

**课 程 实 验 报 告**

**课程名称： C语言程序设计实验**

**专业班级： 2004**

**学 号： U202012098**

**姓 名： 余锦琪**

**指导教师： 唐赫**

**报告日期： 10.19**

**计算机类II**

**目□□录**

[**1□□□表达式和标准输入输出实验 1**](#_Toc404837920)

[1.1□□实验目的 1](#_Toc404837921)

[1.2□□实验内容 1](#_Toc404837922)

[1.3□□实验小结 1](#_Toc404837923)

[**2□□□流程控制实验 2**](#_Toc404837924)

[2.1□□实验目的 1](#_Toc404837921)

[2.2□□实验内容 1](#_Toc404837922)

[2.3□□实验小结 1](#_Toc404837923)

[**3□□□函数与程序结构实验 3**](#_Toc404837929)

[3.1□□实验目的 1](#_Toc404837921)

[3.2□□实验内容 1](#_Toc404837922)

[3.3□□实验小结 1](#_Toc404837923)

[**4□□□编译预处理实验 4**](#_Toc404837934)

[4.1□□实验目的 1](#_Toc404837921)

[4.2□□实验内容 1](#_Toc404837922)

[4.3□□实验小结 1](#_Toc404837923)

[**5□□□数组实验 5**](#_Toc404837938)

[5.1□□实验目的 1](#_Toc404837921)

[5.2□□实验内容 1](#_Toc404837922)

[5.3□□实验小结 1](#_Toc404837923)

[**6□□□指针实验 6**](#_Toc404837943)

[6.1□□实验目的 1](#_Toc404837921)

[6.2□□实验内容 1](#_Toc404837922)

[6.3□□实验小结 1](#_Toc404837923)

[**7□□□结构与联合实验 7**](#_Toc404837948)

[7.1□□实验目的 1](#_Toc404837921)

[7.2□□实验内容 1](#_Toc404837922)

[7.3□□实验小结 1](#_Toc404837923)

[**8□□□文件实验 8**](#_Toc404837953)

[8.1□□实验目的 1](#_Toc404837921)

[8.2□□实验内容 1](#_Toc404837922)

[8.3□□实验小结 1](#_Toc404837923)

[**参考文献 9**](#_Toc404837957)

# 实验1 表达式和标准输入输出实验

**1.1 实验目的**

（1）熟练掌握各种运算符的运算功能，操作数的类型，运算结果的类型及运算过程中的类型转换，重点是C语言特有的运算符，例如位运算符，问号运算符，逗号运算符等；熟记运算符的优先级和结合性。

（2）掌握getchar, putchar, scanf 和printf 函数的用法。

（3）掌握简单C程序的编写方法。

（4）熟悉C语言程序的开发环境，并学会调试程序的方法。

**1.2 实验内容及要求**

**1、程序改错与跟踪调试**

下面的实验1-1程序用来完成以下任务：

（1）输入华氏温度f，将它转换成摄氏温度c后输出。

（2）输入圆的半径值ｒ，计算并输出圆的面积ｓ。

（3）将ｋ的高字节作为结果的低字节，ｐ的高字节作为结果的高字节，拼成一个新的整数后输出。

在这个程序中存在若干语法和逻辑错误，要求先编译程序改正语法错误，再采用单步执行的方式调试程序找出逻辑错误。在单步执行程序的过程中，观察以下变量值：

1. 执行完c = 5/9 \* (f-32)，c的值为多少？
2. 执行完scanf(“%f”, &r)，r的值为多少？
3. 执行完newint = p&0xff00|k>>8， newint的值是多少？表达式k>>8的值是多少？

根据观察结果分析代码并修改程序，使之能够正确完成指定任务。

/\*实验1-1程序改错与跟踪调试题源程序\*/

1 #include<stdio.h>

2 #define PI 3.14159;

3 int main( void )

4 {

5 int f ;

6 short p, k ;

7 double c, r, s ;

8

9 /\* 任务1 \*/

10 printf("Input Fahrenheit: " ) ;

11 scanf(“%d”, f ) ;

12 c = 5/9 \* (f-32) ;

13 printf( “ \n %d (F) = %.2f (C)\n\n ”, f, c ) ;

14

15 /\* 任务2 \*/

16 printf("input the radius r:");

17 scanf("%f", &r);

18 s = PI \* r \* r;

19 printf("\nThe acreage is %.2f\n\n",&s);

20

21 /\* 任务3 \*/

22 k = 0xa1b2, p = 0x8432;

23 newint = p&0xff00|k>>8;

24 printf("new int = %#x\n\n",newint);

25 return 0;

26}

**解答：**

（1）错误修改：

1) 第2行的符号常量定义后不能有分号，正确形式为：

#define PI 3.14159

2) 第11行的scanf()中变量前应有一个&，正确形式为：

scanf("%d",&f);

3）第12行的整型相除会只保留整数部分，正确形式为：

c=5./9\*(f-32);

4）第19行的prinf()函数中的待打印值不能用&，正确形式为：

printf("\nThe acreage is %.2f\n\n",s);

5）第24行的newint变量没有声明，应在第1行声明变量：

 int f,newint;

**.............**

（2）错误修改后运行结果：

文本

描述已自动生成

**1.2.2 源程序修改替换**

下面的程序利用常用的中间变量法实现两数交换，请改用不使用第3个变量的方法实现。该程序中t是中间变量，要求将定义语句中的t删除，修改下划线处的语句，使之实现两数对调的操作。

#include <stdio.h>

int main( )

{

int m, n, k, p, i, d;

printf("input m, n \n");

scanf("%d%d", &m, &n);

if (m<n) /\* 交换m和n \*/

{

int t;

t = m;

m = n;

n = t;

}

k = 0;

while (m%2 == 0 && n%2 == 0) /\* m和n均为偶数 \*/

{

m /= 2; /\* 用2约简m和n \*/

n /= 2;

k++;

}

for (p = 1; i = 0; i<k; i++) p \*= 2; /\* 求p=2k \*/

while((d=m-n)!=n)

{

if(d>n) m = d;

else

{

m = n;

n = d;

}

}

d \*=p;

printf("the greatest common divisor : %d", d);

return 0;

}

**解答：**

替换后的程序如下所示：

#include<stdio.h>

int main()

{

    int m,n,k,p,i,d;

    printf("input m,n\n");

    scanf("%d%d",&m,&n);

    getchar();

    if(m<n)

    {

        n=m^n;

        m=n^m;

        n=m^n;

     }

    k=0;

     while(m&1==0 && n&1==0)

     {

         m/=2;

         n/=2;

        k++;

     }

     for(p=1,i=0;i<k;i++) p=p<<1;

     while ((d=m-n)!=n)

      {

         if(d>n)m=d;

         else

         {

             (m=n)=d;

          }

      }

    d\*=p;

    printf("the greatest common divisor:%d",d);

    getchar();

    return 0;

}

运行结果如下：

文本

描述已自动生成

方案可行性：

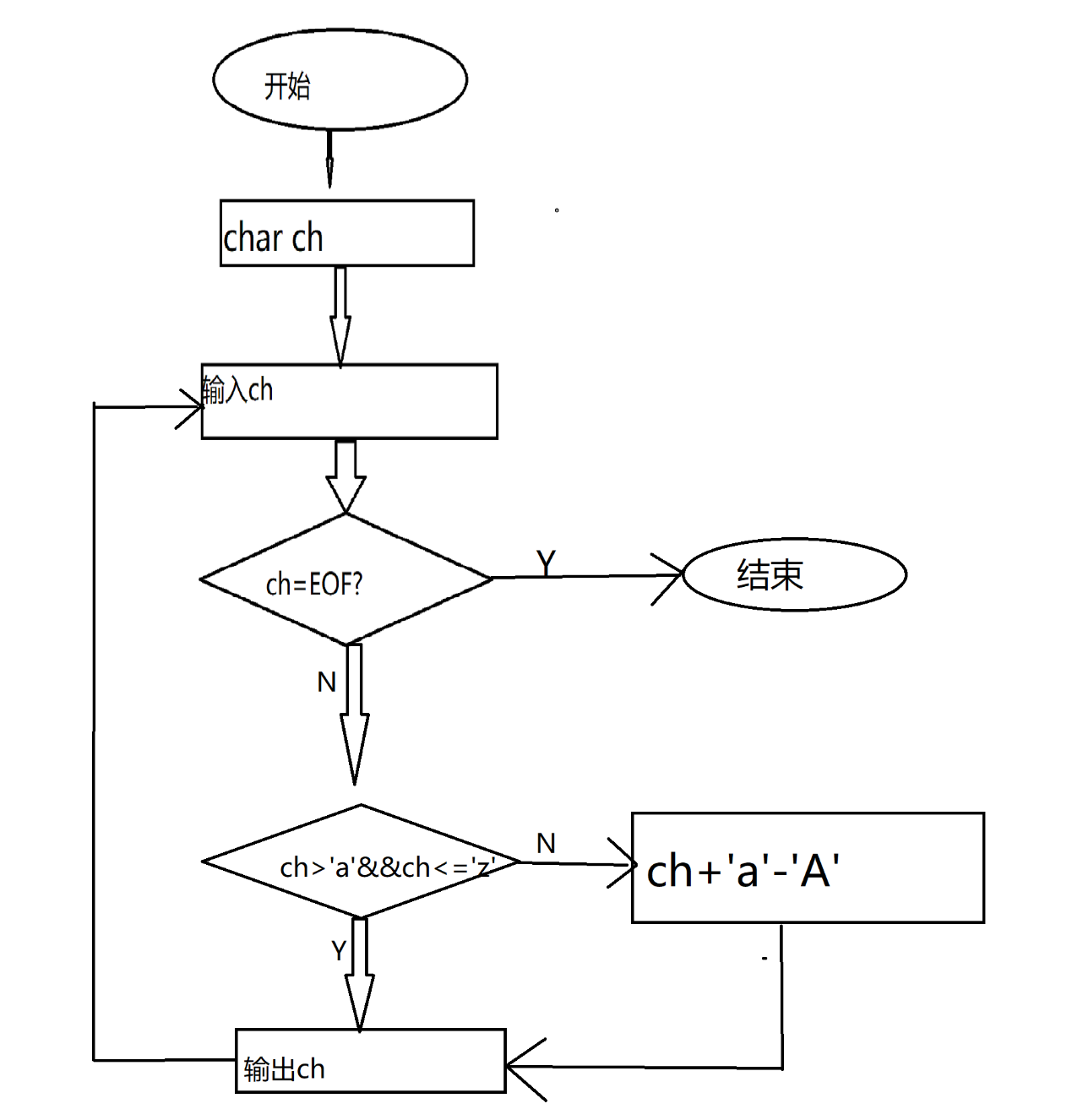
该方案相当于m=m^n^m,表达式的值是n与m按位加后再与m按位加，也就是n按位加m两次，等价于n^(m^m),显然m^m==0，则n^0==n;最后给m赋值为n；同理；在给n赋值时m已经变为了n的初始值，此时n=n^m^n,将m的值赋给了n。

**1.2.3 程序设计**

**（1）**编写一个程序，输入字符ｃ，如果ｃ是大写字母，则将ｃ转换成对应的小写，否则ｃ的值不变，最后输出ｃ。

**解答：**

1. 算法流程如图1.1所示。

 图1.1

2）源程序清单

#include<stdio.h>

int main()

{

    char ch;

    while ((ch=getchar()) !=EOF)

    {

        putchar((ch>'a'&&ch<='z')?ch:ch+'a'-'A');

        printf("\n");

    }

}

3）测试

A）测试数据：A C c

B) 对应测试数据的运行结果截图

图片包含 应用程序

描述已自动生成

（2）编写一个程序，输入无符号短整数x，ｍ，ｎ（0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ）,取出x从第ｍ位开始向左的ｎ位（ｍ从右至左编号为0～15），并使其向左端（第15位）靠齐

**解答：**

1) 解题思路：见程序清单注释

2）程序清单

#include<stdio.h>

int main()

{

    unsigned short int x,m,n;

unsigned short int i,l,r;

/\*逻辑尺 i表示第m+n位，l表示第i位的left，r表示第i位的right\*/

    printf("input:x,m,n\n");

    scanf("%hx %hu %hu",&x,&m,&n);

    getchar();

    if (m>=0&&m<=15&&n>=1&&n<= 16-m)//判断m，n的范围

    {

        i=0x0001;

        l=0xffff;

        r=0xffff;         /\*逻辑尺对应好位置\*/

        i<<=m+n;          /\*从m位开始向左n位\*/

        i&=x;             /\*取出第m+n位\*/

        i<<=15-m-n;       /\*向左靠齐\*/

        l<<=m+n+1;

        l&=x;             /\*取出第m+n位左边的所有位\*/

        l>>=1;            /\*将left的位全向右移一位\*/

        r>>=15-(m+n-1);   /\*取出第m+n位右边的所有位\*/

        x=((i&x)|l|(r&x));/\*将所有位组合到一起\*/

        printf("output:%#hx",x); /\*显示输出\*/

    }

    else

    {

        printf("error");/\*根据m，n的范围报错\*/

    }

    getchar();

    return 0;

}

1. 测试

（a） 测试数据：

叙述选择测试数据的方法。。。如表1-1所示。

表1-1 编程题3的测试数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | | 理 论 结 果 | 运 行 结 果 |
| X | m | N |
| 用例1 | 0100 0110 1000 0000（4680） | 7 | 4 | 计算结果0010 0110 1000 0000 即0x2680 | 0x2680cd 或 截图 |
| 用例2 | 1101 0101 1000 0011（D583） | 16 | 1 | 输入错误（m值超范围） |  |
| 用例3 | 1101 0101 1000 0011（D583） | 13 | 5 | 输入错误（n值超范围） |  |
|  |  |  |  |  |  |

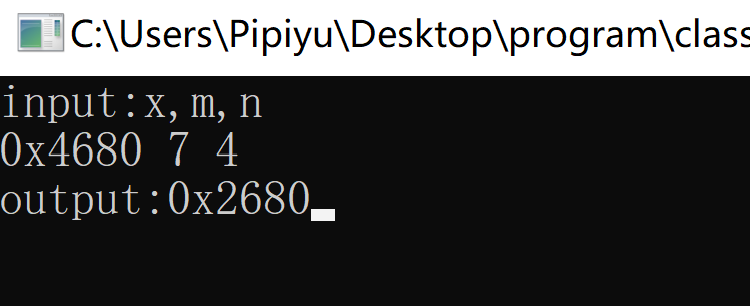
（b） 对应测试测试用例1的运行结果如图1-2所示。

图1-2 编程题3的测试用例一的运行结果

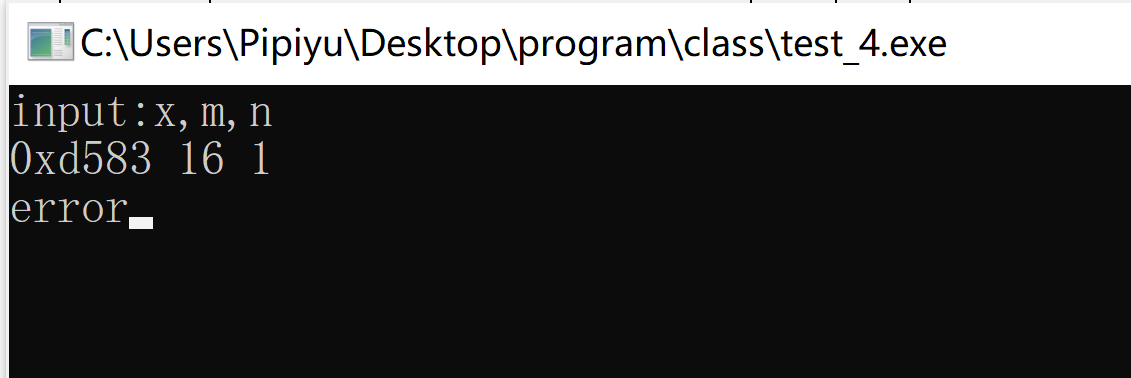
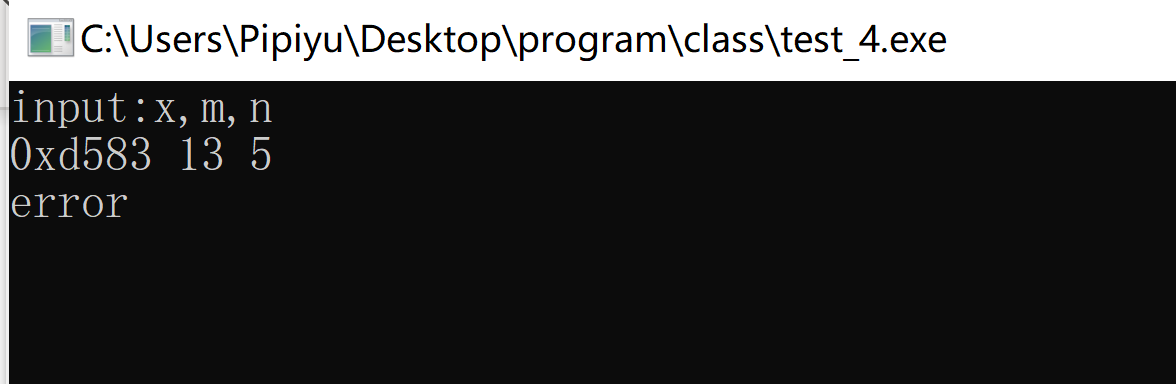
对应测试测试用例2的运行结果如图1-3所示。

