# 计算机程序设计基础(1)
# --- C语言程序设计(4)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193
2022.10.

# 第4章 C表达式与宏定义

## 4.1 赋值运算

- 赋值运算符为"=" 。 赋值表达式为：变量名=表达式

功能是: 首先计算赋值运算符右边的表达式值，然后将计算结果赋给赋值运算符左边的变量，最后该赋值表达式的值也就是该运算结果。赋值表达式可以出现在另一个表达式中参与运算。

例如 假设x与y都是已定义的整型变量，

表达式 x=y=4+5 等价与 x=(y=4+5)

在执行这个表达式时，执行顺序是*自右至左*，首先计算赋值表达式(y=4+5)的值，即计算4+5的值为9，将计算结果赋给变量y，而赋值表达式(y=4+5)的值也为9；然后再将赋值表达式(y=4+5)的值（即9）赋给变量x。因此，通过这个赋值表达式将4+5的计算结果同时赋给了变量x与y。

也可以写成： y=4+5 ; x=y; 但不能写成： x=y; y=4+5;

- 赋值表达式的最后加一个“;”，就是赋值语句，
  赋值语句的形式为：　　变量名=表达式;

说明

1）在C语言中，“=”为赋值运算符，而不是一般意义的等号。
2）赋值运算符"="左边必须是变量名(左值)，不能是表达式。
3）赋值运算符"="两端的类型不一致时，系统将自动进行类型转换。
但编译器往往会给出警告信息，提示两端的类型不一致。

为了简化程序，提高编译效率，C语言允许在赋值运算符"="之前加上其他运算符，构成复合的赋值运算符，一般来说，凡是需要两个运算对象的运算符（即二元运算符），都可以与赋值运算符一起组成复合的赋值运算符。常用的复合算术赋值运算符有：

+=，-=，*=，/=，%=　　其中%为求余运算符

<变量> 复合赋值运算符 <算术表达式>

等价于：

<变量> ＝<变量> 算术运算符 (<算术表达式>)

即首先计算出右侧<算术表达式>的值，然后对左侧变量进行相应的运算。

例如：　　**a += b+3;**

等价于：　**a = a + (b+3);**

**a *= b+3;**

等价于：　**a = a * (b+3);**

**a %= b+3;**

等价于：　**a = a % (b+3);**

## 4.2 算术运算及其表达式

在解决数值型问题时，算术表达式是必不可少的。

在C语言中，基本的算术运算符有以下5个：

| | |
|---|---|
| + | 加法运算符（双目运算符），或正值运算符（单目运算符），如 3+z，+y。 |
| - | 减法运算符（双目运算符），或负值运算符（单目运算符），如 y-8，-z。 |
| * | 乘法运算符（双目运算符），如y*d。 |
| / | 除法运算符（双目运算符），如c/d。 |
| % | 求余运算符（双目运算符）。只适用于整型数据。 |

例如12%5的值为2，32%11的值为10，(-12)%5的值为-2，12%(-5)的值为2 ，(-12)%(-5)的值为-2 。

## 4.2 算术运算及其表达式

这些算术运算符的运算顺序与数学上的运算顺序相同。即：

① 先乘除后加减；

② 乘、除、求余运算优先级相同；

③ 加、减运算优先级相同；

④ 括号可以提高执行优先级；

⑤ 同一优先级运算自左至右执行。

例如：对于表达式a+b-c*d，不会因为*运算优先级高就先执行c*d，而是自左至右扫描表达式，由于b后面的-的运算优先级与b前面的+的运算优先级相同，因此先执行a+b得到r1，表达式变成 r1-c*d，继续自左至右扫描表达式，由于c后面的*的运算优先级高于c前面的-的运算优先级，因此再执行c*d，得到r2，此时表达式变成r1-r2，最后执行r1-r2得到表达式的最终结果。

