计算机程序设计基础（1）
04 表达式与宏定义

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn

上节重点

- 二进制的**转换**与**表示**
- 各种**基本数据类型**
  - int, double, float, char……
- 数据占用内存空间大小：**sizeof()**
- 数据的输入输出
  - 输出函数**printf()**及其**格式控制**
  - 输入函数**scanf()**及其**格式控制**
  - **字符的输入输出函数**

上节重点：格式字符

格式字符	说明
c	输入(出)一个字符
d	输入(出)带符号的十进制整型数
I	输入(出)整型数，整型数可以是带先导0的八进制数，也可以是带先导0x(或0X)的十六进制数，输入时候会转换成十进制
o	输入(出)八进制格式整型数，可以带先导0，也可以不带
x	输入(出)十六进制格式整型数,可以带先导0x或0X,也可以不带
u	输入(出)无符号十进制形式整型数
f (lf)	输入(出)以小数形式浮点数（单精度用f，输入双精度数用lf)
e (le)	输入(出)以指数形式浮点数（单精度用e，输入双精度数用le)
g	输入(出)自动决定输出格式为e和f中较短的一种，不打印无效的零
p	输出变量的内存地址
s	输入(出)一个字符串，直到遇到"\0"

上节重点：格式字符

格式字符	说明
l	输入(出)长度修饰： 整型%ld, %lu, %lo, %lx； 浮点型%lf, %le
h	输入(出)长度修饰： 整型%hd
m	整型输入(出)位宽：超过自动突破；不足右对齐，左补空格，达到位宽 字符串输出宽度： 超过自动突破；不足右对齐，左补空格，达到位宽
m.n	浮点型输入(出)精度位宽和精度，可以只有.n，也可以m.0，还可以.0 m超过自动突破；不足右对齐，左补空格； n超过四舍五入；不足右边补0
.n	浮点型输出精度，见上；字符串输出宽度，超过截断，不足右对齐，左补空格
0	前导0符号：整型输出右对齐，左边补0；可以写作%0m或%.m表示
+	正号符号：输出正数总带+号
-	左对齐符号：输出左对齐
*	输出变位宽输出；输入跳过某个数
%	输入(出)格式说明引导符

本节目录

4.1、C语言表达式

- 赋值运算与表达式
- 算术运算与表达式
- 关系运算与表达式
- 逻辑运算与表达式
- 其他运算符

4.2、标准函数与宏定义

- 标准函数
- 宏定义

C语言表达式

4.1、C语言表达式

- 赋值运算与表达式
- 算术运算与表达式
- 关系运算与表达式
- 逻辑运算与表达式
- 其他运算符

赋值运算与表达式：赋值表达式

●赋值运算符为“=”，赋值表达式为：

变量名 = 表达式

- 赋值表达式的功能是：首先计算“=”右边的表达式值
- 然后将计算结果赋给左边的变量
- 最后该赋值表达式的值也就是运算的结果
- 赋值表达式可以出现在另一个表达式中参与运算

赋值运算与表达式：赋值语句

- 假设x与y都是已定义的整型变量，则表达式 x = y = 4 + 5

等价于 x = (y = 4 + 5)

  - 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**，该赋值表达式(y = 4 + 5)的值就是计算结果9
  - 然后将该表达式的值**赋给变量x**，最终x和y的值均为4+5的计算结果9

- 如果在赋值表达式的最后加一个“;”，就变成了**赋值语句**。即赋值语句的形式为：

变量名 = 表达式;

赋值运算与表达式：注意事项

- 对赋值表达式或赋值语句做几点说明
  - 在C语言中，“=”为赋值运算符，**不是等号**
  - 赋值运算符“=”左边**必须是变量名**，不能是表达式
  - 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**，但编译时有时会给出警告性错误提示（warning）
    - 如下例所示，将一个浮点数值赋给整型变量，编译时会发生警告，结果也会被**强制转换类型**

```
1  #include<stdio.h>
2  void main(){
3    int x;          //定义整型变量x
4    x = 1.5;
5    printf("%d\n",x);
6  }
```

```
warning C4244: “=”：从“double”转换到“int”，可能丢失数据
```

```
1
请按任意键继续……
```

