# 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，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）练习使用集成开发环境中的调试功能：单步执行、设置断点、观察变量值。

## 3.2 实验内容

**3.2.1 程序改错题**

下面是计算s=1!+2!+3!+…+n!的源程序(n<20)。在这个源程序中存在若干语法和逻辑错误。要求对该程序进行调试修改，使之能够输出如下结果：

k=1 the sum is 1

k=2 the sum is 3

k=3 the sum is 9

……

k=20 the sum is 2561327494111820313

/\*实验3-1改错题程序：计算s=1!+2!+3!+…+n!\*/

1. #include <stdio.h>
2. int main(void)
3. {
4. int k;
5. for(k=1;k<=20;k++)
6. printf("k=%d\tthe sum is %ld\n",k,sum\_fac(k));
7. return 0;
8. }
9. long sum\_fac(int n)
10. {
11. long s=0;
12. int i,fac;
13. for(i=1;i<=n;i++)
14. fac\*=i;
15. s+=fac;
16. return s;
17. }

**解答：**

1. 语法错误修改：

1) 没有在文件头声明自定义函数，正确形式为：

long long sum\_fac(int n) ;

2) 第6行的printf("k=%d\tthe sum is %ld\n",k,sum\_fac(k));没有使用正确的占位符，正确形式为：

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

1. 逻辑错误修改：
2. 第9行long sum\_fac(int n)函数定义的返回值类型有误，正确形式为：

long long sum\_fac(int n)

1. 第11，12行变量s和变量fac都应为long long型，正确形式为：

long long s=0,fac=1;

int i;

1. 第15行s+=fac;应该包含在for循环内，正确形式为：

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

{

fac\*=i;

s+=fac; // 该语句应该在for循环内

}

修改后，源程序清单如下：

/\*实验3-1改错题程序：计算s=1!+2!+3!+…+n!\*/

1. #include <stdio.h> // 没有声明计算阶乘和的函数，故在main()中无法进行调用
2. long long sum\_fac(int n) ; // 如上
3. int main(void)
4. {
5. int k;
6. for(k=1;k<=20;k++)
7. printf("k=%d\tthe sum is %lld\n",k,sum\_fac(k)); // 占位符改为lld
8. return 0;
9. }
10. long long sum\_fac(int n) // long long
11. {
12. long long s=0,fac=1;
13. int i; // fac,s 也应为long long类型 ，并应该赋初始值
14. for(i=1;i<=n;i++)
15. {
16. fac\*=i;
17. s+=fac; // 该语句应该在for循环内
18. }
19. return s;
20. }
21. 错误修改后运行结果如图3-1所示。

