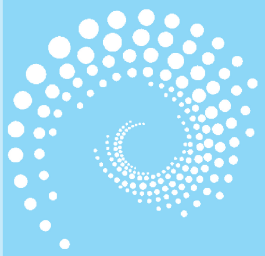
第5章

循环结构

CHAPTER 5



5.1 循环结构与循环语句

程序设计与计算思维有非常密切的关系，计算思维的特点是抽象和自动化。计算思维中的抽象并非指抽象问题的因果关系，而是指抽象问题的计算过程，利用计算机强大的计算能力自动化求解。因此，计算思维是基于计算机的思维，或者说计算思维是以计算机为工具完成问题求解的思维模式。例如，在序列数求和的问题中，数学家总结出了序列数求和的公式：$\frac{(a_1+a_n)n}{2}$，这是典型的逻辑思维；而计算机可以通过循环结构实现序列数的累加得到求和结果，这是典型的计算思维。可见，计算思维通过模拟运算过程利用计算机完成它的计算。一旦能够抽象出问题的计算过程，我们就能利用程序执行这样的过程，获得问题的计算结果。因此，程序设计是实现计算思维的主要手段，或者说程序设计是将计算思维变成现实的手段。

前面章节用计算思维编写的顺序结构和选择结构的程序只涉及一次操作，而在实际的问题中，为了模拟问题的计算过程，往往需要进行多次重复操作，如序列数求和、阶乘计算、方程的迭代求解等。程序能否允许用户连续进行多次重复操作呢？

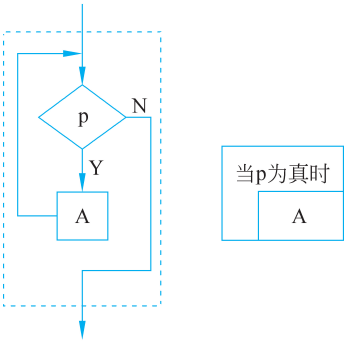
循环是重复执行某些操作的一种程序结构。若已知要重复操作或计算的次数，则这种循环称为计数控制的循环（Counter Controlled Loop）。若未知要重复操作或计算的次数，则需要通过一个条件控制后续重复操作是否继续进行，则这种循环称为条件控制的循环（Condition Controlled Loop）。二者都需要用循环结构来实现。

顺序结构、选择结构和循环结构是结构化程序设计的三种基本结构。根据结构化程序设计的思想,任何复杂问题都可以用这三种基本结构编程实现,它们是复杂程序设计的基础。

第 1 章已经介绍过,循环结构有两种类型:

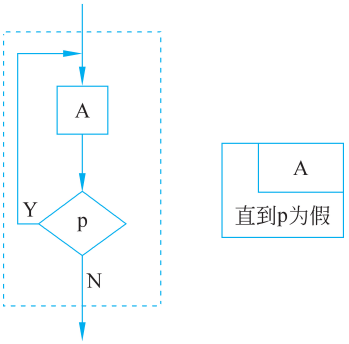
(1) 当型循环结构,当条件 p 为真(成立,用 Y 或 T 表示)时,重复执行 A 操作,直到条件 p 为假(不成立,用 N 或 F 表示)时结束循环,如图 5-1 所示。

(2) 直到型循环结构,先执行 A 操作,再判断条件 p 是否为真(成立),若条件 p 为真(成立),则重复执行 A 操作,直到条件 p 为假(不成立)时结束循环,如图 5-2 所示。



(a) 流程图　　(b) N-S图

图 5-1　当型循环结构



(a) 流程图　　(b) N-S图

图 5-2　直到型循环结构

C 语言提供了 while、do-while、for 三种循环语句来实现循环结构。这些循环语句在循环条件为真的情况下,重复执行一个称为循环体的语句序列。

1. while 语句

在 C 语言的所有循环语句中,while 语句是最简单、最基本的循环语句,它属于当型循环结构。其一般格式为:

```
while(表达式)
{
    循环体
}
```

其中,表达式是循环条件,是判断循环体是否执行的依据。while 语句的执行过程如下:

① 计算表达式的值。

② 若表达式的值为真,则执行循环体,并返回步骤①。

③ 若表达式的值为假,则退出循环,执行 while 语句后面的语句。

为了确保程序的逻辑正确及易于维护,建议将循环体放在一对花括号中,即使循环体只有一条语句也要如此。因为当循环体多于一条语句时,如果忘记写上花括号,那么程序只将 while 后面的第一条语句当作循环体,其他语句没有当作循环体,从而导致逻辑错误。

注意:当表达式的值为假时,while 循环将终止。同时,若一开始表达式的值就为假,那么循环体可能一次也不执行。若表达式的值始终为真,while 循环将无法终止,这就构成了无限循环(又称死循环),例如:

```
while(1){ … }
```

这时除非循环体中包含跳出循环的控制语句(如 break、goto、return 等),或者调用了导致程序终止的函数,否则无限循环的 while 语句将永远执行下去。

2. do-while 语句

do-while 语句属于直到型循环结构,其一般格式为:

```
do
{
    循环体
}while(表达式);
```

其中,表达式是循环控制表达式,它在循环体执行后才进行测试。do-while 语句的执行过程如下:

① 执行循环体;

② 计算表达式的值;

③ 若表达式的值为真,则返回步骤①;

④ 若表达式的值为假,则退出循环,执行 do-while 语句后面的语句。

因此,do-while 语句是先执行循环,再判断表达式的值为真或假,若为真,则继续执行循环体,否则退出循环。这意味着 do-while 循环的循环体至少将被执行一次。

通常,while 语句可以被 do-while 语句等价替换。但要注意,当第 1 次测试循环条件(表达式的值)为假时,while 语句和 do-while 语句是不等价的。例如:

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

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

while 语句先判断后执行,当 i 的值不满足循环条件时,循环体一次也不执行,因此什么都没有打印。而在 do-while 语句中,它的执行过程是先执行一次循环体,然后判断变量 i 的值是否满足循环条件。因此,无论 i 的值是否满足循环条件,都要先执行一次循环体后再进行判断,所以得到打印结果 i*i=100。

注意: *在 while 语句或 do-while 语句中,都要有使循环控制变量变化的语句,不然可能导致死循环。*

3. for 语句

for 语句是使用最多的循环语句,它的使用方式灵活多样,属于当型循环结构。其一般格式为:

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

其中,表达式 1 的作用是初始化循环控制变量,即为循环控制变量赋初值,表达式 1 只执行一次。表达式 2 是循环条件,其作用是控制循环的终止,只要表达式 2 的值为真(不为 0),那么将继续执行循环。表达式 3 是在每次循环的最后被执行的一个操作,它决定循环控制变量如何变化,表达式 3 一般使用自增或自减表达式。在每次执行循环体之前,都要对表

达式 2 进行测试。每次执行完循环体后,都要执行一次表达式 3。

注意:如何对循环控制变量进行控制,决定了循环执行的次数,如果在循环体内再次改变循环控制变量的值,那么将改变循环正常的执行次数。

for 语句与 while 语句可以相互替代,与 for 语句语义等价的 while 语句的一般格式为:

```
初始化表达式 1;
while(表达式 2)
{
    语句序列
    表达式 3;
}
```

注意:for 语句的三个表达式之间用分号分隔,有且仅有两个分号。一般,表达式 2 不能省略,若省略,则表示循环条件一直为真。如果在 for 语句之前已初始化循环控制变量,则可省略表达式 1;如果已在循环体内改变了循环控制变量,则可省略表达式 3。例如,下面两种 for 语句形式与规范的 for 语句形式在语义上是一致的。

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

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

【例 5.1】 编写程序,从键盘输入正整数 n,计算并输出表达式 1+2+…+n 的值。

【解题思路】 本例与例 1.2 解题思路基本一致,不同的是 n 的值未知,需要定义三个变量来完成数据的表示。这种累加求和的方法可以通过自然语言或如图 5-3 所示的流程图来描述。