赋值运算与表达式：复合赋值运算符

- 在实际编程中，常常会用到以下的表达式

$$x = x + 1$$

  - 这样的式子显得较为**冗余**
  - 为了简化程序，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**
  - 例如上式可以写成

$$x \mathrel{+}= 1$$

赋值运算与表达式：复合赋值运算符

- 复合赋值运算符
  - 一般来说，凡是需要两个运算对象的运算符（即双目运算符），如：+、-、*、/、%（模余运算符），都可以与赋值运算符一起构成复合的赋值运算符

<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>)

- 例如： a *= b + 3　等价于　a = a*(b + 3)
- 例如： a %= b + 3　等价于　a = a%(b+3)

算术运算与表达式：基本算术运算符


- C语言中，基本的算术运算符：
  - 在解决数值型问题时，算术表达式是必不可少的
  - “+” **加法运算符**（双目运算符），如 x+y；或**正值运算符**（单目运算符）如 +5
  - “-” **减法运算符**（双目运算符），如 y-8；或**负值运算符**（单目运算符）如 -z
  - “*” **乘法运算符**（双目运算符），如a*b
  - “/” **除法运算符**（双目运算符），如a/b
  - “%” **模余运算符**（双目运算符），如 c%d
    - 只适用于整型数据，如12%5的值为2，(-12)%5的值为-2，(-12)%(-5)的值为-2

算术运算与表达式：运算优先级

- 这些运算符的运算优先级顺序与数学上的运算顺序相同
  - **先乘除后加减**
  - 乘法、除法、模余运算属于同一优先级
  - 加法和减法处于同一优先级
  - 同一优先级情况下，从左至右进行运算
  - 但在**不破坏优先级**的前提下，**能先执行的就会被执行**
    - 因为算术表达式执行是从左至右，表达式a+b+c*d是先执行r1 = a+b，再计算r2 = c*d，最后再执行r1+r2得到最后结果
- 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

算术运算与表达式：注意事项

- 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
  - 例如：(a*b)/(c*d)≠a*b/c*d, (a*b)/(c*d)≠a*b/c/d ？？？
- 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值为1，-2/7的值为0
  - 例如：(1/2)+(1/2)的值为0，而不是1
  - 两个整型数相除的结果的绝对值相当于真实结果的绝对值向下取整，符号与真实结果相同

算术运算与表达式：注意事项

- 表达式数据类型不一致时,系统将**进行类型转换**,原则是**从低到高**
  - ✓ 字符型数据都是用该字符的ASCII码进行计算的。例如表达式'A'+'B'的计算结果为131（字符A的ASCII 码值为65，B则是66，见课本附录）
  - ✓ 短整型(short int)变量也都转换为基本整型(int)数据后才参与计算
  - ✓ 单精度实型(float)变量都转换成双精度(double)后再进行计算
  - ✓ 当基本整型与无符号整型(unsigned int)数据进行运算时，将基本整型转换成无符号整型
  - ✓ 当无符号整型与长整型数据(long)进行运算时，将无符号整型转换成长整型
  - ✓ 当长整型与双精度实型进行运算时，将长整型转换成双精度实型

(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)

算术运算与表达式：注意事项

- 类型转换不改变数据的类型
  - 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
  - 下例中，x的值在计算过程中进行了类型转换，但在输出时依旧是整型数

```
1  #include<stdio.h>
2  void main(){
3    int x = 2;
4    double y = 1.2;
5    printf("%lf %d\n",x*y, x);
6  }
```

```
2.400000 2
请按任意键继续……
```

算术运算与表达式：注意事项

- 类型转换是逐步进行的
  - 在表达式中进行混合运算时自动进行类型转换，但并不是将表达式中的所有量统一进行转换后再行计算，而是在运算过程中**逐步进行转换**
    - 如表达式10/4 + 2.5，在进行计算时，首先计算10/4的值，根据前述整型数除法的规则，计算结果为2
    - 再计算2 + 2.5,此时“+”两端数据类型不匹配，将整型数2转化为实型数2.0进行计算，最后计算结果4.5
    - C语言进行类型转换时只考虑运算符两边数据类型是否匹配，若不匹配则按前述规则转换

```
1  #include<stdio.h>
2  void main(){
3     printf("%lf\n",10/4 + 2.5);
4  }
```

```
4.500000
请按任意键继续……
```