图1-3 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图1-4所示。

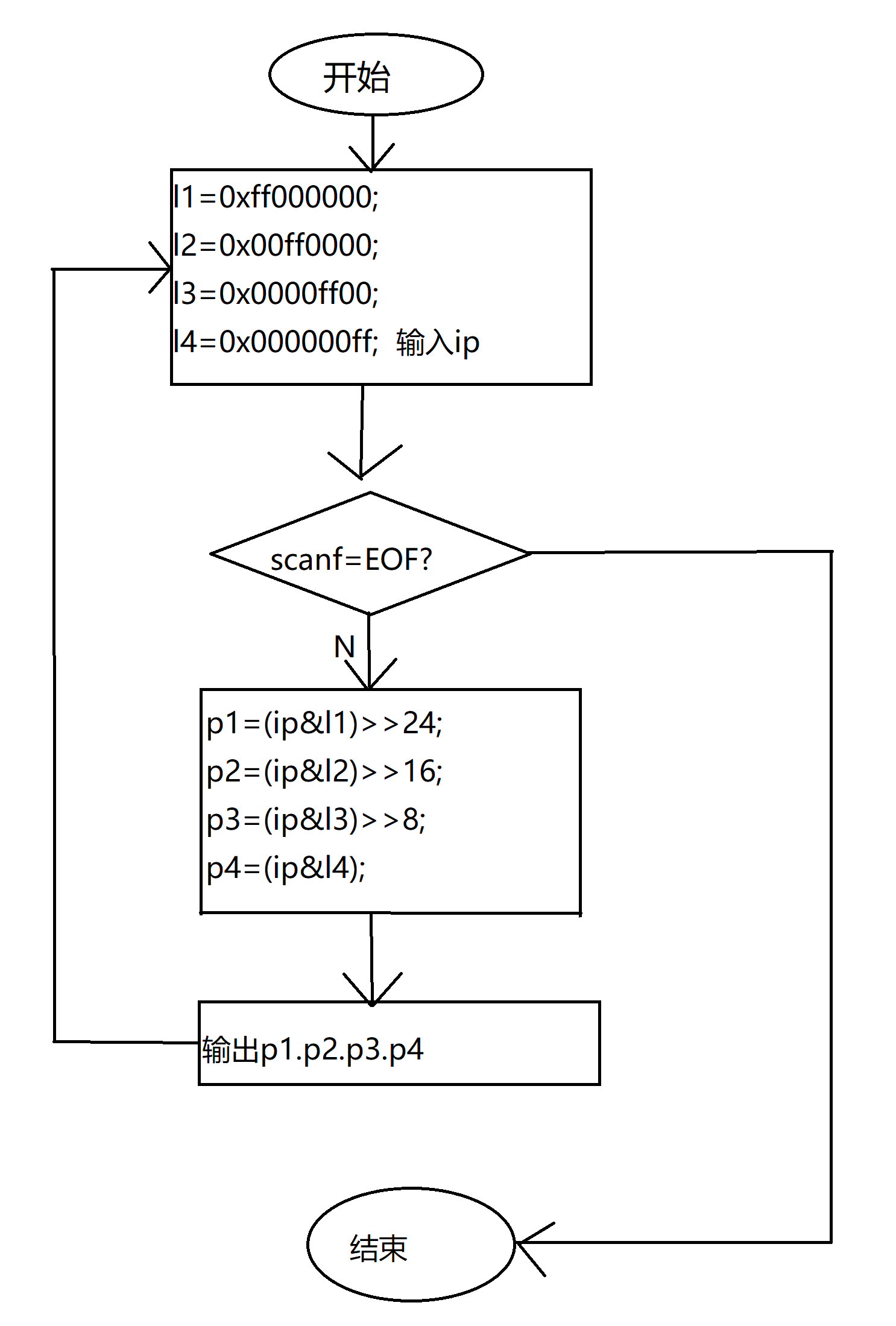
1-4 编程题3的测试图用例三的运行结果

该程序将第m位左边的第n位取出，向左靠齐，左边的数字然后向右补齐位数。在例一中，将第十一位的0向左靠齐，左边的位向右补齐，得0010 0110 1000 0000，即0x2680，程序正确。

（3）IP地址通常是4个用句点分隔的小整数（即点分十进制），但这些地址在机器中是用一个无符号长整型数表示的。例如3232235876，其机内二进制表示就是11000000 10101000 00000001 01100100，按照8位一组用点分开，该IP地址就写成192.168.1.100。

读入无符号长整型数表示的互联网IP地址，对其译码，以常见的点分十进制形式输出。要求循环输入和输出，直至输入Ctrl+Z结束。

1. 解题思路：按照八位一组分别把四组数取出来，并令其靠右，可以得到四个数字，再将四个数字按顺序打印出来，并在中间用点隔开



1. 程序清单：

#include<stdio.h>

int main()

{

    unsigned long int ip,l1,l2,l3,l4,p1,p2,p3,p4;

    printf("Please iuput\n");

    while (scanf("%lu",&ip) !=EOF)

    {

        l1=0xff000000;

        l2=0x00ff0000;

        l3=0x0000ff00;

        l4=0x000000ff;

        p1=(ip&l1);

        p1>>=24;

        p2=(ip&l2);

        p2>>=16;

        p3=(ip&l3);

        p3>>=8;

        p4=(ip&l4);

        printf("address:%lu.%lu.%lu.%lu\n",p1,p2,p3,p4);

    }

    return 0;

}

1. 测试
2. 测试数据

表1-2 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| Ip |  |  |
| 用例1 | 3232235876 | 计算结果192.168.1.100 |  |

（b）对应测试数据的运行结果截图



**4. 选做题**

某加密算法对数据按字节进行加密，具体为：对字节的8个二进制位从右向左用0～7编号，先将0、2、4位分别与1、3、5位两两对应交换，接着对0～5位进行循环左移（左边移出的位接在右边），循环左移的位数有6、7两位的值决定。例如，若6、7位组成的二进制数为01，则将0～5位左移1位，最后得到加密结果，如图1-1所示。

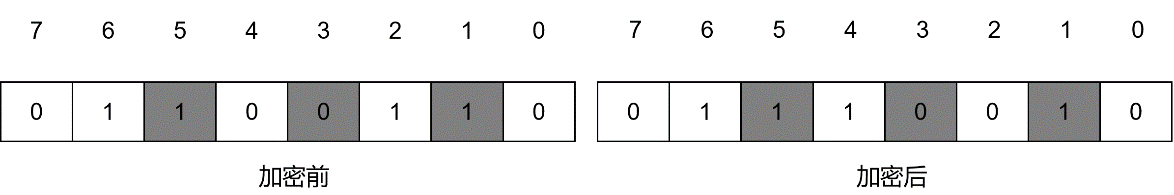


图1-1 加密示意图

输入一行明文字符串，按该算法进行加密后输出密文。例如，输入“abcd”，则输出“dbfp”。

1. 解题思路：输入字符串，再用指针和for循环对字符串中的字符依次操作，操作过程如下：

1.先用逻辑尺将0,2,4位与1,3,5位都取出来，再相应的交换位置，取出67位，然后将分解出来的数重新组合在一起，完成了对应位置的交换。然后我们实现循环左移。

2.将已取出的67位向右端靠齐，得到需要左移的位数，改左移的位数赋值在j上，再将重新组合过一次的字符串的0~5位取出，通过表达式

((mask&num[i])<<j|(mask&num[i])>>6-j)&mask

实现了循环左移，其中mask为0x3f;二进制表示为00111111，可以将字符的0~5位取出。

3．最后将循环左移后的0~5位数与6~7位数重新组合，实现了数据的加密。

4.程序依次对每个数据进行加密后，再将新组成的字符串输出。

2）源程序代码：

#include<stdio.h>

int main()

{

    unsigned char num[5];  /\*待加密数\*/

    unsigned char n0,n1,n2,n3,n4,n5,n6,mask;  /\*逻辑尺\*/

    unsigned char i,t,j;  /\*中间量\*/

    printf("Input:");

    scanf("%s",&num);

    getchar();

    for (i = 0;i<=5; i++)

    {

        n0=0x1;

        n1=0x2;

        n2=0x4;

        n3=0x8;

        n4=0x10;

        n5=0x20;

        n6=0xc0;

        num[i]=(n0&num[i])<<1|(n1&num[i])>>1|(n2&num[i])<<1|(n3&num[i])>>1|(n4&num[i])<<1|(n5&num[i])>>1|(n6&num[i]);//这一行太长了，word放不下

        j=(num[i]&n6)>>6;

        mask=0x3f;

        t=((mask&num[i])<<j|(mask&num[i])>>6-j)&mask;

        num[i]=j<<6 |t;

    }

    printf("Output:%s",num);

    getchar();

    return 0;

}

3）测试

A）测试数据：

表1-3 编程题4的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
|  |  |  |
| 用例1 | abcd | dbfp | dbfp |



**1.3 实验小结**

在解答第二题时，无法实现按下CtrlZ停止输入，通过自己阅读书籍知道了使用whlie循环，并设置whlie循环的终止条件为getchar！=EOF实现了停止操作。

在解答选做题时，对单个字符的数据操作已经到位，但是无法做到对整个字符串进行依次操作，这个时候通过翻阅实验书，了解到通过for循环和指针的操作，可以一次性对所有输入的字符依次进行操作，最后实现了对字符的加密。

体会：本次实验让我加深了对位运算和循环的理解，同时也让我第一次接触到指针，原本的理论知识正在一步步变为实践。

但回过头看，代码的简洁度不够，对循环的理解还有不到位，以后还需多多实践，汲取更多经验。

**2 流程控制实验**

**2.1、实验目的**

（1）掌握复合语句、if语句、switch语句的使用，熟练掌握for、while、do-while三种基本的循环控制语句的使用，掌握重复循环技术，了解转移语句与标号语句。

（2）练习循环结构for、while、do-while语句的使用。

（3）练习转移语句和标号语句的使用。

（4）使用Turbo C 2.0集成开发环境中的调试功能：单步执行、设置断点、观察变量值。

**2.1、实验内容**

**1．源程序改错题**

下面的实验2-1程序是合数判断器（合数指自然数中除了能被1和本身整除外，还能被其它数整除的数），在该源程序中存在若干语法和逻辑错误。要求对该程序进行调试修改，使之能够正确完成指定任务。

/\* 实验2-1改错题程序：合数判断器\*/

1 #include <stdio.h>

2 int main( )

3 {

4 int i, x, k, flag = 0;

5 printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

6 while (scanf("%d", &x) !=EOF) {

7 for(i=2,k=x>>1;i<=k;i++)

8 if (!x%i) {

9 flag = 1;

10 break;

11 }

12 if(flag=1) printf("%d是合数", x);

13 else printf("%d不是合数", x);

14 }

15 return 0;

16 }

**解答：**

（1）错误修改：

1)第4行flag应在循环中初始化，正确形式为：

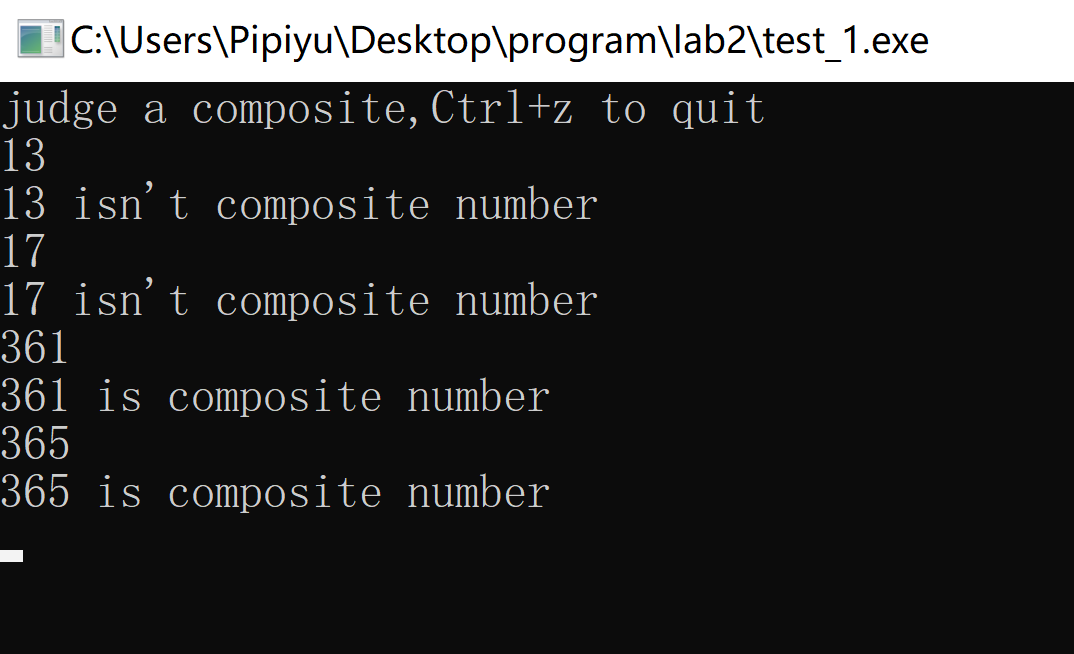
4 int i,x k,flag;

7 for(i=2,flag=0,k=x>>1;i<=k;i++)

2) 第12行的控制表达式中应有两个等号，正确形式为：

12 if(flag==1) printf("%d是合数", x);

（2）错误修改后运行结果：



输入一个数，用小于该数一半且大于2的数依次对该数求模，如果存在模为0，即能整除该数，那么进入if语句，flag=1，最后打印出该数是一个合数；反之，如果不存在模为0的情况，则该数为质数。

**2．程序修改替换**

#include<stdio.h>

int main()

{

    int i,x,k,flag;

    printf("judge a composite,Ctrl+z to quit\n");

    while (scanf("%d",&x)!=EOF)

    {

        for(i=2,flag=0,k=x>>1;i<=k;i++)

            if (!(x%i)){

                flag=1;

                break;

            }

        if (flag==1)

            printf("%d is composite number\n",x);

        else

            printf("%d isn't composite number\n",x);

    }

    return 0;

}

（1）修改实验2-1程序，将内层两出口的for循环结构改用单出口结构，即不允许使用break、goto等非结构化语句。

修改后程序如下：

#include<stdio.h>

int main()

