第 5 章 循环结构程序设计

CHAPTER 5

现实生活中常常会遇到需要重复处理的问题，在这一类问题中有一些步骤是被重复执行的，反映到程序中表现为一组指令或程序段被有条件地反复执行，这种不断被重复执行的结构称为循环结构。循环结构是结构化程序设计的基本结构之一，它的特点是在给定条件成立时，反复执行某程序段，直到条件不成立时停止。其中给定的条件称为循环条件，反复执行的程序段称为循环体。

C 语言中提供了 goto 语句、while 语句、do while 语句和 for 语句实现循环结构，它们可以用来处理同一问题，一般情况下可以相互替换。但是结构化的程序设计不提倡使用 goto 语句，因为它强制改变程序的执行顺序，会给程序的运行带来不可预料的错误。本章主要学习 while、do while 和 for 这 3 种循环控制语句。

本章学习重点：

(1) 循环的概念及特点；

(2) while、do while 和 for 语句的语法格式及应用；

(3) break 语句和 continue 语句的语法格式及应用；

(4) 循环嵌套的应用。

本章学习目标：

(1) 理解循环的概念及特点；

(2) 掌握 while、do while、for 语句的语法格式及使用方法；

(3) 掌握 continue 语句和 break 语句在循环中的应用；

(4) 灵活运用 3 种循环语句及循环嵌套解决实际问题。

5.1 循环结构的引入

如果需要输出 1～10 的 10 个整数，该如何解决？最直接的方法是用 printf()函数输出这 10 个数。

```
printf("%d%d%d%d%d%d%d%d%d%d",1,2,3,4,5,6,7,8,9,10);
```

但这样书写比较麻烦，如果要求输出更多的数，如 1～100、1～1000，这种方法显然不合适，可以尝试使用另一种方法。

```
printf("%d",1);
printf("%d",2);
```

```
…
printf("%d",10);
```

在上面的方法中10条语句基本类似，后面的输出项由1到10，每次递增1，呈规律性变化。如果用变量i表示输出项，用i=i+1实现递增，可将程序改写为

```
int i=1;
printf("%d",i);i=i+1;
…
printf("%d",i);i=i+1;
```

10条语句

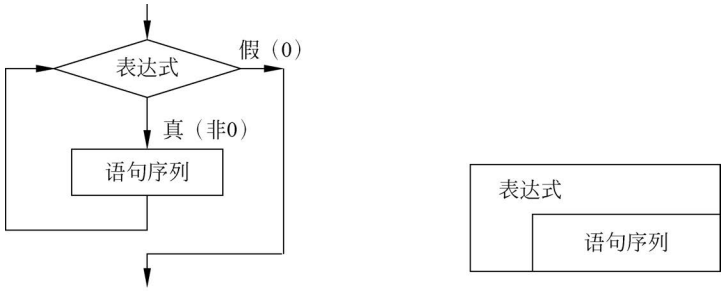
通过观察发现，“printf("%d ",i);”和“i=i+1;”这两条语句是重复执行的，可以将这两条语句作为循环体反复执行。如果输出1～10，则让它们重复执行10次；如果输出1～100，则重复执行100次。那么如何实现重复执行呢？可以借助C语言提供的循环控制语句来实现。

5.2　while 语句

在C语言中，可以使用while语句来实现循环，它的特点是通过判断循环控制条件是否满足来决定是否执行循环，一般形式为

```
while (表达式)
{
    语句序列
}
```

执行流程：先计算表达式的值，如果表达式的值是非0值(逻辑真)，表示循环条件成立，则执行语句序列。并再次计算表达式的值，如果其值仍是非0值(逻辑真)，则继续执行语句序列。此过程反复执行，直到表达式的值为0(逻辑假)时循环结束。while语句执行流程图如图5-1所示。



(a) while循环传统流程图　　(b) while循环N-S流程图

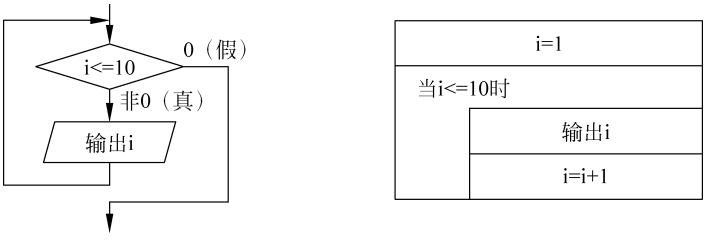
图5-1　while语句执行流程图

说明：

(1) 表达式为循环控制条件，可以是任何类型的表达式，一般情况是关系表达式或逻辑表达式。只要表达式的值非0(逻辑真)，则循环条件成立，执行语句序列。

(2) 语句序列是循环体，可以是一条语句，也可以是多条语句或空语句。如果是多条语句，要加花括号构成复合语句。

利用 while 循环语句解决前面的问题：输出整数 1～10，其算法流程图如图 5-2 所示。



(a) 传统流程图　　(b) N-S流程图

图 5-2　输出 10 个整数流程图

程序如下：

```
int main()
{
    i = 1;
    while (i < = 10)
    {
        printf(" % d\n",i);
        i = i + 1;
    }
    return 0;
}
```

使用 while 语句应注意的问题：

① 给循环变量赋初值。在程序中用来控制循环条件的变量称为循环变量。进入循环前，必须给循环变量赋初值。因为 C 语言不会自动给变量赋初值(除非是全局变量或静态局部变量，系统会为它们自动赋初值为 0)，如果没有给循环变量赋初值，则可能会得到一个该存储空间上一次运算的遗留值，所以导致程序运行的结果不能确定。

例如，

```
int i;
while (i < = 10)
{
    printf(" %d\n",i);
    i = i + 1;
}
```

运行结果为

```
-858993460
-858993461
-858993462
…
```

由于在进入循环体前没有给循环变量 i 赋初值，因此循环变量 i 获得了一个随机值 −858993460，而且这个值满足循环条件 i <=10，所以执行循环体并输出 i 值，然后自增 1 后继续循环。

思考：观察下面的程序段，分析程序运行结果。

```
int i;
```

```
while (i<=10)
{
    i=1;
    printf("%d\n",i);
    i=i+1;
}
```

② 循环的执行次数是由循环条件决定的。表示循环条件的表达式可以是任意类型的表达式，也可以是常量或变量，只要表达式的值非0(逻辑真)，表示循环条件成立，就执行循环体，否则就不执行循环体。如果表达式的值一开始就为0(逻辑假)，那循环体一次也不执行。

例如，已有定义“int i=10,a=1;”，则下面的3个循环条件均为真，都能执行相应的循环体。

```
while (i<=100)  {  …  }
while (i<=100&&i>=10)  {  …  }
while (a) {  …  }
```

而下面的循环语句中因为一开始循环条件就为假，所以循环体一次也没有执行。

```
while (i>=50)  {  …  }              //i的初值为10,所以表达式的值为0
```

