

**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2003班**

**学 号： U202010851**

**姓 名： 侯皓斐**

**指导教师： 唐赫**

**报告日期： 2020.11.20**

**软件工程**

**目 录**

[**3 函数与程序结构实验 1**](#_Toc404837920)

[3.1 实验目的 1](#_Toc404837921)

[3.2 实验内容 1](#_Toc404837922)

[3.3 实验小结 20](#_Toc404837923)

# 3 函数与程序结构试验

## 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 源程序改错**

**1．程序改错**

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

1 /\*实验3-1改错题程序：计算s=1!+2!+3!+…+n!\*/

2 #include <stdio.h>

3 int main(void)

4 {

5 int k;

6 for(k=1;k<=20;k++)

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

8 return 0;

9 }

10long sum\_fac(int n)

11{

12 long s=0;

13 int i,fac;

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

15 fac\*=i;

16 s+=fac;

17}

**解答：**

（1）错误修改：

1) 第2，3行间应增加使用的函数sum\_fac的生命，正确形式为：

long long sum\_fac(int );

2) 第7行的sum\_fac返回值类型为longlong，正确形式为：

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

3) 第10，12，13行，由于计算的是阶乘之和，增长速度极快，数极大应使用longlong作为函数返回值，正确形式为：

long long sum\_fac(int n)

long long s = 0;

long long i, fac = 1;

4) 第16行为计算阶乘之和而非单独一个阶乘，应进入循环中，正确形式为：

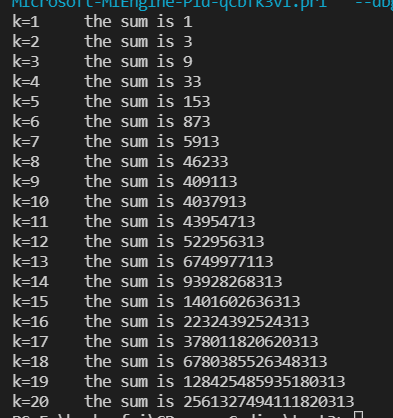
for(i=1;i<=n;i++) {

fac\*=i;

s+=fac; //挪进来

}

（2）错误修改后运行结果：



**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：**

由方程知，可写出递归方程sum\_fac(n) = sum\_fac(n-1) + n!。

而递归的边界条件为当n=0时，sum\_fac(n) = 0。

使用C语言完成递归的程序。

/\*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\*/

#include <stdio.h>

long long sum\_fac(int );

int main(void)

{

    int k;

    for(k=1;k<=20;k++)

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

    return 0;

}

long long sum\_fac(int n)

{

    if(n == 0)

        return 0LL;

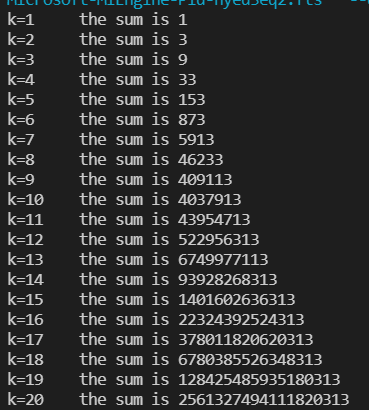
    long long i, fac = 1;

    for(i=1;i<=n;i++)

        fac\*=i;

    return sum\_fac(n-1) + fac;

}



（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\*/

#include<stdio.h>

double mulx(double x,int n);

long fac(int n);

double sum(double x,int n)

{

int i;

double z=1.0;

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

{

z=z+mulx(x,i)/fac(i);

}

return z;

}

double mulx(double x,int n)

{

int i;

double z=1.0;

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

{

z=z\*x;

}

return z;

}

long fac(int n)

{

int i;

long h=1;

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

{

h=h\*i;

}

return h;

}

int main()

{

double x;

int n;

printf("Input x and n:");

scanf("%lf%d",&x,&n);

printf("The result is %lf:",sum(x,n));

return 0;

