第5章 循环结构程序设计

在许多实际问题中需要用到循环控制。例如第4章的简单的计算器程序，若用户需要做多次算术运算呢？例4.4的猜数游戏，若允许用户连续猜多次直到猜对为止呢？再如，求若干个数之和、迭代求根等。要实现这样的功能，都要用到循环结构，即需要重复执行某些操作。循环结构是结构化程序设计的基本结构之一，它和顺序结构、选择结构共同作为各种复杂程序的基本构造单元。C语言提供了while语句、do…while语句和for语句实现循环结构程序设计。

【引例　猜数游戏】

利用随机函数产生一个随机数，让用户反复猜测，直到猜对为止。

分析：例4.4的猜数游戏只猜了一次程序就结束了，可以将例4.4中的代码反复写下去，即输入一个数，进行判断，继续输入，继续判断……但也不知道该重复多少次，因为不知道多少次能猜对。

在解决实际问题时，许多问题可归结为重复执行的操作，重复执行就是循环。本问题用户猜多少次能猜对事先是未知的，即循环次数未知，这是一个条件控制的循环，控制循环的条件是“直到猜对为止”，可以用直到型循环do…while语句来实现。

程序：

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(void)
{
    int guess,magic;
    srand(time(NULL));
    magic = rand() % 100 + 1;
    do
    {
        printf("Please guess a magic number:\n");
        scanf("%d", &guess);                    //输入猜的数
        if(guess>magic)
            printf("Wrong!Too high!\n");
        else if(guess<magic)
            printf("Wrong!Too low!\n");
    }while (guess!= magic);                     //直到人猜对为止
    printf("Right!\n");
    return 0;
}
```

程序运行结果：

```
Please guess a magic number:
85 ↙
Wrong! Too high!
Please guess a magic number:
45 ↙
Wrong! Too low!
Please guess a magic number:
66 ↙
Right!
```

思考：若只给用户最多 5 次猜数机会，该如何修改程序？

5.1 while 语句

while 语句被用来实现当型循环，一般形式如下：

```
while(表达式)
   语句
```

其中，语句又称“循环体”，表达式又称为“循环条件”。

while 语句的执行过程是：

(1) 计算表达式的值。

(2) 若表达式的值为真，则执行循环体。然后重复(1)。

(3) 若表达式的值为假，则 while 语句结束，执行循环结构的后续语句。

while 语句的执行过程如图 5.1 所示。

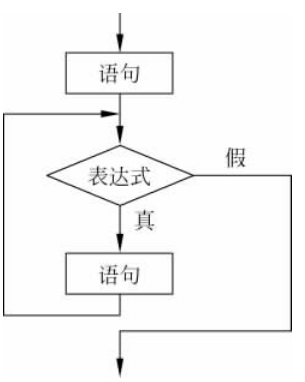

图 5.1 while 循环

说明：

(1) while 循环的特点为先判断后执行，即先判断表达式的值后执行循环体。若循环条件一开始就不成立，则循环体一次都不执行。

(2) 表达式可以是 C 语言的任意表达式，但都将表达式的值按逻辑值来处理。while 语句中的表达式外面的圆括号是语法规定必须有的。

(3) 循环体可以是单条语句，也可以是复合语句。

(4) 在循环体中应有使循环趋于结束的语句，以避免“无限循环”的发生。

【例 5.1】 求 $1+2+\cdots+100=\sum_{n=1}^{100}n$。

分析：学习循环结构要把算法设计作为重点，本题是累加问题，定义变量 sum 存放累加和，定义变量 n 存放累加项，累加问题可以用递推式来描述：

$$\text{sum}_i=\text{sum}_{i-1}+n_i\ (\text{sum}_0=0,n_1=1)$$

即第 i 次的累加和 sum 等于第 $i-1$ 次的累加和加上第 i 次的累加项 n_i。在循环体中，可用赋值语句“sum=sum+n;”来实现。其中，sum 的初值为 0、n 的初值为 1。

累加项 n 的递推式为：

$$n_i=n_{i-1}+1$$

在循环体中可用 $n=n+1$ 或 $n++$ 来实现。

累加求和的算法流程图如图 5.2 所示，图 5.2 (a)和(b)分别是传统流程图和 N-S 流程图。

根据传统流程图描述的算法的具体执行步骤如下：

(1) 求和变量 sum 赋初值为 0，累加变量 n 取初值为 1，开始进入循环结构。

(2) 判断条件“$n\leqslant 100$”是否满足，此时，若表达式 $n\leqslant 100$ 为真，则进入循环体。

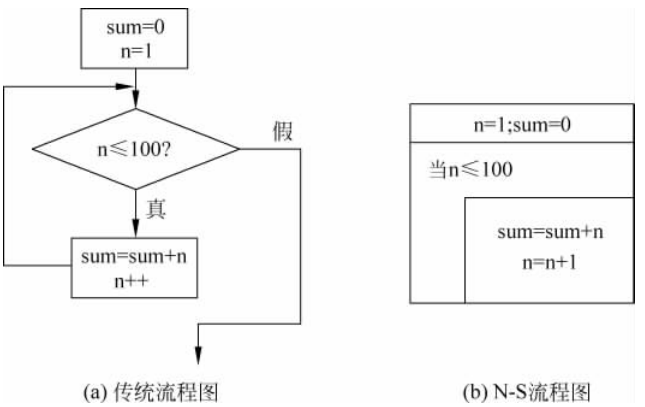

图 5.2　例 5.1 累加求和的算法流程图

(3) 执行 sum＝sum＋n，此时，sum 的值变为 1，然后执行 $n++$，n 变为 2，这是为下一次加 2 做准备，程序流程返回菱形框。

(4) 继续判断“$n\leqslant 100$”条件是否满足，此时，$n=2$，因此“$n\leqslant 100$”的值仍然为真，继续执行矩形框中的循环体。

(5) 执行 sum＝sum＋n，此时，sum 成为 1＋2 的结果，执行 $n++$，n 变为 3，流程再次返回菱形框。

(6) 再次检查 $n\leqslant 100$ 条件是否满足……如此反复执行矩形框中的操作，则 n 的值越来越趋向于 100，sum 逐渐成为 1＋2＋3＋…的结果；当 n 为 100 时，$n\leqslant 100$ 为真，sum 为 1＋2＋3…＋100 的结果；当 n 值累加到 101 时，$n\leqslant 100$ 为假，while 循环结束。

程序：

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

程序运行结果：