对于表达式a+(b+e)-c*d，自左至右先执行(b+e)得到r1，表达式变成 a+r1-c*d，接下来的执行顺序与上例相同。

## 算术表达式是指用算术运算符将运算对象连接起来的式子

对于算术表达式要注意以下几个问题：

① 注意表达式中各种运算符的运算顺序，必要时应加括号，例如，

**(a+b)/(c+d) ≠ a+b/c+d。**

**(a*b)/(c*d) ≠ a*b/c*d。**

② 注意表达式中各运算对象的数据类型，特别是整型相除。C语言规定，两个整型量相除，其结果仍为整型。例如：

**7/6的值为1；**

**4/7的值为0；**

**(1/2)+(1/2)的值为0，而不是1。**

**int n; 当n大于1时，1/n 总是 0。**

③ C语言允许在表达式中进行混合运算，系统将自动进行类型转换，转换的原则是从低到高。

(低) int → unsigned int→ long → double (高)

↑必定转换　　↑必定转换

char或short　　float　(C语言统一用双精度运算)

特别需要说明的是，在混合运算过程中，系统所进行的类型转换并不改变原数据的类型，只是在运算过程中将其值变成同类型后再运算。

④ C语言提供了强制类型转换。强制类型转换的形式为：

(类型名)表达式

例如：　f = (double)1;　n = (long)(f*10);　m = (int)n;

- 虽然C语言允许在表达式中进行混合运算，并自动进行类型转换，但并不是将表达式中的所有量统一进行转换后才进行运算，而是在运算过程中逐步进行转换。

例如，为了计算表达式

**10/4+2.5**

的值，首先执行运算**10/4**，两个整型数据相除，其结果仍为整型，计算值为整型数**2**。然后执行运算**2+2.5**，发现类型不一致，进行类型转换，将整型数**2**转换成实型数**2.0**后再运算，即**2.0+2.5**，最后计算结果为实型数**4.5**。而并不是首先将表达式中的数据统一转换成实型数据后作运算**10.0/4.0+2.5**，这样计算结果就变为**5.0**了。实际上C语言的表达式不是这样执行的。

- 表达式中的数据是否转换成实型数，只看当前运算符两侧的数据项的类型，若其中一个为实型，而另一个不是，则把不是实型的转换为实型后进行运算。若运算符两侧的数据项的类型一致，则不做任何类型转换。

## 4.3 关系运算及其表达式

【例4-1】 已知A，B，C，D四个人中有一人是小偷，并且，这四个人中每个人要么说真话，要么说假话。在审问过程中，这四个人分别回答如下：

A说：B没有偷，是D偷的。

B说：我没有偷，是C偷的。

C说：A没有偷，是B偷的。

D说：我没有偷。

现要求根据这四个人的回答，写出能确定谁是小偷的条件。

- 假设用整型变量a，b，c，d分别代表A，B，C，D四个人，且变量只取值为0和1，值为1表示该人为小偷，值为0表示该人不是小偷。由于四个人中只有一人是小偷，而且不管是不是小偷，他的回答要么是真话，要么是假话。

A b+d=1　B b+c=1　C a+b=1　D a+b+c+d=1

数学表达式：b+d=1 且 b+c=1 且 a+b=1 且 a+b+c+d=1

C表达式：b+d==1 && b+c==1 && a+b==1 && a+b+c+d==1

- 上述条件中的符号“=”在C语言中是赋值运算符，不能作为判断是否“等于”来使用。
- C语言用两个连续的数学上的等号“==”表示“等于”运算符，以区别于赋值运算符“=”。
- C语言提供了表示若干个子条件同时成立的运算符，用“&&”来连接若干个子条件，以表示这若干个子条件“同时成立”，这个运算符称为“与”运算符。其中“等于”运算符“==”属于C语言中的关系运算符，"与"运算符"&&" 属于C语言中的逻辑运算符。

- 关系表达式是指用关系运算符将两个表达式连接起来的有意义的式子

在C语言中，基本的关系运算符有以下六个：

| | |
|---|---|
| < | 小于 |
| <= | 小于或等于 |
| > | 大于 |
| >= | 大于或等于 |
| == | 等于 |
| != | 不等于 |

必须注意，在这六个关系运算符中，前四个(即<、<=、>、>=)运算符的优先级要高于后两个(即==、!=)运算符的优先级。并且还要特别注意，“等于”的关系运算符是“==”，而“=”是赋值运算符，要注意这两个运算符的区别。

- 在C语言中，用1表示关系表达式的值为“真”(即条件满足)，0表示关系表达式的值为“假”（即条件不满足），即关系表达式的值要么是1（条件满足），要么是0（条件不满足）。