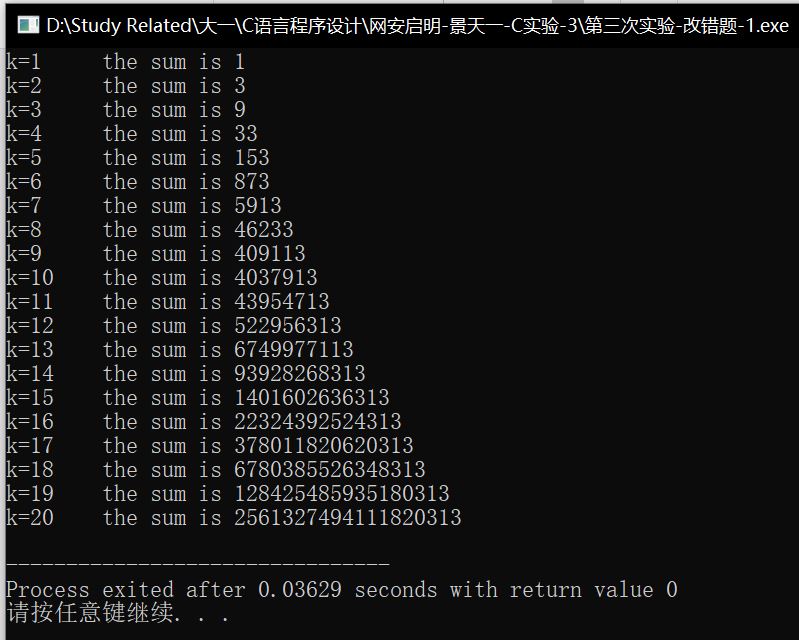
****

图3-1 实验3-1修改后运行结果截图

**3.2.2 源程序修改替换**

1. 根据将实验3-1改错题程序中sum\_fac函数修改为一个递归函数，用递归的方式计算。

**解答：**

按照要求修改后的sum\_fac()函数如下：

1. long long sum\_fac(int n)
2. {
3. if (n==1)
4. return 1 ;
5. long long fac=1, sum=0 ;
6. for (int count = 1;count<=n;count++)
7. fac \*= count ;
8. sum = fac + sum\_fac(n-1) ;
9. return sum ;
10. }

优化和替换后运行结果如图3-2所示。

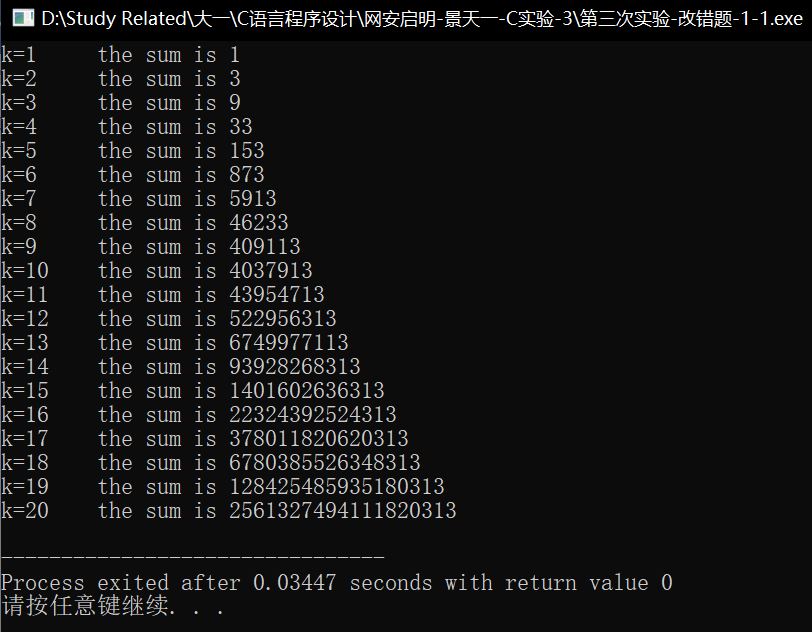


图3-2 实验3-2优化替换后运行结果截图

1. 下面是计算的源程序，其中x是浮点数，n是整数。从键盘输入x和n，然后计算s的值。修改该程序中的sum和fac函数，使之计算量最小。

/\*实验3-2程序修改替换第(2)题程序：根据公式计算 s\*/

1. #include<stdio.h>
2. double mulx(double x,int n);
3. long fac(int n);
4. double sum(double x,int n)
5. {
6. int i;
7. double z=1.0;
8. for(i=1;i<=n;i++)
9. {
10. z=z+mulx(x,i)/fac(i);
11. }
12. return z;
13. }
14. double mulx(double x,int n)
15. {
16. int i;
17. double z=1.0;
18. for(i=0;i<n;i++)
19. {
20. z=z\*x;
21. }
22. return z;
23. }
24. long fac(int n)
25. {
26. int i;
27. long h=1;
28. for(i=2;i<=n;i++)
29. {
30. h=h\*i;
31. }
32. return h;
33. }
34. int main()
35. {
36. double x;
37. int n;
38. printf("Input x and n:");
39. scanf("%lf%d",&x,&n);
40. printf("The result is %lf:",sum(x,n));
41. return 0;
42. }

**解答：**

1. 解答思路：

观察发现，在mulx()函数和fac()函数中存在重复计算的现象，可以通过将每次计算得到的值以静态变量的形式存储起来减少下次运算的计算步骤，当该函数下一次被调用时，仍可利用上一次计算得到的结果。

