第 5 章　循环控制结构

本章导读

在进行程序设计时，仅仅使用前面学过的顺序结构和选择结构，往往解决不了一些较复杂的问题，比如累加、求一个班学生的平均分等。C 语言还提供了一种重要的控制结构——循环结构，利用循环结构可以解决复杂的、重复性的操作。循环结构的作用是使某段程序重复的执行，具体循环的次数会根据某个条件来决定。循环结构的应用非常普遍，使用起来也比较灵活，熟练掌握循环结构对学习编程是非常重要的。循环结构主要包括三种基本形式：while 语句、for 语句、do...while 语句。除了这三种常见形式之外，还有一种 goto 语句，不过这种语句一般不提倡使用。

本章主要介绍循环结构的三种基本语句及其特点，重点讲解常用的循环算法和编程方法，使读者能够熟练运用这三种基本循环控制结构编写程序。

本章知识要点

- while 语句的一般形式及应用
- for 语句的一般形式及应用
- do...while 语句的一般形式及应用
- 多重循环结构的使用
- break 语句和 continue 语句

5.1　while 语句

while 语句属于“当型”循环。“当型”循环是指在循环条件成立时，程序就一直执行循环体语句。while 语句的一般形式如下：

```
while  (表达式)
    循环体语句
```

while 语句的执行过程：首先计算 while 后圆括号内的表达式，当表达式的值为真（非零）时，执行循环体语句，然后继续判断表达式的值，重复上述执行过程，只有当表达式为假（零）时才退出循环，程序跳转到循环体后面的第一行代码处执行。流程图如图 5-1 所示。

说明：

（1）while 是关键字。while 后圆括号内的表达式一般是条件表达式或逻辑表达式，但也可以是 C 语言中任意合法的表达式。

（2）循环体语句可以是一条语句，也可以是多条语句，如果循环体语句包含多条语句，则需要用一对花括号“{}”把循环体语句括起来，采用复合语句的形式。

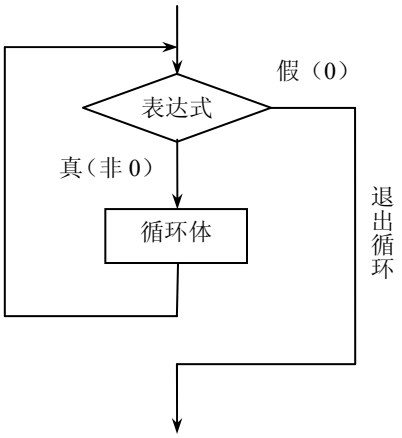

图 5-1　while 语句流程图

例 5-1　求前 100 个自然数的和，即：求 $\sum_{n=1}^{100} n$ 。

分析：这是一个简单的求和问题，需要连续的累加，因此只能使用循环结构实现重复累加的操作。设变量 sum 用于存放循环执行过程中的求和结果，设变量 n 为循环控制变量，同时也是每一次求和运算的基本数据项，然后可以利用 while 循环结构进行循环累加求和。

程序流程图如图 5-2 所示。

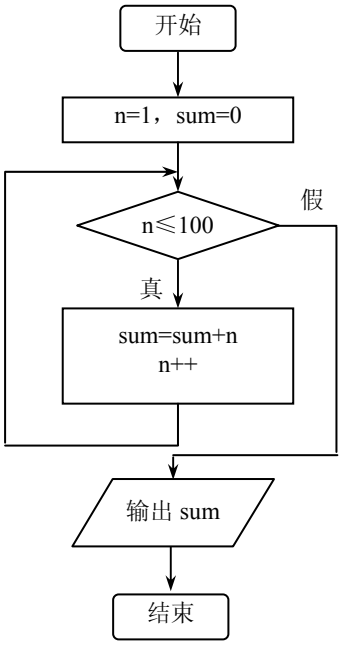

图 5-2　程序流程图

```
#include <stdio.h>
int main()
{
```

```
    int n,sum;
    n=1;sum=0;              /*变量赋初值*/
    while (n<=100)
    {
        sum=sum+n;          /*累加求和*/
        n++;                /*修改基本数据项 n*/
    }
    printf("sum=%d\n",sum);
return 0;
}
```

程序运行结果为：

```
sum=5050
```

在读程序时，正确地分析语句的执行顺序，即正确判断语句的跳转以及确定此时变量的值是非常重要的，是能否正确理解程序的关键，下面就对例 5-1 的程序执行过程及变量值的变化进行具体的分析。

表 5-1　程序执行过程的具体分析

执行顺序	执行语句	执行结果	sum 的值	n 的值	说明
1	n=1;sum=0;		0	1	变量赋初值
2	计算表达式 “n<=100”	1<=100 结果为“真”			判断循环条件
3	sum=sum+n; n++;	sum←0+1， n←1+1，	1	2	执行循环体语句
4	计算表达式 “n<=100”	2<=100 结果为“真”			判断循环条件
5	sum=sum+n; n++;	sum←1+2， n←2+1，	3	3	执行循环体语句
…	…	…	4950	100	…
200	计算表达式 “n<=100”	100<=100 结果为“真”			判断循环条件
201	sum=sum+n; n++;	sum←4950+100， n←100+1，	5050	101	执行循环体语句
202	计算表达式 “n<=100”	101<=100 结果为“假”			判断循环条件
203	printf("sum=%d\n",sum);				退出循环体，执行循环体下面的语句

需要注意的几个问题：

（1）累加求和算法。这个程序采用的算法思想称为累加求和，即不断用新累加的值取代变量的旧值，最终得到求和结果，变量 sum 也叫“累加器”，初值一般为 0。累加求和尽管方法简单，但却是循环结构程序设计中经常采用的一种算法思想，后面的很多复杂程序最终都可

以转化为累加求和或类似累加求和的问题来解决。使用 C 语言的循环结构对若干数进行累加求和一般要包括以下几个步骤：

步骤 1：设置基本数据项的初值。（如上面程序中的 n=1）

步骤 2：设置存放结果变量的初值。（如上面程序中的 sum=0）

步骤 3：循环条件判断，若条件满足转步骤 4，否则转步骤 6。

步骤 4：累加并修改基本数据项。（如上面程序中的 sum=sum+n;n++;）。

步骤 5：转步骤 3。

步骤 6：结束并输出结果。

（2）必须给变量赋初值。在 C 语言中定义的变量必须要赋初值，即使变量的初值为 0，赋初值也不能省略。如果没有给变量赋初值，那么变量的初值就会是一个不可预知的数，结果将没有意义。例如本题中，读者可以省略赋值语句“sum=0;”，调试后查看结果。

（3）正确判断条件的边界值。当 n 得知为 100 时，程序将继续执行循环体，然后控制流程再次判断条件表达式，此时，n 的值为 101（见表 5-1），表达式“n<=100”结果为假，退出循环。退出循环后，循环控制变量 n 的值是 101，而不是 100。

（4）避免出现“死循环”。使用 while 循环一定要注意在循环体语句中出现修改循环控制变量的语句，使循环趋于结束，如在本例中的“n++;”，否则条件表达式的计算结果永远为“真”，就会出现死循环。

（5）可能出现循环体不执行。while 循环是先判断表达式的值，后执行循环体，因此，如果一开始表达式为假，则循环体一次也不执行。

（6）while 后面圆括号内的表达式一般为关系表达式或逻辑表达式，但也可以是其他类型的表达式，如算术表达式等。只要表达式运算结果为非 0，就表示条件判断为“真”；运算结果为 0，就表示条件判断为“假”。例如下面的几种循环结构，它们所反映的逻辑执行过程是等价的，均表示当 n 为奇数时执行循环体，否则退出循环。

```
while (n%2)          while (n%2= =1)          while(n%2!=0)
{                    {                        {
        …                    …                        …
}                    }                        }
```

有时，条件表达式可能只是一个变量，比如有以下程序段：

```
…
P=1;
    while (p)
    {
      …
      P=0;
      …
    }
```

例 5-2　使用 while 语句求 n！。

分析：该题与例 5-1 非常相似，只是把求和改成乘积。另外由于 n 的值并不确定，需要程序执行的时候由用户输入要用到输入函数。

```
#include <stdio.h>
int main()
```

```
{
    int n,i=1;
    double sum=1;
    printf("请输入一个正整数：");
    scanf("%d",&n);
    while (i<=n)
    {
        sum=sum*i;          /*累乘求积*/
        i++;                /*修改基本数据项 i*/
    }
    printf("%d!=%.0f\n",n,sum);
    return 0;
}
```

