

第四章

表达式

● 本章要点

- 算术运算符
- 赋值运算符
- 自增和自减运算符
- 表达式求值
- 表达式语句

表达式

- C语言更多强调表达式（如何计算值的公式）

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}, s = \frac{a+b+c}{2}$$

- 操作数（变量、常量）和运算符构成
 - $a * x^2 + b * x + c$
 - $+$, $*$: 运算符
 - 操作数 $a * x^2$, x , c 各自都是表达式
- 基本运算符:
 - 算术运算符
 - 关系运算符
 - 逻辑运算符

4.1 算术运算符

- 5个二元(两个操作数)算术运算符：
 - + 加 (addition)
 - - 减 (subtraction)
 - * 乘 (multiplication)
 - / 除 (division) :
 - % 取余 (remainder) , mod
 - $10 \% 3$
 - $12 \% 4$
- %应用
 - 密码学, RSA, 密码分配Diffie-Hellman算法
 - 麻将抓牌方位 (筛子: 2-12)
 - 车牌限行, 1,2,3,4,5,6,7,8,9,0

4.1 算术运算符

- 两个一元算术运算符：
 - + 一元正号运算符：无操作
 - $i = +i;$
 - - 一元负号运算符：
 - $j = -i;$

二元算术运算符

- 操作数可为整数或浮点数，也可混合。
- 混合时结果为浮点数（精度自适应攀高）
 - $9 + 2.5f$?
 - $6.7f / 2$?

/ 和 % 运算符

- /: 两操作数都为整数
 - 结果取整
 - $1/2=?$ 0
- %: 操作数必须为整数
 - 否则编译无法通过。
- 0做右操作数（除数）
 - 导致未定义行为
- /和 %用于负操作数，C89中由实现定义（implementation-defined）
 - 结果可向上取整，也可向下取整。 $-9/7=-1$ 或 -2
 - 可正可负
- 在C99中，/的结果是向零截取的。

由实现定义的行为

Implementation-Defined Behavior

- C标准故意对部分内容未加指定
 - 细节由“实现”（特定平台上的编译、链接和执行软件）来定义。
- 尽量避免编写依赖于实现定义行为的程序。

运算符的优先级

- 算数运算符的优先级如下：
 - 最高优先级：+ - (一元)
 - * / %
 - 最低优先级：+ - (二元)
- 例子：
 - $i + j * k$ 等价于 $i + (j * k)$ 、
 - $-i * -j$ 等价于 $(-i) * (-j)$
 - $+i + j / k$ 等价于 $(+i) + (j / k)$
- 允许圆括号分组表达式
 - $(i + j) * k$

运算符的结合性

- 表达式包含两个或多个**相同优先级**运算符，需**结合性**。
- **左结合性**：
 - 运算符从左向右结合。
 - 二元运算符 ($*$, $/$, $\%$, $+$, 和 $-$) 是左结合
 - $i - j - k$ 等价于 $(i - j) - k$
 - $i * j / k$ 等价于 $(i * j) / k$
- **右结合性**：
 - 运算符从右向左结合。
 - 一元运算符 ($+$ 和 $-$) 是右结合

先左结合，不行在右结合

例子: 计算UPC校验位



- 美国和加拿大的货物使用通用产品代码 (Universal Product Code) (12位) :
 - 第一个数字: 表示种类。
 - 第一组五个数字: 标识生产厂商。
 - 第二组五个数字: 标识产品
- 最后一个数字: 校验位 (ck)
 - $ck = f$ (前11位)

例子: 计算UPC校验位



- 校验位计算

- $\text{checksum} = 9 - (\text{first_sum} * 3 + \text{second_sum} - 1) \% 10;$
- 奇位和first_sum: 第1、3、5、7、9、11位的数字相加
 - $0 + 3 + 0 + 1 + 1 + 3 = 8$
- 偶位和second_sum: 第2、4、6、8、10位的数字相加
 - $1 + 8 + 0 + 5 + 7 = 21$
- 奇位和乘3，加偶位和得total。
 - $8 * 3 + 21 = 45$
- 把结果减1，再除10取余数。
 - $(45 - 1) \% 10 = 4$
- 最后用9减余数，得到校验位。
 - $9 - 4 = 5$