③ 在循环控制表达式中，或在循环体内必须有改变循环变量值的语句，使循环趋向结束，否则会形成死循环。所谓死循环，是指无法靠自身的控制终止的循环。例如，

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

由于在循环体内没有改变循环变量i的表达式，所以i的值一直维持1，形成死循环。

【例5-1】　求$\sum_{i=1}^{100}i$。

解题思路：

(1) 根据题目可知s=1+2+3+…+100，这是一个累加求和问题，先后有100个数相加，加法运算要执行100次，所以用循环结构解决。

(2) 反复执行的操作是两个数相加，其中加数总是比前一个加数增加1，如果用变量i来表示加数，则i=i+1。第一个加数是1，最后一个加数是100，所以i值从1增长到100后就不再循环。

(3) 被加数是上一次加法运算的和，如果用变量s存放上一次相加的和，那么加法运算可以写成s+i，然后再将加法运算的结果存入s，因此就有表达式s=s+i。这里s表示累加和，通常称为累加器。需要注意，累加器s的初值一定要设为0，否则会因为s的初值不确定，导致程序运算结果出错。

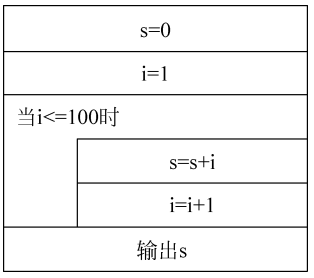


图5-3　求$\sum_{i=1}^{100}i$的N-S流程图

算法流程图如图5-3所示。

程序如下：

```
#include<stdio.h>
int main()
{
    int i=1;                          //定义循环变量 i,初值为 1
    int s=0;                          //定义累加器 s,初值为 0
    while (i<=100)                    //循环执行条件 i<=100
    {
        s=s+i;                        //累加器求和
        i=i+1;                        //加数自增 1,逐渐向循环跳出条件 i>100 靠拢
    }
    printf("s=%d,i=%d\n",s,i);        //输出累加和,以及跳出循环时的 i 值
    return 0;
}
```

运行结果为

```
s=5050,i=101
```

思考：

如何求 1～100 的奇数和？如何求 1～100 的偶数和？求 1～100 中的 5 的倍数和？

【例 5-2】 用公式$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots$求 π 的近似值，直到最后一项的近似值小于 10^{-4} 为止。

解题思路：根据题目可知，这个问题也是累加求和问题，不同之处在于：

(1) 公式中每个分数的分母为奇数 1,3,5,7,9…，每循环一次增长值为 2。如果用变量 n 表示分母，则 n=n+2，n 的初值为 1。

(2) 每个加数的符号是正负交替变化的，可以定义一个专门用于记录符号变化的变量 flag，每循环一次，flag=(-1)*flag，实现正负符号交替变化。

(3) 如果每一个加数(即 1/n)用变量 t 来表示，虽然不知道要加多少项，但可以用最后一项 t 的绝对值小于 10^{-4} 来作为循环结束的条件。所以循环的执行条件可以表示为$|t|\geqslant 10^{-4}$。

程序如下：

```
#include<stdio.h>
#include<math.h>                      //调用 fabs()函数需要包含 math.h 文件
int main()
{
    int flag=1;                       //定义变量 flag 初值为 1
    float n=1.0;                      //定义变量 n 表示分母,初值为 1.0
    float s=0.0;                      //定义累加器 s,初值为 0
    float t=1.0;                      //定义变量 t 表示每一个分数,初值为 1
    while (fabs(t)>=1E-4)             //当加数 t≥10^-4 时执行循环
    {
        s=s+t;
        n=n+2;
        flag=-flag;                   //符号正负交替
        t=flag*(1/n);                 //求出下一次累加的分数项
    }
    s=s*4.0;                          //求 π 值
    printf("pi=%f\n",s);
    return 0;
}
```

运行结果为：

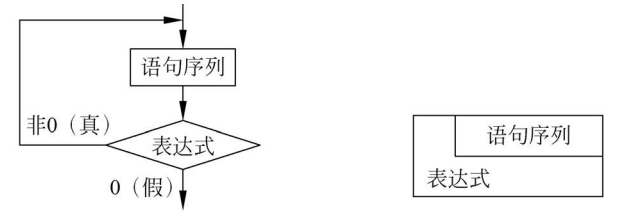
```
pi = 3.141397
```

5.3　do while 语句

do while 语句可以实现直到型循环，一般形式为

```
do
{
      语句序列
}while ( 循环控制表达式 );
```

执行过程：首先无条件地执行一次循环体，然后计算表达式的值，如果表达式的值非 0（逻辑真），表示循环条件成立，则返回重新执行循环体；直到表达式的值为 0（逻辑假），循环条件不成立时结束循环。所以，do while 循环的特点是无论如何都会执行一次循环体。do while 语句执行流程图如图 5-4 所示。



(a) do while循环传统流程图　　(b) do while循环N-S流程图

图 5-4　do while 语句执行流程图

说明：

（1）do 相当于一个标号，它标志着循环结构的开始，注意后面不能加分号“;”。

（2）语句序列无论是一条语句，还是多条语句，都用花括号{ }将它括起来，使得结构清晰，尤其对于初学者来说更容易理解。

（3）do while 语句整体上是一条语句，所以 while(表达式)后面的语句结束标志分号“;”不能缺少。

用 do while 语句改写【例 5-1】，算法流程图如图 5-5 所示。

循环部分的代码如下：

```
i = 1;
do                              //循环开始,后面不能跟分号";"
{
    s = s + i;
    i = i + 1;
}while(i <= 100);               //语句后面的分号";"不能少
```

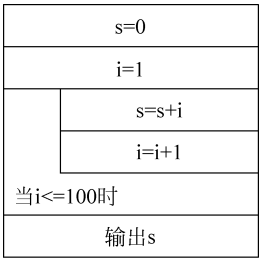


图 5-5　求 $\sum_{i=1}^{100} i$ 的 N-S 流程图

注意：while 语句和 do while 语句处理循环时，它们的区别主要体现在以下两个方面：

① 书写格式上不同。while 语句中 while(表达式)后面没有分号“;”，而 do while 语句在 while(表达式)后面一定要有分号“;”。

② 执行流程不同。while 语句是先判断条件然后再执行循环体,而 do while 语句是先执行循环体,然后再判断条件。如果循环条件一开始就不满足,那么 while 循环一次也不执行,而 do while 循环会执行一次。

思考:观察下面两段代码,分析运行结果。

```
程序(a)
  i = 10;
  while (i < 1)
  {
     printf(" % d",i);
     i = i + 1;
  }
```

```
程序(b)
  i = 10;
  do
  {
     printf(" % d",i);
     i = i + 1;
  }while (i < 1);
```

【例 5-3】 输入两个数 m 和 n,求 m 和 n 的最大公约数。

解题思路:辗转相除法求最大公约数是用 m 除以 n 求余数 r,当 r≠0 时,用除数做被除数,用余数做除数再求余数 r,如此反复,直到 r=0 时,除数即为所求的最大公约数。算法流程图如图 5-6 所示。

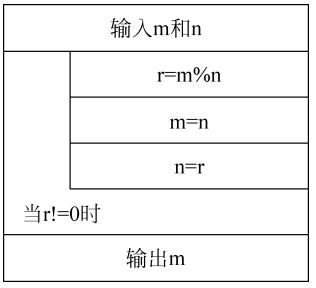


图 5-6 用 do while 求最大公约数的 N-S 流程图

程序如下:

```
#include < stdio.h >
int main()
{
    int m,n,r;
    scanf(" % d % d",&m,&n);
    do
    {
        r = m % n;
        m = n;                  //将除数 n 赋给被除数 m
        n = r;                  //将余数 r 赋给除数 n
    }while (r!= 0);             //直到 r = 0 跳出循环
    printf("最大公约数是 % d\n",m);
    return 0;
}
```

