

**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： 2020.12.29**

**软件工程**

**目 录**

[**1 表达式和标准输入与输出实验 2**](#_Toc91104231)

[1.1 实验目的 2](#_Toc91104232)

[1.2 实验内容 2](#_Toc91104233)

[1.3 实验小结 11](#_Toc91104234)

[**2 流程控制实验 12**](#_Toc91104235)

[2.1 实验目的 12](#_Toc91104236)

[2.2 实验内容 12](#_Toc91104237)

[2.3 实验小结 26](#_Toc91104238)

[**3 函数与程序结构试验 27**](#_Toc91104239)

[3.1 实验目的 27](#_Toc91104240)

[3.2 实验内容 27](#_Toc91104241)

[3.3 实验小结 45](#_Toc91104242)

[**4 编译预处理试验 46**](#_Toc91104243)

[4.1 实验目的 46](#_Toc91104244)

[4.2 实验内容 46](#_Toc91104245)

[4.3 实验小结 59](#_Toc91104246)

[**5 数组实验 60**](#_Toc91104247)

[5.1 实验目的 60](#_Toc91104248)

[5.2 实验内容 60](#_Toc91104249)

[5.3 实验小结 78](#_Toc91104250)

[**6 指针实验 79**](#_Toc91104251)

[6.1 实验目的 79](#_Toc91104252)

[6.2 实验内容 79](#_Toc91104253)

[6.3 实验小结 104](#_Toc91104254)

[**7 结构与联合实验 105**](#_Toc91104255)

[7.1 实验目的 105](#_Toc91104256)

[7.2 实验内容 105](#_Toc91104257)

[7.3 实验小结 118](#_Toc91104258)

[**8 文件操作实验 119**](#_Toc91104259)

[8.1 实验目的 119](#_Toc91104260)

[8.2 实验内容 119](#_Toc91104261)

[8.3 实验小结 126](#_Toc91104262)

[**参考文献 128**](#_Toc91104263)

# 1 表达式和标准输入与输出实验

## 1.1 实验目的

（1）熟练掌握各种运算符的运算功能，操作数的类型，运算结果的类型及运算过程中的类型转换，重点是C语言特有的运算符，例如位运算符，问号运算符，逗号运算符等；熟记运算符的优先级和结合性。

（2）掌握getchar, putchar, scanf 和printf 函数的用法。

（3）掌握简单C程序（顺序结构程序）的编写方法。

## 1.2 实验内容

**1.2.1 源程序改错**

下面给出了一个简单C语言程序例程，用来完成以下工作：

（1）输入华氏温度f，将它转换成摄氏温度c后输出；

（2）输入圆的半径值ｒ，计算并输出圆的面积ｓ；

（3）输入短整数ｋ、ｐ，将ｋ的高字节作为结果的低字节，ｐ的高字节作为结果的高字节，拼成一个新的整数，然后输出；

在这个例子程序中存在若干语法和逻辑错误。要求参照1.3和1.4的步骤对下面程序进行调试修改，使之能够正确完成指定任务。

1 #include<stdio.h>

2 #define PI 3.14159;

3 voidmain( void )

4 {

5 int f ;

6 short p, k ;

7 double c , r , s ;

8 /\* for task 1 \*/

9 printf(“Input Fahrenheit:” ) ;

10 scanf(“%d”, f ) ;

11 c = 5/9\*(f-32) ;

12 printf( “ \n %d (F) = %.2f (C)\n\n ”, f, c ) ;

13 /\* for task 2 \*/

14 printf("input the radius r:");

15 scanf("%f", &r);

16 s = PI \* r \* r;

17 printf("\nThe acreage is %.2f\n\n",&s);

18 /\* for task 3 \*/

19 printf("input hex int k, p :");

20 scanf("%x %x", &k, &p );

21 newint = (p&0xff00)|(k&0xff00)<<8;

22 printf("new int = %x\n\n",newint);

}

**解答：**

（1）错误修改：

1) 第2行的符号常量定义后不能有分号，正确形式为：

#define PI 3.14159

2) 第6行的p后面赋值正好使short符号位为1，为保证右移补0，正确形式为：

unsigned short p, k ;

3) 第10行的使用scanf读入需要对变量取地址，正确形式为：

scanf("%d", &f ) ;

4) 第11行的5/9默认为int的整除，结果为0错误，正确形式为：

c = 5.0 / 9.0 \* (f-32);

5) 第15行的使用scanf读入double型变量要使用%lf，正确形式为：

scanf("%lf", &r);

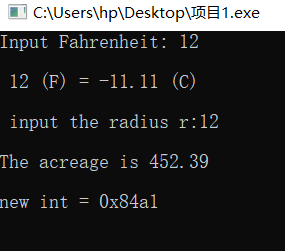
6) 第17行的输出不需要取地址，正确形式为：

printf("\nThe acreage is %.2f\n\n", s);

7) 第20行的newint未被定义，正确形式为：

unsigned short newint = (p&0xff00)|(k>>8);

（2）错误修改后运行结果：



**1.2.2 源程序修改替换**

下面的程序利用常用的中间变量法实现两数交换，请改用不使用第3个变量的方法实现。该程序中t是中间变量，要求将定义语句中的t删除，修改下划线处的语句，使之实现两数对调的操作。

#include<stdio.h>

void main( )

{

int a, b, t;

printf(“Input two integers:”);

scanf(“%d %d”,&a,&b);

t=a ；a=b；b=t；

prinf(“\na=%d,b=%d”,a,b);

}

**解答：**

通过数值的运算可以实现两个数没有第三个变量的交换：a=b-a，求出a与b之间的距离存放在a中；b=b-a，求出原来的a存入b中；a=a+b，从而借助原来的a和a，b间距离求出b存入a中。替换后的程序如下所示：

#include<stdio.h>

void main( )

{

int a, b;

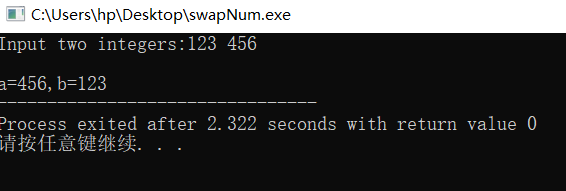
printf(“Input two integers:”);

scanf(“%d %d”,&a,&b);

a=b-a ；b=b-a；a=a+b；

prinf(“\na=%d,b=%d”,a,b);

}



**1.2.3 程序设计**

**（1）**输入字符ｃ，如果ｃ是大写字母，则将ｃ转换成对应的小写，否则ｃ的值不变，输入Ctrl+Z程序结束。要求①用条件表达式；②字符的输入输出用getchar和putchar函数。程序应能循环接受用户的输入，直至输入Ctrl+Z程序结束。

**解答：**

1） 算法流程如图1.1所示。

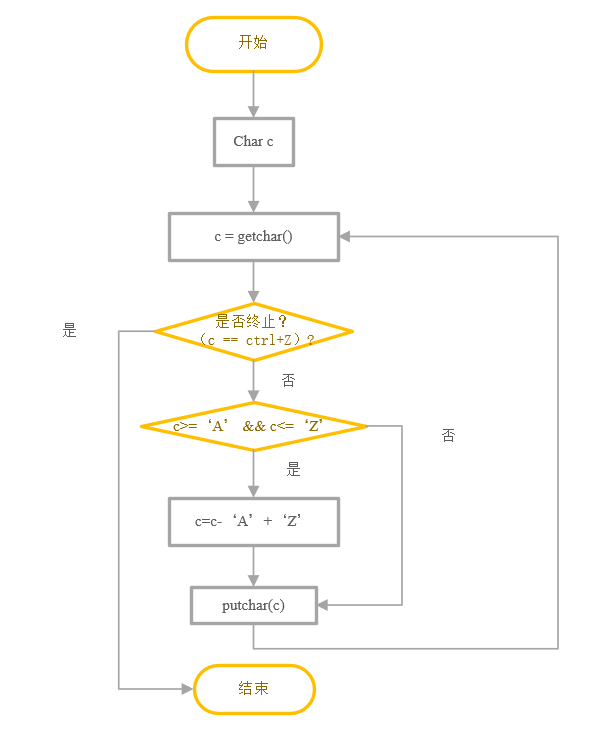


图1-1 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

int main () {

char c;

while((c = getchar()) != EOF) {

if( c >= 'A' && c <= 'Z' )

c = c - 'A' + 'a';

putchar(c);

}

return 0;

}

3）测试

（a） 测试数据：

C

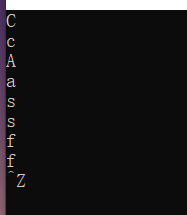
A

s

f

Ctrl+Z

（b） 对应测试数据的运行结果截图



（2）编写一个程序，输入无符号短整数x，ｍ，ｎ（0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ）,取出x从第ｍ位开始向左的ｎ位（ｍ从右至左编号为0～15），并使其向左端（第15位）靠齐。

**解答：**

1) 解题思路：

1.输入x，m，n，为了方便分析测试结果，x的输入采用16进制

2.如果0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ，转2.1，否则转3.

2.1 首先x>>m，将要处理的n位移动到最右；

2.2 再将上一步的结果左移16 - n位，即： 答案为(x >> m) << (16 - n)

2.3 用16进制输出结果并转4.

3. 显示输入错误信息；

4. 结束

2）程序清单

#include<stdio.h>

void main() {

unsigned short x,m,n;

printf("输入x（16进制）、m（0~15）和n（1~16-m）：\n ");

scanf("%hx%hd%hd",&x,&m,&n);

if ( 0 <= m && m <= 15 && 1 <= n && n <= 16-m ) /\*判断m、n的值是否在合理范围内\*/

printf("ans=%hx\n",(x >> m) << (16 - n));

else printf("输入错误!\n");

}

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1-1所示。

表1-1 编程题3的测试数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | | 理 论 结 果 | 运 行 结 果 |
| X | m | N |
| 用例1 | 0100 0110 1000 0000（4680） | 7 | 4 | 计算结果1101 0000 0000 0000 即D000 | D000 或 截图 |
| 用例2 | 1101 0101 1000 0011（D583） | 16 | 1 | 输入错误（m值超范围） | 输入错误! |
| 用例3 | 1101 0101 1000 0011（D583） | 13 | 5 | 输入错误（n值超范围） | 输入错误! |

（b） 对应测试测试用例1的运行结果如图1-2所示。

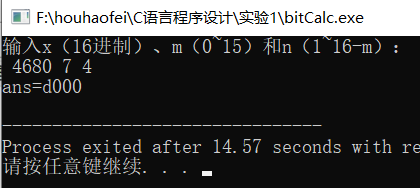


图1-2 编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图1-3所示。

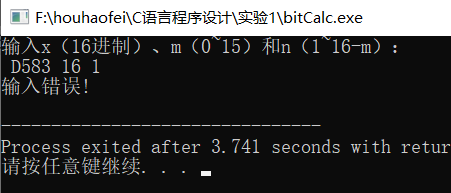


图1-3 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图1-4所示。

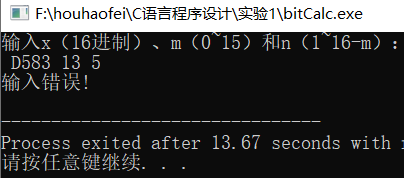


图1-4 编程题3的测试用例三的运行结果

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）IP地址通常是4个用句点分隔的小整数（即点分十进制），但这些地址在机器中是用一个无符号长整型数表示的。例如3232235876，其机内二进制表示就是11000000 10101000 00000001 01100100，按照8位一组用点分开，该IP地址就写成192.168.1.100。

读入无符号长整型数表示的互联网IP地址，对其译码，以常见的点分十进制形式输出。要求循环输入和输出，直至输入Ctrl+Z结束。

**解答：**

1) 解题思路：

1.输入x，如果未输入x，得到 Ctrl+Z，跳转至4结束。

2.定义opt = (1 << 8) – 1,ans[4], i= 1

3.如果i<4执行以下步骤

3.1 首先取出x前八位，ans[i] = x & opt;

3.2 然后x>>8，将要处理的8位移动到最右；

3.3 i = i +1

3. 倒序加结构输出ans数组

4. 结束

2）程序清单

#include <stdio.h>

int main () {

unsigned long long x = 0;

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

unsigned long long opt = (1 << 8) - 1;

unsigned long long ans[4];

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

ans[i] = x & opt;

x >>= 8;

}

for(int i = 3; i >= 1; i--)

printf("%lld.",ans[i]);

printf("%lld\n",ans[0]);

}

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1-2 编程题4的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| X |
| 用例1 | 3232235876 | 192.168.1.100 | 192.168.1.100 |
| 用例2 | 66322345376 | 113.30.173.160 | 113.30.173.160 |
| 用例3 | 3232235876  66322345376  187232376 | 192.168.1.100  113.30.173.160  11.40.240.120 | 192.168.1.100  113.30.173.160  11.40.240.120 |

（b） 对应测试测试用例1的运行结果如图1-5所示。

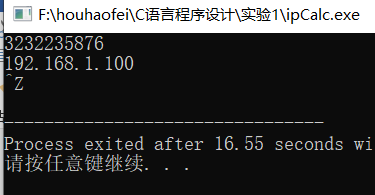


图1-5 编程题4的测试用例一的运行结果

对应测试测试用例2的运行结果如图1-6所示。



图1-6 编程题4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1-7所示。

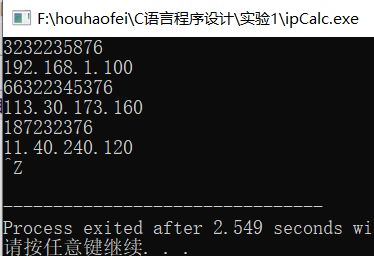


图1-7 编程题4的测试用例三的运行结果

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。

## 1.3 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，发现在devC++环境下无法执行调试环节，在询问老师后了解了必须在debug环境下且创建项目的情况下才能进行调试。过程中有几个bug是比较难以想到的，不断分析输出结果，调试看 k >> 8 的值，突然回顾到了右移运算的补位性质，最终解决了所有改错。

在编程题1中不知道如何不使用第三个变量交换两个数，和同学讨论后，商量出了一种可行的方案。a=b-a，求出a与b之间的距离存放在a中；b=b-a，求出原来的a存入b中；a=a+b，从而借助原来的a和a，b间距离求出b存入a中。

在编程题2中，如何判断程序结束的“ctrl + Z”是个大问题。通过百度搜索相关资料了解了循环应写的形式。在画流程图的过程中，通过百度一些基本功能学习了Visio软件。

# 2 流程控制实验

## 2.1 实验目的

（1）掌握复合语句、if语句、switch语句的使用，熟练掌握for、while、do-while三种基本的循环控制语句的使用，掌握重复循环技术，了解转移语句与标号语句。

（2）练习循环结构for、while、do-while语句的使用。

（3）练习转移语句和标号语句的使用。

（4）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。

## 2.2 实验内容

**2.2.1 源程序改错**

**1．程序改错**

下面的实验2-1程序是合数判断器（合数指自然数中除了能被1和本身整除外，还能被其它数整除的数），在该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，使之能够正确完成指定任务。

/\* 实验2-1改错题程序：合数判断器\*/

#include <stdio.h>

int main( )

{

int i, x, k, flag = 0;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

for(i=2,k=x>>1;i<=k;i++)

if (!x%i) {

flag = 1;

break;

}

if(flag=1) printf("%d是合数", x);

else printf("%d不是合数", x);

}

return 0;

}

**解答：**

（1）错误修改：

1) 第7行的while循环内flag应该每次置为0，正确形式为：

flag = 0;

2) 第9行的k = x >> 1运算数量明显偏多，正确形式为：

for (i = 2, k = (int)sqrt(x); i <= k; i++)

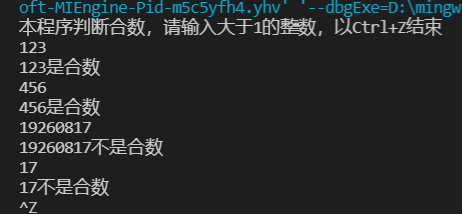
3) 第10行的!运算优先级高，产生错误运算结果，正确形式为：

if (!(x % i))

4) 第14行的判断语句应使用==，正确形式为：

if (flag == 1)

（2）错误修改后运行结果：



**2.2.2 源程序修改替换**

（1）修改实验2-1程序，将内层两出口的for循环结构改用单出口结构，即不允许使用break、goto等非结构化语句。

**解答：**

For循环中的break非结构语句破坏了单出口结构，将flag的判断放入for循环的条件控制语句，这样中途如果出现一个模数使其取模为0，即其为合数，便退出循环。

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：改单出口 \*/

#include <stdio.h>

int main() {

int i, x, k, flag = 0;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

flag = 0;//需添加，否则出现一次合数，之后都是合数

for (i = 2, k = (int)sqrt(x); (!flag) && i <= k; i++)

if (!(x % i))

flag = 1;

if (flag == 1)//if (flag = 1)

printf("%d是合数\n", x);//printf("%d是合数", x);

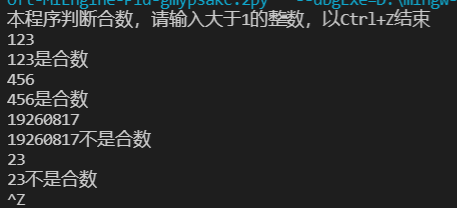
else

printf("%d不是合数\n", x);//printf("%d不是合数", x);

}

return 0;

}



（2）修改实验2-1程序，将for循环改用do-while循环。

**解答：**

将for循环的判断语句放入while的判断语句，若如是，可将flag取消。因为当我们i一直自加到sqrt(x)后，仍未出现一个模数与x取模为0，即可判定其为质数，若i<floor(sqrt(x))+1便可已知其中途退除了循环，发现了其某个因数，便其为合数。

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：改用dowhile循环 \*/

#include <stdio.h>

int main() {

int i, x, k;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

i = 1; k = (int)sqrt(x);

do {

i++;

}while((x % i) && i <= k);

if (i != k+1)//if (flag = 1)

printf("%d是合数\n", x);//printf("%d是合数", x);

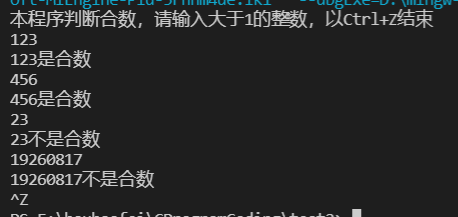
else

printf("%d不是合数\n", x);//printf("%d不是合数", x);

}

return 0;

}



（3）修改实验2-1程序，将其改为纯粹合数求解器，求出所有的3位纯粹合数。一个合数去掉最低位，剩下的数仍是合数；再去掉剩下的数的最低位，余留下来的数还是合数，这样反复，一直到最后剩下一位数仍是合数，这样的数称为纯粹合数。

**解答：**

我们依次枚举X所有的三位数，即从100到999，每个数判断其是否为纯粹合数。我们设立一个表示其为纯粹合数的标志变量pureComFlag，一开始设为1。定义临时变量x=X。然后利用上述代码判断x是否为合数，若为合数则pureComFlag = 0，退出循环。否则x自除10，直到x等于0为止，这样我们就判断了X是否是纯粹合数。

算法流程图如下：

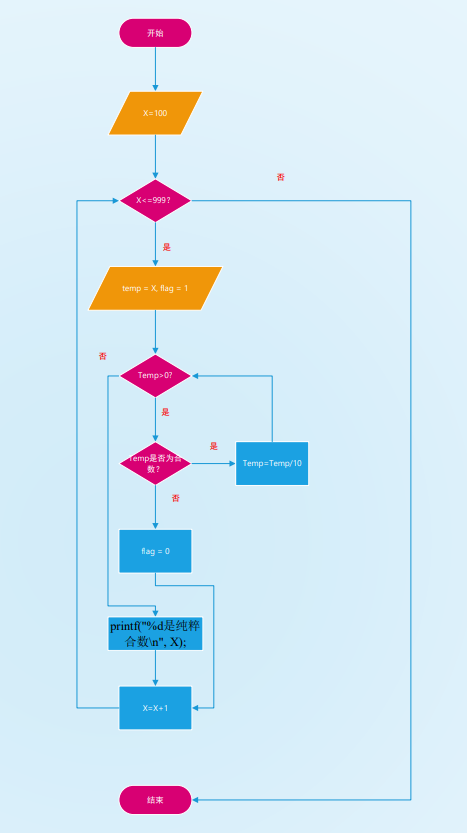


图2-1 源程序修改替换3的程序流程图

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：纯粹合数求解器 \*/

#include <stdio.h>

#include <math.h>

int main() {

int i, x, k, flag = 0;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

for(int X = 100; X <= 999; X++) {

x = X;

int pureComFlag = 1;

while(x) {

flag = 0;

for (i = 2, k = (int)sqrt(x); i <= k; i++)

if (!(x % i)) {

flag = 1;

break;

}

if(!flag) {

pureComFlag = 0;

break;

}

x = x/10;

}

if (pureComFlag == 1)

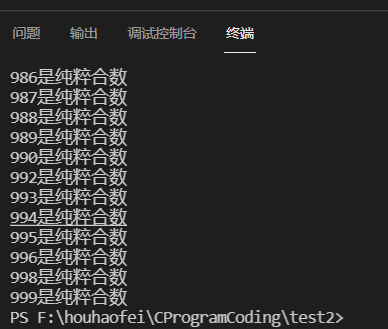
printf("%d是纯粹合数\n", X);

}

return 0;