}

**解答：**

我们观察到计算x的n次方和n!时与前项有着明显的递推关系，若已知前一个值便可O(1)的计算下一个值，即x的n+1次方，(n+1)!可被O(1)轻松计算。便在理论上达到了最小的计算量，为在函数中保存上一次的计算值，应将mulx中的z变量改为static型，fac中的h变量也改为static类型，改变计算方式即可。

/\*实验3-2程序修改替换第(2)题程序：根据公式计算 s\*/

#include<stdio.h>

double mulx(double x,int n);

long long fac(int n);

double sum(double x,int n)

{

int i;

double z=1.0;

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

{

z=z+mulx(x,i)/fac(i);

}

return z;

}

double mulx(double x,int n)

{

static double z=1.0;

z = z \* x;

return z;

}

long long fac(int n)

{

static long long h=1;

h = h \*(long long)n;

return h;

}

int main()

{

double x;

int n;

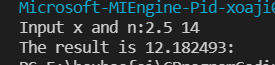
printf("Input x and n:");

scanf("%lf%d",&x,&n);

printf("The result is %lf:",sum(x,n));

return 0;

}





**3.2.3 程序跟踪调试**

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：

**/\*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)

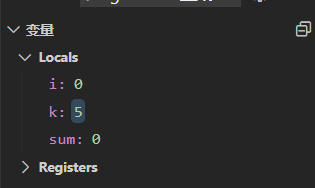
return 1;

