第 5 章　循环控制语句

循环结构是 C 语言程序设计中一种很重要的结构，其特点是在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。C 语言提供了多种循环语句，可以组成各种不同形式的循环结构。

5.1　程序验证

5.1.1　while 语句

while 循环的一般形式为：

```
while (条件)
    语句;
```

该语句用来实现“当型”循环结构，其执行过程是：首先判断条件的真伪，当值为真（非 0）时执行的语句，每执行完一次语句后，再次判断条件的真伪，以决定是否再次执行语句部分，直到条件为假时才结束循环，并继续执行循环程序外的后续语句。这里的语句部分称为循环体，它可以是一条单独的语句，也可以是复合语句。while 语句的逻辑结构如图 5.1 所示。

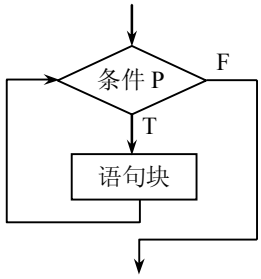

图 5.1　while 循环结构图

在使用 while 语句编写程序时需要注意以下几点：

（1）在 while 循环体内也允许空语句。

例如：

```
while((c=getche())!='\X0D');
```

在该例子中并没有出现执行语句部分，运行时不断地判断条件，直到键入回车为止。

（2）语句可以是语句体，此时必须用“{”和“}”括起来。

```
while(条件)
{
    语句 1;
    语句 2;
    ……
}
```

（3）可以有多层循环嵌套。

例如：

```
while(条件 1)
{
    语句;
    ……
    while(条件 2)
```

```
        {
        语句;
    ……
        }
    }
```

该实例循环体分为内外两层，在求解比较复杂的问题时往往使用嵌套结构。

1. 实验目的和要求

（1）理解循环结构程序段中语句的执行过程。

（2）掌握用 while 语句实现循环的方法。

（3）掌握如何正确地设定循环条件，以及如何控制循环的次数。

2. 实验重点和难点

（1）设定循环条件及控制循环次数。

（2）对程序运行前出现的错误进行调试。

3. 实验内容

（1）输入两个数，求它们的最大公约数和最小公倍数。

程序流程图如图 5.2 所示。

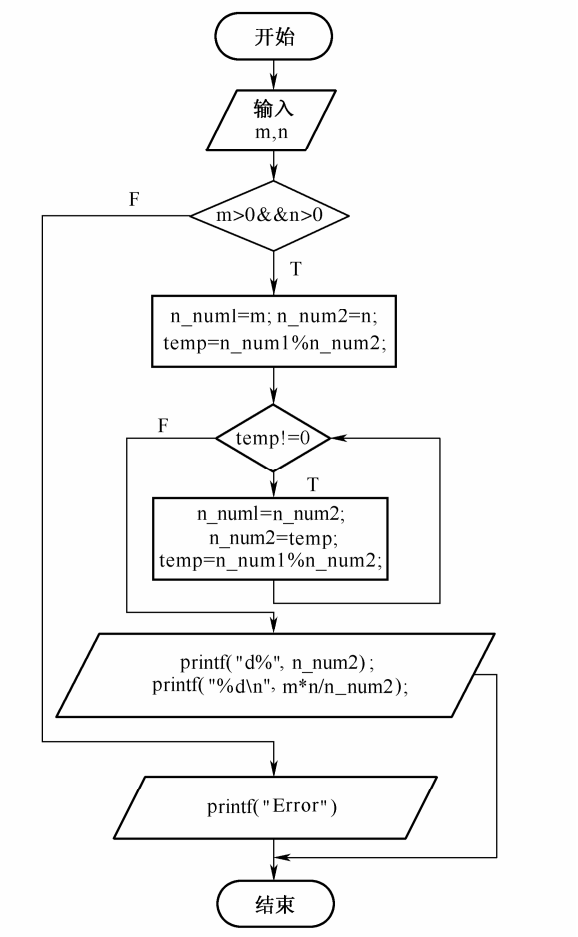

图 5.2　求最大公约数和最小公倍数程序流程图

参考程序

```
#include "stdio.h"
int main()
{
    int m, n;
    int n_num1, n_num2, temp; /*分别表示被除数，除数以及余数*/
    printf("Enter two integer:\n");
    scanf("%d %d", &m, &n);
    if (m > 0 && n >0) /*设置条件判断是否输入数据错误*/
    {
        n_num1 = m;
        n_num2 = n;
        temp = n_num1 % n_num2;
        while (temp != 0) /*判断余数是否为 0，以确定是否结束循环*/
        { /*进行除数与被除数的重新赋值*/
            n_num1 = n_num2;
            n_num2 = temp;
            temp = n_num1 % n_num2;
        }
        printf("最大公约数是: %d\n", n_num2);
        printf("最小公倍数是: %d\n", m * n / n_num2);
    }
    else printf("Error!\n");
    return 0;
}
```

分析：

求取最大公约数和最小公倍数是经典的 C 语言问题，一般采用辗转相除法来解决，该算法描述如下：

1）最大公约数算法描述。

m 对 n 求余为 temp，若 temp 不等于 0。

m <- n，n <- temp 继续求余。

如果 temp 等于 0，则 n 为最大公约数。

2）最小公倍数=两个数的积/最大公约数。

输入如上参考程序，编译运行，输入 15 和 85 两个数，结果如图 5.3 所示。

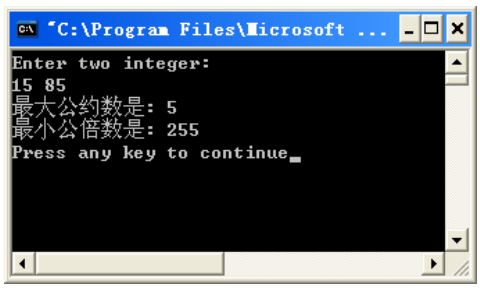

图 5.3　输入值为 15 和 85 的运行结果

如果按照输入格式正确输入参与运算的两个数据，一般能正确运行得到结果，如果在输

入时格式不符合要求，比如两个操作数中间用“,”连接，或者输入数据中存在负数，则出现错误提示，如图 5.4 所示。

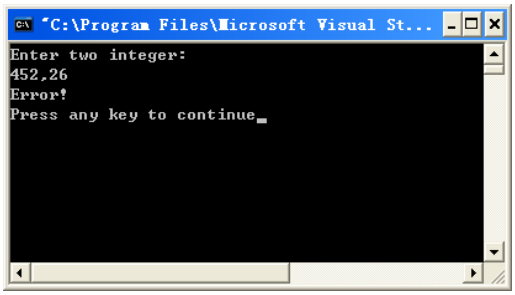

图 5.4　输入时格式错误的运行结果

（2）猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，不过瘾，又多吃了一个，第二天早上又将剩下的桃子吃掉一半，又多吃了一个，以后每天早上都吃了前一天剩下的一半零一个，到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少个桃子？