}

部分程序输出展示如下：



**2.2.3 程序设计**

（1） 假设工资税金按以下方法计算：x ＜ 1000元，不收取税金；1000 ≤ x ＜ 2000，收取5%的税金；2000 ≤ x ＜ 3000，收取10%的税金；3000 ≤ x ＜ 4000，收取15%的税金；4000 ≤ x ＜ 5000，收取20%的税金；5000 ≤ x，收取25%的税金。（注意税金的计算按照阶梯计税法，比如，工资为4500，那么税金=1000\*5% + 1000\*10% + 1000\*15% + 501\*20%）。编写一个程序,输入工资金额，输出应收取税金额度，要求分别用if语句和switch语句来实现。

解答：

1） 本题，可使用累加的计算方法从而避免手动计算大量的公式，简单而直接。

若使用if语句：我们便注意到：

x>=5000的部分(x-4999)应该征收25%的税金，为保证阶梯计税，x=4999

x>=4000的部分(x-3999)应该征收20%的税金，为保证阶梯计税，x=3999

x>=3000的部分(x-2999)应该征收15%的税金，为保证阶梯计税，x=2999

x>=2000的部分(x-1999)应该征收10%的税金，为保证阶梯计税，x=1999

x>=1000的部分(x-999)应该征收5%的税金，为保证阶梯计税，x=999

若使用switch语句：我们注意到case必须是整数值，则可利用int(x/1000)作为参量值，使用deflaut语句避免x>=6000，的情况，我们仍然要使用累加的方法，所以我们从大到小排列case，并不break，令其一路运算到底。

2）源程序清单

/\*salaryCalc\_ifVersion\*/

#include <stdio.h>

int main () {

int x = 0; double tax = 0;

scanf("%d", &x);

if(x >= 5000) {

tax += 0.25\*(x - 4999);

x = 4999;

}

if(x >= 4000) {

tax += 0.2\*(x - 3999);

x = 3999;

}

if(x >= 3000) {

tax += 0.15\*(x - 2999);

x = 2999;

}

if(x >= 2000) {

tax += 0.1\*(x - 1999);

x = 1999;

}

if(x >= 1000) {

tax += 0.05\*(x - 999);

x = 999;

}

printf("tax = %lf\n", tax);

return 0;

}

/\* salaryCalc\_switchVersion \*/

#include <stdio.h>

int main () {

int x = 0; double tax = 0;

scanf("%d", &x);

switch (x/1000) {

default:

case 5:tax += 0.25\*(x - 4999);x = 4999;

case 4:tax += 0.2\*(x - 3999); x = 3999;

case 3:tax += 0.15\*(x - 2999);x = 2999;

case 2:tax += 0.1\*(x - 1999); x = 1999;

case 1:tax += 0.05\*(x - 999); x = 999;

case 0: break;

}

printf("tax = %lf\n", tax);

return 0;

}

3）测试

（a） 测试数据：

600

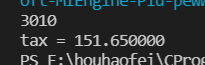
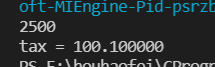
2500

3010

6000

（b） 对应测试数据的运行结果截图

/\*salaryCalc\_ifVersion\*/



/\* salaryCalc\_switchVersion \*/



（2）输入一段以!结尾的短文(最多5行,每行不超过50个字符)，要求将它复制到输出，复制过程中将每行一个以上的空格字符用一个空格代替。

**解答：**

1)解题思路：

1.定义一个字符数组s用以存储读入的字符，数字n作为短文长度，flag表示前一个是否是空格。

2.如果n==0或者s[n-1]!=’!’，执行2.1，否则跳转3

2.1 读入一个字符ch

2.2 如果字符为空格，执行2.2.1，否则执行2.3

2.2.1若flag=1,跳转至2，开始新一轮工作流程。否则跳转2.2.2

2.2.2 s[n++]=ch，flag=1；

2.3 s[n++]=ch，flag=0；

3. 显示输出s；

4. 结束

2）程序清单

#include <stdio.h>

int main () {

char s[1000];int top=0;int flag = 0;

while(s[top - 1]!='!' || top == 0) {

char now = getchar();

if(now == ' ') {

if(flag)

continue;

else {

flag = 1;

s[top++] = now;

}

}

else {

s[top++] = now;

if(flag)

flag = 0;

}

}

s[top] = 0;

printf("%s",s);

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1-1所示。

表2-1 编程题2的测试数据（以’#’表示空格符号）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入文本 |
| 用例1 | I#love###you  Just##like#fish####like  ####water  ! | I#love#you  Just#like#fish#like  #water  ! | I#love#you  Just#like#fish#like  #water  ! |
| 用例2 | Mary#had#a#little#lamb! | Mary#had#a#little#lamb! | Mary#had#a#little#lamb! |
| 用例3 | Mary#had####a#little#lamb.  ####It#is#a#good#pet#for#Mary###! | Mary#had#a#little#lamb.  #It#is#a#good#pet#for#Mary#! | Mary#had#a#little#lamb.  #It#is#a#good#pet#for#Mary#! |

（b） 对应测试测试用例1的运行结果如图2-2所示。

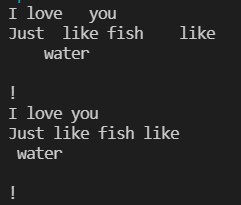


图2-2 编程题2的测试用例一的运行结果

对应测试测试用例2的运行结果如图2-3所示。



图2-3 编程题2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2-4所示。

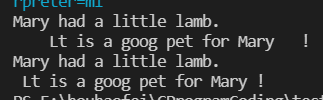


图2-4 编程题2的测试用例三的运行结果

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） 打印如下的杨辉三角形。

1 /\*第0行 \*/

1 1 /\*第1行 \*/

1 2 1 /\*第2行 \*/

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

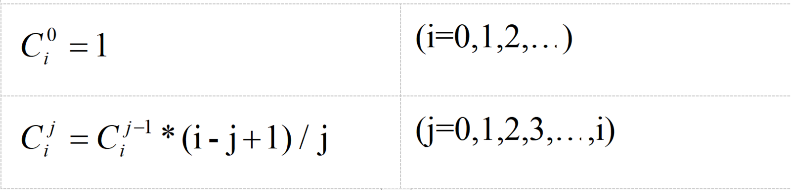
1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

第i行第j列位置的数据值可以由组合C表示，C的计算如下：



根据以上公式，采用顺推法编程，输入最后一行的编号N（0<=N<=6），要求输出金字塔效果的杨辉三角形。

特别要注意空格的数目，每个数字占位4格，例如，1位数后补3个空格。第N行行首是N个空格（每向上一行，行首空格数对应增加）。每行末尾是换行符。

**解答：**

1) 解题思路：

1.输入N.

2.枚举输出的第i行，从0到N:

2.1 输出3N-2i个空格

2.2 定义Cnow =1，并按照加上本身4个格定位输出

2.3 枚举j从1到i

2.3.1 Cnow = Cnow \* (i-j+1) / j;

2.3.2 按照加上本身4个格定位输出

3.结束

2）程序清单

#include <stdio.h>

int main () {

int N;

scanf("%d", &N);

for(int i = 0; i<= N; i++) {

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

printf(" ");

for(int j = 1; j <= 3 \* (N - i); j++)

printf(" ");

int Cnow = 1;

printf("%d ",Cnow);

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

Cnow = Cnow \* (i-j+1) / j;

printf("%d",Cnow);

if(Cnow < 10) printf(" ");

else if(Cnow < 100) printf(" ");

else printf(" ");

}

printf("\n");

}

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2-2 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 | 0 | 1 | 1 |
| 例2 | 3 | 1  1 1  1 2 1  1 3 3 1 | 1  1 1  1 2 1  1 3 3 1 |
| 例3 | 6 | 1  1 1  1 2 1  1 3 3 1  1 4 6 4 1  1 5 10 10 5 1  1 6 15 20 15 6 1 | 1  1 1  1 2 1  1 3 3 1  1 4 6 4 1  1 5 10 10 5 1  1 6 15 20 15 6 1 |

（b） 对应测试测试用例1的运行结果如图2-5所示。



图2-5 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2-6所示。

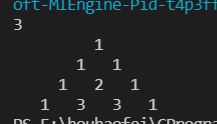


图2-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图2-7所示。

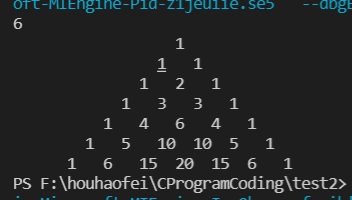


图2-7 编程题3的测试用例三的运行结果

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。

（4） 625这个数很特别，625的平方等于390625，其末3位也是625。请编程输出所有这样的3位数：它的平方的末3位是这个数本身。要求这些数字从小到大排列，每个数字单独占一行。

**解答：**

1. 解题思路：算法流程可用算法流程图表示

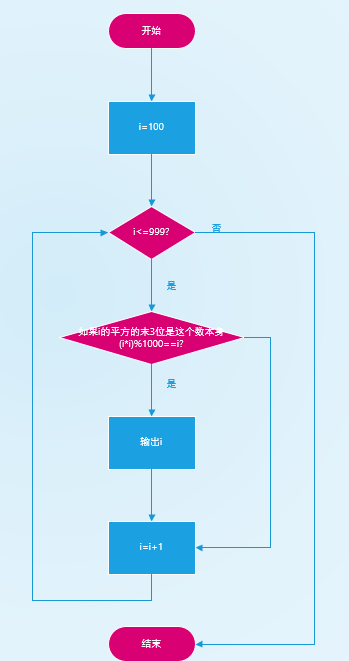


图2-8 编程题4的程序流程图

2）程序清单

#include <stdio.h>

int main () {

for(int i = 100; i < 1000; i++)

if( (i\*i)%1000 == i )

printf("%d\n",i);

return 0;

}

3）测试

（a） 测试数据：

由数学推导已知，这类数的个数是有限的，可演算得仅2个三位数满足条件，即376和625。

（b） 对应测试的运行结果如图2-9所示。



图2-9 编程题4的测试的运行结果

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。

## 2.3 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，采用逐步调试的方法成功找出了语法上的错误，在分析算法复杂度的时候确定了其运算次数过高的第4个问题。最终解决了所有改错。

在修改题1中不知道如何修改实验2-1程序，将内层两出口的for循环结构改用单出口结构，即不允许使用break、goto等非结构化语句。和同学交流过后，明白了单出口的含义，For循环中的break非结构语句破坏了单出口结构，将flag的判断放入for循环的条件控制语句，这样中途如果出现一个模数使其取模为0，即其为合数，便退出循环。既没有破坏单出口的性质，又保证了运算的效率。

在编程题3中，学会了仔细阅读题面然后逐渐分析得出空格的个数解决了问题，明白了学会审题的重要意义。

# 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：

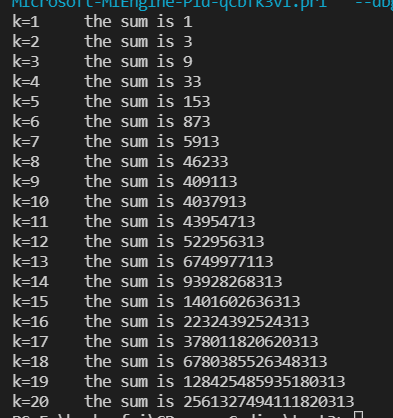
for(i=1;i<=n;i++) {

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;

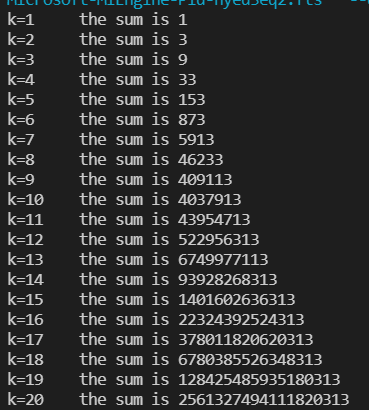
    long long i, fac = 1;

    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;

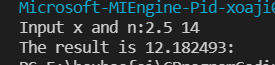
printf("Input x and n:");

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)

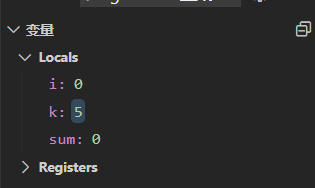
return 1;

else

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

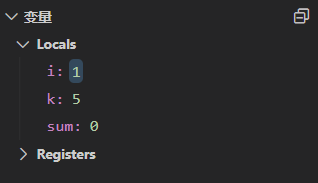
}

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

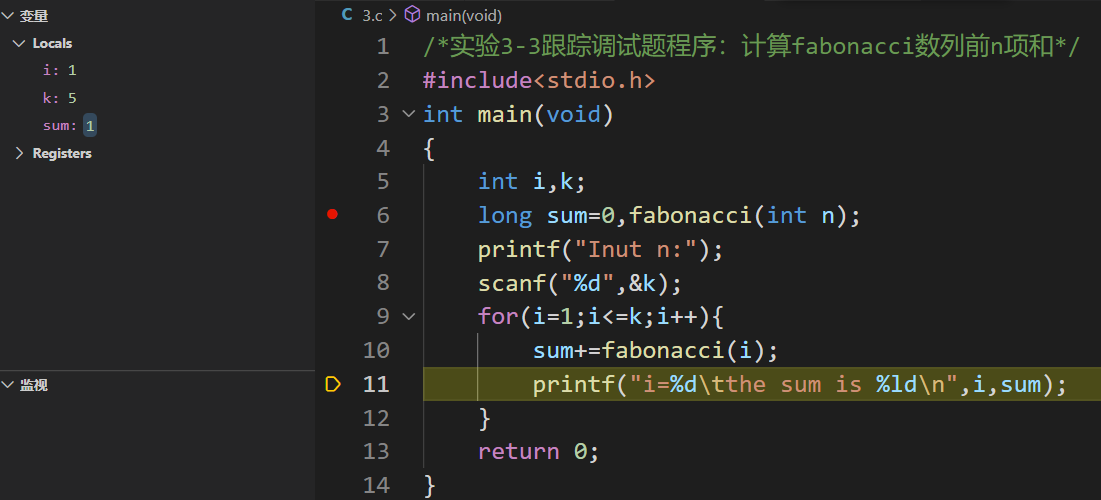


如图，刚执行完“scanf("%d",&k);”语句时，sum、k的值分别为0，5。

（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？

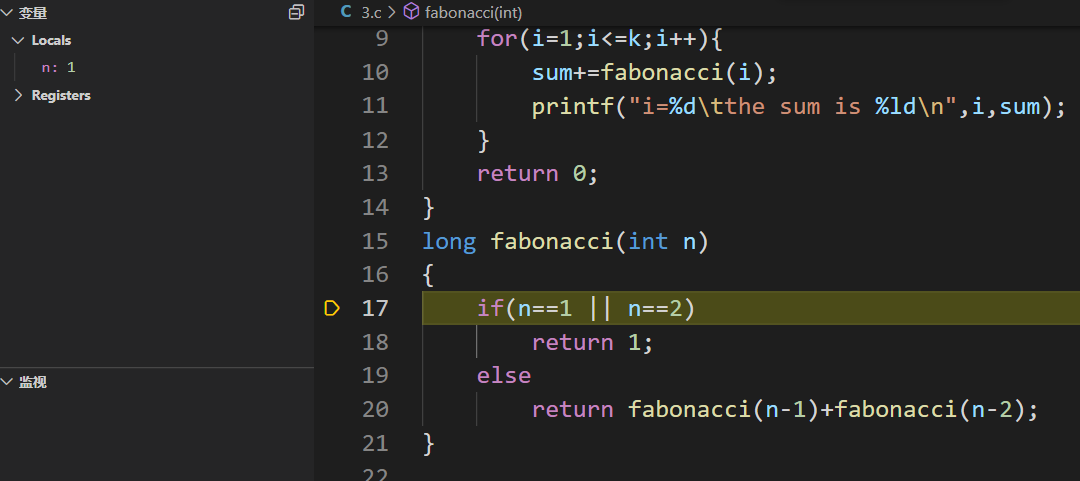


如图，在从main函数第一次进入fabonacci函数前的一刻，观察各变量的值i，k，sum的值分别为1，5，0。



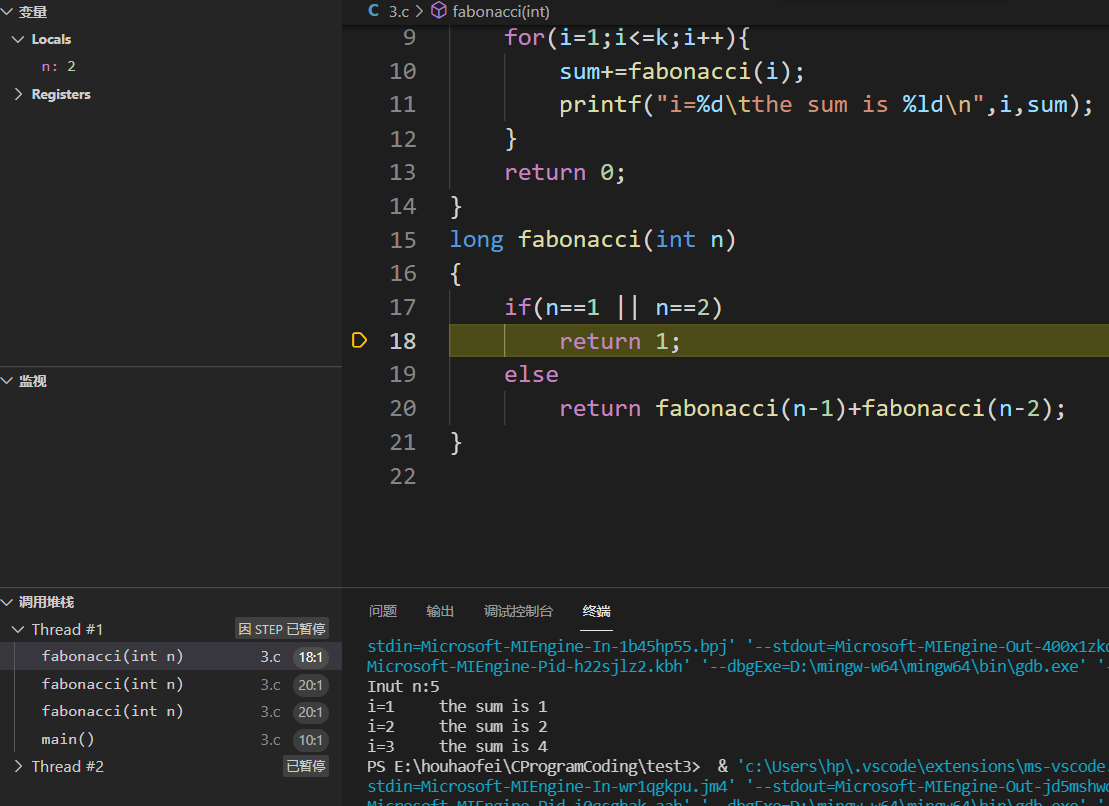
返回后光标停留在如图第11行的输出语句。

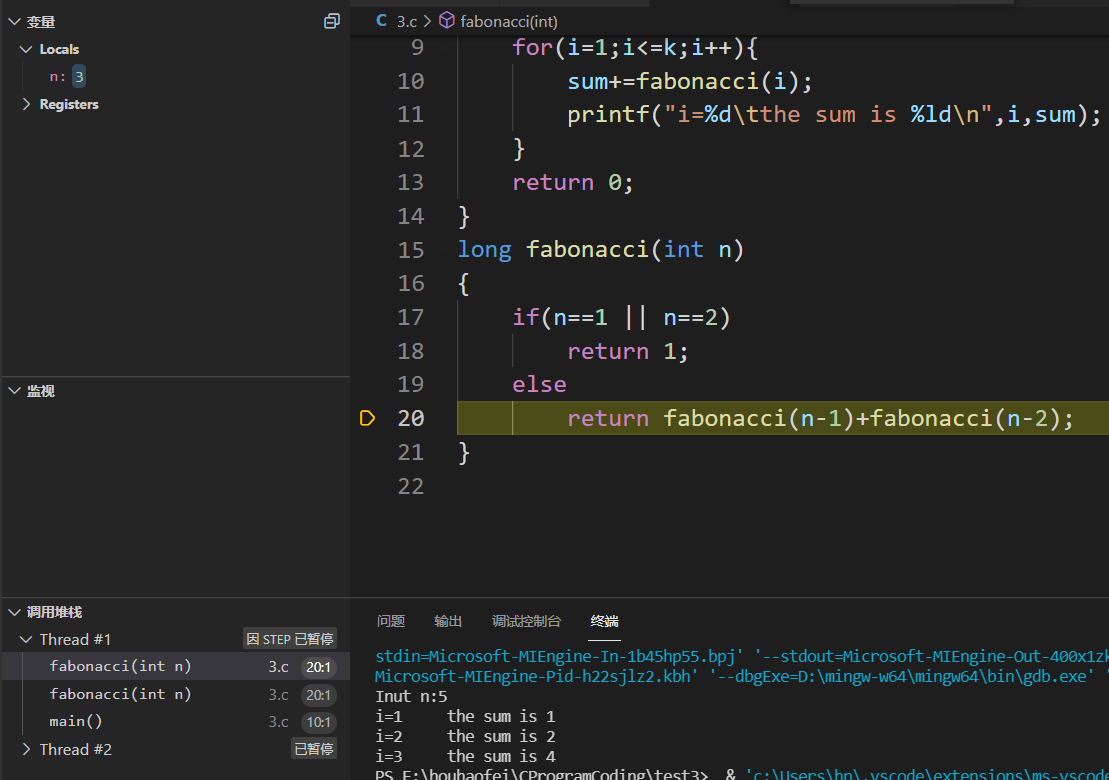
（3）在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了哪里？