算术运算与表达式

- 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：int(x-y) 将表达式x-y的结果强制转换为整型数
- 例如：(double)y/x将y转换为双精度实型后与后面的x进行除法运算
- 与前面的自动类型转换相同，这里的强制类型转换仅在运算时对变量的值进行转换用于计算，而**不对变量本身进行转换**
- 虽然C语言具备**自动类型转换**功能，但对于复杂情形下编译器也无法进行转换，同时如前例所示，自动转换的最终结果也未必正确，因此在平时使用中尽量使用**强制类型转换**的方式

课堂练习4-1

- 求解下列表达式的值
  - (7 + 6)%5/2
  - -3*4%5/6
  - (double)((3+4)%5)/4
- 理解下列表达式的含义
  - x = a += b *= c -= 5
  - a += a -= a *= a

课堂练习4-1解答

- (7 + 6)%5/2
  - 该表达式值为1，首先依序计算：7+6=13, 13%5=3
  - 注意此时模余运算的结果仍然是整型数，整型数除法3/2的结果为1，因此该表达式的计算结果为1
- -3*4%5/6
  - 模余运算、乘法、除法处于同一运算优先级，因此从左至右依次运算
  - 3*4结果为12，模5结果为2，同样是整型数，除以6结果为0，再加上最前面的负号，因此结果为0
- (double)((3+4)%5)/4
  - 首先计算(3+4)%5的值，计算结果为整型数2
  - 之后再经过强制类型转换成为双精度实型数
  - 进行除法运算时，由于除号两边数据类型不一致，再对被除数4进行自动类型转换为双精度实型数，最终计算结果为0.5

课堂练习4-1解答

- x = a += b *= c -= 5
  - 赋值表达式的计算是从右向左进行的
  - 因此上式可以依次分解为以下几个步骤：c=c-5; b=b*c; a=a+b; x=a;
  - 按照此步骤依次对各个变量进行赋值运算
- a += a -= a *= a
  - 同样按照上面的思路，从右向左依次展开
  - 上式可以依次分解为以下几个步骤：a=a*a; a=a-a; a=a+a
  - 按照此步骤依次进行计算，可以不难看出最后的结果为0

关系运算与表达式

- 对于任何实际问题的解决，都几乎离不了**逻辑判断**
- 例4-1：已知A、B、C、D四个人中有一个人是小偷，并且，这四个人中每个人要么说真话，要么说假话，审问过程中，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

□现要求根据四个人的回答，写出能够确定小偷的**条件**

关系运算与表达式

- 由于4个人中只有一个人是小偷，而且无论是否是小偷，他的回答要么是真话，要么是假话。因此可以做如下分析
  - A说：B没有偷，是D偷的。如果这句话为真，则应有b=0（即B不是小偷），d=1（D是小偷）
  - 如果这句话为假则反之亦然，因此有**b+d=1**
  - B说：我没有偷，是C偷的。与上面的分析同理，应有**b+c=1**
  - C说：A没有偷，是B偷的。同理有**a+b=1**
  - D说：我没有偷。如果这句话为真，则d=0，同时a、b、c中有一个为1，其他为0；反之则d=1，a、b、c为0。因此有**a+b+c+d=1**

关系运算与表达式

- 根据以上的分析，为了确定A、B、C、D中谁是小偷，只需**穷举**a、b、c、d为1的情况，分别代入到上面得到的四个式子当中
- 当全部条件全部成立时，即可确定谁是小偷，即以下四个条件**必须全部成立**
  - b+d=1
  - b+c=1
  - a+b=1
  - a+b+c+d=1

关系运算与表达式：注意事项

- 在C语言中，上述表达式不能够直接搬运
  - 因为“=”运算符在之前已经被用于作为赋值运算符，而不能用来表示“等于”
- 在C语言中，使用两个等号，即“==”来表示“等于”运算，用以区别于赋值运算符
  - 例题中的条件可以写为 b+d==1 且 b+c==1 且 a+b==1 且 a+b+c+d==1

关系运算与表达式：关系运算符

- 在C语言中，基本的关系运算符有以下六个
  - 在这6个关系运算符中，前4个（即<,<=,>,>=）运算符的优先级要高于后两个（即==,!=）运算符的优先级
  - 同时注意“==”和赋值运算符“=”的区别