程序运行情况如下：

输入：请输入一个正整数：6↙

输出：6！=720

循环三要素之间的关系。循环变量赋初值、判断控制表达式和修改循环变量是所谓的“循环三要素”。一般来说，进入循环之前，应该给循环变量赋初值，确保循环能够正常开始；在控制表达式中判断循环变量是否达到循环的终止值；在循环体中对循环变量进行修改，以使循环正常的趋向终止。在编写程序时要注意它们的位置关系。循环控制变量的初值可能会影响控制表达式的设计和控制变量修改语句的语序。比如，把例 5-2 中循环变量的初值改为 0，则其他两个要素就要随之改变，修改后的程序如下：

```
int n,i=0;
double sum=1;
printf("请输入一个正整数：");
scanf("%d",&n);
while (i<n)
{
    i++;
    sum=sum*i;
}
printf("%d!=%.0f\n",n,sum);
```

此题虽然和例 5-1 非常像近，但仍有两个需要注意的问题：

（1）变量合理赋初值。变量初值的选取要根据实际情况，本例题中用来存放乘积结果的变量 sum 初值就应赋 1，而不是 0。

（2）防止出现数据溢出错误。累乘结果变量 sum 的结果虽然是整数，在这里不能定义成 int 型数据。由于 int 型变量可以存放数据的范围比较有限（根据编译环境不同有所不同），当用户输入的 n 值比较大时，就可能得到一个非常大的结果，为防止在计算阶乘时发生数据溢出错误，把 sum 定义成 double 类型（但还是要注意输入数据时不能太大）。

例 5-3 编写程序，输入一个字符序列，直至换行为止，统计出大写字母、小写字母、数字、空格和其他字符的个数。

分析：这是一个关于字符处理的问题，首先可以定义一个字符变量 ch，利用 getchar()函数把用户从键盘输入的字符逐个接收，存储在 ch 中，然后对 ch 进行判断分类。当读取的字符

不是换行符时重复执行循环体，直到遇到换行符为止。while 语句的条件表达式可以写成这样（ch!= '\n'）。

程序如下：

```
#include<stdio.h>
int main()
{
    char ch;
    int a,b,c,d,e;
    a=b=c=d=e=0;
    while((ch=getchar())!='\n')          /*接收从键盘输入的字符，并判断是否为换行符，遇到
                                           换行符则停止循环*/
    {
        if(ch>='A'&&ch<='Z')a++;         /*判断是否为大写字母*/
        else if(ch>='a'&&ch<='z')b++;    /*判断是否为小写字母*/
        else if(ch>='0'&&ch<='9')c++;    /*判断是否为数字*/
        else if(ch==' ')d++;
        else e++;
    }
    printf("%d,%d,%d,%d,%d\n",a,b,c,d,e);
    return 0;
}
```

程序运行情况如下：

输入：s059se*(&　De9@GF↙

输出：3,4,4,2,4

注意：

（1）表达式((ch=getchar())!='\n')的执行分两步，首先利用 getchar()函数从终端接收一个字符，存储在 ch 中，然后再判断 ch 是否为'\n'，不能省略内部的括号。如果写成如下形式：

```
(ch=getchar()!='\n')
```

调试结果就会发生错误，因为表达式的关系运算符“!=”运算优先级别高于赋值运算符“=”，上述语句相当于：

```
while(ch=(getchar()!='\n'))
```

即先把接收的字符与'\n'进行关系运算，再把关系运算的结果“真（1）”或者“假（0）”存储在 ch 中，这显然是错误的。

（2）从终端键盘向计算机输入时，是在用户按 Enter 键以后才将一批数据一起送到内存缓冲区中去的。有以下程序段：

```
char ch;
    while((ch=getchar())!='\n')
    {
        printf("%c",ch);
    }
```

程序运行情况：

输入：　abcdefg↙

输出：　abcdefg

结果并不是：

```
aabbccddeeffgg
```

5.2 for 语句

for 语句是循环控制结构中使用最为广泛的一种控制语句，它充分体现了 C 语言的灵活性。for 语句有时也被称为“计数”型循环，因为它特别适合已知循环次数的情况。但事实上，for 循环同样适用于循环次数不确定而只知道循环结束条件的情况。for 循环可以实现所有的循环问题，它是 C 语言中形式最灵活、功能最强大的一种循环控制结构。

for 语句的一般形式如下：

```
for(表达式 1;表达式 2;表达式 3)
    循环体语句
```

从语法形式上看，for 语句语法上要比 while 语句复杂，for 后面的圆括号内有三个表达式，并使用分号“;”分隔，这三个表达式的运算次数、运算时间以及在循环中发挥的作用各不相同。它的执行过程如下：

步骤 1：计算表达式 1。

步骤 2：计算表达式 2，若表达式 2 的值为“真”（非 0），则执行一次循环体语句，然后转步骤 3，若表达式 2 的值为“假”（0），则转步骤 4。

步骤 3：计算表达式 3，然后转步骤 2。

步骤 4：退出循环，执行 for 语句后面的其他语句。

for 语句执行的流程图如图 5-3 所示：

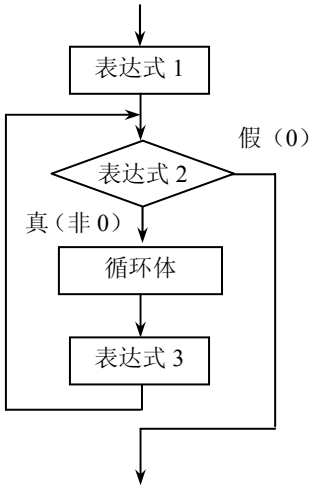

图 5-3 for 语句流程图

其中的表达式 1、表达式 2 和表达式 3 可以是任何一种 C 语言合法的表达式，但最常用、最简单的形式是这样的：即在表达式 1 中给循环变量赋初值；表达式 2 则是循环条件控制表达式；表达式 3 则实现循环控制变量的改变，使循环趋于结束。具体如下：

```
for(循环变量赋初值;循环条件;循环变量增值)
```

for 语句的功能等价于下面 while 语句：

```
表达式 1;
while (表达式 2)
```

```
{    循环体语句
     表达式 3;
}
```

如果用 for 语句改写例 5-1：

```
#include <stdio.h>
int main()
{
    int n,sum=0;
    for (n=1;n<=100;n++)    sum=sum+n;
    printf("sum=%d\n",sum);
    return 0;
}
```

由此可以看出，相对于 while 语句，for 语句在形式上更加简洁、方便。

例 5-4　设 n=30，编写程序，计算并输出 S(n)的值。

$S(n)=(1*2)/(3*4)-(3*4)/(5*6)+(5*6)/(7*8)+\ldots+(-1)^{(n-1)}*[(2n-1)*2n]/[(2n+1)*(2n+2)]+\ldots$

分析：这是一个累加求和的问题，题目明确是求三十项的和，因此选用 for 循环是最合适的。设变量 S 用于存放循环执行过程中的求和结果，设变量 n 为循环控制变量，每一次求和运算的数据项在题目中已经给出，即$(-1)^{(n-1)}*[(2n-1)*2n]/[(2n+1)*(2n+2)]$，数据项的值会随循环变量 n 的改变而改变。

程序流程图如图 5-4 所示。

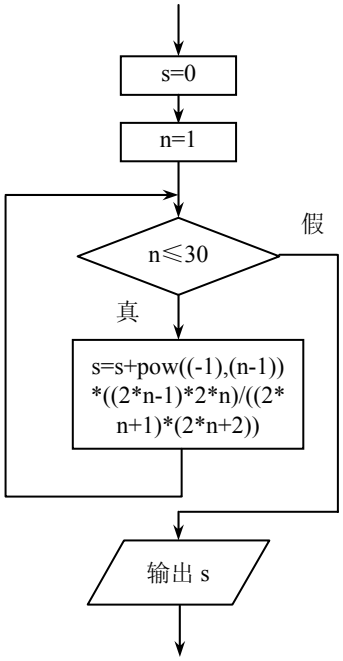

图 5-4　程序流程图

```
#include<stdio.h>
#include<math.h>
int main()
{
```

```
    int n;
    float s=0;
    for(n=1;n<=30;n++)
        s=s+pow((-1),(n-1))*((2*n-1)*2*n)/((2*n+1)*(2*n+2));
    printf("s(n)=%f",s);
    return 0;
}
```

运行结果为：

```
s(n)=–0.459873
```

注意：在程序中如果使用了数学函数，就必须在源文件开头添加预编译命令#include<math.h>，本题目中用到了一个数学函数 pow()，作用是进行幂运算，即计算 x 的 y 次幂的值。函数形式为 double pow(double x，double y)。

