

**课 程 实 验 报 告**

**课程名称： C语言程序设计实验**

**专业班级：软件工程2304班**

**学 号：U202317310**

**姓 名：柯俊翔**

**指导教师：唐赫**

**报告日期：2023-10-23**

**软件学院**

# 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，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

### 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）错误修改：

1. 第3行未声明sum-fac函数，正确形式为：

long long sum\_fac(int n);

1. 第9行的函数返回值类型出错，正确形式为：

long long sum\_fac(int n);

1. 第11行s的数据类型出错，正确形式为：long long s = 0;
2. 第12行fac的数据类型出错且未初始化，正确形式为：

long long fac = 1;

（2）错误修改后运行结果：

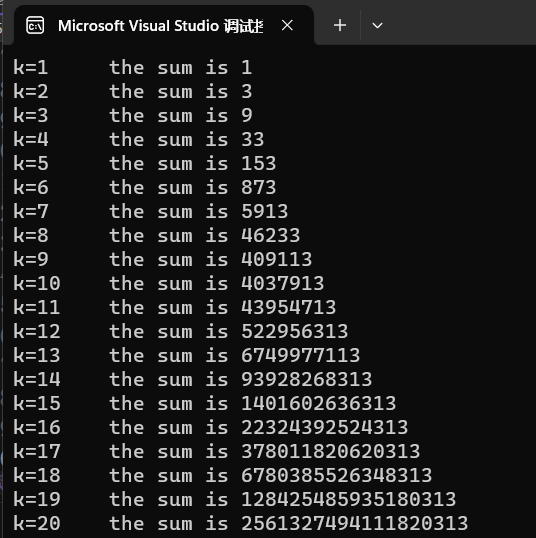


图3-1 错误修改后的程序运行结果示意图

### 2．程序修改替换题

（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：**

替换后的程序如下所示：

1. #include <stdio.h>
2. int main(void)
3. {
4. long long sum\_fac(int n);
5. int k;
6. for (k = 1; k <= 20; k++)
7. printf("k=%d\tthe sum is %lld\n", k, sum\_fac(k));
8. return 0;
9. }
10. long long sum\_fac(int n)
11. {
12. long long s = 0;
13. long long i, fac = 1;
14. for (i = 1; i <= n; i++) {
15. fac \*= i;
16. }
17. s += fac;
18. if (n == 1)
19. return s;
20. else
21. return s + sum\_fac(n - 1);
22. }

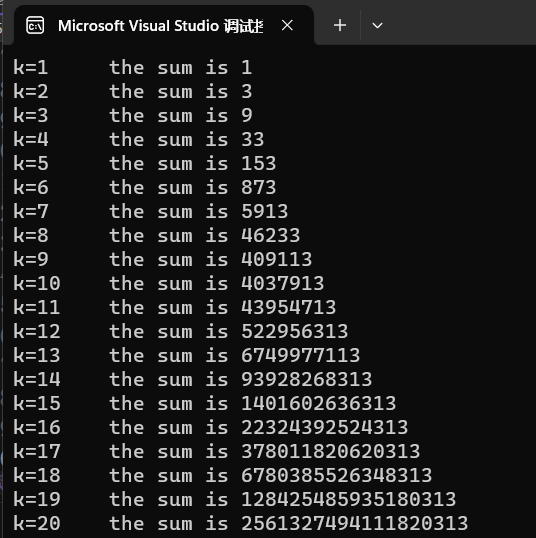


图 3-2 程序修改替换（1）后的程序运行结果示意图

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）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：**