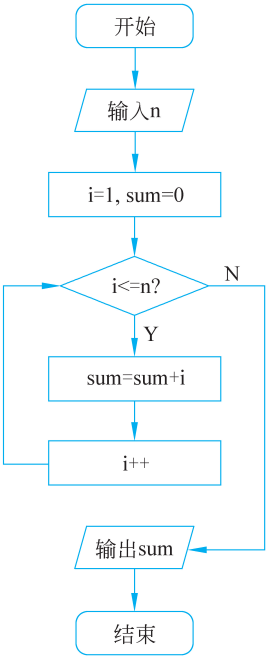


图 5-3 例 5.1 的流程图

step1 从键盘输入 n；
step2 初始化累加和变量，sum＝0；
step3 初始化循环控制变量，i＝1；
step4 若循环控制变量不大于 n，则重复执行 step5～step6，否则，转去执行 step7；
step5 进行累加运算，sum＝sum＋i；
step6 循环控制变量加 1，i＝i＋1，且转 step4；
step7 打印累加和变量 sum。

本例的程序可以通过三种形式的循环来实现。

形式 1：for 语句编程实现。

```
#include<stdio.h>
int main()
{
    int sum,i,n;
    printf("Input n please:");
    scanf("%d",&n);
    sum=0;                                  /*累加和变量初始化为 0*/
    for(i=1;i<=n;i++)
        sum=sum+i;                          /*累加运算*/
    printf("sum=%d",sum);
}
```

形式 2：while 语句编程实现。

```
#include<stdio.h>
int main()
{
    int sum,i,n;
    printf("Input n please:");
    scanf("%d",&n);
    sum=0;                                  /*累加和变量初始化为 0*/
    i=1;
    while(i<=n)
    {
    sum=sum+i;                              /*累加运算*/
    i++;                                    /*循环控制变量加 1*/
    }
    printf("sum=%d",sum);
}
```

形式 3：do-while 语句编程实现。

```
#include<stdio.h>
int main()
{
    int sum,i,n;
    printf("Input n please:");
    scanf("%d",&n);
    sum=0;                                  /*累加和变量初始化为 0*/
    i=1;
    do{
    sum=sum+i;                              /*累加运算*/
    i++;                                    /*循环控制变量加 1*/
```

```
    }while(i<=n);
    printf("sum=%d",sum);
}
```

程序的运行结果如下:

```
Input n please: 5
sum=15
```

上述程序中的表达式 sum=sum+i 为什么能实现整数的累加计算呢?这是因为在 sum=sum+i 中,赋值运算符左、右两边的 sum 执行的操作不同,左侧的 sum 执行的是写操作,右侧的 sum 执行的是读操作。即先将右侧的 sum 值读出来,接着与 i 值相加,得到的和值再写入左侧的 sum 中。每执行一次循环,左、右两侧的 sum 值均会发生变化。

注意:在上述程序中,不要忽略给变量 i 和 sum 赋初值,否则在有些编译器中,它们的值是不可预测的,可能导致结果不正确。

有时候,我们可能喜欢编写有多个初始表达式或多个循环控制变量的 for 语句。使用逗号运算符(comma operator)构成的逗号表达式(comma expression)作为 for 语句的表达式 1 或表达式 3 可以实现这些想法。逗号表达式的一般格式为:

```
表达式 1,表达式 2,…,表达式 n
```

逗号运算符的优先级在所有运算符中最低,且具有左结合性。因此,逗号表达式的计算过程为:先计算表达式 1 的值,然后计算表达式 2 的值,……,最后计算表达式 n 的值,并将表达式 n 的值作为整个逗号表达式的值。

通常,使用逗号表达式的目的并非是要得到和使用这个逗号表达式的值,而是要得到各个表达式的值,主要用在 for 语句中同时为多个变量赋初值或改变变量的值。例如,例 5.1 的程序可以采用逗号表达式实现:

```
#include<stdio.h>
int main()
{
    int sum,i,n;
    printf("Input n please:");
    scanf("%d",&n);
    for(sum=0, i=1, j=n;i<=j;i++, j--)
                                /*变量 sum、i、j 的初始化,变量 i、j 分别自增 1、自减 1*/
        sum=sum+i+j;            /*累加运算*/
    printf("sum=%d",sum);
}
```

在这个例子中,for 语句的表达式 1 和表达式 3 都是逗号表达式。当 n 为偶数时,循环累加操作是从等差数列的两端开始同时进行的,每次循环累加 i 和 j 的值,然后 i 的值自增 1,而 j 的值自减 1。这样,整个循环的循环次数将变为原来的一半。当 $i>j$ 时,退出循环。

