C 语言程序设计要点 2017-10-13（补充）

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。

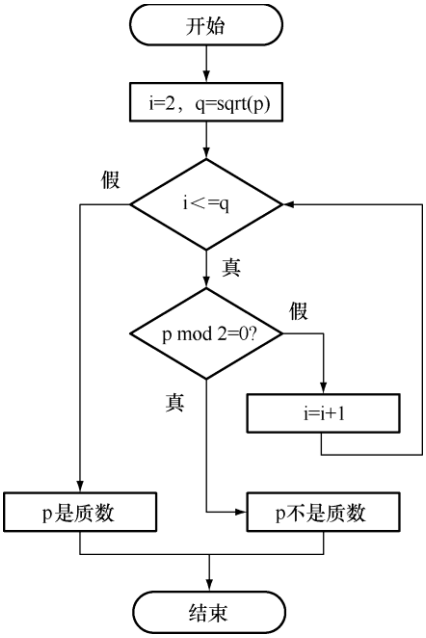

图 1-9　判断质数的算法流程图

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;
}
```

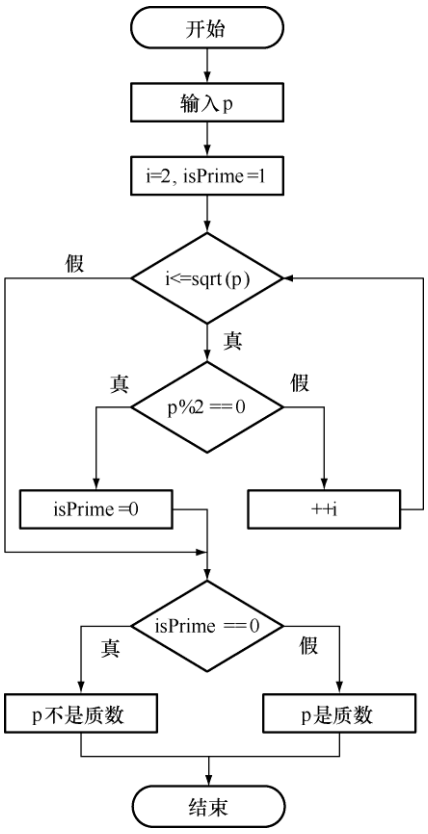

图 5-7 【例 5-5】的流程图

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
```