# 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。

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 }

解答:

1. 错误修改:
2. 第2行的主函数之前应该声明自定义函数sum\_fac，因为自定义函数被写在主函数的后面，正确形式为在1、2行之间添加：

long sum\_fac();

1. 第12行的长整形变量fac应给其赋初值1，否则在执行第14行的fac\*=i时候不能得到需要的结果，正确形式为：

long fac=1;

1. 第13行使用for语句需要在循环语句外围添加花括号{}，否则只执行一次循环语句就将s的值返回，正确形式为：

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

{

fac\*=i;

s+=fac;

}

错误修改后源程序:

#include "stdio.h"

long sum\_fac();

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=1;

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

{

fac\*=i;

s+=fac;

}

return s;

}

1. 错误修改后运行结果如图3-1所示。

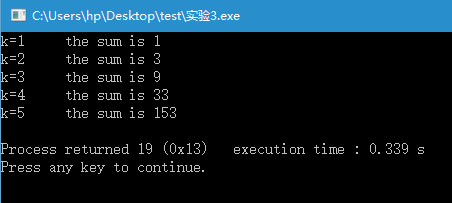


图3-1 源程序改错题 3.2.1错误修改后的运行结果

**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。

解答：

因为函数sum\_fac中的变量只需要赋一次初值，所以可用static全局变量定义变量s、i、fac，即修改sum\_fac函数为：

long sum\_fac(int n)

{

static long s=0;

static int i=1;

static long fac=1;

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

{

fac\*=i;

s+=fac;

}

return s;

}

修改后的程序如下所示：

#include "stdio.h"

long sum\_fac();

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)

{

static long s=0;

static int i=1;

static long fac=1;

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

{

fac\*=i;

s+=fac;

}

return s;

}

运行结果如图3-2所示。

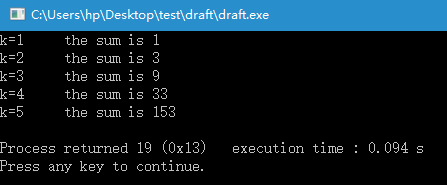


图3-2 源程序修改替换题 3.2.2.(1)sum\_fac后的运行结果

（2）修改第1题中sum\_fac函数，计算。

解答：

将对应变量s、fac以及函数sum\_fac定义为double双精度浮点数即可，即将对应部分替换为：

double sum\_fac(); doubles=0; double fac=1;

另将累加部分替换为s+=(1/fac);

替换后的程序如下所示：

#include "stdio.h"

double sum\_fac();

void main(void)

{

int k;

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

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

}

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;

}

运行结果如图3-3所示。

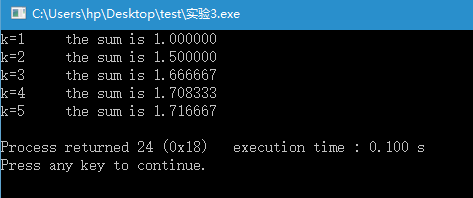


图3-3 源程序修改替换题 3.2.2.(2)替换sum\_fac后的运行结果

**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）。

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. 刚执行完scanf("%d",&k);语句，p,i值如图3-4。

