**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 **实验内容及结果**

**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。

**源程序：**

#include<stdio.h>

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;

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

fac\*=i;

s+=fac;

return s;

}

【**分析及改正**】本程序共存在3处错误，分析如下：

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

错误原因：sum\_fac()使用前未声明。

改正：在#include<stdio.h>后面加上 long sum\_fac(int);

（2）

错误原因：缺少输入

改正：在int k;后面加scanf(“%d”,&k)

（3）for(k=1;k<6;k++)

错误原因：只需有一次输出；

改正：删除此行

（3）for(i=1;i<=n;i++)

fac\*=i;

错误原因：只算了n的阶乘，而不是从一到n的阶乘求和,且使用fac前未初始化。

改正:for(i=1;i<n;i++){

fac=1;

for(int j=1;j<=i;j++)

fac\*=i;

s+=fac;

}

**3.2.2** 源程序修改替换

（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



**正确的原程序代码：**

#include<stdio.h>

long sum\_fac(int);

int main(void)

{

int k;

scanf("%d",&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;

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

fac=1;

for(int k=1;k<n;k++)

fac\*=k;

s+=fac;

}

return s;

}

（1） **【原程序分析】**源程序使用了两层嵌套循环，每次计算n! 的时候都要从头算起。

**【替换方案】**使用变量fac表示n!，只用一层循环即可。

**【程序】**

#include<stdio.h>

long sum\_fac(int);

int main(void)

{

int k;

scanf("%d",&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;

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

s+=(fac\*=i);

return s;

}

【**测试**】如果输入 6，应该输出：k=6 the sum is 873，运行结果如图2.1所示，程序正确。

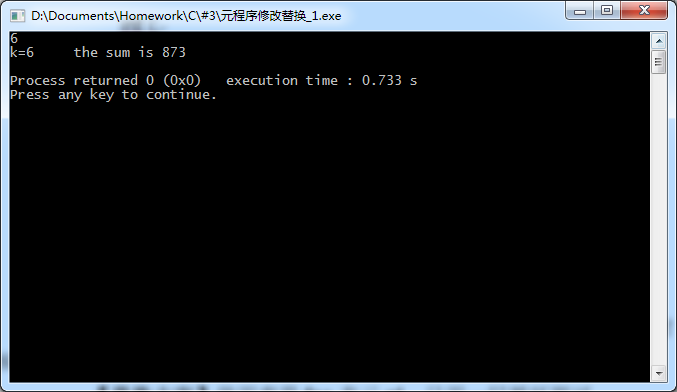


图3.1 程序运行结果截图

（2）【原程序分析】源程序使用fac表示阶乘，计算阶乘的和。

【替换方案】新增一个浮点型变量表示阶乘的倒数，并将程序修改为计算倒数的和，注意输出格式和函数的返回值类型也要修改。

【程序】

#include<stdio.h>

float sum\_fac(int);

int main(void)

{

int k;

scanf("%d",&k);

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

return 0;

}

float sum\_fac(int n)

{

float s=0;

int i;

long fac;

float i\_fac;

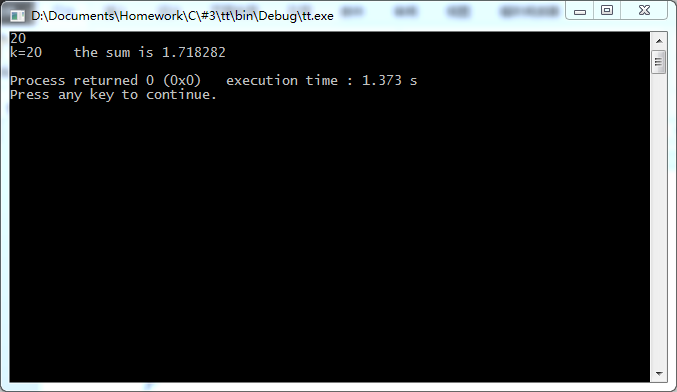
for(i=1,fac=1;i<=n;i++)

fac\*=i,s+=(i\_fac=1.0/(float)fac);

return s;

}

【测试】输入20，结果应该为自然对数的底减一后保留小数点后6位的结果



输出与预想一致

**3.2.3** 跟踪调试

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？

**源程序：**

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);

