# 3函数与程序结构实验

## 3.1 实验目的

熟悉和掌握函数的定义、声明以及函数调用与参数传递方法，能够正确使用函数返回值类型的定义和返回值。将不同存储类型变量的灵活运用。熟悉多文件编译技术。

## 3.2 必做题

### 3.2.1 源程序改错

**【题目】**

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 void main(void)

3 {

4 int k;

5 for(k=1;k<6;k++)

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

7 }

8 long sum\_fac(int n)

9 {

10 long s=0;

11 int i;

12 long fac;

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

14 fac\*=i;

15 s+=fac;

16 return s;

17 }

【**错误原因分析及改错方案**】

经过调试和运行，发现这个程序一共存在三处错误,分析如下。

1. 源程序中使用了函数sum\_fac，但是主函数前面没有进行声明；所以在使用前先声明函数sum\_fac，即在main函数之前；
2. 第12行错误，源程序对变量fac使用\*=运算前未对变量fac赋初始值，使得源程序不能正常运行；只需在声明时对fac赋初始值1，即改为long fac = 1;
3. 第14、15行错误，函数中的fac \*= i; s += fac;未括起，导致后一语句在循环外；则用花括号将这两个语句括起表示为代码块。

**【修改后代码】**

#include<stdio.h>

long sum\_fac(int n);

int main(void)

{

int k;

for (k = 1; k<6; k++)

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

return 0;

}

long sum\_fac(int n)

{

long s = 0;

int i;

long fac = 1;

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

{

fac \*= i;

s += fac;

}

return s;

}

**【程序运行结果】**

该程序没有输入，直接输出。

输出结果预测如下：

k=1 the sum is 1

k=2 the sum is 3

k=3 the sum is 9

k=4 the sum is 33

k=5 the sum is 153

实际输出结果如图，发现实际输出结果与预测完全相符。

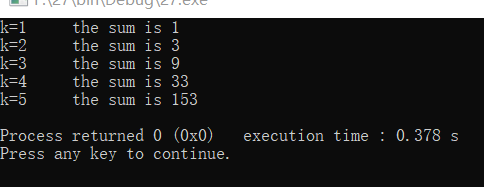


图3-1 实验三必做题3.2.1 运行结果

### 3.2.2 源程序修改替换

**【题目】**

（1）修改上述源程序中的sum\_fac函数，使其计算量最小。

（2）修改上述源程序中的sum\_fac函数，计算



**【解题思路】**

（1）由于按照题目的思路，计算乘法的次数为次计算加法的次数为n-1次，例如当n=5时，原题需计算19次，改写思路为：按照1+2\*（1+3\*（1+4\*（1+5）））来计算，总计算次数仅为7次，大大减少了计算量。同时，巧妙利用for循环将复杂的循环总结成一个语句，减少了代码复杂度。程序框图如图。

（2）由于不要求计算效率，沿用原程序的计算方法，只讲数据类型与计算方改变即可。

**【算法流程图】**

**（1）**

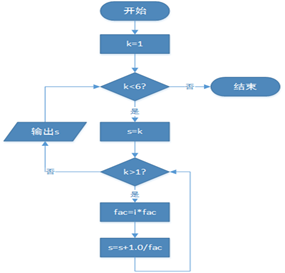
****

图3-2 实验三必做题3.2.2（1）流程图

**【改写代码】**

1. 修改上述源程序中的sum\_fac函数，使其计算量最小。

#include<stdio.h>

long sum\_fac(int n);

int main(void)

{

int k;

for (k = 1; k < 6; k++)

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

return 0;

}

long sum\_fac(int n)

{

long s;

for (s = n; n-- > 1; s = (s + 1) \* n);

return s;

}

（2） 修改上述源程序中的sum\_fac函数，计算



#include<stdio.h>

double sum\_fac(int n);

int main(void)

{

double k;

for (k = 1; k<6; k++)

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

return 0;

}

double sum\_fac(int n)

{

double s = 0;

int i;

double fac = 1;

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

{

fac \*= i;

s += 1 / fac;

}

return s;