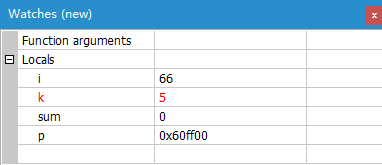


图3-4 跟踪调试题 3.2.3.(1) 刚执行完scanf("%d",&k);语句，p,i的值

1. 从fabonacci函数返回后光条停留在语句printf("i=%d\tthe sum is %ld\n",i,\*p);上，如图3-5。

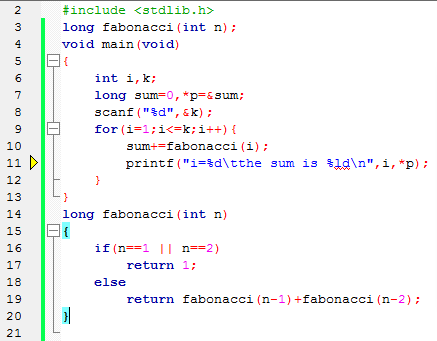


图3-5 跟踪调试题 3.2.3.(2) 从fabonacci函数返回后光条位置

1. 进入fabonacci函数，watch窗口显示的是函数中的变量n，如图3-6。

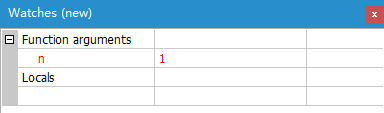


图3-6 跟踪调试题 3.2.3.(3) 进入fabonacci函数时的watch窗口

1. 当i=3，从调用fabonacci函数到返回，n值变化为3→2→1→3，如图3-7。

→

→

→

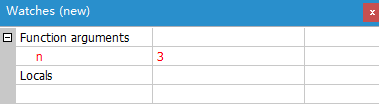


图3-7 跟踪调试题 3.2.3.(4) 当i=3，从调用fabonacci函数到返回，n值变化

**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。

解答：

1. 解题思路：
2. 提示并输入a、b；
3. 判断a、b是否符合要求：如果a、b不等于0，转2.1，否则转3；若输入文件尾，转4；
   1. 用递归思想定义求取两数最大公约数的函数max，即

int max(int a,int b,int n)

{

if(a%n==0&&b%n==0)

return n;

else

return max(a,b,n-1);

}

其中在主函数中将a赋值给n；

* 1. 利用max函数完成验证并输出后转2；

1. 显示输入错误信息；
2. 结束。
3. 程序清单：

#include<stdio.h>

int max(int a,int b,int n)

{

if(a%n==0&&b%n==0)

return n;

else

return max(a,b,n-1);

}

int main(void)

{

int a,b,n;

while(scanf("%d %d",&a,&b)!=EOF)

{

getchar();

if(a!=0&&b!=0)

{

n=a;

printf("%d\n",max(a,b,n));

}

else

{

printf("输入错误信息！\n");

return 0;

}

}

}

1. 测试
2. 测试数据：

为方便分析测试结果，选择a、b分别为1、6，36、99，50、100，同时尽量选取两端的字符。如表3-1所示。

表3-1 编程题3.2.4.(1)的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | 理 论 结 果 |
| a | b |
| 用例1 | 1 | 6 | 计算结果 1 |
| 用例2 | 36 | 99 | 计算结果 9 |
| 用例3 | 50 | 100 | 计算结果 50 |

1. 对应测试测试用例1、2、3的运行结果如图3-8所示。

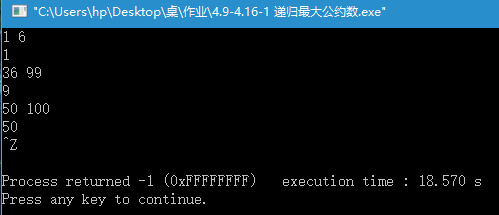


图3-8 编程题3.2.4.(2)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（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：

解答：

1. 解题思路：
2. 定义数组p[101]为0到100中的素数，用以验证100以内的歌德巴赫猜想；
3. 提示并输入开始数与结束数M、N；
4. 判断M、N是否符合要求：如果4 ≤ M且M ≤ N，转3.1，否则转4；若输入文件尾，转5；
   1. 利用for循环对M到N中的每一个数依次验证歌德巴赫猜想，即for(;M<=N;M++)；
   2. 对于每个M，定义由0开始的整形m、n，在素数表中寻找出满足条件“是大于等于四的偶数且是两个素数之和”，即if((p[m]+p[n]==M)&&(M%2==0))，的m、n值后，输出验证结果结果，即printf("%d=%d+%d\n",M,p[m],p[n])；
   3. 输出一组能够验证歌德巴赫猜想的数据后，继续通过break函数对M+1进行验证，直至M>N时，结束循环并转2；
5. 显示输入错误信息并转2；
6. 结束。
7. 程序清单：

#include<stdio.h>

int main (void)

{

int M,N;