- 特别需要指出的是，C语言中的关系表达式与数学中的不等式其意义是不一样的，因此，不能简单地将数学中的不等式作为关系表达式来使用。
- 例如，数学中的不等式 -5<x<5 表示变量x（设为整型变量）的一个取值范围，显然对于区间(-5，5)内的一切整数都满足这个不等式，在此区间外的所有整数都不满足这个不等式。
- 但在C语言中，-5<x<5 是一个合法的关系表达式，这个关系表达式等价于 (-5<x)<5 其中关系表达式(-5<x)的值要么是1（条件满足），要么是0（条件不满足），但不管关系表达式(-5<x)的值是1还是0，都小于5。即不管x的值为多少，关系表达式 (-5<x)<5 的值恒为1。即关系表达式 -5<x<5 的值恒为1。这说明，不管x如何取值，该条件表达式的值恒为1。
- 由上面的分析可以看出，数学中的不等式与C语言中的关系表达式其意义是不同的，在使用过程中应多加小心。
- 不等式 -5<x<5 正确的C语言写法是：-5<x && x<5

## 4.4 逻辑运算及其表达式

- 逻辑表达式是指用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子

逻辑型常量只有两种：值非零表示“真”，值为零表示“假”。

&& (逻辑与) 两个量都为真时为真(1)，否则为假(0)

|| (逻辑或) 两个量中只要有一个为真时为真(1)，都为假时为假(0)

! (逻辑非) 一个量为真(1)时为假(0)，假(0)时为真(1)

逻辑表达式中各种运算符的优先级顺序如下：

!(逻辑非) → 算术运算符→ 关系运算符→ && → || → 赋值运算符

注意： && (逻辑与)的运算优先级高于|| (逻辑或)，写逻辑表达式一定要小心。但逻辑表达式的计算与算术表达式一样，是按照自左至右的顺序执行，5>3 && 0 || 2 < 4 - !0 表达式计算时不会先执行!0和4 - !0，而是先执行5>3 ，这和执行算术表达式a+b-c*d 时先执行a+b的道理是一样的。

【例4-2】逻辑表达式“5>3 && 2 || 8 < 4 - !0”的运算顺序如下：

5>3 && 2 || 8 < 4 - !0

1 && 2 || 8 < 4 - !0

1 || 8 < 4 - !0

1

其中 8 < 4 - !0 将不会被执行。

**注意**：对于||，前一个项为1已经可以断定整个逻辑表达式结果为1，不再执行||后面的表达式。

---

【例4-3】逻辑表达式“5>3 && 0 || 2 < 4 - !0”的运算顺序如下：

5>3 && 0 || 2 < 4 - !0

1 && 0 || 2 < 4 - !0

0 || 2 < 4 - !0

0 || 2 < 4 - 1

0 || 2 < 3

0 || 1

1

【例4-4】 写出判断某一年year是否是闰年的逻辑表达式。

闰年的条件为： （1）能被4整除，但不能被100整除；

（2）能被400整除。

闰年可以用逻辑表达式表示为：

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

- 对于某一个年份year，如果上述表达式的值为1，则表示该年是闰年；如果值为0，则表示该年不是闰年。

非闰年的逻辑表达式为:

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

或

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

- 对于某一个年份year，如果上述两个表达式的值为1，则表示该年不是闰年；如果值为0，则表示该年是闰年。

【例4-5】 有甲，乙，丙三人，每人说一句话如下：

甲说：乙在说谎。

乙说：丙在说谎。

丙说：甲和乙都在说谎。

试写出能确定谁在说谎的条件（即逻辑表达式）。

分别用 a, b, c 表示甲，乙，丙三人，

值为1表示说真话，0表示说假话。

a==1 && b==0 等价于 a&&!b 以此类推

a && !b || !a && b

b && !c || !b && c

c && a+b==0 || !c && a+b!=0

C语言规定，C语言编译系统在对逻辑表达式的求解中，并不是所有部分都会被执行，只是在必须执行后面的运算符才能求出表达式值时，才执行其后的运算。

(a&&!b||!a&&b)&&(b&&!c||!b&&c)&&(c&&a+b==0||!c&&a+b!=0)