采用静态数据类型优化，修改后的程序如下所示：

1. #include <stdio.h>
2. */\* 计算s=1+x+x2/2!+x3/3! \*/*
3. double mulx(double x)
4. {
5. static double z = 1.0;
6. z \*= x;
7. return z;
8. }
9. long fac(int n)
10. {
11. static long f = 1;
12. *//静态局部变量，静态局部变量使用static修饰符定义，*
13. *//即使在声明时未赋初值，编译器也会把它初始化为0。且静态局部变量存储于进程的*
14. *//全局数据区，即使函数返回，它的值也会保持不变。*
15. f \*= n;
16. return f;
17. }
18. double sum(double x, int n)
19. {
20. int i;
21. double z = 1.0;
22. for (i = 1; i <= n; i++)
23. {
24. z = z + mulx(x) / fac(i);
25. }
26. return z;
27. }
28. int main() {
29. double x;
30. int n;
31. printf("Input x and n:");
32. scanf("%lf%d", &x, &n);
33. printf("The result is %lf:", sum(x, n));
34. return 0;
35. }

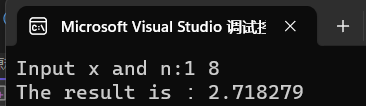


图 3-3 程序修改替换（2）后的程序运行结果示意图

根据数学拟合函数，结果应与e接近，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．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\*/

1. #include<stdio.h>
2. int main(void)
3. {
4. int i,k;
5. long sum=0,fabonacci(int n);
6. printf("Inut n:");
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,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. }

**解答：**

1. sum,k的值分别是0和5。

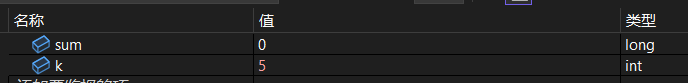


图 3-4 跟踪调试题（1）的程序运行截图

1. 如图，sum,k,i的值分别为0，5，1，光条在sum+=fabonacci(i);语句前。

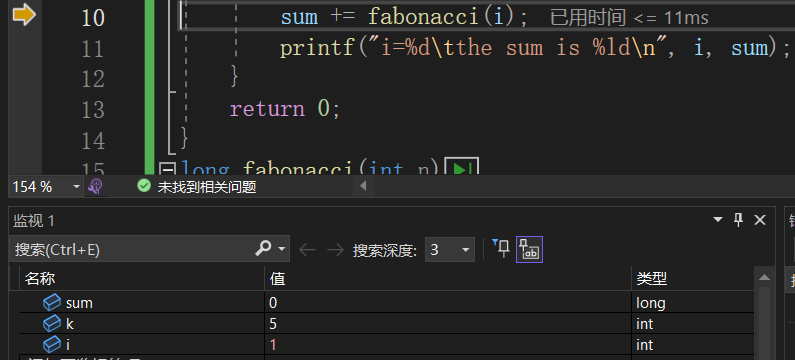


图 3-5 跟踪调试题（2）的程序运行截图

1. 光条调到了fabonacci函数体的开始,如图所示：

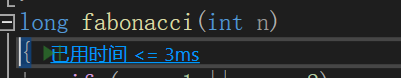


图 3-6 跟踪调试题（3）的程序运行截图

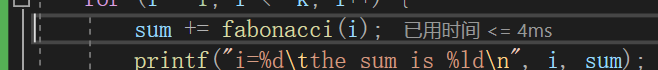
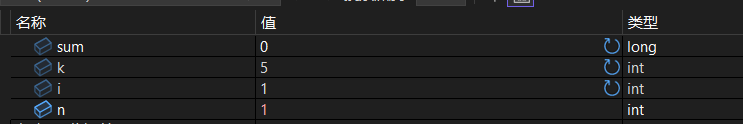
1. 观察递归可发现，当n=1或2时，函数直接返回1，当n>=3时，函数调用自身两次，分别计算fabonacci(n-1) 和 fabonacci(n-2)，若fabonacci(n-1) 和 fabonacci(n-2）中n-1或n-2>=3，函数会继续调用自身，直到n=1或2。当结束时，光条回到sum += fabonacci(i);处,如图：