}

return 0;

}

long fabonacci(int n)

{

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

return 1;

else

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

}

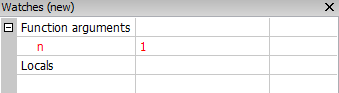
其中，“ long sum=0,\*p=&sum; ”声明p为长整型指针并用&sum取出sum的地址对p初始化。\*p表示引用p所指的变量（\*p即sum）。

【跟踪】改正程序的错误后，输入10，进行跟踪

1. p为0x28ff10，i为2

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

（3）显示的n为1



（4）n的值从3变为2变为1再变为3，实际上不是同一个n，是函数被自身调用时传递的不同的n。

**3.2.4程序设计**

编写并上机调试运行能实现以下功能的程序。

(1) 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。

(2) 编写一个程序,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。

(3) 编写一个程序,证明对于在符号常量BEGIN和END之间的偶数这一猜测成立。例如，如果BEGIN为10，END为20，则程序的输出应为：

GOLDBACH'S CONJECTURE:

Every even number n>=4 is the sum of two primes.

10=3+7

12=5+7

……

20=3+17

（1）【分析】利用辗转相除法和递归算法即可

【程序】

#include<stdio.h>

int biggest\_common\_factor(int i,int j);

int main(void){

int i,j;

scanf("%d %d",&i,&j);

printf("The biggest common factor is %d\n",biggest\_common\_factor(i,j));

return 0;

}

int biggest\_common\_factor(int i, int j){

if(j<=i)

if(i%j==0)

return(j);

else

return (biggest\_common\_factor(j,i%j));

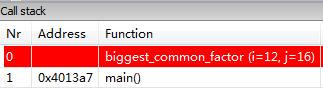
else

return(biggest\_common\_factor(j,i));

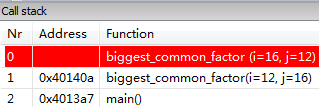
}

【跟踪】输入12 16，并进行跟踪，递归函数压栈结果如下：

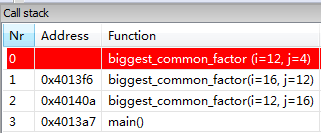
1、



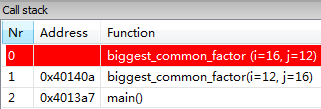
2、



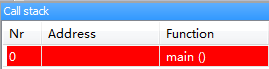
3、



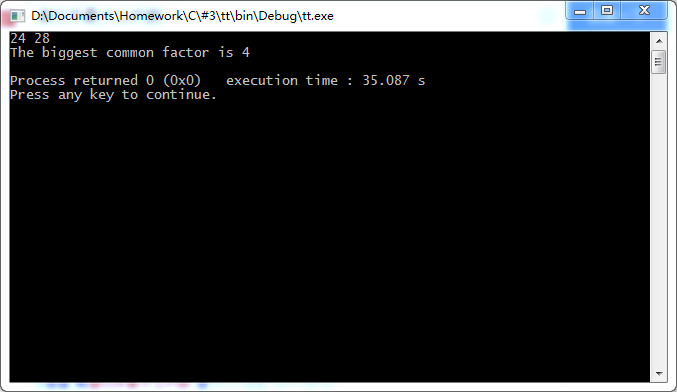
4、



5、



【测试】输入24 28 应该输出最大公约数为4



输出与预想一致

（2）【分析】主要函数是判断一个数是否为素数，以及输入的数减去它是否为素数。

【程序】

#include<stdio.h>

#include<math.h>

#define Y 1

#define N 0

int is\_Prieme(int n);

int main(void){

int to\_judge,i;

scanf("%d",&to\_judge);

for(i=2;i<to\_judge;i++)

if(is\_Prieme(i) && is\_Prieme(to\_judge-i)){

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

break;

}

return 0;