程序流程图如图 5.5 所示。

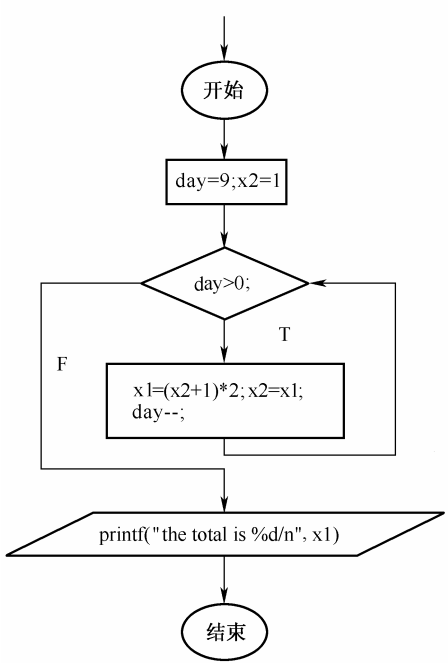

图 5.5　猴子摘桃流程图

参考程序

```
#include "stdio.h"
main()
{
    int day,x1,x2;/*分别代表天数，前一天的桃子数量及当天的桃子数量*/
    day=9;
    x2=1;/*第 9 天时桃子数量为 1*/
    while(day>0)
```

```
    {/*第一天的桃子数是第 2 天桃子数加 1 后的 2 倍*/
        x1=(x2+1)*2;
        x2=x1;
        day--;
    }
    printf("the total is %d\n",x1);
}
```

分析：

在该题目中，第 9 天时剩余一个桃子，因此可以倒退求解出第一天共有多少桃子。由于每一天桃子的数量都按照一定的规则变化，所以在计算的同时要对天数进行计数，考虑使用 while 循环结构解决。

题目中设置变量 x1 和 x2，并设置 x2 的初始值为 1，表示第 9 天时剩余的数量，此时 x1 表示前一天即第 8 天的桃子数量，因为第一天的桃子数是第 2 天桃子数加 1 后的 2 倍，则使用表达式(x2+1)*2 对 x1 进行赋值，并根据天数设置循环条件，最终求取到结果，运行程序后结果如图 5.6 所示。

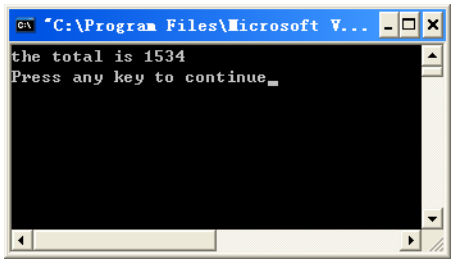

图 5.6　运行结果图

课后思考

（1）实验内容（1）的流程图如果要求以 N-S 图实现，如何修改？

（2）实验内容（2）改为“第 13 天想吃时剩余 2 个桃子”，应如何修改程序得到结果？

5.1.2　for 语句

for 语句是 C 语言所提供的功能更强，使用更广泛的一种循环语句。其一般形式为：

```
for(表达式 1;表达式 2;表达 3)
语句;
```

在该结构中各个参数的作用如下：

表达式 1：通常用来给循环变量赋初值，一般是赋值表达式。也允许在 for 语句外给循环变量赋初值，此时可以省略该表达式。

表达式 2：通常是循环条件，一般为关系表达式或逻辑表达式。

表达式 3：通常可用来修改循环变量的值，一般是赋值语句。

这 3 个表达式都可以是逗号表达式，即每个表达式都可由多个表达式组成。3 个表达式都是任选项，都可以省略。该语句的执行过程如下：

（1）首先计算表达式 1 的值。

（2）其次计算表达式 2 的值，若值为真（非 0）则执行循环体一次，否则跳出循环。

（3）最后计算表达式 3 的值，转回第（2）步重复执行。

在整个 for 循环过程中，表达式 1 只计算一次，表达式 2 和表达式 3 则可能计算多次。循环体可能多次执行，也可能一次都不执行，执行过程如图 5.7 所示。

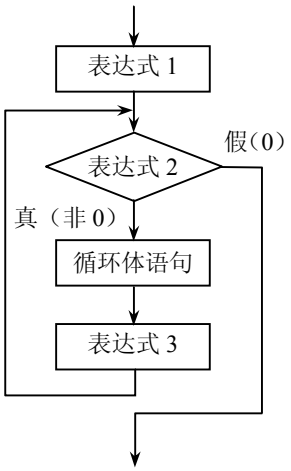

图 5.7　for 循环执行结构示意图

1. 实验目的和要求

（1）理解 for 循环结构的执行过程。

（2）掌握用 for 语句实现循环的方法。

（3）掌握如何正确地设定循环条件，以及如何控制循环的次数。

2. 实验重点和难点

（1）条件均为空时的执行原理。

（2）嵌套循环的执行原理。

3. 实验内容

（1）输入 n 值，输出如下所示高为 n 的等腰三角形。

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

n=6 时的等腰三角形

程序流程图如图 5.8 所示。

参考程序

```
#include "stdio.h"
main( )
{
    int i,j,n;
    printf("\nPlease Enter n: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        {
            for(j=1;j<=n-i;j++)
```

```
            printf("");
        for(j=1;j<=2*i-1;j++)
            printf("*");
        printf("\n");
    }
}
```

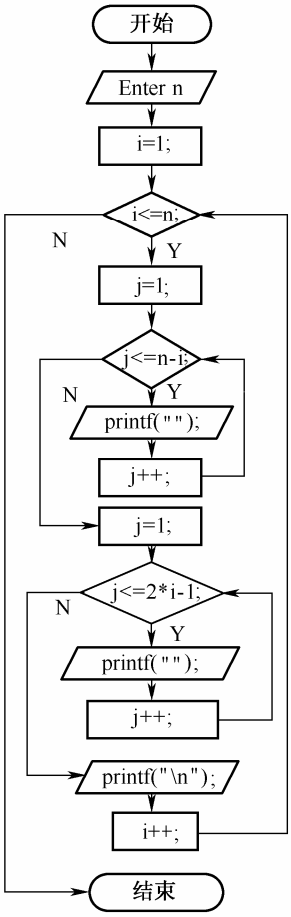

图 5.8　程序执行流程图

分析：

题目要求输出等腰三角形，首先观察图形发现规律：每行“*”前的空格数量恰好为所在行数减去 1，每行中“*”的个数都是奇数且数量逐行增加，其数目恰好为所在行 i 的 2 倍减去 1，知道该规律，就可以设计循环编写程序。

因为要求程序运行时首先输入等腰三角形行数，所以使用 scanf()函数将行数存储在变量 n 中。外层循环从 1 到 n 控制输出行数，内层循环需要 2 个，分别控制空格数目和“*”数目；控制空格时循环变量从 1 到 n-i，符合观察空格时的规律，控制“*”时循环变量从 1 到 2*i-1，同样符合“*”输出的规律。程序运行结果如图 5.9 和图 5.10 所示。.