{

    int i,x,k,flag;

    printf("judge a composite,Ctrl+z to quit\n");

    while (scanf("%d",&x)!=EOF)

    {

        for ( i=2,flag=0,k=x>>1;i<=k&&flag==0;i++){

            if (!(x%i))

                flag=1;

        }

            if (flag==1)

                printf("%d is a composite\n",x);

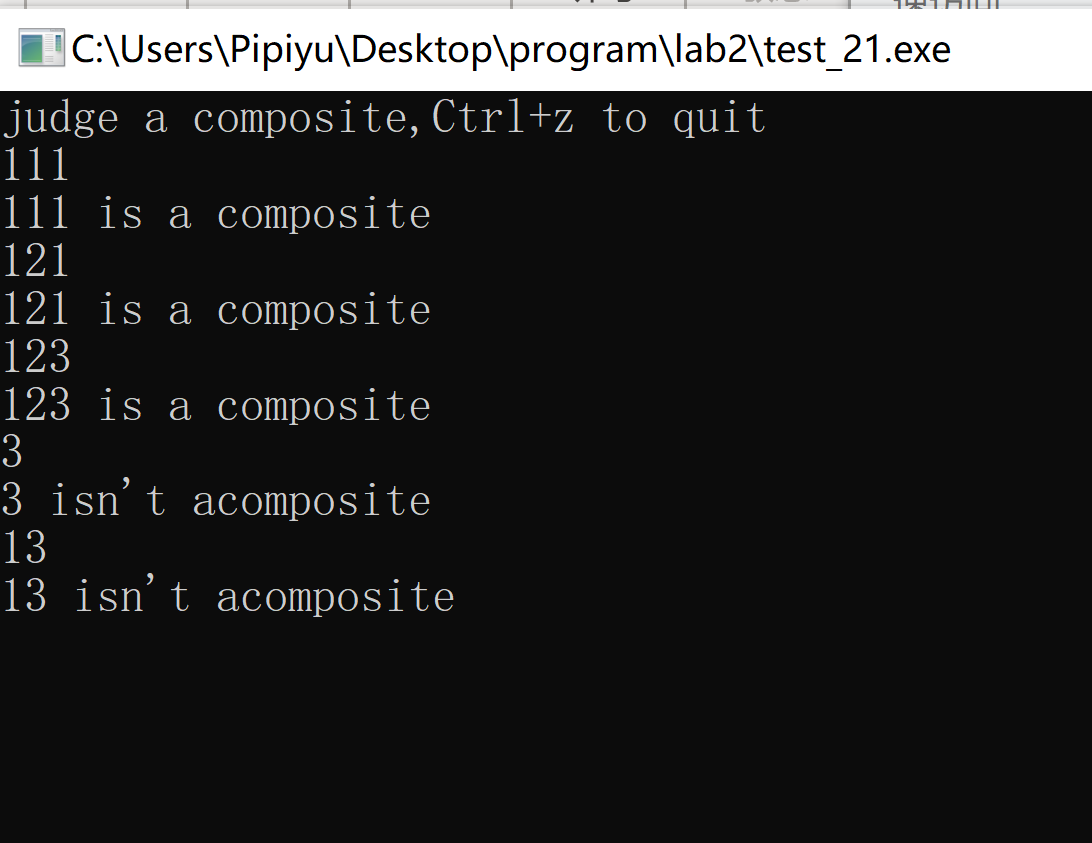
            else

                printf("%d isn't acomposite\n",x);

    }

    return 0;

}



将break去掉后，在for循环的测试条件中加入flag==0；也即当进入for循环一次后，flag立即变为1，退出循环，与使用break语句效果相同。

（2）修改实验2-1程序，将for循环改用do-while循环。

修改后程序如下：

#include<stdio.h>

int main()

{

    int i,x,k,flag;

    printf("judge a composite,Ctrl+z to quit\n");

    while (scanf("%d",&x)!=EOF)

    {

        i=2;

        flag=0;

        k=x>>1;

        do

        {

            if (!(x%i))

            {

                flag=1;

            }

            i++;

        }while (i<=k&&flag==0);

        if (flag==1)

            printf("%d is composite number\n",x);

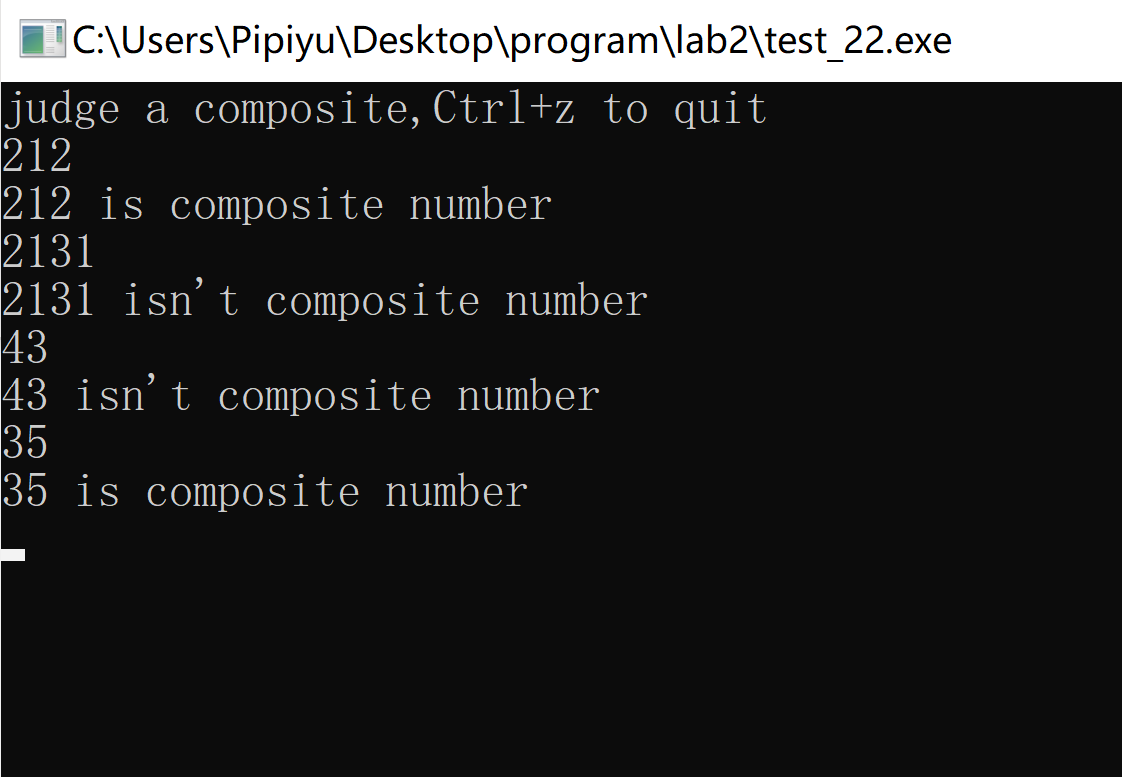
        else

            printf("%d isn't composite number\n",x);

    }

    return 0;

}



将初始化的表达式放在do whlie循环前面，效果与for循环一样。

（3）修改实验2-1程序，将其改为纯粹合数求解器，求出所有的3位纯粹合数的个数。一个合数去掉最低位，剩下的数仍是合数；再去掉剩下的数的最低位，余留下来的数还是合数，这样反复，一直到最后剩下一位数仍是合数，这样的数称为纯粹合数。

修改后程序如下：

#include<stdio.h>

int main()

{

    int x,i,j,p,k,m,n,flag,num,y,z;

    num=0;

    flag=0;

    for (x=400;x<=999;x++)

    {

        for (i=2,k=x>>1;i<=k&&flag==0;i++)

            if(!(x%i))

                flag=1;

        for (j=2,y=x/10,m=y>>1;j<=m&&flag==1;j++)

            if(!(y%j))

                flag=2;

        for (p=2,z=x/100,n=z>>1;p<=n&&flag==2;p++)

            if (!(z%p))

                flag=3;

        if(flag==3){

            num++;

        }

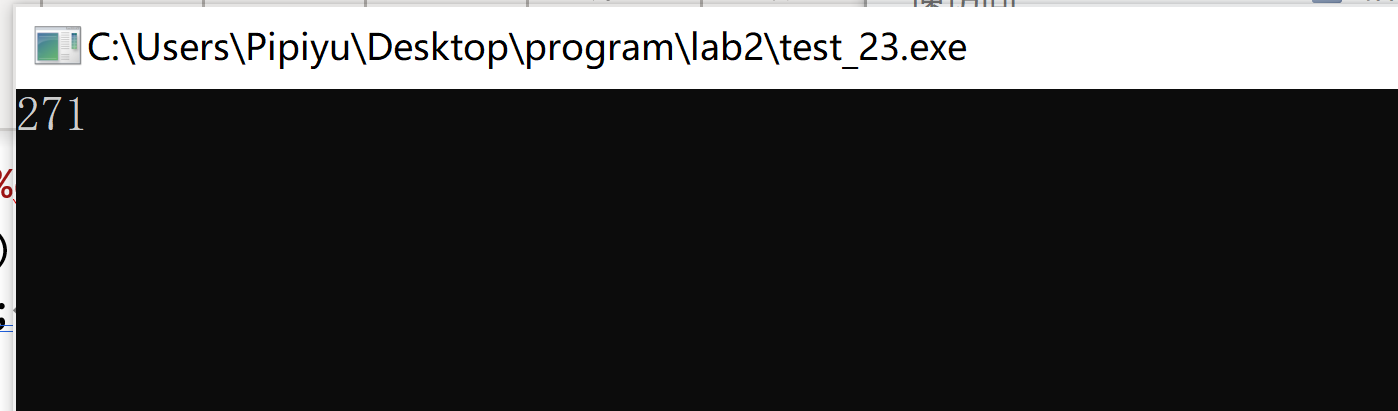
        flag=0;

    }

    printf("%d",num);

    getchar();

    return 0;

}

按照原程序的方法验证一个合数，然后依次验证该合数去掉最低位数是不是合数，如果都是合数，那么num加1，最后得到的num就是纯粹合数的个数。

**3．程序设计**

（1） 假设工资税金按以下方法计算：x ＜ 1000元，不收取税金；1000 ≤ x ＜ 2000，收取5%的税金；2000 ≤ x ＜ 3000，收取10%的税金；3000 ≤ x ＜ 4000，收取15%的税金；4000 ≤ x ＜ 5000，收取20%的税金；x＞5000，收取25%的税金。（注意税金的计算按照阶梯计税法，比如，工资为4500，那么税金=1000\*5% + 1000\*10% + 1000\*15% + 501\*20%）。编写一个程序,输入工资金额，输出应收取税金额度，要求分别用if语句和switch语句来实现。

**解答：**

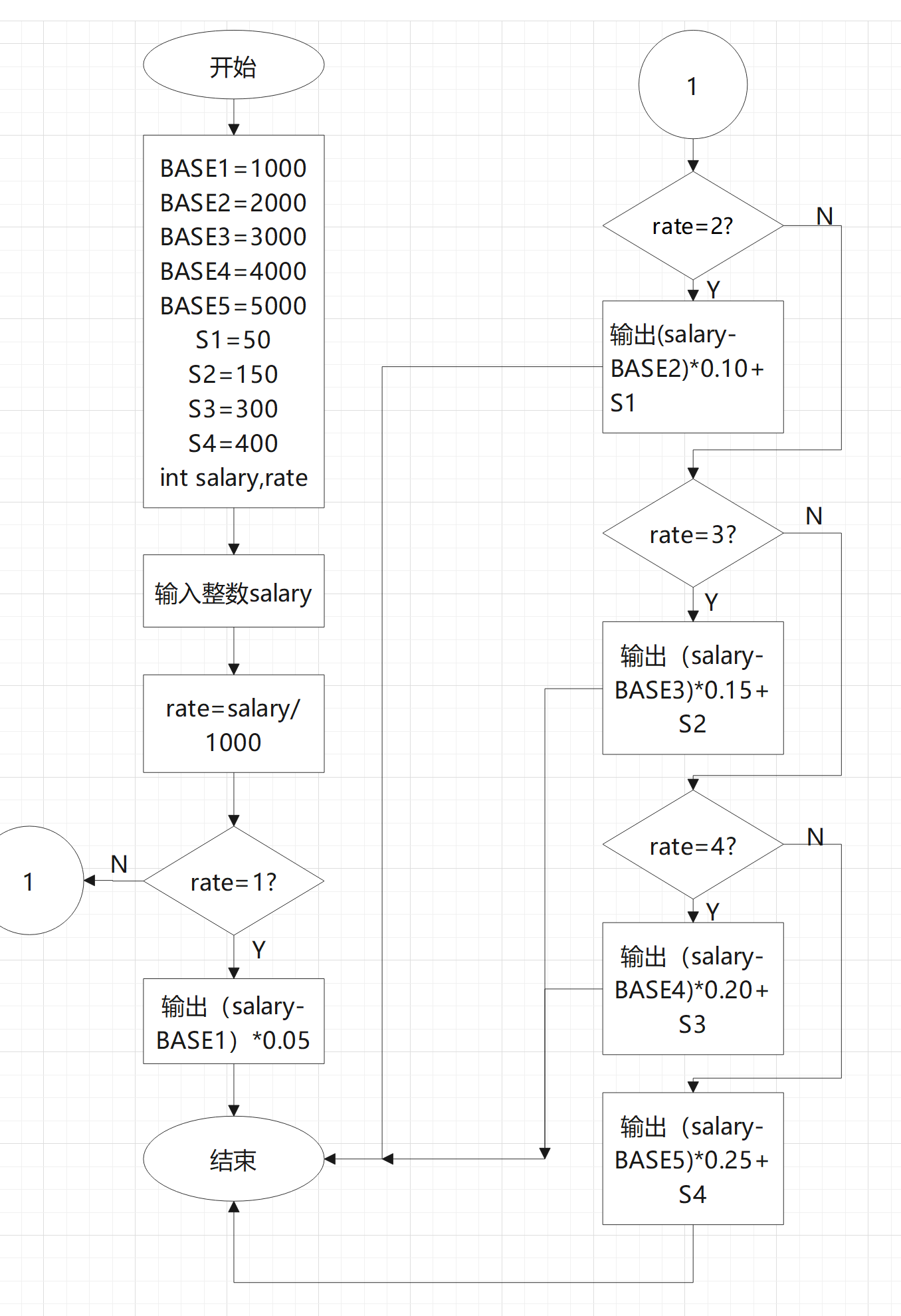
1. 算法流程如图2.1所示。

图2-1 编程题1的程序流程图

2)程序清单：

#include<stdio.h>

#define BASE1 1000

#define BASE2 2000

#define BASE3 3000

#define BASE4 4000

#define BASE5 5000

#define S1 50

#define S2 150

#define S3 300

#define S4 500

int main()

{

    int salary;

    int rate;

    printf("input your salary:");

    while (scanf("%d",&salary)!=!EOF)

    {

        rate=salary/1000;

        if (rate==0)

            printf("0");

        else if (rate==1)

            printf("%.2f\n",(salary-BASE1)\*0.05);

        else if (rate==2)

            printf("%.2f\n",(salary-BASE2)\*0.10+S1);

        else if (rate==3)

            printf("%.2f\n",(salary-BASE3)\*0.15+S2);

        else if (rate==4)

            printf("%.2f\n",(salary-BASE4)\*0.20+S3);

        else

            printf("%.2f\n",(salary-BASE5)\*0.25+S4);