}

int is\_Prieme(int n){

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

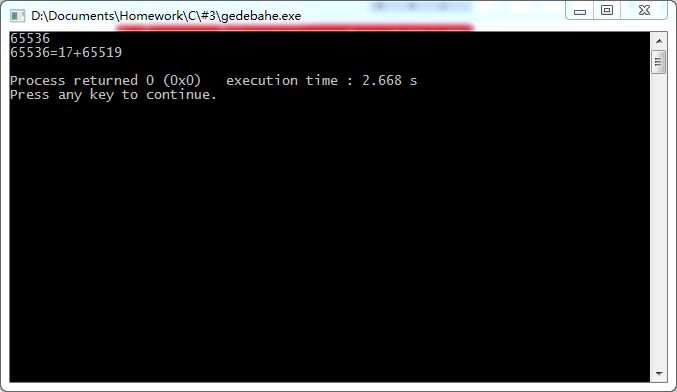
if(n%i==0)

return N;

return Y;

}

【测试】输入65536 应输出65536=17+65519



结果与预想一致

（3）【分析】秩序在第二题基础上加上从BEGIN到END之间的循环即可

【程序】

#include<stdio.h>

#include<math.h>

#define Y 1

#define N 0

#define BEGIN 10

#define END 20

int is\_Prieme(int n);

int main(void){

int to\_judge,i;

for(int to\_judge=BEGIN;to\_judge<=END;to\_judge+=2){

for(i=2;i<to\_judge;i++)

if(is\_Prieme(i) && is\_Prieme(to\_judge-i)){

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

break;

}

}

return 0;

}

int is\_Prieme(int n){

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

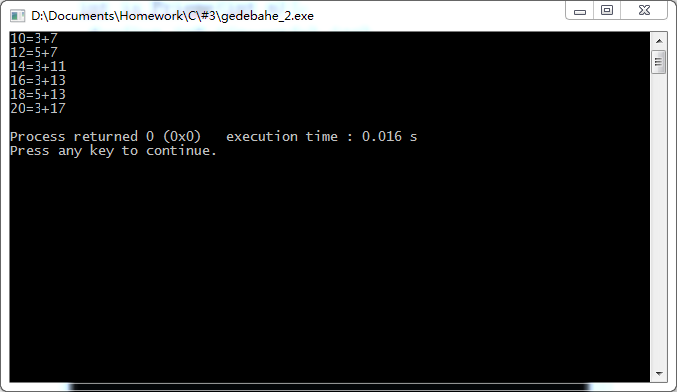
if(n%i==0)

return N;

return Y;

}

【测试】



输出与预想结果一致

**3.2.5选做题**

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 \*/

**【分析】**除了file.h中没有防止重定义的语句，加上即可

**【修正】**file.h

#ifndef FILE\_H\_INCLUDED

#define FILE\_H\_INCLUDED

#include <stdio.h>

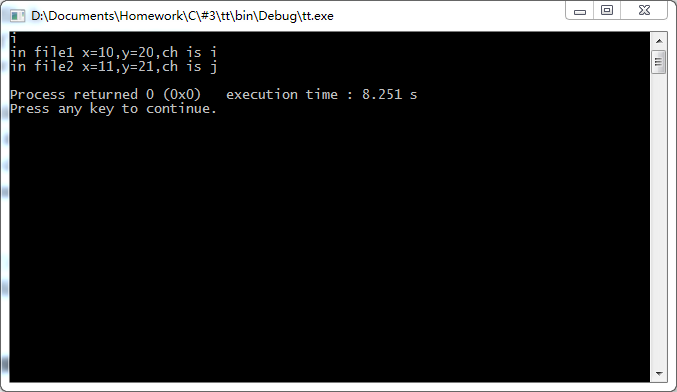
extern int x,y; /\* 外部变量的引用性说明 \*/

extern char ch; /\* 外部变量的引用性说明 \*/

void func1(void); /\* func1函数原型 \*/

#endif // FILE\_H\_INCLUDED

**【测试】**输入i，file1中的printf先运行，并调用file2中的程序。输出结果如下：



输出与预想一致

**3.3 实验体会**

通过这一次实验，我们学会了定义、声明、调用与参数传递、返回值类型的定义和返回值使用。掌握了不同存储类型变量的使用。尤其是在多文件编译技术中的全局变量的使用。了解到了头文件防止重定义的重要性，并且了解到了更多易犯的问题，例如大小写敏感问题等等。更重要的是，我学会了优化程序的基本方法，学会了寻找同时降低时间和空间复杂度的方法，对程序的性能做出了很大的改善 。