第5章

迭代与循环结构

许多问题可以通过重复相同的操作来完成，通过指定次数或设定条件来控制执行过程。多次重复执行的结构称为循环，每一次循环称为迭代。

5.1 循环结构

假如要求系统输出5个随机数，可以重复写5条输出随机数的语句。如果要求输出10个随机数怎么办？100个呢？此时可以利用循环语句，设计一个变量n记录要输出的随机数的个数，再定义一个计数器变量counter，初始值为1，每一次循环，输出一个随机数，然后计数器加1，当计数器大于n时循环结束。这样就可以根据用户的要求，用一条输出语句重复执行若干次得到结果。用流程图表示的这个程序如图5-1所示。

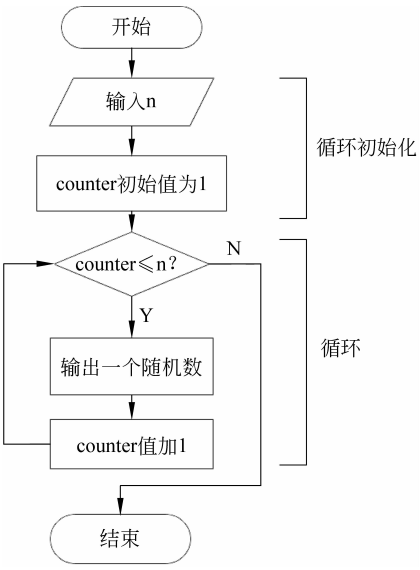

图5-1 输出5个随机数的程序流程图

计算机的运算速度很快。只要找到解决问题的规律，学会循环语句，掌握控制循环条件的方法，其余的工作就可以交给计算机完成。

5.2　循环控制语句

在 C&C ++ 中，常见的循环控制结构有 while 循环、do…while 循环和 for 循环。while 循环与 for 循环是当型循环，当满足条件时开始迭代，然后再判断条件；而 do…while 循环是直到型循环，首先开始迭代，然后判断条件，决定是否重复迭代。

5.2.1　while 语句

while 循环语句的特点是先判断条件，然后确定是否执行循环体。当条件为 true(0) 时，执行循环体，然后再判断条件，重复执行循环体，直到条件为 false(0)。

while 循环语句的一般形式为

```
while (表达式)
{
    循环体
}
```

其中：

(1) 表达式放在 while 后的小括号内，可以是任何表达式(true 或非 0 值表示真，false 或 0 值表示假)，也可以省略表达式(永远为真)。若表达式的值为真，则循环执行循环体的语句，否则退出循环。

(2) while 后面的循环体在逻辑上只能是一条语句，因此循环体一般是一条复合语句，用一对大括号括起来。如果只有一条语句，大括号可省略。

循环能够执行若干次，则条件表达式或循环体中一定有关于循环终止的语句。否则循环一直无限重复，称为无限循环或死循环。

while 循环的执行流程为：①计算表达式的值，若值为真，执行大括号内的语句；②再次计算表达式的值，若值为真，则再次大括号内的语句；③重复，直到表达式值为假，终止循环，继续向下进行。while 循环语句流程图如图 5-2 所示。

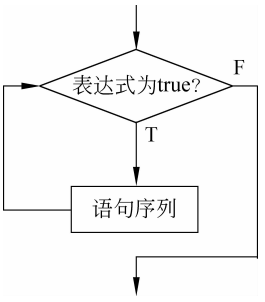

图 5-2　while 循环语句流程图

【例 5-1】 使用 while 循环显示 0～99，各数字间有一个空格，每行显示 5 个数字。

```
#include <iostream>
using namespace std;
int main()
{
    int x=0;
    while(x<100)
{
    cout<<x <<" ";
    x++;
     if(x%5==0)
         cout<<endl;
```

```
12  }
13  return 0;
14  }
```

程序运行时,第 6 行检查条件表达式 x＜100。首次执行循环时,x 的值为 0,表达式的值为 true,开始执行循环体(第 7～12 行)。执行完毕后,再次计算表达式 x＜100 的值,在循环体句中每次迭代后 x 的值自增 1,因此要迭代 100 次,当 x 的值为 100 时循环结束。

第 10、11 行的 if 语句是一种常见的算法,x 的值若是 5 的倍数就输出一个换行符,从而实现在一行显示 5 个数字的功能。

对本程序适当做些修改,就可以实现其他类似的功能:

(1) 将第 9 行的 x＋＋变成 x＋＝2,则输出结果就是输出 0～99 的所有偶数。

(2) 将第 10 行的 x％5＝＝0 改为 x％10＝＝0,就是每行输出 10 个数字。

5.2.2 for 语句

for 语句是一个功能强大的循环控制语句,用法非常灵活,几乎可以用于任何需要循环的场合。for 与 while 一样,都是当型循环。

for 循环语句的一般形式为

```
for(表达式 1; 表达式 2 ; 表达式 3)
{
    循环体
}
```

for 语句将 while 语句中关于循环的 3 个表达式都写在 for 后面的小括号中。表达式 1 主要用于循环条件中的变量初始化,只执行一次;表达式 2 是 while 后面小括号中的循环条件,当表达式 2 为真时执行 for 后面的一条语句,可以是用一对大括号括起来的复合语句或空语句等任何一条语句;执行完一次迭代工作后,计算表达式 3 的值,一般是循环条件中的变量值的修改,然后重复判断表达式 2,为真重复循环过程,为假结束循环。

注意:for 中的 3 个表达式之间用分号分隔,不能省略;而 3 个表达式都可以省略,如果在 for 语句的前面已经对循环变量做了初始化,则表达式 1 可以省略;如果循环条件永远为真,则表达式 2 可以省略;如果在循环体中写表达式 3 的内容,则表达式 3 可以省略。

for 循环语句流程图如图 5-3 所示。

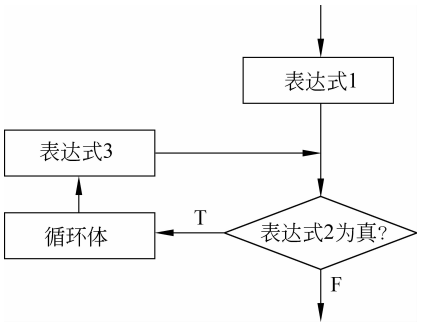

图 5-3 for 循环语句流程图

for 语句完全可以转化为 while 语句:

```
表达式 1;
while(表达式 2)
{
    循环语句序列;
```

```
    表达式 3;
}
```

【例 5-2】　使用 for 循环计算 1～100 的和。

算法分析：在数学中的计算过程为

1＋2 ＝3　3＋3＝6　6＋4＝10　10＋5＝15 15＋6＝21　…

加法运算要重复多次，每次将当前的加数和之前的求和结果进行加法运算，得到当前的求和结果。因此，定义一个记录和的变量 sum，初值为 0，加数的变量 i 从 1 开始，每次与 sum 求和后将值赋给 sum，即 sum＝sum＋i，然后 i 增 1，变为 2，再次重复 sum＝sum＋i，直到 i＝100，求和完毕，共重复 100 次求和操作。

for 语句特别适合循环次数确定的情况，将 3 个表达式都写在 for 后面的小括号中，就可以计算出循环次数，代码比较简洁。

参考代码如下：

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int i, sum=0;
6      for ( i=1; i<=100; i++)
7      {     //本例的 for 语句只有一条赋值语句,可以不写大括号
8          sum =sum +i;          //等价于 sum +=i ;
9      }
10     cout<<"Sum: " <<sum <<"\n";
11     return 0;
12 }
```

对本程序适当做些修改，就可以实现其他类似的功能：

(1) 求 1 ＋ 3 ＋… ＋ 99。

从第 2 个数起，每个数比前一个数大 2，将第 6 行 i＋＋改为 i＋＝2。

(2) 求 1 － 2 ＋ 3 － 4 ＋… －100。

从第 2 个数起，每个数比前一个数的绝对值大 1，符号位相反，则增加一个保存符号的变量 flag，然后在每次求和后修改 flag 的值为－flag，第 5～9 行代码段改为

```
int i, sum=0 ,flag =1;
for ( i=1; i<=100; i++)
{
    sum =sum +flag * i;
    flag =-flag;
}
```

(3) 求 $n!$。与求和代码类似，将积的初始值设为 1，将累加运算改为累乘运算。第 5～9 行代码段改为