或：(a&&!b||!a&&b)&&(b&&!c||!b&&c)&&(c&&!(a+b)||!c&&a+b)

【例4-6】 阅读下列C程序，问输出的m，n，p分别为多少？

```
#include   <stdio.h>
main( )
{ int  a=1, b=2, c=3, d=4;
  int  p, m=1, n=1;
  p = (m=a>b) && (n=c>d) ;
  printf("m=%d\n n=%d\n p=%d\n", m, n, p);
}
```

输出结果为：
m=0
n=1
p=0

如果分解为以下三个语句：

```
m=a>b; n=c>d; p=m && n ;
```

此时，程序输出的结果为

m=0
n=0
p=0

- 注意事项：

C语言规定，C语言编译系统在对逻辑表达式的求解中，并不是所有的运算符都被执行。C语言为了提高执行效率而采用的规则为:

(1) 在有连续几个&&的表达式中，从左向右，只要有一个关系运算结果为假，整个结果将为假，不再执行后面的关系运算。例如：int a=5, b=4, c=3, d=3; 当执行语句：

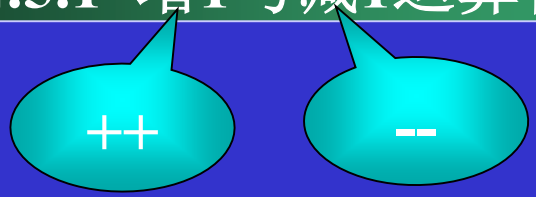
if (b>a && (d=c>a)) 时，由于b>a为假，故不再执行(d=c>a)，所以此语句执行完d仍为3，而不是0。

(2) 在有连续几个||的表达式中，从左向右，只要有一个关系运算结果为真，整个结果将为真，不再执行后面的关系运算。例如：int a=5, b=4, c=3, d=3; 当执行语句：if (a>b || (d=a>c)) 时，由于a>b为真，整个结果将为真，故不再执行(d=a>c)，所以此语句执行完d仍为3，而不是1。

- 因此，为避免出现条件表达式中某个赋值操作可能被执行也可能不被执行的不确定性问题，尽量不要在if语句或其他语句的条件表达式中使用赋值运算、++和--运算。

## 4.5 其他运算符

### 4.5.1 增1与减1运算符




增1与减1运算符位于运算对象的前面时，表示在使用该运算对象之前使它的值先增1或减1，然后再使用它，即使用的是增1或减1后的值。

例如，语句x=++n; 相当于以下两个语句的运算结果：

```
n=n+1; x=n;
```

增1与减1运算符位于运算对象的后面时，表示在使用该运算对象之后才使它的值增1或减1，即使用的是增1或减1前的值。

例如，语句x=n++; 相当于以下两个语句的运算结果：

```
x=n; n=n+1;
```

## 注意事项：

(1) 增1与减1运算符不能用于常量或表达式，只能用于左值量（lvalue，有对应内存单元的变量）。

例如，--5, (i+j)++等都是非法的。

(2) 操作符++和--是一元操作符，两个符号之间不能有空格。

(3) 一元操作符的执行优先级高于所有二元操作符，包括*、/和%运算符。

(4) 在表达式中尽量用空格隔开各个量，增加程序可读性。

例如：k=i+++++j; 不但可读性差，而且编译是错误的，应该写成：k = i++ + ++j;

(5) 特别提醒：k=1; printf("%d,%d,%d,%d,%d\n",k,++k,k,++k,k); 在VS上编译运行，两次++k会在printf语句之前先执行，因此输出结果将是：3,3,3,3,3