    }

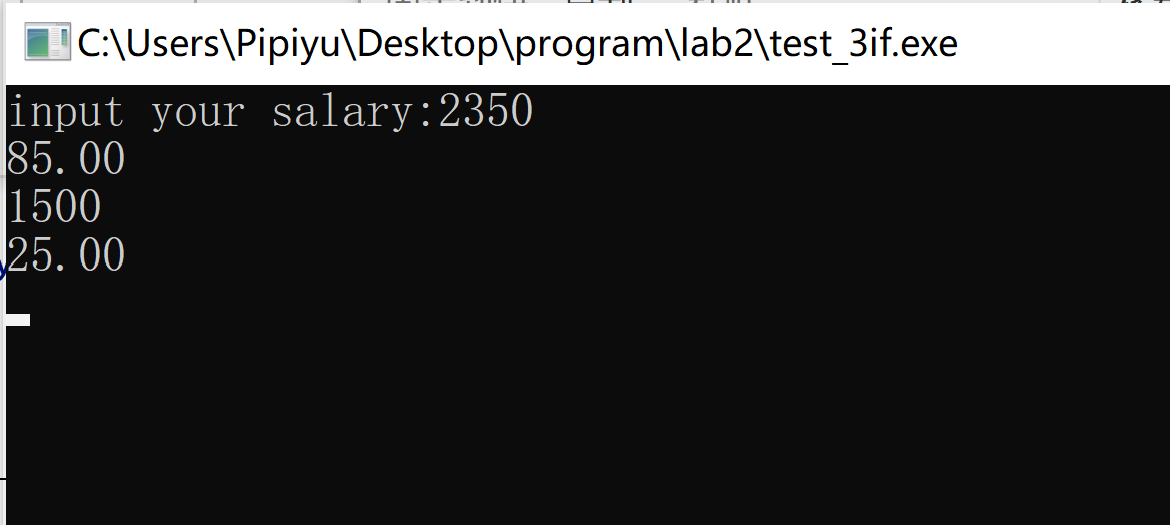
    return 0;

}

3）测试数据

A）数据输入

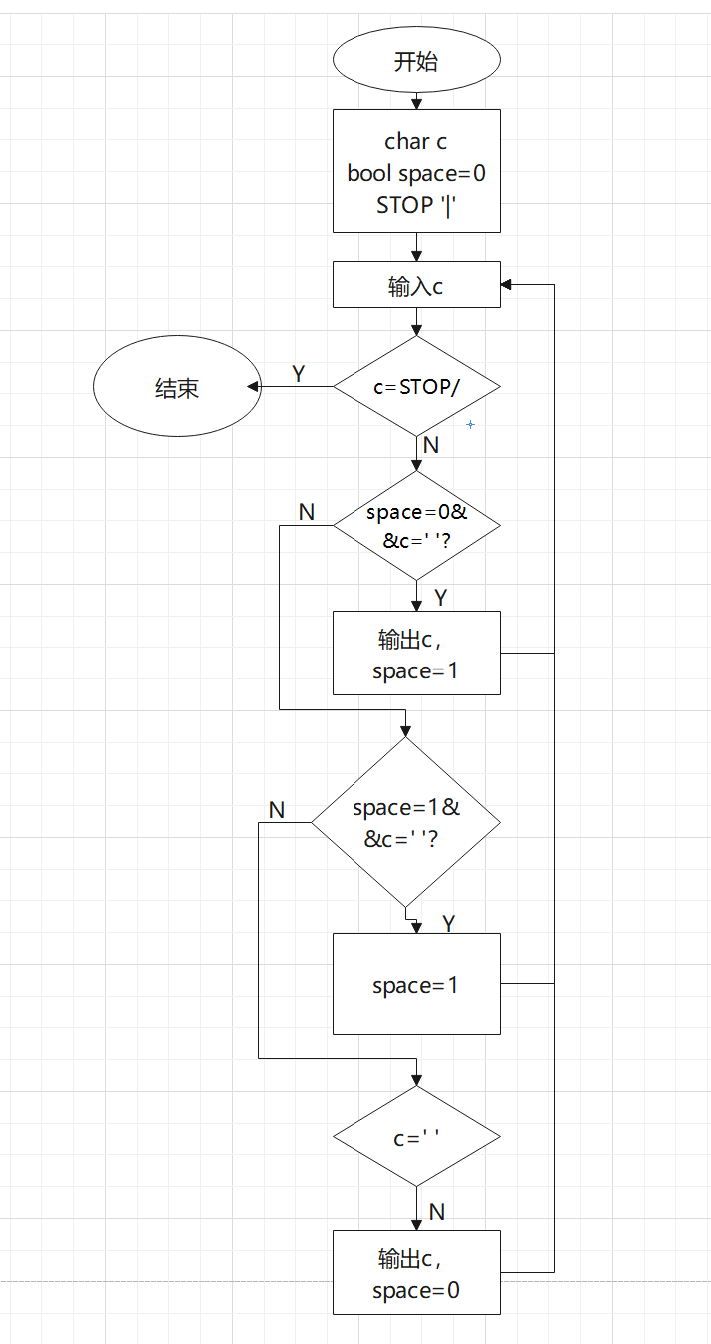
|  |  |  |  |
| --- | --- | --- | --- |
|  | 程序输入 | 理论结果 | 实际结果 |
| 用例1 | 2350 | 85.00 | 85.00 |
| 用例2 | 1500 | 25.00 | 25.00 |

 b)运行结果如下图

（2）将输入的正文复制到输出，复制过程中将每行一个以上的空格字符用一个空格代替。

**解答：**

* 1. 算法流程如图2.2所示



2)程序清单：

#include<stdio.h>

#include<stdbool.h>

#define STOP '|'

int main()

{

    char c;

    bool space=0;

    printf("enter(| to terminate)");

    while ((c=getchar())!=STOP)

    {

        if (space==0&&c==' ')

        {

            putchar(c);

            space=1;

        }

        else if (space=1&&c==' ')

        {

            space=1;

        }

        else if(c!=' ')

        {

            putchar(c);

            space=0;

        }

    }

    return 0;

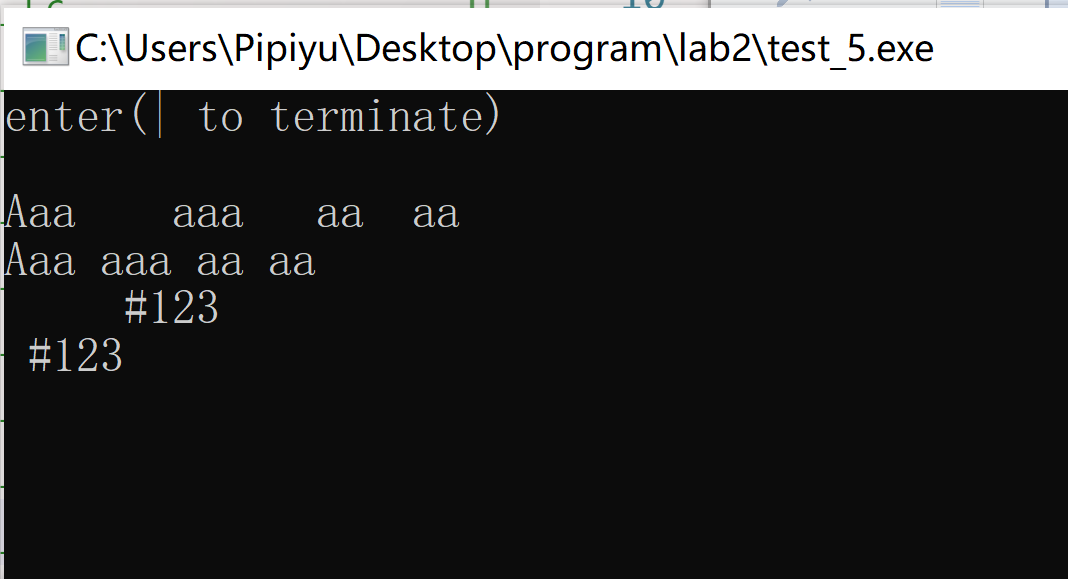
}

3）测试数据

A）数据输入

|  |  |  |  |
| --- | --- | --- | --- |
|  | 程序输入 | 理论结果 | 实际结果 |
| 用例1 | Aaa aaa aa aa | Aaa aaa aa aa | Aaa aaa aa aa |
| 用例2 | #123 | #123 | #123 |

b)运行结果如下图



（3）打印如下的杨辉三角形。

1 /\*第0行 \*/

1 1 /\*第1行 \*/

1 2 1 /\*第2行 \*/

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

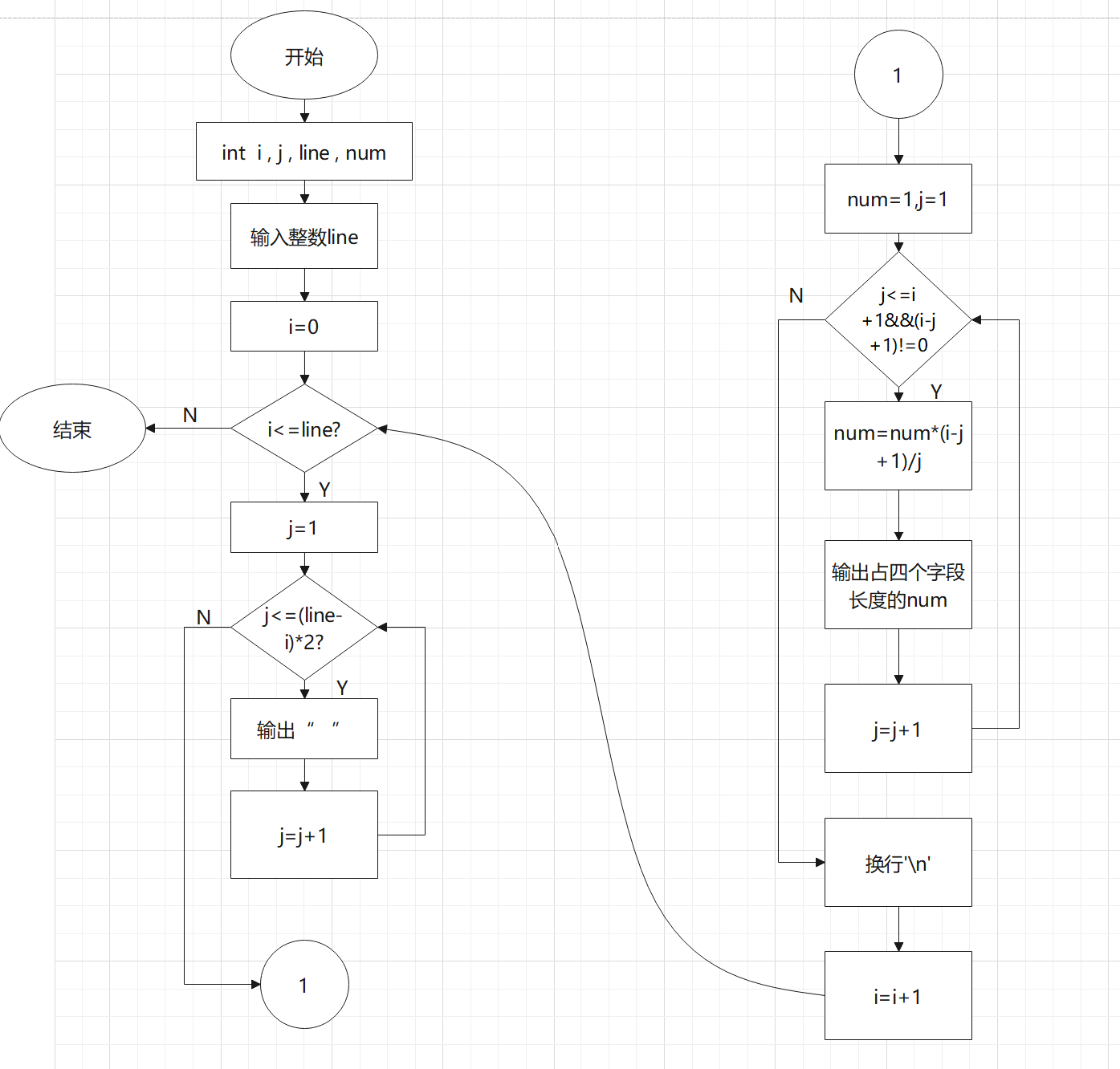
第i行第j列位置的数据值可以由组合表示，而的计算如下：

|  |  |
| --- | --- |
|  | (i=0,1,2,…) |
|  | (j=0,1,2,3,…,i) |

根据以上公式，采用顺推法编程，输出金字塔效果的杨辉三角形。特别要注意空格的数目，一位数之间是3个空格，两位数之间有2个空格，3位数之间只有一个空格。

**解答：**

1. 算法流程图如下



2)程序清单

#include<stdio.h>

int main()

{

    int i,j,line,num;

    scanf("%d",&line);

    getchar();

    for (i = 0; i <= line; i++)

    {

        for ( j = 1; j <=(line-i)\*2 ; j++)

            printf(" ");

        for ( num=1,j=1;j<=i+1; j++)

        {

            printf("1");

            for ( num=1,j=1;j<=i+1; j++)

            {

                if((i-j+1)!=0){

                num=num\*(i-j+1)/j;

                printf("%4d",num);

                }

            }

            printf("\n");

        }

    }

    getchar();

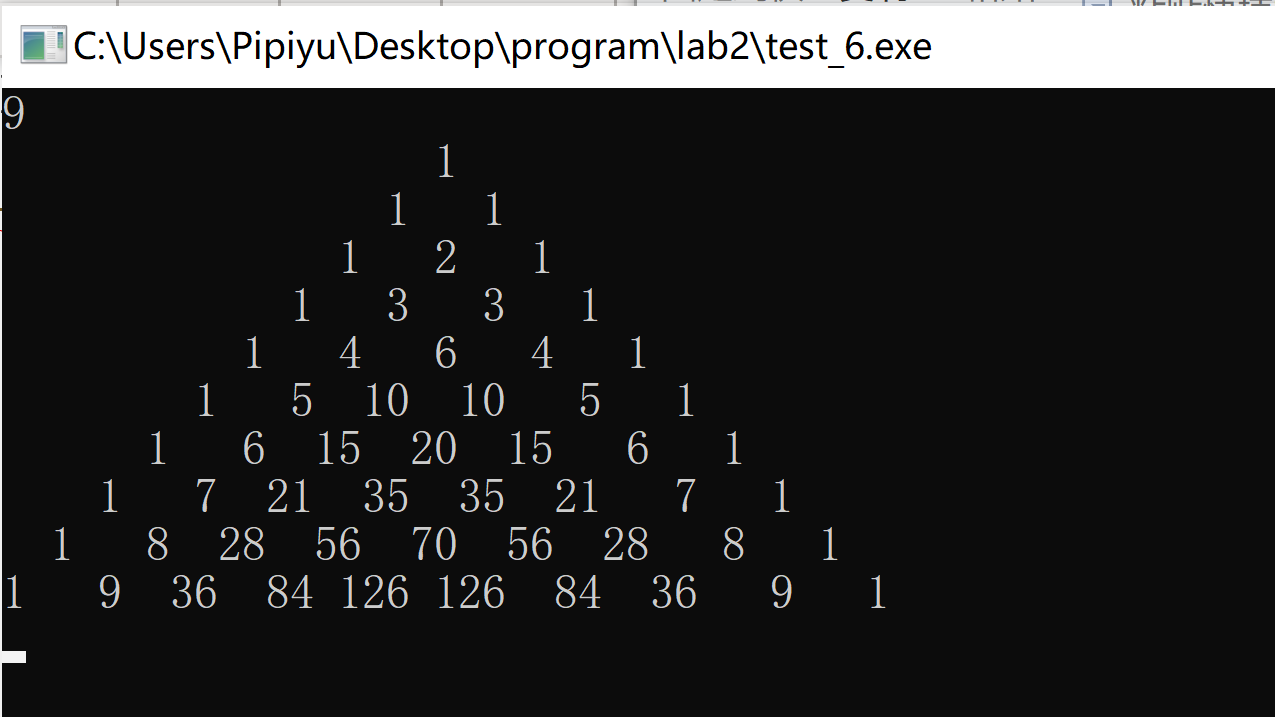
    return 0;