使用循环结构时,经常出现以下问题。

(1) 循环体只包含一个分号,即一条空语句。空语句表示什么也不做,只表示语句的存在,常用于编写延时程序。下面是一个延时程序的例子:

```
for(i=0;i<10000; i++)
{
    ;
}
```

或者

```
for(i=0;i<10000; i++)
{  }
```

或者

```
for(i=0;i<10000; i++) ;
```

但是，最后一种形式的空语句可能会混淆 for 语句后边的语句，无法确定其是否为循环体。例如：

```
for(i=0;i<10000; i++) ;
{
    sum=sum + i;
}
```

不会执行 10000 次的累加运算，只会执行一次累加运算，因为这里使用了空语句，它相当于：

```
for(i=0;i<10000; i++)
{
    ;
}
sum=sum+i;
```

它表示循环体是一条空语句，什么也不做。只有在循环结束之后，才会执行一次累加运算，因此，分号放在 for 后面的做法容易造成累加错误。

(2) 分号用在 while 后面，可能产生无限循环。例如：

```
i=10000;
while(i>0) ;
{
    sum=sum+i;
    i--;
}
```

它相当于下面的语句：

```
i=10000;
while(i>0)
{
    ;
}
sum=sum+i;
i--;
```

由于循环体为空语句，循环控制变量 i 的值没有改变，导致 while 中的循环条件永远为真，从而变为无限循环。

(3) 在 if 语句圆括号后面放置分号，可能会导致逻辑错误。例如：

```
if(i==0) ;
    printf("Error: Division by zero. \n");
```

此时,无论 i 的值是否为 0,都会执行函数调用。因为分号放置在 if 语句圆括号后面,使得 if 语句变成一条空语句,printf()函数调用不在 if 语句内。

5.2 计数控制的循环

若已知循环次数,则这种循环称为计数控制的循环。for 语句是实现计数控制的循环最简便的形式。

【例 5.2】 编写程序,从键盘输入一个大于 3 的整数,判定它是否为素数。

【解题思路】 该问题的数学模型可描述为:$D=n$,其中 n 为键盘输入的任一大于 3 的整数,问题的解是判断 n 除以整数 i 的余数是否为 0(即 $n \bmod i=0$),i 的取值范围为 $2\sim\sqrt{n}$。若都不为 0 则 n 是素数,否则 n 不是素数。这是一个数值计算问题,求解不定方程,算法策略可以使用穷举循环。这是一个循环次数已知的问题,首先设计出一个循环,穷举 n 能否被 $2\sim\sqrt{n}$ 的任意一个整数 i 整除,若能则 n 不是素数,结束循环,此时 i 必然小于 $\sqrt{n}$;如果 n 不能被 i 整除,则 n 是素数,结束循环。具体算法描述如下:

step1 输入 n;
step2 循环控制变量终值赋值 $k=\sqrt{n}$;
step3 除数 i 赋初值 $i=2$;
step4 若循环次数不大于 k,则反复执行 step5 和 step6,否则转向执行 step7;
step5 进行整除运算,若 n 能被 i 整除,则终止循环,转向执行 step7;
step6 除数 i 加 1,$i=i+1$,且转向执行 step4;
step7 打印 n 是否为素数的结论。

程序的流程图和 N-S 图分别如图 5-4 和图 5-5 所示。

程序如下:

```
#include<stdio.h>
#include<math.h>
int main()
{
    int n,i,k;
    printf("please enter a integer number:\n");
    scanf("%d",&n);
    k=sqrt(n);
    for(i=2;i<=k;i++)
    {
        if(n%i==0)
            break;
    }
    if(i<k+1)
        printf("%d is not a prime number.\n",n);
    else
        printf("%d is a prime number.\n",n);
    return 0;
}
```