但 k=1; printf("%d,%d,%d,%d,%d\n",k,k++,k,k++,k); 在VS上编译运行结果是：3,2,3,1,3 因此毫无规律可循。

再次声明：这类问题属于C语言标准中的undefined behavior，希望大家以后不要再抠此类问题！

## 4.5.2 sizeof运算符

给出一个变量或某种数据类型的量在计算机内存中所占的字节数

① 用于求得表达式计算结果所占内存的字节数。

其一般形式为： **sizeof(表达式)** 或 **sizeof 表达式**

② 用于求得某种数据类型的量所占内存的字节数。

其一般形式为： **sizeof(类型名)**

**sizeof** 运算符可以出现在表达式中。

例如， **x=sizeof(float)-2;**

**printf("%d", sizeof(double));**

③ 但结构体的位段（**bit-field**）变量，不能用**sizeof**求所占内存的字节数。关于位段，将在第**13**章“位操作”中详细讲。

```
#include  <stdio.h>
main( )  /*输出各种类型的量在计算机中占的字节数*/
{ printf("sizeof(char)=%d\n",sizeof(char));
  printf("sizeof(int)=%d\n",sizeof(int));
  printf("sizeof(long)=%d\n",sizeof(long));
  printf("sizeof(long long)=%d\n",sizeof(long long));
  printf("sizeof(short)=%d\n",sizeof(short));
  printf("sizeof(float)=%d\n",sizeof(float));
  printf("sizeof(double)=%d\n",sizeof(double));
  printf("sizeof(3)=%d\n",sizeof(3));
  printf("sizeof(3L)=%d\n",sizeof(3L));
  printf("sizeof(3LL)=%d\n",sizeof(3LL));
  printf("sizeof(3.46)=%d\n",sizeof 3.46);
  printf("sizeof(3.46f)=%d\n",sizeof(3.46f));
}
```

在32位VS编译器上运行结果：

| | |
|---|---|
| sizeof(char)=1 | 字符整型量在计算机中占1个字节 |
| sizeof(int)=4 | 整型量在计算机中占4个字节 |
| sizeof(long)=4 | 长整型量在计算机中占4个字节 |
| sizeof(long long)=8 | 超长整型量在计算机中占8个字节 |
| sizeof(short)=2 | 短整型量在计算机中占2个字节 |
| sizeof(float)=4 | 单精度实型量在计算机中占4个字节 |
| sizeof(double)=8 | 双精度实型量在计算机中占8个字节 |
| sizeof(3)=4 | 整型数在计算机中占4个字节 |
| sizeof(3L)=4 | 长整型数在计算机中占4个字节 |
| sizeof(3LL)=8 | 超长整型数在计算机中占8个字节 |
| sizeof(3.46)=8 | 双精度实数在计算机中占8个字节 |
| sizeof(3.46f)=4 | 单精度实数在计算机中占4个字节 |

```
#include  <stdio.h>
main( )
{ printf("%d\n",sizeof('\n'));
}
```

运行结果是：

4

原因是：

在C语言中，常量数只有两种：整数和浮点数。C编译器默认所有整数常量是int型，浮点数常量是double型。 '\n'虽然在我们看来是字符型，但对于C编译器，看到的是这个字符的ASCII值10，也就是int型10，因此sizeof('\n')的值为4。

只有写为：sizeof((char)'\n')，结果才为1。

## 4.5.3 逗号运算符

**分隔符**

- 一个变量说明语句可以同时定义多个相同类型的变量，这些变量之间就用逗号来分隔。如 **int x, y, z;**
- 函数参数表中的各参数之间也是用逗号来分隔的。

```
printf("x=%d\ny=%d\nz=%d\n", x, y, z);
```

**运算符** → 顺序求值运算符

**逗号表达式**

**一般形式为:** 子表达式1, 子表达式2, …, 子表达式n

**求解过程是:** 按从左到右的顺序分别计算各子表达式的值，其中最右边的子表达式n的值就是整个逗号表达式的值。

- 逗号运算符是所有运算符中级别最低的一种运算符。

  例如，下面两个表达式的意义是不同的： ① x=3+4, 5+7, 10*4

  ② x=(3+4, 5+7, 10*4)

  ① x值为7，② x值为40

- 一个逗号表达式又可以与另一个表达式（可以是逗号表达式，也可以不是逗号表达式）连接成新的逗号表达式。

  (a=2*4, a*5), a-3　最后得到整个逗号表达式的值为5。

- 在许多情况下，使用逗号表达式的目的仅仅是为了得到各个子表达式的值，而并不一定要得到或使用整个逗号表达式的值。

  例如，为了实现交换a与b两个变量中的值，就可以使用下列逗号表达式 t=a, a=b, b=t; 它等价于 t=a; a=b; b=t;