 }

3)测试数据

测试结果如下图

打印9行的杨辉三角



4）625这个数很特别，625的平方等于390625，其末3位也是625。请编程输出所有这样的3位数：它的平方的末3位是这个数本身。

**解答：**

1）算法流程图如下



2)程序清单：

#include<stdio.h>

 int main()

 {

     unsigned long num;

     printf("input enter");

     getchar();

     for ( num=100; num<=999; num++)

     {

         if (num==(num\*num)%1000)

             printf("%lu\n",num);

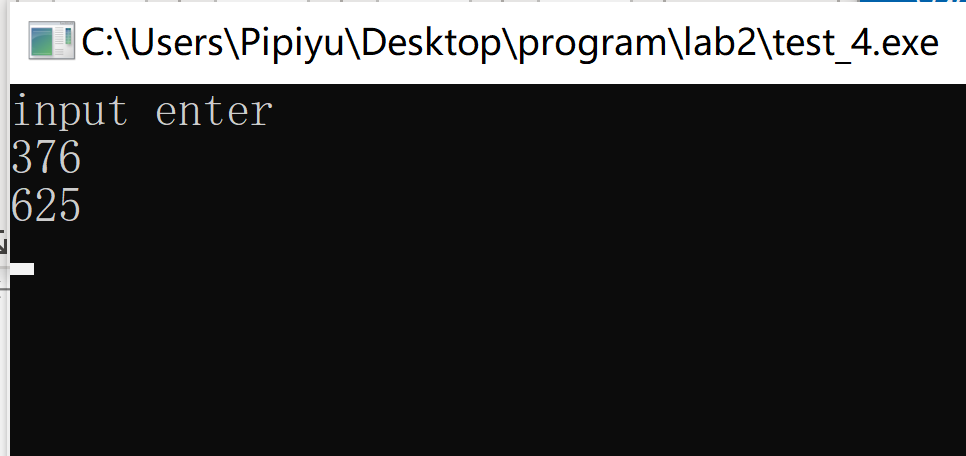
     }

     getchar();

     return 0;

 }

3）测试：



**1.3 实验小结**

写杨辉三角这题时，开始自己无从下手，看到别人都用二维数组，但是自己没有学，经过思考，发现可以用两个for循环，外层循环对行进行操作，内层循环对列进行操作，最后实现了杨辉三角。

经过这次实验，我已经完全理解了循环，并且能够较好的应用。

**3** **函数与程序结构实验**

**3.1 实验目的**

（1）熟悉和掌握函数的定义、声明；函数调用与参数传递方法；以及函数返回值类型的定义和返回值使用。

（2）熟悉和掌握不同存储类型变量的使用。

（3）熟悉多文件编译技术。

**3.2 实验内容**

**3.2.1 源程序改错**

下面是计算s=1!+2!+3!+…+n!的源程序，在这个源程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。

1 #include <stdio.h>

2 int main(void)

3 {

4 int k;

5 for(k=1;k<=20;k++)

6 printf("k=%d\tthe sum is %ld\n",k,sum\_fac(k));

7 return 0;

8 }

9 long sum\_fac(int n)

10 {

11 long s=0;

12 int i;

13 long fac;

14 for(i=1;i<=n;i++)

15 fac\*=i;

16 s+=fac;

17 return s;

18 }

**解答:**

1. 错误修改：

1)第1行与第2行之间缺少函数原型：

unsigned long long sum\_fac(int n);

2)第9行函数返回值类型错误：

unsigned long long sum\_fac(int n)

3)第11行和第13行s和fac类型错误：

unsigned long long s,fac;

4)第13行fac未初始化：

fac=1;

5)第16行表达式应在for循环内：

    for(i=1;i<=n;i++)

    {

        fac\*=i;

        s+=fac;

    }

错误修改后源代码如下：

#include <stdio.h>

unsigned long long sum\_fac(int n);

int main(void)

{

    int k;

    for(k=1;k<=20;k++)

         printf("k=%d\tthe sum is %llu\n",k,sum\_fac(k));

    getchar();

    return 0;

}

unsigned long long sum\_fac(int n)

{

    unsigned long long s,fac;

    int i;

    fac=1;

    s=0;

    for(i=1;i<=n;i++)

    {

        fac\*=i;

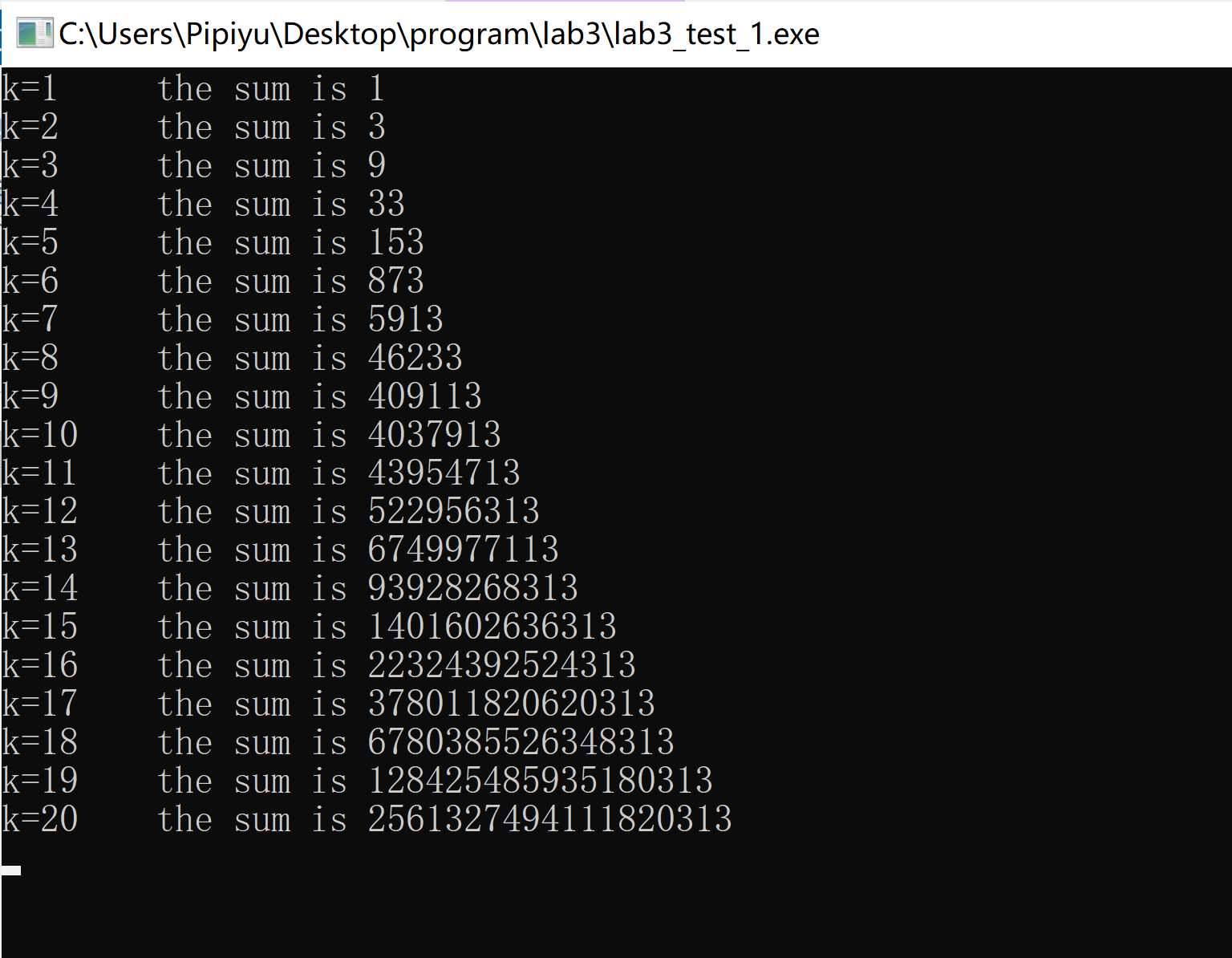
        s+=fac;

    }

    return s;

}

（2）错误修改后运行结果：



**2．程序修改替换题**

（1）根据将实验3-1改错题程序中sum\_fac函数修改为一个递归函数，用递归的方式计算。

**解答：（**1）：修改后程序源代码如下：

#include <stdio.h>

unsigned long long sum\_fac(int n);

int main(void)

{

    int k;

    for(k=1;k<=20;k++)

         printf("k=%d\tthe sum is %llu\n",k,sum\_fac(k));

    return 0;

}

unsigned long long sum\_fac(int n)

{

    unsigned long long s,fac;

    int i;

    fac=1;

    for(i=1;i<=n;i++)

    {

        fac\*=i;

    }

    if(n>1)

        s=sum\_fac(n-1)+fac;

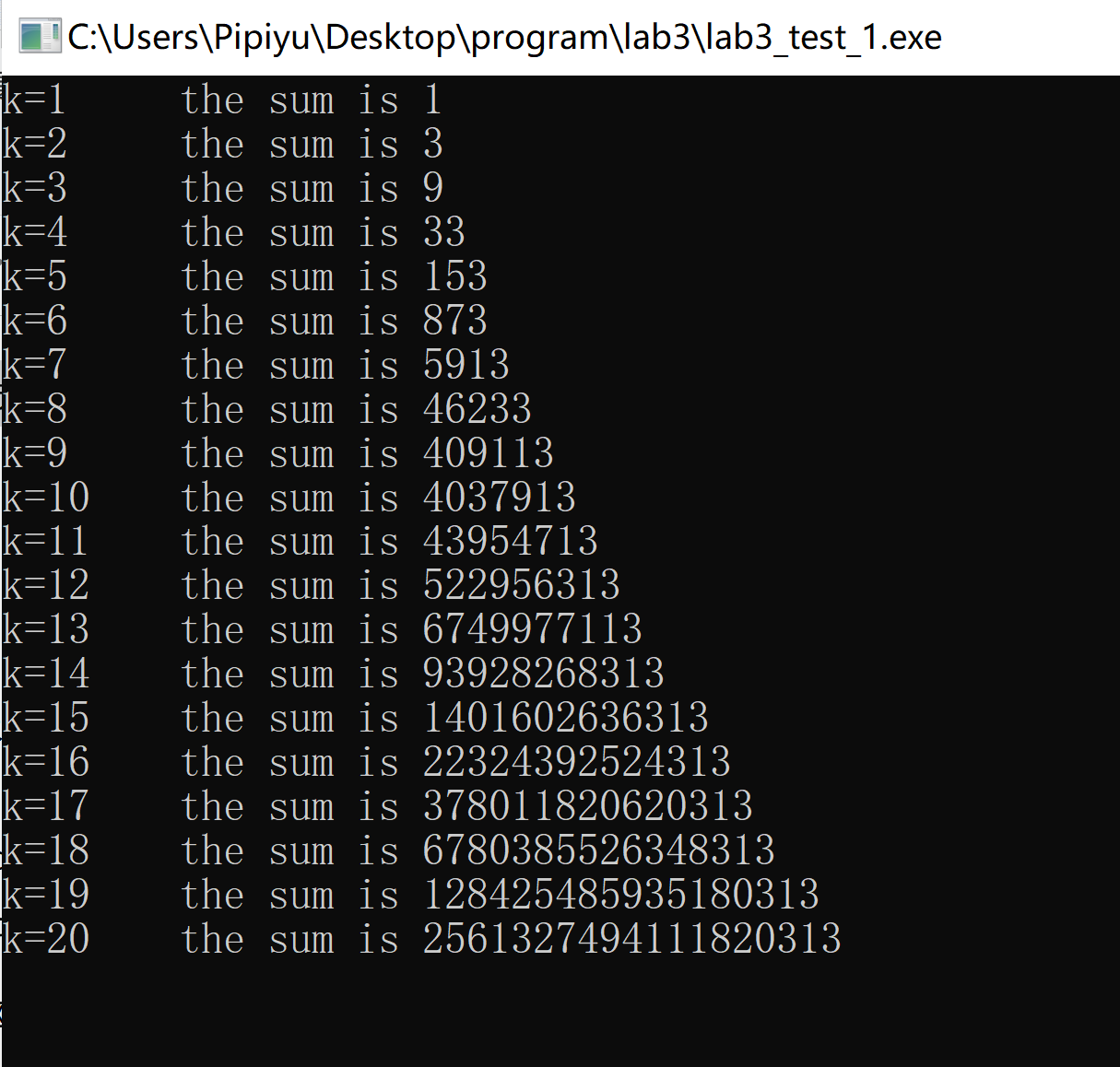
    else

        s=1;

    return s;

}

（2）：运行结果如下图：



（2）下面是计算的源程序，其中x是浮点数，n是整数。从键盘输入x和n，然后计算s的值。修改该程序中的sum和fac函数，使之计算量最小。

/\*实验3-2程序修改替换第(2)题程序：根据公式计算 s\*/

#include<stdio.h>

double mulx(double x,int n);

long fac(int n);

double sum(double x,int n)

{

int i;

double z=1.0;

for(i=1;i<=n;i++)

{

z=z+mulx(x,i)/fac(i);

}

return z;

}

double mulx(double x,int n)

{

int i;

double z=1.0;

for(i=0;i<n;i++)

{

z=z\*x;

}

return z;

}

long fac(int n)

{

int i;

long h=1;

for(i=2;i<=n;i++)

{

h=h\*i;

}

return h;

}

int main()

{

double x;

int n;

printf("Input x and n:");

scanf("%lf%d",&x,&n);

printf("The result is %lf:",sum(x,n));

return 0;

}

**解答：**(1):

在mulx函数和fac函数中分别啊z和h定义为静态变量，同时要去掉原来的for循环，修改后程序源代码如下：

#include<stdio.h>

double mulx(double x,int n);

long fac(int n);

double sum(double x,int n)

{

    int i;

    double z=1.0;

    for(i=1;i<=n;i++)

    {

        z=z+mulx(x,i)/fac(i);

    }

    return z;

}

double mulx(double x,int n)

{

    int i;

    static double z=1.0;

        z=z\*x;

    return z;

}

long fac(int n)

{

    static long h=1;

        h=h\*n;

    return h;

}

int main()

{

    double x;

    int n;

    printf("Input x and n:");

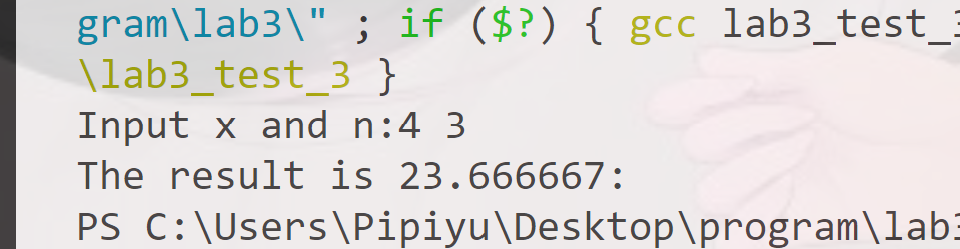
    scanf("%lf%d",&x,&n);

    printf("The result is %lf:",sum(x,n));

    return 0;

}

程序运行结果如图：





**3．跟踪调试题**

下面是计算fabonacci数列前n项和的源程序，现要求单步执行该程序，在watch窗口中观察Ik,sum,n值。具体操作如下：

（1）设输入5，观察刚执行完“scanf("%d",&k);”语句时，sum、k的值是多少？

（2）在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值是多少？返回后光条停留在哪个语句上？

（3）在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了哪里？

（4）在fabonacci函数内部单步执行，观察函数的递归执行过程。体会递归方式实现的计算过程是如何完成数计算的，并特别注意什么时刻结束递归，然后直接从第一个return语句返回到了哪里？

