# 第5章

# 程序的循环结构

## 思考题

1. 计算 1＋2＋3＋…＋的值。

对于这个问题，读者可能首先想到的是如下的数学方法：

$1+2+3+\cdots+100=(1+100)+(2+99)+\cdots+(50+51)=101\times 50=5050$

但与之类似地，对于以下倒数的求和，却不易直接用该数学方法计算：

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{100}$$

其实，无论是计算 1＋2＋3＋…＋100，还是计算 1＋1/2＋1/3＋…＋1/100，二者都有一个共性：重复进行了若干次有规律的加法运算的操作。

又比如，计算 $n$ 的阶乘 $n!$，即计算 $1\times 2\times 3\times\cdots(n-1)\times n$，它只不过是把上述重复进行的若干次加法运算改成重复进行若干次乘法运算，求解过程的差异仅此而已。

2. 输入一个正整数 $n$，输出它的所有因子。

求一个正整数 $n$ 的所有因子，也就是在[1，$n$]区间上逐一用每一个整数去除 $n$。若能够整除，则该数就是 $n$ 的因子之一。它是一个重复进行整除测试的过程。

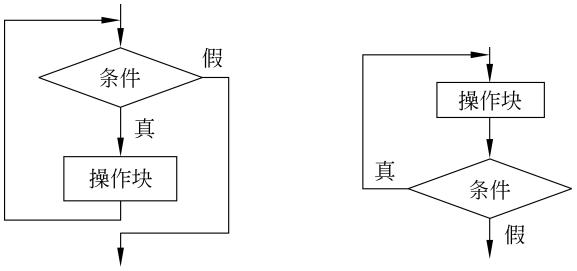
3. 利用下面的极限公式计算圆周率 π 的近似值：

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

计算机的运算速度快，最善于进行重复性的工作。在利用计算机解题时，往往可以把复杂的不容易理解的求解过程转换为易于理解的操作的多次重复，这就是循环。就像上面的计算 $n!$ 和计算 1＋1/2＋1/3＋…＋1/100，虽然我们不能像 1＋2＋3…＋100 那样找出一个简单的数学公式就能得出结果，但我们可以通过构造循环结构的程序，交给计算机快速实现重复操作而得出结果。上面的“思考题 2”和“思考题 3”，同样也是可以通过循环结构的计算机程序来求解问题。

# 5.1 程序的循环控制

在程序设计中，如果需要重复执行某些操作，就要用到循环结构。循环结构有两种形式：当型循环和直到型循环，其对应的程序执行流程如图 5-1 所示。它们都是按照给定的条件重复地执行某一个语句或语句组（复合语句），区别在于：当型循环是先判断循环条件是否满足，才确定是否开始循环操作；直到型循环则是先执行第一次循环操作，再按照循环条件确定是否执行后续的循环操作。




(a) 当型循环结构　　(b) 直到型循环结构

图 5-1　两种循环结构的执行流程图

循环结构是程序的 3 种基本控制结构中最复杂的一种。

对于循环结构的设计，首先要明确两个问题：

（1）哪些操作需要重复执行？

（2）这些操作在什么情况下重复执行？

这两个问题的解决分别对应循环体的设计和循环过程的设计。循环过程由 3 个要素组成：循环过程控制量的初值、循环条件、使循环趋于结束的循环过程控制量的“增值”。只有明确了这些问题，才能完整地设计出一个程序的循环结构，与此对应的程序执行流程如图 5-2 所示。