图 3-7 跟踪调试题（4）的程序运行截图

1. 设输入3，n的值经历了从1-2-3-2-3-1-3的转变，符合上一问中所解释的递归调用过程。定义在函数内部的变量称为局部变量（Local Variable），它的作用域仅限于函数内部， 离开该函数后就是无效的，k,sum定义在main函数内，只在 main 函数内部可见。fabonacci 函数只能访问自己的参数 n，而不能访问 main 函数中的局部变量 k 和 sum。

图 3-8 跟踪调试题（5）的程序运行截图

### 4．程序设计

（1）编程验证歌德巴赫猜想：任何一个大于等于4的偶数都是两个素数之和。要求设计一个函数，接受形参n，以“n=n1+n2”的形式输出结果，若有多种分解情况，取n1最小的一个输出。

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） 算法流程如图3.9所示。



图3-9 编程题1的程序流程图

2）源程序清单

1. #include<stdio.h>
2. #include<math.h>
3. int isPrime(int x)*//素数*
4. {
5. int ret = 1;
6. int i;
7. if (x == 1 || (x % 2 == 0 && x != 2))
8. ret = 0;
9. for (i = 3; i < sqrt(x); i += 2)
10. {
11. if (x % i == 0)
12. {
13. ret = 0;
14. break;
15. }
16. }
17. if (ret == 1)
18. return ret;
19. }
20. void gedebahe(int n)*//哥德巴赫猜想的函数和输出*
21. {
22. for (int i = 2; i <= n / 2; i++)
23. {
24. if (isPrime(i) && isPrime(n - i))
25. {
26. printf("%d=%d+%d\n", n, i, n - i);
27. return;
28. }
29. }
30. }
31. int main()
32. {
33. int n;
34. while (scanf("%d", &n) != EOF)
35. {
36. gedebahe(n);
37. }
38. return 0;
39. }

3）测试

（a） 测试数据：

表3-1 编程题1的测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 |
| 用例1 | 48 | 48=5+43 |
| 用例2 | 90 | 90=7+83 |
| 用例3 | 10 | 10=3+7 |

（b） 对应测试数据的运行结果截图



图3-10 编程题1的测试用例的运行结果

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）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）之和恰好等于它本身。例如6=1+2+3，28=1+2+4+7+14等。

编程寻找10000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，返回1，否则返回0。在main函数中调用该函数求10000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”。程序输出中，每个完全数单独占一行。

**解答：**

1） 算法流程如图3.11所示。



图3-11 编程题2的程序流程图

2）源程序清单

1. #include <stdio.h>
2. int wanshu(int n)
3. {
4. int sum = 0;
5. for (int i = 1; i <= n / 2; i++)
6. {
7. if (n % i == 0) {  *//找到一个因子*
8. sum += i;
9. }
10. }
11. if (sum == n)    *//如果两者相等*
12. {
13. return 1;
14. }
15. else
16. {
17. return 0;
18. }
19. }
20. int main() {
21. for (int num = 1; num <= 10000; num++)
22. {
23. if (wanshu(num))
24. {
25. printf("%d=", num);
26. for (int i = 1; i <= num / 2; i++) {
27. if (num % i == 0) {
28. printf("%d", i);
29. if (i != num / 2) {
30. printf("+");
31. }
32. }
33. }
34. printf("\n");
35. }
36. }
37. return 0;
38. }

3）测试

对应测试数据的运行结果截图

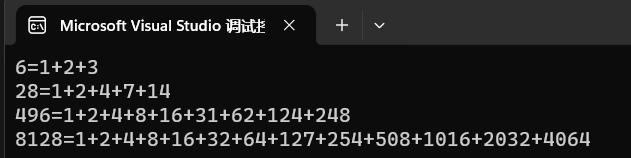


图3-12 编程题2的运行结果

（3）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，返回1；否则，返回0。要求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） 算法流程如图3.13所示。