  但前者是1个语句，后者是3个语句

- 在某些情况下，使用逗号表达式是逼不得已，因为不能使用更多分号等。 例如： for(i=0, j=0; i<10&&j<20; i++, j+=2)

for语句中只能出现两次分号，因此用逗号进行并列操作。

## 4.6 标准函数

在C语言中定义了一些标准函数，称为C库函数，用户在设计程序时可以很方便地调用它们。在本书的附录B中列出了VS2008 常用的库函数。

在使用C编译系统所提供的库函数时，必须要将相应的头文件包含到源程序文件中来，否则，在编译链接时会出错。引用这些头文件的目的是为了：函数的向前引用说明。

```
# include  <stdio.h>
# include  <stdlib.h>
# include  <string.h>
# include  <math.h>
```

## 4.7 宏定义

- 符号常量定义
- 带参数的宏定义
- 带#的宏定义

在C语言中，允许将程序中多处用到的“字符串”定义成一个符号常量。这样的符号常量又称为常量标识符。

### 4.7.1 符号常量定义

**#define 符号常量名 字符串**

好处：

可以减少程序中重复书写某些字符串的工作量。

使用符号常量便于程序的调试。

当需要改变一个常量时，只需改变 #define命令行中的字符串，则程序中所有带有符号常量名的地方全部被修改，而不必每处都一一进行修改。

【例4-7】 在下面的程序中，定义了一个符号常量P代表字符串"printf"，以及符号常量Q代表字符串"sizeof"：

```
#include <stdio.h>
#define P  printf
#define Q sizeof
main( )
{   P("％d\n", Q(int)); /*输出整型量在计算机中占的字节数*/
    P("％d\n", Q(long int)); /*输出长整型量在计算机中占的字节数*/
    P("％d\n", Q(short int)); /*输出短整型量在计算机中占的字节数*/
    P("％d\n", Q(float)); /*输出单精度实型量在计算机中占的字节数*/
    P("％d\n", Q(double)); /*输出双精度实型量在计算机中占的字节数*/
    P("％d\n", Q(3.46)); /*输出实型数在计算机中占的字节数*/
}
```

编译预处理后，会去掉注释和宏定义，程序变为：

```
#include <stdio.h>
main( )
{    printf("%d\n", sizeof(int));
     printf("%d\n", sizeof(long int));
     printf("%d\n", sizeof(short int));
     printf("%d\n", sizeof(float));
     printf("%d\n", sizeof(double));
     printf("%d\n", sizeof(3.46));
}
```

实际上#include <stdio.h> 也会去掉，把stdio.h文件的内容全部插入到上面的程序的开头处。然后开始真正的编译过程。

- 在定义符号常量时要注意以下几个问题：

(1) 由于C语言中的所有保留关键字一般使用小写字母，因此，符号常量名一般用大写字母表示，以便与C语言中的保留关键字相区别，例如　　**#define PAI 3.1415926**

(2) C编译系统对定义的符号常量的处理只是进行简单的字符串替换，不作任何语法检查。但要注意，程序中用双引号(“)括起来的字符串，即使与定义中需要替换的字符串相同，也不进行替换。

(3) **#define** 是一个预处理命令，而不是语句，因此在行末不能加“;”，并且应独立占一行。

(4) **#define** 命令一般应出现在程序中函数的外面，其作用域范围是从 **#define** 符号常量名 字符串 到 **#undef** 符号常量名(或本文件末)。

(5) 一个**#define** 的定义如果一行写不下，可以下一行继续写，本行行尾加续行符‘\’, 这种续行可以一直进行下去。例如：

```
#define ABCDE    234567888*234+1234567+7654321+888888+ \
                 7777777 +666666 +5555555
```

## 4.7.2 带参数的宏定义

在用#define命令定义符号常量时，C编译系统只是简单地进行字符串替换。但如果在定义的符号常量后带有参数，则不仅要对字符串进行替换，还要进行参数替换。这种带有参数的符号常量简称为宏（Macro）。