运行结果为

```
24 10 ↙
最大公约数是: 2
```

5.4 for 语句

C 语言提供了另一个使用非常广泛的语句 for 循环语句,通常适用于已知循环次数的情况,一般形式为

```
for(表达式 1; 表达式 2; 表达式 3)
{
    语句序列
}
```

执行过程:先执行表达式 1,然后判断表达式 2 的循环条件是否为真。如果为真则执行

循环体，然后再执行表达式 3，接下来继续判断表达式 2 的循环条件是否为真，如果为真则继续执行循环体和表达式 3，直到表达式 2 表示的循环条件为假时结束循环，执行 for 语句后面的语句。for 语句执行流程如图 5-7 所示。

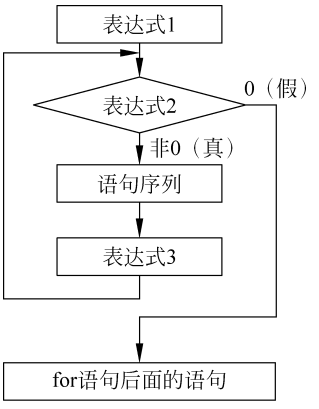


图 5-7　for 语句执行流程图

说明：

（1）表达式 1 一般是赋值表达式，它的作用是为循环变量赋初值。表达式 1 决定了循环的起始条件，只在循环开始前执行一次。

（2）表达式 2 是循环控制条件，用来判断是否继续执行循环，一般是关系表达式或逻辑表达式。每次执行循环体前先计算表达式 2 的值，只要它的值是非 0 值，表示循环条件成立，执行循环体，否则结束循环执行 for 循环后面的语句。

（3）表达式 3 一般为赋值表达式，它的作用是改变循环变量的值，使循环趋向结束，通过表达式 3 定义每执行一次循环后循环变量将如何变化。

（4）循环体中语句序列可以是一条语句，也可以是多条语句，通常用花括号括起来。

（5）通过比较 for 循环和 while 循环的流程图，发现它们的流程都是一样的，只是写法有所不同，所以 while 循环与 for 循环可以相互转换。相比之下，for 语句的结构更加紧凑、清晰。for 循环相当于下面的 while 循环。

```
表达式 1;                    //给循环变量赋初值
while (表达式 2)             //表达式 2 为循环判断条件
{
    语句序列;
    表达式 3;                //改变循环变量的值
}
```

用 for 语句改写【例 5-1】，循环部分代码如下：

```
for (i = 1; i <= 100; i++)
{
    s = s + i;
}
```

【例 5-4】 编写程序，输出斐波那契数列的前 20 项，要求每行输出 10 个数。

解题思路：斐波那契数列是这样一个数列：1、1、2、3、5、8、13、21，规律是从第 3 个数开始后面的每一个数都是前面两个数的和。

斐波那契数列可以用数学上的递推公式来表示：

$F_1 = 1$

$F_2 = 1$

$F_n = F_{n-1} + F_{n-2}$

算法流程如图 5-8 所示。

程序如下：

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

```
    int f1,f2,f,i;
    f1 = 1,f2 = 1;
    printf(" % 6d % 6d",f1,f2) ; //输出 f1,f2 并控制每项占 6 列
    for (i = 3;i < = 20;i++)
    {
        f = f1 + f2;        //从第 3 项开始,每项都是前两项的和
        printf(" % 6d",f);
        f1 = f2;
        f2 = f;
        if (i % 10 == 0) printf("\n") ; //控制每行输出 10 个数
    }
    return 0;
}
```

运行结果为

```
     1     1     2     3     5     8    13    21    34    55
    89   144   233   377   610   987  1597  2584  4181  6765
```

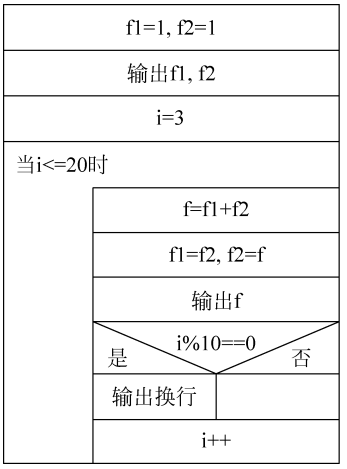


图 5-8　斐波那契数列 N-S 流程图

【例 5-5】 韩信有一队兵,他想知道有多少人,便让士兵报数:按从 1～5 报数,最末一个士兵报的数为 1;按从 1～6 报数,最末一个士兵报的数为 5;按从 1～7 报数,最末一个士兵报的数为 4;按 1～11 报数,最末一个士兵报的数为 10。编程求韩信至少有多少个士兵。

解题思路:本题采用穷举法,它的基本思想是假设各种可能的解,让计算机逐一进行测试,如果测试的结果满足条件,则假设的解就是所要求解的值。从 1 开始逐个取出一个自然数进行判断,如果有一个数除以 5、6、7、11 后的余数分别是 1、5、4、10,那么这个数就是要求的解,此时结束测试并输出这个数。如果没有满足条件,则继续测试下一个数,直到找到该数。

程序如下:

```
#include < stdio.h >
int main()
{
    int n;
    for (n = 1; ;n++)                                    //士兵人数从 1 开始
    {
        if (n % 5 == 1&&n % 6 == 5&&n % 7 == 4&&n % 11 == 10)   //4 个条件都满足时结束循环
        {
            printf("士兵至少有 % d 人\n",n);              //输出人数 n
            break;                                       //跳出循环
        }
    }
    return 0;
}
```

运行结果为

```
士兵至少有 2111 人
```

使用 for 语句时应注意:

① for 语句中表达式 1 可以省略,但其后的分号";"不能省略。此时需要在 for 语句之前给循环变量赋初值。

例如，

```
i = 1;
for (; i <= 100;i++)
    s = s + i;
```

② for 语句中表达式 2 也可以省略，但其后的分号“;”不能省略。如果表达式 2 省略，则默认循环条件恒为“真”，循环将一直执行下去成为死循环，除非循环体内有控制循环退出的语句。

例如，

```
for (i = 1; ;i++)                    //若省略表达式 2,则循环条件恒为真
{
    s = s + i;
    if (i > 100) break;              //如果 i > 100,用 break 跳出循环
}
```

③ for 语句中表达式 3 也可以省略，此时需要在循环体内增加改变循环变量值的表达式，否则会形成死循环。

例如，

```
for (i = 1; i <= 100; )
{
    s = s + i;
    i++;                             //改变循环变量 i 的值,使循环趋向结束
}
```

④ 如果同时省略表达式 1 和表达式 3，只有表达式 2，则需要在 for 语句前给循环变量赋初值，在循环体内改变循环变量的值，此时相当于只给了循环条件，与 while 循环完全相同。