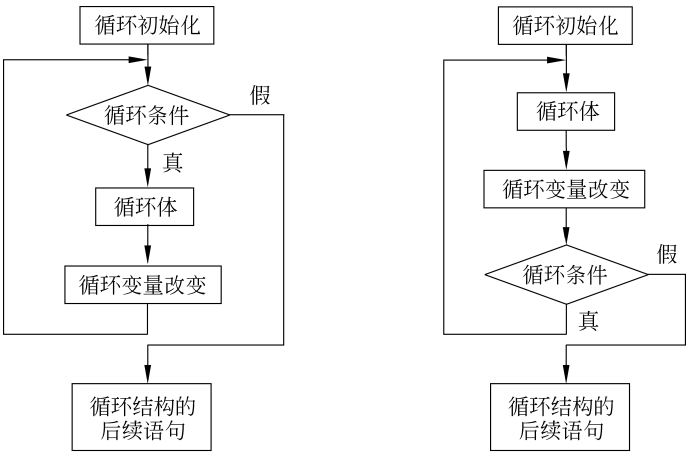
图 5-2 中标示的“循环变量”以及后续的同样表述，指的就是“循环过程控制量”。“循环初始化”则包括循环变量的初始化和循环体相关量的初始化。

例如，计算 1＋2＋3＋…＋99＋100 的值，可以这样设计求解过程：

首先设置一个累加结果变量 sum，其初值为 0，利用 sum＝sum＋i 这个赋值操作，让计算机重复进行加法运算和赋值操作。第一次加法运算时 i 的初值为 1，之后的每次加法运算的 i 值都比前一个 i 值多 1，即：每次加法运算和赋值操作后使 i＝i＋1，从而使 i 值依次取 1，2，3，…，99，100。通过重复 100 次这样的加法运算和赋值操作，即可得到最后的结果。

显然，在这个求解过程中，循环（重复操作）的主体就是 sum＝sum＋i 和 i＝i＋1。那么如何控制循环的进程，确保正确地进行了 100 次的重复操作呢？也就是说，如何设计循环变量的初值、循环条件和循环变量的“增值”？

可以增加一个循环变量 j，j 的初值为 1；每一次循环后使 j 值增 1，即循环变量 j 的“增

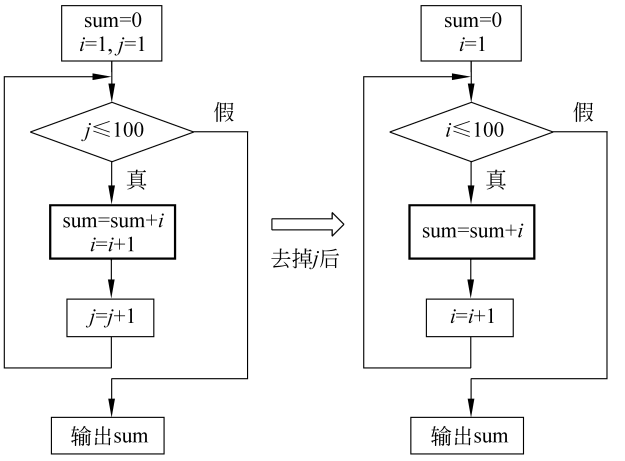



(a) 当型循环结构　　(b) 直到型循环结构

图 5-2　完整的循环结构流程图

值”方式是 j=j+1；当 j 值增加到 100 以后结束循环，即循环条件是 j<=100。

根据以上分析，不难画出图 5-3(a)所示的程序流程图，它是一个当型循环结构。




(a) 有单独的循环控制变量*j*　　(b) *i*起循环控制变量的作用

图 5-3　计算 1+2+3+…+99+100 的循环流程图

对图 5-3(a)所示的流程图进一步分析，可以发现：其中所设置的循环变量 j 与循环体中的 i，在循环过程中其实是完全同步一致的，因而可以将 i 和 j 合二为一，简化程序代码。简化后的程序流程如图 5-3(b)所示。

依照上述“计算 1+2+3+…+99+100”的算法逻辑，我们可以轻松地画出计算 1+1/2+1/3+…+1/100 和计算 n! 的程序流程图，分别如图 5-4 和图 5-5 所示。图 5-5 中的 f 是用于存放 n! 计算结果的变量。