else

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

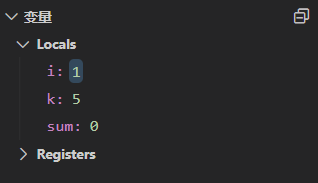
}

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

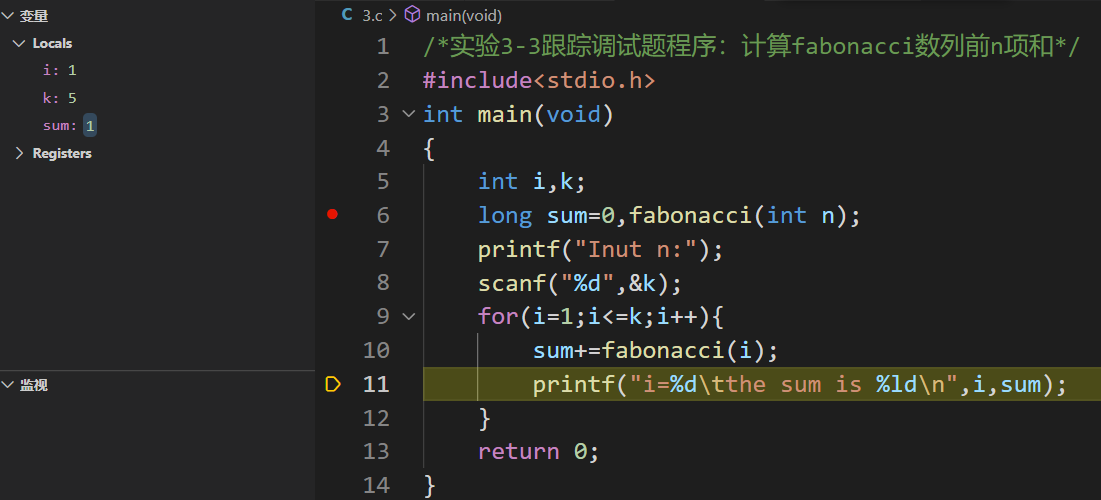


如图，刚执行完“scanf("%d",&k);”语句时，sum、k的值分别为0，5。

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？

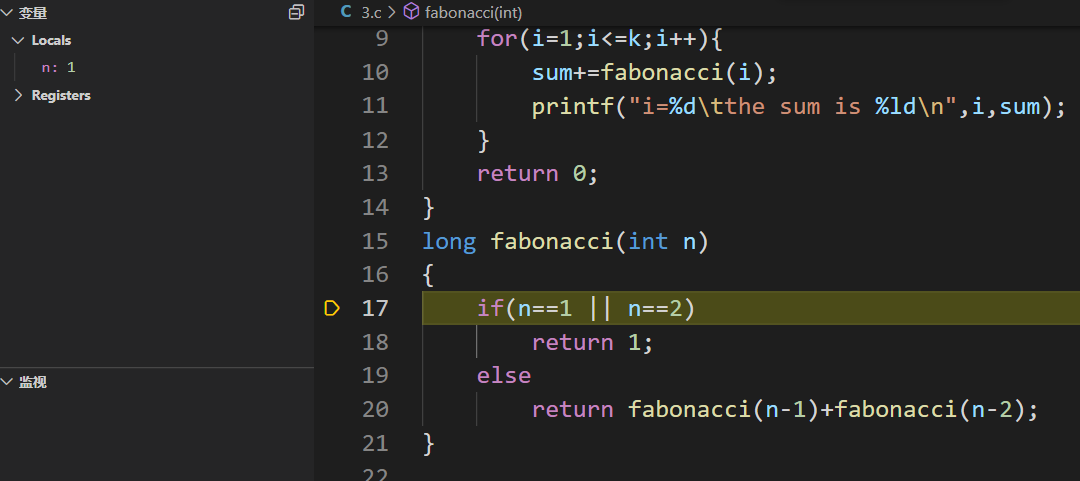


