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 counters，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。

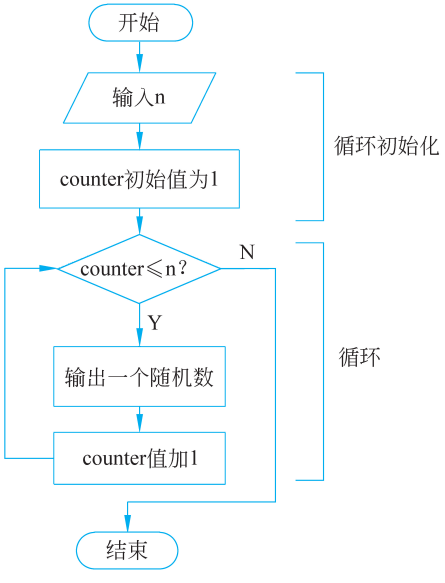

图 5-1 输出 5 个随机数的程序流程图

计算机的运算速度很快，只要找到解决问题的规律，利用循环语句，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,首先开始依次迭代,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(非零)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 后的圆括号内,可以是任何表达式(true 或非 0 值表示真,false 或 0 值表示假)。若表达式的值为真则执行循环体的语句,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,用一对花括号括起来。如果只有一条语句,花括号可省略。

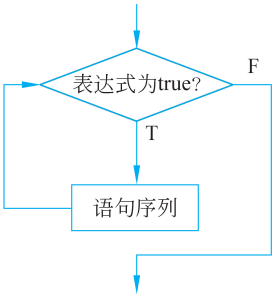

图 5-2 while 循环语句流程图

(3) 循环能够执行若干次,则条件表达式或循环体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,执行花括号内的语句;②再次计算表达式的值,若值为真,则再次执行花括号内的语句;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。

【例 5-1】 输出[1-10]的所有整数,每个数之间占 1 个空格。首尾无空格。

算法分析:定义一个 int 变量 x 初始值为 1,输出 x。然后判断是否继续输出:当 x 小于或等于 10 时执行循环,循环体中输出空格与 x,然后让 x 值增 1。循环代码段如下:

```
int x = 1;
cout << x;
while(x<=10){
    cout << " " << x;
    x++;
}
```

【例 5-2】 输出 n 个[1,100]的随机整数,每个数占 5 位且右对齐,每行显示 5 个。

算法分析:首先将任务分解为两个子功能。

子功能一:生成一个[start,end]的随机整数。

子功能二:循环显示 n 个值。循环体中,每输出 5 个值后输出一个换行符,每个值按指

定宽度格式输出。

将两个子功能分别自定义为函数。

(1) 自定义 RandRange 函数。利用 cstdlib 库中的 rand 函数生成随机数，再按[start，end]范围要求写出数学表达式。

(2) 自定义 PrintRandRange 函数。首先通过 ctime 库中的 time 函数和 cstdlib 库中的 srand 函数初始化随机种子，再通过循环控制 n 次，每次调用 RandRange 函数得到一个随机数，使用 iomanip 库中的 setw 方法指定输出值的格式，再判断输出个数是否 5 的倍数，如果是则输出一个换行符。

主函数接收用户输入的 n，调用 PrintRandRange 函数实现输出。

完整程序如下：

```
#include<iostream>
#include<iomanip>
#include<cstdlib>
#include<ctime>
using namespace std;
int RandRange(int start, int end);                  //生成[start,end]的随机整数
void PrintRandRange(int n,int start,int end);       //输出 n 个随机整数
int main(){
    int n;
    cin >> n;
    PrintRandRange(n,1,100);                        //输出 n 个[1,100]的随机整数
    return 0;
}
int RandRange(int start, int end){
    return rand()%(end - start+1) + start;
}
void PrintRandRange(int n,int start, int end){
    if(n<1)                                         //n 非有效值,结束函数
        return;
    srand(time(0));                                 //初始化随机种子
    int cnt = 1;                                    //定义计数器变量 cnt 初始为 1
    while(cnt <= n) {                               //循环条件: 小于或等于 n 个数
        cout<<setw(5)<<RandRange(start,end);        //输出[start,end]的随机整数
        if(cnt %5 == 0)                             //每 5 个数输出一个换行符
            cout<< endl;
        cnt ++;                                     //计数器加 1
    }
}
```

运行结果如下：

```
23↙
   24   77    4   34   78
   38   38   31   74   75
   20   23   67   38   52
    4   26   90   42   55
   43   64   15
```