符号	含义
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
!=	不等于

关系运算与表达式：关系表达式

- 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，如下列各式子都是有意义的关系表达式
  - c>a+b　　等效于　　c>(a+b)
  - a>b!=c　　等效于　　(a>b)!=c
  - a==b<c　　等效于　　a==(b<c)
- 关系运算符的值可以赋给整型变量或字符型变量
  - 例如a=b>c是一个赋值表达式，等效于a=(b>c)，即**关系运算符的优先级要高于赋值运算符**
  - C语言中用1表示关系表达式的值为真，0表示关系表达式的值为假，即关系表达式的值非0即1

关系运算与表达式：注意事项

- C语言中的**关系表达式**与数学中的**不等式**意义是完全不一样的
  - 不能简单的将数学中的不等式作为关系表达式来使用
  - 数学中的不等式-5<x<5，表示变量x（设为整数）的取值范围
  - 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
  - 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，-5<x这个关系表达式的值非0即1
  - 则不论x取何值，(-5<x)<5恒成立，该表达式的值**恒为1**

逻辑运算与表达式：逻辑运算


- 逻辑型常量只有两种：值非零表示“真”，值为零表示“假”，其基本的逻辑运算符有以下3个
  - **逻辑与** &&　　两个量都为真是为真（1）,否则为假（0）
  - **逻辑或** ||　　两个量只要有一个为真则为真，否则为假
  - **逻辑非** !　　一个量为真是为假，假时为真
- 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

- 逻辑表达式的执行顺序依然是从左向右，能先执行的就先执行

逻辑运算与表达式：注意事项

- 需要说明的是，在C语言中，无论是整型常量还是实型常量，都可以作为逻辑常量，并且是**非假(0)**即**真(1)**
  - 0.3 && 0.1　的值为1
  - 0.3 || 0　　的值为1
  - 0.3 && 0　　的值为0
  - ！ 0.3　　的值为0
  - ！ 0　　的值为1
- 前面在关系表达式中提出的-5<x<5在C语言中，可以用**x>-5 && x<5**这个表达式来表示

逻辑运算与表达式

- 例4-2：逻辑表达式 5>3 && 2 || 8 < 4 -!0的运算顺序
  - 从左至右依次执行
    - 首先运算5>3（而不是!0, 思考原因），该关系表达式的值为1
    - 再对1 && 2进行计算，该逻辑表达式值为真（1）
    - 此时该表达式化简为1 || 8<4-!0
    - 在后面会通过例题验证，在C语言中，进行或运算时，只要前面一项为真，后面一项不再计算，该表达式值直接为1

逻辑运算与表达式

- 例题4-3：逻辑表达式 5>3 && 0 || 2<4 - !0的运算顺序

$$
\begin{aligned}
&\underline{5 > 3} \;\&\&\; 0 \;\|\; 2 < 4 - !0 \\
&\underline{1 \;\&\&\; 0} \;\|\; 2 < 4 - !0 \\
&0 \;\|\; 2 < 4 - \underline{!0} \\
&0 \;\|\; 2 < \underline{4 - 1} \\
&0 \;\|\; \underline{2 < 3} \\
&\underline{0 \;\|\; 1} \\
&1
\end{aligned}
$$

课堂练习4-2

- 写出判断某一年year是否为闰年的逻辑表达式
  - 闰年的条件是：1）能被4整除，但不能被100整除。或

    2）能被400整除
  - 满足条件1），则有year%4 == 0 && year%100 != 0
  - 满足条件2），则有year%400 == 0
  - 上面二者有一个成立则year为闰年

则该表达式为 **(year %4 == 0 && year %100 != 0) || year %400 == 0**

逻辑运算与表达式

- 例4-4： 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）
  - 分别用整型变量abc表示甲乙丙三人，值为1则代表该人说真话，为0则代表在说谎
  - 若甲说的是真话，则有a==1&&b==0，等价于a&&!b，若甲在说谎，则有!a&&b，因此根据甲的话，则有 a&&!b || !a&&b
  - 同理，根据乙的话，则有 b&&!c || !b&&c
  - 若丙是真话，则有c==1&&a+b==0，等价于c&&!(a+b)，若丙在说谎，则有!c&&a+b,因此根据丙的话有 c&&!(a+b) || !c&&a+b
  - 上面三个条件需同时成立，因此是“逻辑与”的关系，最后得到表达式如下