关于 for 语句的几点说明：

（1）循环体语句可以是简单语句也可以是使用一对花括号括起来的复合语句。如果是一个语句，也可以和 for 写在一行上，这样使程序看起来更加简洁；如果循环体包含多条语句，最好是另起一行，采用一对花括号括起来的复合语句形式，以增加程序的可读性。

（2）表达式的省略。for 语句中的三个表达式均可以省略，但是两个分号不能省略。

- 省略表达式 1

如果 for 语句中的表达式 1 被省略，表达式 1 的内容可以放在 for 循环结构之前。表达式 1 的内容一般来说是给循环变量赋初值，那么如果在循环结构之前的程序中循环变量已经有初值，那么表达式 1 就可以省略，但分号不能省。比如例 5-4 中，for 语句中如果省略表达式 1，可以改写成如下形式：

```
…
int n=1;
float s=0;
for(;n<=30;n++)
    s=s+pow((-1),(n-1))*((2*n-1)*2*n)/((2*n+1)*(2*n+2));
printf("s(n)=%f",s);
…
```

- 省略表达式 2

如果表达式 2 省略，就意味着每次执行循环体之前不用判断循环条件，循环就会无休止的执行下去，就形成了“死循环”。例 5-4 如果省略表达式 2，形式如下：

```
…
int n;
float s=0;
for(n=1;;n++)
    s=s+pow((-1),(n-1))*((2*n-1)*2*n)/((2*n+1)*(2*n+2));
printf("s(n)=%f",s);
…
```

相当于如下 while 循环：

```
…
int n=1;
float s=0;
```

```
while(1)
{
    s=s+pow((-1),(n-1))*((2*n-1)*2*n)/((2*n+1)*(2*n+2));
    n++;
}
printf("s(n)=%f",s);
…
```

- 省略表达式 3

如果表达式 3 省略，则必须在程序中另外添加修改循环变量值的语句，保证循环能够正常结束。例 5-4 如果省略表达式 3，程序可以改写成如下形式：

```
int n;
float s=0;
for(n=1;n<=30;)
{
    s=s+pow((-1),(n-1))*((2*n-1)*2*n)/((2*n+1)*(2*n+2));
    n++;
}
printf("s(n)=%f",s);
```

- 同时省略表达式 1 和表达式 3

如果表达式 1 和表达式 3 同时省略，只有表达式 2，也就是说只有循环条件，那就和 while 循环功能一样。下面两段程序是等价的。

```
while (n<=100)
{
   sum=sum+n;
   n++;
}
```

等价于：

```
for (;n<=100;)
{
    sum=sum+n;
    n++;
}
```

- 同时省略三个表达式

当然，for 循环的三个表达式也可同时省略，即：

```
for (;;)
{
    …
}
```

这种形式就会使循环体一直执行下去，形成“死循环”。

（3）表达式 1 和表达式 3 可以和循环变量无关。前面讲到，一般来说表达式 1 是给循环变量赋初值，表达式 3 是修改循环变量的值。但表达式 1 和表达式 3 的内容也可以和循环变量完全无关。

用 for 语句实现求 1 到 100 的和，程序如下：

```
int n,sum=0;
```

```
for (n=1;n<=100;n++)    sum=sum+n;
printf("sum=%d\n",sum);
```

也可以这样写：

```
int sum,n=0;
for(sum=0;n<100;sum=sum+n)    n++;
printf("sum=%d\n",sum);
```

第二种形式虽然结果也正确，但和第一种形式相比，程序的可读性和可维护性就大大降低了。可见，虽然 for 语句使用起来形式非常灵活，但是一般来说还是要遵从常用的形式，不要在表达式 1 和表达式 3 中出现和循环控制变量无关的内容。

请思考，在第二种形式中，为什么 n 的初值设为 0 而不是 1，循环条件也和第一种形式不同。

（4）表达式 1 和表达式 3 可以是一个简单的表达式，也可以是逗号表达式，既包含一个以上的简单表达式，中间用逗号隔开。

比如，在以后的学习中会遇到一些较为复杂的问题，和循环控制相关的变量可能一个以上，如以下程序段：

```
for(i=0,j=10;i<=j;i++,j--)
{
...
}
```

例 5-5　编写程序，输出所有的水仙花数。水仙花数是指一个 3 位数，其各位数字的立方和等于该数本身。例如：$153=1^3+5^3+3^3$，所以 153 就是水仙花数。

分析：因为水仙花数是一个 3 位数，所以可以定义一个变量 i，使 i 在 100～999 之间循环，逐个判断 i 是否为水仙花数，选用 for 循环最合适。另外，题目中要求求各位数字的立方和，这种问题常用“/”和“%”两种运算结合使用来解决。

程序如下：

```
#include<stdio.h>
int main()
{
    int a,b,c,i;
    for(i=100;i<=999;i++)
    {
        a=i/100;                /*求出 i 的百位数字*/
        b=i/10%10;              /*求出 i 的十位数字*/
        c=i%10;                 /*求出 i 的个位数字*/
        if(i==a*a*a+b*b*b+c*c*c)
            printf("%d\n",i);
}
printf("\n");
```

运行结果为：

```
153
370
371
407
```

注意：

（1）在计算机解决实际问题时，常常会用到类似本程序的“穷举法”。“穷举法”解决的问题一般具有这种特点：如果问题有解，一组或多组，必定全在某个集合中；如果这个集合内无解，集合外也肯定无解。这样，在解决问题时，就可以将集合中的元素一一列举出来，验证是否为问题的解。本题就是一一验证 100～999 之间所有的数，最终找出答案。

（2）程序中在做是否相等关系判断（i==a*a*a+b*b*b+c*c*c）使用到了关系运算符“==”，而不是“=”，后者是赋值运算符，在 C 语言中这两种运算符形式是不一样的，要注意区别。

5.3　do…while 语句

do…while 语句属于“直到型”循环，循环体语句一直循环执行，直到循环条件表达式的值为假为止，语句一般形式如下：

```
do
    循环体语句
while{表达式};
```

执行过程：先执行循环体语句，然后计算 while 后圆括号内的表达式，当表达式为“真”（非 0）时，则再次执行循环体语句，重复上述操作直到表达式为“假”（0）时退出循环。其中循环体语句可以是简单语句也可以是用一对花括号“{}”括起来的复合语句。流程图如图 5-5 所示。

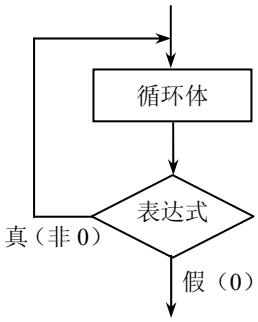

图 5-5　do…while 语句流程图

说明：

（1）do…while 语句中“while{表达式};”后面的分号是不能省略的，这一点是和 while 语句不一样的。

（2）do…while 语句是先执行循环体语句，后判断表达式，因此无论条件是否成立，将至少执行一次循环体。而 while 语句是先判断表达式，后执行循环体语句，因此，如果表达式在第一次判断时就不成立，则循环体一次也不执行。

while 语句和 do…while 语句的比较：

一般来说，对于同一个问题，使用 while 语句或 do…while 语句结果是一样的，也就是说，只要循环体相同，其结果也会相同。比如求 1～100 的和，如果使用 do…while 语句形式如下：

```
int n,sum;
n=1;sum=0;
do
{
    sum=sum+n;
    n++;
}
while (n<=100);
printf("sum=%d\n",sum);
```

但二者并不完全等价，当第一次进行循环时，while 后面的表达式就不成立，那么对于 while 循环来说，循环体语句一次也不执行，程序直接跳过循环结构，执行下面的语句；对于 do…while 循环来说，循环体语句还是要执行一次才跳出循环结构。例如以下两段程序：

```
#include <stdio.h>
int main()
{
    int n,sum=0;
    scanf("%d",&n);
    while (n<=10)
    {
        sum=sum+n;
        n++;
    }
    printf("sum=%d\n",sum);
return 0;
}
```

程序段 1

```
#include <stdio.h>
int main()
{
    int n,sum=0;
    scanf("%d",&n);
    do
    {
        sum=sum+n;
        n++;
    }
    while (n<=10);
    printf("sum=%d\n",sum);
return 0;
}
```

