

第5章

循环结构

在实际应用中,经常会遇到一些需要重复处理的问题,在 C 语言中,这类问题可以通过循环语句解决。例如,在机器博弈中,对弈双方反复交替落子、棋盘的输出等,都需要用到循环结构。C 语言提供了 3 种循环语句,分别是 while 语句、do...while 语句和 for 语句。

5.1 引例



扫一扫

例 5.1 假设一辆公交车从始发站出发,中间经过 19 站到达终点站。编写程序,计算一辆公交车从始发站到终点站总共的上车人数,每一站的上车人数可由随机函数产生。

【分析】

(1) 公交车从始发站到终点站总共有 20 站可能有乘客上车。从实际出发,可以假设每一站的上车人数为 0~10,即在程序中控

制随机数的产生范围是[0,10]。
(2) 每一站进行的都是类似的操作,即产生随机上车乘客人数,并将人数累加。相同或相似操作的反复执行,可以使用循环语句实现。例 5.1 程序流程图如图 5-1 所示。

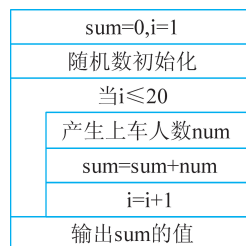


图 5-1 例 5.1 程序流程图

```
01 #include<stdio.h>
02 #include<stdlib.h>
03 #include<time.h>
04 int main(void)
05 {
06     int i=1,sum=0;           //i 变量控制重复次数,初始乘客人数 sum 为 0
07     int num;                 //每一站上车的人数
08     srand((unsigned int)time(NULL)); //初始化随机数发生器
09     while(i<=20)
```

```
10  {
11      num = rand() % 11;        //产生一个 0~ 10 的随机数作为每站上车人数
12      sum = sum + num;          //计算累加和
13      i++;
14  }
15  printf("sum=%d\n", sum);
16  return 0;
17 }
```

程序运行结果如下：

```
sum= 99
```

【注意】 本例中设定公交车始发时车上没有乘客,因此 sum 变量的初始值赋值为 0。对于累加运算,求和变量赋初值为 0 的情况较为多见。

5.2 3 种循环语句

C 语言中的循环语句分为如下两种类型。

(1) 当型循环。该类型循环首先判断条件是否为真,为真时反复执行相应的语句,否则结束循环。while 语句和 for 语句构成的循环属于当型循环。

(2) 直到型循环。该类型循环首先进入循环体,执行相应的语句,然后再判断条件是否为真,为真时反复执行语句,否则结束循环。do...while 语句构成的循环属于直到型循环。

循环结构中反复执行的语句称为循环体。循环体可以只有一条语句,也可以是复合语句。

5.2.1 while 语句

while 语句的一般形式如下：

while(表达式) 语句

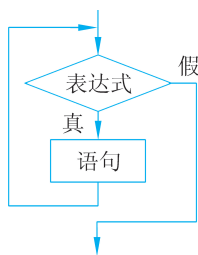


图 5-2 while 语句的流程图

while 语句的流程图如图 5-2 所示,其执行过程如下。

- (1) 判断表达式是否为真；
- (2) 如果表达式为真,那么执行循环体语句,返回步骤(1),否则转步骤(3)；
- (3) 结束循环,执行循环语句后面的语句。

5.2.2 do...while 语句

do...while 语句的一般形式如下：

```
do
    语句
while(表达式);
```

do...while 语句的流程图如图 5-3 所示,执行过程如下。

- (1) 执行循环体语句;
- (2) 判断表达式是否为真,如果表达式为真,则返回步骤(1),否则转步骤(3);
- (3) 结束循环,执行循环语句后面的语句。

可见,while 语句在执行时先判断条件,当条件为真时执行循环体语句,属于当型循环;而 do...while 语句则先执行循环体语句,后判断条件,属于直到型循环。

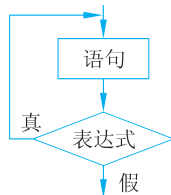


图 5-3 do...while 语句的流程图

例 5.2 天天向上的力量: 分别用 while 语句和 do...while 语句求 x^n 。

【分析】

(1) 采用输入函数分别为变量 x 和 n 输入一个确定的值;