// int p[] 2~100

int p[101],num;

p[1]=0,p[2]=2;

for(num=3;num<=100;num++)

{

int a=0,i;

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

{

if(num%i ==0)

{

a++;

}

}

if(a==0)

{

p[num]=num;

}

else

{

p[num]=0;

}

}

//proof

int m,n;

printf("请输入M、N的值：");

while(scanf("%d %d",&M,&N)!=EOF)

{

getchar();

if(M>=4&&M<=N)

{

for(;M<=N;M++)

{

for(m=0;m<M;m++)

{

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

{

if((p[m]+p[n]==M)&&(M%2==0))

{

printf("%d=%d+%d\n",M,p[m],p[n]);

break;

}

}

if((p[m]+p[n]==M)&&(M%2==0))

{

break;

}

}

}

printf("\n");

}

else

{

printf("请输入正确的M、N!\n\n");

}

printf("请输入M、N的值：");

}

}

1. 测试
2. 测试数据：

为方便分析程序准确性，选择M、N分别为4、10，10、20，90、100。如表3-2所示。

表3-2 编程题3.2.4.(2)的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | 理 论 结 果 |
| M | N |
| 用例1 | 4 | 10 | 4、6、8、10分别分为两个素数 |
| 用例2 | 10 | 20 | 10、12、14、16、18、20分别分为两个素数 |
| 用例3 | 90 | 100 | 90、92、94、96、98、100分别分为两个素数 |

1. 对应测试测试用例1、2、3的运行结果如图3-9所示。

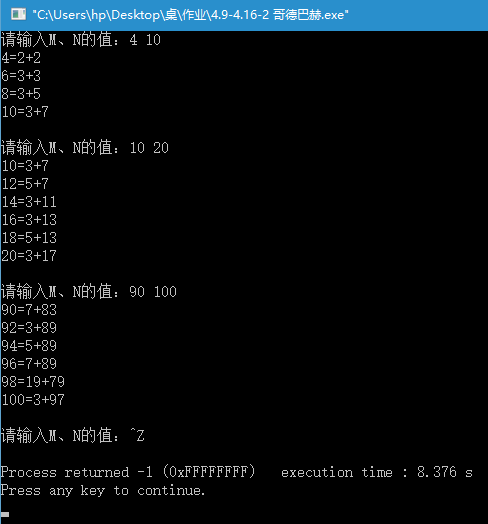


图3-9 编程题3.2.4.(2)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

**3.2.5 选做题**

设file1.c如下：

#include <stdio.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++;

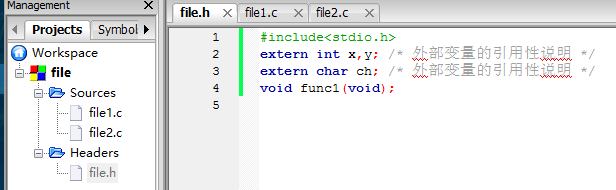
printf("in file2 x=%d,y=%d,ch is %c\n",x,y,ch);

