第 5 章

循 环 结 构

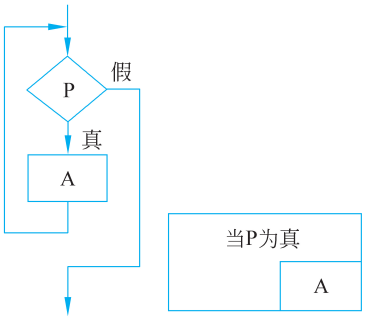
在程序设计中，很多实际问题有重复性操作或规律性操作的需求，因此需要重复执行某些语句。循环结构作为结构化程序设计的 3 个基本结构之一，能够减少代码的重复书写，而计算机的一大特点就是高速运算，能在短时间内完成大量有规律的重复性操作。

本章将对循环结构中的 for 循环、while 循环和 do…while 循环进行介绍。

5.1 循环的相关概念

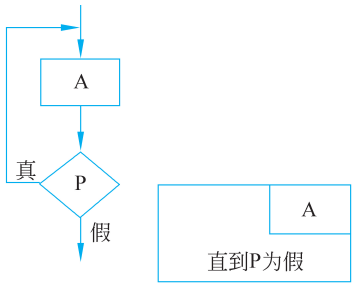
5.1.1 循环结构

循环结构就是在某种条件成立时，反复执行某一组语句的结构。被反复执行的语句称为循环体，控制循环操作的条件称为循环条件。循环结构有两种，一种是当型循环，一种是直到型循环，它们的流程图分别如图 5-1 和图 5-2 所示。



(a) 传统流程图　(b) NS流程图

图 5-1　当型循环结构



(a) 传统流程图　(b) NS流程图

图 5-2　直到型循环结构

当型循环是当循环条件 P 为真时，就执行循环体 A；当循环条件 P 为假时，结束循环。

直到型循环是至少执行一次循环体 A，然后再根据循环条件 P 决定是否继续循环。

可以看出，当型循环是先判断循环条件再决定是否循环，因此有可能一次循环也不执行，而直到型循环至少执行一次循环体。循环结构中的 for 循环和 while 循环都属于当型循环，do…while 循环属于直到型循环。

为使循环体能被正常执行，在设计循环结构时必须考虑两个问题：

(1) 循环次数是确定的还是不确定的？如何为不确定次数的循环设置循环条件。

(2) 每次循环时,循环体中的各项数据是不变的还是有规律性变化的?如何把这种规律性变化与循环的轮次建立起关联。

5.1.2　循环条件的设计

循环条件有两种设计方式:一种是按照循环次数设计,一种是按照终止条件设计,前者用于循环次数确定的循环,后者用于循环次数不确定的循环。例如,类似"用户输入 10 个学生的成绩,计算他们的平均分"的问题就是能确定循环次数的(循环 10 次),而类似"用户输入若干个学生的成绩,计算他们的平均分"的问题就是不能确定循环次数的(循环若干次)。不过,这种无法确定循环次数的问题一定会给出终止循环的条件,如"当用户输入－1 时,表示输入结束"。

1. 按照循环次数设计

在能确定循环次数的程序设计中,需要用一个整型变量来计数循环轮次,这个变量称为循环变量,约定俗成依次用循环变量 i、j、k 为不同层的循环计数循环轮次。编程时,要为循环变量设置 3 个值:

(1) 循环变量的初始值。

(2) 循环变量的目标值。

(3) 循环变量的递增/递减值(也称为步长)。

有了这 3 个值,就能确定循环次数。

假设要循环 n 次,可以把循环变量的初值设置为 1,目标值设置为 n,步长设置为＋1。当然也可以反过来,初始值设置为 n,目标值设置为 1,步长设置为－1,不过人们通常更习惯于正着计数。当然,如果随着循环轮次的增加,循环体里的数据会有规律性变化时,如何确定循环变量的 3 个值就必须具体问题具体分析了。