}

**【程序预测输出结果】**

（1）优化后计算结果应该与原程序相同，理论输出如下。

k=1 the sum is 1

k=2 the sum is 3

k=3 the sum is 9

k=4 the sum is 33

k=5 the sum is 153

（2）计算结果预测输出如下

k=1.000000 the sum is 1.000000

k=2.000000 the sum is 1.500000

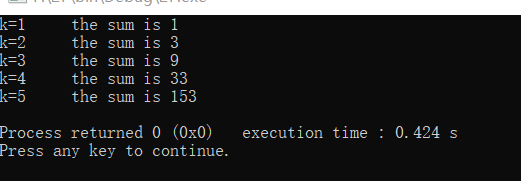
k=3.000000 the sum is 1.666667

k=4.000000 the sum is 1.708333

k=5.000000 the sum is 1.71666**7**

【**程序运行结果**】

图3-3 实验三必做题3.2.2（1）运行结果



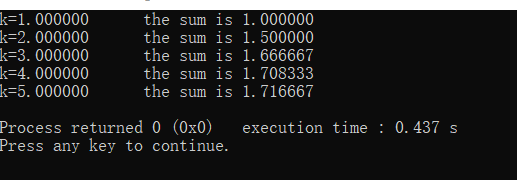


图3-4 实验三必做题3.2.2（2）运行结果

### 3.2.3 跟踪调试

**【题目】**

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，并观察p,i,sum,n值,即：

（1）刚执行完scanf("%d",&k);语句时，p,i值是多少？

（2）从fabonacci函数返回后,光条停留在哪个语句上？

（3）进入fabonacci函数时，watch窗口显示的是什么？

（4）当i=3时，从调用fabonacci函数到返回，n值如何变化？

#include<stdio.h>

long fabonacci(int);

int main(void)

{

int i,k;

long sum=0,\*p=&sum;

scanf("%d",&k);

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

sum+=fabonacci(i);

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

}

}

long fabonacci(int n)

{

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

return 1;

else

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

}

【**调试过程**】

（1）使用Visual Studio 2015加入断点进行调试，输入k=5，待scanf函数执行完之后。通过数据窗格（如图3-5）知：指针p的值是0x00cffb80，指向sum，i的值是-858993460

（2）按继续，刚执行完fabonacci函数返回后，光条停留在调用行。如图3-6所示。

（3）进入函数后，数据窗口中只显示显示局部变量n，因为只有n在这个域内可见。当然，可以通过设置“监视”来检测全局变量的变化，如图3-7

（4）一直继续执行，直到i=3时，刚刚调用时，n的值为3；第一次return fabonacci(n - 1) + fabonacci(n - 2);时，光标返回函数头，n值变为2，入栈；第三次return 1;时，光标调到函数尾，n的值变为3，出栈；之后return fabonacci(n - 1) + fabonacci(n - 2);，n值再次变为1，入栈；在此之后，再次执行return fabonacci(n - 1) + fabonacci(n - 2);，n变为3，出栈，返回2。

