5.2 循环结构的概述

循环结构（*iteration*）是高级程序设计语言三大控制结构之一，其特点是能够重复执行相同的代码。图 5-1 是循环结构的流程图。从图中可以清晰地看到一个回路的存在。

循环结构用程序设计语言中的循环语句来实现。一般情况下，循环语句都包含一个测试条件和一个执行部分（称为“**循环体**”）。当测试条件为真时，循环体将会被反复执行，直到条件不成立为止。如果条件一直为真，那么这个循环将会无休止地进行下去，可能会将计算机资源（特别是 CPU 资源）耗尽，从而使别的程序不能流畅运行。这种无限的循环称为“**死循环**”。因此，在循环语句的执行部分中，一定会有一些修改测试条件的语句，使循环能在有限次的运行后结束。这样的循环是“**有出口的**”，是一种良好的设计。

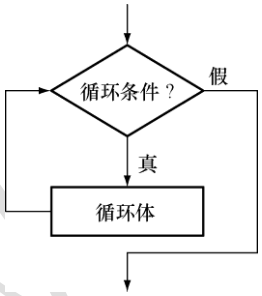

图 5-1 循环结构的流程图

要使循环正确运行的另一个条件是：循环必须从一个确定的起点开始。这称为循环的“**初始化**”。没有起点的循环可能会从一个随机的起点开始，从而导致不正确的结果。

一般地，一种程序设计语言中的循环语句有多种，但都分为两大类：当型和直到型。当型循环的结构是当测试条件成立时循环；直到型循环的结构是执行循环，直到条件不成立。C 语言的 3 种循环语句都属于当型循环。

5.3 while 语句

while 语句是 C 语言中较常使用的循环形式之一，它特别适合已知循环条件、却不明确循环的具体次数的情况。

5.3.1 while 语句的语法

当程序员明确知道循环初始条件，以及循环次数可以为 0 次的情况下，可以使用 while 循环，其语法如下：

```
while(表达式)
{
    循环体
}
```

while 语句的功能是：当表达式的值为真（非 0 值），执行一遍循环体语句，然后再计算表达式的值，当值为真，则继续执行循环体语句，并以此类推；如果表达式的值为假（0 值），则退出循环，执行紧跟在 while 后面的语句。特别地，当第一次表达式的值为假时，则直接退出循环，而不会做循环体语句。图 5-2 是 while 语句的流程图。

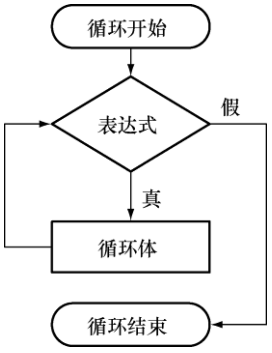

图 5-2 while 语句的流程图

while 语句的条件，表达式中往往包含一个变量，用于控制循环是否继续。这个变量称为“**循环控制变量**”，一般用于计数。当计数达到指定数量时，循环结束。

【例 5-1】求自然数 1 到 10000 的和。

【解题思路】题目是一种典型的级数求和问题。现实中求和常使用的方法是累加。所谓累加，就是设定一个中间变量，然后依次将级数的每一项都加到这个中间变量上。当级数的最后一项加完后，中间变量的值就是累加结果。累加的具体操作描述如下。

首先令中间变量 sum = 0，然后依次做如下累加操作：

sum ← sum + 1;（符号“←”的含义是将右面的结果存回到左面的变量中）

sum ← sum + 2;

sum ← sum + 3;

……

sum ← sum + 10000;

最后的 sum 即是所求的累积和。

从上面的操作中可以清楚地看到每一步完成的加法操作是完全相同的，只是参与操作的数不同。因此，上述的累加操作可以用循环来解决。可以制定这样的算法：

（1）令 sum = 0，i = 1;

（2）如果 i> 10000，转到第 6 步；

（3）否则，sum += i;

（4）++i;