如图，在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值i，k，sum的值分别为1，5，0。



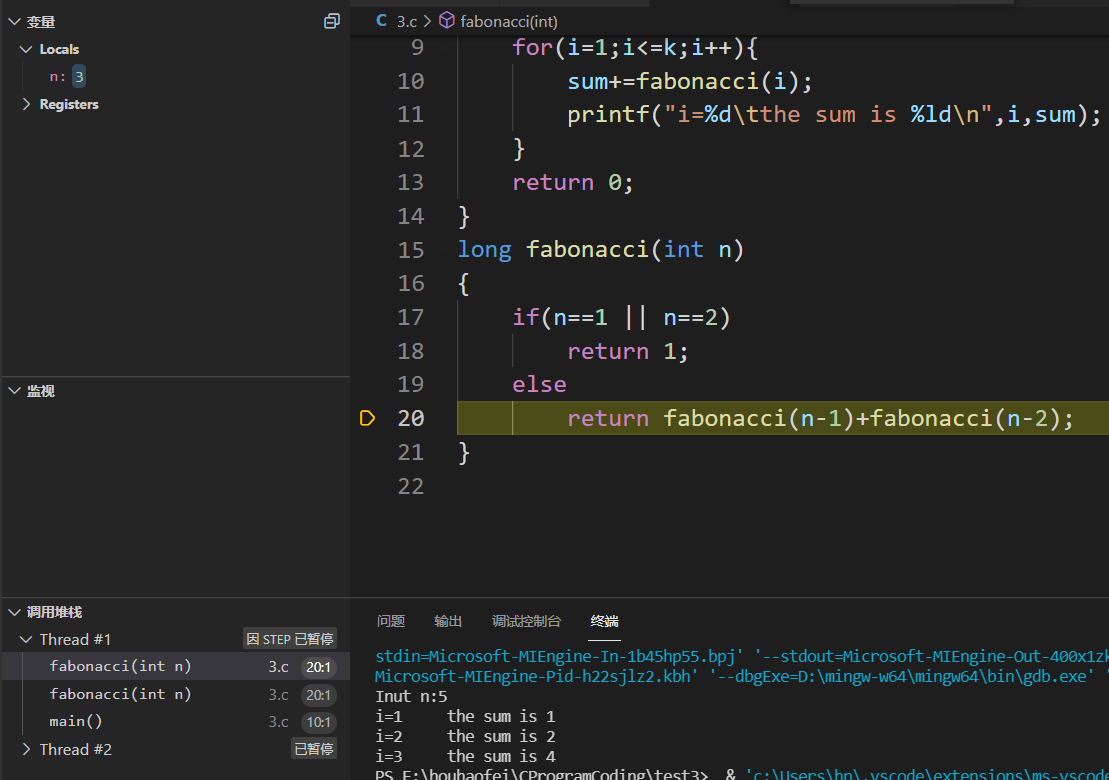
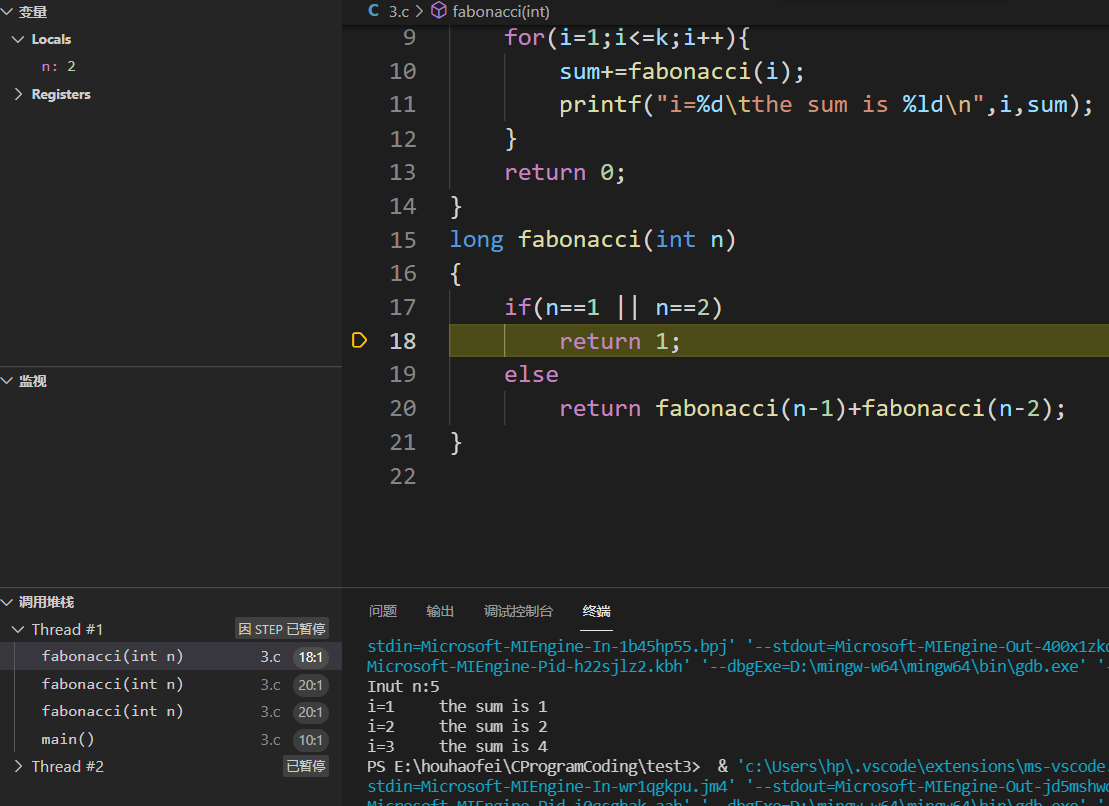
返回后光标停留在如图第11行的输出语句。

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？

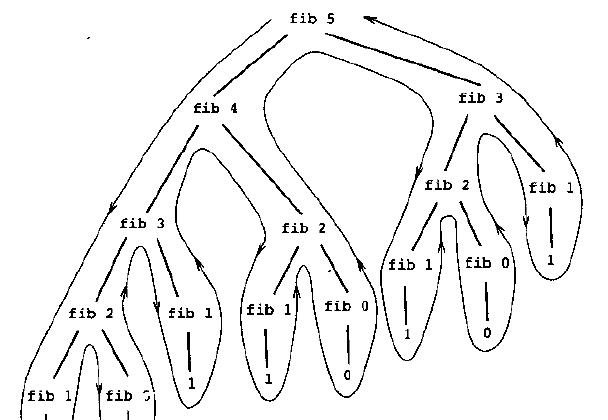


如图，在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了fabonacci函数实际执行的第一行，即第17行的if(n==1 || n==2)语句。