例子: 计算UPC校验位 (upc.c)

- 11位的UPC要求用户分三步录入:

Enter the first (single) digit: 0

Enter first group of five digits: 13800

Enter second group of five digits: 15173

Check digit: 5

- 声明11个整数变量存放分别存储UPC的前11位。
- 厂商标识、产品标识不是按5位数来读取，而按5个1位数读入。
- 使用scanf的 %1d 转换说明，只匹配1位整数。

```
/* Computes a Universal Product Code check digit */
#include <stdio.h>
int main(void)
{
    int d, i1, i2, i3, i4, i5, j1, j2, j3, j4, j5, first_sum, second_sum, total;
    printf("Enter the first (single) digit: ");
    scanf("%1d", &d);
    printf("Enter first group of five digits: ");
    scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);
    printf("Enter second group of five digits: ");
    scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4, &j5);
    first_sum = d + i2 + i4 + j1 + j3 + j5;
    second_sum = i1 + i3 + i5 + j2 + j4;
    total = 3 * first_sum + second_sum;
    printf("Check digit: %d\n", 9 - (total - 1) % 10);
    return 0;
}
```

4.2 赋值运算符

- 简单赋值运算符= (Simple assignment)
- 复合赋值运算符 (Compound assignment)

4.2.1 简单赋值

- 赋值：把值赋予，与“相等：==”不同

- 两者混淆是常犯错误

- $v = e$ ，求表达式 e 的值，把此值赋给 v .

- e 可以是常量、变量或更复杂的表达式：

```
i = 5;                /* i is now 5 */
```

```
j = i;                /* j is now 5 */
```

```
k = 10 * i + j;       /* k is now 55 */
```

- v 和 e 的类型不同，赋值运算符会把 e 的值转换成 v 的类型(v 水晶鞋，削足适履)：

```
int i; float f;
```

```
i = 72.99f;           /* i is now 72 */
```

```
f = 136;               /* f is now 136.0 */
```


简单赋值

- 在很多编程语言中，赋值是语句
- 在C中赋值和+一样是运算符
 - “ $v = e$ ”也是表达式
 - “ $v = e;$ ”变成语句。
- 赋值表达式结果：赋值后 v 的值。

```
int i;
```

```
i = 72.99f;
```

表达式 $i = 72.99f$ 的值为 i 的值

副作用 (side effect)

- 运算符通常不修改操作数
 - 如: $a+5$
- 赋值运算符修改左操作数
 - $v = e$
- 修改操作数的值——运算符的副作用

简单赋值

- 多个赋值运算符可以串联在一起：
 - `i = j = k = 0;`
 - 结合性?
- 右结合，等价于：`i = (j = (k = 0));`
- “=” 串联，可能产生非预期效果：
 - `int i; float f;`
 - `f = i = 33.3f; //f=??`
- 嵌入式赋值：
 - `i = 1;`
 - `k = 1 + (j = i);`
 - `printf("%d %d %d\n", i, j, k); //??`
- 输出 "1 1 2"
- 不便于程序的阅读，也是隐含错误的根源。

默认double,
标f表示float

4.2.2 左值 (Lvalues)

- 赋值运算本质是数据入驻内存的过程，
- 要求左操作数必须是左值
 - 存储在计算机内存中的对象，eg，变量，数组元素
 - 不能是常量或计算的结果（表达式）。
- 下面例子非法：
 - $12 = i;$
 - $i + j = 0;$
 - $-i = j;$
 - $2*i = a;$
- 编译器会检测这种错误，给出错误消息：
 - “invalid lvalue in assignment.”