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

逻辑运算与表达式：注意事项


- 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
  - 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，如下例所示：

```
1  #include<stdio.h>
2  void main(){
3     int x = 0,  y = 0,  p;
4     p = x && (y+=1);
5     printf("x = %d,y = %d\n",x,y);
6  }
```

```
x = 0,y = 0
请按任意键继续……
```

可见后面的“y+=1”并没有执行

逻辑运算与表达式：注意事项


- 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。如下所示：

```
1  #include<stdio.h>
2  void main(){
3    int x = 1, y = 0, p;
4    p = x || (y+=1);
5    printf("x = %d,y = %d\n",x,y);
6  }
```

```
x = 1,y = 0
请按任意键继续……
```

可见“y+=1”依旧没有执行

- 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**

其他运算符：增一与减一运算符

●增一与减一运算符

- 增1运算符“++”和减1运算符“--”是两个单目运算符（只有一个运算对象），它们的运算对象只能是**整型**或**字符型变量**
- 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：x=++n; 等价于 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：x=n++; 等价于 x=n; n=n+1;

其他运算符

- 下面的案例展现了++运算符不同位置的作用

```
1  #include<stdio.h>
2  void main(){
3     int x = 0,  y = 0;
4     printf("x1 = %d,y1 = %d\n",x++,++y);
5     printf("x2 = %d,y2 = %d\n",x,y);
6  }
```

```
x1 = 0,y1 = 1
x2 = 1,y2 = 1
请按任意键继续……
```

其他运算符

- 增一或减一运算符**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**)
  - 诸如--5，(i+j)++这样的写法都是错误的
- 操作符++和--是**一元操作符**，两个符号之间**不能有空格**
- 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
- 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，而且是**错误的**（无法通过编译）
  - 应该写成：k = i++ + ++j;

其他运算符

- sizeof运算符——详见上节课课件
- 逗号运算符
  - 逗号的一种常见用法是用作**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);
```

  - 逗号的另一种用法是用作**运算符**
    - 一般形式：子表达式1，子表达式2，子表达式3……
    - 从左到右顺序计算，最后一个子表达式n的值就是逗号表达式的值。因此，逗号运算符又称为顺序求值运算符

其他运算符

- 逗号运算符是所有运算符中**级别最低**的一种运算符
  - x = 2+5, 7+9, 4*10；和 x = (2+5, 7+9, 4*10) 的意义是不同的，前者是x=7，后者是x=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**
- 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**，而并不一定要得到整个逗号表达式的值
  - 为实现两个数的交换，有t=a, a=b, b=t;与 t=a； a=b； b=t;是等价的

课堂练习4-3

●写出下列程序的输出

```
1  #include<stdio.h>
2  void main(){
3     int x = 0, y = 1, a, b;
4     b = ++x && y-- &&(a=2*4, a*5, a-8);
5     printf("x=%d, y=%d, a=%d, b=%d\n", x, y, a, b);
6  }
```

课堂练习4-3解答

- 逻辑表达式，**从左至右**依次执行
  - 执行++x，首先使x=x+1，此时x=1为真值，继续执行下一个表达式
  - 执行y--，首先判断y=1为真值，可以进行下一步操作，再使y=y-1，执行下一个表达式
  - 逗号运算符依次计算，首先给a赋值8，最后一步a-8为0，因此整个逗号表达式值为0，所以整个逻辑表达式值为**假**，因此b=0

```
1  #include<stdio.h>
2  void main(){
3   int x = 0, y = 1, a, b;
4   b = ++x && y-- &&(a=2*4, a*5, a-8);
5   printf("x=%d, y=%d, a=%d, b=%d\n", x, y, a, b);
6  }
```

```
x=1, y=0, a=8, b=0
请按任意键继续……
```

4.2、标准函数与宏定义

- 标准函数
- 宏定义

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列出了常用的库函数
- 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> :输入输出函数，如printf(), scanf(), gerchar(), putchar()等

#include<stdlib.h>:动态分配存储空间函数，如malloc(), free()等

#include<math.h> :数学函数，如sin(), log(), sqrt(), floor()等

#include<string.h>: 字符与字符串函数，如strlen(),strcat()等

符号常量的宏定义

●符号常量的宏定义

- □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**

  #define 符号常量名 字符串

- □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，而不必每处都要进行修改

符号常量的宏定义

●例4-5：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”：