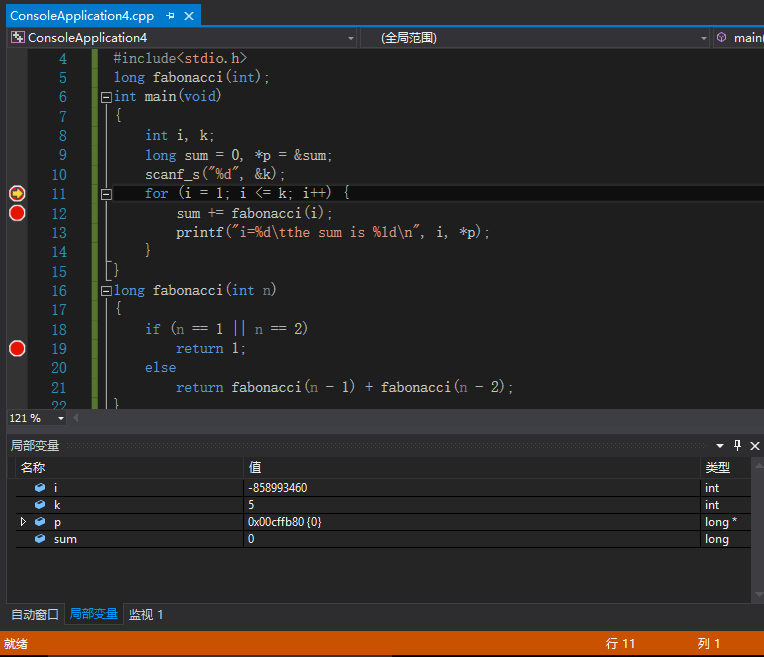


图3-5 实验三必做题3.2.3调试（1）

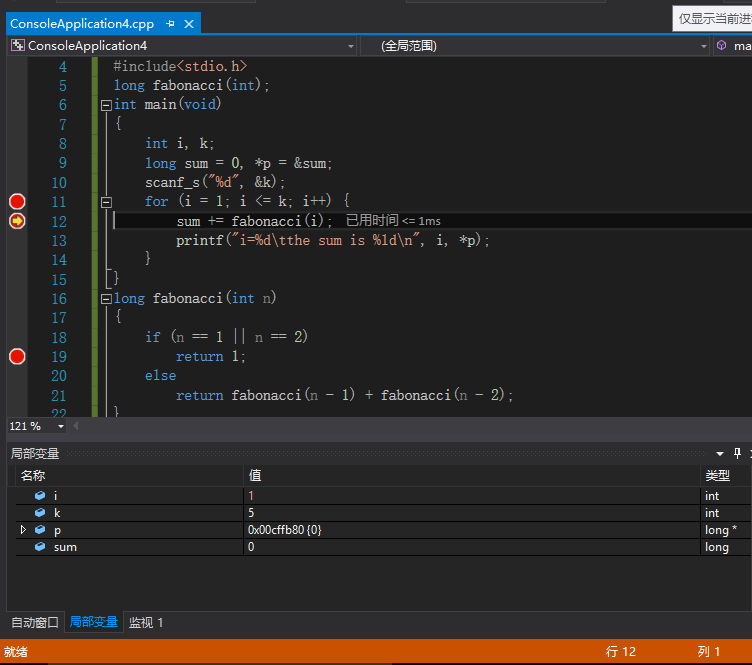


图3-6 实验三必做题3.2.3调试（2）

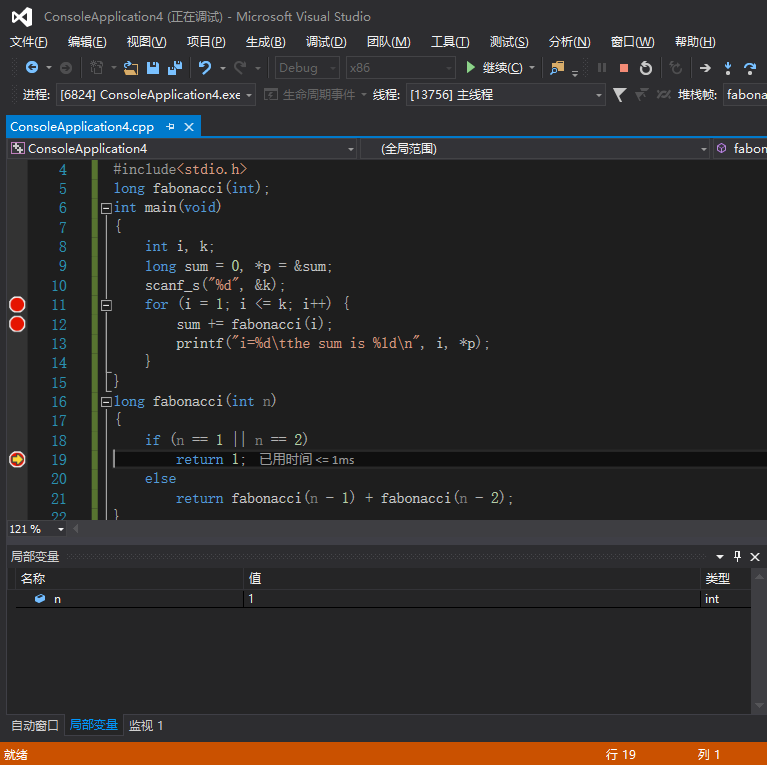


图3-7 实验三必做题3.2.3调试（3）

### 3.2.4 最大公约数

**【题目】**

编写一个程序,让用户输入两个整数，计算两个数的最大公约数并且输出之。要求用递归函数实现求最大公约数,同时以单步方式执行该程序，观察其递归过程。

**【解题思路】**

求解两个数的最大公约数并且使用递归思想的方法就是更相损减法，两个数a，b相减，得到结果c；然后再把b赋给a，c赋给b，执行相同的过程；但需要注意的是两个数a，b需要进行排序。其算法流程图如图

