第5章 周而复始——循环结构程序设计

我们已经了解了顺序和选择结构的程序设计方法，可以解决现实世界的某些问题，但还有一些问题，常常需要周而复始地重复执行某些流程或过程，直到满足某种限定条件才停止。

5.1 实际问题引例

问题 1：一般情况下，两个数相加，我们会直接计算两个数相加的结果；三个数相加，我们也会将这三个数直接相加。但是，如果要计算从 1 累加到 100 的值，通常我们会有一个计数器，其初始值为 0，然后加 1，再加 2，继续加 3，一直加到 100。我们发现，最终进行了 100 次加法，只是每次加的数增加 1。也就是说，加法运算重复进行了 100 次。

问题 2：判断从 1～100 哪些数能够被 3 整除。我们需要从 1 开始，一直到 100，依次验证对应的数是否能被 3 整除。这又是一个不断选择数的过程。这个计算过程需要将 1～100 的所有数反复与 3 进行整除计算，继而选择 1～100 能够被 3 整除的所有数。

如同问题 1 和问题 2 中所描述的那样，在现实生活中，我们经常会遇到累加、计数、重复判断等问题，这就相当于要多次进行同样的操作和过程。这一类现实问题转化为抽象的程序设计，就需要在程序当中对某部分程序代码重复执行多次（其执行的是“一遍又一遍”的算法步骤）。为此，C 语言提供了一种解决此类问题的控制结构——循环结构。

5.2 循环的概念

循环结构是结构化程序设计的基本结构之一，它和顺序结构、选择结构共同作为各种复杂程序的基本构造单元。所谓循环，就是指反复执行相同性质的一个或多个操作步骤。如反复执行加、减、乘、除等。

循环分为两种：无休止循环（无限循环）和有终止循环（有限循环），无限循环是指循环无限次地执行，不会终止。反之，有限循环是指有一定循环次数，或者满足某种条件就会结束的循环。初学者应将重点放在对有限循环的掌握上。

C 语言用如下语句体实现循环结构。

（1）while 语句结构。

（2）do-while 语句结构。

（3）for 语句结构。

(4) 用 goto 和 if 构成循环(已基本不使用)。

无论采用哪种语句结构,在编写循环结构程序时,需要重点设计构成有效循环的条件:循环体内的语句和循环结束条件。

5.3 用 while 语句实现循环

5.3.1 while 语句的基本形式

一般形式:

```
while (表达式)
    循环体语句
```

当表达式为非 0 值时,执行 while 语句体中的循环语句。执行过程如图 5-1 所示。

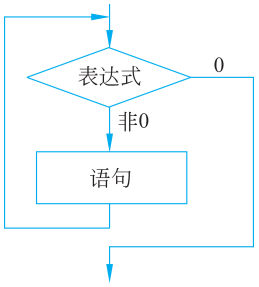

图 5-1 while 循环流程图

5.3.2 while 语句特点和说明

特点:while 循环是先判断表达式,再决定是否执行循环体。

说明:

(1) 循环体有可能一次也不执行。

(2) 循环体可为任意类型语句。

(3) 下列情况,退出 while 循环:

① 条件表达式不成立(为零)。

② 循环体内遇到 break、return。

【例 5.1】 用 while 循环求 $\sum_{n=1}^{100} n$。

【问题分析】

(1) 这是一个典型的累加运算,需要反复执行加法运算,故需要用到循环结构。

(2) 在累加过程中,加数从 1 开始,每次累加的数都比上一次累加上去的数大 1,故可以设置一个变量,例如,变量 i,初值为 1,每次都用上次累加的结果加上变量 i,累加一次后 i 就加 1,当 i>100,就结束循环(循环 100 次)。