要完成 while 循环，需要有 3 个表达式：第 1 个表达式是循环变量 cnt 的定义及初始化

为 1,第 2 个表达式是 while 括号内用于控制执行循环的条件表达式 cnt<=n,第 3 个表达式是循环体中的 cnt++修改了循环变量的值,使得本次循环体结束后再次回到表达式 2,进而控制循环式继续重复执行还是结束。

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 后面的圆括号中。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 后面圆括号中的循环条件,当表达式 2 为真时执行 for 后面的一条语句,可以用一对花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。

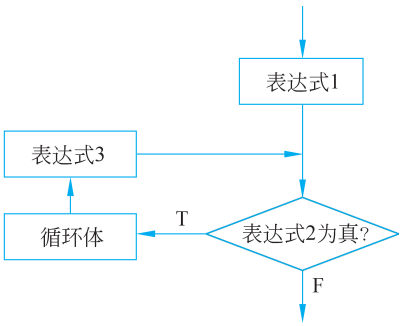

图 5-3 for 循环语句流程图

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。

for 语句完全可以转换为 while 语句:

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

修改例 5-2 代码,将 while 语句改为下面的 for 语句:

```
for(int cnt = 1; cnt <= n; cnt++ ) {                    //循环条件: 小于或等于 n 个数
    cout << setw(5) << RandRange(start,end);            //输出[start,end]的随机整数
    if(cnt %5==0)                                       //每 5 个数输出一个换行符
        cout<< endl;
}
```

由于 cnt 的主要作用是应用在 for 语句中,因此可以在表达式 1 中定义并初始化 cnt,当 for 语句结束后会释放 cnt 的内存。如果 for 语句结束后要使用 cnt 变量,则 cnt 要在 for 语句前定义,此时 for 语句中省略表达式 1。例如改为:

```
int cnt = 1;
for(; cnt <= n; cnt++ ) {                          //循环条件：小于或等于 n 个数
    cout << setw(5) << RandRange(start,end);       //输出[start,end]的随机整数
    if(cnt %5==0)                                  //每 5 个数输出一个换行符
        cout<< endl;
}
```

【例 5-3】　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 后面的圆括号中，就可以计算出循环次数，代码比较简洁。

自定义 Sum 函数如下：

```
int Sum(int start,int end) {          //返回[start,end]的值的和
    int i, sum=0;
    for (i=start; i<=end;i++){        //本例 for 循环体只有一条语句，可以不写花括号
        sum = sum + i;                //等价于 sum += i;
    }
    return sum;
}
```

对本函数适当做些修改，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 for 中 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，修改函数体语句为：

```
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，累加运算改为累乘运算。修改函数体语句为：

```
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。

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。

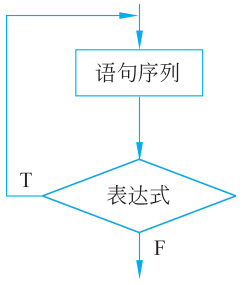

图 5-4 do…while 循环语句流程图

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:先执行花括号里的循环体,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。

【例 5-4】 输入若干字符,以'@'字符结束。统计并输出小写英文字母的个数。

```
#include<iostream>
using namespace std;
int main(){
    char ch;
    cout << "Input Characters:\n";
    int cnt = 0;
    do{
        cin >> ch;
        if(ch >= 'a' && ch <= 'z')          //小写字母
            cnt++;
    } while (ch!='@');
    cout << "There are " <<cnt<< " lowercase(s).\n";
    return 0;
}
```

运行结果如下:

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

初始化变量 cnt 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。if 语句判断条件,若输入的字符是小写字母,则执行计数器 cnt 加 1。然后判断 while 条件表达式,若为真,跳转到循环开始处继续执行,否则结束循环。在本例中,用户输入 3 行共 13 个字符,只有 7 个是小写字母:abdfzfd。

注意: 本例统计小写英文字母的个数。因此空格、回车符都不在统计个数内。接收一个字符使用了 cin>>ch。如果要统计所有字符个数包括空格或回车符,改为 cin.get(ch)。

5.2.4　陷阱：循环的常见问题

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 5-5】　死循环示例。

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

```
        (a)                          (b)                          (c)
1  int x=0;                  int x=0;                  int x=0;
2  while(x<100)              while(x<100){             while(0=<x<100){
3    cout<< x <<" ";           cout<< x <<" ";           cout<< x <<" ";
4    x++;                        if(x%5==0)              x++;
5    if(x%5==0)                    cout<<endl;           if(x%5==0)
6      cout<<endl;             }                           cout<<endl;
7                                                        }
```