（5）在fabonacci函数递归执行过程中观察参数n的变化情况，并回答为什么k、sum在fabonacci函数内部不可见？

**/\*实验3-3跟踪调试题程序：**计算fabonacci数列前n项和**\*/**

#include<stdio.h>

int main(void)

{

int i,k;

long sum=0,fabonacci(int n);

printf("Inut n:");

scanf("%d",&k);

for(i=1;i<=k;i++){

sum+=fabonacci(i);

printf("i=%d\tthe sum is %ld\n",i,sum);

}

return 0;

}

long fabonacci(int n)

{

if(n==1 || n==2)

return 1;

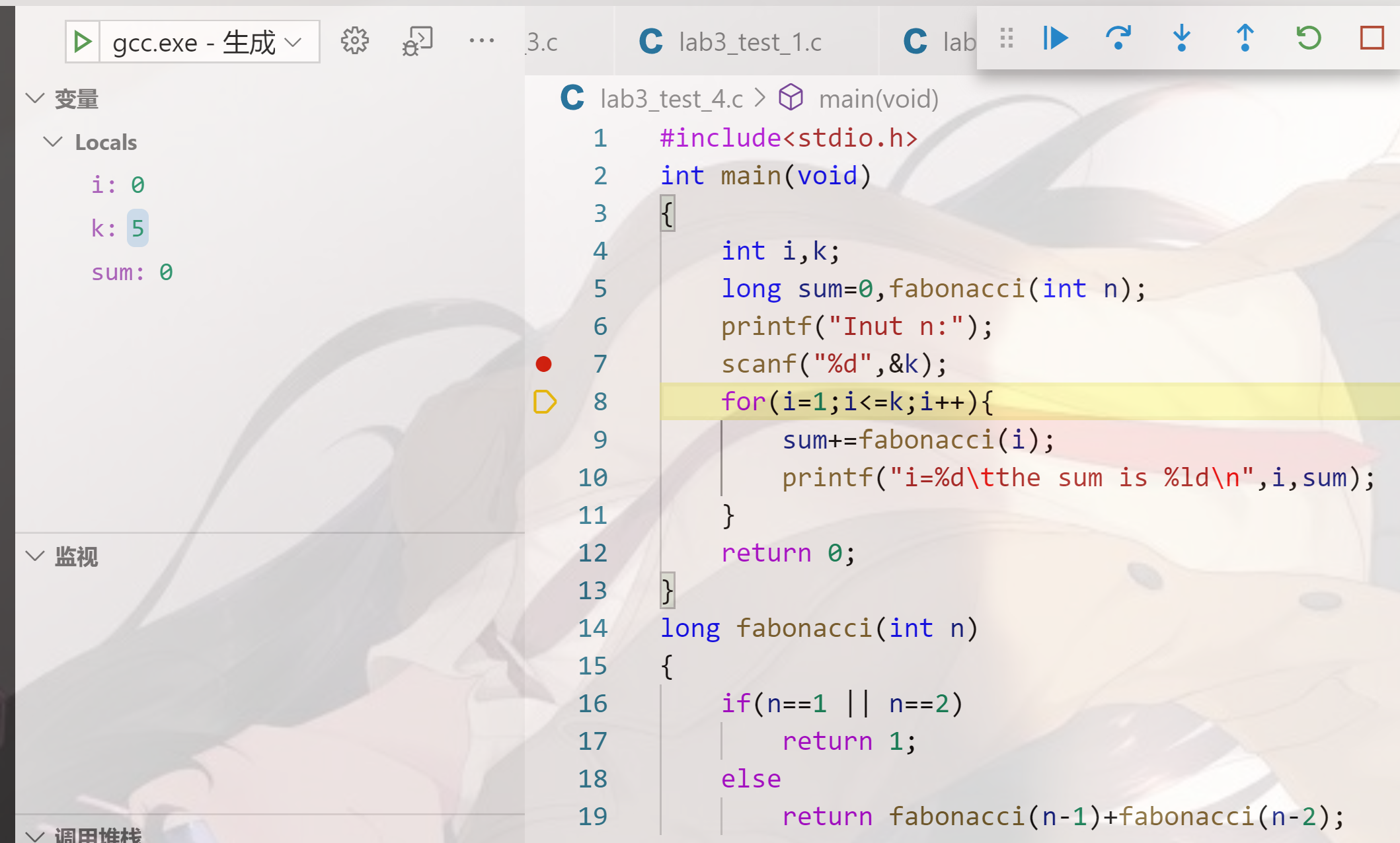
else

return fabonacci(n-1)+fabonacci(n-2);

}

1. 设输入5，观察刚执行完“scanf("%d",&k);”语句时，sum、k的值是多少？

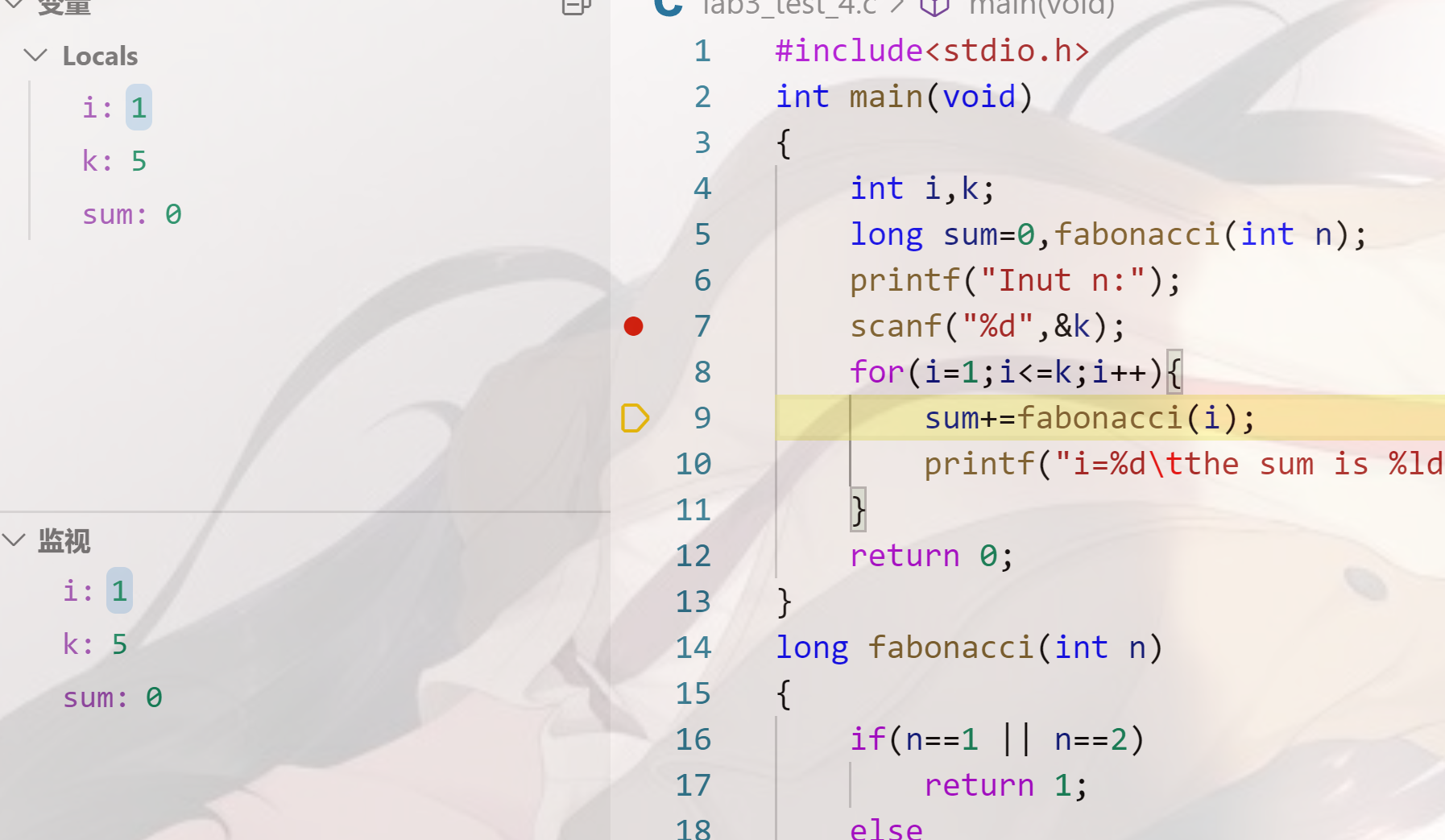
**解答**：

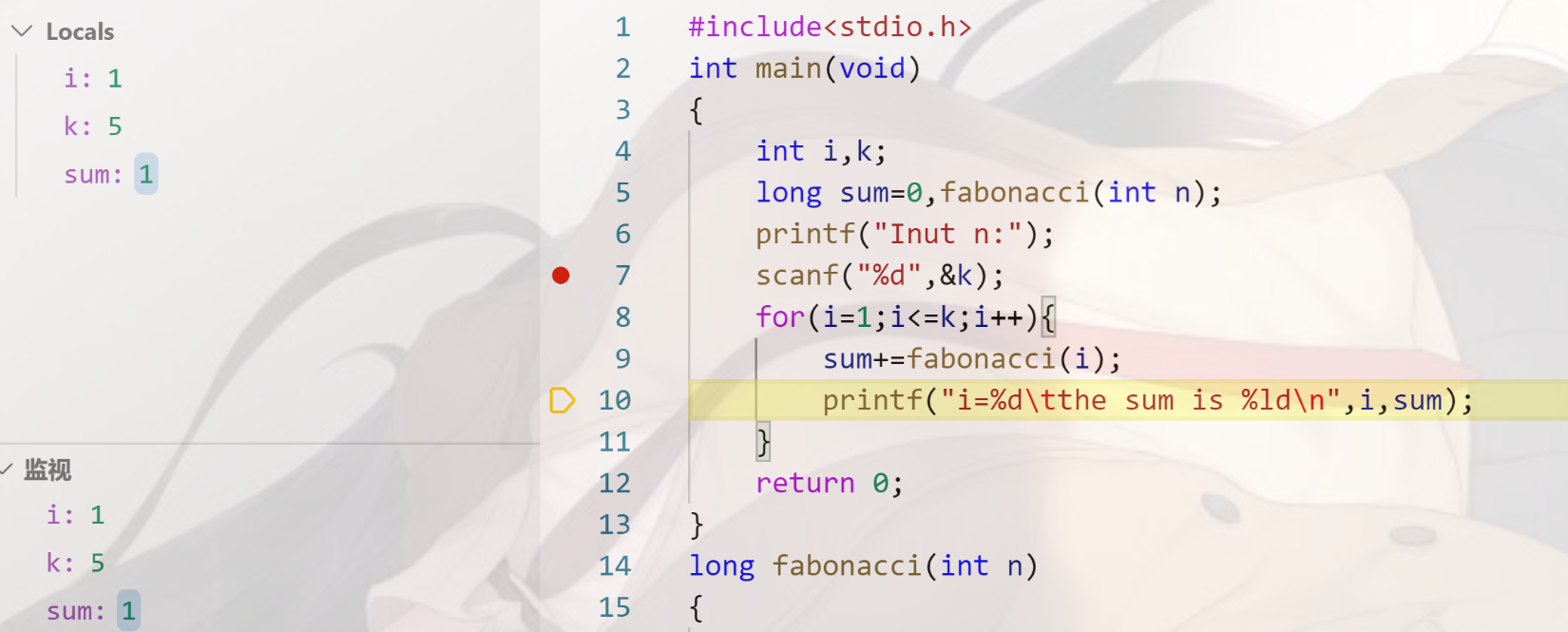
sum=0;k=5****

(2) 在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值是多少？返回后光条停留在哪个语句上？

**解答**：i=1,k=5,sum=0

返回后光条停在printf函数上，此时sun变为1

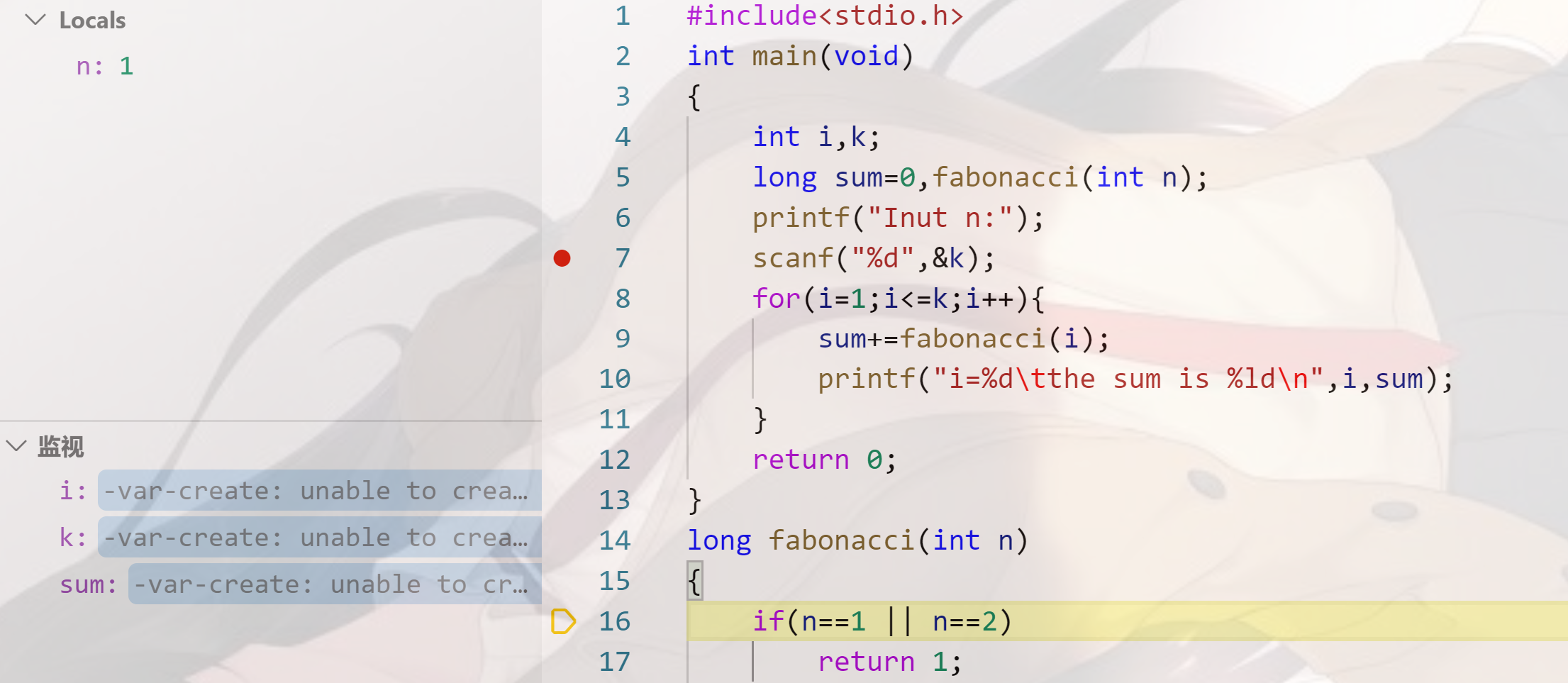




（3）在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了哪里？

**解答：**

调到fabonacci函数的if语句中



（4）在fabonacci函数内部单步执行，观察函数的递归执行过程。体会递归方式实现的计算过程是如何完成数计算的，并特别注意什么时刻结束递归，然后直接从第一个return语句返回到了哪里？

**解答:**

这个函数中存在两个递归，都从第一个return语句结束递归，当第一个递归结束时，第一个return语句直接返回到底部花括号语句，在进行第二个递归，第二个递归结束时，返回到printf函数。