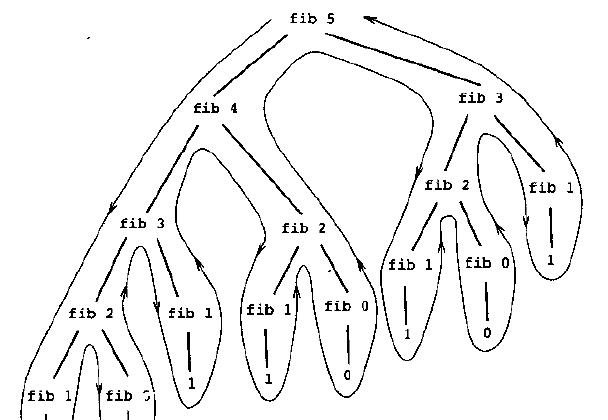
如图，在从main函数第一次进入fabonacci函数后的一刻，观察光条从main函数“sum+=fabonacci(i);”语句调到了fabonacci函数实际执行的第一行，即第17行的if(n==1 || n==2)语句。

（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？





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);。

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。

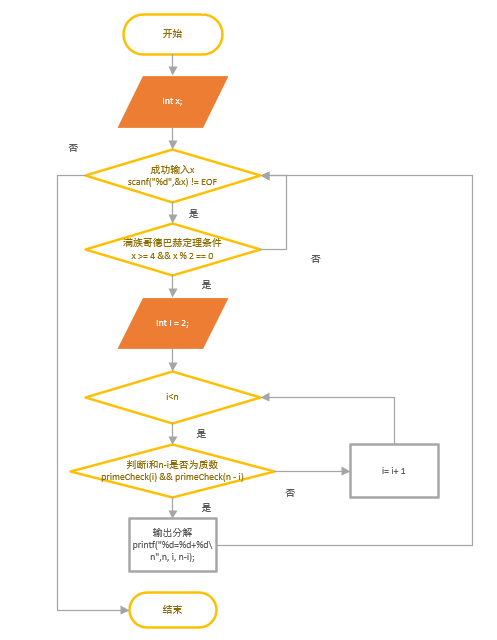


图3-1 编程题1的程序流程图

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

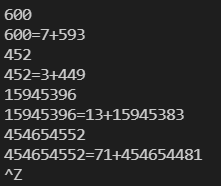
452

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。

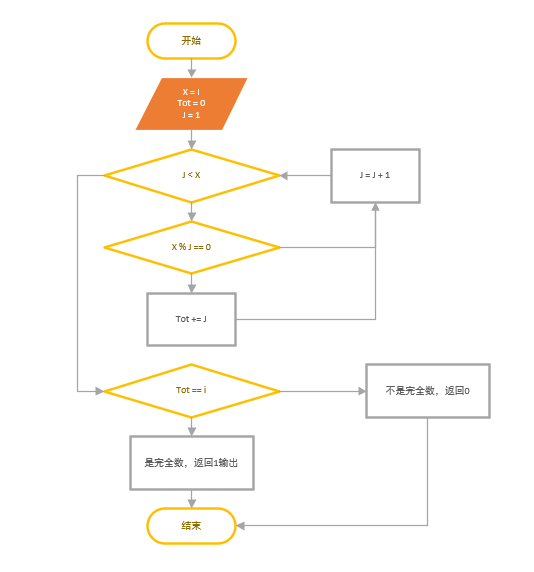


图3-2 编程题2的程序流程图

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

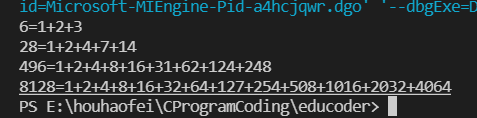


图3-3 编程题2的测试运行结果

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：

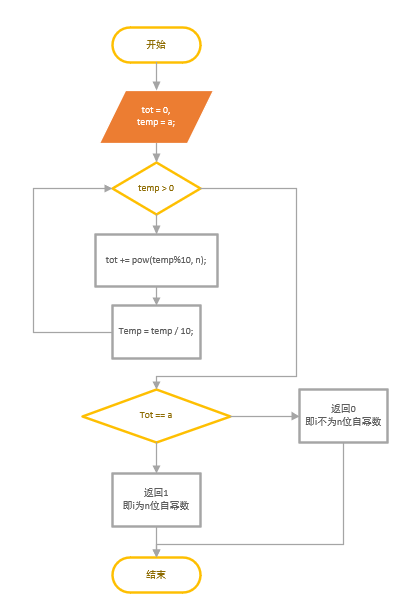


图3-4 编程题3的部分函数程序流程图

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。

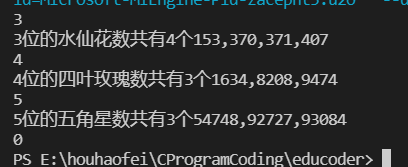


图3-5 编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图3-6所示。

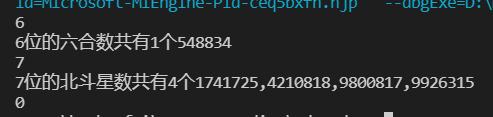


图3-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图3-7所示。

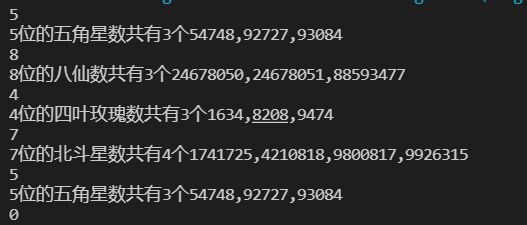


图3-7 编程题3的测试用例三的运行结果

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。

# 4 编译预处理试验

## 4.1 实验目的

（1）掌握文件包含、宏定义、条件编译和assert宏的使用；

（2）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）熟悉多文件编译技术

## 4.2 实验内容

**4.2.1 源程序改错**

**1．程序改错**

下面是用宏来计算平方差、交换两数的源程序.在这个源程序中存在若干错误，要求对该程序进行调试修改，使之能够正确完成指定任务。

1 /\*实验4-1改错与跟踪调试题程序：计算平方差、将换两数\*/

2 #include<stdio.h>

3 #define SUM a+b

4 #define DIF a-b

5 #define SWAP(a,b) a=b,b=a

6 int main()

7 {

8 int a,b;

9 printf("Input two integers a, b:");

10 scanf("%d%d", &a,&b);

11 printf("\nSUM=%d\n the difference between square of a and square of b is:%d",SUM, SUM\*DIF);

12 SWAP(a,b);

13 printf("\nNow a=%d,b=%d\n",a,b);

14 return 0;

15}

**解答：**

（1）错误修改：

1) 第3行为保证计算正确性，应使用括号，正确形式为：

#define SUM (a+b)

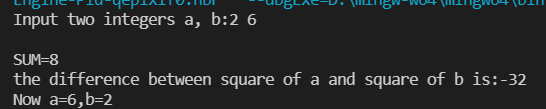
2) 第4行为保证计算正确性，应使用括号，正确形式为：

#define DIF (a-b)

3) 第5行swap不能完成交换的命令，使用交错相减的方法（如实验一），正确形式为：

#define SWAP(a,b) a=b-a,b=b-a,a=a+b

（2）错误修改后运行结果：



**4.2.2 源程序修改替换**

下面是用函数实现求三个数中最大数、计算两浮点数之和的程序。在这个源程序中存在若干语法和逻辑错误。

/\*实验4-2程序修改替换题程序\*/

#include<stdio.h>

int main(void)

{

int a, b, c;

float d, e;

printf("Input three integers:");

scanf("%d %d %d",&a,&b,&c);

printf("\nThe maximum of them is %d\n",max(a,b,c));

printf("Input two floating point numbers:");

scanf("%f %f",&d,&e);

printf("\nThe sum of them is %f\n",sum(d,e));

return 0;

}

int max(int x, int y, int z)

{

int m=z;

if (x>y)

if(x>z) m=x;

else

if(y>z) m=y;

return m;

}

float sum(float x, float y)

{

return x+y;

}

要求：（1）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；

**解答：**

1) 第3行之前为保证通过编译，应对函数进行声明，正确形式为：

int max(int ,int ,int );//进行声明

float sum(float ,float );

2) 第22-26行为保证计算正确性，应正确分析if语句结构，增加花括号，正确形式为：

#define DIF (a-b)

3) 第5行swap不能完成交换的命令，使用交错相减的方法（如实验一），正确形式为：

int max(int x, int y, int z)

{

int m=z;

if (x>y) {

if(x>z) m=x;

}

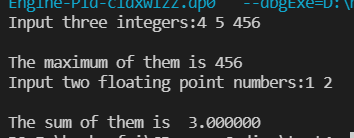
else {

if(y>z) m=y;

}

return m;

}



（2）用带参数的宏替换函数max，来实现求最大数的功能。

**解答：**

我们观察到计算三个数的最大值需要进行多次分类判断，源程序使用if语句，改为带参数的宏之后可以转换为三目运算来进行分类判断。

设需判断x,y,z三个数的最大值。

我们可以先比较x与y。

若x>y，则y不可能是最大值，选出x，z中的最大值即为最大值。

同理，若x<y，则x不可能是最大值，选出y，z中的最大值即为最大值。

宏可如下编写 #define max(a,b,c) (a>b? (a>c? a:c):(b>c? b:c))

因此可编写程序如下：

/\*实验4-2程序修改替换题程序\*/

#include<stdio.h>

#define max(a,b,c) (a>b? (a>c? a:c):(b>c? b:c))

float sum(float ,float );

int main(void)

{

int a, b, c;

float d, e;

printf("Input three integers:");

scanf("%d %d %d",&a,&b,&c);

printf("\nThe maximum of them is %d\n",max(a,b,c));

printf("Input two floating point numbers:");

scanf("%f %f",&d,&e);

printf("\nThe sum of them is %f\n", sum(d,e));

return 0;

}

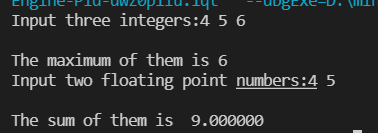
float sum(float x, float y)

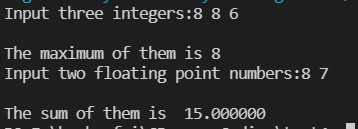
{

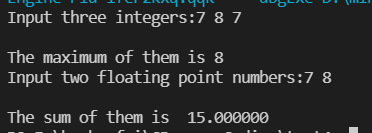
return x+y;

}

程序运行结果如下：







**4.2.3 程序跟踪调试**

下面程序利用R计算圆的面积s，以及面积s的整数部分。

/\*实验4-3跟踪调试题程序利用R计算圆的面积s\*/

#define R

int main(void)

{

float r, s;

int s\_integer=0;

printf ("Input a number: ");

scanf("%f",&r);

#ifdef R

s=3.14159\*r\*r;

printf("Area of round is: %f\n",s);

s\_integer=integer\_fraction(s);

assert((s-s\_integer)<0.5);

printf("The integer fraction of area is %d\n", s\_integer);

#endif

return 0;

}

int integer\_fraction(float x)

{

int i=x;

return i;

}

现要求：

1. 修改程序，使程序编译通过且能运行；

**错误修改：**

1) 第2行之前为保证通过编译，应对使用的库引入，正确形式为：

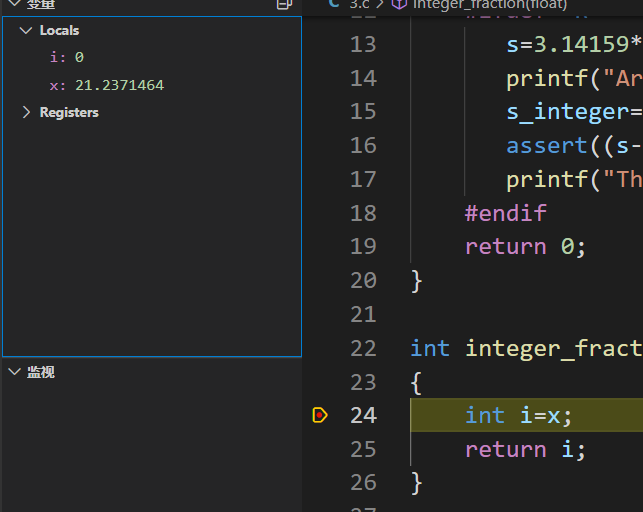
#include <stdio.h>//引用

2) 第4行之前为保证通过编译，应对需要使用的函数声明，正确形式为：

int integer\_fraction(float );//进行声明

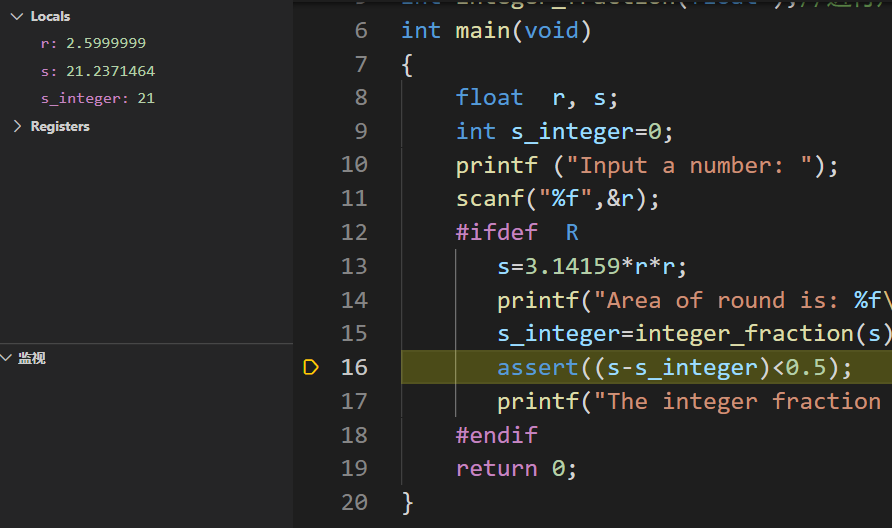
（2）单步执行。进入函数integerl\_fraction时，watch窗口中x为何值？在返回main时, watch窗口中i为何值？设输入5，观察刚执行完“scanf("%d",&k);”语句时，sum、k的值是多少？

**解答：**



当输入x = 2.6时，

进入函数integerl\_fraction时，watch窗口中x为21.2371461



返回main时, watch窗口中不能看到i的值，原因在于i为函数integerl\_fraction内的局部变量，执行结束后i的内存即被释放，不能被main函数使用，也不能被编译器发现。

（3）修改程序，使程序能输出面积s值的整数部分（要求四舍五入），不会输出错误信息assertion failed。

**解答：**

已知断言函数arrset，用于在调试过程中捕捉程序的错误。assert() 会对表达式expression进行检测：如果expression的结果为 0（条件不成立），那么断言失败，表明程序出错，assert() 会向标准输出设备（一般是显示器）打印一条错误信息，并调用 abort() 函数终止程序的执行。

因此我们需要删除（1）题16行的assert((s-s\_integer)<0.5);语句。这样才不会输出错误信息assertion failed。

改为if判断语句，用以进行四舍五入。

若(s-s\_integer)>=0.5 s\_integer++;即可

完整代码如下：

/\*实验4-3跟踪调试题程序利用R计算圆的面积s\*/

#include <stdio.h>//引用

#include <assert.h>

#define R

int integer\_fraction(float );//进行声明

int main(void)

{

float r, s;

int s\_integer=0;

printf ("Input a number: ");

scanf("%f",&r);

#ifdef R

s=3.14159\*r\*r;

printf("Area of round is: %f\n",s);

s\_integer=integer\_fraction(s);

if((s-s\_integer)>=0.5)

s\_integer++;

printf("The integer fraction of area is %d\n", s\_integer);

#endif

return 0;

}

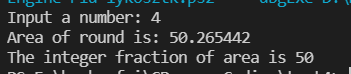
int integer\_fraction(float x) {

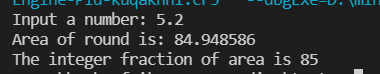
int i=x;

return i;

}

运行截图如下：





**4.2.4 程序设计**

（1） 三角形的面积是，其中，a,b,c为三角形的三边，要求编写程序用带参数的宏来计算三角形的面积。定义两个带参数的宏，一个用来求s，另一个用来求area。

**解答：**

1）本题为简单的顺序流程题目，关键点在于带参数宏的编写。

可构建算法流程图如下：

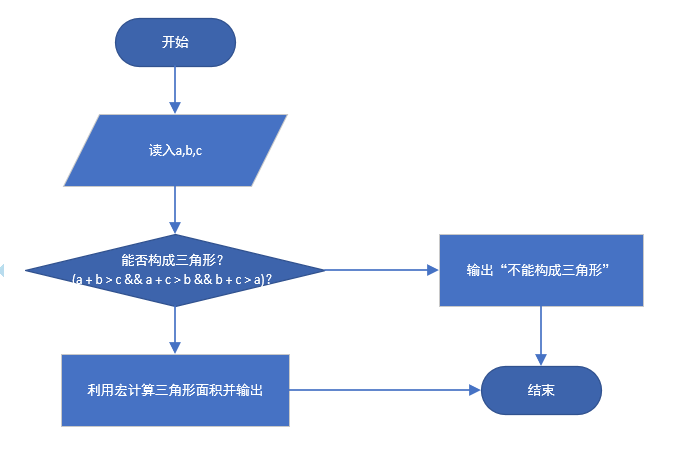


图4-1 编程题1的程序流程图

而编写宏计算三角形面积需要借助海伦－秦九韶公式

假设在平面内，有一个三角形，边长分别为a、b、c，三角形的面积S可由以下公式求得：。而公式里的p为半周长（周长的一半）：。编写带参数的宏p(a,b,c)和S(a,b,c)计算p和S。

对带参数的宏，在展开过程中不仅要进行字符串替换，还要用实参去替换形参。带参宏定义的一般形式为：#define 宏名(形参列表) 字符串

2）源程序清单

#include <stdio.h>

#include <math.h>

#define s(a,b,c) ((a+b+c) / 2.0)

#define area(a,b,c) sqrt(s(a,b,c) \* (s(a,b,c) - a) \* (s(a,b,c) - b) \* (s(a,b,c) - c))

int main () {

float a, b, c;

scanf("%f%f%f", &a, &b, &c);

if(a + b > c && a + c > b && b + c > a)

printf("S = %f\n", area(a, b, c));

else

printf("can not from a triangle\n");

return 0;

}

3）测试

（a） 测试数据：

3 4 5

5 12 13

6 6 6

100 1 1

（b） 对应测试数据的运行结果截图





图4-2 编程题1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。

（2）用条件编译方法来编写程序。输入一行英文字符序列，可以任选两种方式之一输出：一为原文输出；二为变换字母的大小写后输出。例如小写‘a’变成大写‘A’，大写‘D’变成小写‘d’，其他字符不变。用#define命令控制是否变换字母的大小写。例如，#define CHANGE 1 则输出变换后的文字，若#define CHANGE 0则原文输出。

**解答：**

1. 解题思路：

本题需要利用条件编译方法，条件编译是指预处理器根据条件编译指令，有条件地选择源程序代码中的一部分代码作为输出，送给编译器进行编译。主要是为了有选择性地执行相应操作，防止宏替换内容（如文件等）的重复包含。

#if 如果条件为真，则执行相应操作

#elif 如果前面条件为假，而该条件为真，则执行相应操作

#else 如果前面条件均为假，则执行相应操作

#endif 结束相应的条件编译指令

#ifdef 如果该宏已定义，则执行相应操作

#ifndef 如果该宏没有定义，则执行相应操作

其中，#if-#else-#endif在本题中可以使用

#if CHANGE == 0

原文输出

#else

改换大小写后输出（可参考实验1编程题）

#endif

功能为：如果#if后的条件表达式为真，则程序段 1 被选中，否则程序段 2 被选中。

2）程序清单

#include <stdio.h>

#define CHANGE 1

int main () {

char str[10000];

gets(str);

#if CHANGE == 0

printf("%s", str);

#elif CHANGE == 1

for(int i = 0; str[i]; i++) {

if(str[i] >= 'a' && str[i] <= 'z')

str[i] = str[i] - 'a' + 'A';

else if(str[i] >= 'A' && str[i] <= 'Z')

str[i] = str[i] - 'A' + 'a';

}

printf("%s", str);

#endif

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4-1所示。

表4-1 编程题2的测试数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试  用例 | 编译 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| CHANGE | 文本 |
| 例1 | 0 | Hello the World | Hello the World | Hello the World |
| 例2 | 1 | Hello the World | hELLO THE wORLD | hELLO THE wORLD |
| 例3 | 1 | Love the world! AND hello THE world123 | lOVE THE WORLD! and HELLO the WORLD123 | lOVE THE WORLD! and HELLO the WORLD123 |

（b） 对应测试测试用例1的运行结果如图4-3所示。



图4-3 编程题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4-4所示。



图4-4 编程题2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4-5所示。



图4-5 编程题2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）假设一个C程序由file1.c和file2.c两个源文件及一个file.h头文件组成，file1.c、file2.c和file.h的内容分别如下所述。试编辑该多文件C程序，补充file.h头文件内容，然后编译和链接。然后运行最后生成的可执行文件。