**【算法流程图】**

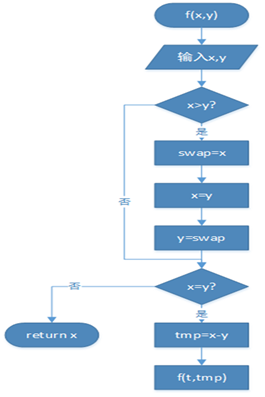
****

图3-9 实验三必做题3.2.4算法流程图

**【程序清单】**

#include<stdio.h>

int findCommonFactor(int x, int y);

int main()

{

int num1, num2;

scanf("%d %d", &num1, &num2);

if (num1 != 0)

{

int cf = findCommonFactor(num1, num2);

printf("%d\n", cf);

}

return 0;

}

int findCommonFactor(int x, int y)

{

int temp;

if(x<y)

{

int m=x;

x=y;

y=m;

}

if(x!=y)

{

temp=x-y;

findCommonFactor(y, temp);

}

else return x;

}

**【测试数据、测试理论结果及测试结果】**

**考虑到有x>y，x=y，x<y这三种情况，设计了如下三组样例进行测试。**

表3-1 实验二必做题3.2.4测试数据、测试理论结果及测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| 样例 | 测试数据 | 测试理论结果 | 测试结果 |
| （1） | 45, 25 | 5 | 如图3-10 |
| （2） | 75， 150 | 75 | 如图3-11 |
| （3） | 15, 15 | 15 | 如图3-12 |

图3-10 实验三必做题3.2.4 样例（1）测试结果

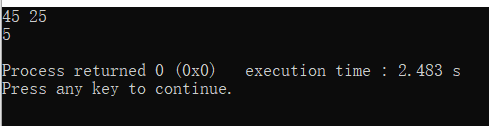


图3-11 实验三必做题3.2.4 样例（2）测试结果

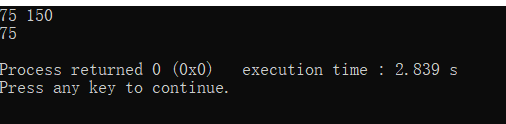
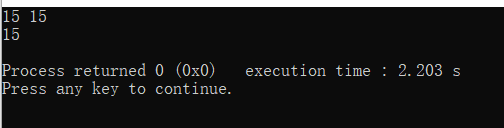


图3-12 实验三必做题3.2.4 样例（3）测试结果



### 3.2.5 验证哥德巴赫猜想

**【题目】**

编写一个程序，验证哥德巴赫猜想：一个大于等于4的偶数都是两个素数之和。

**【算法流程图】**

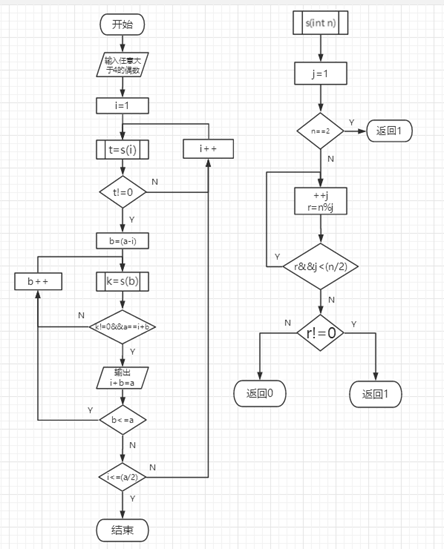


图3-13 实验三必做题3.2.5算法流程图

**【程序清单】**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int zhishu (int m)

{

int i;

if (m==2) return 1;

if(m%2==0) return 0;

for(i=3;i<=sqrt(m);i+=2)

{

if(m%i==0)

return 0;

}

return 1;

}

int main()

{

int a,b,c;

scanf("%d",&c);

a=2;

b=c-a;

for(a=2;a<=b;a++)

{

if (zhishu(a)==0) continue;

else

{

b=c-a;

if (zhishu(b)==1) break;