(2) 可以定义一个变量 s 保存最终计算结果,初值为 1。再令一个变量 i 按照每次增 1 的规律从 1 变到 n ,控制循环执行 n 次,每次将一个 x 的值与 s 相乘,最终求得 x^n 。

例 5.2 方法 1 程序流程图如图 5-4 所示。

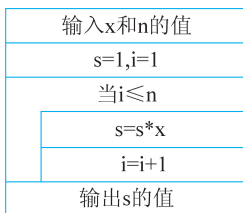


图 5-4 例 5.2 方法 1 程序流程图

方法 1:用 while 语句实现。

```

01 #include<stdio.h>
02 int main(void)
03 {
04     int i,n;
05     float x,s;
06     printf("Enter x,n:");
07     scanf("%f,%d",&x,&n);
08     i=1;
09     s=1; //累乘的初始值为 1
10     while(i<=n)
11     {
12         s=s*x; //累乘运算
13         i++;
14     }
15     printf("s=%.4f\n",s);
16     return 0;
17 }
  
```

程序运行结果如下:

```

Enter x,n: 1.01, 365 ↵
s=37.7833
  
```

方法 2:用 do...while 语句实现。

```

01 #include<stdio.h>
02 int main(void)
03 {
04     int i,n;
05     float x,s;
06     printf("Enter x,n:");
07     scanf("%f,%d",&x,&n);
08     i=1;
09     s=1;
10     do
11     {
12         s=s*x;
13         i++;
14     }while(i<=n);
15     printf("s=%.4f\n",s);
16     return 0;
17 }
  
```

程序运行结果如下:

```

Enter x,n: 0.99, 365 ↵
s=0.0255
  
```

短跑运动员苏炳添用“一厘米”的手势,提醒自己“进步一点点就好”。观察例 5.2 的运行结果,也体现了积少成多的道理。如果每天收获一点点,365 天以后,就有一个大的飞跃;如果每天放弃一点点,365 天以后,就会落后很多。学如逆水行舟,不进则退,希望你也能够日积月累、天天向上!

【说明】

- (1) 计算累乘时要注意结果变量的初始值,忘记赋值或赋值为 0 都将导致程序错误。
- (2) do...while 语句结束的分号切勿省略,否则会导致语法错误。
- (3) 一般情况下,while 和 do...while 语句能够互换使用,解决同一问题时循环体部分是相同的,如例 5.2。但如果 while 后面的表达式一开始就为假,两种循环的执行结果则不同。例如,下面两段程序是不等价的。

```

1  x=6;
2  sum=0;
3  while(x<=5)
4  {
5      sum=sum+x;
6      x++;
7  }
8  printf("%d\n",sum);

```

```

1  x=6;
2  sum=0;
3  do
4  {
5      sum=sum+x;
6      x++;
7  } while(x<=5);
8  printf("%d\n",sum);

```

在上面两段程序中,while 循环由于一开始表达式值为假,循环体没有被执行,输出 sum 的值为初始值 0;do...while 循环由于先执行循环体语句,后判断条件,因此,执行了一次循环体语句后才结束循环,输出 sum 的值为 6。



扫一扫

5.2.3 for 语句

for 语句的一般形式如下:

for(表达式 1; 表达式 2; 表达式 3) 语句

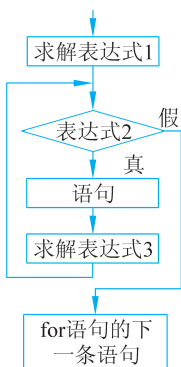


图 5-5 for 语句的流程图

for 语句的流程图如图 5-5 所示,其执行过程如下。

- (1) 求解表达式 1;
- (2) 判断表达式 2;
- (3) 如果表达式 2 为真,那么执行循环体语句,转步骤(4),否则转步骤(5);
- (4) 求解表达式 3,返回步骤(2);
- (5) 结束循环,执行循环语句后面的语句。

可见,for 语句与 while 语句类似,也是在执行时先判断条件,当条件为真时才执行循环体语句,属于当型循环。

例 5.3 用 for 语句计算 x^n 。

```

01  #include<stdio.h>
02  int main(void)
03  {
04      int i,n;
05      float x,s;
06      printf("Enter x,n:");
07      scanf("%f,%d",&x,&n);
08      s=1;
09      for(i=1;i<=n;i++)