/\*源文件file1.c的内容\*/

#include "file.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 \*/

int main(void)

{

x=10;

y=20;

ch=getchar();

printf("in file1 x=%d,y=%d,ch is %c\n",x,y,ch);

func1();

return 0;

}

/\*源文件file2.c的内容为：\*/

#include "file.h"

void func1(void)

{

x++;

y++;

ch++;

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

}

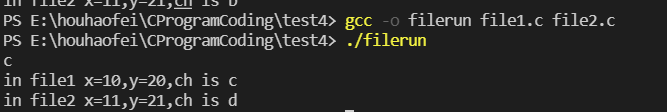
**解答：**

1. 解题思路：

头文件.h文件可以实现所需函数的声明，定义的外部变量可以跨文件，举例来说，如果文件a.c需要引用b.c中变量int v，就可以在a.c中声明extern int v，然后就可以引用变量v。能够被其他模块以extern修饰符引用到的变量通常是全局变量。

为保证编译通过且能够执行需要声明函数void func1(void)，使源文件file2.c实现。也需要声明外部变量int x,y;char ch;为了避免头文件被重复包含，应学会使用#ifndef语句，保证头文件编译时只被包含一次。

而我们要想编译和运行多文件的程序时，应使用命令行输入如下命令。



2）程序清单

/\*源文件file1.c的内容\*/

#include "file.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 \*/

int main(void)

{

x=10;

y=20;

ch=getchar();

printf("in file1 x=%d,y=%d,ch is %c\n",x,y,ch);

func1();

return 0;

}

/\*源文件file2.c的内容为：\*/

#include "file.h"

void func1(void)

{

x++;

y++;

ch++;

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

}

/\*头文件file.h的内容为：\*/

#ifndef \_\_FILE\_H\_\_ //防止重复包含

#define \_\_FILE\_H\_\_

#include <stdio.h>

extern int x,y; /\* 外部变量的定义性说明 \*/

extern char ch; /\* 外部变量的定义性说明 \*/

void func1();

#endif

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4-2所示。

表4-2 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| ch |
| 例1 | a | in file1 x=10,y=20,ch is a  in file2 x=11,y=21,ch is b | in file1 x=10,y=20,ch is a  in file2 x=11,y=21,ch is b |
| 例2 | D | in file1 x=10,y=20,ch is D  in file2 x=11,y=21,ch is E | in file1 x=10,y=20,ch is D  in file2 x=11,y=21,ch is E |

（b） 对应测试测试用例1的运行结果如图4-6所示。

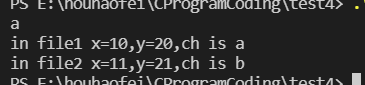


图4-6 编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图4-7所示。

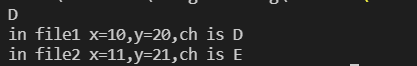


图4-7 编程题3的测试用例二的运行结果

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。

## 4.3 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。

本次实验通过改错题，程序修改替换题，编程题，结合上网搜索资料，较深入的掌握了宏定义，带参数的宏的编写。在跟踪调试题中再次强化了集成开发环境中的调试功能：单步执行、设置断点、观察变量值的能力，了解了第一次遇到的assert宏的使用。通过编程题2了解了条件编译的相关知识，对编译编译预处理的相关知识有了更深入的理解。编程题3卡了我很久，在与同学交流后，明白了自己的误区，外部变量也许在头文件中声明，且需要加extern修饰符，在同学的帮助下，使用命令行完成了多文件编译技术。

# 5 数组实验

## 5.1 实验目的

（1）掌握数组的说明、初始化和使用。

（2）掌握一维数组作为函数参数时实参和形参的用法。

（3）掌握字符串处理函数的设计，包括串操作函数及数字串与数之间转换函数实现算法。

（4）掌握基于分治策略的二分查找算法和选择法排序算法的思想，以及相关算法的实现。

## 5.2 实验内容

**5.2.1 源程序改错**

**1．程序改错**

在下面所给的源程序中，函数strcate(t,s)的功能是将字符串s连接到字符串t的尾部；函数strdelc(s,c)的功能是从字符串s中删除所有与给定字符c相同的字符，程序应该能够输出如下结果：

Programming Language

ProgrammingLanguage Language

ProgrmingLnguge

跟踪和分析源程序中存在的问题，排除程序中的各种逻辑错误，使之能够输出正确的结果。

1 /\*实验5-1程序改错与跟踪调试题程序\*/

2 #include<stdio.h>

3 void strcate(char [],char []);

4 void strdelc(char [],char );

5 int main(void)

6 {

7 char a[]="Language", b[]="Programming";

8 printf("%s %s\n", b,a);

9 strcate(b,a); printf("%s %s\n",b,a);

10 strdelc(b, 'a'); printf("%s\n",b);

11 return 0;

12}

13void strcate(char t[],char s[])

14{

14 int i = 0, j = 0;

15 while(t[i++]) ;

16 while((t[i++] = s[j++] )!= '\0');

17}

18void strdelc(char s[], char c)

19{

20 int j,k;

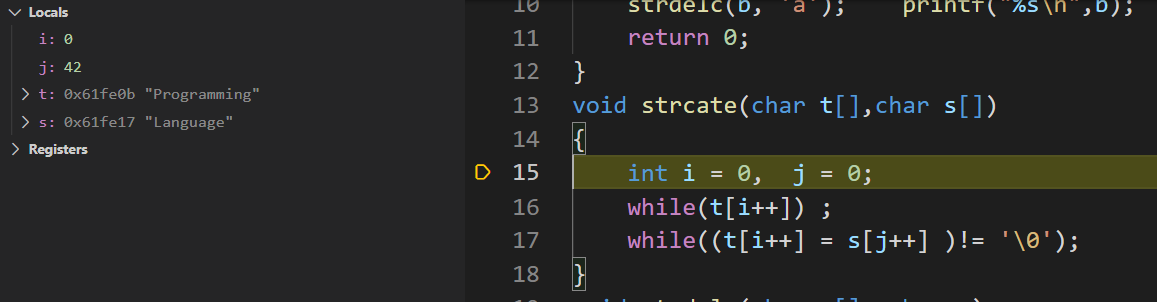
21 for(j=k=0; s[j] != '\0'; j++)

22 if(s[j] != c) s[k++] = s[j];

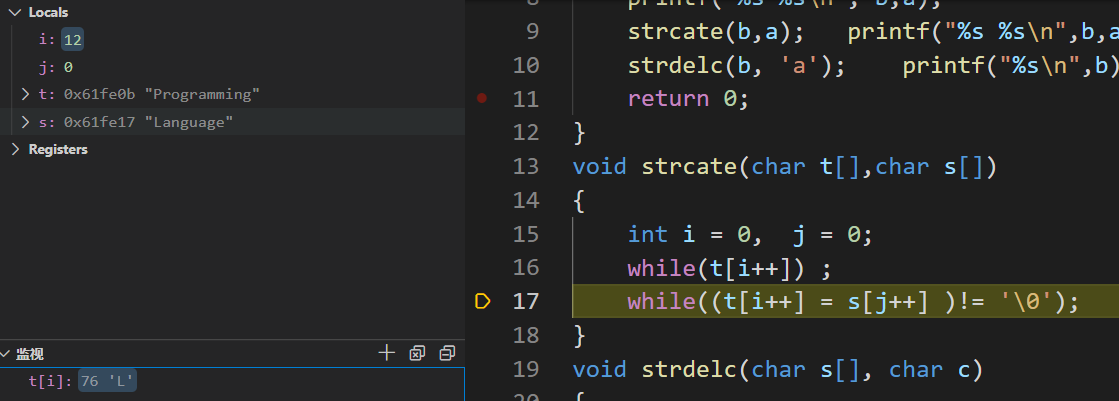
23}

**解答：**

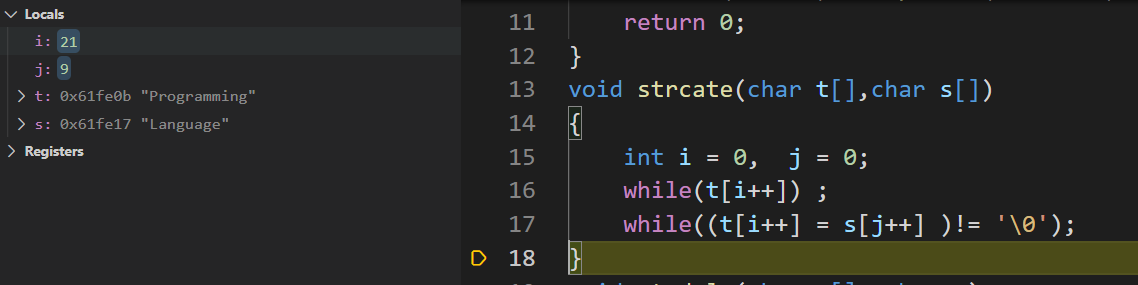
（1）单步执行源程序。进跟踪进入strcate时，观察字符数组t和s中的内容，分析结果是否正确。当单步执行光条刚落在第二个while语句所在行时，i为何值？t[i]为何值？分析该结果是否存在问题。当单步执行光条落在strcate函数块结束标记即右花括号“}”所在行时，字符数组t和s分别为何值？分析是否实现了字符串连接。



单步执行源程序。进跟踪进入strcate时，观察字符数组t和s中的内容，结果并无问题。

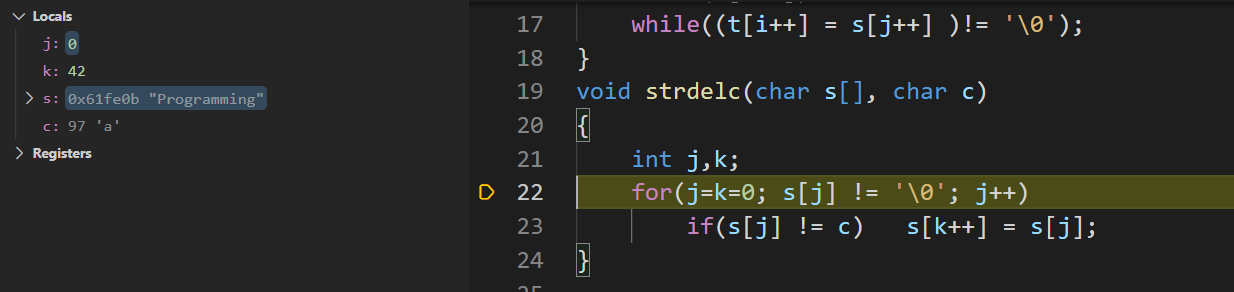


当单步执行光条刚落在第二个while语句所在行时，i为12，t[i]为’L’。结果出现了问题，i本应该为t数组的长度10，t[i]指向’\0’。应该是循环中的t[i++]语句使得，程序在运行到t[i]=’\0’时先i++再进行判断，使得i偏大。而两个字符数组相连，出现越界，产生重大问题。



当单步执行光条落在strcate函数块结束标记即右花括号“}”所在行时，字符数组t和s分别仍为原值不变，分析得是因为数组越界后，t[i]与s[j]的增长完全保持一致，没有起到字符串连接的作用。

（2）跟踪进入函数strdelc时，观察字符数组s中的内容和字符c的值，分析结果是否正确。单步执行for语句过程中，观察字符数组s, j和k值的变化，分析该结果是否存在问题。当单步执行光条落在strdelc函数块结束标记“}”所在行时，字符串s为何值？分析是否实现了所要求的删除操作。



跟踪进入函数strdelc时，字符数组s中的内容并未发生改变，与上一题分析的原因相关，而字符c的值正确。

单步执行单步执行for语句过程中，观察字符数组s, j和k值的变化，k始终小于等于j，保证了删除的正确性，但是由于最后’g’字符未作处理，产生了删除错误，应当删除后重新添加结尾标识符’\0’。

（3）错误修改：

1) 第7行为保证后序修改时不出现数组越界的情况，应大开数组，正确形式为：

char a[1000]="Language", b[1000]="Programming";

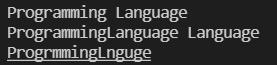
2) 第16行，保证指针指向’\0’的点，正确形式为：

while(t[++i]) ;

3) 第24行为使字符串正确的输出，不会出现运行时错误，正确形式为：

s[k++] = '\0';//增加终止

（4）错误修改后运行结果：



**5.2.2 源程序修改替换**

下面的源程序用于求解瑟夫问题：M个人围成一圈，从第一个人开始依次从1至N循环报数，每当报数为N时报数人出圈，直到圈中只剩下一个人为止。

（1）请在源程序中的下划线处填写合适的代码来完善该程序。

#include<stdio.h>

#define M 10

#define N 3

int main(void)

{

int a[M], b[M]; /\* 数组a存放圈中人的编号，数组b存放出圈人的编号 \*/

int i, j, k;

for(i = 0; i < M; i++) /\* 对圈中人按顺序编号1—M \*/

a[i] = i + 1;

for(i = M, j = 0; i > 1; i--){

/\* i表示圈中人个数，初始为M个，剩1个人时结束循环；j表示当前报数人的位置 \*/

for(k = 1; k <= N; k++) /\* 1至N报数 \*/

if(++j > i - 1) j = 0;/\* 最后一个人报数后第一个人接着报，形成一个圈 \*/

b[M-i] = j ? \_\_\_\_\_\_\_:\_\_\_\_\_\_; /\* 将报数为N的人的编号存入数组b \*/

if(j)

for(k = --j; k < i; k++) /\* 压缩数组a，使报数为N的人出圈 \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

for(i = 0;i < M-1; i++) /\* 按次序输出出圈人的编号 \*/

printf(“%6d”, b[i]);

printf(“%6d\n”, a[0]); /\* 输出圈中最后一个人的编号 \*/

return 0;

}

**解答：**

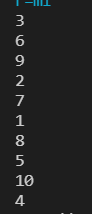
1) 第19行将报数为N的人（现在仅剩的人数中的第j个人）的编号存入数组b，当j>0时，由a数组的定义（a数组从0到i-1存放仅剩余的i个人，而0到i-1依次对应剩下的第1到第i个人的编号）知，报数为N的人的编号为a[j-1]。而j=0时，即j=i而被取模回来，故报数为N的人的编号为a[i-1]。正确形式如下：

b[M-i] = j ? a[j-1]:a[i-1]; /\* 将报数为N的人的编号存入数组b \*/

2) 第22行，压缩数组a，使报数为N的人出圈。即现在仅剩的人数中的第j个人，a[j-1]元素删除，j=j-1。第22行的正确形势故如下：

a[k] = a[k+1];

运行结果如下：



（2）上面的程序中使用数组元素的值表示圈中人的编号，故每当有人出圈时都要压缩数组，这种算法不够精炼。如果采用做标记的办法，即每当有人出圈时对相应数组元素做标记，从而可省掉压缩数组的时间，这样处理效率会更高一些。请采用做标记的办法修改程序，并使修改后的程序与原程序具有相同的功能。

**解答：**

我们仍然用j从0到M-1为M个人标号，以保证数组使用的正确性。而1到M-1依次对应为第1到第M-1人的编号，0为第M个人的编号。并为他们建立是否出圈的标记c[M]数组。若其仍在圈中c[i] = 1，否则为0。

可构建算法流程图如下：

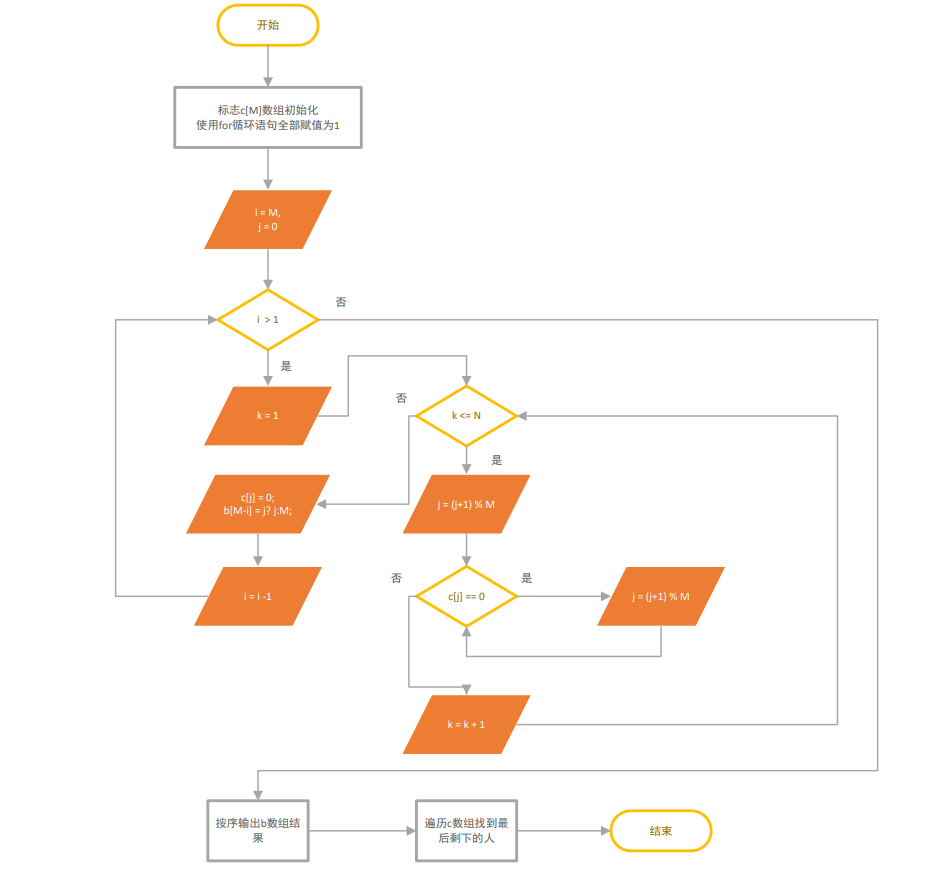


图5-1 源程序修改替换题的程序流程图

#include<stdio.h>

#define M 10

#define N 3

int main(void)

{

int c[M], b[M];

int i, j, k;

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

c[i] = 1;

for(i = M, j = 0; i > 1; i--){

for(k = 1; k <= N; k++) {/\* 1至N报数 \*/

j = (j + 1) % M;

while(c[j] == 0) {

j = (j + 1) % M;

}

}

c[j] = 0;

b[M-i] = j? j:M;

}

for(i = 0;i < M-1; i++) /\* 按次序输出出圈人的编号 \*/

printf("%d\n", b[i]);

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

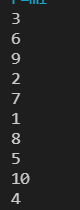
if(c[i])

printf("%d\n",i? i: M);

return 0;

}

运行结果如下：



**5.2.3 程序设计**

（1） 输入一个整数，将它在内存中二进制表示的每一位转化成对应的数字字符并且存放到一个字符数组中，然后输出该整数的二进制表示。

**解答：**

1）本题为简单的循环流程题目，但是需要借助数组进行编写。

可构建算法流程图如下：

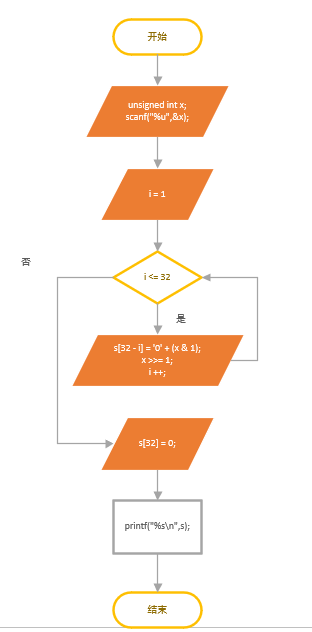


图5-2 编程题1的程序流程图

2）源程序清单

#include <stdio.h>

char s[100];

int main () {

unsigned int x;

scanf("%u",&x);

for(int i = 1; i <= 32; i++) {

s[32 - i] = '0' + (x & 1);

x >>= 1;

}

s[32] = 0;

printf("%s\n",s);

return 0;

}

3）测试

（a） 测试数据：

7

-7

456456456

-987987987

（b） 对应测试数据的运行结果截图









图5-3 编程题1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。

（2）编写一个C程序，要求采用模块化程序设计思想，将相关功能用函数实现，并提供菜单选项。该程序具有以下功能：

①“成绩输入”，输入n个学生的姓名和C语言课程的成绩。

②“成绩排序”，将成绩按从高到低的次序排序，姓名同时进行相应调整。成绩相同的，按照输入先后次序排列。

③“成绩输出”，输出排序后所有学生的姓名和C语言课程的成绩。

④“成绩查找”，输入一个C语言课程成绩值，用二分查找进行搜索。如果查找到有该成绩，则输出该成绩学生的姓名和C语言课程的成绩；否则，输出提示“not found!”。

**解答：**

1）解题思路：

本题分析与建模后应完成如下函数：

1. input函数：输入学生数量n，完成n个学生的姓名和C语言成绩的输入至两个储存信息的数组Name[1000][1000], Score[1000]。Sorted标志变量设为0，保证成绩输出和查找时的信息有序。
2. sort函数：按照成绩将Name和Score数组排序，本程序使用了冒泡排序法。比较相邻的元素的Score。如果第一个比第二个小，就交换他们（包含Name和Score），对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最小的数。持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。越大的元素会经由交换慢慢"浮"到数列的顶端。Sorted标志变量置为1，表明学生已经按照成绩排序，信息有序。
3. output函数：若Sorted标志变量为0，则需调用sort函数排序后按序输出。若Sorted标志变量为1，则直接按序输出。
4. Find函数：若Sorted标志变量为0，则需调用sort函数排序后再进行二分查找。二分查找就是将查找的键和子数组的中间键作比较，如果被查找的键小于中间键，就在左子数组继续查找；如果大于中间键，就在右子数组中查找，否则中间键就可能是要找的元素。若ans == n + 1 或者 Score[ans] != x，可能不存在该成绩的同学。而也有可能多位同学有同一成绩，应通过向前遍历找到同一成绩的同学的数目。