（5）在fabonacci函数递归执行过程中观察参数n的变化情况，并回答为什么k、sum在fabonacci函数内部不可见？

**解答：**

Fabonacci函数多次递归，n的值发生了多次递减的变化直到n变为1或2；而且每次递归n的初始值不同；

因为k，sum没有作为参数传递进fabonacci函数，所以在fabonacci函数中k和sum的值是不可见的。

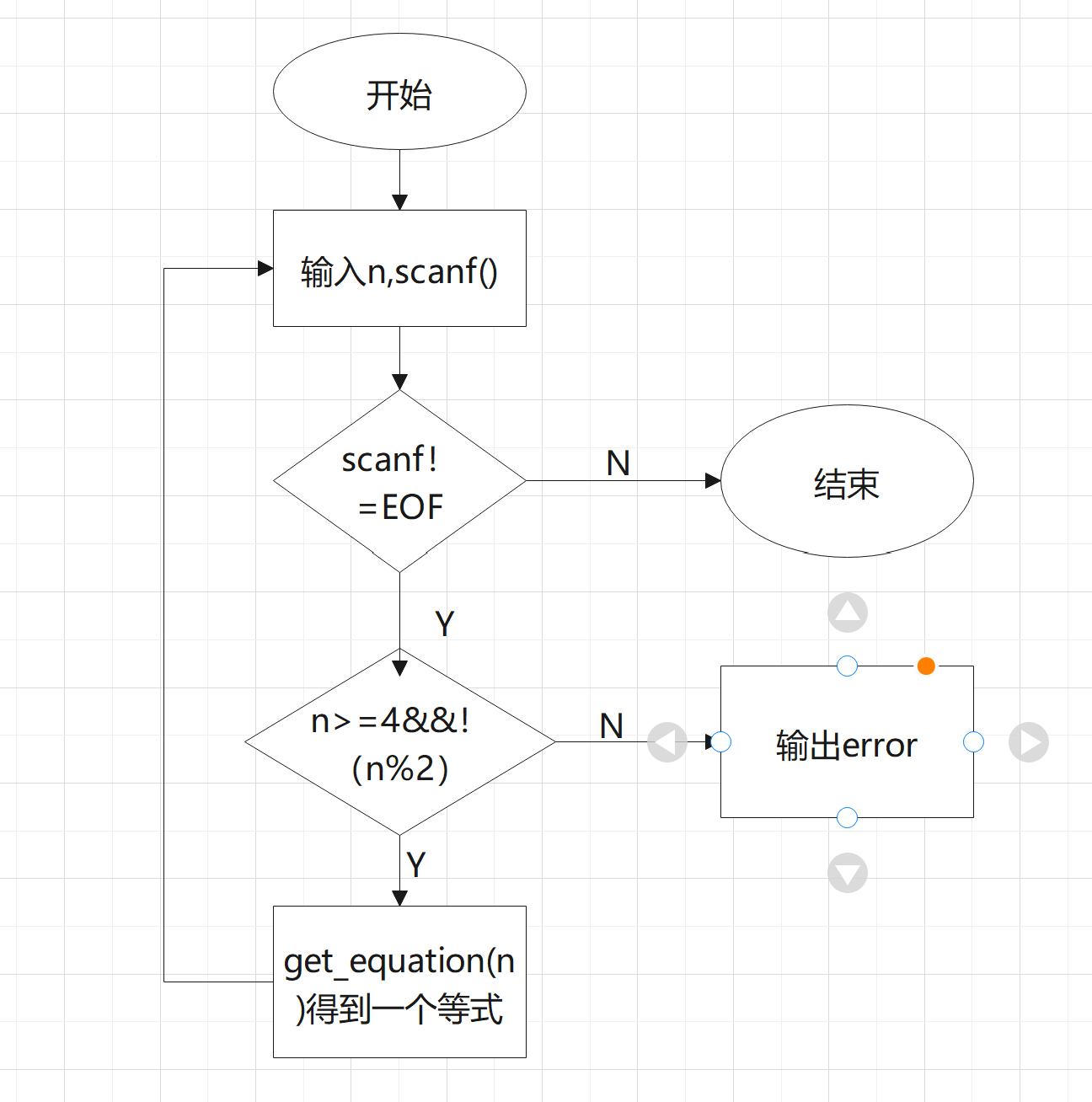
**4．程序设计**

（1）编程验证歌德巴赫猜想：一个大于等于4的偶数都是两个素数之和。要求设计一个函数对其形参n验证哥德巴赫猜想，并以“n=n1+n2”的形式输出结果。例如：n=6，输出“6=3+3”。main函数循环接收从键盘输入的整数n，如果n是大于或等于4的偶数，调用上述函数进行验证。

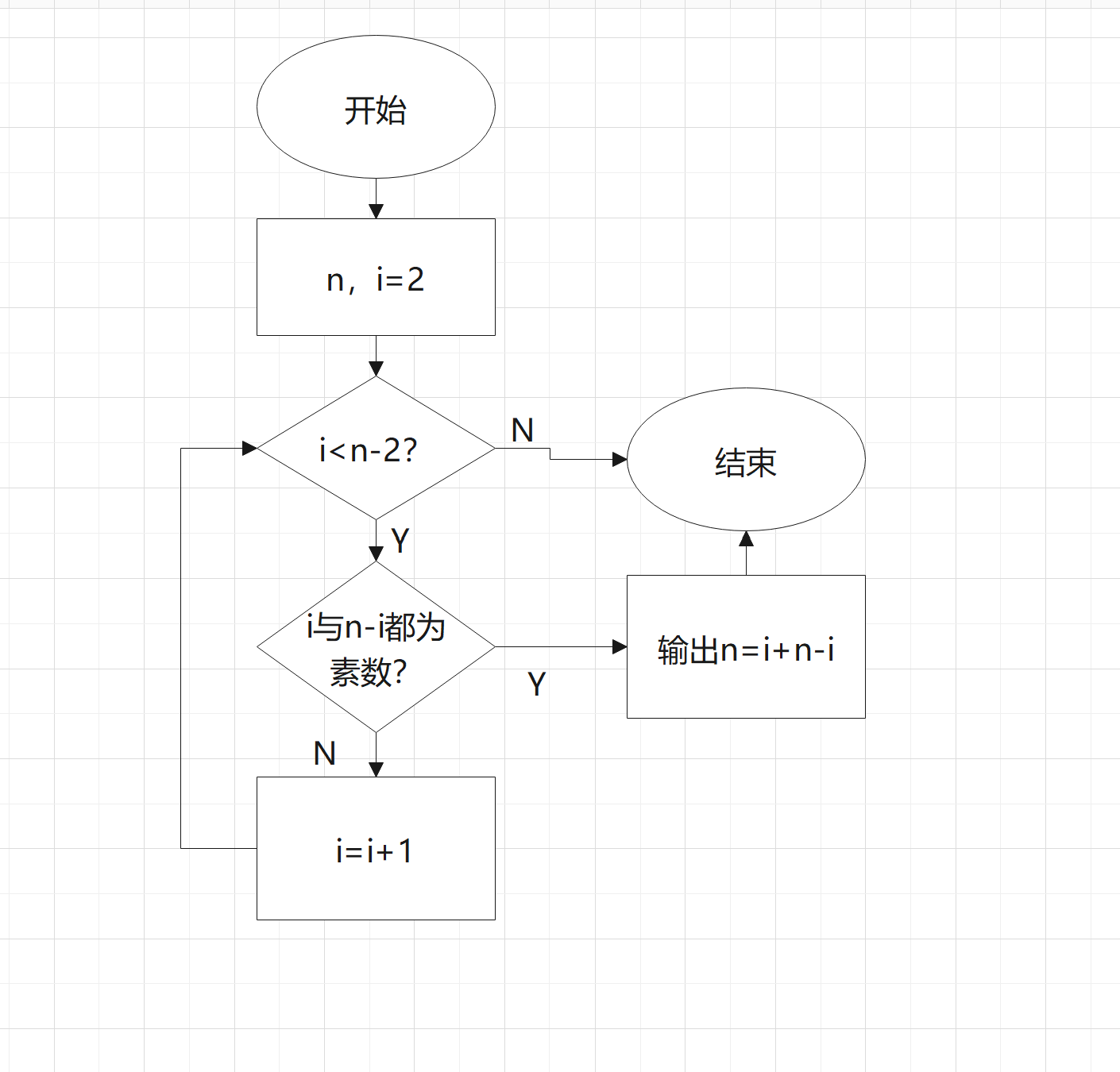
**解答：**

（1）算法流程如图

1. 主调函数流程图



1. get\_equation(n)流程图



（2）程序清单：

#include<stdio.h>

#include<stdbool.h>

void get\_equation(int n);

bool judge\_prinum(int n);

int main()

{

    int n;

    printf("Input a num.\n");

    while (scanf("%d",&n)!=EOF)

    {

        if(n>=4&&!(n%2))

            get\_equation(n);

        else

            printf("Please input a num higher than 4 and even.\n");

    }

    return 0;

}

bool judge\_prinum(int n)

{

    int i;

    bool flag;

    flag=1;

    for ( i = 2;i<=(n>>1); i++)

    {

        if(!(n%i))

        {

            flag=0;

            break;

        }

    }

    return flag;

}

void get\_equation(int n)

{

    int i;

    for (i = 2; i<=n-2; i++)

    {

        if(judge\_prinum(i)&&judge\_prinum(n-i))

        {

            printf("%d=%d+%d\n",n,i,n-i);

            break;

        }

    }

}

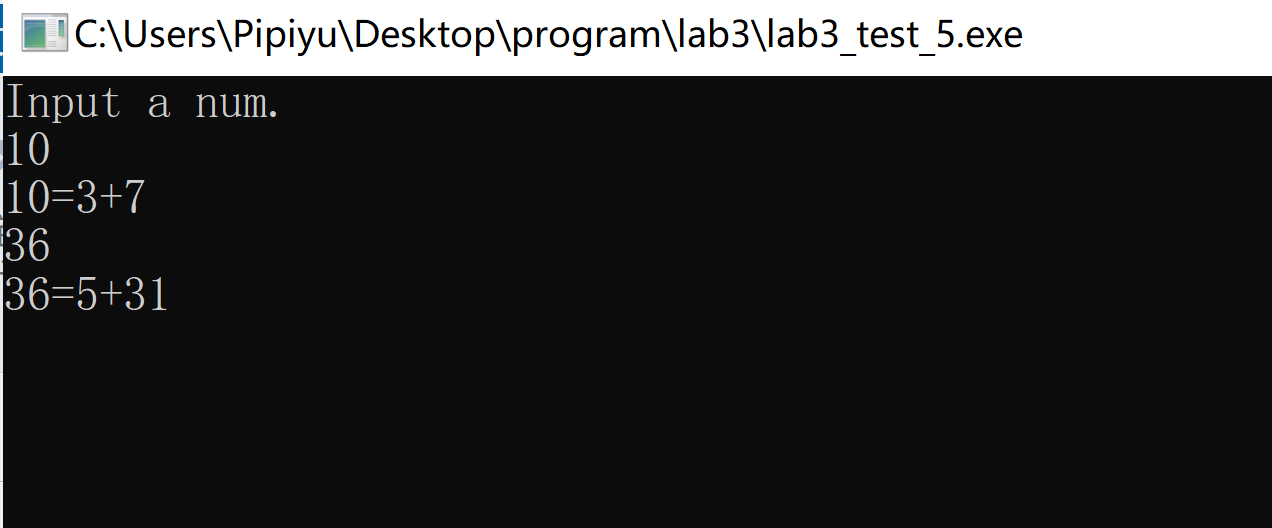
（3）测试

（a） 测试数据

表3-2 编程设计题2测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 输入数据 | 理论结果 |
| 用例1 | 10 | 10=3+7 |
| 用例2 | 36 | 36=5+31 |

（b）对应测试数据的运行结果如图所示。

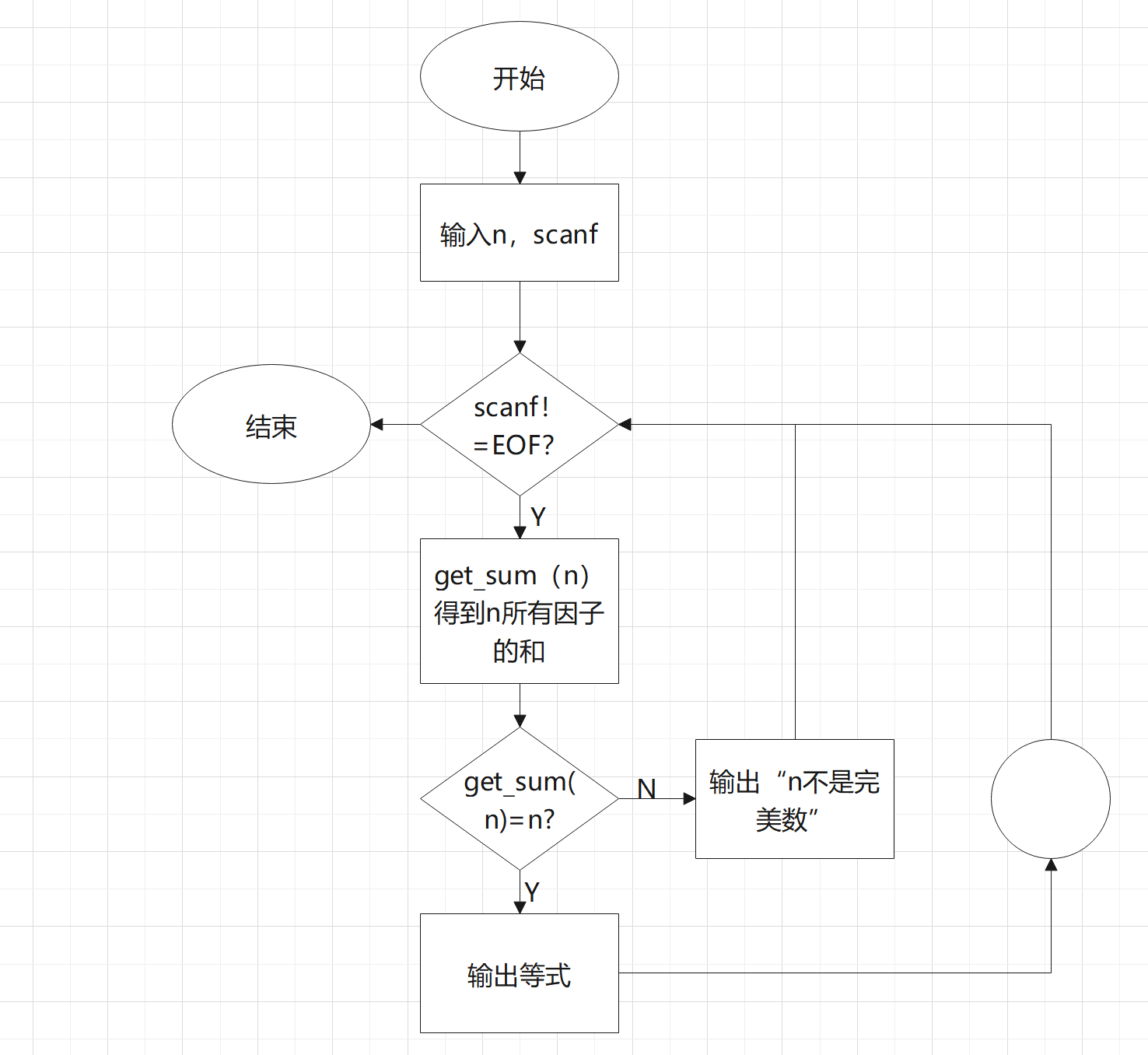


（2）完全数（Perfect number），又称完美数或完备数，特点是它的所有真因子（即除了自身以外的约数，包括1）之和恰好等一它本身。例如6=1+2+3，28=1+2+4+7+14等。编程寻找108以内的所有完全数。要求设计一个函数，判定形参n是否为完全数，如果是，则以n的真因子之和的形式输出结果，例如“6=1+2+3”；否则，输出“not a perfect number”，例如“5 is not a perfect number”。