```
1  #include<stdio.h>
2  #define P printf
3  #define Q sizeof
4  void main(){
5    P("%d\n", Q(int));          //输出整型量在计算机中占的字节数
6    P("%d\n", Q(long int));     //输出长整型量在计算机中占的字节数
7    P("%d\n", Q(short int));    //输出短整型量在计算机中占的字节数
8    P("%d\n", Q(float));        //输出单精度实型量在计算机中占的字节数
9    P("%d\n", Q(double));       //输出双精度实型量在计算机中占的字节数
10   P("%d\n", Q(3.46));         //输出实型数在计算机中占的字节数
11 }
```

```
4
4
2
4
8
8
请按任意键继续……
```

符号常量的宏定义

- 在定义符号常量时要注意以下几个问题：
  - 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，如

```
#define PI 3.1415926
```

  - 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**
  - 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**

符号常量的宏定义

- #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
- #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)
- 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，如：

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

带参数的宏定义

●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**
- 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。当参数表中的参数多于一个时，各参数之间要用逗号分隔

带参数的宏定义

●例4-6：用带参数宏重写例4-5中的程序

```
1  #include <stdio.h>
2  #define P(x) printf ("%d\n", x)
3  #define Q sizeof
4  void main(){
5    P(Q(int));          //输出整型量在计算机中占的字节数
6    P(Q(long int));     //输出长整型量在计算机中占的字节数
7    P(Q(short int));    //输出短整型量在计算机中占的字节数
8    P(Q(float));        //输出单精度实型量在计算机中占的字节数
9    P(Q(double));       //输出双精度实型量在计算机中占的字节数
10   P(Q(3.46));         //输出实型数在计算机中占的字节数
11 }
```

带参数的宏定义

- 前例的程序还可以重写为

```
1  #include <stdio.h>
2  #define Q sizeof
3  #define P(x) printf ("%d\n", Q(x))
4  //后一个宏定义可以使用前一个宏
5  void main(){
6    P(int);          //输出整型量在计算机中占的字节数
7    P(long int);     //输出长整型量在计算机中占的字节数
8    P(short int);    //输出短整型量在计算机中占的字节数
9    P(float);        //输出单精度实型量在计算机中占的字节数
10   P(double);       //输出双精度实型量在计算机中占的字节数
11   P(3.46);         //输出实型数在计算机中占的字节数
12 }
```

带参数的宏定义

- 例4-7：计算两个长方体体积之和，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
  - 将计算长方体体积定义为宏，这个程序经编译宏展开后赋值语句

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;

```
1  #include <stdio.h>
2  #define V(a, b, c) a*b*c
3  void main(){
4    double vsum;
5    vsum=V(3.0, 4.0, 5.0)+V(11.0, 23.0, 45.0);
6    printf("vsum=%f\n", vsum);
7  }
```

带参数的宏定义

- 预编译处理后，程序变为

```
1  #include <stdio.h>
2  void main(){
3    double vsum;
4    vsum= 3.0*4.0*5.0+11.0*23.0*45.0;
5    printf("vsum=%f\n", vsum);
6  }
```

- 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**，否则经过宏展开后，可能会出现意想不到的错误

带参数的宏定义

- 例4-8：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

结果肯定不对！

```
1  #include <stdio.h>
2  #define F(x) x*x*x
3  void main(){
4    double f, x;
5    printf("input x: ");
6    scanf("%lf", &x);
7    f=F(x)+F(x+1);
8    printf("f=%f\n", f);
9  }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3    double f, x;
4    printf("input x: ");
5    scanf("%lf", &x);
6    f= x*x*x + x+1*x+1*x+1;
7    printf("f=%f\n", f);
8  }
```

带参数的宏定义

- 将F(x)重写成为

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

```
1  #include <stdio.h>
2  #define F(x) x*x*x
3  void main(){
4    double f, x;
5    printf("input x: ");
6    scanf("%lf", &x);
7    f=F(x)+F(x+1);
8    printf("f=%f\n", f);
9  }
```

