# 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"

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;

return s;

}

**解答：**

（1）错误修改：

1) 未在主函数前声明sum\_fac(int n)函数正确形式为：

long sum\_fac(int n);

2) 第2行的voidmain( void )有误，正确形式为：

int main( void )

3) 第10行的fac未定义，正确形式为：

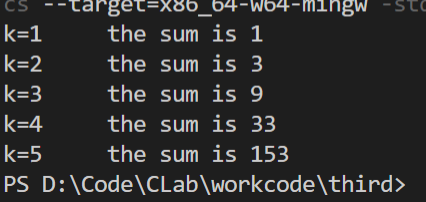
long fac=1;

4) 第11行的for循环需要修改，正确形式为：

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

{fac\*=i; s+=fac;}

（2）错误修改后运行结果如图3-1所示：



**图3-1 改错题的测试的运行结果**

**3.2.2 修改替换题**

**（1）**修改第1题中sum\_fac函数，使其计算量最小。

**解答：**

#include <stdio.h>

long sum\_fac(int n);

long s=0;

long fac = 1;

int 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){

fac\*=n;

s+=fac;

return s;

}

**（2）**修改第1题中sum\_fac函数，计算。

#include <stdio.h>

double sum\_fac(int n);

double s=0;

double fac = 1;

int 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){

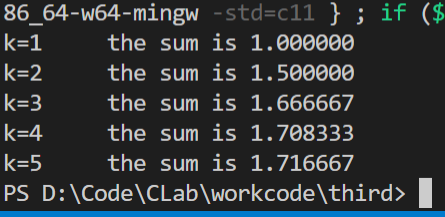
fac \*= n;

s += 1/fac;

return s;

}

替换后运行结果如图3-2所示：



**图3-2 替换题的测试的运行结果**

### 3.2.3 跟踪调试题

下面是计算fabonacci数列前n项和的程序，要求单步执行程序，观察p,i,sum,n值。

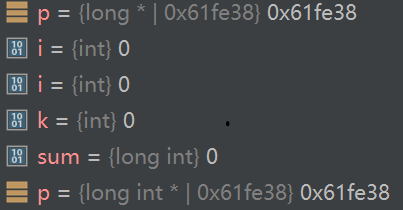
**源程序：**

1. #include <stdio.h>
2. long fabonacci(int);
3. int main(void)
4. {
5. int i,k;
6. long sum=0,\*p=&sum; /\* 声明p为长整型指针，指向sum \*/
7. scanf("%d",&k);
8. for(i=1;i<=k;i++){
9. sum+=fabonacci(i);
10. printf("i=%d\tthe sum is %ld\n",i,\*p); /\* \*p等价于sum \*/
11. }
12. return 0;
13. }
14. long fabonacci(int n)
15. {
16. if(n==1 || n==2)
17. return 1;
18. else
19. return fabonacci(n-1)+fabonacci(n-2);
20. }

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）。

1. 刚执行完scanf("%d",&k);语句，p,i值是多少？

答：执行完该语句，进入for循环前，p值为0x61fe38, i为0.//此处有疑问，i，k均为0，可能由于gcc编译器的优化选项引起。



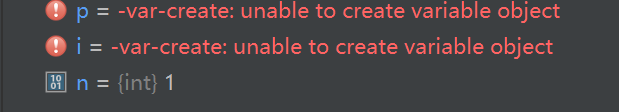
**图3-3 跟踪调试题的测试结果（1）**

1. 从fabonacci函数返回后光条停留在哪个语句上？

答：从fabonacci函数返回后光条停留在printf("i=%d\tthe sum is %ld\n",i,\*p);上

1. 进入fabonacci函数，watch窗口显示的是什么？

答：watch窗口显示了n。



**图3-4 跟踪调试题的测试结果（3）**

1. 当i=3，从调用fabonacci函数到返回，n值如何变化？

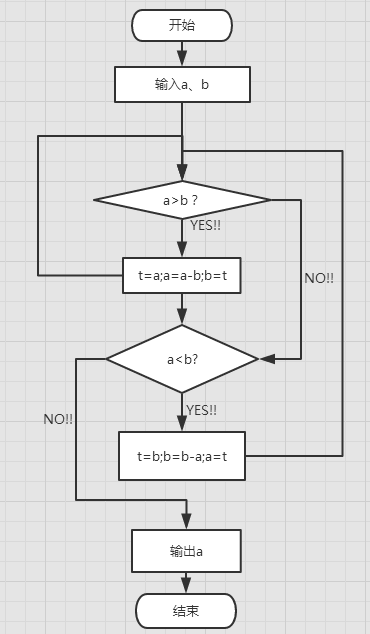
答：从3到2到1，然后返回。

**3.2.4程序设计**

（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。

**解答：**

1. 解题思路：



**图3-5 编程题1的算法流程图**

2）源程序清单

#include <stdio.h>

int gcd(int x, int y);

int main(void)

{

int x, y;

scanf("%d%d", &x, &y);

printf("The gcd of %d and %d is %d\n", x, y, gcd(x, y));

return 0;

}

int gcd(int x, int y)

{

if (x > y)

return gcd(x - y, y);

else if (x < y)

return gcd(x, y - x);