```
sum = 5050
```

说明：

(1) 在进入循环体前必须为循环控制变量 n 赋初值为 1，为累加求和变量 sum 赋初值为 0。若不赋初值，其值是随机的。

(2) $n<=100$ 是循环控制条件，若该条件为真，则执行循环体，实现累加，否则结束 while 循环。

(3) $n++$ 既更新了累加项的值，又使循环趋于结束。

若在 while 循环条件后面添加了分号，则会改变循环的意图，例如：

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

这个程序将变成无限循环，此时，循环体为空语句，“sum=sum+n;n++;”已经不是循环的一部分了。

(4) 请调试程序，跟踪 sum 和 n 值的变化。

思考：

(1) 程序中“sum+=n;”和“n++;”两个语句能否交换顺序？

(2) 修改程序，求 1+1/2+1/3+…+1/50 的值。

(3) 修改程序，求 1 到 m 的累加和，m 的值从键盘输入。

本程序用的方法称为迭代法，迭代法是一种不断用变量的旧值递推新值的过程。

利用迭代算法解决问题的基本思路如下。

(1) 确定迭代变量：在可以用迭代算法解决的问题中至少存在一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量。

(2) 建立迭代关系式：所谓迭代关系式是指如何从变量的前一个值推出其下一个值的公式(或关系)。在累加求和的例子中，迭代关系式是 sum=sum+n 和 $n=n+1$。

(3) 对迭代过程进行控制：迭代过程的控制通常可分为两种情况，一种是所需的迭代次数是确定的值；另一种是所需的迭代次数无法确定。对于前一种情况，可以构建一个固定次数的循环来实现对迭代过程的控制；对于后一种情况，需要进一步分析出结束迭代过程的条件。

【例 5.2】 求两个正整数 m、n 的最大公约数。

方法 1：

分析：求最大公约数可以采用“辗转相除法”，解此题的算法如下。

(1) $m\%n$ 得余数 r。

(2) 若 r 等于 0，则 n 即为两数 m、n 的最大公约数。

(3) 如果 $r\neq0$，则 $m=n$、$n=r$，转去执行(1)。如果 r 仍不等于 0，重复上述过程，直到 $r=0$ 为止。

程序：

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

```
{
    int m,n,r;
    printf("Please input m,n:\n");
    scanf("%d,%d",&m,&n);
    while((r=m%n)!=0)                    //等价于 while(r=m%n)
    {
        m=n;
        n=r;
    }
    printf("The greatest common divisor is %d\n",n);
    return 0;
}
```

程序运行结果：

```
Please input m,n:
165,90↙
The greatest common divisor is 15
```

思考：无论 m 比 n 大还是小都能求出最大公约数吗？请继续求出两个数的最小公倍数。

方法 2：

可以根据最大公约数的定义来求最大公约数。

(1) 找出 m、n 中的较小者赋给变量 t，最大公约数肯定在 t、$t-1$、$t-2$、…、1 之间。

(2) 测试 $m\%t$ 和 $n\%t$ 是否同时为 0，若余数同时为 0，则 t 即为最大公约数；否则 $t=t-1$，重复第(2)步。

读者可根据以上算法写出程序。

5.2 do…while 语句

do…while 语句被用来实现直到型循环，一般形式如下：

```
do
    语句
while(表达式);
```

do…while 语句的执行过程是：

(1) 执行循环体语句。

(2) 计算表达式。

(3) 若表达式的值为真(非 0)，则再次执行(1)。若表达式的值为假(0)，则 do …while 语句结束，执行循环结构的后续语句。do…while 语句的执行过程如图 5.3 所示。

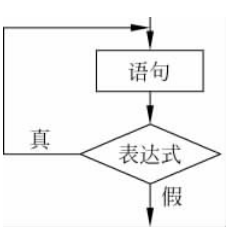

图 5.3 do…while 语句的执行过程

说明：

(1) do…while 循环的特点为先执行后判断，即先执行循环体一次，再判断表达式的值，确定是否再继续执行循环体，因此 do…while 语句的循环体至少被执行一次。

(2) while(表达式)后面必须有分号“;”，表示 do…while 语句到此结束。

【例 5.3】 用 do…while 循环求 $1+2+\cdots+100=\sum_{n=1}^{100} n$。

程序：

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

程序运行结果：

```
sum = 5050
```

说明：从本程序看出，do…while 循环和 while 循环可以完成相同的任务，都可以计算 1 到 100 的数的和。在一般情况下，两种循环处理同一问题可得到相同的结果。但当 while 后的表达式一开始就为假时，两种循环结果是不同的，请分析以下两个程序：

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

程序运行结果：

```
sum = 0
```

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

```
    return 0;
}
```

程序运行结果：

```
sum = 1
```

这两个程序的运行结果为什么不同呢？在什么条件下 do…while 和 while 能完成相同的任务，得到相同的结果？

5.3 for 语句

在 C 语言中，for 语句是最为灵活、方便的语句，它既能用于循环次数确定的情况，还能用于只给出循环结束条件的情况，用它可以完全代替 while 语句。其一般形式如下：