(3) 还需要一个变量存放每次累加的结果,例如,变量 sum,不断地在 sum 上做累加,循环结束后 sum 的值就是最终的累加结果。

【问题求解】

```
1   /* 程序代码 5.1:求 1~100 的和 */
2   #include<stdio.h>
3   int main()
4   {
5       int i,sum;
6       i = 1;
7       sum = 0;
8       while(i <= 100)
9       {
10          sum = sum + i;
```

```
11          i = i+1;
12      }
13      printf("sum=%d\n", sum);
14      return 0;
15  }
```

程序运行结果：

```
sum=5050
```

【应用扩展】

（1）循环体如果包含一个以上的语句，应该用花括号括起来，以复合语句形式出现。

（2）循环开始前，需要对循环变量赋初值，如程序中的“i=1;”。

（3）在循环体中应有使循环趋向于结束的语句。如程序中的“i++;”，i 的自增保证了循环结束条件最后变为不满足。循环变量的增值称为步长，可以为正（增加），也可以为负（减小），此处步长为 1。

（4）循环变量赋初值，循环结束条件，循环变量增、减值称为循环的三要素，它们直接影响到循环的执行次数。

【例 5.2】 输入一批整数，输入的数为 0 时终止输入，计算这批数之积。

【问题分析】

（1）由于要反复执行乘法求积运算，考虑使用循环。

（2）由于是以输入 0 作为输入的结束，也就是说循环次数无法预知，输入 0 后退出循环，不再进行乘法运算。

【问题求解】

```
1   /* 程序代码 5.2:求一批整数的乘积 */
2   #include<stdio.h>
3   int main()
4   {
5       int i, mul=1;
6       scanf("%d", &i);
7       while(i != 0)
8       {
9           mul=mul * i;
10          scanf("%d", &i);                    //输入新的乘数
11      }
12      printf("mul=%d",mul);
13      return 0;
14  }
```

程序运行结果：

```
1↙
3↙
5↙
7↙
0↙
mul=105
```

【应用扩展】

注意循环体最后一句修改了循环变量值，否则循环有可能无法结束。

【例 5.3】 用 while 语句实现 $n!$。

【问题分析】

根据题意，需要运用 while 循环语句来实现数学问题的求解，即 $n!=n\times(n-1)\times(n-2)\times(n-3)\times\cdots\times2\times1$。注意要考虑输入的为 0 或者 1 的情况。

【问题求解】

```
1   /* 程序代码 5.3:求 n! */
2   #include "stdio.h"
3   int main()
4   {
5       int i, n;
6       float s;                                    //累乘结果变量
7       i = 1;
8       s = 1.0;
9       scanf("%d", &n);
10      while(i <= n)
11      {
12          s = s * i;
13          i = i+1;
14      }
15      printf("%f\n", s);
16      return 0;
17  }
```

程序运行结果：

```
5↙
120.000000
```

【应用扩展】

若要解决 $n!$ 的问题，还可以使用递归的方法(第 7 章)，即多次自调用自定义函数进行迭代计算。

【例 5.4】 用 while 语句判断 1～100 哪些数能够被 3 整除，并输出这些数(5.1 节的问题 2)。

【问题分析】

根据题意，分析此题的特点，需要利用 while 控制 1～100 的数据，从数据中循环寻找满足被 3 整除条件的数并输出。

【问题求解】

```
1   /* 程序代码 5.4:求 1~100 能够被 3 整除的数 */
2   #include<stdio.h>
3   int main()
4   {
5       int i;
6       i = 1;
7       while(i <= 100)
8       {
9           if(i%3 == 0)
10          printf("%d  ", i);
11          i = i + 1;
12      }
13      return 0;
14  }
```

程序运行结果：

```
3  6  9  12  15  18  21  24  27  30  33  36  39  42  45  48  51  54  57  60  63
66  69  72  75  78  81  84  87  90  93  96  99
```

【应用扩展】

由该例题拓展，我们也可以编写程序来求解能够被 5 整除的数，既能够被 3 整除又能够被 5 整除的数，求解能被 2 整除的数（偶数）等。

5.4 用 do-while 语句实现循环

5.4.1 do-while 语句的基本形式

一般形式：

```
do
    循环体语句
while (表达式);
```

先执行一次指定的循环体语句，然后判别表达式，当表达式的值为非零时，返回重新执行循环体语句，如此反复，直到表达式的值等于 0 为止，此时循环结束。执行过程如图 5-2 所示。