（2）数字 1、2、3、4 能组成多少个互不相同且无重复数字的三位数？请打印输出。

程序流程图如图 5.11 所示。

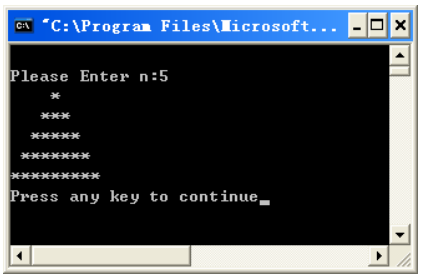

图 5.9　当 n 为奇数 5 时的运行结果图

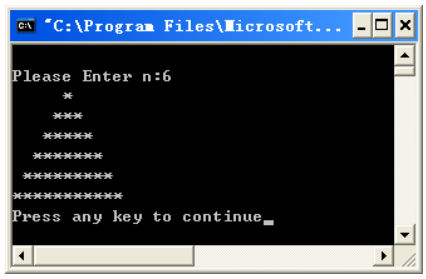

图 5.10　当 n 为偶数 6 时的运行结果图

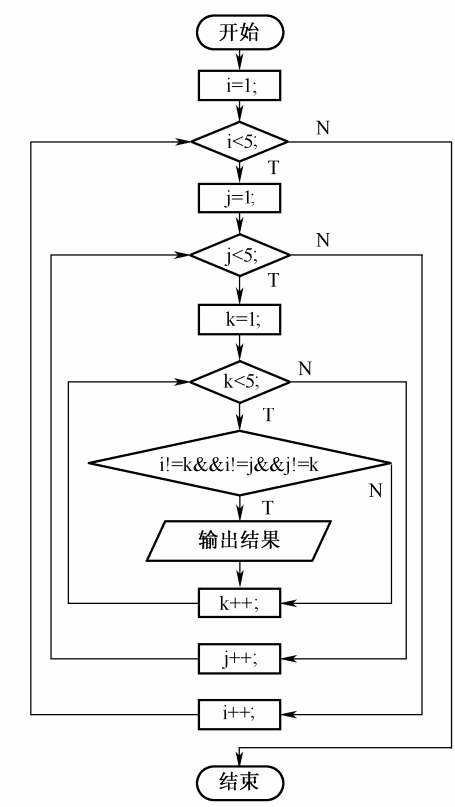

图 5.11　程序运行流程图

参考程序

```
#include "stdio.h"
main()
{
    int i,j,k,m=0;
    printf("\n");
    for (i=1;i<5;i++)
        for (j=1;j<5;j++)
            for (k=1;k<5;k++)
            {
                if (i!=k&&i!=j&&j!=k)
                {
```

```
                    printf("%d,%d,%d",i,j,k);
                    printf("\t");
                    m++;
                    if (m%5==0) printf("\n");
                }
            }
        printf("\n");
    }
```

分析：

对该类问题求解时，可以考虑使用穷举法。程序要求得到 3 位数，而每位可取的值是 1 到 4 中的任何一个，所以使用循环时循环变量从 1 到 4，但需要控制的是无重复出现，所以控制条件为“i!=k&&i!=j&&j!=k”，只要满足该条件就算符合要求。

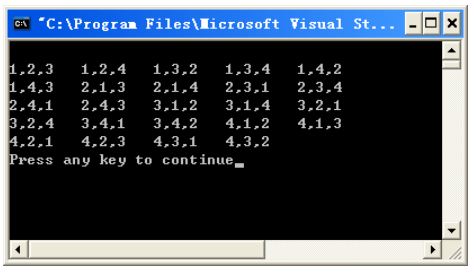

图 5.12　程序运行结果图

为了控制输出结果时的美观程度，可以设置变量 m，该变量初始值为 0，每找到一个满足要求的结果，该值进行自加操作，当该值是 5 的倍数时控制换行，这样做就可以控制每行输出 5 个结果，视觉效果更好，该做法在很多程序中都可以应用。程序运行结果如图 5.12 所示。

（3）求 10000 内的整数，使它加上 100 后是一个完全平方数，再加上 168 又是一个完全平方数，求出该数并输出。

程序流程图如图 5.13 所示。

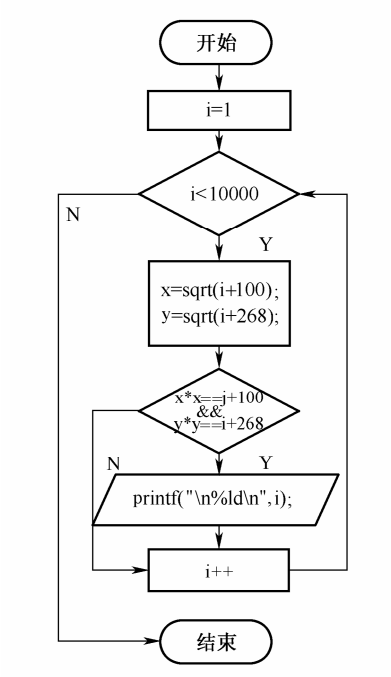

图 5.13　程序运行流程图

参考程序

```
#include "stdio.h"
#include "math.h"
int main()
{
    long int i,x,y;
    for (i=1;i<10000;i++)
    {
        x=sqrt(i+100);/*x 为加上 100 后开方后的结果*/
        y=sqrt(i+268); /*y 为再加上 168 后开方后的结果*/
        /*如果一个数的平方根的平方等于该数，这说明此数是完全平方数*/
        if (x*x==i+100 && y*y==i+268)
            printf("\n%ld\n",i);
    }
    return 0;
}
```

分析：

假定 i 为题目要求的整数，该整数加上 100 后得到完全平方数 m，再加 168 后得到另一个完全平方数 n。因为 m 为完全平方数，所以 m 为正数，且 m=x*x；y 为正整数；同样的道理 n 为正数，且 n=y*y，所以最终判断条件为 x*x==i+100 && y*y==i+268 是否为真，如果为真，表明 i 就是所求的结果。程序运行结果如图 5.14 所示。