```
for(表达式 1;表达式 2;表达式 3)
 语句
```

以上 for 语句等价于以下 while 循环：

```
表达式 1;
while(表达式 2)
{  语句
   表达式 3;
}
```

图 5.4 表示了 for 语句的执行过程。

for 语句的执行过程如下：

(1) 求解表达式 1。

(2) 求解表达式 2，若其值为真(非 0)，则执行循环体中的语句，转(3)；若其值为假(0)，则转(4)。

(3) 求解表达式 3，转(2)。

(4) 结束循环，执行 for 循环后的语句。

循环结构应该有循环变量赋初值、循环条件和循环变量值更新三要素。在一般情况下，对 for 语句的一般形式中的 3 个“表达式”可以做下面的理解：

```
for (循环变量赋初值;循环条件;循环变量值更新)
  循环体
```

即 for 语句的表达式 1 是循环变量赋初值，表达式 3 为循环变量值更新，表达式 2 为循环条件。

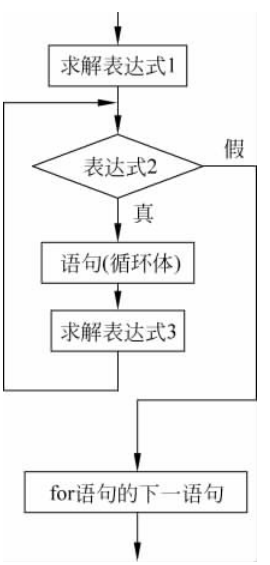

图 5.4　for 语句的执行过程

注意：若循环体包含一条以上语句，则必须用一对花括号括起来构成复合语句，否则仅把第一条语句作为循环体的内容，这样会产生逻辑错误。

for 语句的应用最为广泛、灵活，不仅适用于计数型循环，也适用于条件型循环，下面是 for 语句的使用说明。

1. 在 for 语句中省略表达式

for 语句中的 3 个表达式可以部分或全部省略，下面以 $\sum_{n=1}^{100} n$ 的程序为例说明 for 语句的各种用法。

（1）省略表达式 1，相当于省略了循环初始条件，应在 for 语句之前给循环变量赋初值。例如：

```
n = 1,sum = 0;
for( ;n <= 100;n++)  sum += n;
```

（2）省略表达式 2，即不判断循环条件，循环将无终止地进行下去，也就是认为表达式 2 始终为真，循环成为无限循环。

如：

```
for(n = 1; ;n++)                                    //无限循环
  sum += n;
```

等价于

```
n = 1;
while(1)
{ sum += n;
  n++;
}
```

（3）省略表达式 3，则必须在循环体中设法保证循环能正常结束，即有使循环趋于结束的语句。例如：

```
for(sum = 0,n = 1;n <= 100;)
 { sum += n;
   n++;                                             //表达式 3 移至循环体中
}
```

表达式 1 和表达式 3 还可同时省略，只保留表达式 2。

（4）3 个表达式都省略，但两个分号必须保留。即：

```
for(;;)语句
```

此时的循环成为“无限循环”。

2. 在 for 语句中使用逗号表达式

对于表达式 1 和表达式 3，既可以是一个简单表达式，也可以是逗号表达式，例如：

```
for(sum = 0,n = 1;n <= 100;n++)  sum += n;
```

等价于：

```
sum = 0;
for(i = 1,j = 100;i <= j;i++,j --)  sum += i + j;
```

3. 循环体为空语句

这时，for 语句的形式为：

```
for(表达式 1;表达式 2;表达式 3)
  ;
```

对于求 $\sum_{n=1}^{100} n$ 的程序段，当循环体为空语句时，可以写成：

```
for(sum = 0,n = 1;n <= 100;sum += n,n++)
 ;
```

有时为了产生一段延时，也可以用空语句作为循环体语句。例如变量 i 循环 60 000 次，但什么也不做，目的就是耗时间，则程序段可以写成：

```
for(i = 0;i < 60000;i++)
;
```

从以上讨论可知，for 语句的书写形式十分灵活，在 for 的一对括号中允许出现各种表达式，有的甚至与循环控制毫无关系，这在语法上是正确的。但初学者一般不要这样做，这样会使程序显得杂乱，降低程序的可读性。

【例 5.4】 用 for 循环求 $1+2+\cdots+100=\sum_{n=1}^{100} n$。

方法 1

程序：

```
#include <stdio.h>
int main(void)
{
    int n,sum = 0;
    for(n = 1;n <= 100;n++)
        sum += n;
    printf("1 + 2 + … + 100 = %d\n",sum);
    return 0;
}
```

程序运行结果：

```
1 + 2 + … + 100 = 5050
```

思考：程序运行结束后，n 的值是多少？

方法 2

程序：

```
#include <stdio.h>
int main(void)
{
    int i,j,sum = 0;
    for(i = 1,j = 100;i <= j;i++,j--)          //有两个循环控制变量 i、j
      sum = sum + i + j;
    printf("1 + 2 + … + 100 = %d\n",sum);
    return 0;
}
```

程序运行结果：

