临时变量的长度往往会影响运算结果，但是 C 语言编译器隐藏了这些细节，临时变量详见 12.4 节。如果对这些细节不明白，则编写的 C 程序的计算结果会出错。

把上式代入：c＝1，f＝10，a＝3.14，b＝7.9，d＝5，g＝1.414，使用计算器手工计算的结果如图 5-1 所示。

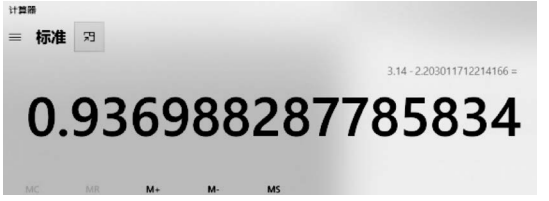

图 5-1　计算器的计算结果

用程序进行计算，查看计算结果和手工计算是否一致，代码如下：

```
//2.2.1/test_3.0.c
#include <stdio.h>
```

```
int main(void)
{
        double c = 1,f = 10,a = 3.14,b = 7.9,d = 5,g = 1.414;          //为什么使用 double
        a = b = !c * f + a - b/(d - g);

        printf("a = %0.15f, b = %0.15f,\n", a, b);
        return 0;
}
```

命令如下：

```
root@Ubuntu:~/C_prog_lessons/lesson_2.2.1#./test_3.0
a = 0.936988287785834, b = 0.936988287785834,
```

和使用计算器的计算结果是完全一样的。读者可以把 double 换成 float，查看结果有什么不同。

5.2　表达式中的隐式规则

如何写出一个能按照编程者意图输出的表达式，不仅要深刻理解 C 语言运算符优先级和结合性的概念，以及由此推导出的表达式运算逻辑，也必须明白 C 语言表达式中的一些隐晦的类型转换规则。

5.2.1　整型提升

C 语言的整型算术运算总是默认以整型类型的精度来进行计算的。为了获得这个精度，表达式中的字符(char)和短整型(short)操作数在使用之前要被转换为普通整型，这种转换被称为“整型提升”。

整型提升的意义：表达式的整型运算要在 ALU 中执行，ALU 操作数的字节长度一般默认为 int 类型长度，它往往也是 CPU 通用寄存器的长度。

因此，即使是两个 char 类型的数据相加，在 CPU 中执行时也要先转换为 ALU 默认的整型长度，这也导致通用 CPU 难以实现两个 8 位数的直接相加运算(虽然机器指令中可能有这种字节相加指令)，所以表达式中各种长度小于 int 长度的整型值都必须先转换为 int 或者 unsigned int 类型，然后才能送入 CPU 中执行运算，而这种转换是由编译器自动完成的。

新建示例，代码如下：

```
//2.2.2/test_1.0.c
#include <stdio.h>

int main(void)
{
```

```
        char a = 1;
        printf("a length is %ld \n", sizeof(a));              //输出 a 的长度
        printf("temp length is %ld \n", sizeof(a * 3));       //输出 a * 3 的长度
        return 0;
}
```

查看运行结果，命令如下：

```
root@Ubuntu:~/C_prog_lessons/lesson_2.2.2# ./test_1.0
a length is 1
temp length is 4
```

由运行结果可以得知，a 的长度是 1 字节，但是 a×3 的运算结果的长度却成了 4 字节。由程序的运行结果得出，a×3 运算的（中间）数据和 int 类型长度一致，这就是隐式整型提升。

5.2.2 隐式转换

C 语言编译器在以下 4 种情况会自动进行隐式转换：

（1）在算术运算中，低长度类型会被转换为高长度类型。

（2）在赋值表达式中，右边表达式结果值的数据类型会被自动隐式地转换为左边变量的类型，并赋值给它。

（3）当函数参数传递时会隐式地将实参类型转换为形参的类型，再赋给形参。

（4）当函数有返回值时会隐式地将返回表达式结果值的数据类型转换为返回值类型。

在算术运算中，有以下类型转换规则：

（1）字符必须被转换为整数（字符型和整型数据可通用，char 和 int 可直接运算）。

（2）short 型转换为 int 型（同属于整型）。

（3）float 型数据在运算时一律转换为双精度（double）类型，以提高运算精度（同属于实型）。

其次，当不同类型的数据进行运算时，应先将其转换为相同的数据类型再进行操作，转换规则同样是由低长度向高长度转换。