（5）转到第 2 步；

（6）结束。

上述算法的流程图如图 5-3 所示。

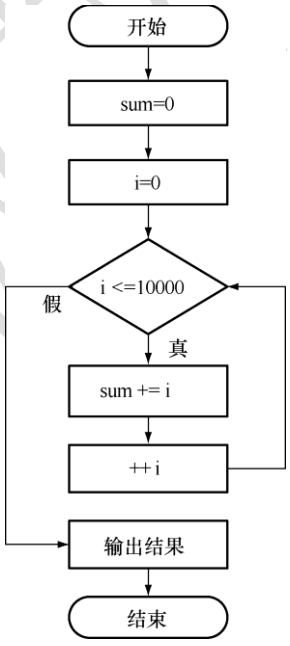

图 5-3　例 5-1 的流程图

根据上述算法写成的程序如下所示。

```
//5-1.c
#include<stdio.h>

int main()
{
    int i = 1, sum = 0;

    while (i<=10000)
    {
        sum += i;
        ++i;
    }

    printf("1 到 10000 的和=%d\n",sum);

    return 0;
}
```

程序的运行结果如下：

```
1 到 10000 的和=50005000
```

5.3.2　死循环

当 while 的条件为永真时就构成了一个死循环，例如，

```
while (1) { … }
```

甚至简单到

```
while (1);
```

死循环常常为某些特定的应用而设，这里提醒读者尽量避免死循环的出现。

> 初学者使用 while 语句时常犯一些错误。
>
> 1. 存在多余的分号。
>
> ```
> i = 0;
> while (i< 100); //请注意这个分号的存在
> {…}
> ```
>
> while 后面的分号将其后的复合语句排除在循环体之外，而真正的循环体只是一条空语句。这样一来，由于循环控制变量 i 在循环中不会改变，因此不可避免地造成死循环。

2．在循环体中没有改变循环控制变量的值，从而造成死循环。例如：

```
i = 0; sum = 0;
while (i< 100)//i 的值永远是 0，因此循环不会终止
{sum += i;}
```

C 语言程序设计要点 循环

一、算法设计

判断一个给定的数是否为质数。

【解题思路】首先来看看质数的定义。质数是除了 1 和本身外没有因子的数，例如 3、17、41 等。那么根据定义，要确定数 p 是否为质数，就可以通过测试 p 有没有整数因子来确定。如果有，则不是质数；反之则是。问题的关键是怎么去测试。一个笨的但却是最直接的方式就是让 p 一个个地去除以 2 到 p-1 之间的所有整数，只要其中一个能被整除，那么 p 肯定不是质数；如果所有的都不能被整除，则 p 一定是质数。

再深入分析下去。如果 p 比较大，那么 2 到 p-1 之间的数就会很多，因而测试的次数也很多。显然，为了减少次数，就不能让除数的范围太大。那么除数的上限应该是多少合适呢?

假设 p 有一个因子 q，不妨设 p=q×r，r 是某个整数。可以看到，其实 r 也是 p 的一个因子，也就是说，r 和 q 是“对称的”，即当 q 超过某一个临界值后，将重复先前 r 的值。把 q 和 r 当做函数中的两个自变量，它们的积是一个固定值，那么当且仅当 q 等于 r 时，函数达到一个临界点。此时 q 是 p 的正平方根，而这时的 q 就是测试中除数的最大值。

好了，有了以上分析，就可以设计出如下的算法。

（1）给定数 p，令 $q=\sqrt{p}$；

（2）令 i=2；

（3）令 r = p 除以 i 的余数；

（4）如果 r=0，则表明 p 不是质数，转到第 7 步；否则，令 i 等于它的后继值，即 i=i+1；

（5）如果 i≤q，那么转向第 3 步；

（6）否则，p 一定是质数；

（7）结束。

一旦设计出算法，那么就可以根据算法进行编码。但对于初学者，笔者强烈建议多做一步，就是根据算法画出程序的流程图。上述算法的流程图如图 1-9 所示。流程图中：

- 圆角矩形表示开始或结束；
- 矩形表示一般性处理过程；
- 菱形表示判断，结果取真假之一；
- 带箭头的线条表示流程的流向；
- sqrt(p)表示求 p 的平方根；
- p mod 2 表示求 p 除以 2 的余数。