**#define　宏名(参数表)　字符串**

其中字符串中应包含在参数表中所指定的参数，并且，当参数表中的参数多于一个时，各参数之间要用逗号分隔。

重写【例4-7】的程序：

```
#include <stdio.h>
#define P(x)  printf ("%d\n", x)
#define Q sizeof
main( )
{    P(Q(int)); /*输出整型量在计算机中占的字节数*/
     P(Q(long int)); /*输出长整型量在计算机中占的字节数*/
     P(Q(short int)); /*输出短整型量在计算机中占的字节数*/
     P(Q(float)); /*输出单精度实型量在计算机中占的字节数*/
     P(Q(double)); /*输出双精度实型量在计算机中占的字节数*/
     P(Q(3.46)); /*输出实型数在计算机中占的字节数*/
}
```

还可以重写【例4-7】的程序为：

```
#include <stdio.h>
#define Q  sizeof
#define P(x)  printf ("%d\n", Q(x))
 /* 后一个宏定义可以使用前一个宏 */
main( )
{  P(int); /*输出整型量在计算机中占的字节数*/
   P(long int); /*输出长整型量在计算机中占的字节数*/
   P(short int); /*输出短整型量在计算机中占的字节数*/
   P(float); /*输出单精度实型量在计算机中占的字节数*/
   P(double); /*输出双精度实型量在计算机中占的字节数*/
   P(3.46); /*输出实型数在计算机中占的字节数*/
}
```

【例4-8】下面是计算两个长方体体积之和的程序。其中第一个长方体各边的边长分别为3, 4, 5，第二个长方体各边的边长分别为11, 23, 45。

```
#include  <stdio.h>
#define  V(a, b, c)      a*b*c
main( )
{ double  vsum;
   vsum=V(3.0, 4.0, 5.0) + V(11.0, 23.0, 45.0);
   printf("vsum=%f\n", vsum);
}
```

在这个程序中，将计算长方体体积定义为宏。这个程序经编译宏替换展开后， 赋值语句 vsum=V(3.0, 4.0, 5.0)+V(11.0, 23.0, 45.0);

等价于 vsum=3.0*4.0*5.0+11.0*23.0*45.0;

预编译处理后，程序为：

```
#include  <stdio.h>
main( )
{ double  vsum;
   vsum= 3.0*4.0*5.0 + 11.0*23.0*45.0;
   printf("vsum=％f\n", vsum);
}
```

注意事项：

- 在使用带参数的宏定义时，一般应将宏定义字符串中的参数都用括号括起来，否则经过宏替换展开后，可能会出现意想不到的错误。下面的例子说明了这个问题。

【例4-9】　计算下列函数值：

$f(x)=x^3+(x+1)^3$　　其中自变量x的值从键盘输入。

如果将计算$x^3$的值定义为一个带参数的宏，即

```
#define  F(x)  x*x*x
```

计算函数值f (x)的C程序写成如下：

```
#include  <stdio.h>
#define  F(x)  x*x*x
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f=F(x)+F(x+1);
  printf("f=％f\n", f);
}
```

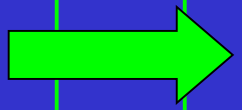
预处理后：

```
#include  <stdio.h>
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f= x*x*x + x+1*x+1*x+1;
  printf("f=％f\n", f);
}
```

结果肯定不对！

计算函数值f (x)的C程序写成如下：

```
#include  <stdio.h>
#define  F(x)  x*x*x
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f=F(x)+F(x+1);
  printf("f=％f\n", f);
}
```



重写宏F(x)：

```
#include  <stdio.h>
#define  F(x)   (x)*(x)*(x)
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f=F(x)+F(x+1);
  printf("f=％f\n", f);
}
```

计算函数值f (x)的C程序写成如下：

```
#include  <stdio.h>
#define  F(x)  (x)*(x)*(x)
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f=F(x)+F(x+1);
  printf("f=％f\n", f);
}
```

预处理后：

```
#include  <stdio.h>
main( )
{ double  f, x;
  printf("input x：  ");
  scanf("％lf", &x);
  f=(x)*(x)*(x)+(x+1)*(x+1)*(x+1);
  printf("f=％f\n", f);
}
```

结果正确但仍不完美！

注意事项：

- 在使用带参数的宏定义时，除了应将宏定义字符串中的参数都要用括号括起来，还需要将整个字符串部分也要用括号括起来，否则经过宏替换展开后，还会可能出现意想不到的错误。