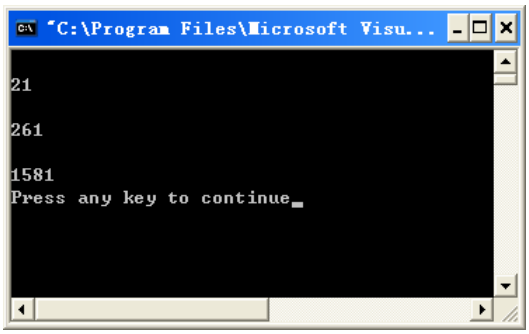

图 5.14　程序运行结果图

课后思考

（1）实验内容（1）中如果要求打印倒立的等腰三角形，如何实现？

（2）修改实验内容（3），改变循环结构为 while 循环，如何实现？

5.1.3　do…while 语句

do…while 循环的一般格式为：

```
do
  语句;
while(条件);
```

该结构实现“直到型”循环结构，与 while 循环的不同在于：它先执行循环中的语句，然后再判断条件是否为真，如果为真则继续循环；如果为假，则终止循环。因此，do…while 循环至少要执行一次循环语句。同样当有许多语句参加循环时，要用“{”和“}”把它们括起来。

do…while 语句循环结构如图 5.15 所示。

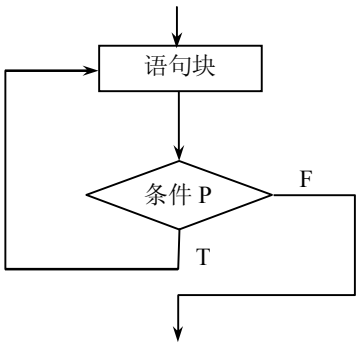

图 5.15　do…while 循环结构图

1. 实验目的和要求

（1）理解循环结构程序段中语句的执行过程。

（2）掌握用 do…while 语句实现循环的方法。

（3）掌握如何正确地设定循环条件，以及如何控制循环的次数。

2. 实验重点和难点

（1）设定循环条件及控制循环次数。

（2）对程序运行前出现的错误进行调试。

3. 实验内容

（1）求 sn=a+aa+aaa+aaaa+aaaaa+…，表达式最后一个数是 5 位，a 是一个数字。

程序流程图如图 5.16 所示。

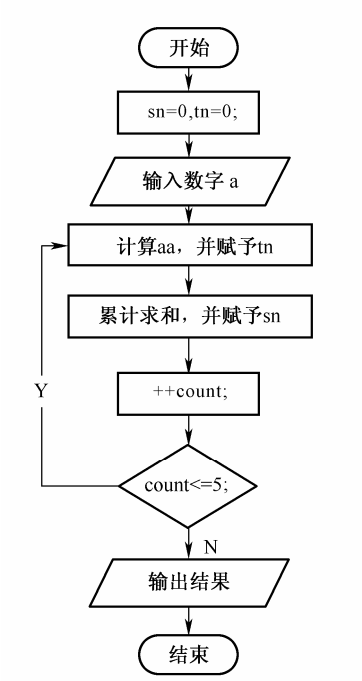

图 5.16　程序流程图

参考程序

```
#include "stdio.h"
void main()
{
    int a,count=1;
    long int sn=0,tn=0;
    printf("please input a number(1~9):");
    scanf("%d",&a);
    do
    {
        tn=tn+a;
        sn=sn+tn;
        a=a*10;
        ++count;
    }
    while (count<=5);
    printf("a+aa+...=%ld\n",sn);
}
```

分析：

观察该题目要求，发现待求和的数据存在一定的规律性，只要能拼合出一系列符合该规律的数据，进行合计运算即可。

为实现拼合数据的要求，设置临时变量 tn，设置初始值为 0。首先将用户输入的整数 a 与 tn 相加求和，然后使用赋值语句“a=a*10”改变 a 的值再与 a 相加求和，这样就可以得到两位数的符合规律的数据，同理可以重复得到不同位数的符合题目规律要求的数据。设置计数器变量 count，执行 do…while 循环时判断 count 是否小于 5，若符合条件，进行累加操作，否则打印最终结果。题目运行结果如图 5.17 所示。

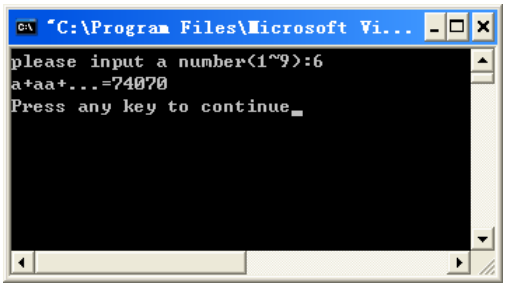

图 5.17　程序运行结果图

（2）编写程序，输入三角型的三条边长，求其面积。要求检查三条边是否满足构成三角形的条件，如不能，给出错误提示。

程序流程图如图 5.18 所示。

参考程序

```
#include "stdio.h"
#include "math.h"
void main()
{
    int flag=0;
```

```
    float a,b,c,s;
    do
    {
        printf("Please enter a b c:");
        scanf("%f %f %f",&a,&b,&c);
        if(a>b+c || b>a+c || c>a+b)
            {
            flag=1;
            printf("您输入的边长度不能构成三角形!\n");
            }
    }
    while(flag);
    s=(a+b+c)/2;
    printf("S=%f\n",s=sqrt(s*(s-a)*(s-b)*(s-c)));
}
```

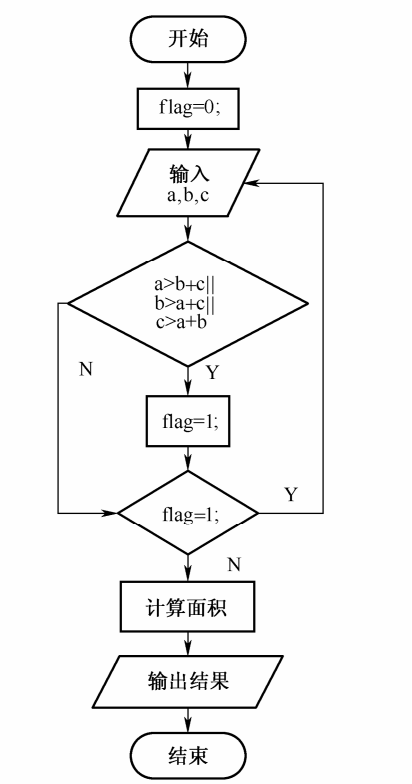

图 5.18　程序运行流程图

分析：

求三角形面积是一个非常简单的题目，但由于题目要求判断三角形三条边的长度是否符合三角形要求，因此使用 do…while 语句实施判断。

设置标志位 flag，并初始化为 0，程序运行时首先输入三条边的长度，如果任意两条边的长度之和小于第三条边的长度，就设置 flag=1，并给出错误提示。判断 flag 是否为 1 决定是否再次执行循环体中的语句，如果为 1 说明输入的三条边不符合要求，重新输入，否则计算面积并输出显示。程序运行结果如图 5.19 和图 5.20 所示。