1. 按照要求优化后的程序源代码如下：
2. #include<stdio.h>
3. double mulx(double x) ;
4. long fac(int n);
5. double sum(double x,int n)
6. {
7. int i;
8. double z=1.0;
9. for(i=1;i<=n;i++)
10. {
11. z=z+mulx(x)/fac(i); // 去掉了一个参数
12. }
13. return z;
14. }
16. double mulx(double x)
17. {
18. static double z=1.0 ; // 利用静态变量减少多次计算的计算量
19. z = z \* x;
20. return z ;
21. }
22. long fac(int n)
23. {
24. static long h = 1 ; // 利用静态变量减少多次计算的计算量
25. h \*= n ;
26. return h ;
27. }
28. int main()
29. {
30. double x;
31. int n;
32. printf("Input x and n:");
33. scanf("%lf%d",&x,&n);
34. printf("The result is %lf:",sum(x,n));
35. return 0;
36. }
37. 测试
38. 测试数据

测试数据如表3-1所示：

表3-1 改错与替换题2的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程序输入 | | 预计输出 |
| x | n |
| 用例1 | 3 | 3 | 13.000000 |
| 用例2 | 4 | 5 | 42.866667 |

1. 对应测试测试用例1的运行结果如图3-3所示。

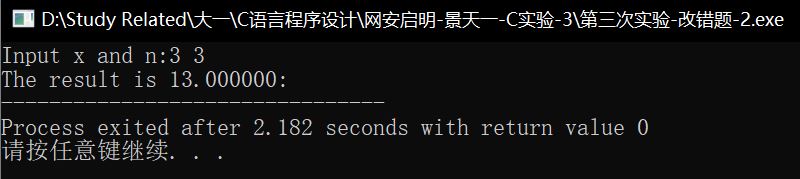


图3-3 改错与替换题2的测试用例一的运行结果

对应测试测试用例2的运行结果如图3-4所示。

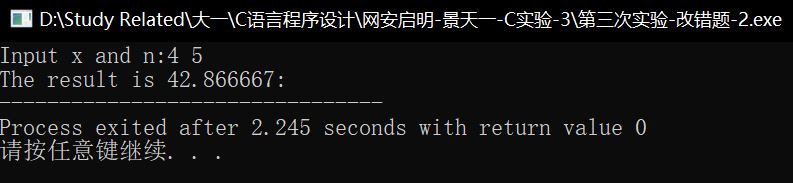


图3-4 改错与替换题2的测试用例二的运行结果

实际输出与预计输出相符，说明程序运行正确。

**3.2.3 跟踪调试题**

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，在watch窗口中观察Ik,sum,n值。具体操作如下：

（1）设输入5，观察刚执行完“scanf("%d",&k);”语句时，sum、k的值是多少？

（2）在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值是多少？返回后光条停留在哪个语句上？

（3）在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了哪里？

（4）在fabonacci函数内部单步执行，观察函数的递归执行过程。体会递归方式实现的计算过程是如何完成数计算的，并特别注意什么时刻结束递归，然后直接从第一个return语句返回到了哪里？

（5）在fabonacci函数递归执行过程中观察参数n的变化情况，并回答为什么k、sum在fabonacci函数内部不可见？

**/\*实验3-3跟踪调试题程序：**计算fabonacci数列前n项和**\*/**

#include<stdio.h>

int main(void)

{

int i,k;

long sum=0,fabonacci(int n);

printf("Inut n:");

scanf("%d",&k);

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

sum+=fabonacci(i);

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

}

return 0;

}

long fabonacci(int n)

{

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

return 1;

else

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

}

**解答：**

1. 设输入5，观察刚执行完“scanf("%d",&k);”语句时，sum、k的值是多少？

答：sum的值为0，k的值为5；

1. 在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值是多少？返回后光条停留在哪个语句上？

答：sum的值为0，k的值为0，i的值为0；返回后光条停留在printf("i=%d\tthe sum is %ld\n",i,sum);语句上

1. 在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了哪里？

答：调到了if(n==1 || n==2)