C 语言提供了 while 语句、do-while 语句和 for 语句这 3 种循环语句来构造程序的循环结构。

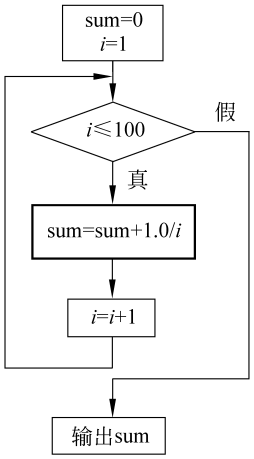



图 5-4　计算 1＋1/2＋1/3＋…＋1/99＋1/100 的流程图

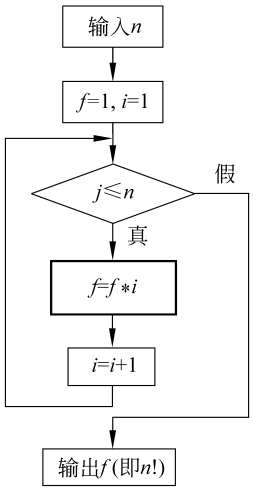



图 5-5　计算 n！的流程图

## 5.2　while 语句

while 语句的基本形式为：

```
while(表达式)
    循环体语句;
```

或

```
while(表达式)
{
    多条循环体语句;
}
```

while 语句的常用形式：

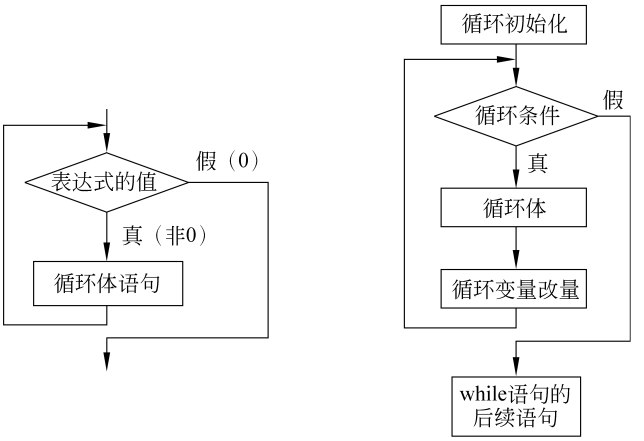
```
循环初始化;
while(循环条件)
{
    循环体;
    循环变量改变;
}
```

while 语句的流程如图 5-6 所示。

**说明：**

(1) while 语句构造的循环是一个当型循环，一般用于根据某个条件控制循环的情形。

(2) 在 while 语句的基本形式中，while 后面的表达式是作为循环条件用的。当表达式的值为真(非 0)时，就反复执行循环体语句；一旦表达式的值为假(0 值)，即结束循环，它的执行流程如图 5-6(a)所示。为了使循环能够结束，循环体语句中一般应含有使循环趋于结束的循环变量“增值”的语句。




(a) 基本形式　　(b) 常用形式

图 5-6　while 语句的流程图

(3) 如前所述，一个完整的循环结构包括两大部分：循环主体和循环过程控制。同样地，使用 while 语句构造一个完整的循环结构，也需要将这两个方面都构造完整。这正是图 5-6(b)所示的 while 语句常用形式所体现出来的循环结构的算法逻辑和程序流程。

(4) 为了明确标识循环体的范围，注意正确使用锯齿形书写格式，使循环体语句向右缩进对齐。

(5) 当循环体中的语句不止一条时，必须用{}将循环体语句组合成一条复合语句。即使循环体中的语句只有一条，也可以用{}将其括起来，以明确标识其循环体的性质，提高程序的可读性。

(6) 还需要注意的是：正常情况下，while(表达式)后面紧跟的是一个非空操作的循环体语句，而不能紧跟一个分号“;”。如果 while(表达式)后面紧跟的是一个分号，则表示循环体为空，同时该分号还表示 while 语句已结束，其后的语句已不是循环体语句，从而造成了空循环操作，并极有可能造成死循环。一旦程序运行出现死循环，一般可按 Ctrl＋Break 组合键以终止程序的运行。