程序段 2

程序运行情况如下：

第一次运行：

输入：8↙

输出：sum=27

第二次运行：

输入：10↙

输出：sum=10

第三次运行：

输入：11↙

输出：sum=0

输入：8↙

输出：sum=27

输入：10↙

输出：sum=10

输入：11↙

输出：sum=11

对以上例子进行分析，当输入 n 的值小于或等于 10 时，两段程序输出的结果是一样的，当输入 n 的值为 11 时，两段程序输出的结果就不同了。当 n=11 时，对于 while 循环来说，第一次判断表达式“n<=10”结果为假，循环体一次也没有执行，直接输出 sum 的初值；对于 do…while 循环来说，程序先执行一次循环体，sum 的值变为 11，再判断表达式“n<=10”结果为假，退出循环结构。

例 5-6　编写程序，实现对用户输入口令的校验。用户输入的口令如果与预设口令不一致，则需要重新输入，直到与预设口令一致为止。

分析：定义一个字符型变量 C 用来存放用户输入的口令。循环的条件是用户输入的口令和预设的口令不一致，用户需要先输入口令然后进行判断，因此选用 do…while 循环更合适一些。

程序如下：

```
#include<stdio.h>
int main()
{
    char c;
    do
    {
        c=getchar();            /*接收用户输入的口令*/
    }while(c!='A');             /*假定预设口令是字符'A'*/
    printf("校验成功\n");
    return 0;
}
```

程序运行情况如下：

```
输入：D↙
      R↙
      b↙
      A↙
输出：校验成功
```

例 5-7　用公式 $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots$ 求 π 的近似值，直到最后一项的绝对值小于 10^{-6} 为止。

分析：本程序属于累加求和问题，可以定义浮点型变量 d 存放每一个基本数据项，注意题目中相邻基本数据项的符号不同，因此定义变量 sign 表示当前数据项的符号，初值为正号，即：sign=1，每循环一次，都使 sign 的符号取反，即：sign=-sign，其他步骤与一般的累加求和问题相同。

```
#include<stdio.h>
#include <math.h>
int main()
{
    double n,d,pi;
    int sign;
    sign=1;
    d=1.0;pi=0.0;n=1.0;
    do
    {
        pi=pi+d;
        n=n+2;
        sign=-sign;             /*改变数据项的符号*/
```

```
            d=sign/n;                    /*求出数据项*/
        }
        while (fabs(d)>=1.0e-6);
        pi=4.0*pi;
        printf("pi=%10.7f\n",pi);
        return 0;
    }
```

程序运行结果为：

```
pi= 3.1415907
```

注意：语句的先后顺序有时也非常重要，比如例 5-7 如果改写成如下形式：

```
double n,d,pi;
int sign;
sign=1;
d=1.0;pi=0.0;n=1.0;
do
{
    n=n+2;
    sign=-sign;
    d=sign/n;
    pi=pi+d;
}
while (fabs(d)>=1.0e-6);
pi=4.0*pi;
printf("pi=%10.7f\n",pi);
```

程序运行结果为：

```
pi= -0.8584053
```

结果显然不正确，只是修改了循环体中的一个语句的顺序，结果就会产生错误，如果将程序在此基础上作进一步相应的修改，如下：

```
double n,d,pi;
int sign;
sign=1;
d=1.0;pi=1.0;n=1.0;
do
{
    n=n+2;
    sign=-sign;
    d=sign/n;
    pi=pi+d;
}
while (fabs(d)>=1.0e-6);
pi=pi-d;
pi=4.0*pi;
printf("pi=%10.7f\n",pi);
```

程序运行结果为：

```
pi= 3.1415907
```

由此可以看出，变量初值改变了，循环体中语句的顺序就要做相应的调整，同时循环的次数可能也会受到影响，在编写程序时一定要注意考虑这些因素。请思考：为什么在循环体后要添加语句“pi=pi-d;”。

三种循环的比较：

上面介绍的三种循环语句 while、do…while 和 for 形式虽然不同，但主要结构成分都是循环三要素。三种语句都可以实现循环，一般来说，可以互相替代。但它们也有一定的区别，使用时应根据语句特点和实际问题需要选择合适的语句。它们的区别和特点如下：

（1）while 和 do…while 语句一般实现标志式循环，即无法预知循环的次数，循环只是在一定条件下进行；而 for 语句大多实现计数式循环。

（2）一般来说，while 和 do…while 语句的循环变量赋初值在循环语句之前，循环结束条件是 while 后面圆括号内的表达式，循环体中包含循环变量修改语句；一般 for 循环则是循环三要素集于一行。因此，for 循环语句功能更强大，形式更简洁，使用更灵活。

（3）while 和 for 是先测试循环条件，后执行循环体语句，循环体可能一次也不执行。而 do…while 语句是先执行循环体语句，后测试循环条件，所以循环体至少被执行一次。

知道了三种循环各自的特点，在实际使用时就要根据特点合理选择。

5.4　多重循环结构

在处理实际问题时，有时仅仅使用前面学过的循环是不够的，可能在已有循环结构的循环体语句中还需要包含循环结构，这就是多重循环。

一个程序中的多个循环语句之间存在两种关系：并列关系和嵌套关系。循环不允许有交叉。循环之间的关系如图 5-6 所示。

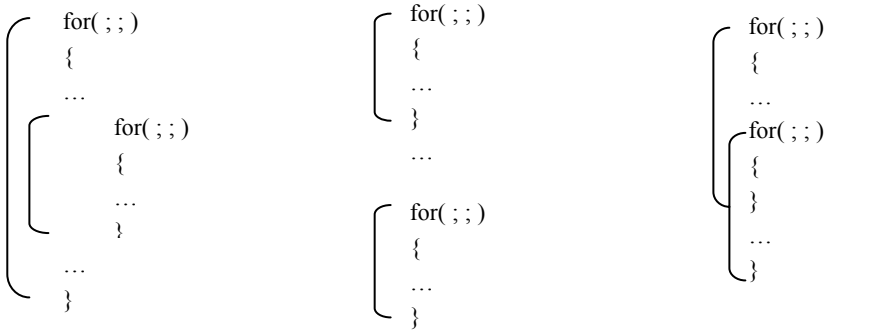

图 5-6　循环之间的关系

循环的嵌套是指一个循环语句的循环体内包含另一个完整的循环结构。前述三种循环结构（while 循环、for 循环、do…while 循环）可以任意组合嵌套。例如：

```
（1）while (     )
     {
                …
          while   (      )
          {
                   …
          }
               …
     }
（2）for (     ;      ;     )
     {
          …
        while   (      )
        {
                   …
        }
          …
     }

（3）do
     {
          …
        for   (   ;    ;   )
        {
            …
        }
          …
     } while (     )
（4）for (     ;      ;     )
     {
          …
        for (     ;      ;     )
        {
            …
        }
          …
     }
```

内层　外层　内层　外层　内层　外层　内层　外层

这种嵌套层次数为两层的循环嵌套称为双重循环嵌套，它的执行过程是：首先进行外层循环的条件判断，当外层循环条件成立时顺序执行外层循环体语句，遇到内层循环，则进行内层循环条件判断，并在内层循环条件成立的情况下反复执行内层循环体语句，当内层循环因循环条件不成立而退出后重新返回到外层循环并顺序执行外层循环体的其他语句，外层循环体执行一次后，重新进行下一次的外层循环条件判断，若条件依然成立，则重复上述过程，直到外层循环条件不成立时，退出双重循环嵌套，执行后面其他语句。例如下面循环嵌套形式的程序流程图如图 5-7 所示。