  用下面的例子说明这个问题。

【例4-10】 计算下列函数值：

$$f(x,y)=[x^3+x^2][(y+1)^3+(y+1)^2]$$

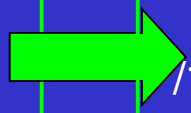
其中自变量x与y的值从键盘输入。

如果将计算$x^3+x^2$的值定义为一个带参数的宏，即

```
#define  F(x)  (x)*(x)*(x)+(x)*(x)
```

计算函数值f (x, y)的C程序预处理宏替换后：

```
#include  <stdio.h>
#define  F(x)  (x)*(x)*(x)+(x)*(x)
main( )
{ double  f, x, y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f=F(x)*F(y+1);
  printf("f=％f\n", f);
}
```



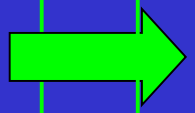
```
#include  <stdio.h>
main( )
{ double  f, x, y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f= (x)*(x)*(x)+(x)*(x) *
    (y+1)*(y+1)*(y+1)+(y+1)*(y+1);
  printf("f=％f\n", f);
}
```

结果也肯定不对！

带参数的宏定义

计算函数值f (x, y)的C程序可以写成如下形式：

```
#include  <stdio.h>
#define  F(x)  (x)*(x)*(x)+(x)*(x)
main( )
{ double  f, x, y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f=F(x)*F(y+1);
  printf("f=％f\n", f);
}
```



```
#include  <stdio.h>
#define  F(x)  ((x)*(x)*(x)+(x)*(x))
main( )
{ double  f, x, y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f=F(x)*F(y+1);
  printf("f=％f\n", f);
}
```

计算函数值f (x, y)的C程序预处理宏替换后：

```
#include  <stdio.h>
#define  F(x)  ((x)*(x)*(x)+(x)*(x))
main( )
{ double  f, x, y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f=F(x)*F(y+1);
  printf("f=％f\n", f);
}
```

```
#include  <stdio.h>
main( )
{ double  f, x,y;
  printf("input x, y：  ");
  scanf("％lf, ％lf", &x, &y);
/*输入的两个数据之间用逗号分隔*/
  f= ((x)*(x)*(x)+(x)*(x)) *
    ((y+1)*(y+1)*(y+1)+(y+1)*(y+1));
  printf("f=％f\n", f);
}
```

结果完全正确！

## 4.7.3 带#的宏定义

#define中有两种情况可以使用#。

① 一种是变量的字符串化（stringizing）。即

#标识符 ➔ "标识符"

例如，若有程序：

```
#include <stdio.h>
#define   PR(x)    printf("%s=%d\n", #x, x)
main( )
{  int a=15, b2=123;
   PR(a);
   PR(b2);
}
```

经编译预处理后程序变为：

```
#include <stdio.h>
main( )
{   int a=15, b2=123;
    printf ("%s=%d\n", "a", a);
    printf ("%s=%d\n", "b2", b2);
}
```

运行结果：

**a=15**

**b2=123**

不但打印变量的值，还打印出了变量的名字。

## 4.7.3 带#的宏定义



② 另一种是所谓 字符串连接（token-pasting），即

<标识符> ## <宏变量> ➔ <标识符><宏变量>

例如，若有程序：

```
#include <stdio.h>
#define  MP(x)  printf("%d", a##x)
main( )
{   int a1=2, a5=4;
    MP(1);
    MP(5);
 }
```

经编译预处理后程序变为：

```
#include <stdio.h>
main( )
{   int a1=2, a5=4;
    printf("%d", a1);
    printf("%d", a5);
}
```

- 编译预处理的替换，将MP(1);替换为printf("%d", a1);其中的a##x将标识符a与宏变量x的值串1串接起来，形成变量名a1；将MP(5);替换为printf("%d", a5);其中的a##x将标识符a与宏变量x的值串5串接起来，形成变量名a5。

若把程序改为：

```
 #include <stdio.h>
#define  MP(x)  printf("%d", a##x)
main( )
{   int a1=2, a5=4, i;
    i=1; MP(i);
    i=5; MP(i);
 }
```

编译结果：

test.c(5) : error C2065: "ai": 未声明的标识符

test.c(6) : error C2065: "ai": 未声明的标识符

因为：MP(i);　宏展开后为：printf("%d\n", ai);

第4次作业

教程 p.81-84

习题 1，2，3，8，9，10，12，13