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？



编译器可在左下角看当前进程堆栈使用情况，结合此观察项，在多次调试观察过程中，明白递归方式实现的计算过程大概如下。



当函数接收到n时，若n为1或2，达到递归终止条件直接返回答案1。否则，将当前状态压入函数栈，递归调用fabonacci(n-1)和fabonacci(n-2)。

当两个计算完成时，当前状态从函数栈弹出，继续完成本函数，返回答案值。

注意到结束递归时，然后直接从第一个return语句返回到了计算fabonacci(n-1)和fabonacci(n-2)之和并返回的第20行return fabonacci(n-1)+fabonacci(n-2);。

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？

fabonacci函数递归执行过程中观察参数n的变化情况可见上问。

K，sum为main函数的局部变量，在另一个函数中并未声明。故在另一个函数体内不可见。

**3.2.4 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”并换行。

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，直至输入Ctrl+Z程序结束。

解答：

1）通过分析本题的算法流程，可建立如下算法流程图。

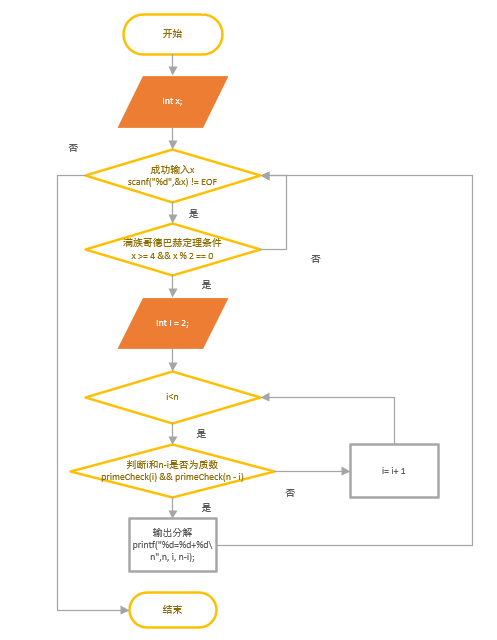


图3-1 编程题1的程序流程图

其中易分析知，其中i<n最终必然成立，因为哥德巴赫猜想虽然并未被证明，已有众多计算机科学家使用高性能计算器，对至少前2^64个数进行了检验，满足可以拆分的结果而我们为int最大的读入值仅为2^31大小。

而判断i与n-i是否为质数的函数，算法流程如实验2.2.1源程序改错实验即可，其中注意枚举的范围是2-sqrt(n)内均不为n的因数即可完成判断n为质数。

2）源程序清单

#include <stdio.h>

#include <math.h>

int primeCheck (int n) {

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

if(n % i == 0)

return 0;

return 1;

}

void mathCheck (int n) {

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

if(primeCheck(i) && primeCheck(n - i)) {

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

return ;

}

}

int main () {

int x;

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

if(x >= 4 && x % 2 == 0)

mathCheck(x);

return 0;