例如，

```
i = 1;                                   i = 1;
for (;i <= 100; )                        while (i <= 100)
{                                        {
    sum = sum + i;          等价于               sum = sum + i;
    i++;                                     i++;
}                                        }
```

⑤ 如果表达式 1、表达式 2、表达式 3 都省略，那么此时没有循环条件，默认循环条件恒为“真”，既没有给循环变量赋初值，也没有改变循环变量的值，则会无休止地执行循环体，形成死循环。

例如，

```
for (; ;)  {   …   }
```

等价于：

```
while (1)  {   …   }
```

⑥ 表达式 1 和表达式 3 既可以是赋值表达式，也可以是逗号表达式。如果是逗号表达式，则可以在给循环变量赋初值的同时，给其他变量赋值；或是在改变循环变量的同时，给其他变量赋值。

例如，

```
int i,j;
for (i = 1,j = 10;i < = j;i++,j -- )  //i,j 初值都为 1,每循环一次 i 递增,j 递减,直到 i > j 时跳出循环
    printf("i = %d,j = %d\n",i,j);
```

⑦ 循环体也可以是空语句。

例如，

```
for (i = 1,s = 0;i < = 100;i++,s += i);  //循环体为空
```

5.5 循环嵌套

循环嵌套是指在一个循环体内又包含另一个完整的循环结构。嵌套在循环体内的循环称为**内层循环**，外面的循环称为**外层循环**。内嵌的循环中还可以嵌套循环，形成多层循环，while 循环、do while 循环和 for 循环可以互相嵌套。例如，下面几种都是合法的形式：

```
(1)
while ()
{   ⋮
    while ()
    {…}
}
```

```
(2)
do
{   ⋮
    do
    {
      …
    }while ();
}while ();
```

```
(3)
for ( ; ; )
{    ⋮
    for ( ; ; )
      {…}
      ⋮
}
```

```
(4)
for ( ; ; )
{   ⋮
    while ()
    {…}
     ⋮
}
```

【例 5-6】 编程输出如图 5-9 所示的九九乘法口诀表。

```
1*1= 1   1*2= 2   1*3= 3   1*4= 4   1*5= 5   1*6= 6   1*7= 7   1*8= 8   1*9= 9
2*1= 2   2*2= 4   2*3= 6   2*4= 8   2*5=10   2*6=12   2*7=14   2*8=16   2*9=18
3*1= 3   3*2= 6   3*3= 9   3*4=12   3*5=15   3*6=18   3*7=21   3*8=24   3*9=27
4*1= 4   4*2= 8   4*3=12   4*4=16   4*5=20   4*6=24   4*7=28   4*8=32   4*9=36
5*1= 5   5*2=10   5*3=15   5*4=20   5*5=25   5*6=30   5*7=35   5*8=40   5*9=45
6*1= 6   6*2=12   6*3=18   6*4=24   6*5=30   6*6=36   6*7=42   6*8=48   6*9=54
7*1= 7   7*2=14   7*3=21   7*4=28   7*5=35   7*6=42   7*7=49   7*8=56   7*9=63
8*1= 8   8*2=16   8*3=24   8*4=32   8*5=40   8*6=48   8*7=56   8*8=64   8*9=72
9*1= 9   9*2=18   9*3=27   9*4=36   9*5=45   9*6=54   9*7=63   9*8=72   9*9=81
```

图 5-9 九九乘法口诀表

解题思路：

(1) 观察乘法表中第 1 行的变化规律：被乘数 1 保持不变，乘数从 1 变化到 9，每次增加 1，因此构造如下循环即可实现乘法表第 1 行的输出。

```
for (j = 1; j < = 9; j++)
    printf("%1d * %1d = %2d ", 1 * j);
```

(2) 再观察乘法表中第 2 行的变化规律：被乘数 2 保持不变，乘数从 1 到 9 每次递增 1，与第 1 行的处理过程一样，只需将被乘数改为 2，然后将上面的循环再执行一次即可。

(3) 同理，第 3 行、第 4 行、……、第 9 行，处理过程都一样，只需将被乘数从 1 变化到 9，对上面循环执行 9 次。因此在上面循环的外面再加上一个循环构成双重循环，就可以输出九九乘法口诀表。算法流程图如图 5-10 所示。

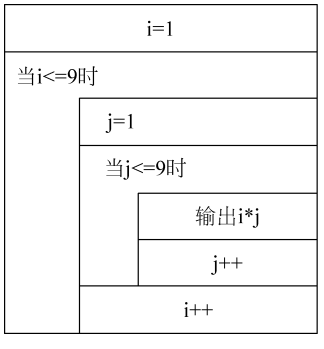


图 5-10 乘法口诀表 N-S 流程图

程序如下：

```
#include<stdio.h>
int main()
{
    int i, j;
    for (i=1; i<=9; i++)                                    //外层循环变量 i,控制被乘数变化
    {
        for (j=1; j<=9; j++)                                //内层循环变量 j,控制乘数变化
            printf("%1d*%1d=%2d  ", i, j, i*j) ;            //输出格式控制
        printf("\n");                                       //输出一行后换行
    }
    printf("\n");
    return 0;
}
```

双重循环的执行过程是：首先执行外层循环，当外层循环变量 i 取初值 1 时执行内层循环，内层循环变量 j 的值将从 1 变化到 9，在这个过程中 i 值始终不变，直到内层循环执行完毕。外层循环 i 的值才增长为 2，然后再执行一遍内层循环，j 值从 1 变化到 9。如此反复执行下去，直到外层循环变量 i 的值超过终值 9，整个双重循环才执行完毕。可以看出，在双重循环的执行过程中，外层循环执行一次，内层循环就要执行一遍。

思考：修改上面的程序，输出如图 5-11 所示的乘法口诀表。

```
1*1= 1  1*2= 2  1*3= 3  1*4= 4  1*5= 5  1*6= 6  1*7= 7  1*8= 8  1*9= 9
        2*2= 4  2*3= 6  2*4= 8  2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
                3*3= 9  3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
                        4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
                                5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
                                        6*6=36  6*7=42  6*8=48  6*9=54
                                                7*7=49  7*8=56  7*9=63
                                                        8*8=64  8*9=72
                                                                9*9=81
```

图 5-11　乘法口诀表示意图

【例 5-7】 求 100～200 的全部素数。

解题思路：

(1) 如果一个整数只能被 1 和它本身整除，那么这个数(除了整数 1)就是素数。如 2、3、5、7 是素数，而 4、6 不是素数。

(2) 判断某数 m 是素数的方法，用 2，3，4，……，m－1 逐个去除 m，看它能否被整除。如果 m 能被其中一个数整除，那么 m 就不是素数，如果 m 不能被其中任一个数整除，那么 m 就是素数。当 m 较大时，除法执行的次数过多降低了程序的执行效率，所以可以对算法进行优化，只需用 m 除以 2，3，……，$\sqrt{m}$就可以了。

(3) 因为偶数一定不是素数可以直接排除，所以外层循环使循环变量 m 的值为 101～199，每循环 1 次循环变量增长 2。

```
for (m=101;m<=199;m+=2)
```