}

}

printf("%d=%d+%d\n",c,a,b);

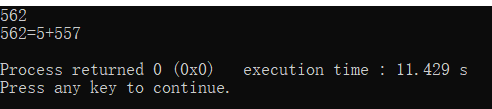
}

**【测试数据、测试理论结果及测试结果】**

表3-2 实验二必做题3.2.5测试数据、测试理论结果及测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 样例 | 测试数据 | 测试理论结果 | 测试结果 | 备注 |
| （1） | 562 | 562=5+557 | 如图3-14 | 三位数 |
| （2） | 1000 | 3+97 | 如图3-15 | 四位数 |
| （3） | 12 | 12=5+7 | 如图3-16 | 二位数 |

图3-14 实验三必做题3.2.5样例（1）测试结果



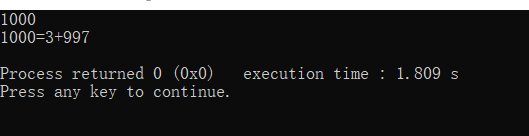


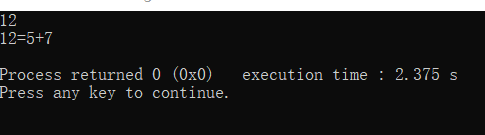
图3-15 实验三必做题3.2.5样例（2）测试结果

图3-16 实验三必做题3.2.5样例（3）测试结果

### 3.2.6 验证部分哥德巴赫猜想

**【题目】**

编写一个程序,证明对于在符号常量BEGIN和END之间的偶数哥德巴赫猜想成立。

**【算法流程图】**

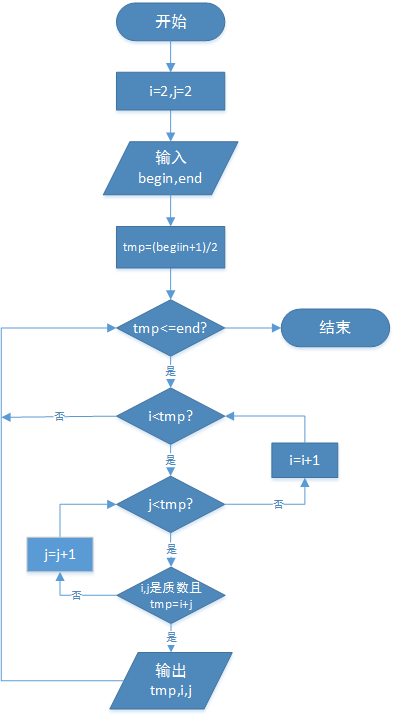
****

图3-17 实验三必做题3.2.6 算法流程图

**【程序清单】**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int zhishu (int m)

{

int i;

if (m==2) return 1;

if(m%2==0) return 0;

for(i=3;i<=sqrt(m);i+=2)

{

if(m%i==0)

return 0;

}

return 1;

}

int main()

{

int a,b,c,BEGIN,END;

printf("GOLDBACH'S CONIJECYURE:\n");

printf("Every even number n>=4 is the sum of two primes.\n");

scanf("%d %d",&BEGIN,&END);

if(BEGIN%2==0)

c=BEGIN;

else c=BEGIN+1;

while(c>=BEGIN&&c<=END)

{

a=2;

b=c-a;

for(a=2;a<=b;a++)

{

if (zhishu(a)==0) continue;

else

{

b=c-a;

if (zhishu(b)==1) break;

}

}

printf("%d=%d+%d\n",c,a,b);

c+=2;

}

}

**【测试数据、测试理论结果及测试结果】**

表3-3 实验二必做题3.2.6测试数据、测试理论结果及测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 样例 | 测试数据 | | 测试理论结果 | 测试结果 | 备注 |
| Begin | End |
| （1） | 4 | 12 | 4=2+2 6=3+3  8=3+5 10=3+7  12=5+7 | 如图3-18 | Begin与End都是偶数 |
| （2） | 5 | 13 | 6=3+3 8=3+5  10=3+7 12=5+7 | 如图3-19 | Begin与End都是奇数 |
| （3） | 8 | 15 | 8=3+5 10=3+7  12=5+7 14=3=11 | 如图3-20 | Begin是偶数，End是奇数 |
| （4） | 7 | 14 | 8=3+5 10=3+7  12=5+7 14=3=11 | 如图3-21 | Begin是奇数，End是偶数 |