```
1 + 2 + … + 100 = 5050
```

说明：

(1) 方法2中，for语句中有两个循环控制变量，其中表达式1和表达式3都是逗号表达式。循环累加是从等差数列两边同时进行的，因此分别对 i 和 j 赋初值为1和100。循环体中累加 i 和 j，然后执行 $i++$，$j--$。

(2) 这样设计循环可以使循环次数减少为50次。最后一次执行循环体时，i 的值为50，j 的值为51。退出for循环后，i 值为51，j 值为50。

【例5.5】 兔子繁殖问题。设有一对新生兔子，从第3个月开始每个月都生一对兔子，按此规律，并假设没有兔子死亡，一年后共有多少兔子？

分析： 每月兔子的个数呈1、1、2、3、5、8、13、21、34……规律，这就是著名的Fibnacci数列。Fibnacci数列有以下特点，第1、2两个数为1、1，从第3个数开始，该数是其前面的两个数之和。Fibnacci数列公式为：

$$f(1)=1 \qquad (n=1)$$
$$f(2)=1 \qquad (n=2)$$
$$f(n)=f(n-1)+f(n-2) \qquad (n>2)$$

设当前项为 f，第一项为f1，第二项为f2。首先根据f1和f2推出f，f=f1+f2，然后更新f1、f2。f1=f2，f2=f，为计算下一个新项做准备，如此反复迭代下去：

```
          1       1       2       3       5
第一次    f1  +   f2  →   f
                  ↓       ↓       ↓
第二次            f1  +   f2  →   f
                          ↓       ↓
第三次                    f1  +   f2  →   f
```

程序：

```
#include <stdio.h>
int main(void)
{
  int i,f1,f2,f;
  f1 = 1;
  f2 = 1;
  printf("%6d%6d",f1,f2);                    //输出前两项
  for(i = 3;i < 13;i++)                      //迭代次数,计算数列第3项到第12项
     {
        f = f1 + f2;                         //迭代关系式
        printf("%6d",f);
        f1 = f2;                             //更新f1
        f2 = f;                              //更新f2
     }
  printf("\n");
  return 0;
}
```

程序运行结果：

```
1     1     2     3     5     8     13     21     34     55     89     144
```

注意： 当程序中有多个数据需要更新时要注意更新顺序。例如，本程序中“f1=f2;”必

须位于“f2＝f;”之前。

5.4 循环的嵌套

一个循环体内又包含另一个完整的循环结构称为循环的嵌套。在内嵌的循环中还可以嵌套循环，这就是多重循环。while 循环、do…while 循环和 for 循环可以互相嵌套，do…while 嵌套和 do…while、for 混合嵌套及 for 循环嵌套如图 5.5 所示。

多重循环的嵌套层数可以是任意的，可以按照嵌套层数分别叫二重循环、三重循环等。在设计多重循环时，要注意以下几点：

（1）使用复合语句，以保证逻辑上的正确性。

（2）内层和外层循环控制变量不能同名，以免造成混乱。

（3）采用右缩进格式书写，以保证层次的清晰性。

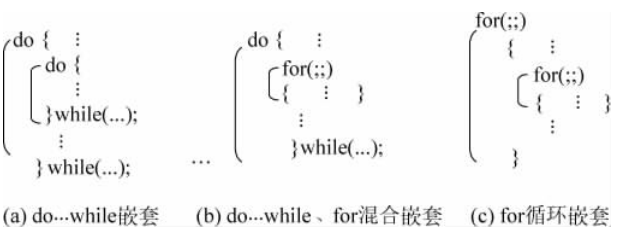

图 5.5 循环嵌套示例

【例 5.6】 循环嵌套的执行过程。

程序：

```
#include<stdio.h>
int main(void)
{
    int a,b;
    for (a = 1; a<= 4; a++)                    //外层循环开始
    {                                          //e1
        printf("a is now %d:\n", a);
        for (b=1; b<=5; b++)                   //内层循环开始
            printf("b=%d ", b);                //内层循环结束
        printf("\n");
    }                                          //e2,外层循环结束
    return 0;
}
```

程序运行结果：

```
a is now 1:
b=1 b=2 b=3 b=4 b=5
a is now 2:
b=1 b=2 b=3 b=4 b=5
a is now 3:
b=1 b=2 b=3 b=4 b=5
a is now 4:
b=1 b=2 b=3 b=4 b=5
```

说明：

(1) 本程序用for语句实现了二重循环，在a从1到4递增的for语句中嵌套了b从1到5递增的for语句。程序执行过程如下：

当a为1的时候，执行b从1递增到5的循环操作；

当a为2的时候，执行b从1递增到5的循环操作；

……

当a为4的时候，执行b从1递增到5的循环操作；

外层循环控制变量a每改变一次，内循环要完整地循环一遍。

这样，“printf("b＝%d ", b);”语句共执行了$4\times5=20$次，即总的循环次数等于外层循环次数和内层循环次数的乘积。

(2) 程序运行结果中1、3、5、7行表明了外层循环控制变量a的变化情况和执行次数，2、4、6、8行则表明了在每一次执行外层循环时内层循环控制变量b的变化情况和执行次数。

思考：在以上程序中，将e1、e2行的一组大括号去掉，程序运行结果是什么？为什么？请单步调试程序，观察变量a、b的变化情况，进一步加深对循环嵌套的理解。

【例5.7】 编写程序，输出阶梯式的乘法口诀表。

分析：乘法表具有如下特点。

(1) 共有9行，设变量i控制行号，每行式子的个数有规律，即第i行有i个式子，乘数就是i。

(2) 在第i行，被乘数从1变到i。

用循环嵌套实现，外层循环控制变量i既作为乘数，又控制输出的行数，i的取值范围为1～9；内层循环控制变量作为被乘数，被乘数j的取值范围为1～i；每输出完一行，要换行。

程序：