}

3）测试

（a） 测试数据：

600

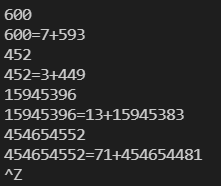
452

15945396

454654552

Ctrl + Z

（b） 对应测试数据的运行结果截图



（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. 解题思路：

通过分析本题的算法流程，可对需要设计的函数建立如下算法流程图。

函数传入i值，返回其是否为完全数。

Main函数内for循环遍历从1到10000，调用函数判断并输出。

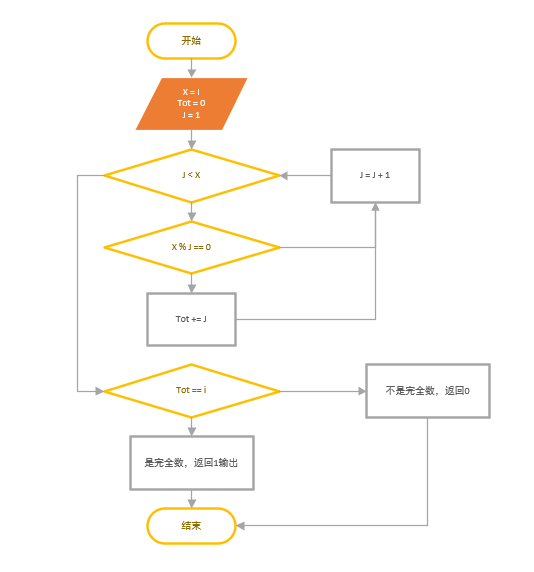


图3-2 编程题2的程序流程图

2）程序清单

#include <stdio.h>

int check (int i) {

int tot = 0;

for(int j = 1; j < i; j++)

if(i%j == 0)

tot += j;

if(tot == i)

return 1;

return 0;

}

int main () {

for(int i = 1; i < 10000 ; i++) {

if(check(i)) {

printf("%d=",i);

int temp = 0;

for(int j = 1; j < i; j++)

if(i%j == 0) {

if(temp + j == i)

printf("%d", j);

else {

temp = temp + j;

printf("%d+", j);

}

}

printf("\n");

}

}

return 0;

}

3）测试

（a） 测试数据：

本测试无输入，有固定答案，为答案输出题。

可以理论分析知1-10000内有4个完全数。

6=1+2+3

28=1+2+4+7+14

496=1+2+4+8+16+31+62+124+248

8128=1+2+4+8+16+32+64+127+254+508+1016+2032+4064

（b） 对应测试数据的运行结果截图

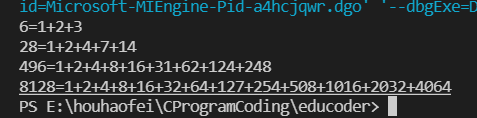


图3-3 编程题2的测试运行结果

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） 自幂数是指一个n位数，它每位数字的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所有k位的自幂数，输出相应信息并换行，例如“3位的水仙花数共有4个153,370,371,407\n”。当k=0时程序结束执行。

**解答：**

1. 解题思路：

程序main函数内构建整体运行逻辑如下：

1.输入N.直到输入0为止，跳转4.

2.调用find函数寻找k位自幂数

2.1.枚举从10^n – 10^(n+1)-1的数i。

2.2调用check(I, n)函数判断i是否为n位的自幂数，若是，跳转2.2.1

2.2.1. ans[top++] = I;即把自幂数在数组中记录

3.对find出的答案进行输出。

4.结束

实现的check(I, n)函数判断i是否为n位的自幂数的算法流程如下：

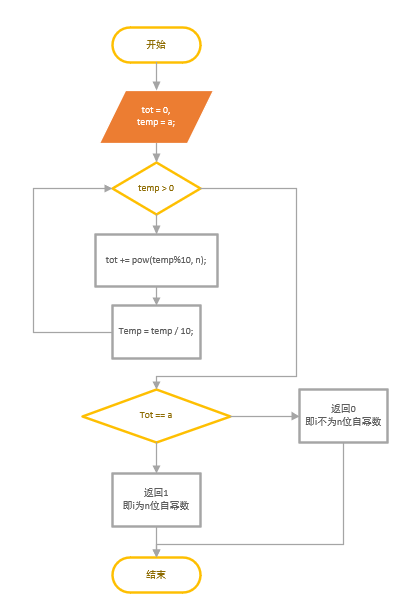


图3-4 编程题3的部分函数程序流程图

2）程序清单

#include <stdio.h>

char Name[100][100] = {"", "", "", "水仙花数", "四叶玫瑰数", "五角星数", "六合数", "北斗星数", "八仙数"};

int pow (int a,int b) {

int ans = 1;

for(int i =1; i <= b; i++)

ans = ans\*a;

return ans;