```
int i, sum=1;
for(i=1; i<=10; i++)
{
    sum =sum * i;                    //等价于 sum *=i;
}
```

for 循环功能强大而灵活,因此它的变化形式也很多。下面仍以求 1～100 的和为例,介绍 for 的几种变化形式。

5.2.2.1 for 的变化形式 1:表达式 1 由多个表达式组成

如果在表达式 1 中给多个变量赋值,必须用逗号分隔它们(称为逗号表达式,表达式的值就是最后一个表达式的值)。例如:

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

上述程序段中,声明了两个 int 型变量 i 和 sum,在 for 循环语句的表达式 1 中为两个变量初值,两个赋值语句以逗号分隔,且以分号结尾。

5.2.2.2 for 的变化形式 2:表达式 1 为空

可将表达式 1 的语句移到 for 循环语句之前,但 for 语句中的分号不可省略。例如:

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

5.2.2.3 for 的变化形式 3:表达式 3 为空

可将表达式 3 移到循环体内,但 for 语句中分的分号不可省略。例如:

```
int i, sum=0;
for(i=1; i<=100;)
{
    sum =sum +i;
    i++;
}
```

5.2.2.4 for 的变化形式 4:表达式 2 为空

若表达式 2 为空,则循环条件一直为真,可以将表达式 2 的值写到循环体内,用 break 语句退出循环。同样,for 语句中的分号不可省略。例如:

```
int i, sum=0;
for(i=1; ; i++)
```

```
{
    sum =sum +i;
    if(i>100)
        break;                    //结束循环
}
```

5.2.2.5　for 的变化形式 4：3 个表达式全为空

for 循环语句内的 3 个表达式都可以省略，但分号不可以省略。此时 for 循环语句退化为 while(true)，表示循环条件永远为真。

```
int i=1,sum=0;
for(;;)
{
    sum=sum+i;
    i++;
    if(i>100)
        break;
}
```

5.2.2.6　for 的变化形式 5：表达式 3 由逗号表达式组成

可将 for 循环体内的语句转移到表达式 3 中，此时表达式 3 由多个表达式组成，它们用逗号分隔，按顺序执行。为了保证程序的可读性，不提倡这种用法。例如：

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

或

```
for(i=1; i<=100; sum=sum+i, i++);      //末尾有分号，表示条件为真时执行空语句
```

或：

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

这 3 个 for 语句的功能相同。

5.2.3　do…while 语句

do…while 循环语句是先执行循环体，再判断是否继续下一轮迭代的循环语句。do…while 是直到型循环，也就是说，首先执行一次迭代，然后再判断条件，若满足了就重复。直到型循环与当型循环的区别是：当循环条件初始值为假时，直到型循环要执行一次迭代，而当型循环什么也不做。

do…while 语句的一般形式为

```
do
{
    循环体
} while(表达式);
```

do…while 语句执行流程为：先执行大括号里的循环体，再计算表达式的值。若表达式为真，继续执行循环体，重复判断表达式的值，直到表达式为假时终止循环。do…while 循环语句流程图如图 5-4 所示。

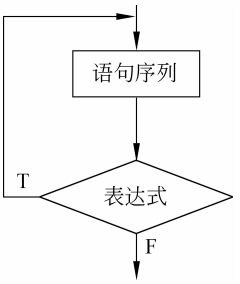

图 5-4 do…while 循环语句流程图

【例 5-3】 输入若干个字符，以'@'字符结束。统计并输出小写英文字母的个数。

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int count =0;
6      char ch;
7      cout <<"Input characters:\n";
8      do
9      {
10         cin >>ch;
11         if(ch >='a' && ch <='z')
12             count++;
13     } while(ch!='@');
14     cout <<"There are " <<count <<" lowercase(s).\n";
15     return 0;
16 }
```

运行示例如下：

```
Input characters:
ab123
dfIOz
fd@
There are 7 lowercase(s).
```

初始化变量 count 值为 0，作为计数器。在 do…while 循环中，检查条件前进入循环体，因此循环体至少执行一次。第 11 行判断条件，若输入的字符是小写字母，则执行第 12 行的语句：计数器 count 加 1。执行到第 13 行，循环条件若为真，就跳转到循环的开始处(第 8 行)继续执行，否则结束循环，从第 14 行继续向下执行执行。在本例中，用户输入 3 行字符，只有 7 个是小写字母：abdfzfd。

5.2.4　陷阱：循环的常见问题

5.2.4.1　死循环

死循环是程序设计中经常会遇到的一个现象，指程序的循环条件一直为真，程序一直陷入在循环语句中，也称为无限循环。一旦程序进入死循环，光标会一直闪烁，按任何键均无反应，此时只能按 Ctrl＋C 组合键强行终止程序的运行。