```
#include <stdio.h>
int main(void)
{
    int i,j;
    for(i = 1;i <= 9;i++)                    //外层循环控制变量 i 为乘数
    {
        for(j = 1;j <= i;j++)                //内层循环控制变量 j 为被乘数
            printf("%d * %d = %-4d", j,i,i * j);
        printf("\n");                        //换行
    }
    return 0;
}
```

程序运行结果：

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

说明：这是一个内层与外层相关的循环嵌套，每一行输出的乘法算式的个数是不同的。

5.5 循环语句的特点

3 种循环语句的共同特点如下：

(1) 当循环控制条件非零时执行循环体语句，否则终止循环。

(2) 语句可以是任何语句，简单语句、复合语句、空语句均可以。

(3) 在循环体内或循环条件中必须有使循环趋于结束的语句，否则会出现“死循环”。

(4) 在循环体中可以用 break 语句跳出整个循环或者用 continue 语句结束本次循环。

循环结构程序的实现要点如下：

(1) 归纳出哪些操作需要反复执行，即确定循环体。

(2) 这些操作在什么情况下重复执行，即确定循环控制条件。

一旦确定了循环体与循环条件，可选择 3 种语句来实现循环。3 种循环可以处理同一问题，一般情况下可以互相代替，但在实际应用中要根据具体情况来选用不同的循环语句：

(1) 计数型循环用于处理循环次数已知的循环过程，循环控制变量在循环过程中有规律地变化。计数型循环常选用 for 语句实现。

(2) 条件型循环用于处理循环次数未知的循环过程。条件型循环常用 while 语句或 do…while 语句。

while 和 for 语句先判断循环控制条件，do…while 语句是先执行循环体，再判断循环控制条件，所以 while 和 for 语句的循环体可能一次也不执行，而 do…while 语句的循环体至少要执行一次。

5.6 与循环有关的控制语句

在循环体内使用 break 语句、continue 语句可以改变循环状态。

5.6.1 break 语句

在第 4 章中已经介绍过用 break 语句可以使流程跳出 switch 结构，继续执行 switch 语句下面的语句。break 语句也可用于循环语句中，其作用是从循环体内跳出循环体，即提前结束循环。break 语句对循环过程的影响如下：

```
while(表达式 1)
{
    语句 1
    if(表达式 2) break;

    语句 2
}
循环后的第一条语句
```

含有 break 语句的循环流程如图 5.6 所示。

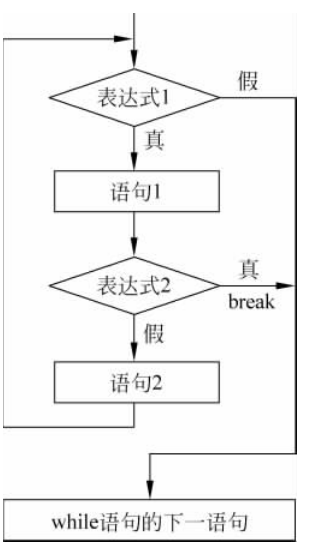

图 5.6 含有 break 的循环流程

通常，break 与 if 联合使用，表明程序在一定条件下提前结束循环。

在使用 break 语句时应注意以下几点。

(1) 由于循环语句可以嵌套使用，在这种情况下，break 语句只能跳出它所在的循环，而不能同时跳出(或终止)多重循环。例如：

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

此时的 break 语句只能从内层的 while 循环体中跳到外层的 for 循环体中，而不能同时跳出二重循环体。

(2) break 语句只能用于 switch 语句或循环语句中。

【例 5.8】 判断一个整数 $n(n\geqslant 2)$是否为素数。

方法 1

分析：素数是指除 1 和它本身之外再无其他约数的数。判断素数的方法是：

判断 $n\%k==0$ 是否成立，k 取值范围为 $2\sim(n-1)$ 或 $2\sim\sqrt{n}$。若 $n\%k=0$ 成立，则 n 不是素数，否则 n 是素数。用 N-S 结构化流程图表示算法如图 5.7 所示。

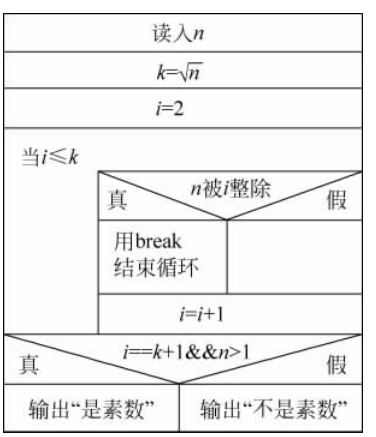

图 5.7　判断素数的算法流程图

程序：

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    int n,i,k;
    printf("Please input a number(n>1):\n");
    scanf("%d",&n);
    k = sqrt(n);
    for(i = 2;i <= k;i++)                    //从 2 开始逐一检查是否被 i 整除
        if(n%i == 0) break;                  //若不是素数可终止循环
        if(i == k + 1&&n > 1)
            printf("%d is a prime number.\n",n) ;
        else
            printf("%d is not a prime number.\n",n);
    return 0;
}
```

说明：

(1) break 语句常和 if 语句一起使用，表示当条件满足时终止循环。

(2) 那么循环结束后如何判断 n 是否为素数呢？分析 for 循环结构可知，能退出循环有两个出口，一是 $i<=k$ 条件为假；二是 break。若循环没有从 break 退出，说明 n 不能被任何一个 i 整除，是素数，因此退出循环后 i 的值是 $k+1$。

程序运行结果：

```
Please input a number(n>1):
131↙
131 is a prime number.
```

方法 2

分析：用户也可以用标记法来判断素数。首先假设 n 是素数，给标记变量 flag 赋初值为 1，在循环过程中，如果 n 能够被某一个 i 整除，则给 flag 赋值为 0，flag 置 0 后，循环提前结束。循环结束后，若标记变量 flag 的值为 0，则 n 不是素数，否则 n 是素数。