2. 按照终止条件设计

在循环次数不确定的程序设计中,要根据要求设置相应的循环条件。题目给出的通常是终止循环的条件,因此在编程时要取反处理,把终止条件变成循环条件。

例如,对于"用户输入若干个非负数,计算它们的和,当输入－1 时,表示输入结束",循环条件就是"当输入不为－1 时,进入循环求和"。

再如,对于"输入 s,计算自然数数列前多少项之和刚好超过 s",循环条件就是"当数列项之和不超过 s 时,进入循环继续累加下一个数列值"。

5.1.3　循环体的设计

循环体就是每次循环时要执行的语句,这些语句要用花括号括起来形成一个语句组,只有当循环体是单条语句时,才可以省略花括号。

循环体可分为无变化规律的循环体和有变化规律的循环体两种。对于无变化规律的循环体,只需把要进行循环的语句用花括号括起来即可。对于有变化规律的循环体,要找到变

化规律与循环轮次的关联，并在循环体的相关数据中使用这种关联。

例如，“用户输入n，在屏幕上输出n个*”，循环体其实就是一条printf("*")语句，而“用户输入n，在屏幕上输出自然数数列的前n个数”，每次输出的数值就是有变化规律的，在设计循环体语句时，就需要考虑这种变化规律与循环轮次的关系。

本章将在5.7节编程实战中以数据统计（如求和、求平均值、求最值等）、特殊性质数的判断（如素数、完数、水仙花数等）、字符处理（如分类统计、大小写互换等）、图形打印和穷举等问题为例，介绍不同类型问题的循环体设计思路。

5.1.4 循环效率的分析

算法设计中有一个“高效性”的指标，分别从时间复杂度和空间复杂度上对算法的效率进行分析。

1. 时间复杂度

时间复杂度用于估算算法的运行时间。假设CPU在处理每个操作单元时消耗的时间都是相同的，那么就可以根据算法的操作单元数量大致估计出算法的运行时间。假设算法的问题规模为n，操作单元数量用函数f(n)来表示。如果随着问题规模n的增大，算法运行时间的增长率与f(n)的增长率相同，就称它是时间复杂度，记为O(f(n))，即时间复杂度是f(n)的函数。

如果不管n为多少，程序代码始终只执行一次，算法的时间复杂度就是O(1)。

如果n为多少，程序代码就大概运行多少次，算法的时间复杂度就是O(n)。

类似地，还有$O(n^2)$、O(logn)、O(nlogn)等时间复杂度，图5-3显示了5种时间复杂度的对比。

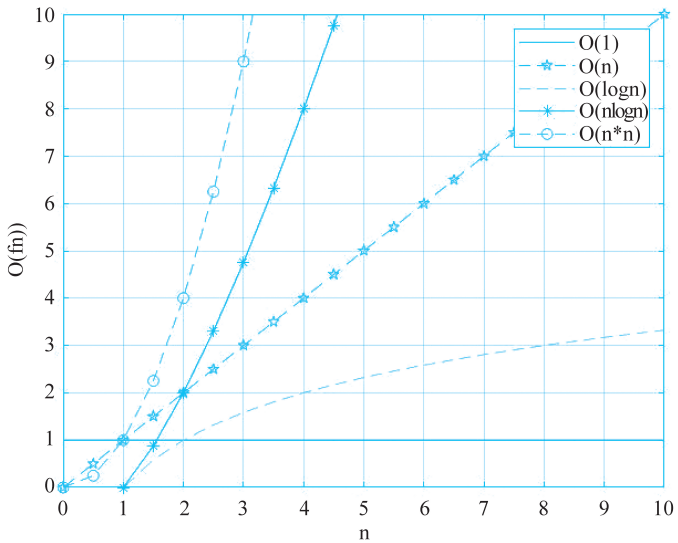


图5-3 5种时间复杂度的对比

2. 空间复杂度

空间复杂度是指一个算法在运行过程中占用存储空间大小的量度。它包括算法本身占用的存储空间、算法的输入输出数据占用的存储空间和算法在运行过程中临时占用的存储空间 3 部分。

