实验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 源程序改错题

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。

#include "stdio.h"

**// long sum\_fac(int);**

void main(void)

{

int k;

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

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

}

long sum\_fac(int n)

{

long s=0;

int i;

long fac;

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

fac\*=i;

s+=fac;

**/\***

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

**\* fac\*=i;**

**\* s+=fac;**

**\* }**

**\*/**

return s;

}

3.2.2 源程序修改替换题

（1）修改第1题中sum\_fac函数，使其计算量最小。

**源程序：**

long sum\_fac(int n) {

static sum;

static tmp;

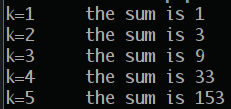
tmp \*= n;

sum += tmp;

return sum;

}

**运行结果截图：**



（2）修改第1题中sum\_fac函数，计算。

**源程序：**

#include "stdio.h"

double sum\_fac(int);

void main(void) {

int k;

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

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

}

}

double sum\_fac(int n) {

double sum = 0;

int i;

long fac = 1;

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

fac \*= i;

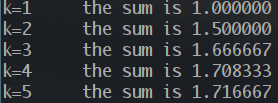
sum += (1./fac);

}

return sum;

}

**运行结果截图：**



3.2.3 跟踪调试题

计算fabonacci数列前n项和的程序如下：

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）。

#include “stdio.h”

long fabonacci(int);

void 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);

}

单步执行程序，观察p,i,sum,n值。

（1）刚执行完scanf("%d",&k);语句，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，watch窗口显示的是什么？

（4）当i=3，从调用fabonacci函数到返回，n值如何变化？

**解答：**

（1）p = (long int \*) 0x28ff10; i = 2

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

（3）p = (long int \*) 0x28ff10; i = 2; sum = 0; n = 1

（4）2（上次遗留的值） -> 3（第一层） -> 2 -> 1 -> 3（回到第一层）

3.2.4 编程设计题

（1）编程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，观察递归过程。

**递归算法思路：**

1. 定义全局变量result
2. 传入两个参数a, b
3. 使result为a, b中较小的那个数
4. 判断：

result是公约数：返回result

result不是公约数：result-1，再次调用判断函数

（这里，参与递归过程的函数只判断result是否是a, b的公约数）

**源程序清单：**

#include "stdio.h"

unsigned int result;

unsigned int commonDivisor(unsigned, unsigned);

void main(void) {

unsigned int a, b;

while (

scanf("%u %u", &a, &b),

getchar(),

a!=0 && b!=0

) {

result = 0;

printf("%u\n", commonDivisor(a, b));

}

}

unsigned int commonDivisor(unsigned a, unsigned b) {

if (a > b) {

b += a;

a = b - a;

b -= a;

}

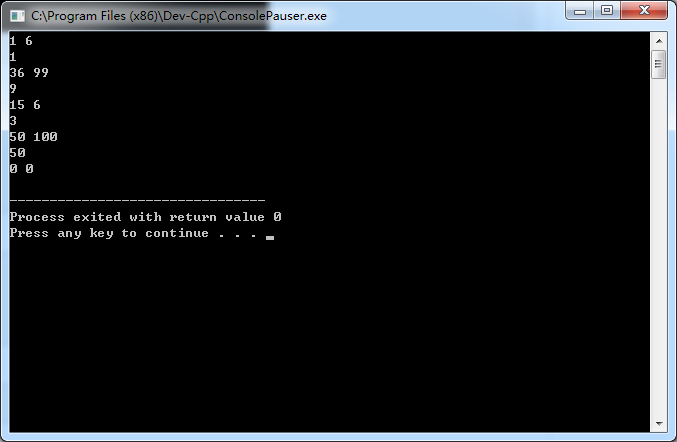
if ( result == 0 ) { result = a; }

if ( !(b%result) && !(a%result) ) { return result; }

else { --result; return commonDivisor(a, b); }

}

**运行结果截图：**



（2）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。

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，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

**算法流程：**

1. 输入范围
2. 对范围中每一个偶数current（程序中直接使用begin）

将循环变量i初始化为最小质数——2

判断i与current-i是否均为质数：

是：哥德巴赫猜想对偶数current正确

否：i+1，继续判断

**源程序清单：**

#include "stdio.h"

unsigned char isPrime(unsigned int);

void main(void) {

puts("GOLDBACH'S CONJECTURE:");

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

unsigned int begin, end;

unsigned int i;

while (

scanf("%u %u", &begin, &end),

getchar(),

(begin!=0) && (begin>=6) && (end>=begin)

) {

if ( begin % 2 ) { begin = begin / 2 \* 2 + 2; }

for (; begin <= end; begin+=2) {

for (i = 2; i < begin; ++i) {

if ( isPrime(i) && isPrime(begin-i) ) {

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

break;