程序：

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    int n,i,flag,k;
    printf("Please input a number(n>1):\n");
    do
    scanf("%d",&n);
    while(n<2);                              //输入数据的有效性检查
    k=sqrt(n);
    for(i=2,flag=1;i<=k&&flag;i++)
        if(n%i==0) flag=0;                   //flag 为 0 表示此时的 n 不是素数,可终止循环
        if(flag)
            printf("%d is a prime number.\n",n) ;
        else
            printf("%d is not a prime number.\n",n);
    return 0;
}
```

程序运行结果：

```
Please input a number(n>1):
90↙
90 is not a prime number.
```

5.6.2 continue 语句

continue 语句只能用在循环体中，其功能是结束本次循环，即跳过循环体中在 continue 语句之后的其他语句，接着进行下一次是否执行循环的判定。它的一般形式如下：

```
continue;
```

continue 语句对循环过程的影响如下：

```
while(表达式 1)
{
    语句 1
    if(表达式 2) continue;

    语句 2
}
```

含有 continue 语句的循环流程图如图 5.8 所示。

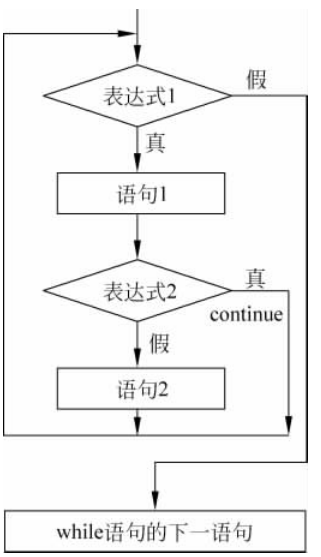

图 5.8　含有 continue 的循环流程

在 while 循环和 do…while 循环中遇到 continue 语句时，程序会转到条件表达式 1，开始下一次是否执行循环的判定；在 for 语句中若遇到 continue，程序会转到表达式 3，更新循环变量，接着进行下一次是否执行循环的判定。

continue 语句和 break 语句的区别在于，continue 语句只结束本次循环，而不是终止整个循环的执行；break 语句是结束整个循环过程，不再判断执行循环的条件是否成立。

【例 5.9】 阅读以下程序，分析 continue 语句在程序中的作用。

```
#include<stdio.h>
int main(void)
{
    int n,sum;
    sum=0;
    for(n=1;n<=100;n++)
    {
        if (n%2==0) continue;
        sum=sum+n;                          //e1
    }
    printf("sum=%d",sum);
    return 0;
}
```

程序运行结果：

```
sum=2500
```

说明： 当 if 语句条件满足时，即 n 为偶数时执行 continue，则 e1 行的累加语句被跳过，程序转到 for 循环的表达式 3，执行 n++，继续新的循环，因此 sum 的结果是 1 到 100 间奇数的和。

5.7　循环结构程序设计实例

【例 5.10】 编程求 100～1000 的全部“水仙花数”。“水仙花数”是这样一个整数，它的每一位数字的立方和正好等于这个 3 位数。例如 153 是水仙花数，因为 $1^3+5^3+3^3=153$。

方法 1

分析： 由于水仙花数是 3 位数，所以穷举 100～999 的所有数值。利用 for 循环控制 n

从 100 到 999，对每个 n 分解出个位、十位、百位，进而判断水仙花条件是否满足。

程序：

```
#include <stdio.h>
int main(void)
{
    int i,j,k,n;
    printf("The numbers are:\n");
    for(n=100;n<1000;n++)
    {
        i=n/100;                                    //分解出百位
        j=n/10%10;                                  //分解出十位
        k=n%10;                                     //分解出个位
        if(i*i*i+j*j*j+k*k*k==n)                    //判断水仙花条件
            printf("%-5d",n);
    }
    return 0;
}
```

程序运行结果：

```
The numbers are:
153  370  371  407
```

方法 2

假设整数 i、j、k 分别表示 3 位整数 n 的百位、十位和个位，用 i、j、k 作为循环变量建立嵌套循环，如果这 3 个变量组成的 3 位数 $n(n=i*100+j*10+k)$ 满足水仙花数的构成条件，则这个 3 位数就是水仙花数。

请读者根据以上算法写出程序。

本程序采用的方法称为穷举法，即把所有可能的情况一一测试，筛选出符合条件的各种结果进行输出。

采用穷举法求解问题的思路如下：

(1) 确定穷举变量；

(2) 确定穷举范围；

(3) 确定判定条件。

【例 5.11】 求 100～200 的所有素数。

分析：可分下面两步实现。

(1) 判断一个数是否为素数可采用例 5.8 的程序。

(2) 对指定范围 100～200 的每一个数判断其是否为素数。本程序用循环嵌套来实现，本程序中 i 是穷举变量，穷举范围是 100～200，判定条件即素数的判定条件。

程序：

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    int n,i,j,k;
    for(i=101,n=0;i<200;i+=2)                   //外循环：为内循环提供一个奇数 i
```

```
    {
        k = sqrt(i);
        for(j = 2;j <= k;j++)                    //内循环: 判断整数 i 是否为素数
            if(i % j == 0) break ;
            if(j == k + 1)                       //整数 i 是素数: 输出,计数器加 1
            { n = n + 1;
            printf("%6d",i);
            }
            if(n % 10 == 0) printf("\n");        //输出 10 个数换一行
    }
    return 0;
}
```