程序如下：

```
#include<stdio.h>
#include<math.h>                          //调用 sqrt()函数需要包含 math.h
int main()
{
    int m,k,i,n=0,flag;                   //设置标志变量 flag,标记 m 是否被某数整除
```

```
    for (m = 101;m <= 199;m = m + 2)
    {
        flag = 0;                          //flag 初始值为 0
        k = sqrt(m);
        for (i = 2;i <= k;i++)             //循环变量 i 的值从 2 增长到√m
        {
            if (m % i == 0)                //如果 m 被 i 整除,则 m 不是素数
            {
                flag = 1;                  //标志变量 flag 设为 1 并停止测试,跳出内层循环
                break;
            }
        }
        if (flag == 0)                     //如果标志变量为 0,说明 m 是素数
        {
            printf(" %4d",m);              //输出素数
            n++;                           //计数器加 1
            if (n % 7 == 0)
                printf("\n");              //输出 7 个素数就换行
        }
    }                                      //外循环结束
    return 0;
}
```

运行结果为

```
101 103 107 109 113 127 131
137 139 149 151 157 163 167
173 179 181 191 193 197 199
```

使用循环的嵌套结构要注意的问题：

(1) 外层循环应完全包含内层循环,不能发生交叉。

例如,下面这种形式是不允许的：

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

(2) 循环嵌套中循环变量尽量不要同名,以免造成混乱。

例如,在下面的程序中,内、外层循环使用了相同的循环变量,因此改变了原有的循环次数。

```
for (i…)
{…
   for (i…)
     {…}
}
```

(3) 注意,使用缩进格式来明确嵌套循环的层次关系,以增加程序的可读性。

5.6 break 语句和 continue 语句

前面学习了 while 语句、do while 语句和 for 语句实现的循环结构,它们的共同特点是当循环条件满足时执行循环,只有在循环条件不满足的情况下才结束循环。然而有时需要

根据一定的条件提前跳出循环，或者在满足某种条件时提前结束本次循环，这就要用到break语句和continue语句。

5.6.1　break语句

break语句是限定转向语句，它能够使流程跳出所在的结构，转向执行该结构后面的语句。前面学习过可用break语句使流程跳出switch语句；break语句还可以使流程跳出它所在的循环结构，提前结束本层循环转去执行该循环结构后面的语句。一般形式为

```
break;
```

执行过程：程序执行时先计算表达式1的值，如果表达式1的值非0(逻辑真)，则执行循环体。在循环体内，如果表达式b的值非0(逻辑真)，满足了提前跳出条件，则执行break跳出本层循环，程序的流程转去执行循环结构后面的语句。如果表达式b的值为0(逻辑假)，则继续执行循环体。执行流程如图5-12所示。从流程图可以看出，循环跳出的条件有两个：要么表达式1不成立时正常结束循环，要么表达式b成立时提前结束循环。

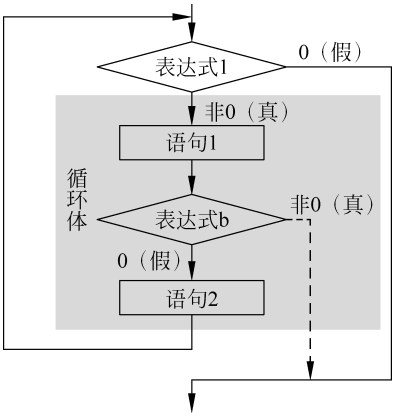


图5-12　break语句功能流程图

说明：

(1) break语句只能用于switch语句和循环语句中。

(2) break通常与if语句一起使用。当满足if的判断条件时提前终止循环，跳出相应的循环结构。

(3) 当break处于嵌套结构中时，它将使break语句所处的该层及内层结构循环中止，即break语句只能跳出它所在的循环，而对外层的结构没有任何影响。

【例5-8】 读程序，分析程序运行结果。

程序如下：

```
#include <stdio.h>
int main()
{
    int  i, n;
    for (i=1;i<=5;i++)                    //循环应执行5次
    {
        printf("Please enter n: ");
        scanf("%d", &n);
        if (n<0)  break;                  //如果输入的n值小于0,则提前跳出循环
        printf("n = %d\n", n);
    }
    printf("Program is over!\n");
    return 0;
}
```

运行结果为

```
Please enter n: 15 ↙
n = 15
Please enter n: 20 ↙
```

```
n = 20
Please enter n: -3↙
Program is over!
```

上面程序中循环变量 i 的值从 1～5 每循环 1 次递增 1,正常情况下应该执行 5 次循环。如果读入的 n 值小于 0,则执行 break 语句提前跳出循环,转去执行循环后面的语句。只有当读入的 n 值都大于或等于 0 时,才能保证循环执行 5 次后正常结束。

5.6.2 continue 语句

continue 语句也是限定转向语句,它的功能是提前结束本次循环,即跳过循环体中 continue 语句后面尚未执行的语句,重新开始下一次循环。一般形式为

```
continue;
```

说明:

(1) continue 语句只能用在 while、do while 和 for 这 3 种循环语句中。通常和 if 配合使用,当条件满足时提前结束本次循环并开始下一次循环,但是并没有终止整个循环的执行。

(2) continue 语句在 3 种循环语句的执行流程略有不同。

- 在 while 和 do-while 语句中,当循环条件满足时,则执行循环体中的语句 1,然后判断表达式 b 是否满足,如果满足则越过 continue 后面的语句 2,流程转去判断表达式 1,从而开始下一次新的循环。如果表达式 b 的条件不满足,则继续执行语句 2。如图 5-13 所示。
- 在 for 语句中,当表达式 b 的条件满足时,程序流程会跳过循环体中还未执行的语句 2,转去执行表达式 3,改变循环变量的值,然后再去执行表达式 2 进行循环条件的判断,根据判断结果决定 for 循环是否继续执行。如图 5-14 所示。

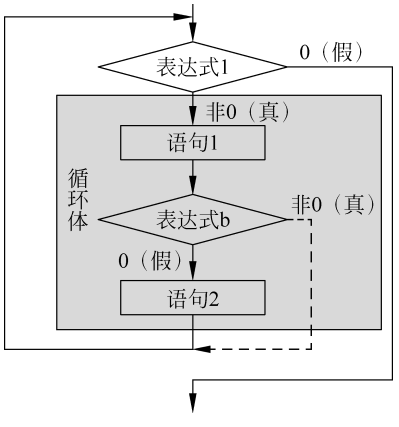


图 5-13 continue 语句功能流程图在 while、do while 语句中

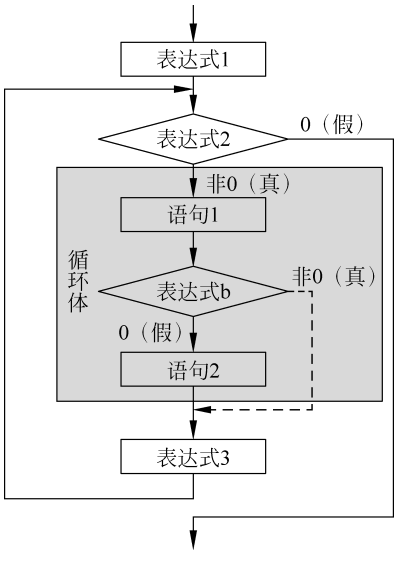


图 5-14 continue 语句功能流程图在 for 语句中

【例 5-9】 求输入的 10 个整数中正数的个数及其和。

程序如下：

```
#include <stdio.h>
int main()
{
    int i,x,s = 0,n = 0;
    for (i = 1;i <= 10;i++)
    {
        printf("请输入第 %d 个数: ",i);
        scanf("%d",&x);
        if (x < 0) {n++;continue;}
        s = s + x;
    }
    printf("10 个数中正数有 %d 个,其和: %d\n",10 - n,s);
    return 0;
}
```