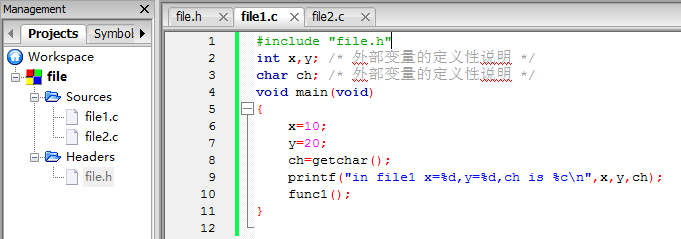
}

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。

解答：

用CodeBlocks进行多文件编译和链接如图3-10。





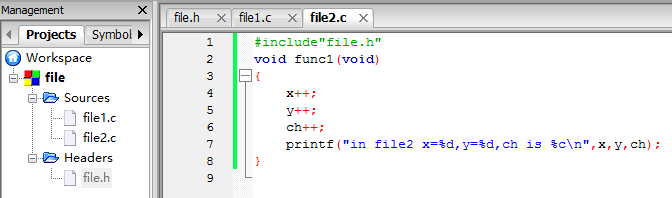


图3-10 编程题3.2.5多文件编译与链接

运行生成的可执行文件如图3-11。

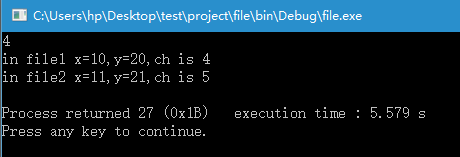


图3-11 编程题3.2.5生成的可执行文件的运行结果

## 3.3 自设题

**（1）**自设实验题目：输入一个行列式（4×3），通过函数实现该行列式的转置。

**（2）**实验目的：通过设计实验程序，熟练掌握函数的定义、声明、函数调用与参数传递方法。

**（3）**实验程序：

#include <stdio.h>

int trans(int a[3][4], int b[4][3])

{

int i, j;

for(i=0; i<3; i++)

for(j=0; j<4; j++)

b[j][i] = a[i][j];

return b;

}

void main()

{

int a[3][4];

int b[4][3];

int i, j;

printf("before transform :\n");

for(i=0; i<3; i++)

{

for(j=0; j<4; j++)

{

scanf("%d", &a[i][j]);

}

}

trans(a, b);

printf("after transform :\n");

for(i=0; i<4; i++)

{

for(j=0; j<3; j++)

{

printf("%4d", b[i][j]);

}

printf("\n");

}

}

**（4）**实验用例：如图3-12。

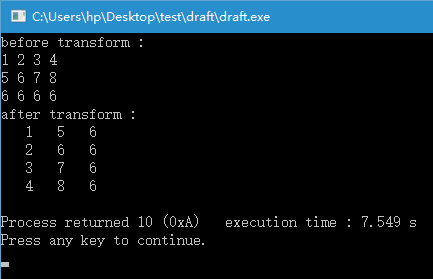


图3-12 自设题3.3测试用例及其输出

**（5）**实验结论：通过测试用例验证了程序的正确性，能够熟练掌握函数的使用。

## 3.4 实验小结

在实验中主要遇到的问题有：

1. 在3.2.1源程序改错中开始没有发现主函数前未声明自定义函数，即long sum\_fac()，编写程序时应注意自定义函数时或将函数定义置于主函数之前，或在主函数前声明自定义函数后，再在主函数后编写函数；
2. 在3.2.2. (1) 源程序修改替换时，起初并没有使用全局定义static来减少程序的计算量，而是使用了递归思想来改造函数，后在仔细思考后用新的方法解决了问题；
3. 在解决问题3.2.3跟踪调试题时，因为初次使用CodeBlocks的调试功能，操作比较不熟练，在熟悉过后能够顺利的进行断点的设置，step into，逐行执行等操作；
4. 在选做题3.2.5中最后没有成功使用TCC编译器，最后选用了CodeBlocks新建Project实现了多文件的编译和链接，并成功执行程序。

体会：

1. 在编写程序时若需要使用自定义函数，需严格注意定义函数时的格式，或将自定义函数置于主函数前，或在主函数前进行声明后再将函数定义于主函数之后。个人可以在这个方面形成良好的习惯，从而避免编程时出现不必要的错误；
2. 学习了新的调试技巧，通过在CodeBlocks中新建项目，能够使用软件中的调试功能，设置断点、逐行运行、或是跳至下一个函数，通过利用这些小技巧能够有效提升编程时修改程序、或调试错误程序的速度；
3. 通过本次实验，我熟悉和掌握函数的定义和声明、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。在编写程序时可尝试着尽可能多的去使用函数，从而使程序结构更加的清晰明了，便于本人进行调试也方便他人的阅读，但这对程序员编写函数的功底也是一个考验，所以还需要更加深入的了解函数的编写以及使用，进而改善编写出的程序；
4. 通过本次实验，尝试着进行了多文件编译与链接，了解了编译器对多文件编译的处理方式，为以后编写多文件程序打下了基础。