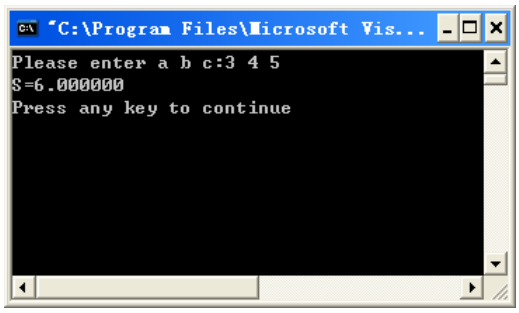

图 5.19　输入正确时程序运行结果图

图 5.20　输入错误时程序运行结果图

课后思考

（1）请修改实验内容（1）中的程序，将循环方式改成 for 循环实现。

（2）将实验内容（2）中的循环改成 while 循环应该如何实现？

5.1.4　多重循环结构

在循环体语句中又包含有另一个完整的循环结构的形式，称为循环的嵌套（又称双重循环）。如果内循环体中又有嵌套的循环语句，则构成多重循环。嵌套在循环体内的循环体称为内循环，外面的循环称为外循环。

while、do…while、for 三种循环都可以互相嵌套。一般的双重循环嵌套形式如下所示：

（1）while ()
```
{  …
   while ()
  …
}
```
（2）for (;;)
```
{
   …
   for (;;)
   …
}
```
（3）do{
```
   …
```
（4）while ()
```
{…
   for (;;)
   …
}
```
（5）for (;;)
```
{
   …
   while (;;)
   …
}
```
（6）do{
```
   …
```

```
do{                                    for (;;);
…                                      …
}while ();                             }while ();
…
}while ();
```

一般的双重循环嵌套流程图如图 5.21 所示。

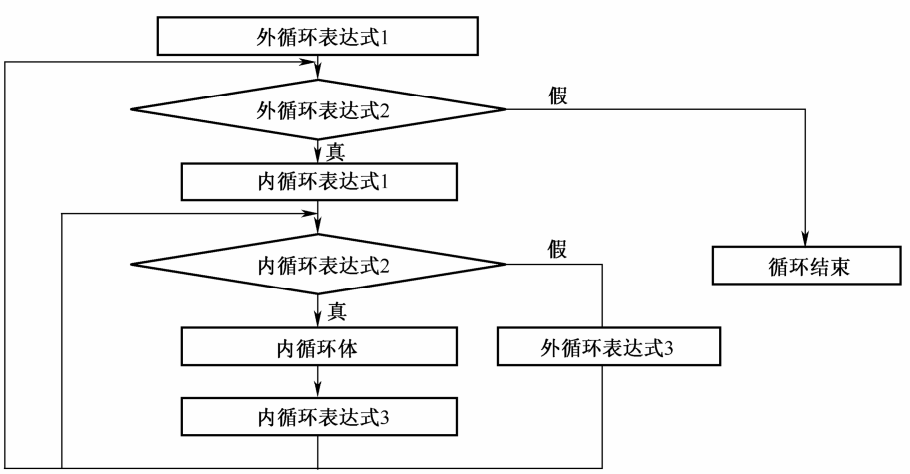

图 5.21　双重循环流程图

1. 实验目的和要求

（1）理解循环嵌套程序段中语句的执行过程。

（2）掌握循环嵌套时各循环变量的设置。

（3）掌握各种双层循环嵌套的适用情况。

2. 实验重点和难点

（1）使用嵌套循环时循环终止条件。

（2）使用不同组合的循环嵌套结构解决问题。

3. 实验内容

（1）求 1!+2!+3!+...+10!。

程序流程图如图 5.22 所示。

参考程序

```
#include <stdio.h>
void main()
{
    int i, j;
    double s = 0, t;
    for (i=1;i<11;i++)
    {
        j=1;
        t=1;
        while (j<i+1)
        {
```

```
                t=t*j;
                j++;
            }
            s=s+t;
        }
        printf( "1!+2!+3!+...+10!=%.2f\n", s );
    }
```

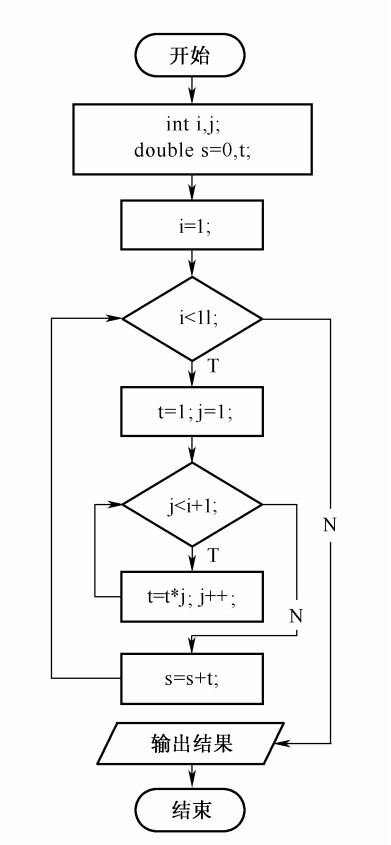

图 5.22　程序执行流程图

分析：

求取阶乘的和是一道经典的 C 语言运算题，由于既要控制累加的次数，又要求出不同数字的阶乘，所以考虑使用循环嵌套。

要求某个数字的阶乘，可以设置结果变量 t，初始值为 1。设置控制循环变量 j 从 1 开始，到该数字结束，循环过程中反复执行 t*j，到循环条件结束时得到阶乘结果。由于题目要求得到从 1 的阶乘加到 10 的阶乘，因此外层循环控制变量的变化范围为 1 到 10，设置累加和变量 s，将内层循环求得的阶乘不断和 s 相加，得到最终结果。程序运行结果如图 5.23 所示。

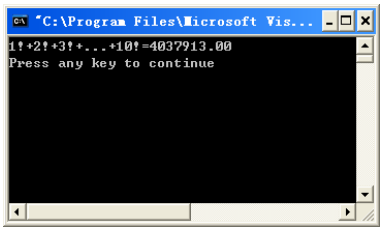

图 5.23　程序运行结果图

（2）求 s=1+(1+2)+(1+2+3)+…+(1+2+…n)，n 要求从键盘输入。

程序流程图如图 5.24 所示。

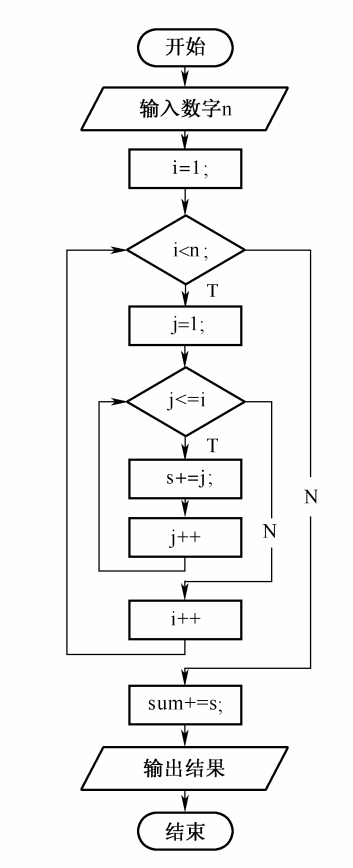

图 5.24　程序运行流程图

参考程序

```
#include "stdio.h"
void main()
{
    int i,j,n,s=0,sum=0;
    printf("请输入一个正整数:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        for(j=1;j<=i;j++)
            s+=j;
    sum+=s;
    printf("1+(1+2)+...+(1+2+...n)=");
    printf("%d\n",sum);
}
```