(1) 算法本身所占用的存储空间与算法的代码长度成正比，所以想减少这部分的存储空间，就要编写出较短的算法。

(2) 算法的输入输出数据所占用的存储空间是由要解决的问题决定的，不会随着算法的不同而改变。

(3) 算法在运行过程中临时占用的存储空间与算法的设计有关。如果算法只需要占用少量的临时存储空间，并且这些空间不会随着问题规模的大小而改变，就是节省存储空间的算法。

与时间复杂度类似，空间复杂度主要有 O(1)、O(n)和 $O(n^2)$等。

5.2　for 循环

for 循环用于实现当型循环，使用方式非常灵活，它的一般形式为：

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

其中，

(1) 表达式 1 的作用是对循环变量设置初值，其他一些需要在循环开始前设置初值的变量也可以在表达式 1 中进行初值的设置。可使用逗号表达式同时为多个变量赋初值，如 i=1，s=0。

(2) 表达式 2 的作用是设置循环条件，当循环条件的结果为真时进入循环。

(3) 表达式 3 的作用是设置循环变量的步长，与表达式 1 类似，可使用逗号表达式同时为多个变量设置步长。

(4) 3 个表达式之间用分号进行间隔，且两个分号不可省略。

(5) 循环体要用花括号括起来，只有当循环体是单条语句时，才可以省略花括号。

图 5-4 显示了 for 循环的执行过程。

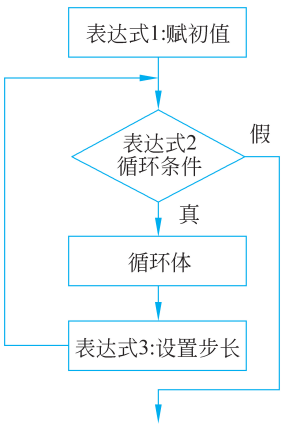


图 5-4　for 循环的执行过程示意图

可以发现：

(1) 表达式 1 在整个循环过程只执行一次。因此，如果在循环开始之前的语句中已对相关变量设置了初值，表达式 1 可

省略不写。

(2) 循环体执行完毕后会先执行表达式3,然后再执行表达式2即循环条件的判断。因此,如果循环体中已对循环变量的值进行了更新,表达式3可省略不写,或者说可以把表达式3写进循环体。

(3) 如果表达式2的结果为真就执行循环,否则就退出循环。事实上,表达式2也可以省略不写,不过这就意味着循环条件始终为真,无法正常退出循环,这种情况被称为"死"循环。为了避免"死"循环,当表达式2被省略时,必须在循环体内有终止循环的判断语句(参见5.5节),也可以理解为把循环条件取反处理后,当作终止条件写进循环体。

以下列举了for循环的几种变体形式,读者们也可以写出其他的变体形式。

(1)

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

(2)

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

(3)

```
表达式1;
for(     ;       ;     )
{
    if(!表达式2)                    //将循环条件取反后转为终止条件
        break;                      //break用于退出它所在的循环
    循环体;
    表达式3;
}
```

例5-1 用户输入一个不超过20的正整数n和一个字符c,在一行中输出n个字符c。例如用户输入6 *,输出******。

【问题分析】

(1) 循环次数是确定的吗?

是确定的。用户输入n,就执行n次循环,时间复杂度是O(n)。

(2) 每次循环时,循环体中的内容有变化吗?

没有。每次循环的任务都是输出一个同样的字符。

(3) 题目对 n 的取值范围作了说明，它的作用是方便编程者定义数据类型。通常情况下，如果没有明确取值范围，整型用 int，浮点型用 float 或 double。

【程序设计】

```
int main()
{
    int i, n;
    char a;
    scanf ("%d%*c%c",&n,&a);          //%*c 用于跳过(虚读)数值和字符之间的分隔符
    for(i=1;i<=n;i++)                 //for 循环的 3 个表达式能很直观地呈现循环次数
        printf( "%c",a);
    return 0;
}
```