```

```

10      s=s * x;
11      printf("s=% .4f\n",s);
12      return 0;
13  }

```

程序运行结果如下：

```

Enter x,n:1.01,730↵
s=1427.5784

```

【说明】

(1) 从例 5.3 可见,for 语句后面可以包含多个表达式,使程序更为简洁。一般情况下,上述 3 种循环语句可以互相替代。

(2) 如果不是循环体为空,不要在 for 语句和 while 语句的后面随意添加空语句,即分号“;”,否则会使原本的循环体语句失去应有的作用,甚至导致死循环。例如:

```

i=1;
while(i<=n);
{
    s=s * x;
    i++;
}

```

```

for(i=1;i<=n;i++);
    s=s * x;

```

上述两个程序段均在循环表达式的右侧添加了分号,因此,两个循环的循环体都变成了空语句,即循环体什么都不做,程序出现了逻辑错误。当 n 的值大于或等于 1 时,while 循环语句将会变成死循环。

【思考】 若要参照例 5.2 或例 5.3 计算 $n!$,该如何修改程序?

5.3

计数循环

在程序中,循环次数已知的循环称为计数控制的循环。例如,查找所有的水仙花数、求阶乘等问题一般都采用计数循环求解。习惯上,用 for 语句实现计数控制的循环可使程序更为简洁。

5.3.1 计数循环的基本应用

例 5.4 计算 $1\sim m$ 所有奇数的和。

【分析】

(1) 可以定义一个循环变量 i ,使其从 1 变到 m 控制累加项的范围;

(2) i 的值每次递增 2,可以计算出下一个累加项的值。

例 5.4 程序流程图如图 5-6 所示。

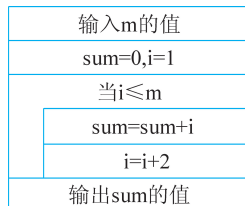


图 5-6 例 5.4 程序流程图

```

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

```

```

03 {
04     int i,m;
05     long sum;
06     printf("Enter m:");
07     scanf("%d",&m);
08     for(i=1,sum=0;i<=m;i+=2) //表达式 1 中应用了逗号表达式
09         sum=sum+i;
10     printf("sum=%ld\n",sum); //长整型变量输出用 ld 格式符
11     return 0;
12 }

```

程序运行结果如下：

```

Enter m:5↵
sum=9

```

【说明】

(1) for 语句后面的表达式 1 和表达式 3 既可以是简单的表达式,也可以是逗号表达式,即包含多个简单表达式,中间用逗号分隔。例 5.4 中,for 语句的表达式 1 就是逗号表达式。虽然逗号表达式的应用使程序更为简洁,但过多地使用逗号表达式会使程序的可读性变差。因此,尽量不要把与循环控制无关的语句写到 for 语句的表达式中。

(2) for 语句中的 3 个表达式均可根据需要省略不写,但间隔的分号不能省略。当 for 语句的表达式 2 省略时,循环条件按其值为真进行认定。例如,例 5.4 中的 for 语句写成如下形式也是合法的。

```

for(i=1,sum=0;i<=m;)
{
    sum=sum+i;
    i+=2;
}

```

```

i=1;
sum=0;
for(;i<=m;i+=2)
    sum=sum+i;

```

例 5.5 完善例 4.5,输出所有的水仙花数。

```

01 #include<stdio.h>
02 int main(void)
03 {
04     int i,x,y,z,c;
05     for(i=999;i>=100;i--) //在所有三位数中查找
06     {
07         x=(int)(i/100);
08         y=(int)(i%100/10);
09         z=i%10;
10         c=x*x*x+y*y*y+z*z*z;
11         if(c==i) printf("%-4d",i); //输出水仙花数,占 4 列宽,右侧补空格
12     }
13     return 0;
14 }

```

程序运行结果如下：

```

407 371 370 153

```

5.3.2 蒙特卡洛方法求 π 的近似值

蒙特卡洛(Monte Carlo)方法,又称随机抽样或统计试验方法。当所要求解的问题是某种事件出现的概率,或某随机变量的期望值时,可以通过某种“试验”的方法求解。