分析：

题目要求由键盘输入正整数 n，然后计算要求的结果。观察规律，发现给定 n 后，可以通过 n 控制外层循环，实现最终的 n 个数字相加求和。而进一步观察发现，每个参与求和的数据又是一个可以通过循环实现求和的数字，由此确定内层循环，得到最终结果。程序运行效果如图 5.25 所示。

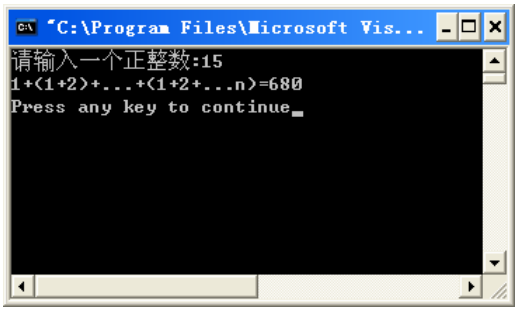

图 5.25　程序运行结果图

课后思考

（1）如果两个实验内容中的加法都改为乘法，应如何修改程序？

（2）实验内容（2）中输入较大的正整数可能造成溢出错误，如何避免，请修改原程序。

5.1.5　break 语句和 continue 语句

break 语句的一般格式：

break;

前面用 break 语句可以使程序流程跳出 switch 结构，继续执行 switch 语句下面的一个语句。实际上，break 语句还可以用于循环结构中，即当流程执行 break 语句时提前结束循环，接着执行循环下面的语句。

break 语句对于减少循环次数，加快程序的执行起着重要的作用。

continue 语句的一般格式：

continue;

continue 语句作用为结束本次循环，直接进行下一轮循环的判断。continue 语句和 break 语句的区别是：continue 语句只是结束本次循环，就是本次循环的 continue 后面其余语句不执行了，接着开始下一轮循环，而不是终止整个循环的执行，而 break 语句则是结束循环，不再进行条件判断。continue 语句只能用于 for，while，do…while 语句中，常与 if 语句配合，起到加速循环的作用。

如果有以下两个循环结构：

（1）while (表达式 1)
{…
if(表达式 2) break;
…}

（2）while (表达式 1)
{…
if(表达式 2) continue;
…}

两种结构的流程如图 5.26 所示，请注意图中当“表达式 2”为真时流程的转向。结构（1）的表达式 2 为真时，直接退出了循环，而结构（2）的表达式 2 为真时，则是退出当前循环进入下一轮循环。

1. 实验目的和要求

（1）理解 break 和 continue 语句的执行过程。

（2）掌握 break 和 continue 语句对应流程图的画法。

（3）掌握 break 和 continue 语句的区别。

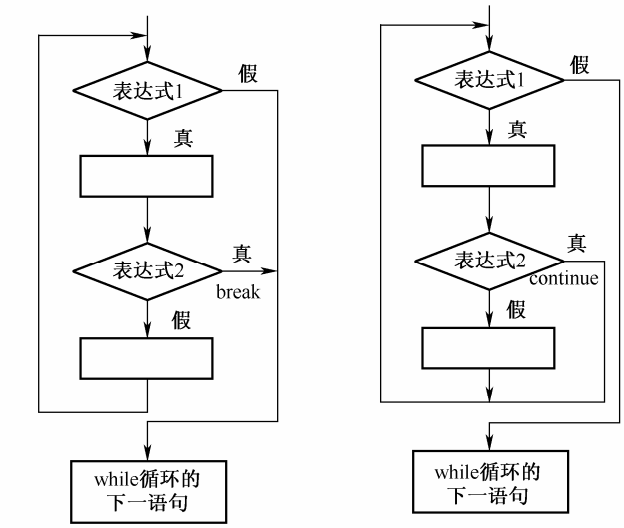

图 5.26　break 语句和 continue 语句流程示意图

2. 实验重点和难点

（1）熟练使用 break 和 continue 语句解决问题。

（2）掌握 break 和 continue 语句的区别。

3. 实验内容

（1）打印 1000 以内个位数字是 6 且能被 3 整除的所有数，要求每行打印 6 个数据。程序流程图如图 5.27 所示。

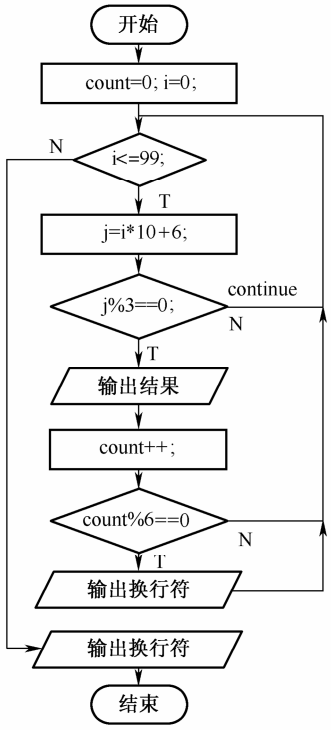

图 5.27　程序运行流程图

参考程序

```
#include "stdio.h"
void main()
{
    int i,j,count=0;
    for (i=0;i<=99;i++)
    {
        j=i*10+6;
        if (j%3!==0)
            continue;
        else
        {
            printf("%d\t",j);
            count++;
            if (count%6==0)
                printf("\n");
        }
    }
    printf("\n");
}
```

分析：

首先观察满足要求数据的特点，1000 以内个位数字为 6，如果将循环设置为从 0 到 1000，则势必采用穷举法尝试，并且在尝试过程中进行末尾数字为 6 的判断，效率比较低。更好的方法是将循环设置为从 0 到 99，在循环体内首先进行扩大 10 倍并且加 6 的运算，这样就可以得到所有满足要求的数据。用这些数据进行整除 3 的判断，一旦满足就打印出来，否则跳过当前数据，继续进行循环体。根据这种特点，采用 continue 语句实现可以满足要求。程序运行结果如图 5.28 所示。