1. 在fabonacci函数内部单步执行，观察函数的递归执行过程。体会递归方式实现的计算过程是如何完成数计算的，并特别注意什么时刻结束递归，然后直接从第一个return语句返回到了哪里？

答：当n等于1或2时结束递归；从第一个return语句返回到printf("i=%d\tthe sum is %ld\n",i,sum);

1. 在fabonacci函数递归执行过程中观察参数n的变化情况，并回答为什么k、sum在fabonacci函数内部不可见？

答：sum和k是在main函数内定义的，其作用域只在main函数内，在fabonacci函数中未作外部变量引用声明，故在fabonacci函数内部不可见。

**3.2.4 程序设计题**

1. 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。要求设计一个函数对其形参n验证哥德巴赫猜想，并以“n=n1+n2”的形式输出结果。例如：n=6，输出“6=3+3”。main函数循环接收从键盘输入的整数n，如果n是大于或等于4的偶数，调用上述函数进行验证。

**解答：**

1. 解题思路：

程序设计题1的解题思路如图3-5所示：

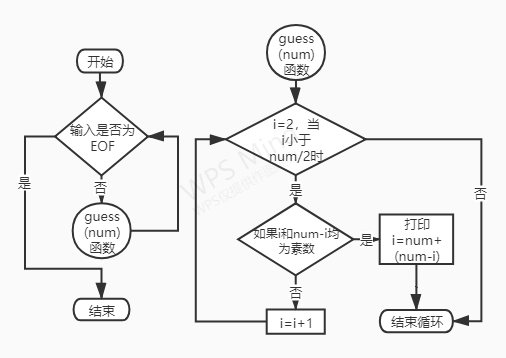


图3-5 程序设计题1的程序流程图

1. 程序清单
2. #include<stdio.h>
3. int composite(int x)
4. {
5. int i, k, flag = 0;
6. for(i=2,k=x>>1;i<=k;i++)
7. if (!(x%i) )
8. {
9. flag = 1;
10. break;
11. }
12. return flag ;
13. }
14. int guess(int num )
15. {
16. for (int i=2;i<=(num>>1);i++)
17. {
18. if((composite(i)==0)&&(composite(num-i)==0))
19. {
20. printf("%d=%d+%d\n",num,i,num-i) ;
21. break ;
22. }
23. }
24. return 0 ;
25. }
26. int main(void)
27. {
28. int num ;
29. printf("输入一个大于4的偶数：") ;
30. while(scanf("%d",&num)!=EOF)
31. {
32. guess(num) ;
33. }
34. return 0 ;
35. }
36. 测试

（a） 测试数据：

编程设计题1的测试数据如表3-2所示。

表3-2 编程题1的测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 预计输出 |
| 用例1 | 6 | 6=3+3 |
| 用例2 | 8 | 8=3+5 |
| 用例3 | ^Z | 结束 |

（b） 对应测试测试用例1，2，3的运行结果如图3-6所示。

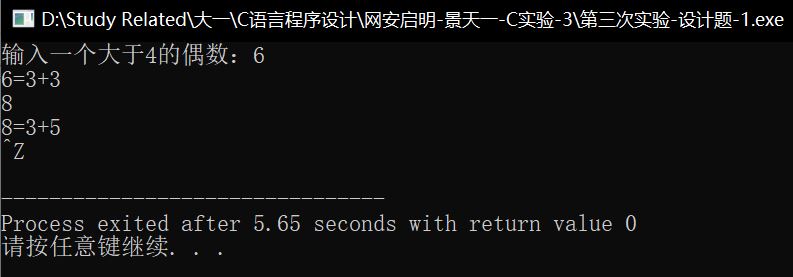


图3-6 程序设计题1的测试用例1，2，3的运行结果

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。

1. 完全数（Perfect number），又称完美数或完备数，特点是它的所有真因子（即除了自身以外的约数，包括1）之和恰好等一它本身。例如6=1+2+3，28=1+2+4+7+14等。编程寻找104以内的所有完全数。要求设计一个函数，判定形参n是否为完全数，如果是，则返回1，否则返回0。在main函数中调用该函数求104以内的所有完全数，并以完全数的真因子之和的形式输出结果，例如“6=1+2+3”；。