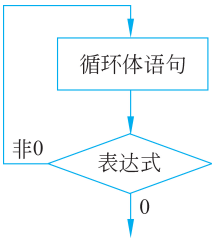

图 5-2 do-while 循环流程图

5.4.2 do-while 语句特点和说明

特点：do-while 循环是先执行循环体，再判断表达式的值。

说明：

（1）循环体至少执行一次。

（2）循环体可为任意类型语句。

（3）下列情况，退出 do-while 循环：

① 条件表达式不成立（为零）。

② 循环体内遇到 break、return。

【例 5.5】 用 do-while 循环求 $\sum_{n=1}^{100} n$。

【问题分析】

根据题意，该题与例 5.1 相似，不过这里要求使用 do-while 语句实现循环结构。

【问题求解】

```
1   /* 程序代码 5.5:do-while 循环求 1~100 的和 */
2   #include<stdio.h>
3   int main()
4   {
5       int i,sum;
6       i = 1;
7       sum = 0;
8       do
9       {
10          sum = sum + i;
11          i = i + 1;
```

```
12      } while(i <= 100);
13      printf("sum=%d\n", sum);
14      return 0;
15  }
```

程序运行结果：

```
sum=5050
```

【应用扩展】

很多情况下，我们可用 while 循环和 do-while 循环处理同一问题，若两者的循环体部分是一样的，那么结果通常也一样。如例 5.1 和例 5.5 程序中的循环体是相同的，得到的结果也相同。那么 while 循环和 do-while 循环是否在任何情况下都完全等价呢？

5.4.3 while 和 do-while 循环的比较

while 循环与 do-while 循环通常可相互替换，但两者并不完全等价。

在一般情况下，用 while 语句和用 do-while 语句处理同一问题时，若两者的循环体部分是一样的，它们的结果也一样。但是如果循环结束条件一开始就为假时，两种循环的结果是不同的。

【例 5.6】 求 1～10 的累加和，分别用 while 和 do-while 循环。

```
1   /* while 循环求 1~10 的累加和 */
2   #include<stdio.h>
3   void main()
4   {
5       int sum=0,i;
6       scanf("%d",&i);
7       while (i<=10)
8       {
9          sum=sum+i;
10         i++;
11      }
12      printf("sum=%d\n",sum)
13  }
```

```
1   /* do-while 循环求 1~10 的累加和 */
2   #include<stdio.h>
3   void main()
4   {
5       int sum=0,i;
6       scanf("%d",&i);
7       do
8       {
9           sum=sum+i;
10          i++;
11      }
12      while (i<=10);
13      printf("sum=%d\n",sum);
14  }
```

程序运行结果：

```
1↙
sum=55
```

```
1↙
sum=55
```

再运行一次结果：

```
11↙
sum=0
```

```
11↙
sum=11
```

【应用扩展】

可以看到，当 while 后面表达式的第一次值为“真”时，两种循环得到的结果相同。否则，两者结果不相同。

5.4.4 while 与 do-while 循环程序举例

【例 5.7】 用 while 循环输出华氏—摄氏温度转换表，将华氏温度 30～35°F 的每一度都转换成相应的摄氏温度。

【问题分析】

(1) 华氏-摄氏温度转换公式为 $C=5\times(F-32)/9$，其中 C 为摄氏温度，F 为华氏温度。

(2) 由于要转换 30～35℉的每一度，意味着转换公式会反复执行，故考虑用循环。

(3) 循环变量初值为 30℉，终值为 35℉，步长(增值)为 1。

【问题求解】

```
1   /* 程序代码 5.7:输出华氏-摄氏温度转换表 */
2   #include "stdio.h"
3   int main()
4   {
5       int fah;
6       float cent;
7       fah=30;
8       while(fah<=35)
9       {
10          cent=5.0*(fah-32)/9;
11          printf("Fah=%d  Cent= %.1f \n", fah,cent);
12          fah++;
13      }
14      return 0
15  }
```