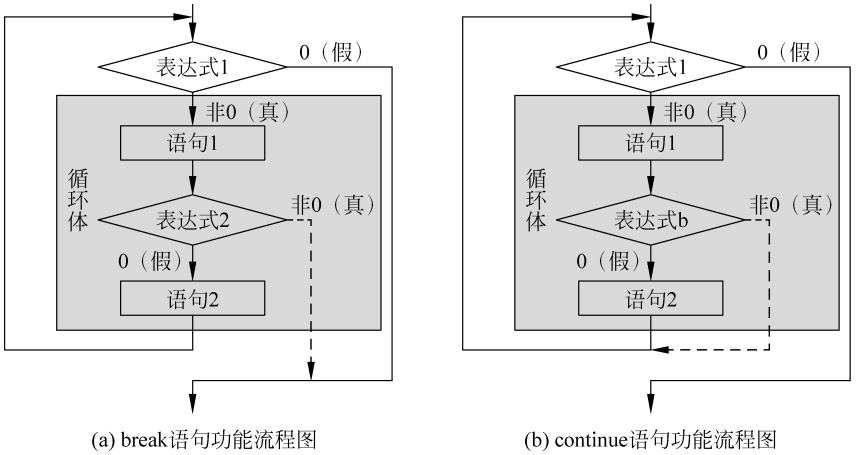
运行结果为

```
请输入第 1 个数: 25 ↙
请输入第 2 个数: 95 ↙
请输入第 3 个数: -87 ↙
请输入第 4 个数: 95 ↙
请输入第 5 个数: 25 ↙
请输入第 6 个数: -47 ↙
请输入第 7 个数: -5 ↙
请输入第 8 个数: 24 ↙
请输入第 9 个数: 3 ↙
请输入第 10 个数: 12 ↙
10 个数中正数有 7 个,其和: 279
```

从上面程序的执行过程可以看出，执行 continue 语句后不会改变程序原来设定的循环次数，只是会跳过 continue 后面的语句，继续下一次循环。

5.6.3 break 语句和 continue 语句的比较

break 语句和 continue 语句都是流程转移语句，都可以应用到循环语句中，而且都要与 if 联合使用才有意义，使用时应注意它们的区别，如图 5-15 所示。



(a) break语句功能流程图　　(b) continue语句功能流程图

图 5-15　break 语句和 continue 语句功能比较

- break 语句可以用在 switch 语句和循环语句中,其作用是跳出 switch 语句或跳出本层循环,转去执行 switch 语句或循环后面的语句。break 一旦执行,就会改变原来程序的循环次数。
- continue 语句只能用于循环语句中,其作用是结束本层本次循环,不再执行循环体中 continue 语句之后的语句,转入执行下一次循环。即使执行了 continue 语句,也不会改变原定的循环次数。

5.7 循环结构程序设计举例

【例 5-10】 输入一行字符,分别统计出其中英文字母、空格、数字和其他字符的个数。

解题思路:

(1) 将回车符作为输入结束标志。

(2) 每输入一个字符,就判断字符的类型:英文(大写字母'A'～'Z',小写字母'a'～'z')、空格(' ')、数字('0'～'9'),否则是其他字符。为每一类字符定义一个计数器,确定类型后就给相应的计数器加 1。

程序如下:

```
#include <stdio.h>
int main()
{
    char c;
    int letter = 0,space = 0,digit = 0,other = 0;          //定义各类型字符的计数器,并初始化为 0
    while ((c = getchar())!= '\n')                         //当输入的字符不是回车则执行循环
    {
        if (c >= 'a'&&c <= 'z'||c >= 'A'&&c <= 'Z')        //如果是字母,计数器 letter + 1
            letter++;
        else if (c == ' ')                                 //如果是空格,计数器 space + 1
            space++;
        else if (c >= '0'&&c <= '9')                       //如果是数字,计数器 digit + 1
            digit++;
        else                                               //如果是其他字符,计数器 other + 1
            other++;
    }
    printf("letter = %d,space = %d,digit = %d,other = %d\n",letter,space,digit,other);
    return 0;
}
```

运行结果为

```
7a8b90 & * d2 #!abc ↙
letter = 6,space = 2,digit = 5,other = 4
```

【例 5-11】 输入一个正整数 m,输出它是几位数,并将其逆序输出。

程序如下:

```
#include <stdio.h>
int main()
{
    int m,n = 0;
```

```
    scanf("%d",&m);                          //输入一个整数 m
    do
    {
        printf("%d",m%10);                   //输出个位上的数字
        m=m/10;                              //将 m 更新为去掉个位数的新 m
        n++;                                 //位数计数器+1
    }while (m!=0);                           //直到新 m=0 时结束循环
    printf("这个数是%d位数\n",n);
    return 0;
}
```

运行结果为

```
123456↙
6 5 4 3 2 1 这个数是 6 位数
```

【例 5-12】 输出如图 5-16 所示的菱形。

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

图 5-16　菱形图形

解题思路：

(1) 图形的输出一般用双重循环来实现。外层循环控制行数，内层循环控制每行中的列数及每列的内容，然后在内外层循环之间加上 printf("\n") 实现换行。

(2) 本题中的菱形可以分成两部分输出(上三角和下三角)。

- 上三角中随着行数 i 的增加每行多 2 个“*”，而且“*”的个数 k 和行数 i 满足关系 k=2*i-1。同时注意控制每行第一个“*”的输出位置，可以通过输出逐行递减的空格实现。
- 对于下三角，设置行数 i 由 3 递减到 1，随着行数 i 的递减每行少 2 个“*”，而且“*”的个数 k 和行数 i 满足关系 k=2*i-1。同时注意控制每行第一个“*”的输出位置，可以通过输出逐行递增的空格实现。

程序如下：

```
#include <stdio.h>
int main()
{
    int i,j,k;
    for (i=1;i<=4;i++)                       //输出上三角
    {
        for (j=1;j<=(11-i);j++)              //输出逐行递减的空格
            printf(" ");
        for (k=1;k<=2*i-1;k++)               //输出 2*i-1 个"*"
            printf("*");
        printf("\n");                        //换行
    }
    for (i=3;i>=1;i--)                       //输出下三角,控制输出 3 行
    {
        for (j=1;j<=(11-i);j++)              //输出逐行递增的空格
            printf(" ");
    for (k=1;k<=2*i-1;k++)                   //输出 2*i-1 个"*"
            printf("*");
         printf("\n");                       //换行
    }
```

```
    return 0;
}
```

上面程序中将菱形分为上三角和下三角两部分输出,而两部分输出的方法类似,只是外层循环变量的值不同。可用一个循环结构输出菱形,输出部分的程序段可优化为:

```
for (i = -3;i <= 3;i++)
{
    for (j = 1;j <= 2 + abs(i);j++)
        printf(" ");
    for (k = 1;k <= 6 - (abs(i) * 2 - 1);k++)
        printf("*");
    printf("\n");
 }
```