**解答：**

1. 解题思路：

程序设计题2的解题思路如图3-7所示：

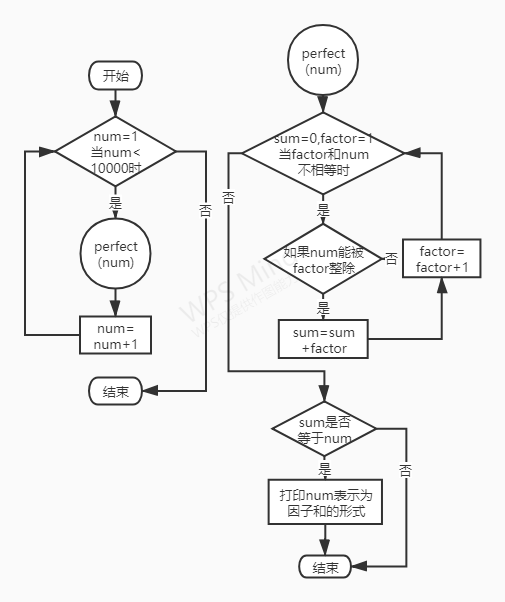


图3-7 程序设计题2的程序流程图

1. 源程序清单
2. int perfect(int num) // 定义一个输出完美数的函数
3. {
4. int sum = 0 ;
5. for (int factor=1;factor!=num;factor++)
6. {
7. if ((num%factor)==0)sum+=factor ;
8. }
9. if (sum==num)
10. {
11. printf("%d=1",num) ;
12. for (int factor=2 ;factor!=num ;factor++)
13. {
14. if ((num%factor)==0)printf("+%d",factor) ;
15. }
16. printf("\n") ;
17. }
18. return 0 ;
19. }
20. int main(void)
21. {
22. int top = 10000 ;
23. for (int num=1;num < top;num++)
24. {
25. perfect(num) ;
26. }
27. return 0 ;
28. }
29. 测试

程序设计题2运行结果如图3-8所示：

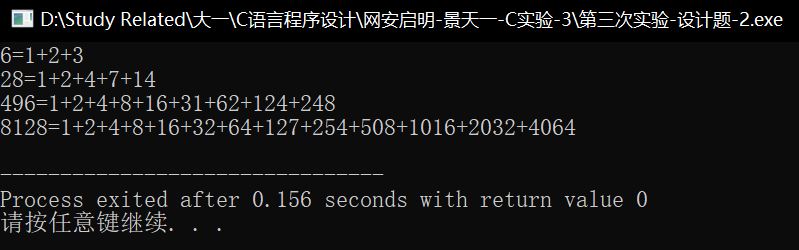


图3-8 程序设计题2的运行结果

1. 自幂数是指一个n位数，它的每个位上的数字的n次幂之和等于它本身。水仙花数是3位的自幂数，除此之外，还有4位的四叶玫瑰数、5位的五角星数、6位的六合数、7位的北斗星数、8位的八仙数等。编写一个函数，判断其参数n是否为自幂数，如果是，则返回1；否则，返回0。main函数能反复接收从键盘输入的整数k，k代表位数，然后调用上述函数求k位的自幂数，输出所有k位自幂数，并输出相应的信息，例如“3位的水仙花数共有4个153，370，371，407”。当k=0时程序结束执行。