程序运行结果：

```
Fah=30  Cent=-1.1
Fah=31  Cent=-0.6
Fah=32  Cent=-0.0
Fah=33  Cent= 0.6
Fah=34  Cent= 1.1
Fah=35  Cent= 1.7
```

【应用扩展】

是否能用 do-while 循环实现相同的功能呢？试着做一做。

【例 5.8】 用 do-while 循环求 $n!$。

【问题分析】

(1) 求阶乘的公式为

$$s=1\times2\times3\times\cdots\times(n-1)\times n$$

(2) 可以看到反复执行了乘法，考虑用循环。

(3) 循环变量初值为 1，步长为 1，流程图如图 5-3 所示。

【问题求解】

```
1   /* 程序代码 5.8:用 do-while 循环求 n! */
2   #include<stdio.h>
3   void main()
4   {
5       int  i=1,n;
6       long s=1;
7       scanf(" %d ",&n);
8       do
9       {
10          s*=i;
11          i++;
```

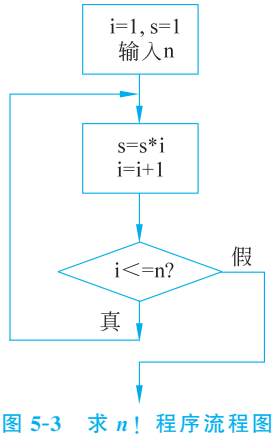

图 5-3 求 $n!$ 程序流程图

```
    } while (i<=n);
    printf("%d!=%ld\n",n,s);
}
```

程序运行结果：

```
5↙
5!=120
```

【应用扩展】

在例 5.3 中，我们曾用 while 循环实现了求 $n!$，比较两个程序有什么不同？

5.5 用 for 语句实现循环

5.5.1 for 语句的一般形式

一般形式：

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

for 循环先执行表达式 1，再判断表达式 2，根据表达式 2 的值决定是否执行循环体语句，如果表达式 2 的值为非零，则执行循环体语句，执行完循环体后再执行表达式 3，然后再次判断表达式 2(表达式 1 不再执行)，根据表达式 2 的值再次决定是否执行循环体语句，如此反复执行，直到表达式 2 的值为零，退出循环。具体执行流程如图 5-4 所示。

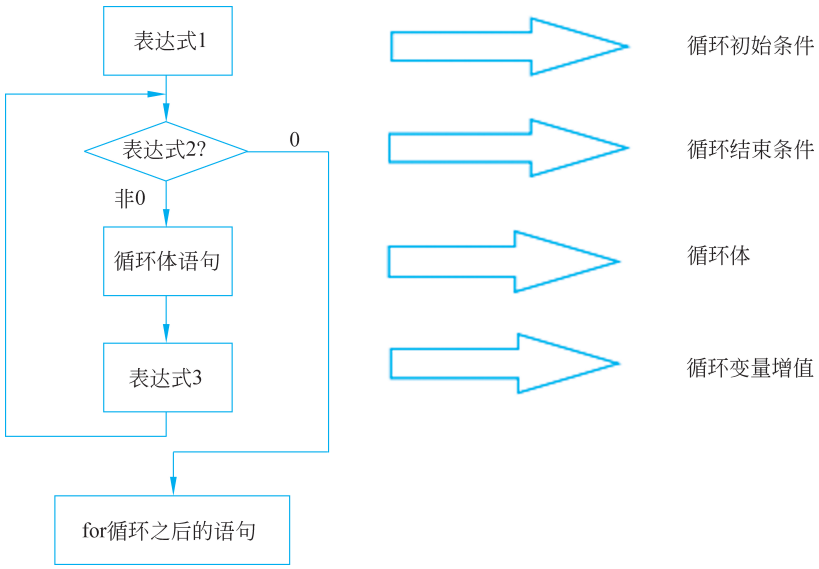

图 5-4 for 循环执行流程图

说明：

(1) 表达式 1 只执行一次。

(2) 循环体可为任意类型语句。

(3) 表达式 2 执行完并不马上执行表达式 3。

(4) 下列情况，退出 for 循环：

① 表达式 2 不成立(为零)。

② 循环体内遇到 break、return。

通常情况下,for 语句可以理解为

```
for(循环变量赋初值;循环条件;循环变量增值)
```

例如:

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

上面的 for 循环相当于:

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

可以看出用 for 循环,程序会更简洁。

5.5.2 for 语句的各种形式

for 语句相当灵活,形式变化多样。

(1) for 语句的一般形式中的"表达式 1"可以省略,此时应在 for 语句之前给循环变量赋初值。注意省略表达式 1 时,其后的分号不能省略。例如:

```
for(;i<=100;i++)  sum=sum+i;
```

执行时,跳过"求解表达式 1"这一步,其他不变。

(2) 如果表达式 2 省略,即不判断循环条件,循环无终止地进行下去。也就是认为表达式 2 始终为真。例如:

```
for(i=1; ;i++)    sum=sum+i;
```

表达式 1 是一个赋值表达式,表达式 2 空缺,相当于循环结束条件永真。

(3) 表达式 3 也可以省略,但此时程序设计者应另外设法保证循环能正常结束。例如:

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

在上面的 for 语句中只有表达式 1 和表达式 2,而没有表达式 3。"i++"操作不放在 for 语句的表达式 3 的位置处,而作为循环体的一部分,效果是一样的,都能使循环正常结束。

(4) 可以省略表达式 1 和表达式 3,只有表达式 2,即只给循环条件。例如:

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

在这种情况下,完全等同于 while 语句。可见 for 语句比 while 语句功能强,除了可以

给出循环条件外，还可以赋初值，使循环变量自动增值等。

(5) 3个表达式都可省略，即不设初值，不判断条件(认为表达式2为真值)，循环变量不增值。无终止地执行循环体。例如：

```
for(; ;)
```

相当于语句

```
while(1)
```

(6) 表达式1可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式(通常不要这么做)。例如：

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

表达式3也可以是与循环控制无关的任意表达式(通常不要这么做)。

表达式1和表达式3可以是一个简单的表达式，也可以是多个表达式，中间用逗号间隔，形成逗号表达式。例如：

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

(7) 表达式2一般是关系表达式(如i<=100)或逻辑表达式(如a<b && x<y)，但也可以是数值表达式或字符表达式，只要其值为非零，就执行循环体。例如：

```
for(i=0;(c=getchar())!='\n';i+=c);
```

在表达式2中先从终端接收一个字符赋给变量c，然后判断此赋值表达式的值是否不等于'\n'(换行符)，如果不等于'\n'，就执行循环体。

注意：此for语句直接以分号结束，没有循环体，这种语句称为空循环体语句，把本来要在循环体内处理的内容放在表达式3中。可见C语言的for语句功能强，使用灵活，可以使程序短小简洁。但过分地利用这一特点会使for语句显得杂乱，可读性低，最好不要把与循环控制无关的内容放到for语句中。

5.5.3 for循环程序举例

【例5.9】 用for语句计算1～100的累加值，并输出结果。

【问题分析】

根据题意，需要使用for语句，那么需明确在本题中for语句的执行过程。“i=1”给循环变量i设置初值为1，“i<=100”是循环条件。当循环变量i的值小于或等于100时，循环继续执行。“i++”的作用是使循环变量i的值不断变化，以便最终满足终止循环的条件，使循环得以结束。

【问题求解】

```
1   /*程序代码5.9:for循环求1~100的和*/
2   #include<stdio.h>
3   int main()
4   {
5       int i,sum;
6       for(i=1, sum=0; i<=100; i++)
7       {
```