图3-13 编程题3的程序流程图

2）源程序清单

1. #include <stdio.h>
2. #include <math.h>
3. int shuixianhua(int n) {
4. int t = n;
5. int sum = 0;
6. int d = 0;
7. while (t != 0)
8. {
9. d++;
10. t /= 10;
11. }
12. t = n;
13. while (t != 0)
14. {
15. int a = t % 10;
16. sum += pow(a, d);
17. t /= 10;
18. }
19. if (sum == n)
20. return 1;
21. else
22. return 0;
23. }
24. int main()
25. {
26. int arr[10];
27. int k;
28. while (1)
29. {
30. scanf("%d", &k);
31. if (k == 0) {
32. break;
33. }
34. int count = 0;
35. switch (k)
36. {
37. case 3:
38. printf("%d位的水仙花数共有", k);
39. break;
40. case 4:
41. printf("%d位的四叶玫瑰数共有", k);
42. break;
43. case 5:
44. printf("%d位的五角星数共有", k);
45. break;
46. case 6:
47. printf("%d位的六合数共有", k);
48. break;
49. case 7:
50. printf("%d位的北斗星数共有", k);
51. break;
52. case 8:
53. printf("%d位的八仙数共有", k);
54. break;
55. default:
56. printf("%d位的数共有", k);
57. break;
58. }
59. for (int i = pow(10, k - 1); i < pow(10, k); i++)
60. {
61. if (shuixianhua(i))
62. {
63. arr[count] = i;
64. count++;
65. }
66. }
67. printf("%d个", count);
68. for (count = count - 1; count >= 0; count--)
69. {
70. printf("%d", arr[count]);
71. if (count != 0)
72. {
73. printf(",");
74. }
75. }
76. printf("\n");
77. }
78. return 0;
79. }

3）测试

（a） 测试数据：

表3-2 编程题3的测试数据

|  |  |  |
| --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 |
| 用例1 | 3 | 3位的水仙花数共有4个407,371,370,153 |
| 用例2 | 4 | 4位的四叶玫瑰数共有3个9474,8208,1634 |
| 用例3 | 5 | 5位的五角星数共有3个93084,92727,54748 |

（b） 对应测试数据的运行结果截图

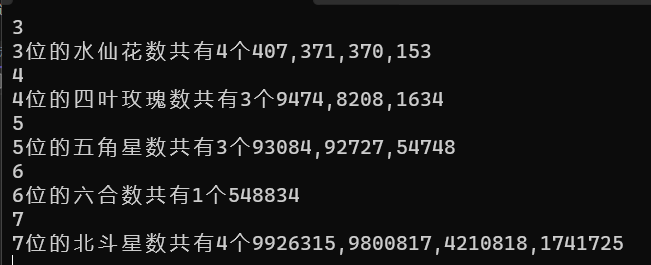


图3-14 编程题3的测试用例的运行结果

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。

## 3.3 实验小结

在这次上机实验中，我们旨在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，同时了解不了同存储类型变量的使用。此外，我们练习了使用集成开发环境的调试功能，包括单步执行、设置断点、观察变量值，并借此观察函数递归的过程，以下是我在本次实验中遇到的问题和收获。

1. **熟悉了函数的使用**：通过实验，我学习了函数的定义，声明，返回值类型和使用，在跟踪调试过程中，我了解的递归的过程，并知道了其缺点，通过调试功能观察到变量在函数调用，递归过程中的变化，对计算机有了更深的了解。
2. **纠错能力得到提升：**在改错题中，我意识到数据类型错误，却忘记了printf输出中格式符的变化，通过实验，锻炼了我思维的周密性。
3. **重视数学与编程相结合：**本次实验的程序设计题均为数学相关问题，在实验过程中，经常不知道如何确定循环的终止条件，这时需要借助数学只是判断，如完数一定是偶数，可确定for循环中i<=n/2。
4. **了解了算法的优化：**通过程序修改替换题，我学会了利用静态数据类型优化算法，减少计算量。在水仙花数题中，通过摸索我知道了可以用数组储存后面需要用到的变量，避免了重复的运算。本次实验对算法思维有一定帮助。