【例 5-13】 设有红、黄、绿 3 种颜色的球,其中红球 3 个、黄球 3 个、绿球 6 个,现将这 12 个球混放在一个盒子里,从中任意摸出 8 个球,编程计算摸出球的各种颜色搭配。

解题思路:利用穷举法列出所有可能的组合,然后根据条件筛选出符合条件的组合。其中,红球可能值为:0,1,2,3,黄球可能值为:0,1,2,3,绿球可能值为:2,3,4,5,6。

程序如下:

```
#include <stdio.h>
int main()
{
    int x,y,z,n = 0;                                    //定义 x、y、z 分别表示红、黄、绿球个数
    printf("red\tyellow\tgreen\n");
    for (x = 0;x <= 3;x++)                              //红球可能值为 0～3
        for (y = 0;y <= 3;y++)                          //黄球可能值为 0～3
            for (z = 2;z <= 6;z++)                      //绿球可能值为 2～6
                if (x + y + z == 8)                     //如果红黄绿三色球之和为 8
                {
                    printf("%d\t%d\t%d\n",x,y,z);
                    n++;                                //统计可能的组合方案个数
                }
    printf("共有%d种\n",n);
    return 0;
}
```

运行结果为

```
red     yellow  green
0       2       6
0       3       5
1       1       6
1       2       5
1       3       4
2       0       6
2       1       5
2       2       4
2       3       3
3       0       5
3       1       4
3       2       3
3       3       2
共有 13 种
```

【例 5-14】 输出 1～1000 的同构数。同构数是指一个数 n 恰好出现它的平方数的右端，如 25 和 625 是同构数。

解题思路：依次取出 1～1000 的整数并求其平方，然后从个位数开始，取出平方数和该数对应的每一位上的数字进行比较，如果每一位上的数都相同，则是同构数，否则不是同构数。

程序如下：

```
#include<stdio.h>
int main()
{
    int m,n,j;
    long k;
    for (m=1;m<1000;m++)
    {
        k=(long)m*m;
        n=m;
        while (n>0&&k%10==n%10)                    //将 k 和 n 的相应位置上的数进行比较
        {
            k=k/10;
            n=n/10;
        }
        if (n==0) printf("%4d%10d\n",m,m*m);
    }
    return 0;
}
```

运行结果为

```
   1         1
   5        25
   6        36
  25       625
  76      5776
 376    141376
 625    390625
```

【例 5-15】 求解定积分$\int_0^2 (x^2+1)\mathrm{d}x$的近似值。

解题思路：可以用矩形法求$\int_a^b f(x)\,\mathrm{d}x$的近似值，其算法为：

(1) 将积分区间[a,b]分成长度相等的 n 个小区间，区间端点分别为 $x_0, x_1, x_2, \cdots, x_n$，其中 $x_0=a$，$x_n=b$。其中，n 值取得越大，其近似程度越好。

(2) 每一个小曲边梯形的面积用对应的矩形面积来代替，如图 5-17 所示。

(3) n 个小矩形的面积之和，就是定积分的近似值。

用矩形法求定积分的公式为

$$\int_a^b f(x)\,\mathrm{d}x=\sum_{i=0}^{n-1} f(x_i)(b-a)/n$$

每个小矩形的宽度为

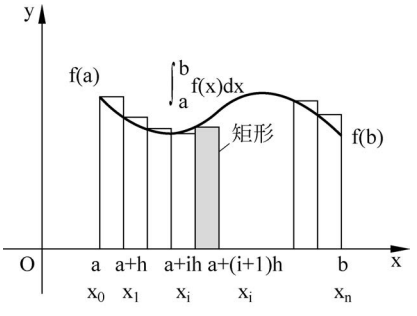


图 5-17　矩形法求定积分示意图

$$h=\frac{b-a}{n}$$

端点为

$$x_i=a+ih$$

对 $\int_0^2(x^2+1)dx$ 而言，有

$$a=0,b=2,h=(2-0)/n,f(x)=x^2+1$$

程序如下：

```
#include<stdio.h>
int main()
{
    int n,i;
    double a,b,x,h,f,s=0;
    printf("请输入划分的区间数 n: ");
    scanf("%d",&n);                          //输入划分的区间个数 n,n 值越大近似程度越好
    printf("请输入积分的下限和上限: ");
    scanf("%lf%lf",&a,&b);                   //输入积分的下、上限
    h=(b-a)/n;                               //求矩形的宽度
    for (i=0;i<n;i++)
    {
        x=a+i*h;
        f=x*x+1;                             //矩形高度
        s=s+f*h;
    }
    printf("积分值为: %.2f\n",s);
    return 0;
}
```

运行结果为

```
请输入划分的区间数 n: 500↙
请输入积分的下限和上限: 0 2↙
积分值为: 4.66
```

小结

在程序中不断被重复执行的结构称为循环结构。C 语言提供了 4 种实现循环的语句：while 语句、do while 语句、for 语句，以及非限定性跳转语句 goto 语句。其中 goto 语句必须与 if 结合使用才能实现循环结构，但结构化的程序设计一般不提倡使用 goto 语句，因此本书不作介绍。

(1) while 语句、do while 语句和 for 语句都可以用来处理同一问题，一般情况下它们可以互相代替。

(2) while 语句和 do while 语句用在循环次数不确定，但循环条件明确的循环中；for 语句使用最为灵活且结构紧凑，不仅可以用于循环次数已经确定的循环结构，还可以用于循环次数不确定只给出循环结束条件的循环结构中，它完全可以代替 while 语句。

(3) while 语句、do while 语句和 for 语句可以相互嵌套组成多重循环，嵌套的循环之间

可以并列但不能交叉。

(4) while 循环、do while 循环和 for 循环，可以用 break 语句提前跳出循环，用 continue 语句结束本次循环。而对用 goto 语句和 if 语句构成的循环，不能用 break 语句和 continue 语句进行控制。

本章常见错误分析

常见错误实例	常见错误解析

`while (i<=10)`
`{`
`  s=s+i;`
`  i++;`
`}`
	在循环开始前，没有给循环变量 i 和累加器 s 赋初值。可改为：
	`int i=0,s=0;`
	`while(i<=10)`
	`{  s=s+i; i++;  }`

`int i=0,s=0;`
`while (i<=10)`
`  s+=i;`
`  i++;`
	while 语句的循环体两端没有加{ }使之成为复合语句。应改为：
	`int i=0,s=0;`
	`while(i<=10)`
	`{  s+=i; i++;  }`

`for (i=1;i<=10;i++);`
`{ … }`
或
`while (i<=10);`
`{  …`
`  i++;`
`}`
	for 语句、while 语句后面加了分号，使循环体成为空循环。可改为：
	`for(i=1;i<=10;i++)`
	`  { … }`
	或
	`while(i<=10)`
	`{  …`
	`  i++;`
	`}`

`int i=0,s=0;`
`while (i<=10)`
`{`
`  s+=i;`
`}`
	循环体中少了改变循环变量的表达式，使循环无法趋向结束，形成死循环。应改为：
	`int i=0,s=0;`
	`while(i<=100)`
	`{  s+=i;  i++; }`