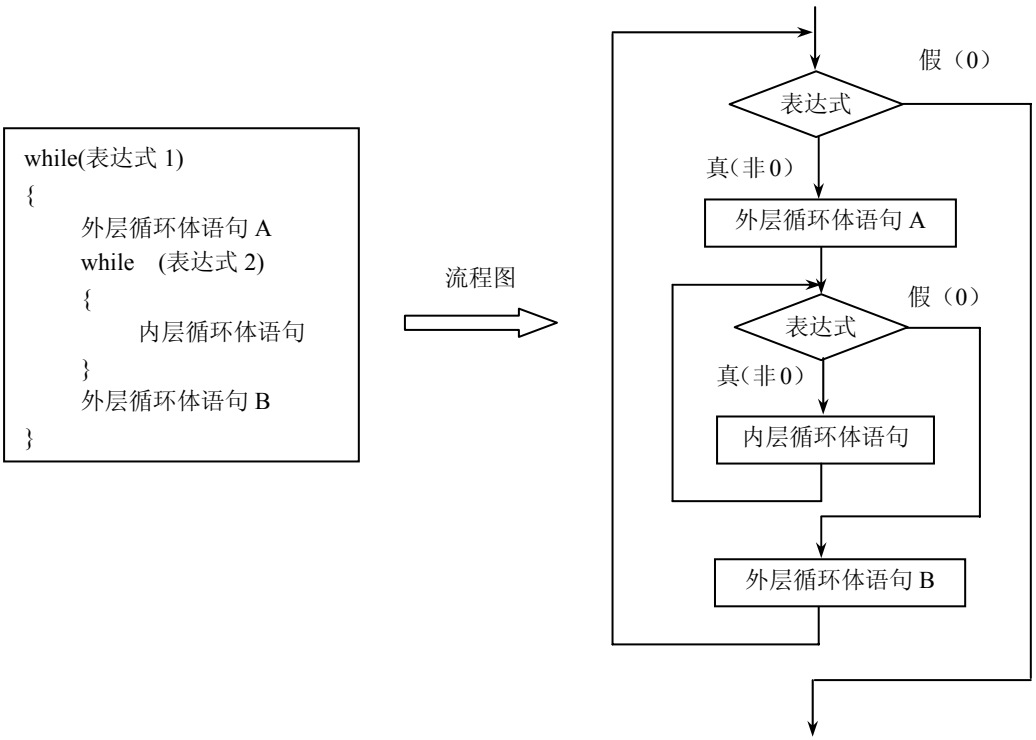

图 5-7　双重循环嵌套流程图

多重循环不仅包含双重循环结构，C 语言还允许循环结构的多重嵌套。如果一个循环的外面有两层循环就叫三重循环，如图 5-8 所示就是一个三重循环结构。当然，还允许有四重、五重等更多重循环。理论上嵌套可以是无限的，但一般使用两重或三重的比较多，若嵌套层数太多，就降低了程序的可读性和执行效率。

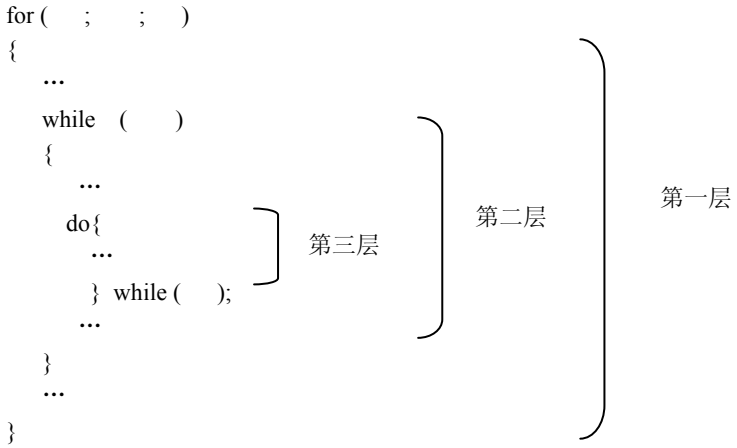

图 5-8　三重循环结构

例 5-8　编写程序，输出 1000 以内所有的完数。如果一个整数的因子之和等于这个数本身，这个数就被称为完数。例如：1、2、3 是 6 的因子，并且 6=1+2+3，所以 6 是完数。

分析：此题应该分成两步来做。

第一步：判断一个数 n 是否为完数。可以定义一个变量 s 作为“累加器”，此问题仍然需要用前面讲过的“穷举法”，从 1～n-1 逐一去除 n，如果能除尽，就说明是 n 的因子，把它累加到 s 上。可以选用 for 循环。

第二步：外层循环对 1000 以内的所有正整数一一进行判断，利用第一步的方法，逐个判断 n 的因子之和 s 是否等于 n。若相等，则显示输出。

同样选用 for 循环的程序思路如下：

```
for(n=2;n<=1000;n++)
{
    求 n 的所有因子之和赋给 s
    若 n=s，则显示输出
}
```

把上面的内循环改成代码，程序如下：

```
#include<stdio.h>
int main()
{
    int i,n,s;
    for(n=2;n<=1000;n++)          /*外循环*/
    {
        s=0;
        for(i=1;i<n;i++)          /*内循环，求出 n 的所有因子之和*/
            if(n%i==0)
            s+=i;
        if(n==s)                  /*判断 s 是否等于所有因子之和*/
            printf("%d\n",n);
    }
    return 0;
}
```

运行结果为：

```
6
28
496
```

例 5-9 打印九九乘法口诀表。

分析：乘法口诀表的形式如下：

```
1*1=1
1*2=2    2*2=4
1*3=3    2*3=6    3*3=9
1*4=4    2*4=8    3*4=12   4*4=16
…
1*9=9    2*9=18   3*9=27   4*9=36   5*9=45   6*9=54   7*9=63   8*9=72   9*9=81
```

九九乘法口诀表是一个二维图文表，这种表的处理常采用双重循环来实现。外循环控制输出行，内循环控制输出某行中的具体内容（即列）。求解此类问题的关键是分析图表的规律，九九乘法表的规律如下：

（1）乘法表共有 9 行。用外循环控制行，是定数循环，选用 for 循环比较合适。

（2）每行算式个数规律：第几行就有几列算式。用内循环输出每行的算式，内循环每执行一次，输出一个算式，因此内循环执行次数=外循环变量的值。内循环每次也是定数循环，选用 for 循环。

（3）每个算式既与所在行有关，又与所在列有关，规律是：列*行=积。

程序如下：

```
#include<stdio.h>
int main()
{
    int i,j;
    for(i=1;i<=9;i++)                    /*外循环控制输出行*/
    {
        for(j=1;j<=i;j++)                /*输出该行的内容*/
            printf("%2d*%d=%2d",j,i,i*j);
        printf("\n");                    /*每行结束后，输出换行*/
    }
    return 0;
}
```

注意：如果是多重循环，外循环和内循环应选用不同的循环控制变量。

5.5　break 语句和 continue 语句

前面讲到的三种循环结构 while 语句、for 语句和 do…while 语句，在一般情况下只有当循环控制条件为假时，循环才会结束，break 语句和 continue 语句则用于改变控制流。

5.5.1　break 语句

break 语句可以使流程跳出 switch 结构，它也可以用在 while 语句、for 语句和 do…while 语句中。当 break 用于这三种循环语句时，可使程序跳出本层循环结构，接着执行循环体下面的语句。其一般形式如下：

```
break;
```

比如下面输出圆面积的例子，要求当圆的面积大于 100 时停止输出。程序如下：

```
int r;
float area,pi=3.14159;
for(r=1;r<=10;r++)
{
    area=pi*r*r;
    if(area>100) break;
    printf("r=%d,area=%f\n",r,area);
}
```

当 r=6 时，条件 area>100 为真，执行 break 语句，提前结束循环，即不再继续执行其余的几次循环。程序跳转到 for 循环下面的语句接着执行。

说明：

（1）break 语句只能用于 while、for 和 do…while 循环语句以及 switch 语句中，不能用于

其他语句。

（2）如果 break 语句用在多重循环结构体中，使用 break 语句只能使程序退出 break 语句所在的最内层循环。如以下程序段：

```
int i,k;
for(i=1;i<=3;i++)
{
    printf("第%d 行：",i);
    for(k=1;k<=100;k++)
    {
        if(k>10)
            break;
        printf("%d,",k);
    }
    printf("\n");
}
```

程序执行结果如下：

```
第一行：1,2,3,4,5,6,7,8,9,10,
第二行：1,2,3,4,5,6,7,8,9,10,
第三行：1,2,3,4,5,6,7,8,9,10,
```

可见，当程序执行 break 语句时，仅跳出了内层的 for 循环，跳转到 printf("\n");语句接着执行，外循环不受影响。

5.5.2 continue 语句

continue 语句的作用是结束本次循环，即跳过循环体中下面尚未执行的语句，接着进行下一次是否执行循环体的判断。其一般形式如下：

```
continue;
```

continue 语句只能用于循环结构中。

对于 while 和 do…while 语句，continue 语句使程序结束本次循环，跳转到循环条件的判断部分，根据条件判断是否进行下一次循环；对于 for 语句，continue 语句使程序不再执行循环体中下面尚未执行的语句，直接跳转去执行“表达式 3”，然后再对循环条件“表达式 2”进行判断，根据条件判断是否进行下一次循环。

例 5-10 输入若干学生的成绩，求平均值。

程序如下：

```
#include<stdio.h>
int main()
{
    int i,n,score;
    float sum=0,aver;
    printf("请输入学生的个数:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("请输入学生的成绩:");
```

```
            scanf("%d",&score);
            if(score<0||score>100)                    /*学生成绩输入有误*/
            {
                printf("输入成绩有误，请重新输入!\n");
                i--;                                  /*此次输入成绩不算，计数应减去 1*/
                continue;
            }
            sum=sum+score;
        }
        aver=sum/n;
        printf("%.2f\n",aver);
        return 0;
    }
```