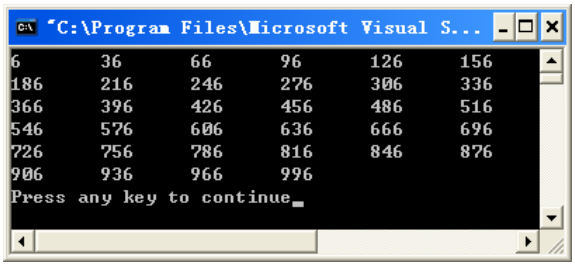

图 5.28　程序运行结果图

（2）编写程序求出 555555 的约数中最大的三位数是多少。

程序流程图如图 5.29 所示。

参考程序

```
#include <stdio.h>
void main()
{
    int j;
    long n=555555; /*使用长整型变量，以免超出整数的表示范围*/
```

```
        for(j=999;j=>100;j--) /*可能取值范围在 999 到 100 之间，j 从大到小*/
            if(n%j == 0 ) /*若能够整除 j，则 j 是约数，输出结果*/
                {
                    printf("The max factor with 3 digits in %ld is: %d.\n",n,j);
                    break; /*控制退出循环*/
                }
}
```

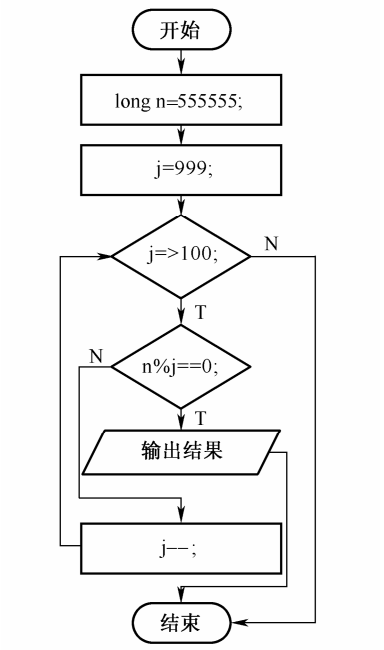

图 5.29　程序运行流程图

分析：

要求出约数中最大的三位数，可以使用穷举法。本例采用 for 循环，为提高求解效率，可以考虑从 999 向 100 逐渐减少进行尝试，一旦得到满足要求的约数，立刻通过 break 语句结束循环，并打印输出结果。运行结果如图 5.30 所示。

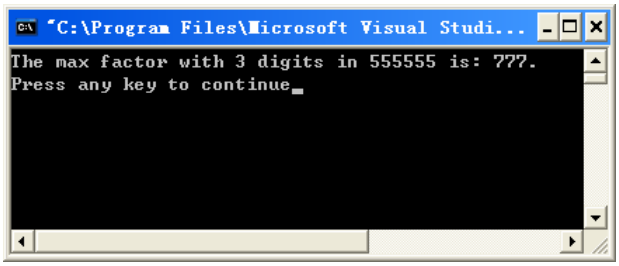

图 5.30　程序运行结果图

课后思考

（1）实验内容（1）中如果要求每行输出 10 个数据，应该如何修改程序？

（2）实验内容（2）中如果要求求取约数中最大的两位数，如何实现？

（3）请尝试将实验内容（2）中的循环改成 while 循环实现。

5.2 举一反三

通过对各种循环结构的学习，应该掌握各种结构的特点及适用条件，以便更好的解决问题。但在实际应用当中，有些问题可以用多种循环结构实现，达到同样的效果，下面就通过一个实例，使用不同的循环结构进行解决，以达到深入认识的目的。

题目要求：计算 100 以内所有素数的和。

参考方法一：

由于题目要求是 100 以内所有素数的和，所以要使用循环进行是否是素数的判断，进而进行求和运算，得到满足要求的结果。而判断某个数据是否为素数，只需要用从 2 到该数一半大小进行整除判断即可，因此解决该题目需使用循环嵌套实现。

参考流程图如图 5.31 所示。

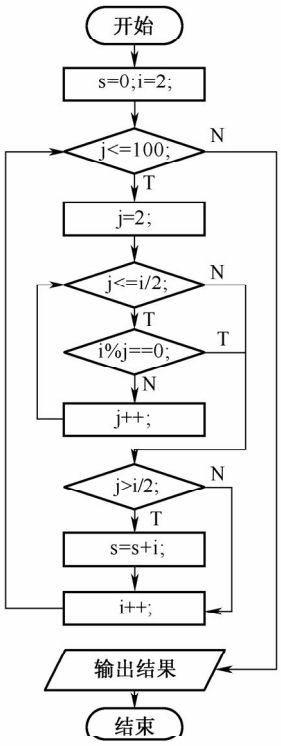

图 5.31 程序运行流程图

参考程序

```
#include <stdio.h>
void main()
{    int i,j,s=0;
     for(i=2;i<=100;i++)          /* 设置循环产生 2～100 之间的数 */
     {
     for(j=2;j<=i/2;j++)          /* 用 2～i/2 的数去除 i */
          if(i%j==0)   break;     /* 有能整除 i 的 j，说明 i 不是素数，退出 */
     if(j>i/2)                    /* i 是素数，因为 2～i/2 没有 i 的因子 */
```

```
            s=s+i;
        }
        printf("100 以内素数之和为：%d\n",s);
    }
```

程序运行结果如图 5.32 所示。

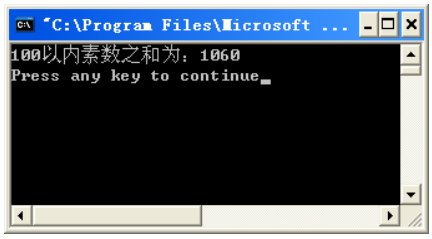

图 5.32　程序运行结果图

参考方法二：

除了使用两个 for 循环求解问题外，还可以使用 while 循环配合 for 循环实现题目的求解，请读者参考流程图 5.31 思考如何实现。

参考程序

```
#include <stdio.h>
void main()
{
    int i,j,s=0;
    i=2;
    while (i<=100)
    {
        for(j=2;j<=i/2;j++)          /* 用 2～i/2 的数去除 i */
            if(i%j==0)   break;      /* 有能整除 i 的 j，说明 i 不是素数，退出 */
        if(j>i/2)                    /* i 是素数，因为 2～i/2 没有 i 的因子 */
            s=s+i;
        i++;
    }
    printf("100 以内素数之和为：%d\n",s);
}
```

参考方法三：

当然在解决问题时也可以不用到 for 循环，而直接使用两个 while 循环达到完成题目要求的功能。请读者在思考该问题后参考如下程序上机实践。

参考程序

```
#include <stdio.h>
void main()
{
    int i,j,s=0;
    i=2;
    while (i<=100)
    {
        j=2;
        while (j<=i/2)
        {
            if(i%j==0)   break;
```

```
                j++;
            }
            if(j>i/2)
                s=s+i;
            i++;
        }
        printf("100 以内素数之和为：%d\n",s);
    }
```

由以上实例的解题思路中可以看出，对不同问题使用多层循环嵌套结构解决问题时，可以适用不同的结构，以上仅仅是其中的一部分解决方案，请同学们思考使用 do…while 循环和 for 以及 while 循环如何嵌套解决该问题。