循环变量 i 的初值为 1，每次循环后加 1，如果 i≤n，就继续执行循环。显然，当变量 i 增加到比 n 大时就会退出循环，退出循环时的 i 值是 n+1。

```
8 #
########
Process returned 0 (0x0)
```

图 5-5　例 5-1 的运行结果示例

程序的运行结果如图 5-5 所示。

注意

在 C99 标准中，允许在 for 循环的表达式 1 中临时定义循环变量，如：

```
for(int i=1;i<=n;i++)
{
    循环体;
}
```

此时变量 i 的作用域只限于 for 循环体内，当循环结束后，变量 i 所占用的存储空间就会被系统释放。

5.3　while 循环

while 循环用于实现当型循环，它的一般形式为：

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

while 循环与 for 循环无条件等价，只是各个表达式的位置不一样而已。

与 for 循环一样，循环体要用花括号括起来。

用 while 循环实现例 5-1 的程序如下。

```
int main()
{
    int i, n;
    char a;
    scanf("%d%*c%c",&n,&a);
    i=1;
    while(i<=n)                          //while 循环的 3 个表达式位置与 for 循环不同
    {
        printf("%c",a);
        i++;
    }
    return 0;
}
```

5.4 do…while 循环

do…while 循环用于实现直到型循环，它的一般形式为：

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

注意 while 语句后面的分号不可缺少。

do…while 循环是“先执行，后判断”，会至少执行一次循环体，而 for 循环与 while 循环是“先判断，后执行”，有可能一次循环都不执行。

用 do…while 循环实现例 5-1 的程序如下。

```
int main()
{
    int i, n;
    char a;
    scanf("%d%*c%c",&n,&a);
    i=1;
    do
    {
        printf("%c",a);
        i++;
    }while(i<=n);                        //不可缺少分号
    return 0;
}
```

5.5　循环的跳转

有时，需要在满足某种条件时提前跳出循环，或者提前结束本轮循环，转而开始下一轮循环，这时就需要使用 break 语句或 continue 语句实现程序的跳转。

1. break 语句

break 语句已在 switch 结构中得到应用，其作用是让程序跳出其所在的 switch 结构。同理，在循环结构中使用 break 语句，也能起到让程序跳出其所在循环层的作用。在循环结构中使用 break 语句时，一般会给出某个特定条件，当符合这个条件时，就提前退出循环。

不过，使用 break 语句会带来一个现实问题：程序既可以通过循环条件正常退出循环，也可以通过特定条件提前退出循环，那怎么才能知道程序是正常退出循环，还是提前退出循环的呢？

此处列举两种解决方法：

（1）在退出循环后，判断循环变量是否仍满足循环条件，如果满足，就说明程序是提前退出循环的。

（2）设置一个标识及初值（如 int flag＝0），在提前退出时修改标识值（如 flag＝1）。循环结束后，就可以根据标识值判断退出循环的方式。

2. continue 语句

continue 语句的作用是提前结束当前轮次的循环，转而执行下一轮循环。

可以想象现在正在举行“接力跑圈”的团队比赛，前一个队员跑完圈后，下一个队员接着开始跑圈。如果规定只要有队员犯规，全队就结束比赛，就相当于使用了 break 语句；如果规定犯规队员不再继续跑圈，转而由下一个队员开始跑圈时，就相当于使用了 continue 语句。

题 5-1　以下程序的运行结果是（　　）。

```
int main()
{
    int a,b=1;
    for(a=1; a<=30; a++)
    {
        if(b>7)
            break;
        a++;
        if(b%3==1)
        {
            b+=3;
            continue;
        }
```

```
            a++;
        }
        printf("a=%d,b=%d", a, b);
        return 0;
    }
```

A. a=31,b=46　　B. a=7,b=10

C. a=31,b=31　　D. a=10,b=10

【题目解析】

答案：B。

第一轮循环开始时，a=1，循环条件 a<=30 的结果为真，因此进入循环。由于此时的 b 值为 1，因此 b>7 的结果为假，不执行 if 语句块。接着执行 a++，a 值变为 2。由于条件表达式 b%3==1 的结果为真，因此执行 if 语句块。b+=3 后 b 值变为 4，随后的 continue 语句提前结束了第一轮循环，转而执行表达式 3，a++后 a 值变为 3，并且仍满足 a<=30 的循环条件，开始第二轮循环。