程序运行情况如下：

```
输入：请输入学生的个数：5↙
      请输入学生的成绩：102↙
      输入成绩有误，请重新输入!
      请输入学生的成绩：22↙
      请输入学生的成绩：98↙
      请输入学生的成绩：-2↙
      输入成绩有误，请重新输入!
      请输入学生的成绩：78↙
      请输入学生的成绩：65↙
      请输入学生的成绩：80↙
输出：68.60
```

当程序执行时，用户输入的成绩如果不在 0～100 之间，即 if 语句的条件“score<0||score>100”成立，程序就会输出错误信息，计数变量 i 减去 1，执行 continue;语句，这时，程序就会结束本次循环，不再执行循环体中下面尚未执行的语句“sum=sum+score;”，直接跳转去执行“表达式 3（i++）”，接着判断“表达式 2（i<=n）”，决定是否进行下一次循环。

continue 语句和 break 语句的区别是：continue 语句只是结束本次循环，而不是终止整个循环的执行。而 break 语句则是结束当前所在循环过程，执行循环体后面的语句。比如有以下两个循环结构，如图 5-9 所示。

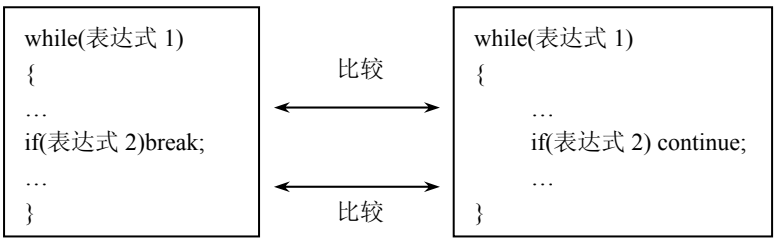

图 5-9　break 语句和 continue 语句比较

它们的流程图分别如图 5-10 所示。

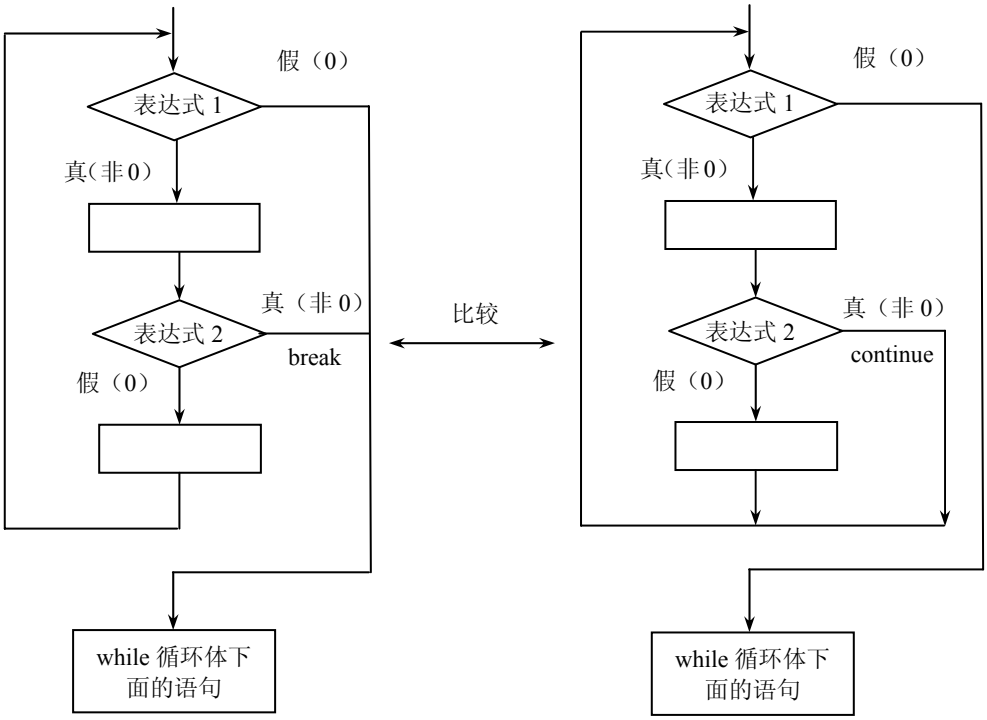

图 5-10　break 语句和 continue 语句流程图

注意比较当“表达式 2”为真时，两个流程图流程的转向。

5.6　应用程序举例

例 5-11　判断 m 是否为素数。

分析：所谓素数，就是一个正整数，除了本身和 1 以外并没有任何其他因子。例如 2，3，5，7 就是素数。

方法一：可以采用这种算法：定义一个整数 i 作为循环变量，定义一个整数 $k=\sqrt{m}$。让 m 被 2 到 k 除，如果 m 能被 2～k 之中任何一个整数整除，则提前结束循环，此时 i 必然小于或等于 k；如果 m 不能被 2～k 之间的任何一个整数整除，则在最后完成一次循环后，i 还要加 1，i=k+1，然后循环才能终止。因此，在循环结束之后判断 i 的值是否大于或等于 k+1，若是，则表明未曾被 2～k 之间任一整数整除过，因此 m 就是素数，否则 m 就不是素数。程序如下：

```
#include <stdio.h>
#include <math.h>
int main()
{
    int m,i,k;
    scanf("%d",&m);
    k=sqrt(m);
    for (i=2;i<=k;i++)
    if(m%i==0) break;                    /*m 已不是素数，不用再除了，跳出循环*/
    if(i>k) printf("%d is a prime number\n",m);
```

```
    else printf("%d is not a prime number\n",m);
    return 0;
}
```

程序运行情况如下：

输入：32↙

输出：32 is not a prime number

再次运行程序

输入：31↙

输出：32 is a prime number

方法二：还可以采用这种办法，先定义一个变量 flag，用它来表示 m 是否为素数，可以假定 flag 的值为 1 时表示 m 是素数，m 的值为 0 时表示 m 不是素数。这个变量 flag 通常被称为“标志变量”，在以后的学习中还会碰到这种变量。可以事先假定 m 是一个素数，即把 m 赋初值为 1，当在 m 被 2 到 k 除的循环过程中，如果 m 能被 2～k 之中任何一个整数整除，那么就把 flag 的值置为 0。这样，在循环结束时，通过 m 的值就可以判断出 m 是否为素数。程序如下：

```
#include <stdio.h>
#include <math.h>
int main()
{
    int m,i,k,flag;                          /*定义标志变量*/
    scanf("%d",&m);
    k=sqrt(m);
    flag=1;                                  /*假设 m 是素数*/
    for (i=2;i<=k;i++)
        if(m%i==0)
        {
            flag=0;                          /*表示 m 不是素数*/
            break;                           /*跳出循环*/
        }
    if(flag==1) printf("%d is a prime number\n",m);
    else printf("%d is not a prime number\n",m);
    return 0;
}
```

程序运行情况如下：

输入：85↙

输出：85 is not a prime number

再次运行程序

输入：79↙

输出：79 is a prime number

请思考：在方法二中，如果把下面的 if 语句：

```
if(flag==1) printf("%d is a prime number\n",m);
```

改为：

```
if(flag) printf("%d is a prime number\n",m);
```

这二者等价吗？

例 5-12 从键盘输入两个正整数 m 和 n，求它们的最大公约数和最小公倍数。

分析：求两个数的最大公约数有两种算法，求最小公倍数的方法为两个数的乘积除以它们的最大公约数。

方法一：根据最大公约数的数学定义，使用 for 循环查找既能整除 m 又能整除 n 的最大的数就是 m、n 的最大公约数。

```
#include <stdio.h>
int main()
{
    int m,n,k,max,x,y,z;
    printf("input m and n:\n");
    scanf("%d",&m);
    scanf("%d",&n);
    x=m;
    y=n;
    for (k=1;k<=(m<n?m:n);k++)                    /*循环变量 k 的最大值应是 m 和 n 中的较小数*/
        if (m%k==0&&n%k==0) max=k;
    z=x*y/max;                                    /*求最小公倍数*/
    printf("m 和 n 的最大公约数为：%d，\n 最小公倍数为：%d\n",max,z);
    return 0;
}
```

程序运行情况如下：

```
输入：input m and n:
      96↙
      56↙
输出：m 和 n 的最大公约数为：8，
      最小公倍数为：672
```