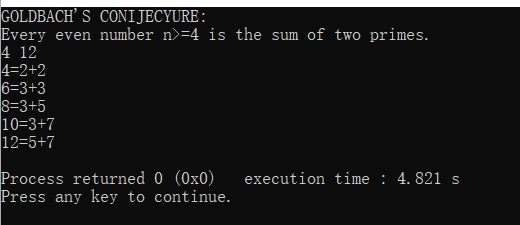


图3-18 实验三必做题3.2.6样例（1）测试结果

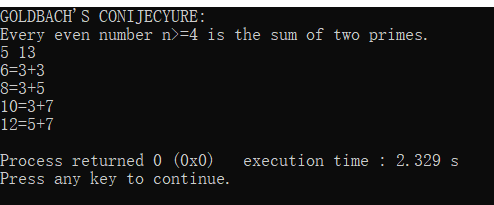


图3-19 实验三必做题3.2.6样例（2）测试结果

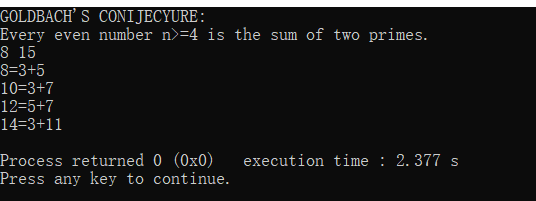


图3-20 实验三必做题3.2.6样例（3）测试结果

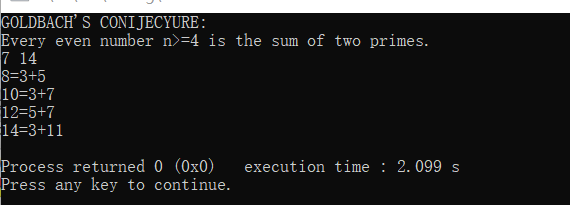


图3-21 实验三必做题3.2.6样例（4）测试结果

## 3.3 选做题

### 3.3.1 建立多源文件程序

**【题目】**

假设一个C程序由file1.c和file2.c两个源文件及一个file.h头文件组成，file1.c、file2.c和file.h的内容分别如下所述。试编辑该多文件C程序，并编译和链接。然后运行生成的可执行文件。

源文件file1.c的内容为：

#include "file.h"

int x, y; /\* 外部变量的定义性说明 \*/

char ch; /\* 外部变量的定义性说明 \*/

int main(void)

{

x = 10;

y = 20;

ch = getchar();

printf("in file1 x=%d,y=%d,ch is %c\n", x, y, ch);

func1();

return 0;

}

源文件file2.c的内容为：

#include "file.h"

void func1(void)

{

x++;

y++;

ch++;

printf("in file2 x=%d,y=%d,ch is %c\n", x, y, ch);

}

头文件file.h的内容为：

#include <stdio.h>

extern int x, y; /\* 外部变量的引用性说明 \*/

extern char ch; /\* 外部变量的引用性说明 \*/

void func1(void); /\* func1函数原型 \*/

**【方法】**

按照教材上的一步步建立多源文件，从建立工程，去除main函数，写入头函数与两个源文件，编译器自动分类不同文件，当运行file1的时候就会调用file.h头文件，和file2中的funcl函数，如当输入1时，则此时x=10，y=20，ch=1,输出此时的值，之后跨文件调用file2.c的funcl函数，x=11，y=21，ch=2，并输出，运行结果如图

**【运行结果】**

图3-22 实验三选做题运行结果

## 3.4 小结

在源程序修改替换中，要求更好的优化算法。于是借助秦九韶算法的思想，设计了一套求阶乘的简单算法，减少了大半的计算量。同时，巧妙地使用for循环一句代码完成了整个函数而又不失效率，体会到了代码之美。跟踪调试中，通过逐步调试体会到了迭代的思想；选做题中，了解了头文件与源文件的联系与区别，同时在自设题中联系了这种将源文件与头文件分开封装的方法，更深的体会到这种方法。在验证哥德巴赫猜想中，体会到了计算机在其他领域的应用以及发展前景。