**例 5.1**　使用 while 语句建立循环结构的程序，计算 1＋2＋3＋…＋99＋100 的值。

在本章开头，已经就本问题的求解过程做了详细分析和描述，它可以用迭代与递推的循环结构来求解，迭代关系式是 sum＝sum＋i，i 值依次取 1，2，3，…，99，100。按照 while 语句的语法格式，对比图 5-6(b)和图 5-3(b)所示的流程图，不难写出求解本问题的程序代码。

```
01    #include<stdio.h>
02    main()
03    { int i=1,sum=0;
04      while(i<=100)
05      {  sum=sum+i;
06         i++;
07      }
08      printf("sum=%d\n",sum);
09    }
```

程序运行结果：

```
sum=5050
```

**说明：**

(1) 当循环体中的语句不止一条时，必须用{}将循环体语句组合成一条复合语句。如果将本例中 while 语句循环体的{}去掉：

```
while(i<=100)                    while(i<=100)
{                      ⟹          sum=sum+i;
    sum=sum+i;          变成       i++;
    i++;
}
```

则 while 语句将变成死循环。因为此时的循环体语句只有“sum＝sum＋i;”，而“i＋＋;”已不属于循环体的语句，从而导致循环变量 i 不会出现任何变化，i 总是保持初值 1 不变，循环条件永远满足，这就造成了死循环(此时可按 Ctrl＋Break 组合键以中断程序)。

(2) 初学者还会常犯另一个错误：在 while(表达式)后面紧跟一个分号。例如：

```
while(i<=100)                    while(i<=100);
{                      ⟹        {
    sum=sum+i;          变成         sum=sum+i;
    i++;                             i++;
}                                }
```

此时，while(表达式)后面紧跟的分号“;”代表的就是循环体，只不过“它什么也没做”。之后的{}部分已不属于循环体。这样也导致了循环变量 i 不会出现任何变化，循环条件永远满足而造成死循环。

(3) 如果将本题循环迭代的 i 值取值方向反过来，即 i 值依次取 100、99、…、3、2、1，那么，作为循环变量的 i，其“增值”则是一个逐渐减值方向的“减值”，即 i＝i－1，同时循环条件要调整为 while(i＞0)。循环部分的代码如下：

```
01    int i=100,sum=0;
02    while(i>0)
03    { sum=sum+i;
04      i--;
05    }
```

**例 5.2** 使用 while 语句建立循环结构的程序，计算以下公式的值。

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{99}+\frac{1}{100}$$

本问题的求解方法与例 5.1 相同，它可以用迭代与递推的循环结构来求解，迭代关系式是 sum＝sum＋1.0/i，i 值依次取 1、2、3、…、99、100，程序流程如图 5-4 所示。

但需要注意的是：

(1) 存储累加和的变量 sum，必须定义成浮点数类型(double 型或 float 型)。

(2) 每次循环迭代进来的是一个分数项 1.0/i，要特别注意分数项的计算特点：分子和分母要以浮点数方式进行运算。因为分子和分母不能是两个整数相除，否则依据 C 语言整数除法的语法规则，分子和分母两个整数相除的结果只能保留整数部分。

要使分子和分母以浮点数方式进行运算，有两种解决办法：一是优先保障分子和分母

整型数据的性质不变，但在进行分数运算时，将分子和分母的任一方临时转换为浮点型数据来参与分数的运算；二是将分子和分母的任一方直接定义为浮点型变量，但这改变了分子和分母整型数据的性质。本例按第一种办法进行处理。

```
#include<stdio.h>
main()
{ double sum=0;
  int i=1;
  while(i<=100)
  {  sum=sum+1.0/i;        //或 sum=sum+1/(double)i;
     i++;
  }
  printf("sum=%f\n",sum);
}
```