2）程序清单：

#include <stdio.h>

#include <string.h>

void sort(char Name[][1000], int Score[],int \*n, int \*sorted) {

for(int i = 1; i < (\*n); i++)

for(int j = 1; j <= (\*n) - i; j++) {

if(Score[j] < Score[j+1]) {

int temp = Score[j+1];

Score[j+1] = Score[j];

Score[j] = temp;

char TEMP[1000];

memcpy(TEMP, Name[j+1], sizeof(TEMP));

memcpy(Name[j+1], Name[j], sizeof(TEMP));

memcpy(Name[j], TEMP, sizeof(TEMP));

}

}

(\*sorted) = 1;

}

void input(char Name[][1000], int Score[],int \*n, int \*sorted) {

\*sorted = 0;

scanf("%d", n);

for(int i = 1; i <= \*n; i++)

scanf("%s%d",Name[i], &Score[i]);

printf("%d records were input!\n",\*n);

}

void Find(char Name[][1000], int Score[],int \*n, int \*sorted) {

if(sorted == 0)

sort(Name, Score, n, sorted);

int x = 0;

scanf("%d", &x);

int L = 1, R = \*n, mid, ans = \*n + 1;

while(L <= R) {

mid = (L + R) >> 1;

if(Score[mid] >= x) {

ans = mid;

L = mid + 1;

}

else

R = mid - 1;

}

if(ans == \*n + 1 || Score[ans] != x)

printf("not found!\n");

else {

int tot = 0;

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

if(Score[ans - i + 1] == x)

tot ++;

for(int i = ans - tot + 1; i <= ans; i++)

printf("%s %d\n",Name[i], Score[i]);

}

}

void output(char Name[][1000], int Score[],int \*n, int \*sorted) {

for(int i = 1; i <= \*n; i++)

printf("%s %d\n",Name[i],Score[i]);

}

int main () {

int opt;

char Name[1000][1000];int Score[1000];int n;int sorted = 0;

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

if(opt == 0)

break;

else if(opt == 1)

input(Name, Score, &n, &sorted);

else if(opt == 2) {

sort(Name, Score, &n, &sorted);

printf("Reorder finished!\n");

}

else if(opt == 3)

output(Name, Score, &n, &sorted);

else if(opt == 4)

Find(Name, Score, &n, &sorted);

}

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5-1所示。

表5-1 编程题2的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 操作与信息 |
| 例1 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  0 | 6 records were input! | 6 records were input! |
| 例2 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  0 | 6 records were input!  Reorder finished! | 6 records were input!  Reorder finished! |
| 例3 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  3  0 | 6 records were input!  Reorder finished!  Jack 95  Andy 95  Mike 90  Rose 89  Sophia 77  Joe 75 | 6 records were input!  Reorder finished!  Jack 95  Andy 95  Mike 90  Rose 89  Sophia 77  Joe 75 |
| 例4 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4 89  0 | 6 records were input!  Reorder finished!  Rose 89 | 6 records were input!  Reorder finished!  Rose 89 |
| 例5 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4  88  0 | 6 records were input!  Reorder finished!  not found! | 6 records were input!  Reorder finished!  not found! |
| 例6 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4 77  0 | 6 records were input!  Reorder finished!  Sophia 77 | 6 records were input!  Reorder finished!  Sophia 77 |
| 例7 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  4 95  0 | 6 records were input!  Jack 95  Andy 95 | 6 records were input!  Jack 95  Andy 95 |

（b） 对应测试测试用例1的运行结果如图5-4所示。

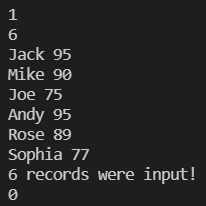


图5-4 编程题2的测试用例一的运行结果

对应测试测试用例2的运行结果如图5-5所示。

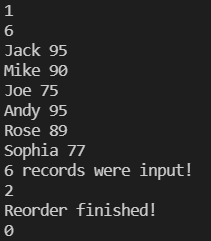


图5-5 编程题2的测试用例二的运行结果

对应测试测试用例3的运行结果如图5-6所示。

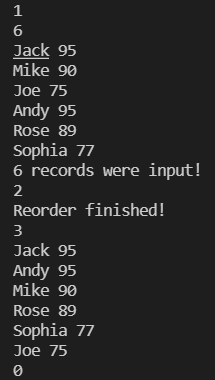


图5-6 编程题2的测试用例三的运行结果

对应测试测试用例4的运行结果如图5-7所示。

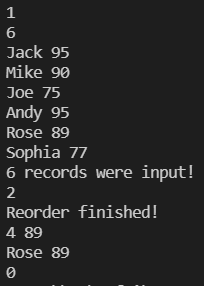


图5-7 编程题2的测试用例四的运行结果

对应测试测试用例5的运行结果如图5-8所示。

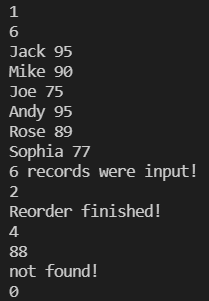


图5-8 编程题2的测试用例五的运行结果

对应测试测试用例6的运行结果如图5-9所示。

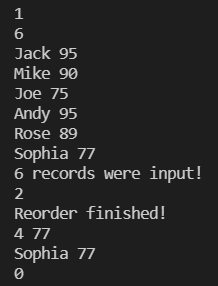


图5-9 编程题2的测试用例六的运行结果

对应测试测试用例7的运行结果如图5-10所示。

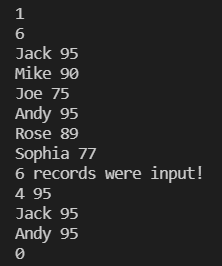


图5-10 编程题2的测试用例七的运行结果

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）求解N皇后问题，即在N×N的棋盘上摆放N个皇后，要求任意两个皇后不能在同一行、同一列、同一对角线上。输入棋盘的大小N（N取值1-10），如果能满足摆放要求，则输出所有可能的摆放法的数量，否则输出“无解。”

**解答：**

1）解题思路：

求解N皇后问题是算法中回溯法应用的一个经典案例。

回溯算法也叫试探法，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。

算法基本流程如下：

1.算法开始, 清空棋盘，当前行设为第一行，当前列设为第一列

2.在当前行，当前列的位置上判断是否满足条件(即保证经过这一点的行,列与斜线上都没有两个皇后)，若不满足，跳到第4步

3.在当前位置上满足条件的情形：

3.1在当前位置放一个皇后，若当前行是最后一行，记录一个解；

3.2若当前行不是最后一行，当前行设为下一行, 当前列设为当前行的第一个待测位置；

3.3若当前行是最后一行，当前列不是最后一列，当前列设为下一列；

3.4若当前行是最后一行，当前列是最后一列，回溯，即清空当前行及以下各行的棋盘，然后，当前行设为上一行，当前列设为当前行的下一个待测位置；

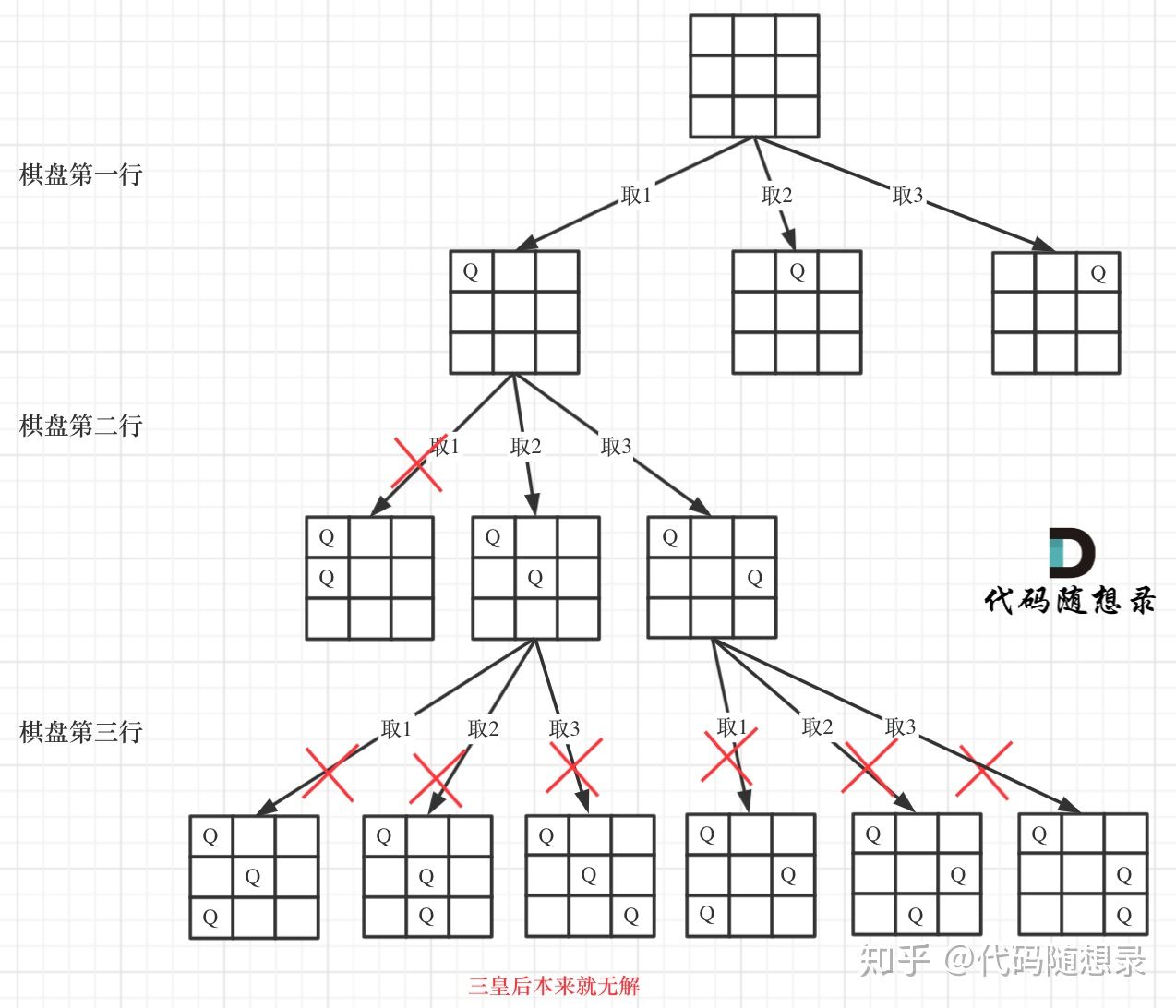
3.5以上返回到第2步

4.在当前位置上不满足条件的情形：

4.1若当前列不是最后一列，当前列设为下一列，返回到第2步;

4.2若当前列是最后一列了，回溯，即，若当前行已经是第一行了，算法退出，否则，清空当前行及以下各行的棋盘，然后，当前行设为上一行，当前列设为当前行的下一个待测位置，返回到第2步;

可有如下模拟（来源自互联网）：



而判断当前位置是否能够放入皇后时，需要判断之前的列，左对角线，右对角线是否为空，我们需要建立三个标记数组，用以标记每个列，左对角线，右对角线是否有放置皇后，因此需对每一个点进行编号，初始化maplex，maprex数组（左对角线编号，右对角线编号），建立一个N×N数对到1到2\*N – 1的映射。

2）程序清单

#include <stdio.h>

int N, tot, row[100], lex[100], rex[100];

int maplex[100][100], maprex[100][100];

void pre () {

int count = 0;

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

count ++;

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

maplex[j][N-i+j] = count;

}

for(int i = 2; i <= N; i++) {

count ++;

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

maplex[i+j-1][j] = count;

}

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

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

maprex[i][j] = maplex[i][N-j+1];

}

void dfs (int n) {

if(n == N + 1) {

tot ++;

return ;

}

for(int j = 1; j <= N; j ++) {

if((!row[j]) && (!lex[maplex[n][j]]) && (!rex[maprex[n][j]])) {

row[j] = lex[maplex[n][j]] = rex[maprex[n][j]] = 1;

dfs(n+1);

row[j] = lex[maplex[n][j]] = rex[maprex[n][j]] = 0;

}

}

}

int main () {

scanf("%d", &N);

pre();

dfs(1);

if(tot == 0)

printf("无解");

else

printf("%d\n", tot);

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5-2所示。

表5-2 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 | 1 | 1 | 1 |
| 例2 | 2 | 无解 | 无解 |
| 例3 | 6 | 4 | 4 |
| 例4 | 9 | 352 | 352 |

（b） 对应测试测试用例1的运行结果如图5-11所示。



图5-11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5-12所示。



图5-12 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5-13所示。



图5-13 编程题3的测试用例三的运行结果

对应测试测试用例4的运行结果如图5-14所示。



图5-14 编程题3的测试用例四的运行结果

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。

## 5.3 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。

通过本次实验熟练地掌握了数组的说明、初始化和使用，和一维数组作为函数参数时实参和形参的用法。在改错题一中掌握了部分字符串处理函数的写法，了解了字符串信息的特殊性，也更深刻的理解C语言应时时刻刻关注数组越界问题的内涵。也在分析约瑟夫问题的过程中，加深了自己对数学模型的分析与建构能力。编程题2中掌握了基于分治策略的二分查找算法和选择法排序算法的思想，以及相关算法的实现，初步使用函数写了比较大型的C语言代码。在编程题3的N皇后问题中又一次加深了对递归策略的理解，使用深度优先搜索的方法实际枚举解决问题。

# 6 指针实验

## 6.1 实验目的

（1）熟练掌握指针的说明、赋值、使用。

（2）掌握用指针引用数组的元素，熟悉指向数组的指针的使用。

（3）熟练掌握字符数组与字符串的使用，掌握指针数组及字符指针数组的用法。

（4）掌握指针函数与函数指针的用法。

（5）掌握带有参数的main函数的用法。

## 6.2 实验内容

**6.2.1 源程序改错**

**1．程序改错**

在下面所给的源程序中，函数strcopy(t, s)的功能是将字符串s复制给字符串t，并且返回串t的首地址。请单步跟踪程序，根据程序运行时出现的现象或观察到的字符串的值，分析并排除源程序的逻辑错误，使之能按照要求输出如下结果：

Input a string:

programming↙ （键盘输入）

programming

Input a string again:

language↙ （键盘输入）

language

1 #include<stdio.h>

2 char \*strcopy(char \*, const char \*);

3 int main(void)

4 {

5 char \*s1, \*s2, \*s3;

6 printf("Input a string:\n", s2);

7 scanf("%s", s2);

8 strcopy(s1, s2);

9 printf("%s\n", s1);

10 printf("Input a string again:\n", s2);

11 scanf("%s", s2);

12 s3 = strcopy(s1, s2);

13 printf("%s\n", s3);

14 return 0;

15}

16

17/\*将字符串s复制给字符串t，并且返回串t的首地址\*/

18char \* strcopy(char \*t, const char \*s)

19{

20 while(\*t++ = \*s++);

21 return (t);

22}

**解答：**

1. 错误修改：

1) 第2行为保证编译正确，应引入新的库，正确形式为：

#include <stdlib.h>//导入新的库

2) 第7行为保证后序输入和处理时不会发生段超越的错误，应对指针初始化，正确形式为：

s1 = (char \*)malloc(sizeof(char) \* 100);//分配内存.

s2 = (char \*)malloc(sizeof(char) \* 100);

s3 = (char \*)malloc(sizeof(char) \* 100);

3) 第24，25行为使函数返回正确的字符指针首地址，应设立新的指针，正确形式为：

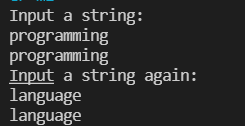
char \*p = t;//不修改t，修改p

while(\*p++ = \*s++);

4) 第26行为使字符串正确输出应增加终止符，正确形式为：

\*p++ = '\0';//终止符

（2）错误修改后运行结果：



**6.2.2** **源程序修改替换**

（1）下面程序中函数strsort用于对字符串进行升序排序，在主函数中输入N个字符串（字符串长度不超过49）存入通过malloc动态分配的存储空间，然后调用strsort对这N个串按字典序升序排序。

①请在源程序中的下划线处填写合适的代码来完善该程序。

#include<stdio.h>

#include<\_\_\_\_\_\_\_\_\_>

#include<string.h>

#define N 4

/\*对指针数组s指向的size个字符串进行升序排序\*/

void strsort(char \*s[], int size)

{

\_\_\_\_\_\_\_temp;

int i, j;

for(i=0; i<size-1; i++)

for (j=0; j<size-i-1; j++)

if (\_\_\_\_\_\_\_\_\_\_\_)

{

temp = s[j];

\_\_\_\_\_\_\_\_\_\_\_;

s[j+1] = temp;

}

}

int main()

{

int i;

char \*s[N], t[50];

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

{

gets(t);

s[i] = (char \*)malloc(strlen(t)+1);

strcpy(\_\_\_\_\_\_\_);

}

strsort(\_\_\_\_\_\_\_\_);

for (i=0; i<N; i++) {puts(s[i]); free(s[i]);}

return 0;

}

**解答：**

1. 第2行，为正确使用malloc函数，正确形式如下：

#include<stdlib.h>

2) 第3行，由二维数组存储的本质实际为指向一维字符数组的指针已知，为完成数组的交换，应定义一维字符指针变量，故正确形式如下：

char \*temp;

3) 第12行，为实现按字典序冒泡排序，应使用strcmp函数，strcmp(char \*a, char \*b) 当a的字典序大于b时返回正数，故正确形式如下：

if (strcmp(s[j], s[j+1]) > 0)

4) 第15行，为实现s[j]和s[j+1]的交换，故正确形式如下：

s[j] = s[j+1];

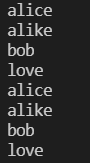
5) 第28行，正确保存每一个字符串，正确形式如下：

strcpy(s[i], t);

6) 第30行，正确调用定义的void strsort(char \*s[], int size)函数，正确形式如下：

strsort(s, N);

程序运行结果如下：



②数组作为函数参数其本质类型是指针。例如，对于形参char \*s[]，编译器将其解释为char \*\*s，两种写法完全等价。请用二级指针形参重写strsort函数，并且在该函数体的任何位置都不允许使用下标引用。

**解答：**

为保证函数体的任何为止都不允许下表的使用，我们使用两个二维字符指针q,s指向冒泡排序中我们要交换的两个一维字符串数组。随着j增加，两个二维指针不断自加1，可以直接指到下一个字符串。使用\*q, \*s取出字符串。

void strsort(char \*\*s, int size)

{

char \*temp;

int i, j;char \*\*q, \*\*r;

for(i=0; i<size-1; i++)

for (q = s, r = s+1, j=0; j<size-i-1; j++,q++,r++)

if (strcmp(\*q, \*r) > 0)

{

temp = \*q;

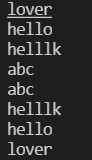
\*q = \*r;

\*r = temp;

}

}

运行结果如下：



（2）下面源程序通过函数指针和菜单选择来调用库函数实现字符串操作；串复制strcpy、串连接strcat或串分解strtok。

①请在源程序中的下划线处填写合适的代码来完善该程序，使之能按照要求输出下面结果：

1 copy string.

2 connect string.

3 parse string.

4 exit.

input a number (1-4) please!

2↙ （键盘输入）

input the first string please!

the more you learn,↙ （键盘输入）

input the second string please!

the more you get. ↙ （键盘输入）

the result is the more you learn, the more you get.