方法二：辗转相除法。求两个数的最大公约数可以使用辗转相除法，算法的思想如下：

首先定义一个变量 r，用来存储 m 除以 n 的余数。

步骤 1：将两个数中的大者放在 m 中，小者放在 n 中。

步骤 2：求 m 除以 n 的余数 r，即 r=m%n。

步骤 3：若 r 不等于 0，转步骤 4；若 r 等于 0，则此时的 n 就是最大公约数，转步骤 5。

步骤 4：把 n 的值赋给 m，把 r 的值赋给 n，即 m←n，n←r，然后转步骤 2。

步骤 5：跳出循环结构，执行循环结构的下一个语句。

程序如下：

```
#include<stdio.h>
int main()
{
    int m,n,r,x,y,z,k;
    printf("input m and n:\n");
    scanf("%d",&m);
    scanf("%d",&n);
    if(n>m)                    /*若 n>m，则交换 m、n 中的数据*/
        {k=m;m=n;n=k;}
```

```
    x=m;                    /*保存最初两个数的值给 x 和 y，以备求最小公倍数时使用*/
    y=n;
    r=m%n;
    while(r!=0)
    {
        m=n;
        n=r;
        r=m%n;
    }
    z=x*y/n;                /*求最小公倍数*/
    printf("m 和 n 的最大公约数为：%d，\n 最小公倍数为：%d\n",n,z);
    return 0;
}
```

程序运行情况如下：

输入：input m and n:

96↙

56↙

输出：m 和 n 的最大公约数为：8，

最小公倍数为：672

例 5-13　求 Fibonacci 数列前 30 项，每行输出 5 个数。

分析：

（1）问题背景：Fibonacci 数列是中世纪意大利数学家在《算盘书》中提出的一个关于兔子繁殖的问题：如果一对兔子每月能生一对小兔，而每对小兔在它出生后的第三个月里，又能开始生一对小兔，假定在不发生死亡的情况下，每个月有多少对兔子？

（2）通过分析可以得出每个月兔子的对数应该是：

月份	1	2	3	4	5	6	7	…
兔子数	1	1	2	3	5	8	13	…

通过观察可以发现，每个月的兔子数量是有规律可循的，即：

第 i 个月兔子的对数=第（i–1）个月兔子对数+第（i–2）个月兔子对数。

（3）算法设计思想：可以设 f1 表示第（i–2）个月兔子对数，f2 表示第（i–1）个月兔子对数，f3 表示第 i 个月兔子的对数。即：f3=f1+f2。

```
1    1    2    3    5    8    13
f1   f2   f3
     f1   f2   f3
          f1   f2   f3
               f1   f2   f3
                    f1   f2   f3
```

从数据可以看出，先从第一项开始，f1、f2 分别表示第一项和第二项，初值均为 1，第三项的值 f3=f1+f2。计算第四项的值：这时的 f3 表示的是第四项，那么 f1 表示的就应该是第二项，即刚才的 f2，f2 表示的就应该是第三项，即刚才的 f3。因此，在使用公式 f3=f1+f2 计算第四项的值之前，需要先把 f2 的值赋给 f1（f1←f2），把 f3 的值赋给 f2（f2←f3）。然后，再

使用公式计算，得出的值 f3 就是第四项的值。

以此类推，就可以求出后面各项的值。如下所示：

f3 的值：　　f1=1，f2=1，f3=f1+f2；

f4 的值：　　f1= f2，f2= f3，f3=f1+f2；

f5 的值：　　f1= f2，f2= f3，f3=f1+f2；

…

上面的分析可以得知，从数列的第三项开始，每一项的值都依赖于其前两项，这种方法叫递推法。递推算法的基本思想是：从初值出发，归纳出新值与旧值间的关系，直到推出所需值为止。即新值的求出依赖于旧值，不知道旧值就无法推导出新值，类似于数学上的递推公式。

程序代码如下：

```
#include <stdio.h>
int main()
{
    int f1,f2,f3,i;
    f1=1;f2=1;
     printf("%10d%10d",f1,f2);
    for(i=3; i<=30; i++)                    /*从第三项开始计算*/
    {
        f3=f1+f2;
        printf("%10d",f3);
       if(i%5==0) printf("\n");        /*每输出 5 个后换行*/
       f1=f2;
       f2=f3;
         }
     return 0;
}
```

程序运行结果为：

```
         1         1         2         3         5
         8        13        21         4        55
        89       144       233       377       610
       987      1597      2584      4181      6765
     10946     17711     28657     46368     75025
    121393    196418    317811    514229    832040
```

请思考：上面程序中的两个语句

```
f1=f2;
f2=f3;
```

如果交换顺序，即写成：

```
f2=f3;
f1=f2;
```

是否正确？

例 5-14　百钱买百鸡问题。这是中国古代数学家张丘建在他的《算经》中提出的问题。

问题大意为：公鸡 5 元一只、母鸡 3 元一只、小鸡 1 元三只，问用 100 元钱买 100 只鸡，公鸡、母鸡、小鸡各应多少只？

分析：本问题可使用穷举法实现。

设买公鸡 i 只，母鸡 j 只，小鸡 k 只，根据题义 i、j、k 应满足下面条件：

5i+3j+k/3=100

i+j+k=100

这是一个三元方程，只有两个算术式，方程会有多个解。所以此问题可归结为求不定方程的整数解。

由程序设计实现不定方程的求解与手工计算不同。在分析确定方程中未知数变化范围的前提下，可通过对未知数可变范围的穷举，验证方程在什么情况下成立，从而得到相应的解。在所有可能的买鸡方案中选出满足上述两个条件的母鸡、公鸡和小鸡数。由于公鸡 5 元一只，因此 100 元最多买 20 只公鸡；母鸡 3 元一只，100 元最多买 33 只母鸡；虽然小鸡 1 元三只，但最多只能够买 100 只小鸡。程序就需要用到三重循环，分别设三个循环变量 i，j，k，分别表示购买公鸡、母鸡和小鸡的数量。

程序如下：

```
#include "stdio.h"
int main()
{
    int i,j,k,n;
    int money;
    printf(" 公鸡   母鸡   小鸡\n",i,j,k);
    for (i=0;i<=20;i++)                    /*最外层循环控制公鸡数*/
        for (j=0;j<=33;j++)                /*二重循环控制母鸡数*/
            for (k=0;k<=100;k++)           /*三重循环控制小鸡数*/
            {
                n=i+j+k;
                money=5*i+3*j+k/3;
                if(k%3==0&&n==100&&money==100)
                    printf("%5d%5d%5d\n",i,j,k);
            }
    return 0;
}
```

程序运行结果为：

```
 公鸡    母鸡    小鸡
   0      25      75
   4      18      78
   8      11      81
  12       4      84
```

例 5-15　编写程序，输出如下所示图形。

```
D D D D D D D
  C C C C C
    B B B
      A
```

分析：此题仍然属于图形输出的问题，可以采用双重循环实现。外循环控制行输出，内循环控制每行输出的字符。图形的每行可视为由行前导空格和行中字母构成，且把每行的字母视为一个整体。其规律是行号与每行不同字符个数有确定的对应关系：

具体规律：

前导空格：第 i 行（1，2，3，4）对应的空格数为 i–1（0，1，2，3）

每行字母：第 i 行（1，2，3，4）对应的字母个数为 9–2*i（7，5，3，1）

程序如下：

```
#include <stdio.h>
int main()
{int i,j;
 for(i=1;i<=4;i++)
    {
      for(j=1;j<10+i;j++)
         printf(" ");
      for(j=1;j<=9-2*i;j++)
         printf("%c",69-i);
      printf("\n");
    }
  return 0;
}
```

说明：程序中每行输出的空格数并不是分析中的“i–1”，而是“10+i”，但这并不影响结果的输出。原因是输出每行前面的空格时关键是要保证行与行之间空格数的相对关系要正确，即相邻的下一行比上一行的空格数多一，只要这一点得到保证，图形就能正确的输出，至于每行内容前空格的绝对数目并不重要。程序中用到的“10+i”，只不过是让每行内容的前面都多加了十个空格，结果只会让输出图形整体右移十个字符的位置。

图形输出问题的一般方法：可以采用双重循环实现。外循环控制行输出，内循环控制每行输出的字符。输出具体内容时，要找出每行内容之间的规律，对具体的字符进行输出。每行各种字符的个数往往和行号是有一定关系的，可以利用 for 语句进行输出。每行或每列字符的内容即使不同，行与行或列与列字符之间一定存在某种联系，找出这种联系进行输出。必要时可以把每行的内容分为几部分分别进行输出，比如例 5-15。