4.2.3 复合赋值

- 利用变量原值计算新值并又赋给该变量（更新）：

- 计算存款; $d=d*(1+i\%)$

- $i = i + 2;$

- C语言使用 += 复合赋值简化这种写法：

$i += 2;$ /* same as $i = i + 2;$ */

- 复合运算符-=, *=, /=, %= （其他的参见20章）

$i -= 2;$ /* same as $i = i - 2;$ */

$i *= 2;$ /* same as $i = i * 2;$ */

$i /= 2;$ /* same as $i = i / 2;$ */

$i \% = 2;$ /* same as $i = i \% 2;$ */

复合赋值

- 优先级低于算术运算符

$$i *= j + k$$

- 等价于 $i = i * (j + k)$ ，而不是 $i = i * j + k$
- 注意：
 - $i =+ j$ 相当于 $i = (+j)$ ，和 $i += j$ 完全不一样
- 结合性：和 $=$ 一样，右结合
 - $i += j += k$ 相当于 $i += (j += k)$

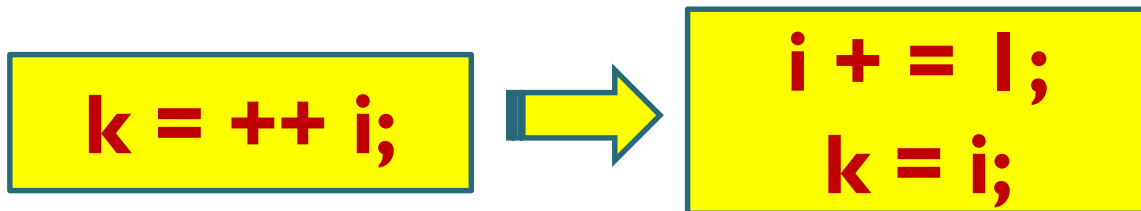
4.3 自增和自减运算符

- 自增（加1）和自减（减1）：
 - $i = i + 1;$
 - $j = j - 1;$
- 用复合赋值运算符可以简化为：
 - $i += 1;$
 - $j -= 1;$
- C提供更简化的 ++（自增）和--（自减）运算符
 - 可作前缀使用： $++i$ 和 $--i$
 - 也可作后缀使用： $i++$ 和 $i--$

自增\自減（前綴\后綴）

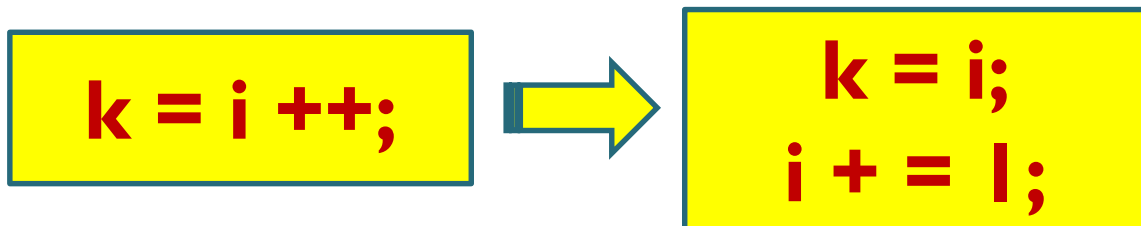
- 前綴：先变后用

- 先修改（自增或減），再取值（从内存中取出至CPU寄存器，使用变量值）：



- 后綴：先用后变

- 先取值再修改变量：



自增和自减运算符

- 前缀：先变后用——先修改（自增或自减）变量再取值（使用变量值）：

```
i = 1;
printf("i is %d\n", ++i);
printf("i is %d\n", i);
printf("i is %d\n", --i);
printf("i is %d\n", i);
```

```
/* "i is 2" */
/* "i is 2" */
/* "i is 1" */
/* "i is 1" */
```