以此类推，b 值会不断增加。当 b 值符合 b>7 时，执行 break 语句，退出循环。

长知识

问：有什么办法能在运行程序时查看每个变量值的变化？

(1) 可以利用 printf 语句打印中间计算结果。

(2) 可以利用 IDE 提供的调试工具。以 Codeblocks 为例，在调试之前先确定项目路径和项目名称不含中文，然后按照图 5-6～图 5-8 的顺序依次操作，就可以调试程序并查看变量值。

```
4   int main()
5   {
6       int a, b=1;
7       for (a=1; a<=30; a++)
8       {
9           if (b>7)
10           break;
11          a++;
12          if (b%3==1)
13          {
14              b+=3;
15              continue;
16          }
17          a++;
18      }
19      printf ("a=%d,b=%d", a, b);
20       return 0;
21  }
```

图 5-6　在要开始调试的代码行号后面单击，生成红色断点，再次单击会取消断点

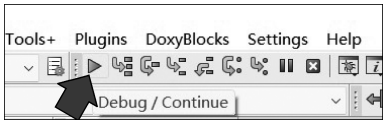


图 5-7　单击调试工具栏的三角图标或按 F8 快捷键开始调试

- 在需要开始调试的代码行号后面单击添加断点。
- 以调试方式运行程序，并打开 Watches 窗口。
- 在调试工具栏上选择需要的方式执行代码，并查看变量值的变化。

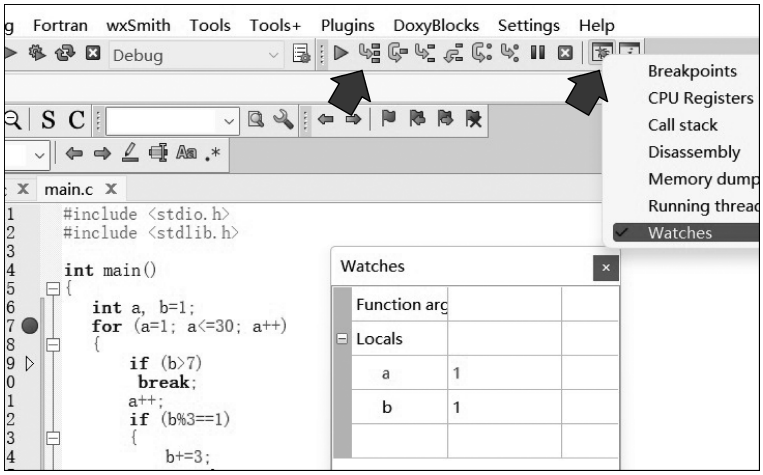


图 5-8　根据需要单击相应图标执行代码，并查看 Watches 窗口中的变量

题 5-2　以下程序的运行结果是(　　)。

```
int main()
{
    int x=2;
    while(x--)
        printf("%d",x);
    x=2;
    while(--x);
        printf("%d",x);
    return 0;
}
```

A. 100　　B. 101　　C. -10　　D. 1010

【题目解析】

答案：A。

建议通过调试工具查看 x 的变化过程。

(1) 先看第一个 while 循环。x 的初值是 2，循环条件 x－－是后自减 1，因此 x 先参与判断，结果为真，再自减 1 变为 1 后进入循环，输出 1。此时循环尚未结束，因此要再次进入循环条件的判断，x 先参与判断，结果为真，再自减 1 变为 0 后进入循环，输出 0。继续判断，x 为 0，0 代表逻辑假，不满足循环条件，再自减 1 后退出循环。可以发现，在退出循环时，x 值为－1。

(2) 重新给 x 赋值为 2 后再看第二个 while 循环。细心的读者想必已发现 while 语句后有分号，这个分号代表的是空语句，相当于

```
while(--x)
    ;                                    //循环体是空语句
```