例 5.6 编写程序,用蒙特卡洛方法求圆周率 π 的近似值。

【分析】

(1) 用蒙特卡洛方法求解 π 近似值的思路:构造单位正方形及其内切单位圆,然后随机向单位正方形内抛洒大量点数,判断每个点是否在圆内,将圆内与正方形内的离散点数比模拟为面积比,计算 π 的值。四分之一单位圆与单位正方形随机抛点的情况如图 5-7 所示。

(2) 圆的面积/正方形的面积 $= \pi r^2 / (2r)^2 = \pi/4$,当抛洒的点数足够多时,将圆和正方形的面积比用它们的点数比替代,此时,圆内点数/正方形内点数 $= \pi/4$,则 $\pi = 4 \times (\text{圆内点数} / \text{正方形内点数})$ 。例 5.6 程序流程图如图 5-8 所示。

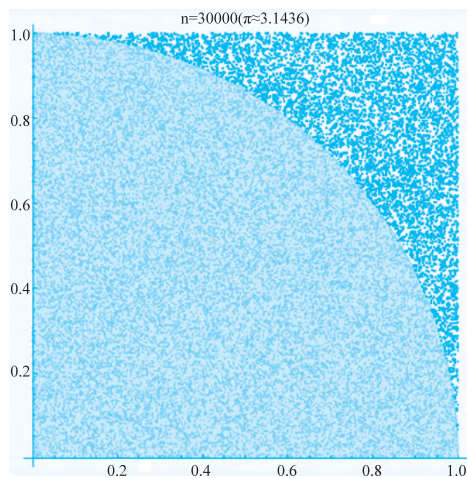


图 5-7 蒙特卡洛法求 π 近似值模拟图

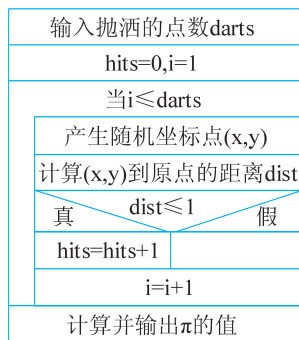


图 5-8 例 5.6 程序流程图

```
01 #include<stdio.h>
02 #include<math.h>
03 #include<time.h>
04 #include<stdlib.h>
05 int main(void)
06 {
07     long darts,i;
08     double x,y,dist,pi,hits;
09     srand((unsigned int)time(NULL)); //初始化随机数发生器
10     scanf("%ld",&darts); //输入抛洒的点数
11     hits=0.0; //落在圆内的点数
12     for(i=1;i<=darts;i++)
13     {
14         x=1.0 * rand() / RAND_MAX; //产生[0,1]区间随机数字 x 坐标
15         y=1.0 * rand() / RAND_MAX; //产生[0,1]区间随机数字 y 坐标
16         dist=sqrt(x * x + y * y); //计算该点到原点的距离
17         if(dist<=1.0) //如果距离小于或等于 1,则表示在圆内
18             hits=hits + 1; //圆内点的个数加 1
19     }
20     pi=4.0 * hits / darts;
21     printf("pi≈%.4f\n",pi);
22     return 0;
23 }
```

```
19     }
20     pi=4 * (hits/darts);           //计算  $\pi$  的值
21     printf("%f\n",pi);
22     return 0;
23 }
```

程序运行结果 1 如下：

```
30000✓
3.145067
```

程序运行结果 2 如下：

```
500000✓
3.140552
```

【思考】 上述程序中, π 值的精确程度与什么直接相关?

【说明】 蒙特卡洛方法是机器博弈中的基础算法,应用于爱恩斯坦棋、德州扑克、五子棋、不围棋等多项棋牌游戏。

5.3.3 井字棋随机落子

例 5.7 编写程序,在空的井字棋棋盘上随机落一个棋子,并打印棋盘当前状态。

【分析】

(1) 按行访问井字棋棋盘,将其看作线性排列的 9 个位置,并依次编号为 1~9。

(2) 在 1~9 产生随机数作为落子位置,输出棋盘时在空白位置输出“+”,在落子位置输出“●”或“○”,并进行换行控制。

```
01 #include<stdio.h>
02 #include<stdlib.h>
03 #include<time.h>
04 int main(void)
05 {
06     int i,position;
07     srand((unsigned int)time(NULL));
08     position=rand()%9+1;           //产生落子位置
09     for(i=1;i<=9;i++)
10     {
11         if(position==i)
12             printf("●");           //落子
13         else
14             printf("+");
15         if(i%3==0) printf("\n");    //控制换行
16     }
17     return 0;
18 }
```