}

int check(int a,int n) {

int tot = 0, temp = a;

while(a) {

tot += pow(a%10, n);

a /= 10;

}

return tot == temp;

}

int main () {

int n = 0;

while(1) {

scanf("%d", &n);

if(n == 0)

break;

int top = 0, ans[10000], begin = pow(10, n-1), end = pow(10, n);

for(int i = begin; i < end ; i++)

if(check(i, n))

ans[top++] = i;

printf("%d位的%s共有%d个", n, Name[n], top);

for(int i = 0; i < top - 1; i++)

printf("%d,",ans[i]);

printf("%d\n",ans[top-1]);

}

return 0;

}

3）测试

（a） 测试数据：

根据题目要求以及各种可能出现的错误情况构造数据如表1-2所示。

表3-1 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N |
| 例1 | 3  4  5  0 | 3位的水仙花数共有4个153,370,371,407  4位的四叶玫瑰数共有3个1634,8208,9474  5位的五角星数共有3个54748,92727,93084 | 3位的水仙花数共有4个153,370,371,407  4位的四叶玫瑰数共有3个1634,8208,9474  5位的五角星数共有3个54748,92727,93084 |
| 例2 | 6  7  0 | 6位的六合数共有1个548834  7位的北斗星数共有4个1741725,4210818,9800817,9926315 | 6位的六合数共有1个548834  7位的北斗星数共有4个1741725,4210818,9800817,9926315 |
| 例3 | 5  8  4  7  5  0 | 5位的五角星数共有3个54748,92727,93084  8位的八仙数共有3个24678050,24678051,88593477  4位的四叶玫瑰数共有3个1634,8208,9474  7位的北斗星数共有4个1741725,4210818,9800817,9926315  5位的五角星数共有3个54748,92727,93084 | 5位的五角星数共有3个54748,92727,93084  8位的八仙数共有3个24678050,24678051,88593477  4位的四叶玫瑰数共有3个1634,8208,9474  7位的北斗星数共有4个1741725,4210818,9800817,9926315  5位的五角星数共有3个54748,92727,93084 |

（b） 对应测试测试用例1的运行结果如图3-5所示。

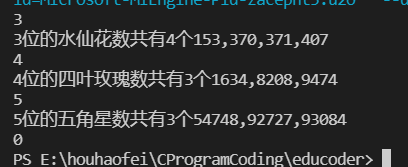


图3-5 编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图3-6所示。

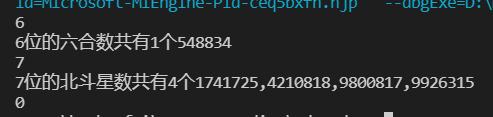


图3-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图3-7所示。

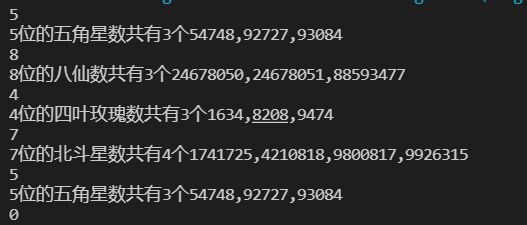


图3-7 编程题3的测试用例三的运行结果

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

主要叙述实验过程中遇到的问题，如何解决的，通过分析、结果问题后的体会。

在实验1源程序改错中，知道了具体数据类型在具体应用问题内的选取方法。

在修改题1中初步了解了递归的实现和意义，使用递归完成了阶乘之和的实现。通过分析算法结构，知道了具有递推特征的（可递推的）数学量计算复杂度可简化至O(1)，并且运用了static修饰符实现了之前计算量的存储；在逐步调试和观察调试代码时，更加深入的理解了递归的实现方式，和运算方式。

在编程题中，我也学会了合理设计函数，用以简化程序的编写。