```
1  #include <stdio.h>
2  #define F(x) (x)*(x)*(x)
3  void main(){
4    double f, x;
5    printf("input x: ");
6    scanf("%lf", &x);
7    f=F(x)+F(x+1);
8    printf("f=%f\n", f);
9  }
```

带参数的宏定义

- 改写后程序预处理前后如下

这次虽然结果正确，但仍有一定问题，下个例题会进行说明

```
1  #include <stdio.h>
2  #define F(x) (x)*(x)*(x)
3  void main(){
4    double f, x;
5    printf("input x: ");
6    scanf("%lf", &x);
7    f=F(x)+F(x+1);
8    printf("f=%f\n", f);
9  }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3    double f, x;
4    printf("input x: ");
5    scanf("%lf", &x);
6    f= x*x*x +(x+1)*(x+1)*(x+1);
7    printf("f=%f\n", f);
8  }
```

带参数的宏定义

- 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**用括号括起来，还需要将**整个字符串部分**也用括号括起来
  - 否则经过宏展开后，还会可能出现意想不到的错误
- 例4-9：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)
```

带参数的宏定义

- 计算函数值f (x, y)的C程序预处理宏替换后

结果显然不对

```
1  #include <stdio.h>
2  #define F(x) (x)*(x)*(x)+(x)*(x)
3  void main(){
4    double f, x, y;
5    printf("input x, y: ");
6    scanf("%lf, %lf", &x, &y);
7    /*输入的两个数据之间用逗号分隔*/
8    f=F(x)*F(y+1);
9    printf("f=%f\n", f);
10 }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3    double f, x, y;
4    printf("input x, y: ");
5    scanf("%lf, %lf", &x, &y);
6    /*输入的两个数据之间用逗号分隔*/
7    f= (x)*(x)*(x)+(x)*(x) *
8    (y+1)*(y+1)*(y+1)+(y+1)*(y+1);
9    printf("f=%f\n", f);
10 }
```

带参数的宏定义

- 将F(x)重写为

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

```
1  #include <stdio.h>
2  #define F(x)  (x)*(x)*(x)+(x)*(x)
3  void main(){
4    double f, x, y;
5    printf("input x, y: ");
6    scanf("%lf, %lf", &x, &y);
7    /*输入的两个数据之间用逗号分隔*/
8    f=F(x)*F(y+1);
9    printf("f=%f\n", f);
10 }
```

```
1  #include <stdio.h>
2  #define F(x)  ((x)*(x)*(x)+(x)*(x))
3  void main(){
4   double f, x, y;
5   printf("input x, y: ");
6   scanf("%lf, %lf", &x, &y);
7   /*输入的两个数据之间用逗号分隔*/
8   f=F(x)*F(y+1);
9   printf("f=%f\n", f);
10 }
```

带参数的宏定义

- 改写后程序预处理宏替换前后如下

```
1  #include <stdio.h>
2  #define F(x) ((x)*(x)*(x)+(x)*(x))
3  void main(){
4   double f, x, y;
5   printf("input x, y: ");
6   scanf("%lf, %lf", &x, &y);
7   /*输入的两个数据之间用逗号分隔*/
8   f=F(x)*F(y+1);
9   printf("f=%f\n", f);
10 }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3    double f, x,y;
4    printf("input x, y: ");
5    scanf("%lf, %lf", &x, &y);
6    /*输入的两个数据之间用逗号分隔*/
7    f= ((x)*(x)*(x)+(x)*(x)) *
8    ((y+1)*(y+1)*(y+1)+(y+1)*(y+1));
9    printf("f=%f\n", f);
10 }
```

结果完全正确

带#的宏定义

- 带#的宏定义
  - “#define” 中有两种情况可以使用#
  - 一种是变量的**字符串化**（stringizing）

#标识符 -> ”标识符”

```
1  #include <stdio.h>
2  #define PR(x) printf("%s=%d\n", #x, x)
3  void main(){
4     int a=15, b2=123;
5     PR(a);
6     PR(b2);
7  }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3   int a=15, b2=123;
4   printf ("%s=%d\n", "a", a);
5   printf ("%s=%d\n", "b2", b2);
6  }
```

```
a=15
b2=123
请按任意键继续……
```

带#的宏定义

- 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>

```
1  #include <stdio.h>
2  #define MP(x) printf("%d", a##x)
3  void main(){
4     int a1=2, a5=4;
5     MP(1);
6     MP(5);
7  }
```