`do`
`{`
`  …`
`}while (i<=10)`
	while 后面缺少了分号，会产生编译错误，应改为：
	`do`
	`{`
	`  …`
	`}while(i<=10);`

`for (i=1,i<10,i++)  { … }`	for 循环中表达式 1，表达式 2 和表达式 3 之间应该用分号分隔，不是逗号，应改为： `for(i=1;i<=10;i++)  { … }`

`while (a=3)`
`{ … }`
	while 后面的表达式是赋值表达式，使循环成为一个条件恒为真的死循环，应改为：
	`while(a==3)`
	`{ … }`

习题 5

1. 基础篇

(1) C 语言中常用的实现循环结构的控制语句有(　　)语句、(　　)语句和(　　)语句。

(2) do while 语句和 while 语句的区别主要是(　　)。

(3) break 语句和 continue 语句的区别是(　　)。

(4) 阅读程序,分析程序运行结果为(　　)。

```
int i, k;
for (i = 0,k = -1;k = 1;i++,k++)
    printf("! ! ! ");
```

(5) 阅读程序,分析下述循环的循环次数为(　　)。

```
int k = 2;
while (k == 0)
    printf("%d",k);
    k--;
printf("\n");
```

(6) 阅读下列程序,分析程序的运行结果为(　　)。

```
#include<stdio.h>
int main()
{
    int y = 10;
    while (y--);
    printf("y = %d\n", y);
    return 0;
)
```

2. 进阶篇

(1) 阅读下列程序,分析程序的运行结果为(　　)。

```
#include<stdio.h>
int main()
{
    int x = 23;
    do
    {
        printf("%d", x--);
    }while (!x);
    return 0;
}
```

(2) 阅读下列程序,分析程序的运行结果为(　　)。

```
#include<stdio.h>
int main()
{
    int x = 3,y;
    do
    {
        y = x - 1;
```

```
        if (!y)  { printf("*"); break;  }
        printf("#");
    } while(x>=1&&x<=2);
    return 0;
}
```

(3) 阅读程序,若下述程序运行时输入的数据是'A',则输出结果是(　　)。

```
#include<stdio.h>
int main()
{
    int c;
    while ((c=getchar())!= '\n')
    {
        switch (c-'2')
        {  case 0 :
           case 1 : putchar(c+4);
           case 2 : putchar(c+4); break;
           case 3 : putchar(c+3);
           default: putchar(c+2); break;
      }
  }
  printf("\n");
  return 0;
}
```

(4) 阅读程序,下述程序的输出结果是(　　)。

```
#include<stdio.h>
int main()
{
    int i,j,x=0;
    for (i=0;i<2;i++)
    {
        x++;
        for (j=0;j<=3;j++)
        {
            if (j%2) continue;
            x++;
        }
        x++;
    }
    printf("x=%d\n", x);
    return 0;
}
```

(5) 计算一个数列的前 n 项之和。该数列的前两项是由键盘输入的正整数,以后各项按下列规律产生:先计算前两项之和,若和小于 200,则该和作为下一项,否则用该和除以前两项中较小的一项,将余数作为下一项。为实现上述功能,请在[填空 1]、[填空 2]、[填空 3]处填入正确内容。

```
#include<stdio.h>
int main()
{
    int n, k1, k2, k3, m, ms, j;
    scanf("%d%d%d",&n,&k1,&k2);
    m=k1+k2;
```

```
    [填空 1]
    for (j = 3;j < = n;j++)
    {
        if ( [填空 2] )
            if (k1 < k2)
                k3 = m % k1;
            else
                k3 = m % k2;
        else
            [填空 3];
        ms = ms + k3;
        k1 = k2;
        k2 = k3;
        m = k1 + k2;
    }
    printf(" % d\n", ms);
    return 0;
}
```

(6) 下面程序的功能是：输入一组数，输出其中最大值和最小值，输入 0 时结束。为实现上述功能，请在[填空 1]、[填空 2]、[填空 3]处填入正确内容。

```
int main()
{
    float x,max,min;
    scanf(" % f",&x);
    max = x;min = x;
    while (  [填空 1]  )
    {
        if (x > max) max = x;
        if ( [填空 2] ) min = x;
        [填空 3]
    }
    printf("max = % f\nmin = % f\n", max, min);
    return 0;
}
```

3. 提高篇

(1) 下面程序的功能是：输出 5!+6!+7!+8!+9!+10! 的结果。程序中有 3 处错误，请找出错误并改正。注意：不得增行或删行，也不得更改程序的结构。

```
#include < stdio.h >
int main()
{
    long s,t;
    int i,j;
    for (i = 5;i < = 10;i++)
    {
        t = i;
        for (j = 1;j < = i;j++)
            t = t * j;
        s = t;
    }
    printf(" % ld\n",s);
```

```
    return 0;
}
```

(2) 以下程序的功能是：求出 1＊1＋2＊2＋…＋n＊n＜＝1000 中满足条件的最大的 n。程序中有 3 处错误，请找出错误并改正。注意：不得增行或删行，也不得更改程序的结构。

```
#include <stdio.h>
int main()
{
  int n,s;
  s == n = 0;
  while (s > 1000)
  {
    ++n;
    s += n * n;
  }
  printf("n = %d\n",&n - 1);
}
```

(3) 编程计算 a＋aa＋aaa＋…＋aa…a(n 个 a)的值，n 和 a 从键盘输入。

(4) 输出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数，其各位数字立方和等于该数本身。例如，153 是一个水仙花数，$153=1^3+5^3+3^3$。

(5) 输出 1～1000 中能同时被 3 和 5 整除的前 10 个数。

(6) 求 $1+\frac{1}{1\times2}+\frac{1}{2\times3}+\frac{1}{3\times4}+\cdots+\frac{1}{n\times(n+1)}$，直到最后一项的值小于 10^{-2}，如果累加到第 20 项(即 n＝19)时，最后一项的值还不小于 10^{-2}，则不再计算，要求输出 n 的值、最后一项的值、多项式之和。

(7) 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子 1、2、3，而 6＝1＋2＋3，因此 6 是“完数”。编程序找出 1000 之内的所有完数。

(8) 设 N 是一个 4 位数，它的 9 倍恰好是其反序数，求 N。反序数就是将整数的数字倒过来形成的整数。例如：1234 的反序数就是 4321。

(9) 猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半多一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少个桃子。

(10) 编程设计一个简单的猜数游戏。程序运行时自动产生一个随机整数，用户输入猜的数字，如果猜对了，则显示“Right”，如果猜错了，则显示“Wrong!”，并告诉用户所猜的数是大还是小。

(11) 百钱百鸡问题。中国古代数学家张丘建在他的《算经》中提出了著名的“百钱买百鸡问题”：鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一，百钱买百鸡，问翁、母、雏各几何？

(12) 编写程序：输出如下图形。

```
          1
        2 3 4
      5 6 7 8 9
    0 1 2 3 4 5 6
  7 8 9 0 1 2 3 4 5
6 7 8 9 0 1 2 3 4 5 6
7 8 9 0 1 2 3 4 5 6 7 8 9
```