程序运行结果：

```
sum=5.187378
```

**例 5.3** 输入一个正整数 n，输出它的所有因子。

**分析**：求一个正整数 n 的所有因子可以采用穷举法，对 1～n 的全部整数进行测试判断，凡是能够整除 n 的均为 n 的因子。

程序：

```
#include<stdio.h>
main()
{ int n,i=1;
  printf("请输入一个正整数: ");
  scanf("%d",&n);
  printf("它的因子是: ");
  while(i<=n)
  {  if(n%i==0) printf("%d, ",i);
     i++;
  }
}
```

例如，输入 18 得到的程序运行结果为：

```
请输入一个正整数: 18
它的因子是: 1, 2, 3, 6, 9, 18,
```

## 5.3 do-while 语句

do-while 语句的基本形式：

```
do
   循环体语句;
while(表达式);
```

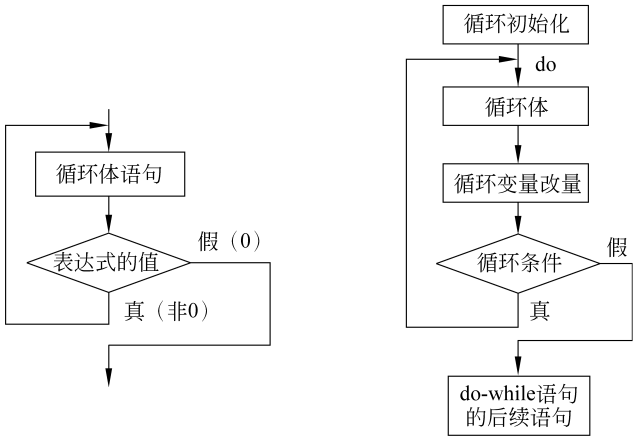
或

```
do
{
    多条循环体语句;
}while(表达式);
```

do-while 语句的常用形式：

```
循环初始化;
do
{
    循环体;
    修改循环变量;
}while(循环条件);
```

do-while 循环语句的流程如图 5-7 所示。




(a) 基本形式　　(b) 常用形式

图 5-7　do-while 语句的流程图

**说明：**

(1) do-while 语句构造的循环是一个直到型循环，一般用于根据某个条件控制循环。

(2) do-while 语句与 while 语句的区别：do-while 语句先执行一次循环体语句，然后再对循环条件进行判断，循环体语句至少被执行一次；while 语句则先判断后循环，有可能一次也不执行循环体语句。当 while 后面表达式的值第一次循环就为真(非 0) 时，while 语句与 do-while 语句完全等价。

(3) 还需要注意的是：在 do-while 语句中，while(表达式)后面有分号“;”，它是 do-while 语句结束的标志。如果漏掉了该分号，将造成语法错误，程序不能编译通过。而在 while 语句中，正常情况下 while(表达式)后面不能紧跟一个分号“;”，否则将造成空循环，并极有可能导致死循环。前面已对此做了详细说明，在此不再赘述。

**例 5.4**　将例 5.1 中计算 1＋2＋3＋…＋99＋100 的程序，改用 do-while 语句实现。

对比图 5-6(b)所示的 while 语句流程图和图 5-7(b)所示的 do-while 语句流程图，再参

照例 5.1 的处理方法，按照 do-while 语句的语法格式，不难写出如下的程序代码。

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

## 5.4 for 语句

for 语句的基本形式：

```
for(表达式 1; 表达式 2; 表达式 3)
    循环体语句;
```

或

```
for(表达式 1; 表达式 2; 表达式 3)
{
    多条循环体语句;
}
```

for 语句的流程图如图 5-8 所示。

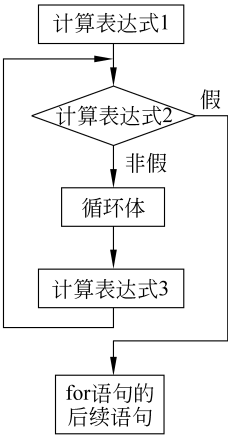


图 5-8　for 语句的流程图

**说明：**

(1) 与 while 语句一样，for 语句构造的循环也是一个当型循环。

如图 5-8 所示，for 语句的执行流程是：首先计算表达式 1 的值(只计算一次)，再计算表达式 2(循环条件)的值，并根据表达式 2 的值判断是否执行内嵌的循环体语句；如果表达式 2 的值为真(非 0) 时，则执行循环体语句，并紧接着计算表达式 3 的值，再回头计算表达式 2 的值，并根据表达式 2 的值判断是否执行循环体，……，如此循环往复。一旦表达式 2 的值为假(0 值)，即结束循环。