程序运行结果：

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

说明：

(1) 大于 2 的素数全为奇数,所以 i 从 101 开始,每循环一次 i 值加 2。

(2) n 的作用是统计素数的个数,控制每行输出 10 个素数。

【例 5.12】 用公式“π/ 4 ≈1－1/ 3＋1/5－1/ 7＋…”求 π 的近似值,直到最后一项的值小于 10^{-4} 为止。

分析：

(1) 这也是一个求累加和的问题,本程序的关键是找出前后项之间的递推关系。

分母 n 的取值：1、3、5、7 …

奇偶项符号 s 的变化：＋、－、＋、－

因此公式中的第 i 项是前一项分母加 2、符号取反得到的。

(2) 循环条件：fabs(t)＞＝10^{-4},t 为某项值,因此选用 while 语句实现循环。

算法流程图如图 5.9 所示。

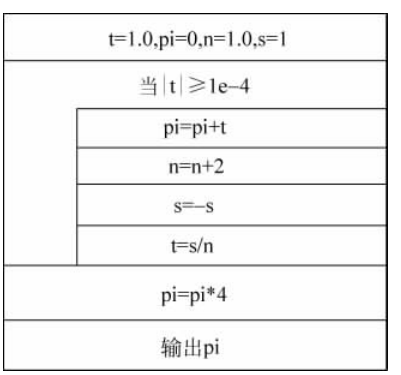

图 5.9　求 π 值的算法流程图

程序：

```
#include <math.h>
#include <stdio.h>
int main(void)
{
    int s;
    double n,t,pi;
    n = 1.0; s = 1; t = 1.0; pi = 0;
    while(fabs(t)>= 1e - 4)                  //判断某项
    {
        pi = pi + t;                         //累加和
        n = n + 2;                           //改变分母
        s =- s;                              //符号变反
        t = s/n;                             //计算下一项
```

```
    }
    printf("pi = %10.6f\n", pi * 4);
    return 0;
}
```

程序运行结果：

```
pi = 3.141393
```

思考：本程序还有其他什么算法？能否构造一个通项公式，然后累加呢？

【例 5.13】 百钱买百鸡问题。公元前 5 世纪，我国古代数学家张丘建在《算经》一书中提出了“百钱百鸡”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，问鸡翁、鸡母、鸡雏各几何？

方法 1

分析：本题可用穷举法。设 3 种鸡的个数分别为 x、y、z。则有以下不定方程组：

$$\begin{cases} x+y+z=100 \\ 5x+3y+\dfrac{z}{3}=100 \end{cases}$$

将鸡的个数作为穷举变量，将所有可能的 x、y、z 值一个一个去试，看是否满足以上方程组中的两个判定条件，从而找到问题的解。

程序：

```
#include <stdio.h>
int main(void)
{
    int x,y,z;
    for(x = 1;x <= 20;x++)                          //枚举公鸡的可能数量,最多为 20
        for(y = 1;y <= 33;y++)                      //枚举母鸡的可能数量,最多为 33
            for(z = 3;z <= 100;z += 3)              //枚举小鸡的可能数量,注意 z 是 3 的倍数
            {
                if(x + y + z == 100&&5 * x + 3 * y + z/3 == 100)  //判定条件
                    printf("cocks = %d,hens = %d,chickens = %d\n",x,y,z);
            }
    return 0;
}
```

程序运行结果：

```
cocks = 4,hens = 18,chickens = 78
cocks = 8,hens = 11,chickens = 81
cocks = 12,hens = 4,chickens = 84
```

说明：在本程序中，循环体执行了 $20\times33\times33=21\,780$ 次。

方法 2

分析：在百钱买百鸡问题中，由于 3 种鸡的和是固定的，因此只需要枚举两种鸡的个数，第 3 种鸡的个数可以根据判定条件求得，这样循环就变成了二重循环。

程序：

```
#include <stdio.h>
```

```
int main(void)
{
    int x,y,z;
    for(x=1;x<=20;x++)
        for(y=1;y<=33;y++)
        {
          z=100-x-y;
            if(5*x+3*y+z/3==100&&z%3==0)
              printf("cocks=%d,hens=%d,chickens=%d\n",x,y,z);
        }
    return 0;
}
```

说明：

(1) 在本程序中，循环体只需执行 $20\times33=660$ 次。

(2) 穷举算法的思路简单，但运算量大。当问题的规模变大时，循环嵌套的层数增多，执行速度变慢，因此应该注意优化算法，减少运算量。

思考：能否进一步优化该程序，使循环体执行的次数更少？

【例 5.14】 用牛顿迭代法求方程 $3x^3-2x^2+5x-1=0$ 在 0 附近的近似值。

分析：牛顿迭代法是以切线与 x 轴的交点作为曲线与 x 轴交点的近似值以逐步逼近解。用牛顿迭代法求 $f(x)$的根的方法如下：

(1) 任选一个接近于真实根x_a的近似根 x_0。

(2) 过$(x_0,f(x_0))$做曲线 $f(x)$的切线，交 x 轴于x_1。

(3) 再过$(x_1,f(x_1))$做 $f(x)$的切线，交 x 轴于x_2；重复以上步骤，求出 x_1、x_2、x_3、…、x_n、x_{n+1}，直到前、后两次求出的近似根之差的绝对值$|x_{n+1}-x_n|$达到所给精度要求为止，此时认为 x_{n+1}是足够接近于真实根的近似根。