- 后缀：先用后变——先取值再自增或自减变量：

```
i = 1;
printf("i is %d\n", i++);
printf("i is %d\n", i);
printf("i is %d\n", i--);
printf("i is %d\n", i);
```

```
/* "i is 1" */
/* "i is 2" */
/* "i is 2" */
/* "i is 1" */
```

自增和自减运算符

- 同一个表达式中多次使用 ++ 或 -- 结果难理解：

```
i = 1;
```

```
j = 2;
```

```
k = ++i + j++;
```

- i=2, j=3, k=4

```
i=1;
```

```
++i++; error: invalid lvalue in increment
```

```
++(i++)=++1
```

自增/自减操作数须为左值！

4.4 表达式求值（运算优先级排位）

优先级	名称	符号	结合性
1	（后缀）自增	++	左结合
	（后缀）自减	--	
2	（前缀）自增	++	右结合
	（前缀）自减	--	
	一元正号	+	右结合
	一元负号	-	
3	乘法类	* / %	左结合
4	加法类	+ -	
5	赋值	= *= /= %= += -=	右结合

先一后二，先左后右，赋值殿后

表达式求值

- 例子:

优先级

- $a = b += c++ - d + --e / -f$

- $a = b += (c++) - d + --e / -f$

1

- $a = b += (c++) - d + (--e) / (-f)$

2

- $a = b += (c++) - d + ((--e) / (-f))$

3

- $a = b += (((c++) - d) + ((--e) / (-f)))$

4

- $(a = (b += (((c++) - d) + ((--e) / (-f)))))$

5

子表达式的求值顺序

- 运算符的优先级和结合性规则
 - 分解复杂表达式为子表达式。
- 当有多个子表达式(并列)时
 - eg. $(a + b) * (c - d)$
 - 无法确定 $(a + b)$ 是否在 $(c - d)$ 之前求值。
- C没有定义表达式的求值顺序
 - 除含逻辑与(或)、条件、逗号等运算符的子表达式。

子表达式的求值顺序

- 无论计算顺序如何，大多数表达式值相同
eg. $(a + b) * (c - d)$
- 但当子表达式会改变某个操作数值时（赋值副作用），结果可能不一致：
a = 5;
c = (b = a + 2) - (a = 1);
先执行b=a+2，则b = 7，c = 6。
先执行a=1，则b = 3，c = 2。
- 上述语句的执行效果未定义——未定义行为
- 有些编译器会产生警告信息
“operation on 'a' may be undefined”

子表达式的求值顺序

- 为了避免此类问题，不在子表达式中使用赋值运算符，而是采用分离的赋值表达式：

```
a = 5;
```

```
b = a + 2;
```

```
a = 1;
```

```
c = b - a;
```

- 自增自减也可以改变操作数（副作用）：

```
◦ i = 2;
```

```
◦ j = i * i++;
```

```
◦ 使用本书义，可能具4式C
```

这里运算符优先级可确定计算顺序
等价于 $j = i++ * i;$

温故而知新——表达式

- 表达式构成
 - 运算符+操作数
- 基本运算符
 - 算术运算符： $\%(\text{mod})$, $7\%(\text{mod})3=?$
 - 关系运算符
 - 逻辑运算符
- 赋值运算符：数据存入内容
 - 简单赋值： $v=e$
 - 左值lvalue：具备可改写的内存空间的对象
 - v 、 e 类型不同： e 值转换成 v 类型(削足适履)
 - 复合赋值， $+=$, $-=$, $*=$, $/=$ ； $a+=b+2$ 、
- 自增\自减
 - 前缀：先变后取
 - 后缀：先取后变

温故而知新——运算优先级

优先级	名称	符号	结合性
1	(后缀) 自增	++	左结合
	(后缀) 自减	--	
2	(前缀) 自增	++	右结合
	(前缀) 自减	--	
	一元正号	+	右结合
	一元负号	-	
3	乘法类	* / %	左结合
4	加法类	+ -	
5	赋值	= *= /= %= += -=	右结合