代码如下：

# include<stdio.h>

# include<string.h>

int main (void)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

char a[80], b[80], \*result;

int choice;

while(1)

{

do

{

printf("\t\t1 copy string.\n");

printf("\t\t2 connect string.\n");

printf("\t\t3 parse string.\n");

printf("\t\t4 exit.\n");

printf("\t\tinput a number (1-4) please.\n");

scanf("%d", &choice);

}while(choice<1 || choice>4);

switch(choice)

{

case 1: p = strcpy; break;

case 2: p = strcat; break;

case 3: p = strtok; break;

case 4: goto down;

}

getchar();

printf("input the first string please!\n");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

printf("input the second string please!\n");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

result = \_\_\_\_\_\_\_\_\_\_\_(a, b);

printf("the result is %s\n", result);

}

down:

return 0;

}

**解答：**

1. 第2行，为正确声明函数指针，正确形式如下：

char\* (\*p) (char \*a,char \*b);

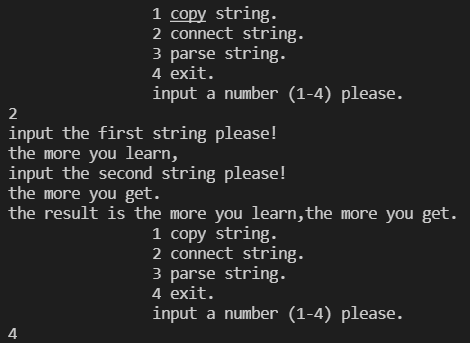
2) 第29，31行，使用C语言读入一行输入，故正确形式如下：

gets(a); gets(b);

3) 第32行，为正确使用函数指针调用所指代的函数，故正确形式如下：

result = p(a, b);

程序运行结果如下：



②函数指针的一个用途是用户散转程序，即通过一个转移表（函数指针数组）来实现多分枝函数处理，从而省去了大量的if语句或者switch语句。转移表中存放了各个函数的入口地址（函数名），根据条件的设定来查表选择执行相应的函数。请使用转移表而不是switch语句重写以上程序。

**解答：**

定义函数指针数组，然后通过函数指针数组实现拥护函数分支的处理。

数组是一个存放相同类型数据的存储空间，我们已经学习了指针数组，比如：

int \*arr[10];//数组的每个元素是int\*

那要把函数的地址存到一个数组中，那这个数组就叫函数指针数组，比如：

int (\*parr1[10])();

分析：parrl先和[]结合，说明parr1是数组，数组的内容是什么呢?是int (\*)() 类型的函数指针。

代码实现如下：

# include<stdio.h>

# include<string.h>

int main (void)

{

char\* (\*p[]) (char\* , char\*) = {0, strcpy, strcat, strtok};

char a[80], b[80], \*result;

int choice;

while(1)

{

do

{

printf("\t\t1 copy string.\n");

printf("\t\t2 connect string.\n");

printf("\t\t3 parse string.\n");

printf("\t\t4 exit.\n");

printf("\t\tinput a number (1-4) please.\n");

scanf("%d", &choice);

}

while(choice<1 || choice>4);

if(choice == 4)

break;

getchar();

printf("input the first string please!\n");

gets(a);//

printf("input the second string please!\n");

gets(b);//

result = p[choice](a, b);//

printf("the result is %s\n", result);

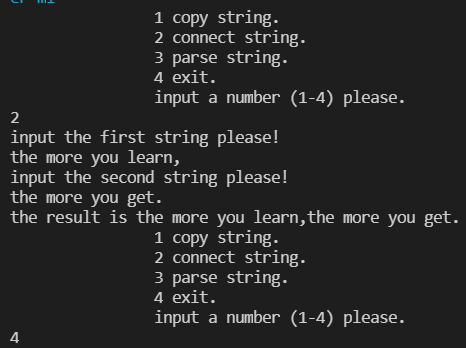
}

down:

return 0;

}

代码运行结果如下：



**6.2.3 跟踪调试题**

请按下面的要求对源程序进行操作，并回答问题和排除错误。

#include "stdio.h"

char \*strcpy(char \*,char \*);

void main(void)

{

char a[20],b[60]="there is a boat on the lake.";

printf("%s\n",strcpy(a,b));

}

char \*strcpy(char \*s,char \*t)

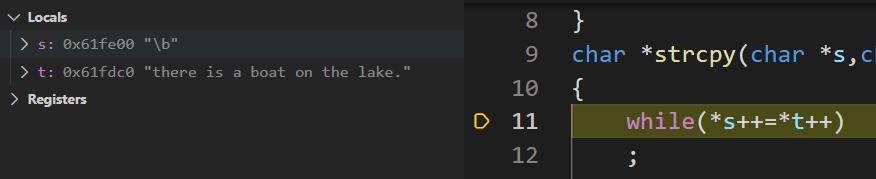
{

while(\*s++=\*t++);

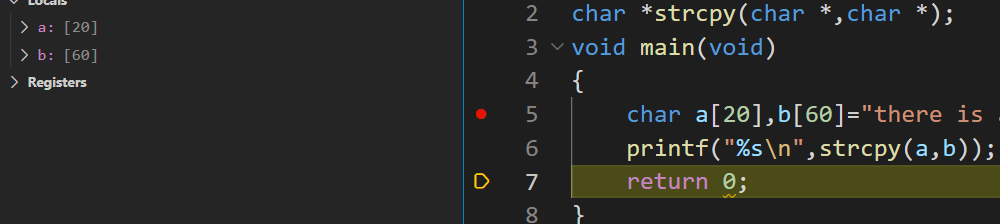
return (s);

}

（1）单步执行。进入strcpy时watch窗口中s为何值？返回main时, watch窗口中s为何值？



由上图容易知道，进入strcpy时watch窗口中s为一些随机序列组成的数组。



返回main时, watch窗口中s不能被发现，因为他是strcpy函数的形参，在外界无法访问。而由于s在strcpy函数内发生了移动，故直接返回s指针导致了程序异常。

（2）排除错误，使程序输出结果为：there is a boat on the lake.

**解答：**

错误修改如下：

1. 第5行，为a数组能容纳下b字符串的复制结果，正确形式如下：

char a[60],b[60]="there is a boat on the lake."

2) 第11行，为避免将主函数的指针指向的位置发生变化，应定义新的临时指针变量指向s字符串首地址，最后返回这个未改变的指针，故正确形式如下：

char \*p = s;

3) 第13行，为保证正确的输出字符串，应增加终止符，故正确形式如下：

\*s++ = '\0';

4) 第14行，为保证返回值正确，应返回指针p，故正确形式如下：

return (p);

程序运行结果如下：



**6.2.4 程序设计**

（1） 指定main函数的参数

在IDE（比如DevC++）中，选择“运行”｜“参数”菜单，在 “传递给主程序的参数”文本框中输入main函数的参数arg1 arg2 arg3，只输入命令行中文件名后的参数，文件名不作为参数输入，参数间以空格隔开。编写程序在命令行输出这三个参数。（注意不同IDE输入参数的方式不相同，可参考各个IDE的使用手册。）

**解答：**

1. 本实验学习指定main函数的参数。

由于main函数不能被其它函数调用，因此不可能在程序内部取得实际值。那么，在何处把实参值赋予main函数的形参呢？实际上，main函数的参数值是从操作系统命令行上获得的。当我们要运行一个可执行文件时，在DOS提示符下键入文件名，再输入实际参数即可把这些实参传送到main的形参中去。

DOS提示符下命令行的一般形式为：

C:\>可执行文件名 参数 参数 ……;

但是应该特别注意的是，main 的两个形参和命令行中的参数在位置上不是一一对应的。因为，main的形参只有二个，而命令行中的参数个数原则上未加限制。argc参数表示了命令行中参数的个数（注意：文件名本身也算一个参数），argc的值是在输入命令行时由系统按实际参数的个数自动赋予的。

gcc 编译器的常见用法

./a.out a.c b.c c.c

此时，我们知道：

argc ---> 4

argv[0] ---> a.out

argv[1] ---> a.c

argv[2] ---> b.c

argv[3] ---> c.c

我们在DevC++中设置参数较为容易。

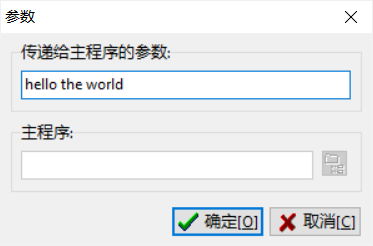


图6-1 编程题1的参数传递添加结果

1. 源程序清单

#include <stdio.h>

int main (int argc, char \*args[]) {

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

printf("%s\n", args[i]);

return 0;

}

3）测试

（a） 测试数据：

参数为Hello the world

（b） 对应测试数据的运行结果截图



图6-2 编程题1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。

（2）一个长整型变量占4个字节，其中每个字节又分成高4位和低4位。输入一个长整型变量，要求从高字节开始，依次取出每个字节的高4位和低4位并以十六进制数字字符的形式进行显示，通过指针取出每字节。

样例输入：15

样例输出：0000000F

**解答：**

1. 解题思路：

本题的关键在于对于C语言指针以及内存管理的理解。当我读到本题，心中一愣好像描述的不是非常清晰。

但是仔细分析。Longlong类型为四个字节，char类型为一个字节。用一个char\*的指针p来存放操作数的地址。p = p + 3，得到最高位，然后进行四次循环，便可得到计算地结果。

其算法流程图如下：

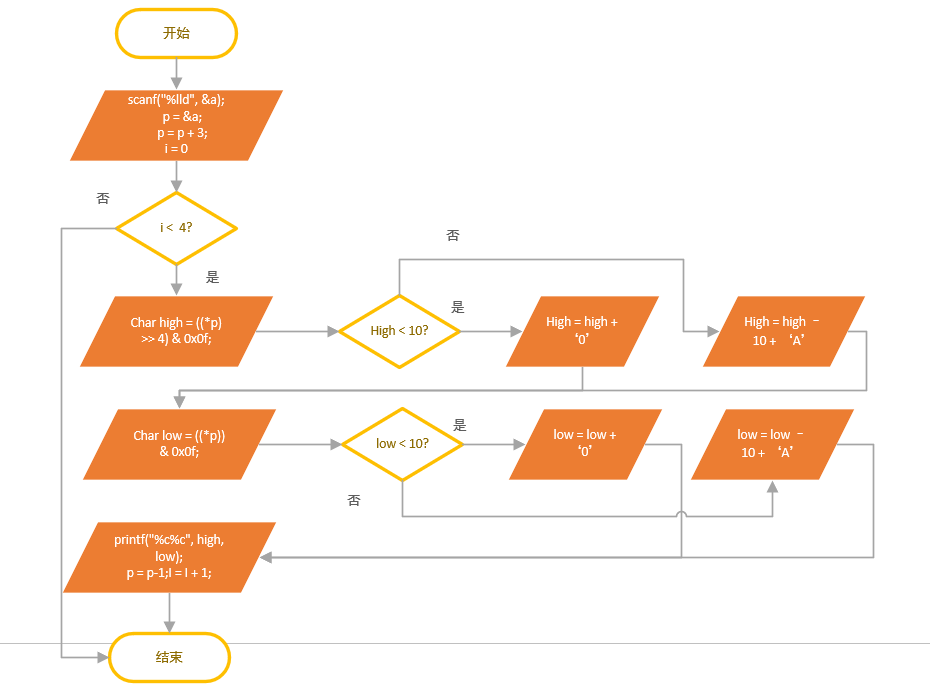


图6-3编程题2的算法流程示意图

2）程序清单：

#include <stdio.h>

int main () {

long long a;

char \*p;

scanf("%lld", &a);

p = &a;

p = p + 3;

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

char high = ((\*p) >> 4) & 0x0f;

if(high < 10)

high = high + '0';

else

high = high - 10 + 'A';

char low = (\*p) & 0x0f;

if(low < 10)

low = low + '0';

else

low = low - 10 + 'A';

printf("%c%c", high, low);

p = p-1;

}

return 0;

}

3）测试

（a） 测试数据：

15

2147483647

456456456

（b） 对应测试数据的运行结果截图







图6-4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）旋转是图像处理的基本操作，编程实现一个将一个图像逆时针旋转90°。提示：计算机中的图像可以用一个矩阵来表示，旋转一个图像就是旋转对应的矩阵。将旋转矩阵的功能定义成函数，通过使用指向数组元素的指针作为参数使该函数能处理任意大小的矩阵。要求在main函数中输入图像矩阵的行数n和列数m，接下来的n行每行输入m个整数，表示输入的图像。输出原始矩阵逆时针旋转90°后的矩阵。

**解答：**

1. 解题思路：

本题主要考察指针作为函数参数程序的编写。像数组、字符串、动态分配的内存等都是一系列数据的集合，没有办法通过一个参数全部传入函数内部，只能传递它们的指针，在函数内部通过指针来影响这些数据集合。

二维数组作为函数的参数，实参可以直接使用二维数组名，在被调用函数中可以定义形参所有维数的大小，也可以省略一维大小的说明。例如：

void find（char a[3][10]）;

void find (char a[ ][10]);

也可以使用数组指针来作为函数参数，例如：

void find （char (\*p)[10]);

但是不能像下面这样使用，例如：

void find（char a[ ][ ]）;

void find (char a[3][ ]);

而本题程序实现并不困难，可在函数中定义临时数组b用以存放逆时针旋转90°后的矩阵a，然后交换n，m。将临时数组b复制回a中。

而逆时针旋转九十度，即将a[i][j]放置于b[m - j + 1][i] 中（第一维和第二维均从1开始编号）。

2）程序清单

#include <stdio.h>

void RotateLeft90(int (\*a)[200], int\* n, int\* m) {

int b[200][200];

for(int i = 1; i <= \*n; i++)

for(int j = 1; j <= \*m; j++)

b[\*m - j + 1][i] = a[i][j];

int temp = \*n;

\*n = \*m;

\*m = temp;

for(int i = 1; i <= \*n; i++)

for(int j = 1; j <= \*m; j++)

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

}

int main () {

int n, m;int a[200][200];

scanf("%d%d", &n, &m);

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

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

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

RotateLeft90(a, &n, &m);

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

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

printf("%d%s",a[i][j],j == m? "\n":" ");

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6-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N和带旋转的矩阵 |
| 例1 | 1 1  5 | 5 | 5 |
| 例2 | 2 3  1 5 3  3 2 4 | 3 4  5 2  1 3 | 3 4  5 2  1 3 |
| 例3 | 10 10  38 7719 21238 2437 8855 11797 8365 32285 10450 30612  5853 28100 1142 281 20537 15921 8945 26285 2997 14680  20976 31891 21655 25906 18457 1323 28881 2240 9725 32278  2446 590 840 18587 16907 21237 23611 12617 12456 867  29533 6878 28223 17887 31597 20584 12212 31111 7578 17066  7629 29404 12279 13505 24388 11649 12329 7176 2331 19264  22114 14136 26928 1102 21652 8404 24337 27856 5598 24772  14097 13213 4683 16703 15260 15942 2747 27375 28871 18004  16673 3152 11819 23504 239 4186 2804 28937 3023 10335  20533 21393 16020 11574 25983 13961 624 7065 27569 12830 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 |

（b） 对应测试测试用例1的运行结果如图6-5所示。



图6-5 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6-6所示。

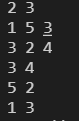


图6-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-7所示。

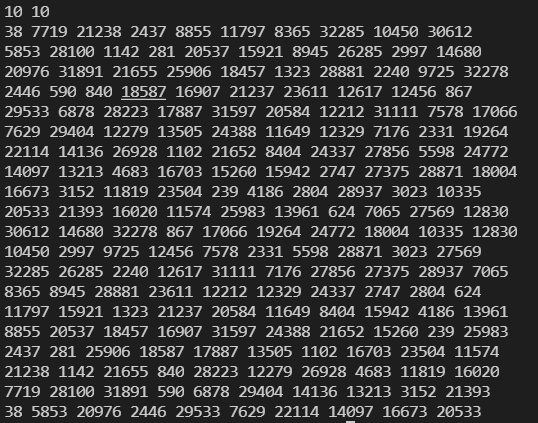


图6-7编程题3的测试用例三的运行结果

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）旋转是图像处理的基本操作，编程实现一个将一个图像逆时针旋转90°。提示：计算机中的图像可以用一个矩阵来表示，旋转一个图像就是旋转对应的矩阵。将旋转矩阵的功能定义成函数，通过使用指向数组元素的指针作为参数使该函数能处理任意大小的矩阵。要求在main函数中输入图像矩阵的行数n和列数m，接下来的n行每行输入m个整数，表示输入的图像。输出原始矩阵逆时针旋转90°后的矩阵。

**解答：**

1. 解题思路：

本题主要考察指针作为函数参数程序的编写。像数组、字符串、动态分配的内存等都是一系列数据的集合，没有办法通过一个参数全部传入函数内部，只能传递它们的指针，在函数内部通过指针来影响这些数据集合。

二维数组作为函数的参数，实参可以直接使用二维数组名，在被调用函数中可以定义形参所有维数的大小，也可以省略一维大小的说明。例如：

void find（char a[3][10]）;

void find (char a[ ][10]);

也可以使用数组指针来作为函数参数，例如：

void find （char (\*p)[10]);

但是不能像下面这样使用，例如：

void find（char a[ ][ ]）;

void find (char a[3][ ]);

而本题程序实现并不困难，可在函数中定义临时数组b用以存放逆时针旋转90°后的矩阵a，然后交换n，m。将临时数组b复制回a中。

而逆时针旋转九十度，即将a[i][j]放置于b[m - j + 1][i] 中（第一维和第二维均从1开始编号）。

2）程序清单

#include <stdio.h>

void RotateLeft90(int (\*a)[200], int\* n, int\* m) {

int b[200][200];

for(int i = 1; i <= \*n; i++)

for(int j = 1; j <= \*m; j++)

b[\*m - j + 1][i] = a[i][j];

int temp = \*n;

\*n = \*m;

\*m = temp;

for(int i = 1; i <= \*n; i++)

for(int j = 1; j <= \*m; j++)

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

}

int main () {

int n, m;int a[200][200];

scanf("%d%d", &n, &m);

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

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

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

RotateLeft90(a, &n, &m);

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

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

printf("%d%s",a[i][j],j == m? "\n":" ");

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6-2所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N和待旋转的矩阵 |
| 例1 | 1 1  5 | 5 | 5 |
| 例2 | 2 3  1 5 3  3 2 4 | 3 4  5 2  1 3 | 3 4  5 2  1 3 |
| 例3 | 10 10  38 7719 21238 2437 8855 11797 8365 32285 10450 30612  5853 28100 1142 281 20537 15921 8945 26285 2997 14680  20976 31891 21655 25906 18457 1323 28881 2240 9725 32278  2446 590 840 18587 16907 21237 23611 12617 12456 867  29533 6878 28223 17887 31597 20584 12212 31111 7578 17066  7629 29404 12279 13505 24388 11649 12329 7176 2331 19264  22114 14136 26928 1102 21652 8404 24337 27856 5598 24772  14097 13213 4683 16703 15260 15942 2747 27375 28871 18004  16673 3152 11819 23504 239 4186 2804 28937 3023 10335  20533 21393 16020 11574 25983 13961 624 7065 27569 12830 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 |

（b） 对应测试测试用例1的运行结果如图6-8所示。



图6-8 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6-9所示。

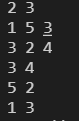


图6-9 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-10所示。

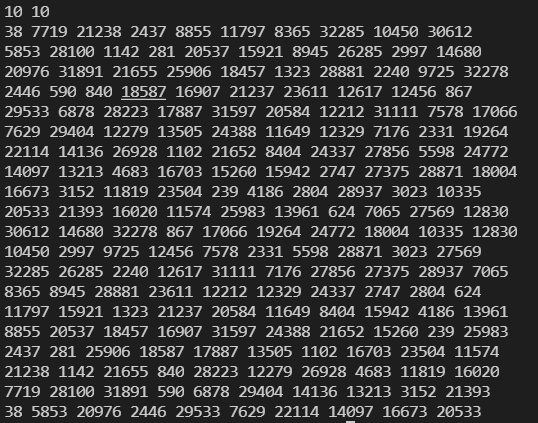


图6-10编程题3的测试用例三的运行结果

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。

（4）输入n行文本，每行不超过80个字符，用字符指针数组指向键盘输入的n行文本（n不作为输入，可理解为循环输入多行，以CTRL+Z结束），删除每一行中的前置空格（' '）和水平制表符（'\t'）。

要求：将删除一行文本中前置空格和水平制表符的功能定义成函数，在main函数中输出删除前置空格符的各行。（并在最后输出换行符。）

**解答：**

1. 解题思路：分析题目，本题的算法流程图可如下表示。

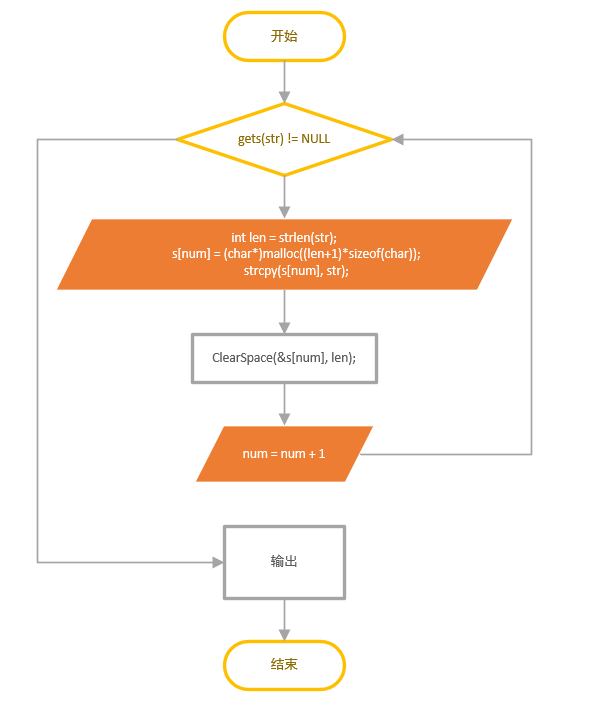


图6-11 编程题4的算法流程示意图

而核心函数ClearSpace(char \*\*a, int len)的算法流程也如下

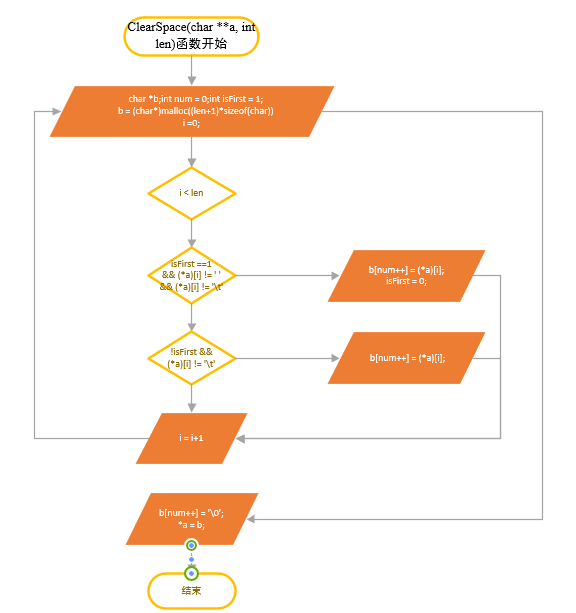


图6-12 编程题4的ClearSpace(char \*\*a, int len)函数的算法流程示意图

我们在此处意识到为了改变字符指针数组中某个元素s[i]指向的内容，必须用指针指向它，所以函数声明时采用了ClearSpace(char \*\*a, int len)的声明方式，调用时s[i]要取地址。函数内部使用数组时也要按照二维字符指针进行去单个字符。