}

}

}

putchar('\n');

}

}

unsigned char isPrime(unsigned x) {

unsigned i = 2;

unsigned max = x/2 + 1;

for (; i <= max; ++i) {

if ( !(x%i) ) { break; }

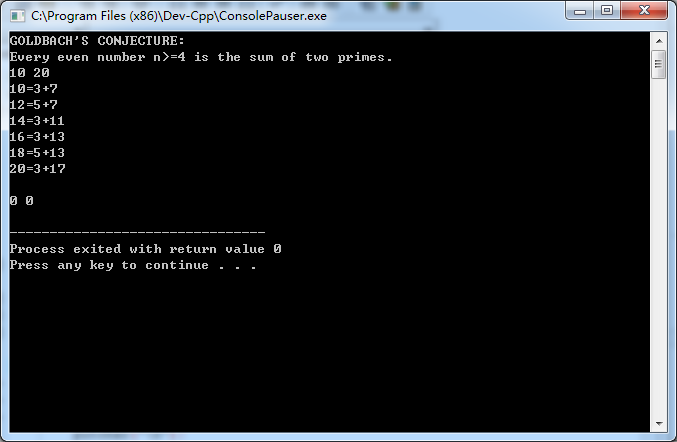
}

if (i==max+1) { return 1; }

else { return 0; }

}

**运行结果截图：**



3.2.5 选做题

设file1.c如下：

#include <stdio.h>

#include "file2.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 \*/

void main(void)

{

x=10;

y=20;

ch=getchar();

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

func1();

}

file2.c如下：

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)

{

x++;

y++;

ch++;

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

}

file2.h如下：

#include <stdio.h>

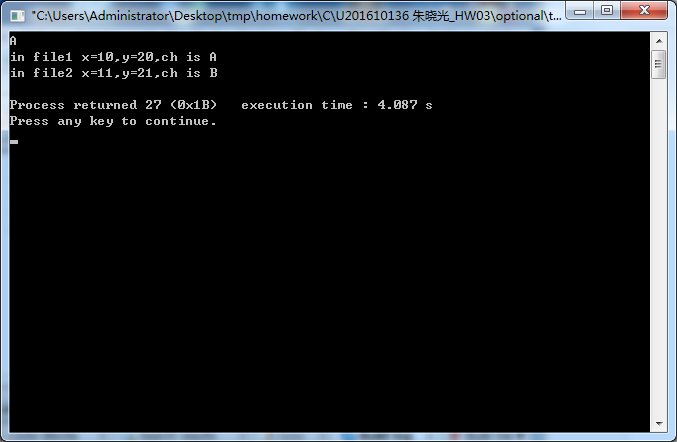
extern int x,y;

extern char ch;

void func1(void);

试用TCC进行多文件编译和链接。然后在DOS环境下运行生成的可执行文件。

**运行结果截图：**



3.3 自设题

探索数组名与各元素地址之间的关系，并探索Linux下项目的编译和链接。

**源程序清单：**

**/\* first.c \*/**

#include "stdio.h"

#include "second.h"

int \* copy(int \*);

void main(void) {

int array[5] = {1, 2, 3, 4, 5};

printf("%p -> %d\n", &array[2], array[2]);

printf("%p -> %d\n",&copy(&array[1])[2], copy(&array[1])[2]);

}

**/\* second.h \*/**

int \* copy(int \*);

**/\* second.c \*/**

#include "stdio.h"

int \* copy(int \* name) {

return name;

}

**/\* makefile \*/**

main: first.c second.h second.c

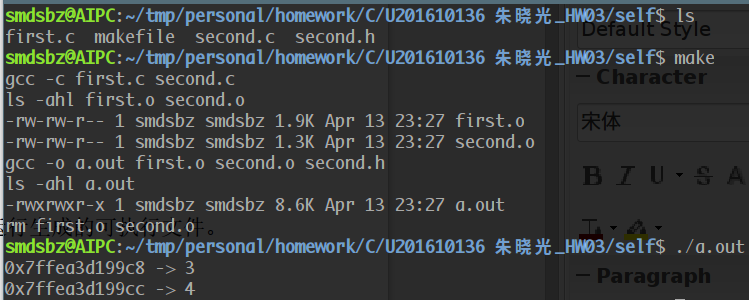
gcc -c first.c second.c

ls -ahl first.o second.o

gcc -o a.out first.o second.o second.h

ls -ahl a.out

rm first.o second.o

**运行结果截图：**

3.4 实验小结

通过本次实验，我学习了多文件编译、链接的基本方法。期间还发现了原题目中的一些错误，加深了对头文件作用的理解与对常见错误的认识。