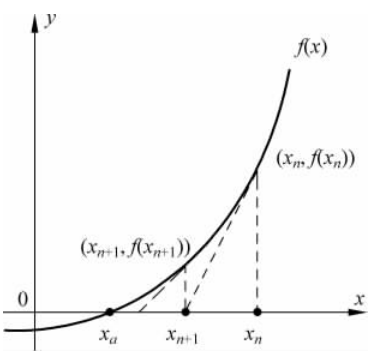

图 5.10　牛顿迭代法示例

如图 5.10 所示，过$(x_n,f(x_n))$点做切线，切线方程为：

$$f'(x_n)=\frac{-f(x_n)}{x_{n+1}-x_n}$$

即

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}\quad(n=0,1,2,3\cdots)$$

这就是牛顿迭代公式。

根据题目：

$$f(x)=3x^3-2x^2+5x-1$$

求出：

$$f'(x)=9x^2-4x+5$$

程序：

```
#include<stdio.h>
```

```
#include <math.h>
int main(void)
{
    int n = 1;
    double x0,x,f1,f2;
    printf("Please input x:\n");
    scanf("%lf",&x);
    do
    {
        x0 = x;
        f1 = 3 * x * x * x - 2 * x * x + 5 * x - 1;
        f2 = 9 * x * x - 4 * x + 5;
        x = x0 - f1/f2;                          //迭代关系式
        n = n + 1;
    }while(fabs(x0 - x)> 1e - 6);                //当未满足精度要求时继续循环
    printf("x = %-15.10lf(n = %d)\n",x,n);
    return 0;
}
```

程序运行结果：

```
Please input x:
0↙
x = 0.2122861022   (n = 5)
```

说明：迭代就是一个不断地用新值取代旧值的过程。为了使迭代变量的新、旧值能进行比较，必须在求新值之前将旧值保存在另一个变量中。在该程序中用 x0 存放旧值，用 x 存放新值。

【例 5.15】 例 4.11 使用 switch 语句实现了从主菜单中选择一项并显示相关信息，本例使用 do…while 循环实现在程序中重复显示学生成绩管理系统菜单，供用户输入选项。

程序：

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    do
    {
        int chose;
        printf("\t\t*****************************************\n");
        printf("\t\t          学生成绩管理系统                \n\n");
        printf("\t\t\t  1. 输入数据 \n");
        printf("\t\t\t  2. 按学号查询 \n");
        printf("\t\t\t  3. 删除成绩信息 \n");
        printf("\t\t\t  4. 输出学生成绩信息 \n");
        printf("\t\t\t  0. 退出 \n");
        printf("\t\t*****************************************\n");
        printf("\n 请输入 0～4 选择操作:\n");
        do
        scanf("%d", &chose);
```

```
        while(chose<0||chose>4);
        system("cls");                          //清屏
        switch(chose)
        {
        case 1:
            printf("你选择的是'输入数据'\n");
            break;
        case 2:
            printf("你选择的是'按学号查询'\n");
            break;
        case 3:
            printf("你选择的是'删除成绩信息'\n");
            break;
        case 4:
            printf("你选择的是'输出学生成绩信息'\n");
            break;
        case 0:
            printf("\n THANKS FOR USEING! \n");
            exit(0);
        default:
            printf("\n\n\n 错误,请重新输入! \n");
        }

    } while(1);
    return 0;
}
```

说明：本程序只是完成重复显示主菜单，在选择某一选项后，并没有真正完成相应的处理操作，相应操作将在后续章节介绍。

习　　题

1. 验证“哥得巴赫猜想”，即任何一个大于 6 的偶数均可表示为两个素数之和，要求输入一个大于 6 的偶数 n，输出 6 到 n 之间所有偶数分解为两个质数和的情况。例如输入 12，输出：

6＝3＋3

8＝3＋5

10＝3＋7

12＝7＋5

方法提示：

依次判断 3 到 m 之间的奇数 a 是否为素数，若是则判断 $m-a$ 的结果 b 是否也是素数，满足条件则输出，其中 m 取 6 到 n 之间的偶数。

2. 从键盘输入一行字符，直到遇到换行符为止，分别统计其中字母（不区分大小写）、数字字符和其他字符的个数。

3. 国王的许诺。相传国际象棋是古印度舍罕王的宰相达依尔发明的，舍罕王十分喜

欢象棋，决定让宰相自己选择何种赏赐。一位聪明的宰相指着8×8共64格的象棋盘说：陛下，请您赏给我一些麦子吧，就在棋盘的第一个格子中放1粒，第2格中放两粒，第3格放4粒，以后每一格都比前一格增加一倍，依此放完棋盘上的64个格子，我就感恩不尽了。

舍罕王让人扛来一袋麦子，他要兑现他的许诺。国王能兑现他的许诺吗？试编程计算舍罕王共要多少麦子赏赐他的宰相，这些麦子合多少立方米？（已知1立方米麦子约1.42e8粒）

4. 从键盘上输入10个学生百分制成绩score(0～100)，按下列原则输出每个学生成绩相应的等级：score≥90，等级为优秀；80≤score<90，等级为良好；70≤score<80，等级为中等；60≤score<70，等级为及格；score<60，等级为不及格。若输入的成绩不在0～100范围内，则输出“成绩超出了范围”。

5. 有1020个西瓜，第一天卖一半多两个，以后每天卖剩下的一半多两个，问几天以后能卖完？

6. 从键盘输入一个整数，统计该数的位数。例如输入12534，输出5；输入－99，输出2；输入0，输出1。

7. 现有12亿人，按年2%的增长速度，10年后将有多少人？

8. 韩信有一队兵，他想知道有多少人，便让士兵排队报数。按从1至5报数，最末一个士兵报的数为1；按从1至6报数，最末一个士兵报的数为5；按从1至7报数，最末一个士兵报的数为4；最后按从1至11报数，最末一个士兵报的数为10。请编程计算韩信至少有多少兵？

9. 搬砖问题：36块砖，36人搬，男搬4，女搬3，两个小孩抬一块砖（小孩不分性别），问有多少男人，多少女人，多少小孩？

10. 两个乒乓球队进行比赛，各出3人。甲队为A、B、C 3人，乙队为X、Y、Z 3人，已抽签决定比赛名单。有人向队员打听比赛的名单，A说他不和X比，C说他不和X、Z比，编程找出3对赛手的名单。

11. 输出以下图形。

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