5.7 错误解析

1．形成“死循环”。

当在 while、for 或 do…while 语句中的循环条件一直都为真时，就会形成“死循环”。其中，由于 for 循环形式的特点，一般“表达式 2”是循环条件控制表达式；“表达式 3”实现循环控制变量的改变，使循环趋于结束。因此，一般来说，使用 for 语句不容易出现“死循环”现象。使用 while 或 do…while 语句时就要特别注意。

比如以下程序：

```
int s,i;
s=0;
```

```
i=1;
while(i<=100)
{
    if(i%2==0)s=s+i;
}
printf("%d\n",s);
```

在循环结构语句中缺少改变循环变量 i 值的语句，致使循环控制条件一直为真，程序就会一直循环执行。

解决方法：在循环体语句中，特别注意 while 和 do…while 语句，必须有使循环变量改变的语句，以使循环趋于结束。

上面的程序段的正确表述如下：

```
int s,i;
s=0;
i=1;
while(i<=100)
{
    if(i%2==0)s=s+i;
    i++;
}
printf("%d\n",s);
```

另外还有一种情况，就是不能在 while 语句后面直接跟随分号。比如以下程序：

```
int s,i;
s=0;
i=1;
while(i<=100);
{
    if(i%2==0)s=s+i;
    i++;
}
printf("%d\n",s);
```

这样也会出现“死循环”现象。

2．首次循环条件不成立。

这种情况多出现在 while 语句中，程序在第一次进行循环条件的判断时，循环控制条件即为假，程序直接跳过循环体，不再执行循环语句。

比如以下程序：

```
int m,n,r,x,y,z;
scanf("%d",&m);
scanf("%d",&n);
x=m;
y=n;
while(r!=0)
{
    m=n;
    n=r;
```

```
        r=m%n;
    }
    z=x*y/n;
    printf("m 和 n 的最大公约数为：%d，最小公倍数为：%d\n",n,z);
```

这段程序是例 5-12 常见的错误程序，程序在进入循环之前没有给变量 r 赋值，导致第一次判断循环条件即不成立，程序无法顺利进入循环结构。

解决方法：一般的在进入循环结构之前要注意和循环控制条件相关的变量值，注意给这些变量赋值，使程序能够顺利进入循环结构。

3．使用多重循环时，循环变量一样。

在用到两重以上循环的嵌套时，内外重循环变量使用一个变量，导致程序错误。比如以下程序段：

```
for (i=0;i<=10;i++)
    for (i=0;i<=50;i++)
        for (i=0;i<=100;i++)
        {
            sum=sum+i;
            printf("%d",sum);
        }
```

这段程序用到了三重循环，循环变量一样，显然是错误的。

注：本程序段没有实际意义，只是为了说明这种错误形式。

解决方法：在循环的嵌套结构中，每一重的循环变量都不能一样。但一定要分清楚，循环结构之间的关系，如果是并列关系，就可以使用相同的循环变量。比如例 5-15 程序可以改写如下：

```
int i,j,k;
for(i=1;i<=4;i++)
{
    for(j=1;j<=4-i;j++)
        printf(" ");
    for(j=1;j<=2*i-1;j++)
        printf("*");
    printf("\n");
}
for(i=3;i>=1;i--)
{
    for(j=1;j<=4-i;j++)
        printf(" ");
    for(j=1;j<=2*i-1;j++)
        printf("*");
printf("\n");
}
```

4．在循环的嵌套结构中，语句的位置不对。

某个语句的具体位置是放在内循环外，还是放在内循环内，经常容易出错。比如下面程序段：

```
int i,n,s;
for(n=2;n<=1000;n++)
{
    for(i=1;i<n;i++)
        {
            s=0;
            if(n%i==0)
            s+=i;
        }
    if(n==s)
        printf("%d,",n);
}
```

这段程序是例 5-8 常见的错误形式。

解决方法：在用到循环的嵌套时，要一层一层进行分析，必要时可以像例 5-8 一样，先把一部分程序用文字表示，然后再把文字部分换成程序段。

本章小结

本章介绍的循环结构是基本控制结构中最重要的一种，这种结构用于实现需要重复执行某些操作的程序。本章主要介绍了循环结构的特点和几种基本形式：while 语句、do…while 语句和 for 语句等循环结构语句。三种基本循环结构的特点总结如下：

（1）while 语句。

一般形式

while　(表达式)
　　循环体语句

while 语句属于“当型”循环。while 循环也称“当”循环，当循环控制表达式的值为非零，执行循环体；当循环控制表达式的值为零值，不执行循环体，或者退出循环体。

注意：*在程序中一定要有使循环开始执行和使循环趋向结束的语句存在。*

（2）for 语句。

一般形式

for(表达式 1;表达式 2;表达式 3)
循环体语句

for 语句也被称为“计数”型循环，它特别适合已知循环次数的情况。for 语句的结构较为紧凑，有助于初学者养成良好的编写循环程序的习惯。当然，它也同样适用于循环次数不确定而只知道循环结束条件的情况。for 语句是 C 语言中形式最灵活，功能最强大的一种循环控制结构，它充分体现了 C 语言的灵活性。

（3）do…while 语句。

一般形式

do
循环体语句
while{表达式};

do…while 语句属于“直到型”循环。由于控制条件出现在循环体之后，循环体至少被执行一次。

break 和 continue 语句在循环结构中的作用：

- 在循环体中可以使用 break 和 continue 语句改变循环执行过程
- 使用 continue 语句，可以跳过本次循环体中那些尚未执行的语句。
- 在 while 或 do…while 循环体中出现 continue 语句，流程将直接跳到循环控制条件的测试部分；对于 for 循环，则跳到执行“表达式 3”的位置。
- 使用 break 语句可使流程跳出本层循环，尤其在多层次的循环结构中，利用 break 语句可以提前结束内层循环。

（4）多重循环结构。

循环的嵌套是指一个循环语句的循环体内包含另一个完整的循环结构。前述三种循环结构（while 循环、for 循环、do…while 循环）可以任意组合嵌套。如以下形式：

```
（1）for (   ;   ;   )
     {
       …
       while  (    )
       {                              内层          外层
             …
       }
       …
     }
```

```
（2）do
     {
         …
       for  (  ;   ;  )
       {                              内层          外层
         …
       }
         …
     } while (   )
```

两层的循环嵌套称为双重循环嵌套，它的执行过程是：首先进行外层循环的条件判断，当外层循环条件成立时顺序执行外层循环体语句，遇到内层循环，则进行内层循环条件判断，并在内层循环条件成立的情况下反复执行内层循环体语句，当内层循环因循环条件不成立而退出后重新返回到外层循环并顺序执行外层循环体的其他语句，外层循环体执行一次后，重新进行下一次的外层循环条件判断，若条件依然成立，则重复上述过程，直到外层循环条件不成立时，退出双重循环嵌套，执行后面其他语句。

多重循环不仅包含双重循环结构，C 语言还允许循环结构的多重嵌套，即三重或三重以上循环的嵌套。一般循环的嵌套只用到两重或三重。

习题五

1．编写程序，求 100～2000 之间所有 3 的倍数之和，当和大于 1000 时结束。

2．编写程序，计算并输出下面数列前 n 项的和（设 n=20,x=0.5），要求结果保留 3 位小数。cos(x)/x,cos(2x)/2x,cos(3x)/3x,…,cos(n*x)/(n*x),…（其中，cos(x)为余弦函数）。

3．编写程序，计算并输出下面数列前 20 项的和。要求结果保留 4 位小数。

数列为：2/1,3/2,5/3,8/5,13/8,21/13…

4．编写程序，求 $\sum_{n=1}^{20} n!$（即求 1！+2！+3！+4！+…+20！）

5．编写程序，读入一个整数，分析它是几位数。

6．编写程序，统计并逐行显示（每行 5 个数）在区间[10000,50000]上的回文数。回文数的含义是从左向右读与从右向左读是相同的，即对称，如 12321。

7．编写程序，求所有三位数中的素数。

8．使用双重循环输出以下图形：

```
1
1  2
1  2  3
1  2  3  4
1  2  3  4  5
```

9．编写程序，用双重循环输出下面的图形。

```
  *******
 *  *****  *
 **  ***  **
 ***  *  ***
 ****  ****
 ***  *  ***
 **  ***  **
 *  *****  *
  *******
```

10．一个球从 100 米高度自由落下，每次落地后又跳回原高度的一半，再落下。求它在第 10 次落地时，共经过多少米？第 10 次反弹多高？

11．猴子吃桃问题。猴子摘了若干个桃子，第 1 天吃掉一半多一个；第 2 天接着吃了剩下桃子的一半多一个；以后每天都吃剩余桃子的一半多一个，到第 8 天早上要吃时只剩下一个了。问小猴最初摘了多少个桃子。