我们也应该注意到为了判断是否开头，我们应该设置标记判断其是否为开头，若为开头，则要把空格和制表符全部删去，并把标记清空；否则，只需要删除制表符即可。

2）程序清单

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void ClearSpace(char \*\*a, int len) {

char \*b;int num = 0;int isFirst = 1;

b = (char\*)malloc((len+1)\*sizeof(char));

for(int i = 0; i < len; i++) {

if(isFirst && (\*a)[i] != ' ' && (\*a)[i] != '\t') {

b[num++] = (\*a)[i];

isFirst = 0;

}

else if(!isFirst && (\*a)[i] != '\t')

b[num++] = (\*a)[i];

}

b[num++] = '\0';

\*a = b;

}

int main () {

int num = 0;

char \*s[1000];

while(1) {

char str[100];

if(gets(str) == NULL)

break;

int len = strlen(str);

s[num] = (char\*)malloc((len+1)\*sizeof(char));

strcpy(s[num], str);

ClearSpace(&s[num], len);

num++;

}

for(int i = 0; i < num; i++)

printf("%s\n",s[i]);

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6-3所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入文本信息 |
| 例1 | 123 45  67 89  Ctrl+Z | 12345  6789 | 12345  6789 |
| 例2 | aab c  bcd e  Ctrl+Z | aab c  bcde | aab c  bcde |
| 例3 | Ab c d  Lover | Abc d  Lover | Abc d  Lover |

（b） 对应测试测试用例1的运行结果如图6-13所示。

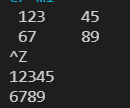


图6-13 编程题4的测试用例一的运行结果

对应测试测试用例2的运行结果如图6-14所示。

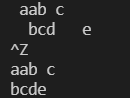


图6-14 编程题4的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-15所示。

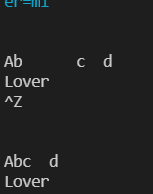


图6-15编程题4的测试用例三的运行结果

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。

（5）编写8个任务函数，一个scheduler调度函数和一个execute执行函数。仅在main函数中调用scheduler函数，scheduler函数要求用最快的方式调度执行用户指定的任务函数。

①先设计task0, task1, task2, task3, task4, task5, task6, task7共8个任务函数，每个任务函数的任务就是输出该任务被调用的字符串。例如，第0个任务函数输出“task0 is called!”，第1个任务函数输出“task1 is called!”，以此类推。

②scheduler函数根据键盘输入的数字字符的先后顺序，一次调度选择对应的任务函数。例如，输入：1350并回车，则scheduler函数一次调度选择task1, task3, taks5, task0，然后以函数指针数组和任务个数为参数将调度选择结果传递给execute函数并调用execute函数。

③execute函数根据scheduler函数传递的指针数组和任务个数为参数，按照指定的先后顺序依此调用执行选定的任务函数。

当输入13607122并回车，程序运行结果应当如下：

task1 is called!

task3 is called!

task6 is called!

task0 is called!

task7 is called!

task1 is called!

task2 is called!

task2 is called!

**解答：**

1. 解题思路：

本题的难点在于函数指针数组和函数指针的使用。且在函数中来回传递这些函数指针与数组。

可在scheduler函数中初始化一个用于分配人物的函数指针数组，其中放好了写的9个函数。void (\*p[]) () = {task0, task1, task2, task3, task4, task5, task6, task7, task8, task9};这样便可以避免使用switch的复杂，直接遍历字符数组得到需要的函数，压入work函数指针数组。

将work函数指针数组和总任务数n传递至execute函数，便可轻松地实现函数的调用。

2）程序清单

#include <stdio.h>

void task0 () { printf("task0 is called!\n");}

void task1 () { printf("task1 is called!\n");}

void task2 () { printf("task2 is called!\n");}

void task3 () { printf("task3 is called!\n");}

void task4 () { printf("task4 is called!\n");}

void task5 () { printf("task5 is called!\n");}

void task6 () { printf("task6 is called!\n");}

void task7 () { printf("task7 is called!\n");}

void task8 () { printf("task8 is called!\n");}

void task9 () { printf("task9 is called!\n");}

void execute(void (\*p[])(), int n) {

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

p[i]();

}

void scheduler() {

void (\*p[]) () = {task0, task1, task2, task3, task4, task5, task6, task7, task8, task9};

char a[1000];

void (\*work[1000]) ();

scanf("%s",a);

int n = 0;

for(n = 0; a[n] != '\0'; n++)

work[n] = p[a[n]-'0'];

execute(work, n);

}

int main () {

scheduler();

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6-4所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入序列 |
| 例1 | 13607122 | task1 is called!  task3 is called!  task6 is called!  task0 is called!  task7 is called!  task1 is called!  task2 is called!  task2 is called! | task1 is called!  task3 is called!  task6 is called!  task0 is called!  task7 is called!  task1 is called!  task2 is called!  task2 is called! |
| 例2 | 123450019941560198 | task1 is called!  task2 is called!  task3 is called!  task4 is called!  task5 is called!  task0 is called!  task0 is called!  task1 is called!  task9 is called!  task9 is called!  task4 is called!  task1 is called!  task5 is called!  task6 is called!  task0 is called!  task1 is called!  task9 is called!  task8 is called! | task1 is called!  task2 is called!  task3 is called!  task4 is called!  task5 is called!  task0 is called!  task0 is called!  task1 is called!  task9 is called!  task9 is called!  task4 is called!  task1 is called!  task5 is called!  task6 is called!  task0 is called!  task1 is called!  task9 is called!  task8 is called! |
| 例3 | 5210 | task5 is called!  task2 is called!  task1 is called!  task0 is called! | task5 is called!  task2 is called!  task1 is called!  task0 is called! |

（b） 对应测试测试用例1的运行结果如图6-16所示。

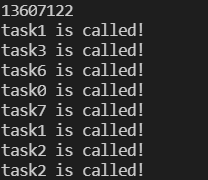


图6-16 编程题5的测试用例一的运行结果

对应测试测试用例2的运行结果如图6-17所示。

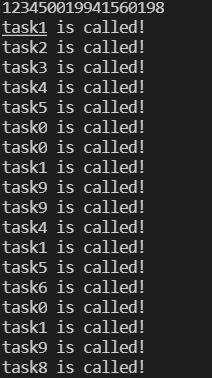


图6-17 编程题5的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-18所示。

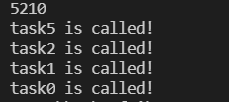


图6-18编程题5的测试用例三的运行结果

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。

## 6.3 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。

通过本次实验，我觉得我对C语言地重难点——指针有了更深刻的理解。熟练地掌握指针的说明、赋值、使用。通过源程序修改替换题（1）掌握了用指针引用数组的元素，熟悉指向数组的指针的使用。也在众多习题中对如何用函数传递数组，修改指针数组指向地址，这些都是非常有用的小技巧。而我也初次解除了函数指针，函数指针数组这一比较奇怪的东西，并在程序设计题和替换修改题（2）中亲手实现了他们并且理会到了这种方式可以起到分配任务，省去分支处理的妙用。我也在程序设计题（1）中第一次打开了DEVC++的参数工具，使用它这么多年来，就像老朋友突然发现了新的特长，十分亲切。

最让我印象深刻的是程序设计题（2），我对题意始终不了解，在同学的点拨下，我终于有所领悟，也更加深入的理解了指针的机制与丰富

# 7 结构与联合实验

## 7.1 实验目的

（1）通过实验，熟悉和掌握结构的说明和引用、结构的指针、结构数组、以及函数中使用结构的方法。

（2）通过实验，掌握动态储存分配函数的用法，掌握自引用结构，单向链表的创建、遍历、结点的增删、查找等操作。

（3）了解字段结构和联合的用法。

## 7.2 实验内容

**7.2.1 表达式求值的程序验证题**

设有说明：

char u[]="UVWXYZ";

char v[]="xyz";

struct T{

int x;

char c;

char \*t;

}a[]={{11,ˊAˊ,u},{100, ˊBˊ,v}},\*p=a;

请先自己计算下面表达式的值，然后通过编程计算来加以验证。

(各表达式相互无关)

**解答：**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **表达式** | **计算值** | **验证值** |
| 1 | (++p)->x | 100 | 100 |
| 2 | p++,p->c | B | B |
| 3 | \*p++->t,\*p->t | x | x |
| 4 | \*(++p)->t | x | x |
| 5 | \*++p->t | V | V |
| 6 | ++\*p->t | V | V |

1) 先计算++p，且表达式的值为a[1]，取其x值，便为100。

2) 逗号表达式，先计算前面的表达式，p指向a[1]，但逗号表达式的值为最后一个表达式的值，表达式2的值因此为B。

3) 逗号表达式，先计算前面的表达式，p指向a[1]，由于成员选择运算符（指针）->的优先级最高，然后取出字符数组v的首元素。表达式3的值因此为x。

4) 先计算++p，且表达式的值为a[1]，由于成员选择运算符（指针）->的优先级最高，取其t指针指向的首元素，表达式4的值因此为x。

5) 由于成员选择运算符（指针）->的优先级最高，取其t指针（指向数组u）。自增运算符++和取值运算符\*优先级相同，但从右向左结合，先是指针指向u[1]。表达式5的值因此为V。

6) 由于成员选择运算符（指针）->的优先级最高，取其t指针（指向数组u）。自增运算符++和取值运算符\*优先级相同，但从右向左结合，先是取值u[0]，然后自加。表达式6的值因此为V。

**7.2.2 源程序修改替换题**

给定一批整数，以0作为结束标志且不作为结点，将其建成一个先进先出的链表，先进先出链表的指头指针始终指向最先创建的结点（链头），先建结点指向后建结点，后建结点始终是尾结点。

#include "stdio.h"

#include "stdlib.h"

struct s\_list{

int data; /\* 数据域 \*/

struct s\_list \*next; /\* 指针域 \*/

} ;

void create\_list (struct s\_list \*headp,int \*p);

void main(void)

{

struct s\_list \*head=NULL,\*p;

int s[]={1,2,3,4,5,6,7,8,0}; /\* 0为结束标记 \*/

create\_list(head,s); /\* 创建新链表 \*/

p=head; /\*遍历指针p指向链头 \*/

while(p){

printf("%d\t",p->data); /\* 输出数据域的值 \*/

p=p->next; /\*遍历指针p指向下一结点 \*/

}

printf("\n");

}

void create\_list(struct s\_list \*headp,int \*p)

{

struct s\_list \* loc\_head=NULL,\*tail;

if(p[0]==0) /\* 相当于\*p==0 \*/

;

else { /\* loc\_head指向动态分配的第一个结点 \*/

loc\_head=(struct s\_list \*)malloc(sizeof(struct s\_list));

loc\_head->data=\*p++; /\* 对数据域赋值 \*/

tail=loc\_head; /\* tail指向第一个结点 \*/

while(\*p){ /\* tail所指结点的指针域指向动态创建的结点 \*/

tail->next=(struct s\_list \*)malloc(sizeof(struct s\_list));

tail=tail->next; /\* tail指向新创建的结点 \*/

tail->data=\*p++; /\* 向新创建的结点的数据域赋值 \*/

}

tail->next=NULL; /\* 对指针域赋NULL值 \*/

}

headp=loc\_head; /\* 使头指针headp指向新创建的链表 \*/

}

（1）源程序中存在什么样的错误（先观察执行结果）？对程序进行修改、调试，使之能够正确完成指定任务。

**解答：**

1. 错误修改：

通过运行程序发现，程序并未有输出的值。通过逐步调试发现，经过函数create\_list后，main函数内指针并未发生改变，指向位置仍然是空。具体原因是因为，void create\_list(struct s\_list \*headp,int \*p)函数是值传递，整个函数内的headp指针为局部变量，并未影响到main函数内的指针。解决方案是再取指针指向这个指向链表的指针，通过传递指针达到修改值的目的。

1) 第8行为保证编译正确，正确形式为：

int main()//void main(void)

2) 第7行为了能成功修改main函数内的需要再取指针，正确形式为：

void create\_list (struct s\_list \*\*headp,int \*p);

3) 第20行同2，正确形式为：

void create\_list(struct s\_list \*\*headp,int \*p)

4) 第36行为使头指针指向新创建的链表，正确形式为：

\*headp=loc\_head;

（2）错误修改后运行结果：



（2）修改替换create\_list函数，将其建成一个后进先出的链表，后进先出链表的头指针始终指向最后创建的结点（链头），后建结点指向先建结点，先建结点始终是尾结点。

**解答：**

整体代码框架基本保持不变，而修改为后进先出的链表仅需要修改创建链表的过程，其算法流程图如下：

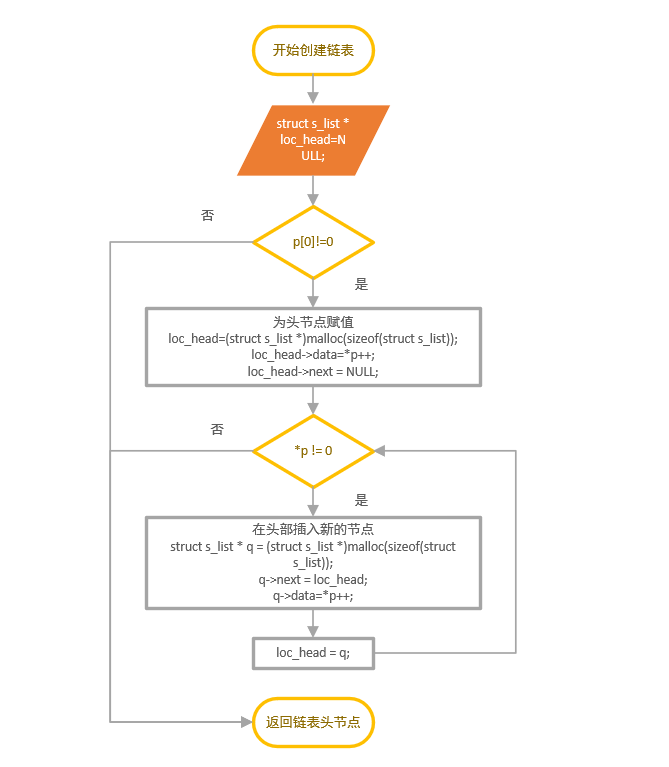


图7-1 源程序修改替换题（2）的算法流程示意图

代码实现如下：

void create\_list(struct s\_list \*\*headp,int \*p)

{

struct s\_list \* loc\_head=NULL;

if(p[0]==0) /\* 相当于\*p==0 \*/

;

else { /\* loc\_head指向动态分配的第一个结点 \*/

loc\_head=(struct s\_list \*)malloc(sizeof(struct s\_list));

loc\_head->data=\*p++; /\* 对数据域赋值 \*/

loc\_head->next = NULL;

while(\*p){ /\* tail所指结点的指针域指向动态创建的结点 \*/

struct s\_list \* q = (struct s\_list \*)malloc(sizeof(struct s\_list));

q->next = loc\_head; /\* tail指向新创建的结点 \*/

q->data=\*p++; /\* 向新创建的结点的数据域赋值 \*/

loc\_head = q;

}

}

\*headp=loc\_head; /\* 使头指针headp指向新创建的链表 \*/

}

**7.2.3 程序设计**

（1） 用单向链表建立一张班级成绩单，包括每个学生的学号、姓名、英语、高等数学、普通物理、C语言程序设计四门课程的成绩。用菜单实现下列功能：

① 输入每个学生的各项信息。

② 输出每个学生的各项信息。

③ 修改指定学生的指定数据项的内容。

④ 统计每个同学的平均成绩（保留2位小数）。

⑤ 输出各位同学的学号、姓名、四门课程的总成绩和平均成绩。

**解答：**

1. 解题思路：

本题分析和建模后应建立如下的链表节点域：

typedef struct Node {

int id; char Name[1000]; int a, b, c, d, sum;double ave;

struct Node \*next;

}Node;本题分析与建模后应完成如下函数：

1. Input函数：输入学生数量n，完成每个学生的学号、姓名、英语、高等数学、普通物理、C语言程序设计四门课程的成绩的输入和保存至链表。
2. Output函数：遍历这个单向链表输出所有学生的各项信息。
3. Change函数：根据输入的学生的学号，和一个数字，0对应修改其名字，1到4对应修改其四门课的成绩。遍历单向链表，找到可以修改的地方然后修改。
4. CalcAve函数：遍历单向链表，统计每个同学的平均成绩保存并输出。
5. OutAve函数：遍历这个单向链表输出所有学生的总成绩和平均成绩。

为了保证能够修改头指针指向的链表，我们应该设立头节点，即头指针指向的第一个节点不记录任何信息，同时也便于插入节点程序的编写。

1. 源程序清单

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

int id; char Name[1000]; int a, b, c, d, sum;double ave;

struct Node \*next;

}Node;

void input(Node \*S) {

int N;

scanf("%d", &N);

struct Node \*p = S;

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

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

scanf("%d%s%d%d%d%d", &newNode->id, newNode->Name, &newNode->a, &newNode->b, &newNode->c, &newNode->d);

p->next = newNode;

newNode->next = NULL;

p = newNode;

}

printf("完成了%d位同学的成绩输入。\n", N);

}

void output(Node \*S) {

struct Node \*p = S->next;

while(p != NULL) {

printf("%d %s %d %d %d %d\n", p->id, p->Name, p->a, p->b, p->c, p->d);

p = p->next;

}

}

void Change(Node \*S) {

int id, waitChange;

scanf("%d%d",&id, &waitChange);

struct Node \*p = S->next;

while(p != NULL) {

if(p->id == id) {

if(waitChange == 0) {

char newName[100];

scanf("%s",newName);

memcpy(p->Name, newName, sizeof(newName));

}

else if(waitChange == 1) {

int A;scanf("%d", &A);p->a = A;

}

else if(waitChange == 2) {

int B;scanf("%d", &B);p->b = B;

}

else if(waitChange == 3) {

int C;scanf("%d", &C);p->c = C;

}

else if(waitChange == 4) {

int D;scanf("%d", &D);p->d = D;

}

p->sum = p->a + p->b + p->c + p->d;

p->ave = p->sum/4.0;

printf("%d %s %d %d %d %d\n", p->id, p->Name, p->a, p->b, p->c, p->d);

break;

}

p = p->next;

}

}

void CalcAve(Node \*S) {

struct Node \*p = S->next;

while(p != NULL) {

int sum = 0;

sum = p->a + p->b + p->c + p->d;

p->sum = sum;

p->ave = sum/4.0;

printf("%d %s %0.2lf\n", p->id, p->Name, sum/4.0);

p = p->next;

}

}

void OutAll (Node \*S) {

struct Node \*p = S->next;

while(p != NULL) {

printf("%d %s %d %0.2lf\n", p->id, p->Name, p->sum, p->ave);

p = p->next;

}

}

int main () {

struct Node\* S = (struct Node\*) malloc(sizeof(struct Node));

S->id = S->a = S->b = S->c = S->d = 0; (S->Name)[0] = 0;

S->next = NULL;

int opt;

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

if(opt == 0) {

//printf("（退出）\n");

break;

}

else if(opt == 1)

input(S);

else if(opt == 2)

output(S);

else if(opt == 3)

Change(S);

else if(opt == 4)

CalcAve(S);

else if(opt == 5)

OutAll(S);

}

}

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7-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 操作与信息 |
| 例1 | 1  5  2021001 Jack 90 92 87 95  2021002 Mike 85 70 75 90  2021003 Joe 77 86 90 75  2021004 Andy 95 97 92 95  2021005 Rose 90 87 88 89  0 | 完成了5位同学的成绩输入。 | 完成了5位同学的成绩输入。 |
| 例2 | 1  5  2021001 Jack 90 92 87 95  2021002 Mike 85 70 75 90  2021003 Joe 77 86 90 75  2021004 Andy 95 97 92 95  2021005 Rose 90 87 88 89  3  2021003 0 Joan  3  2021003 2 96  4  5  0 | 完成了5位同学的成绩输入。  2021003 Joan 77 86 90 75  2021003 Joan 77 96 90 75  2021001 Jack 91.00  2021002 Mike 80.00  2021003 Joan 84.50  2021004 Andy 94.75  2021005 Rose 88.50  2021001 Jack 364 91.00  2021002 Mike 320 80.00  2021003 Joan 338 84.50  2021004 Andy 379 94.75  2021005 Rose 354 88.50 | 完成了5位同学的成绩输入。  2021003 Joan 77 86 90 75  2021003 Joan 77 96 90 75  2021001 Jack 91.00  2021002 Mike 80.00  2021003 Joan 84.50  2021004 Andy 94.75  2021005 Rose 88.50  2021001 Jack 364 91.00  2021002 Mike 320 80.00  2021003 Joan 338 84.50  2021004 Andy 379 94.75  2021005 Rose 354 88.50 |