5.3 程序实例

在本章第一部分程序验证中，各个知识点都通过具体的程序加以验证，并通过程序流程图详细描述了语句执行的先后顺序，对掌握各个知识点的特点及熟练使用这些知识点解决各类实际问题都有非常好的帮助。在本节中将通过两个具体实例对这些内容加以巩固，以达到更好的掌握目的。

（1）编写程序输出 50 以内质数。

程序流程图如图 5.33 所示。

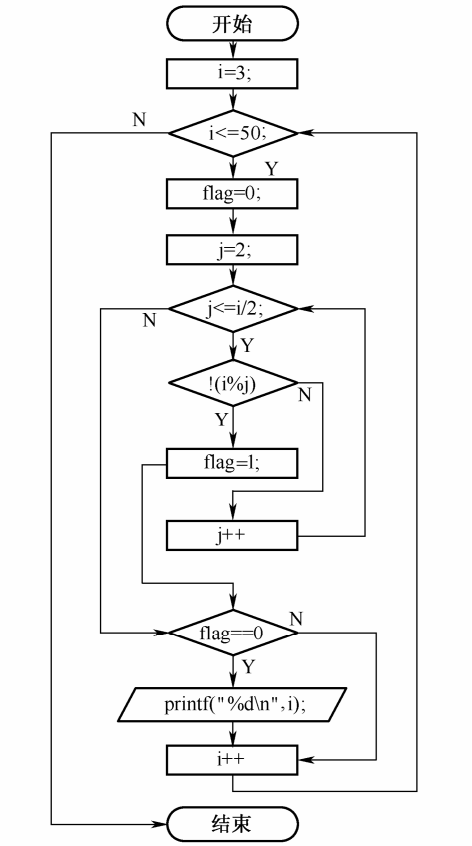

图 5.33 求解 50 以内质数流程图

参考程序

```
#include <stdio.h>                    /*包含输入输出头文件*/
void main()                           /*主函数*/
{
    int flag;                         /*标志是否为质数*/
    printf("2    ");
    for (int i=3;i<=50;i++)           /*从 3 开始数数到 50*/
    {
        flag=0;                       /*初始化，为质数*/
        for (int j=2;j<=i/2;j++)      /*从除以 2 开始，一直除以到 i/2*/
        {
            if (!(i %j) )             /*如果整除（%为取余数）*/
            {
                flag=1;               /*制标志为 1（不为质数）*/
                break;                /*跳到下一个数*/
            }
        }
        if (flag==0) printf("%d\n    ",i);
    }
    printf("\n");
}
```

分析：

题目要求出 50 以内的质数，可以考虑使用穷举法解决问题，设置外层循环变量时从 3 开始，因为 2 是唯一的一个偶数并且为质数，所以不用考虑，在输出结果时直接输出就可以了，外层循环到 50 结束，一一尝试。考虑质数的特点，只能被 1 和其本身整除，比本身小的最大约数应该是该数字的一半，即 n/2。所以内层循环变量可以设置为从 2 到 n/2，逐个考察数字 n 能不能被其中任何一个数字整除，如果有满足条件的数字出现，则说明该数字不是质数，此时可借助 break 语句结束内层循环。本程序运行结果如图 5.34 所示。

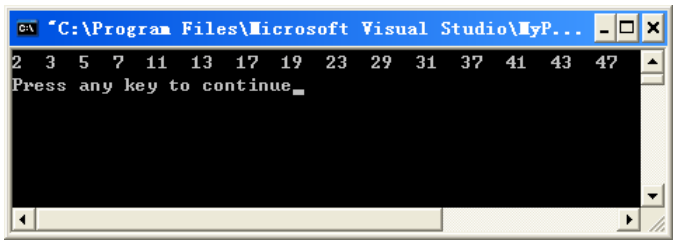

图 5.34　程序运行结果图

（2）从键盘输入 10 个整数，用插入法对输入的数据按照从小到大的顺序进行排序，将排序后的结果输出。

程序流程图如图 5.35 所示。

参考程序

```
#include "stdio.h"
void main()
{
    int i,j,num,a[10];
```

```
    for (i=0;i<10;i++)
    {
        printf("input no. %d:",i+1);
        scanf("%d",&num);
        for (j=i-1;j=>0&&a[j]>num;j--)
            a[j+1]=a[j];
        a[j+1]=num;
    }
    printf("the sorted number:\n");
    for (i=0;i<10;i++)
        printf("%d   ",a[i]);
    printf("\n");
}
```

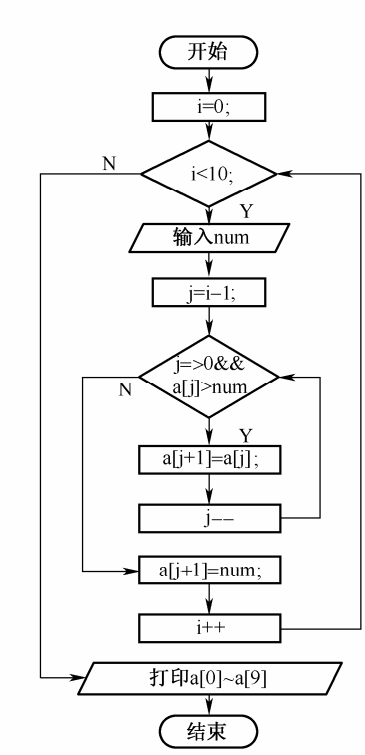

图 5.35　程序运行流程图

分析：

首先分析插入法排序的基本思想。即输入一个数据，检查数组列表中的每个数据，将其插入到一个已经排好序的数列中的适当位置，使数列依然有序，当最后一个数据放入合适位置时，该数组排序完毕。

本题目要求从键盘输入 10 个数据，可以考虑使用 for 循环逐个输入，并设置一个数组存放输入的 10 个数据，在输入数据的过程中直接进行数据比较。例如已经输入的 i-1 个数据是有序数列，输入第 i 个数据时，可以从第 i-1 个数据一直到第一个数据依次和该数比较，如果该数大于比较数据，则继续向前比较，并将这些数据逐个后移，为第 i 个数据腾出插入空间，

直到找到小于该数的数据为止，插入该数据到正确位置即可。程序运行结果如图 5.36 所示。

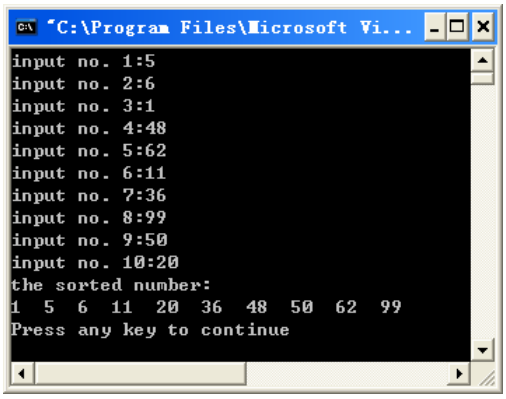

图 5.36　程序运行结果图