**解答：**

1. 解题思路：

程序设计题3的解题思路如图3-9所示：

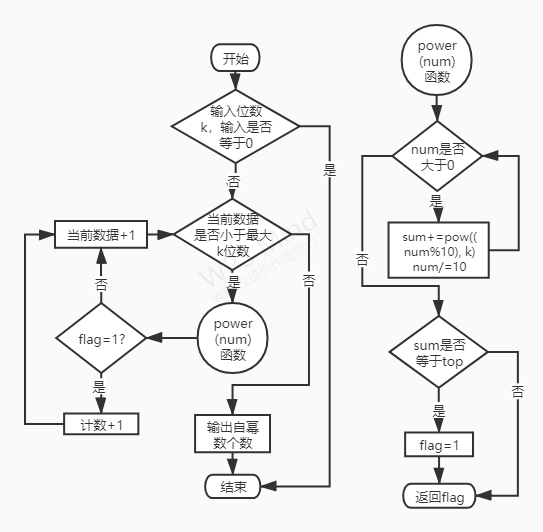


图3-9 程序设计题3的程序流程图

1. 源程序清单
2. #include<stdio.h>
3. #include<math.h>
4. int lenth(int x) // 定义了一个用于计算长度的函数
5. {
6. int n ;
7. for (n = 0; x > 0; n++) // 每次对输入数据除以10，当n等于0时终止
8. {
9. x /= 10;
10. }
11. return n ;
12. }
13. int power(int num) // 判断是否为自幂数，是输出1，否输出0
14. {
15. int flag=0, sum=0, k=0, top =0 ;
16. top = num ; // 用于存储输入数据，设定循环界限
17. k = lenth(num) ;
18. for (;num > 0 ;num /= 10) // num每次除以10，在取10的模，得到最后一位
19. {
20. sum += pow((num%10), k) ;
21. }
22. if (sum==top) // 若各位幂之和等于输入数据
23. flag = 1 ;
25. return flag ;
26. }
27. int main(void)
28. {
29. int k=1, top=0, min=0 ;
30. printf("输入所求自幂数的位数:") ;
31. for(;;)
32. {
33. scanf("%d",&k) ;
34. if (k==0) // 输入0时结束
35. break ;
36. min = pow(10,(k-1)) ; // 确定 k 位数边界
37. top = pow(10,(k)) ;
38. int count=0 ;
39. for(;min<top;min++)
40. {
41. if (power(min)==1) // 每出现一个自幂数，count加 1
42. count++ ;
43. }
44. printf("%d位的自幂数共有%d个:",k,count) ;
45. for(min = pow(10,(k-1));min<top;min++) // 再次遍历，输出所有 k 位自幂数
46. {
47. if (power(min)==1)
48. printf("%d ",min) ;
49. }
50. printf("\n") ;
51. }
52. return 0 ;
53. }
54. 测试
55. 测试数据

程序设计题3的测试用例如表3-3所示：

表3-3 编程题3的测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 预计输出 |
| 用例1 | 3 | 3位的自幂数共有4个:153 370 371 407 |
| 用例2 | 4 | 4位的自幂数共有3个:1634 8208 9474 |
| 用例3 | 0 | 结束 |

1. 对应测试测试用例1，2，3的运行结果如图3-10所示。

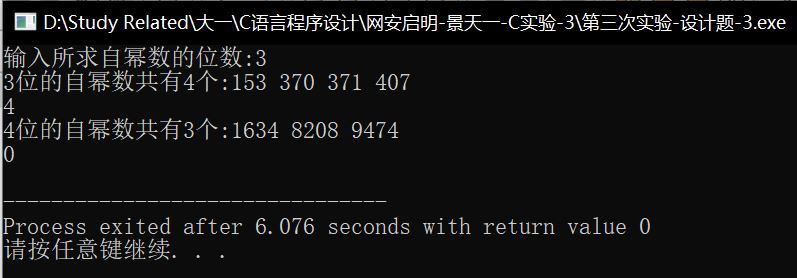


图3-8 程序设计题2的运行结果

实际输出结果与预计结果一致，说明程序运行正确。

## 3.3 实验小结

在减少计算量时，可以考虑使用static静态变量减少重复计算，节省内存空间；通过自定义函数，可以减少代码编写的时间，将不同功能模块进行封装，在进行测试时也可以更好地判断错误的所在。

本次实验里的一些题目，我感到还有修改优化的余地，在进行某个功能的实现和开发时应尽可能采用最优解法。另外，编辑器调试功能的使用可以方便进行改错，需要多加利用。