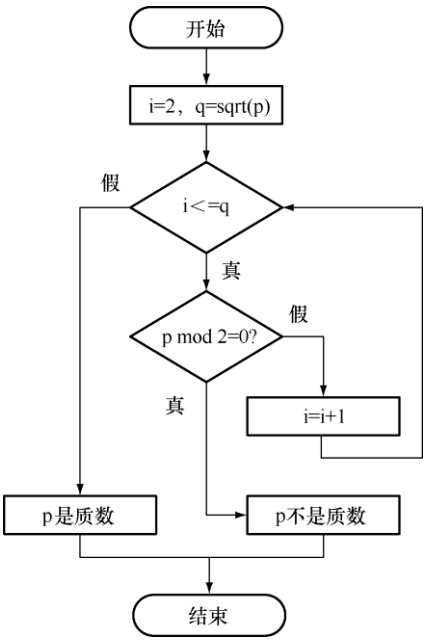

图 1-9　判断质数的算法流程图

从图 1-9 可以清晰地看到选择结构的存在，同时还可以看到一个回路，这表明了循环结构的存在。

二、程序设计

从键盘输入一个任意正整数，编程判断它是否是质数。若是，输出“Yes”，否则输出“No”。

【解题思路】在前面的章节里已经详细探讨过判断素数的方法，这里再来复习一下设计好的算法。

（1）给定数 p，令 q = $\sqrt{p}$；

（2）令 i=2。

（3）令 r = p%i。

（4）如果 r == 0，则表明 p 不是质数，转到第 7 步；否则，++i。

（5）如果 i<=q，那么转向第 3 步。

（6）否则，p 一定是质数，输出“Yes”，转到第 8 步。

（7）输出“No”。

（8）结束。

可以看到，这类算法与迭代法不同。迭代法试图找出级数中的递推关系，而这类算法试图在一定范围内将所有的可能都测试一遍，直到找到正确答案，或者发现无解为止。这种循环的模式称为“**穷举法**”（*enumerative method*），也是一种常用的解决方案。

读者应该已经确定，上述算法的实现肯定要用到循环语句。但现在的问题是：步骤 6 和步骤 7 都是循环语句之后的步骤，并且只有一步能被执行。如何保证做到这一点呢？一个常用的技巧就是在得到结论后，先不急着输出，而是用一个指示变量来保存结论的指示结果，然后在循环之外对这个指示变量进行测试，最后再输出结论。图 5-7 是本例的流程图。

```
//5-5.c
#include <stdio.h>
#include <math.h>

int main()
{
    int p, i, isPrime = 1;   //isPrime 指示变量

    printf("请输出一个正整数:");
    scanf("%d", &p);
    i = 2;
    while (i<= sqrt(p))
     {
        if (p % i == 0)
        {
              isPrime = 0;  //肯定不是质数
              break;
        }
        ++i;
     }
    printf("%s", isPrime == 1 ? "Yes" : "No");

    return 0;
}
```

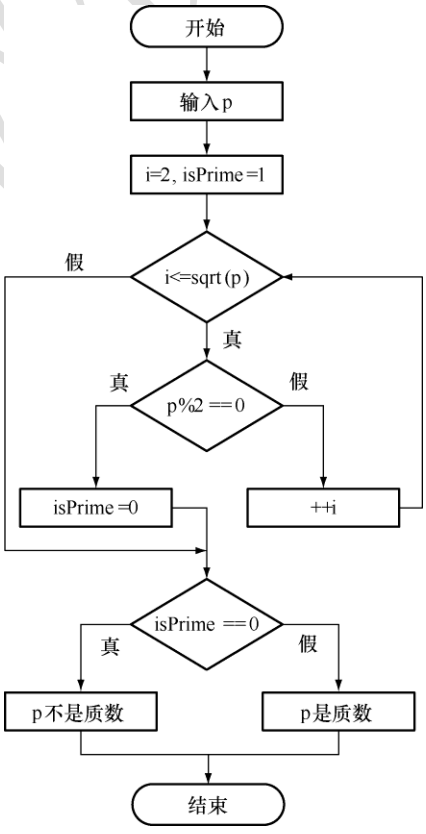

图 5-7 【例 5-5】的流程图

运行结果如下。

```
请输入一个正整数:17↙
Yes
```

另一次运行结果为：

```
请输入一个正整数:24↙
No
```