造成死循环的主要原因是循环中缺少能让循环条件变假的操作。

【例 5-4】 死循环示例。

下面几个程序都会陷入死循环，请观察它们与例 5-1 的区别，找到错误并修改。

(a)

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x=0;
6      while(x<100)
7          cout<<x <<" ";
8          x++;
9      if(x%5==0)
10         cout<<endl;
11     return 0;
12 }
13
14
```

(b)

```
#include <iostream>
using namespace std;
int main()
{
    int x=0;
    while(x<100)
    {
        cout<<x <<" ";
        if(x%5==0)
            cout<<endl;
    }
    return 0;
}
```

(c)

```
#include <iostream>
using namespace std;
int main()
{
    int x=0;
    while(0=<x<100)
    {
        cout<<x <<" ";
        x++;
        if(x%5==0)
            cout<<endl;
    }
    return 0;
}
```

说明：

(1) 程序(a)中 while 后面没有大括号，所以循环体语句只有一条：cout＜＜x＜＜" ";，执行完毕后就继续检查 while 小括号中的条件 x＜100，永远为真，因此程序陷入死循环。要将第 7～11 行的代码用一对大括号括起来，形成一条复合语句。

(2) 程序(b)的循环体中没有修改循环变量 x 的语句，x＜100 永远为真，因此程序陷入死循环。要在第 8 行后面增加一条 x＋＋;语句。

(3) 程序(c)的第 6 行 0＜＝x＜100 永远为真，因此程序陷入死循环。可以改为 0＜＝x && x＜100。

5.2.4.2　多余的分号

看下面的代码段：

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

看起来,这段代码的作用是求 1～100 的和,但是运行程序时却发现,循环结束后,sum 的值是 1,只执行了一次 sum = sum + i;语句。原因是 for 语句括号后面多了一个分号,而分号是语句的结束标记,也表示空语句。该代码段实际是

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

循环体是一条空语句,循环结束后执行 sum = sum + i;语句。

再看下面的程序段:

```
int x=0;
while( x<100 );
{
    cout<<x <<" ";
    if(x%5==0)
        cout<<endl;
}
```

由于 while 后面的右括号处多了一个分号,导致循环语句变为

```
while(x<100)
    ;
```

循环条件一直为真,执行空语句,陷入死循环。

5.3 循环和迭代的提前结束

在循环体中允许提前结束循环,相应的语句包括 break(退出整个循环)、continue(退出本次迭代)、goto(转到指定语句行)。本节介绍 break 和 continue。

5.3.1 break 语句

break 语句可以在循环体中使用,用来终止整个循环的执行,而不需要等待表达式值为假(这意味着通过 break 结束循环后,循环条件表达式仍然为真)。

通常 break 语句总是与 if 语句连用,即满足条件时便跳出所属的循环语句。

【例 5-5】 自定义函数,利用迭代公式求 x 的立方根。

求 $\sqrt[3]{x}$ 的迭代公式为

$$x_{n+1} = \frac{1}{3}\left(2x_n + \frac{x}{x_n^2}\right)$$

精度要求为

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| < 10^{-6}$$

算法分析:该循环没有明确的次数条件,当满足 $|(x_{n+1} - x_n)/x_n| < 10^{-6}$ 时结束循

环,因此至少要定义两个变量 x1 和 x2 以保存前后相邻的两个值。

代码如下:

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
double Cbrt(double x);        //自定义函数求 x 的立方根,为了与库函数区分,首字母大写
int main()
{
    double x;
    cin >>x;
    cout <<fixed <<setprecision(15);
    cout <<Cbrt(x) <<endl;        //调用自定义函数求 x 的立方根
    cout <<pow(x,1.0/3) <<endl;  //调用 cmath 中的 pow 函数求 x 的 1/3 次幂
    cout <<cbrt(x);               //调用 cmath 中的 cbrt 函数求 x 的立方根
    return 0;
}
    double Cbrt(double x)
{
    double x1=1, x2 ;
    while(1)                      //也可以写为 while(true)或 while()
    {
        x2 = ( 2 * x1 +x / (x1 * x1 ) ) / 3;
        if ( fabs ( ( x2 -x1 ) / x1 ) <1e-6 )
            break;
        x1 =x2;
    }
    return x2;
}
```

运行示例如下:

```
29
3.072316825686780
3.072316825685847
3.072316825685848
```

说明:本例可以不用 break 语句,Cbrt 函数定义可修改为

```
double Cbrt(double x)
{
    double x1,x2=1;
    do
    {
```