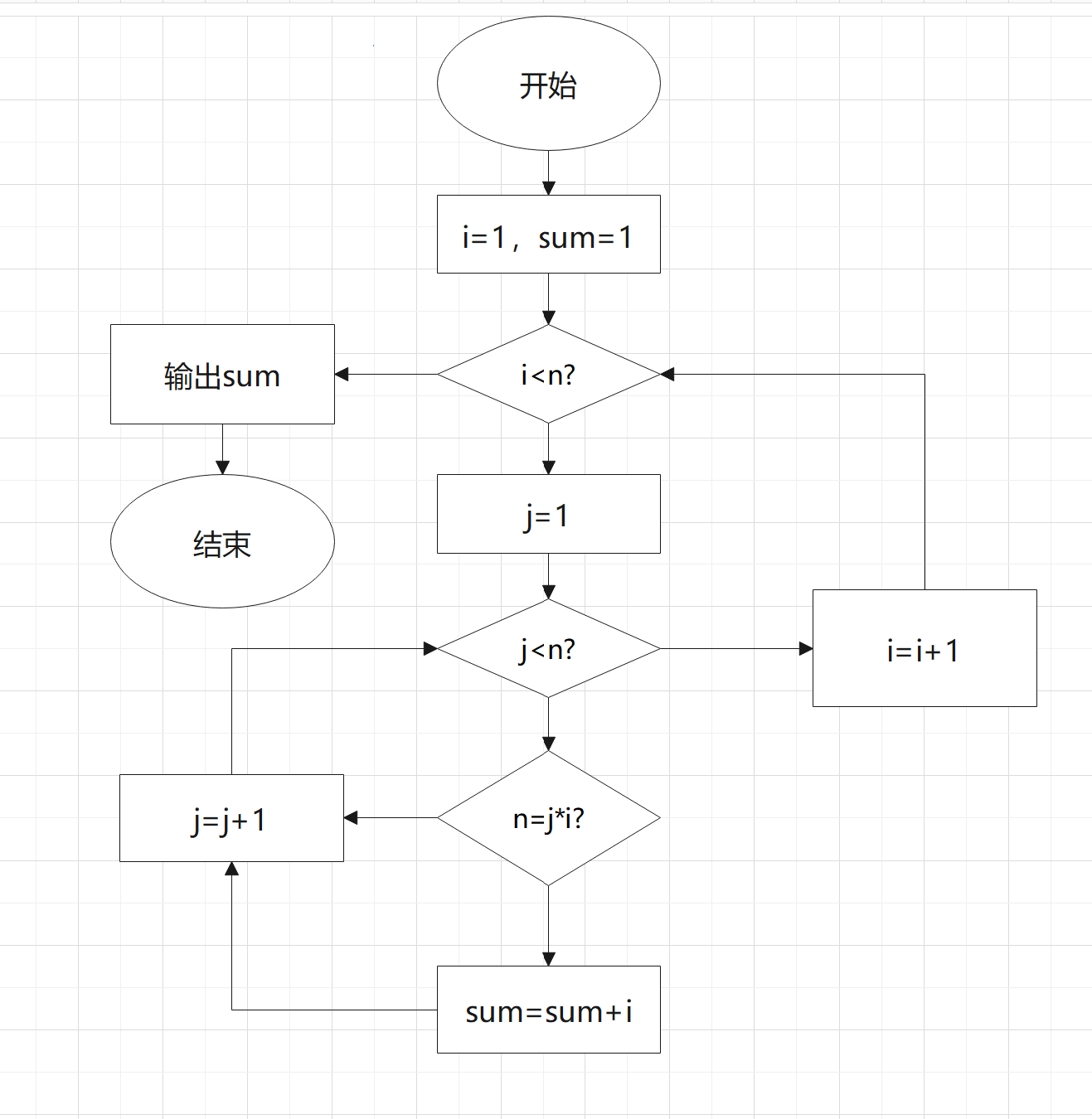
在main函数中调用该函数求108以内的所有完全数。

1）流程图

A)主调函数流程图



B）get\_sum(n)函数流程图



2）程序清单：

#include<stdio.h>

long get\_sum(long n);

void get\_equ(long n);

int main()

{

    long n;

    printf("Input a number:");

    while (scanf("%ld",&n)!=EOF)

    {

        if(get\_sum(n)==n)

        {

            printf("%ld is a perfect number.\n",n);

            printf("%ld=1",n);

            get\_equ(n);

            printf("\n");

        }

        else

            printf("%ld isn't a perfect number.\n",n);

    }

    return 0;

}

long get\_sum(long n)

{

    long i,j,sum;

    sum=0;

    for ( i = 1; i < n; i++)

    {

        for ( j = 1; j < n; j++)

        {

            if(n==j\*i)

                sum+=i;

        }

    }

    return sum+1;

}

void get\_equ(long n)

{

    long i,j,sum;

    sum=0;

    for ( i = 1; i < n; i++)

    {

        for ( j = 1; j < n; j++)

        {

            if(n==j\*i)

            {

                sum+=i;

                printf("+%ld",i);

            }

        }

    }

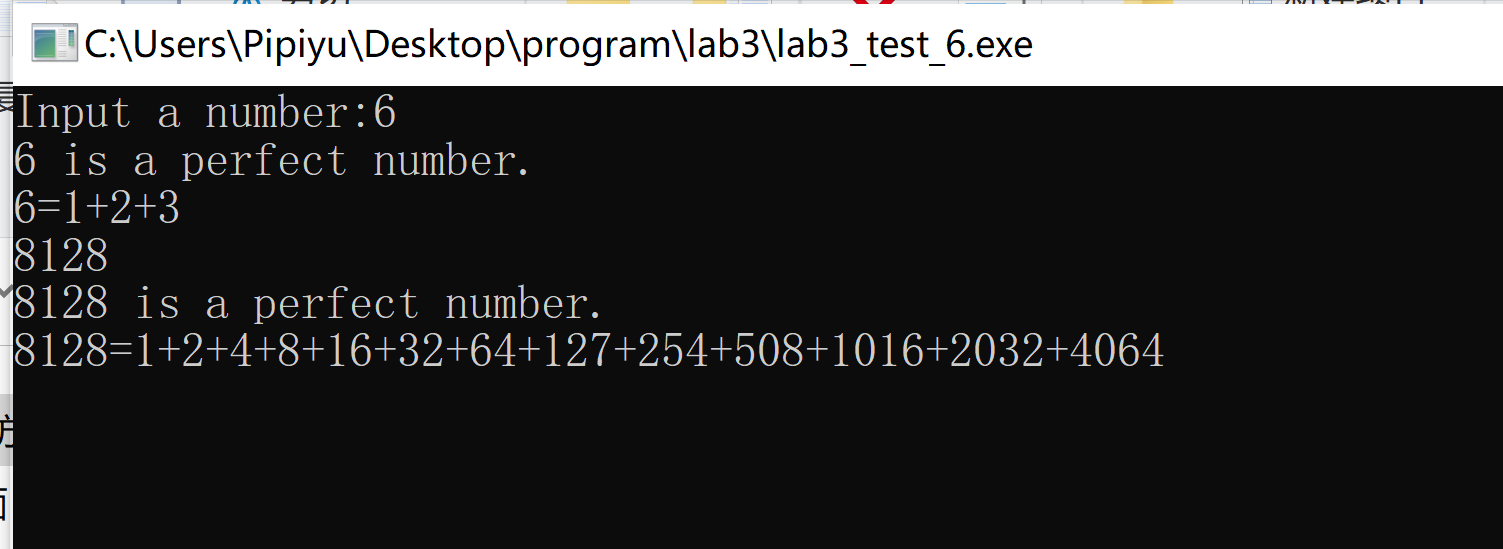
}

3）测试

A．测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 输入数据 | 理论结果 |
| 用例1 | 6 | 6=1+2+3 |
| 用例2 | 8128 | 8128=1+2+4+8+16+32+64+128+256+512+1024+2032+4064 |

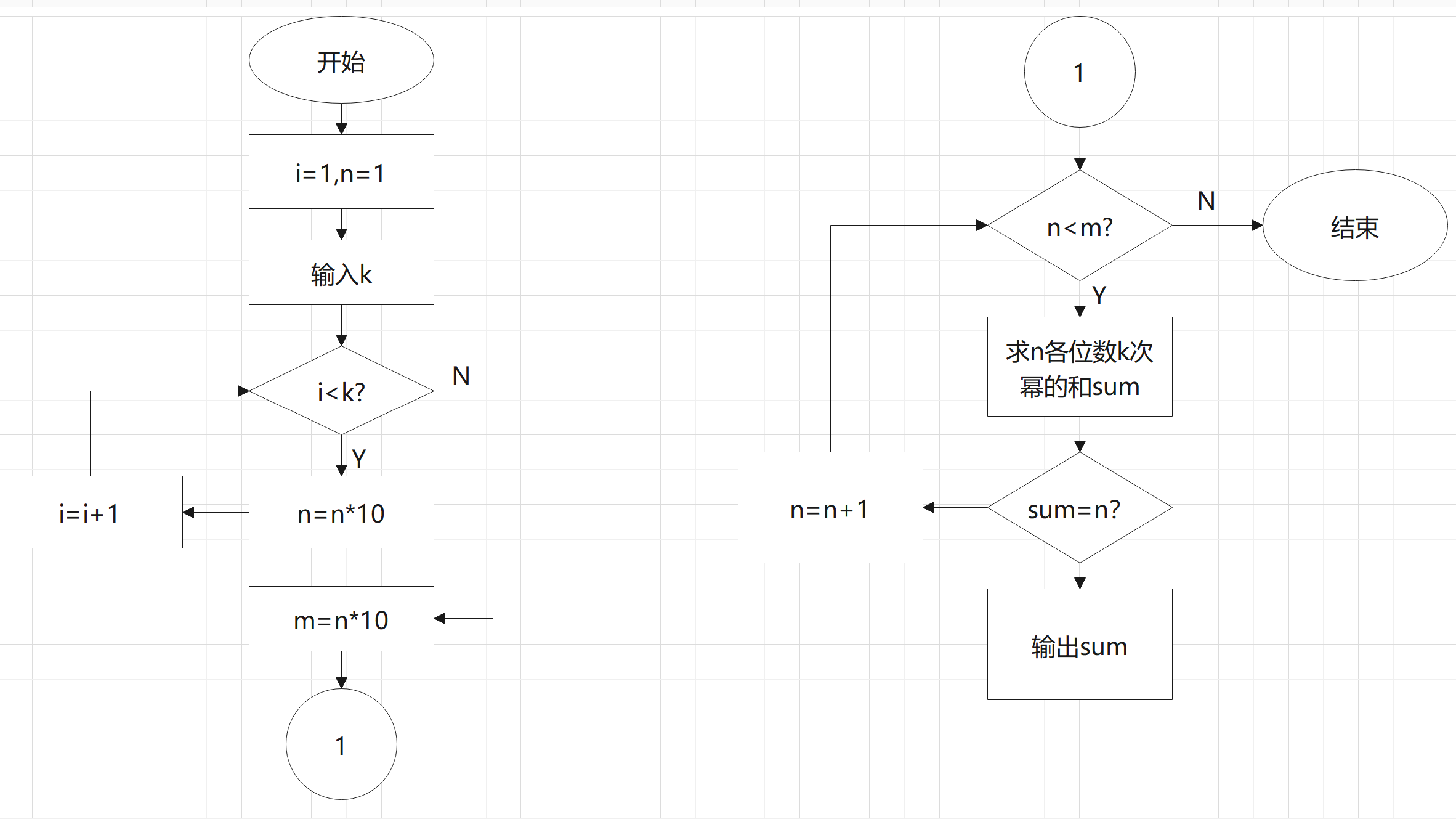
B.测试结果如图



（3）自幂数是指一个n位数，它的每个位上的数字的n次幂之和等于它本身。水仙花数是3位的自幂数，除此之外，还有4位的四叶玫瑰数、5位的五角星数、6位的六合数、7位的北斗星数、8位的八仙数等。编写一个函数，判断其参数n是否为自幂数，如果是，则返回1；否则，返回0。main函数能反复接收从键盘输入的整数k，k代表位数，然后调用上述函数求k位的自幂数，输出所有k位自幂数，并输出相应的信息，例如“3位的水仙花数共有4个153，370，371，407”。当k=0时程序结束执行。

**解答：**

（1）流程图



(2)程序清单：

#include<stdio.h>

long get\_sum(int n,int k);

long get\_power(int m, int k);

int main()

{

    int n,i,m,k;

    printf("Input k :\n");

    while (scanf("%d",&k)!=EOF)

    {

        n=1;

        for ( i = 1; i < k; i++)

            n=n\*10;

        for ( m=n\*10;  n< m; n++)

        {

            if(get\_sum(n,k)==n)

                printf("%d is a self-power.\n",n);

        }

    }

    return 0;

}

long get\_sum(int n,int k)

{

    int m,i;

    long sum,power;

    sum=0;

    for ( i = 1; i <= k; i++)

    {

        m=n%10;

        n=n/10;

        power=get\_power(m,k);

        sum+=power;

    }

    return sum;

}

long get\_power(int m,int k)

{

    int i;

    long power;

    power=1;

    for ( i = 1; i <= k; i++)

        power\*=m;

    return power;

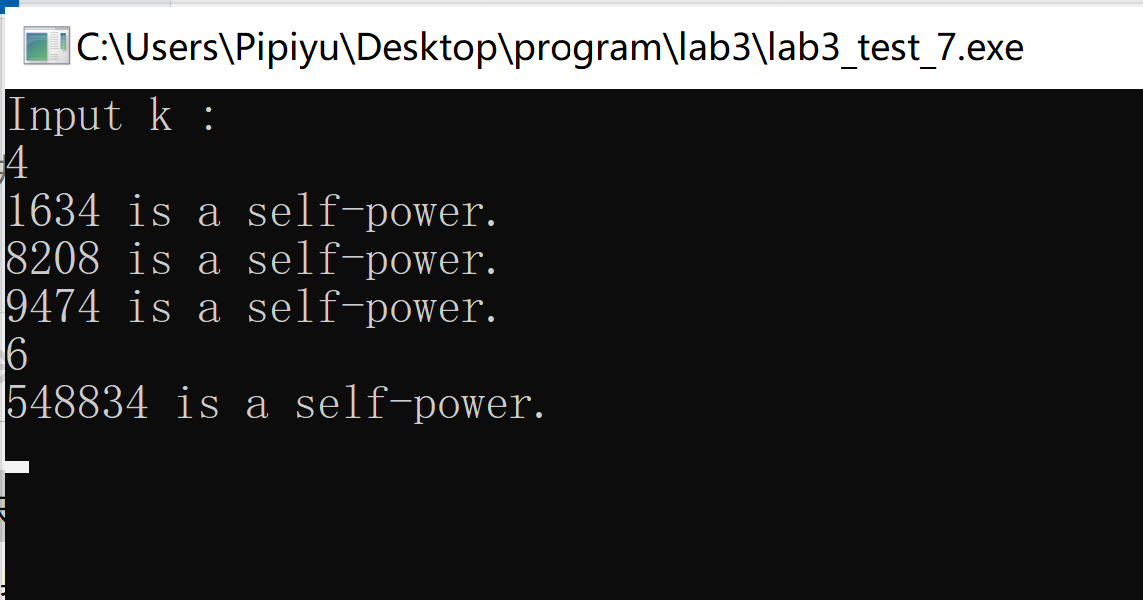
}

（3）测试

A)测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 输入数据 | 理论结果 |
| 用例1 | 4 | 1634,8208,9474 |
| 用例2 | 6 | 548834 |

B)运行结果：



**3 实验小结**

通过本次实验，初步了解了简单的递归并能较好的运用，有了对函数的基本认识。

通过对函数的学习，也理解了模块化处理问题的方式。