经预处理后

```
1  #include <stdio.h>
2  void main(){
3     int a1=2, a5=4;
4     printf("%d", a1);
5     printf("%d", a5);
6  }
```

```
24请按任意键继续……
```

带#的宏定义

- 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
  - 如果将前面的程序修改，改为以下程序则编译不通过

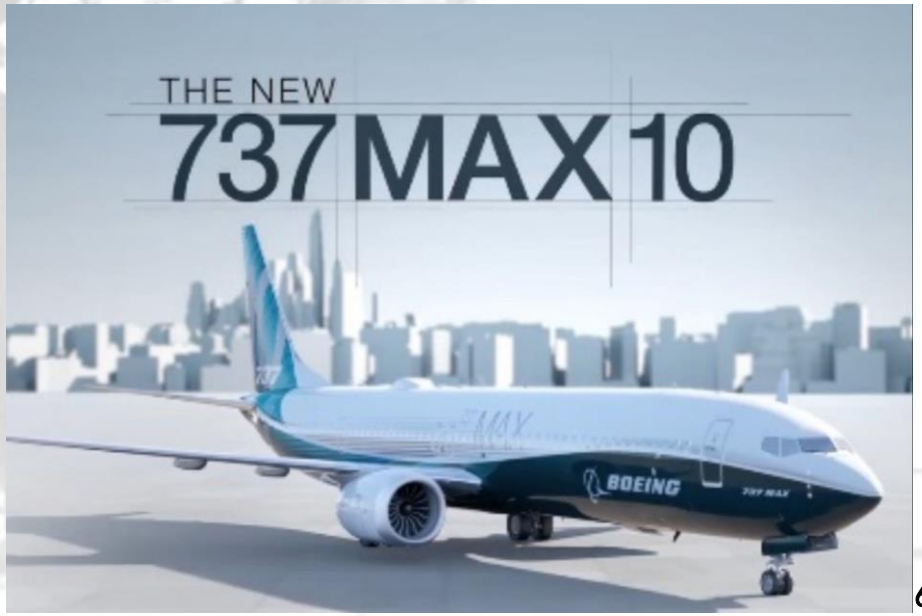
```
1  #include <stdio.h>
2  #define MP(x) printf("%d", a##x)
3  void main(){
4    int a1=2, a5=4, i;
5    i=1; MP(i);
6    i=5; MP(i);
7  }
```

error C2065: “ai”：未声明的标识符

因为 MP(i); 宏展开后为：printf(“%d\n”,ai);
宏定义替换**只进行简单的替换**，而不考虑i的值

程序设计中的严谨性

- 机器是依靠逻辑来识别语句，逻辑必须**严谨**缜密
  - 前面例题中的宏定义编写时如果不仔细思考，很容易就会使程序出现错误的运行结果
  - 遇到bug后善加利用错误信息，**快速定位bug并解决**
  - 波音737 MAX飞机自动驾驶程序有一个隐藏地很深的bug，这个修复拖了近四个月，直到埃航事故发生
  - 编程世界里从来没有“**差不多**”，只有**严谨和准确**



本节重点

- 赋值、算术、关系、逻辑等运算符与表达式
  - 运算符含义、运算优先级

一目运算符>！ > 算术运算符 >关系运算符>&&>||>赋值运算>逗号运算符

  - 类型转换：**自动**与**强制**
- 标准函数
- 宏定义
  - 符号常量的宏定义
  - 带参数的宏定义
  - 带#的宏定义

本节作业

- 第四次作业：课本第四章习题1, 3, 5, 8, 9, 11
  - 写出解答过程，电子版 or 纸质版完成后拍照上传到word文档**上传到网络学堂**
- 第三次上机实验：课本第四章习题12, 14
  - 上机实验请将**源代码**和**运行结果截图粘贴到word文档**中提交到网络学堂

本节作业

- 附加作业
  - 输入一个四位数$\overline{abcd}$，首先分别打印其各位数字a,b,c,d。之后，判断该四位数是否是a,b,c,d各种所有可能的排列中最大的，若是则输出1，反之则输出0。
  - 输入三角形三边长a,b,c，判断该三角形是否为钝角三角形，若是则输出1，反之则输出0。
  - 输入任意整型数a，再令b=(a+=1)&&(a+=a*=a-=a=++a)，之后打印a,b的值，并分析原因


THANKS