程序的运行结果如下：

```
please enter a integer number:
17
17 is a prime number.
```

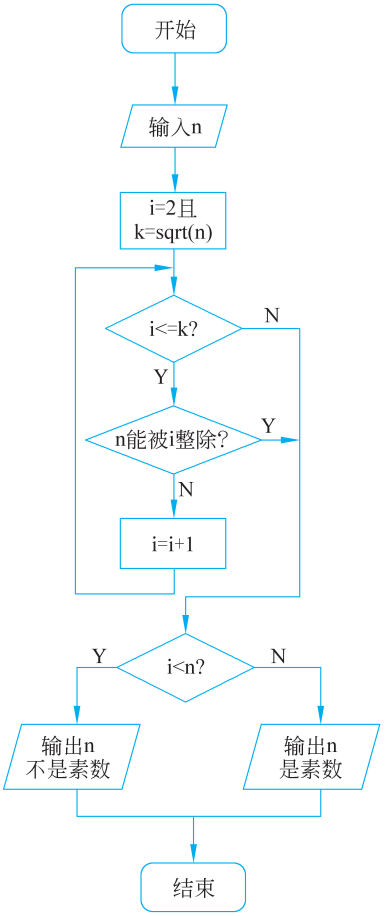


图 5-4　例 5.2 的流程图

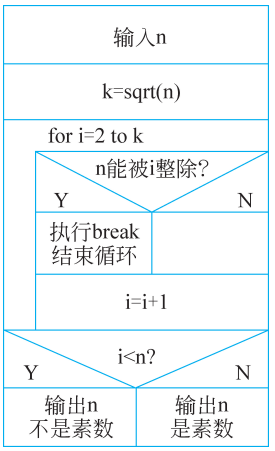


图 5-5　例 5.2 的 N-S 图

【例 5.3】　一年 365 天，每周有 5 个工作日，每个工作日进步 1%；每周 2 个休息日，每个休息日退步 1%。试问：这种工作日的力量结果如何呢？

【解题思路】　如果每天都进步 1%或退步 1%，那么可以用数学公式 1.01^{365} 或 0.99^{365} 来计算，这是一种数学思维的方法。本例的情况虽然复杂一点，但也可以找到相应的计算公式，采用数学思维的方法解决。现在本例不采用数学思维的方法，而是尝试向采用程序解决问题的计算思维转变。由于一年有 365 天，一周有 7 天，5 天是工作日，2 天是休息日，因此，我们让计算机模拟一年 365 天的过程，当处于工作日时，工作的力量向上增加；当处于休息日时，工作的力量向下减少。这样累计循环的结果就是一年 365 天的总结果。因此，我们可以不使用公式，而是将过程抽象并用程序模拟完成过程，即可得到需要的结果。

程序如下：

```
#include<stdio.h>
int main()
{
    int i;
    float dayup=1.0;
    float dayfactor=0.01;
    for(i=0;i<365;i++)
    {
        if(i%7==0||i%7==6) /* 如果余数为 0 或 6,则说明是休息日,否则就是工作日 */
        {
            dayup=dayup*(1-dayfactor);
        }
        else
        {
            dayup=dayup*(1+dayfactor);
        }
    }
    printf("向上:%.2f\n",dayup);
    return 0;
}
```

程序的运行结果如下:

```
dayup=4.63
```

程序通过 for 语句和 if-else 语句,最终获得的 dayup 就是满足问题要求的工作日的力量结果。

5.3 嵌套循环

【例 5.4】 编写程序,求出 100～200 的所有素数。

【解题思路】 该问题的数学模型可描述为:$D=\{a_i\}$,其中 a_i 为 100～200 的任一整数,问题的解是判断 a_i 对整数 i 的余数是否为 0(即 $a_i \bmod i=0$),若都不为 0,则 a_i 是素数,否则 a_i 不是素数,其中 i 的取值范围为 2～$\sqrt{a_i}$。这是一个数值计算问题,求解不定方程,算法策略可以使用穷举循环。

在例 5.2 的基础上,只需增加一个外循环,分别对 100～200 的全部整数一一进行判定即可,即使用嵌套的 for 循环。由于偶数不是素数,因此不必对偶数进行判定,只检查奇数即可,所以循环控制变量 n 从 101 开始,每次增 2。程序如下:

```
#include<stdio.h>
#include<math.h>
int main()
{
    int n,i,k,m=0;
    for(n=101;n<=200;n+=2)
    {
        k=sqrt(n);
        for(i=2;i<=k;i++)
        {
            if(n%i==0)
```