先一后二，颠倒前后，赋值殿后，先左后右

实验

i=2;

printf("i*i++= %d\n",i*i++);

i=2;

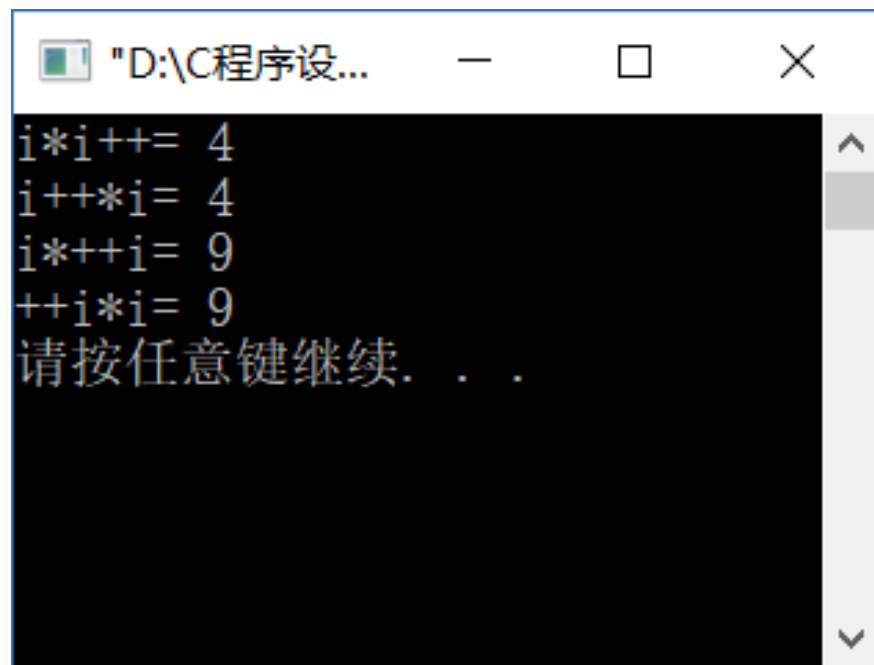
printf("i++*i= %d\n",i++*i);

i=2;

printf("i*++i= %d\n",i*++i);

i=2;

printf("++i*i= %d\n",++i*i);



```
"D:\C程序设..."  
i*i++= 4  
i++*i= 4  
i*++i= 9  
++i*i= 9  
请按任意键继续...
```

结果与乘数顺序无关，只与前后缀有关

这些例子说明的那样，`++i`意味着“立即自增`i`”，而`i++`则意味着“现在先用`i`的原始值，稍后再自增`i`”。这个“稍后”有多久呢？**Q&A** C语言标准没有给出精确的时间，但是可以放心地假设`i`将在下一条语句执行前进行自增。

未定义行为(undefined behavior)

- 和前面讲的由实现定义的行为不同。
 - 面包肉，实现定义：包子、饺子、抄手、馄饨等
 - 肉包面，未定义
- 其后果是严重的：
 - 不同的编译器给出的编译结果不同。
 - 程序可能无法编译。
 - 或编译了无法运行，运行了可能崩溃，不稳定或产生无意义结果。
- 应该避免未定义行为。

4.5 表达式语句

- C语言规定任何表达式都可以用作语句(加分号)
 - `++i;`
 - `3.14*5*5;`
- 不使用表达式的值(丢掉), 表达式语句没有意义, 除非有副作用:
 - `3+2*i;` `/* not useful */`
 - `i = i;` `/* useful */`
 - `i--;` `/* useful */`
 - `i * j - i;` `/* not useful */`

表达式语句

- 键盘上的误操作容易造成“什么也不做”表达式，比如：
 - 输入：`i = j;`，误输入为：`i + j;`
 - `if(i==0) fire();`——» `if(i=0) fire();`
- 某些编译器会检查无意义表达式，显示“statement with no effect.”警告。