(2) 对比图 5-8“for 语句的流程图”和图 5-2(a)“完整的循环结构流程图-当型循环结构”，显而易见，两者可以完全对应。for 语句括号中的 3 个表达式“表达式 1、表达式 2、表达式 3”，分别对应“循环初始化、循环条件、循环变量改变”。即与循环结构基本逻辑完全对应的 for 语句的应用形式：

```
for(循环初始化;循环条件;循环变量改变)
{
    循环体语句;
}
```

(3) 注意在“for(表达式1；表达式2；表达式3)”中，分隔3个表达式的是两个分号“；”而不是逗号。

(4) 与while语句和do-while语句一样，当循环体中的语句不止一条时，必须用{}将循环体语句组合成一条复合语句。同样还要注意正确使用锯齿形书写格式，使循环体语句向右缩进对齐，以明确标识循环体的范围，提高程序的可读性。

(5) 还需要注意的是：正常情况下，“for(表达式1；表达式2；表达式3)”后面紧跟的是一个非空操作的循环体语句，而不能紧跟一个分号“；”，否则将造成空循环。这样的空循环，虽然不会像while语句那样导致死循环，但循环的结果却是“什么都没做”。

(6) 表达式2是空语句时，认为是非0，即真。如“for(i=0;;i++);”就是死循环。

**例5.5** 将例5.1中计算1+2+3+…+99+100的程序，改用for语句实现。

**分析**：for语句与while语句一样属于当型循环，它们求解问题的算法逻辑是一致的。因而可以依照图5-3(b)所示的程序流程图和for语句的语法格式，写出下面的程序代码。

```
01    #include<stdio.h>
02    main()
03    { int i,sum;
04      for(sum=0,i=1;i<=100;i++)
05      {  sum=sum+i;
06      }
07      printf("sum=%d\n",sum);
08    }
```

**例5.6** 按照格雷戈里公式：$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$，求π的近似值，直到最后一项的绝对值小于$10^{-6}$。

**分析**：图5-4中已经给出了计算1+1/2+1/3+…+1/100的程序流程图，而本题所使用的格雷戈里公式与其非常相似，同样可以把它看成一个不断累加求和的过程，所不同的是：每一次累加进来的分数项，正负符号位要改变，分母取值的变化率不同，累加的次数是未知的，但累加结束的时机是确定的，即循环条件仍然是明确的。

若累加和用sum表示，每次被加的分数项用t表示，分数项的分母用n表示，分数项的符号位(正负号)用sign表示。参照前面类似问题求解过程的分析和循环流程的设计，可以发现求解本问题的循环结构，其循环体包括sum=sum+sign*t，以及t的取值(t=1/n)和符号位sign的转换。sign的初值取1，则每一次循环，通过sign=-sign的运算，即可使符号位正负转换。显然，这也是通过迭代算法构造的循环，迭代关系式有两个：sum=sum+sign*t和sign=-sign。

那么循环的进程又该如何设计呢？可以将分数项的分母n直接作为循环变量，n的初值取1，每一次循环后使n=n+2，从而使n值依次取1、3、5、7……直到1/n小于$0.25\times10^{-6}$，即循环条件是$t\geqslant10^{-6}$。

通过以上分析，不难写出下面的程序代码。

```
01    #include<stdio.h>
02    main()
03    { int sign=1,n=1;          //sign用于设置正负号,n代表分母
```