程序运行结果如下：

```
+++
+++
++●
```

【说明】 在实际的博弈程序中,井字棋棋盘一般需要采用数组(第6章介绍)等方式保存起来,以便于落子的合法性判断以及着法生成等其他博弈算法的应用。

5.4

条件循环

循环次数未知时,一般用一个条件控制循环体是否执行,这种类型的循环称为条件控制的循环。该类循环采用 while 语句或 do...while 语句实现更为方便。

5.4.1 条件循环的基本应用

例 5.8 验证“角古猜想”。它是指对于一个自然数,若该数为偶数,则除以 2;若该数为奇数,则乘以 3 再加 1,将得到的数再重复按该规则运算,最终可得到 1。

【分析】 对于任意输入的自然数,采用 if 语句进行奇偶性的判断,然后按照相应的运算规则反复进行运算。由于循环次数未知,采用 while 语句或 do...while 语句比较方便。例 5.8 程序流程图如图 5-9 所示。

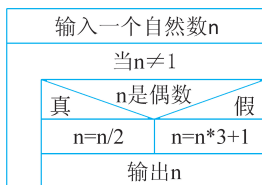


图 5-9 例 5.8 程序流程图

```
01 #include<stdio.h>
02 int main(void)
03 {
04     int n,i;
05     printf("Please enter a number:");
06     scanf("%d",&n);
07     while(n!=1)           //当 n 值未到达 1 时,重复执行循环体语句
08     {
09         if(n%2==0)        //n 为偶数时
10             n=n/2;
11         else              //n 为奇数时
12             n=n*3+1;
13         printf("%d ",n);  //输出验证过程
14     }
15     return 0;
16 }
```

程序运行结果如下:

```
Please enter a number:20✓
10 5 16 8 4 2 1
```

例 5.9 继续完善例 4.7,随机产生一个 1~50 的整数由用户去猜,直到猜对为止。猜测过程中给出相应提示信息,若猜测数与随机数相等,显示猜中;若猜测数小于随机数,显示猜小了;若猜测数大于随机数,显示猜大了。例 5.9 程序流程图如图 5-10 所示。



图 5-10 例 5.9 程序流程图

```

01 #include<stdio.h>
02 #include<stdlib.h>
03 #include<time.h>
04 int main(void)
05 {
06     int magic,guess;
07     srand((unsigned int)time(NULL));
08     magic =rand() %50 +1;
09     do
10     {
11         printf("Please guess a magic number:");
12         scanf("%d",&guess);    //用户输入猜测的数
13         if(guess==magic) printf("猜中!\n");
14         else if(guess>magic) printf("猜大了!\n");
15         else printf("猜小了! \n");
16     }while(guess!=magic);    //循环直到猜中为止
17     return 0;
18 }

```

程序运行结果如下:

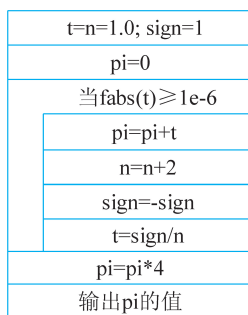


图 5-11 例 5.10 程序流程图

```

Please guess a magic number:25 ✓
猜小了!
Please guess a magic number:35 ✓
猜小了!
Please guess a magic number:40 ✓
猜大了!
Please guess a magic number:38 ✓
猜中!

```

例 5.10 根据近似公式: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots +$

$(-1)^{n-1} \frac{1}{2n-1}$, 求圆周率 π 的值。要求最后一项的绝对值

小于 10^{-6} 时结束求解, 例 5.10 程序流程图如图 5-11 所示。

```

01 #include<stdio.h>
02 #include<math.h>
03 int main(void)
04 {
05     double t=1.0,pi=0.0,n=1.0;
06     int sign=1;
07     while(fabs(t)>=1e-6)
08     {
09         pi=pi+t;
10         n=n+2;
11         sign=-sign; //控制正负号
12         t=sign/n;   //计算累加项
13     }
14     pi=pi*4;
15     printf("pi=%10.8lf\n",pi);
16     return 0;
17 }

```