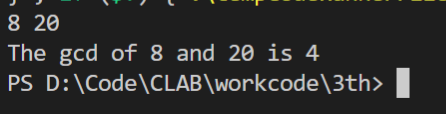
else

return x;

}

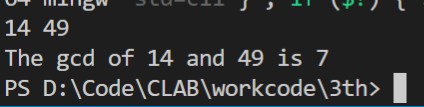
3）测试

对应测试数据的运行结果截图



**图3-6 编程题1的测试运行结果（1）**

对应测试测试用例2的运行结果如图1-3-3所示。



**图3-7 编程题1的测试运行结果（2）**

**（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

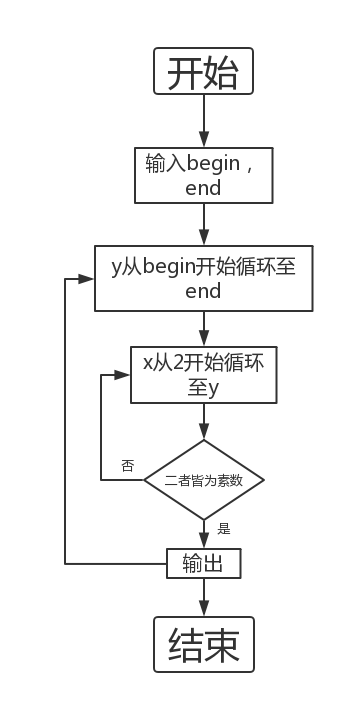
12=5+7

……

20=3+17

**解答：**

1) 解题思路：



**图3-8 编程题2的算法流程图**

2）程序清单

#include<stdio.h>

#include<math.h>

#define BEGIN 4

#define END 20

int IsPrime(int n);

int main(void){

int i, j, k = 0;

printf("GOLDBACH'S CONJECTURE:\nEvery even number n>=4 is the sum of two primes.\n");

for (j = BEGIN; j <= END; j += 2){

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

if(IsPrime(i) && IsPrime(j-i))

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

k++;

if(!(k%5))

break;

}

}

return 0;

}

int IsPrime(int n){

int i;

if(n == 1)

return 0;

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

if(n%i == 0)

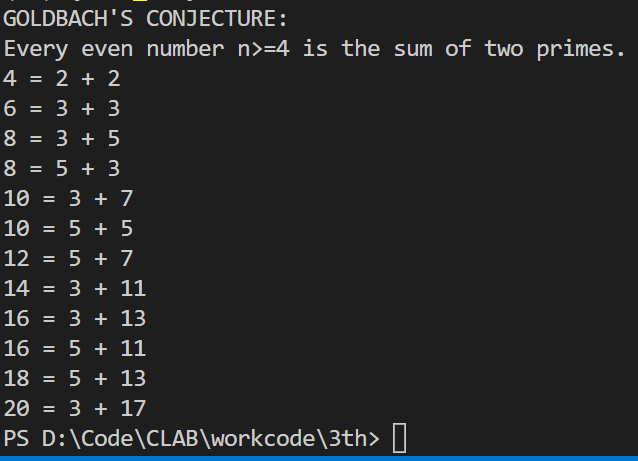
return 0;

return 1;

}

3）测试

对应测试的运行结果如图3-9所示。



**图3-9 编程题2的测试运行结果**

## 3.3 实验小结

第三次C语言上机实验，主要掌握并熟悉了函数的定义、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、不同存储类型变量的使用以及熟悉多文件编译技术。

**（1）改错实验**

在改错实验中，主要学习到了对函数的基本使用。对一些基本概念有了一定的了解。认识到了函数在结构化程序设计中的重要意义。函数可以把大的计算过程分解成若干个小的任务。程序设计人员可以以函数为基础，进一步编写程序，而无须重复编写相同代码。同时，一个设计优秀的函数还可以将程序中不需了解或可不关心的计算过程隐藏，做到“黑箱”效果，使程序结构更加清晰。

**（2）程序修改替换实验**

在修改替换题中，第一题开始我使用全局变量，将每次的阶乘以及阶乘和都用到了下一次的计算中。经过助教学长点拨，后来更换成为了局部静态变量，也意识到了随意使用全局变量的缺陷。第二题继承第一题方式，略作修改完成。

**（3）跟踪调试题**

此题分别使用gdb与CLion中的调试器完成，gdb由于gcc优化选项问题无法查看到两变量值。CLion中，p显示出了sum在变量中的地址，而代码中原来未初始化的变量值全为0，考虑也许是编译器优化问题。做完此题，感受到了现代编译器对代码的优化技术之深奥与强大。

**（4）程序设计实验**

第一题在使用递归技术的同时，领会到了更相减损术的思想，感受到了中国古代数学研究的美丽，同时通过观察递归中函数调用栈的变化，对递归有了进一步了解。第二，三题合并为一题完成。此题起初并未考虑到一个偶数可能有多种拆分方式的问题，在后续网上提交作业系统中加以改正.

1. **选做**

通过此题，掌握到了利用gcc与CLion进行多文件编译的方式。

通过此次C语言上机实验，我感到编程是需要不断练习，才可以得到进步的。