说明：

(1) 程序(a)中 while 后面没有花括号，所以循环体语句只有一条：“cout<<x<<" ";”，执行完毕后就继续检查 while 圆括号中的条件 x＜100，永远为真，因此程序陷入死循环。要将第 3～6 行的代码用一对花括号括起来，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 3 行后面增加一条“x＋＋;”语句。

(3) 程序(c)的第 2 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。

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,即满足条件时跳出所属的循环语句。

【例 5-6】 自定义函数,利用迭代公式求 x 的立方根。

求 $\sqrt[3]{x}$ 的公式为:

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

精度要求:

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

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 x_1 和 x_2 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 (true) {                            //也可以写为 while(1)
        x2 = (2 * x1 + x / (x1 * x1)) / 3;
        if (fabs ((x2 - x1) / x1) < 1e-6)     //满足条件则提前结束循环
        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 函数定义修改为：

```
double Cbrt(double x){
    double x1,x2=1;
    do{
        x1 = x2;
        x2 = (2 * x1 + x / (x1 * x1)) / 3;
    }while (fabs ((x2 - x1) / x1) > 1e-6);
    return x2;
}
```

【例 5-7】 自定义函数，判断一个整数是否为素数。

素数又称为质数，指除了 1 和它本身以外不再有其他因数的自然数。显然，设计一个循环变量，值在 $2\sim n-1$，判断变量是否整除 n。若能整除，说明 n 不是素数，结束判断；如果所有的数都不能整除，说明 n 是素数。

从效率出发，应尽可能减少循环重复的次数，判断一个数是否为素数，只要能找到一个因子就可以证明该数不是素数。而任何一个非素数（合数）一定是两个数相乘，至少有一个数会小于该数的平方根，因此循环变量可以缩减为 $2\sim\sqrt{n}$（注意，一定要包含边界）。进一步优化看，任何偶数都不可能是素数，因此循环变量可以从 3 开始直到 $\sqrt{n}$，每次增 2。自定义函数 isPrime 的参考代码如下：

```
bool isPrime(int n){                  //自定义函数判断 n 是否为素数
    int i, k = sqrt(n);
    for(i = 3; i <= k; i+=2)          //注意循环条件不要写 i<k
        if (n % i == 0)               //找到 n 的因子，就中止循环，否则继续循环
            break;
    if (i <= k)                       //循环条件为真，一定是执行了 break 提前中止循环
          return false;               //n 不是素数
```

```
    else                          //循环条件为假,没有找到 n 的因子
        return true;              //n 是素数
}
```

调用函数示例:

```
bool flag =  isPrime(15);         //flag 值为 false
```

说明:

(1) 在 for 语句中,有一条简单的 if 语句。如果找到 n 的因子,就说明 n 不是素数,结束循环;如果 i 不是 n 的因子,什么也不做,继续下一次循环,i+=2,然后再判断 i<=k 的循环条件,重复循环体语句。注意 for 循环中的 if 不要加 else。

(2) 循环结束后,检查循环的两个出口。一个是通过 break 提前结束了循环,循环条件 i<=k 为真,说明不是素数;另一个是循环体中的 if 条件一直是假,完成了全部的循环后退出,此时循环条件 i<=k 为假,说明是素数。

(3) 求素数的算法设计思想和代码框架是 break 语句的经典用法,循环体中用 if 判断某种情况下中止循环,循环语句后面利用 if 语句判断循环的出口,进而决定执行哪个分支。

(4) 在本例中,for 语句中的 if 条件为真时,已经明确了 n 不是素数,也可以直接用 return 返回 false 值到函数调用点。循环语句结束后说明 n 是素数,再用 return 返回 true 值。一个函数内部可以有多条 return 语句,遇到一条 return 语句就结束函数。

例如,isPrime 函数定义可改写为:

```
bool isPrime(int n){
    int i, k = sqrt(n);
    for(i = 3; i <= k; i+= 2)
        if (n % i == 0)
            return false;          //n 不是素数
    //循环结束,没有找到 n 的因子
    return true;                   //n 是素数
}
```

注意: return true;语句是循环结束后的语句,不要在前面加 else。

* 5.3.2 continue 语句退出迭代

continue 语句只能在循环体语句中使用,用来结束循环的本次迭代。在循环中遇到 continue 语句时,将跳过循环体内余下的语句,开始下一次迭代。

【例 5-8】 显示前 20 个能被 5 整除的正整数。

```
1   #include<iostream>
2   using namespace std;
3   int main(){
4       int x = 0, cnt = 0;
5       while (cnt<20){
6           x++;
7           if (x%5 != 0)
8               continue;
9           cout << x << " ";
```