（b） 对应测试测试用例1的运行结果如图7-2所示。

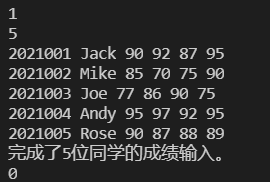


图7-2 编程题1的测试用例一的运行结果

对应测试测试用例2的运行结果如图7-3所示。

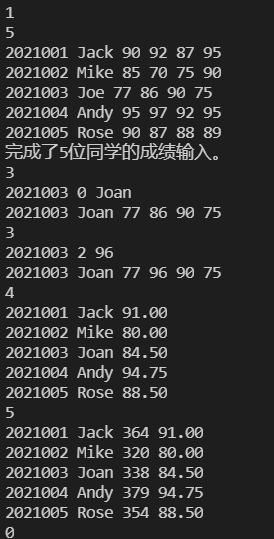


图7-3 编程题1的测试用例二的运行结果

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）对程序设计题第（1）题的程序，

⑥增加按照平均成绩进行升序排序的函数，写出用交换结点数据域的方法升序排序的函数，排序可用选择法或冒泡法。

**解答：**

1）解题思路：

冒泡排序的中心思想：通过对无序序列的相邻元素两两比较交换，每次把最大（小）的数据放在最后，这个过程就像是冒泡一样，所以叫他冒泡排序。交换结点数据域的方法升序排序的函数，即与冒泡排序写法没有大的区别，遍历方式发生了改变，判断是否排序完的判断也发生了改变。而由于链表没有记录存储的学生信息数量如下：

2）部分函数清单：

void swap1 (int \*a,int \*b) {

int temp = \*a; \*a = \*b; \*b = temp;

}

void swap2 (double \*a,double \*b) {

double temp = \*a; \*a = \*b; \*b = temp;

}

void swap3 (char \*a,char \*b) {

char c[1000];

memcpy(c, a, sizeof(a));

memcpy(a, b, sizeof(b));

memcpy(b, c, sizeof(c));

}

void Sort1 (Node \*S) {

for(Node \*q = S->next; q != NULL; q = q->next) {

for(Node \*p = S->next; p->next != NULL; p = p->next) {

if((p->sum) > (p->next->sum)) {

swap1(&p->id, &p->next->id);

swap3(p->Name, p->next->Name);

swap1(&p->a, &p->next->a);

swap1(&p->b, &p->next->b);

swap1(&p->c, &p->next->c);

swap1(&p->d, &p->next->d);

swap1(&p->sum, &p->next->sum);

swap2(&p->ave, &p->next->ave);

}

}

}

Node \*p = S->next;

while(p != NULL) {

printf("%d %s %0.2lf\n", p->id, p->Name, p->ave);

p = p->next;

}

}

3）测试

（a） 测试数据：

1

5

2021001 Jack 90 92 87 95

2021002 Mike 85 70 75 90

2021003 Joe 77 86 90 75

2021004 Andy 95 97 92 95

2021005 Rose 90 87 88 89

3

2021003 0 Joan

3

2021003 2 96

4

5

6

0

（b） 对应测试数据的运行结果截图

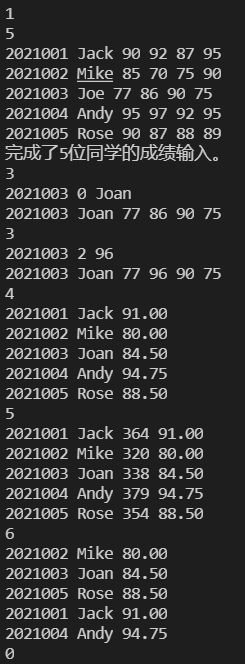


图7-4编程题2的测试运行结果

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）对程序设计题第（2）题，进一步写出用交换结点指针域的方法升序排序的函数。

**解答：**

1. 解题思路：

分析排序算法后，可以建立流程图如下：

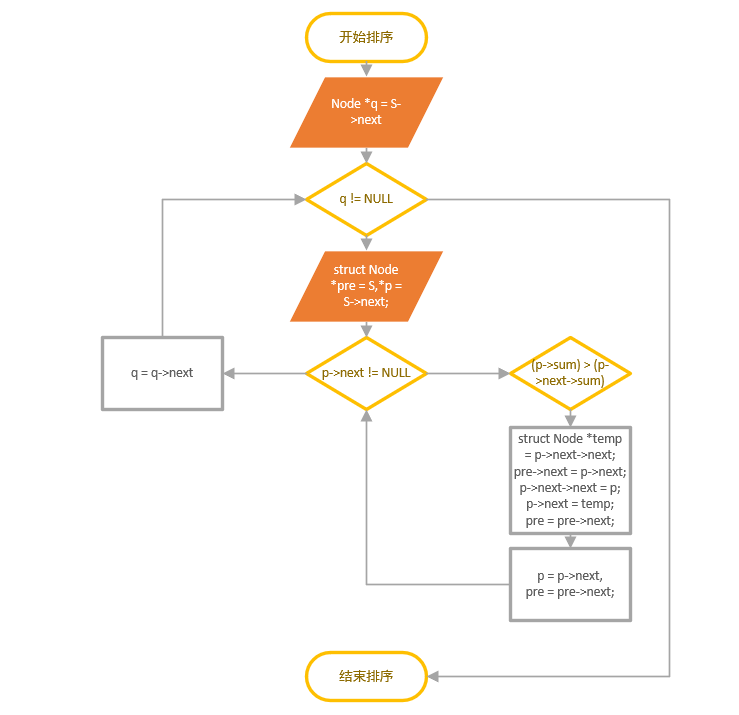


图7-5编程题3排序部分的算法流程图

2）程序清单

void Sort2(Node \*S) {

for(Node \*q = S->next; q != NULL; q = q->next) {

struct Node \*pre = S,\*p = S->next;

while(p->next != NULL) {

if((p->sum) > (p->next->sum)) {

struct Node \*temp = p->next->next;

pre->next = p->next;

p->next->next = p;

p->next = temp;

pre = pre->next;

continue;

}

p = p->next, pre = pre->next;

}

}

struct Node \*p = S->next;

while(p != NULL) {

printf("%d %s %0.2lf\n", p->id, p->Name, p->ave);

p = p->next;

}

}

3）测试

（a） 测试数据：

1

5

2021001 Jack 90 92 87 95

2021002 Mike 85 70 75 90

2021003 Joe 77 86 90 75

2021004 Andy 95 97 92 95

2021005 Rose 90 87 88 89

3

2021003 0 Joan

3

2021003 2 96

4

5

6

0

（b） 对应测试测试用例1的运行结果如图7-6所示。

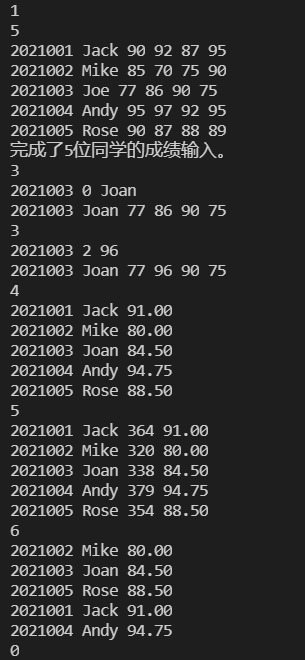


图7-6 编程题3的测试用例的运行结果

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。

## 7.3 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。

通过本次实验，我较深入地理解了结构体和联合的实际写法和在具体问题中的应用。编写链表程序加深了对数据结构相关知识的理解。在表达式求值的程序验证题中，我将原先指针等知识整合，分析了他们的运算优先级，为编写正确的程序做好了准备。我也学会了链表交换节点的两种方式，节点值交换，节点来回倒。

# 8 文件操作实验

## 8.1 实验目的

（1）熟悉文本文件和二进制文件在磁盘中的存储方式；

（2）熟练掌握流式文件的读写方法。

## 8.2 实验内容

**8.2.1 文件类型的程序验证题**

设有程序：

#include <stdio.h>

int main(void)

{

short a=0x253f,b=0x7b7d;

char ch;

FILE \*fp1,\*fp2;

fp1=fopen("d:\\abc1.bin","wb+");

fp2=fopen("d:\\abc2.txt","w+");

fwrite(&a,sizeof(short),1,fp1);

fwrite(&b,sizeof(short),1,fp1);

fprintf(fp2,"%hx %hx",a,b);

rewind(fp1); rewind(fp2);

while((ch = fgetc(fp1)) != EOF)

putchar(ch);

putchar('\n');

while((ch = fgetc(fp2)) != EOF)

putchar(ch);

putchar('\n');

fclose(fp1);

fclose(fp2);

return 0;

}

（1）请思考程序的输出结果，然后通过上机运行来加以验证。

**解答：**

文件类型Fp1按照二进制位输出，低字节在先，读入时按照字节读入，按照ASCII码对应3f -> ? 25->% 7d->} 7b->{，其输出结果应该为?%}{

而文件类型fp2按照字符读写，输出时直接输出其十六进制数的字符串，读入时读入字符串并输出，其结果应该为253f 7b7d。

测试结果如下：



（2）将两处sizeof(short)均改为sizeof(char)结果有什么不同，为什么？

**解答：**

文件类型Fp1按照二进制位输出，低字节在先，但是short是两个字节，char是一个字节，其故只输出了低字节。读入时按照字节读入，按照ASCII码对应3f -> ? 7d->}，其输出结果应该为?}

测试结果如下：



（3）将fprintf(fp2,"%hx %hx",a,b) 改为 fprintf(fp2,"%d %d",a,b)结果有什么不同。

**解答：**

文件类型fp2按照字符读写，输出时直接输出其十进制数的字符串，读入时读入字符串并输出，其结果应该为9535 31613。



**8.2.2 源程序修改替换题**

将指定的文本文件内容在屏幕上显示出来，命令行的格式为：type filename

#include<stdio.h>

#include<stdlib.h>

int main(int argc, char\* argv[])

{

char ch;

FILE \*fp;

if(argc!=2){

printf("Arguments error!\n");

exit(-1);

}

if((fp=fopen(argv[1],"r"))==NULL){ /\* fp 指向 filename \*/

printf("Can't open %s file!\n",argv[1]);

exit(-1);

}

while(ch=fgetc(fp)!=EOF) /\* 从filename中读字符 \*/

putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

return 0;

}

（1）源程序中存在什么样的错误（先观察执行结果）？对程序进行修改、调试，使之能够正确完成指定任务。

**解答：**

1. 错误修改：

通过运行程序发现，程序输出乱码空格字符串。逐步调试发现，ch结果永远为1.故发现while括号内语句存在问题。

1) 第16行因为!=运算优先级高于赋值运算，若ch始终只能是布尔值0或1，正确形式为：

while((ch=fgetc(fp))!=EOF)

1. 错误修改后运行结果：

参数设置为本程序”2.c”。

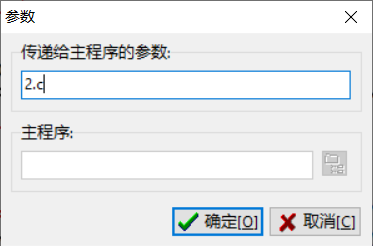


图8-1 源程序修改替换题的参数

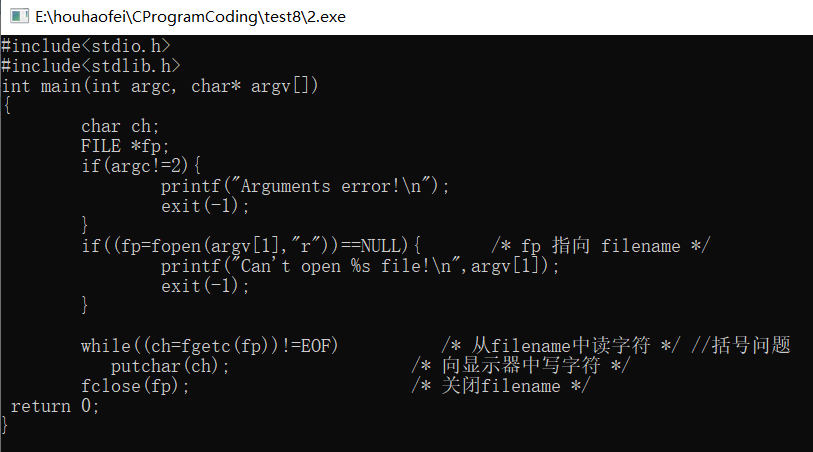


图8-2 源程序修改替换题的运行结果

（2）用输入输出重定向freopen改写main函数。

**解答：**

整体代码框架基本保持不变。

freopen()函数用于文件流的的重定向，一般是将 stdin、stdout 和 stderr 重定向到文件。

所谓重定向，就是改变文件流的源头或目的地。stdout（标准输出流）的目的地是显示器，printf()是将流中的内容输出到显示器；可以通过freopen()将stdout 的目的地改为一个文件（如output.txt），再调用 printf()，就会将内容输出到这个文件里面，而不是显示器。

freopen()函数的原型为：

FILE \*freopen(char \*filename, char \*type, FILE \*stream);

filename为要重定向到的文件名；type为文件打开方式， stream为被重定向的文件流（一般是 stdin、stdout 或 stderr）。成功则返回指向filename文件的指针，否则返回NULL。

代码实现如下：

#include<stdio.h>

#include<stdlib.h>

int main(int argc, char\* argv[])

{

char ch;

if(argc!=2){

printf("Arguments error!\n");

exit(-1);

}

if(freopen(argv[1],"r",stdin) == NULL) { /\* fp 指向 filename \*/

printf("Can't open %s file!\n",argv[1]);

exit(-1);

}

while(scanf("%c", &ch)!=EOF) /\* 从filename中读字符 \*/ //括号问题

putchar(ch); /\* 向显示器中写字符 \*/

fclose(stdout); /\* 关闭filename \*/

return 0;

}

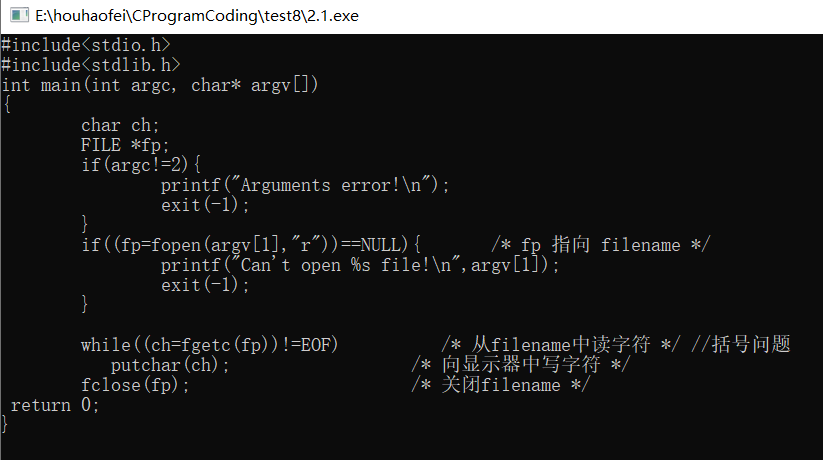


图8-3 源程序修改替换题（2）的运行结果

**8.2.3 程序设计**

（1） 本关任务：编写一个程序，用给定的字符串替换文件中的目标字符串，并显示输出替换的个数。

注意：读取的文件路径请使用experiment/src/step8/source.txt

**解答：**

1. 解题思路：

输出流不需要修改。故可以使用分件类型打开该文件，读取所需修改的文件。利用fgets函数循环将文件内容全部读入（而fgets函数如果发生错误，返回一个空指针）。

而此处查找替换字符串可以使用暴力查找匹配。其时间复杂度为。也可以使用KMP算法查找。但此处已经知道文件不大，故设计算法流程图如下：

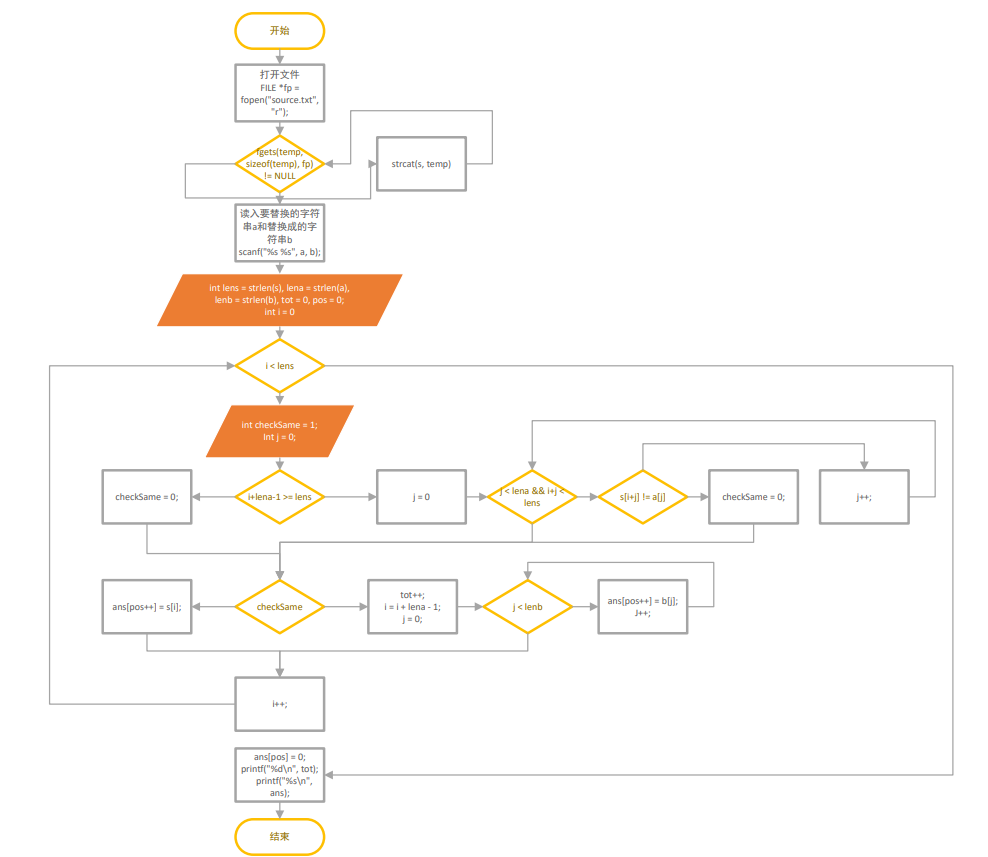


图8-4程序设计题1的算法流程示意图

1. 源程序清单

#include <stdio.h>

#include <string.h>

char s[100000], temp[10000], ans[100000];char a[1000], b[1000];

int main () {

FILE \*fp = fopen("source.txt", "r");

s[0] = '\0';

while(fgets(temp, sizeof(temp), fp) != NULL)

strcat(s, temp);

scanf("%s %s", a, b);

int lens = strlen(s), lena = strlen(a), lenb = strlen(b), tot = 0, pos = 0;

for(int i = 0; i < lens; i++) {

int checkSame = 1;

if(i+lena-1 >= lens)

checkSame = 0;

else

for(int j = 0; j < lena && i+j < lens; j++)

if(s[i+j] != a[j]) {

checkSame = 0;

break;

}

if(checkSame) {

tot++;

i = i + lena - 1;

for(int j = 0; j < lenb; j++)

ans[pos++] = b[j];

}

else

ans[pos++] = s[i];

}

ans[pos] = 0;

printf("%d\n", tot);

printf("%s\n", ans);

return 0;

}

3）测试

（a） 测试数据：

测试文章source.txt如下：

There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!

I love you! Do you love me?

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8-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 被替换串与替换串 |
| 例1 | you they | 5  There are moments in life when they miss someone so much that they just want to pick them from theyr dreams and hug them for real!  I love they! Do they love me? | 5  There are moments in life when they miss someone so much that they just want to pick them from theyr dreams and hug them for real!  I love they! Do they love me? |
| 例2 | want love | 1  There are moments in life when you miss someone so much that you just love to pick them from your dreams and hug them for real!  I love you! Do you love me? | 1  There are moments in life when you miss someone so much that you just love to pick them from your dreams and hug them for real!  I love you! Do you love me? |
| 例3 | work w | 0  There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!  I love you! Do you love me? | 0  There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!  I love you! Do you love me? |

（b） 对应测试测试用例1的运行结果如图8-5所示。

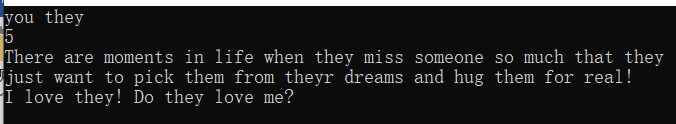


图8-5 编程题1的测试用例一的运行结果

对应测试测试用例2的运行结果如图8-6所示。

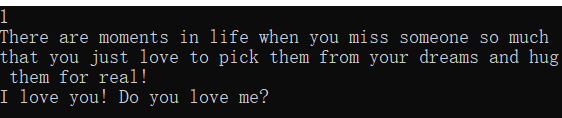


图8-6 编程题1的测试用例二的运行结果

对应测试测试用例3的运行结果如图8-7所示。

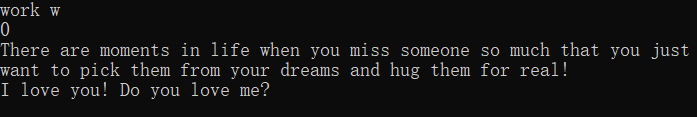


图8-7 编程题1的测试用例三的运行结果

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。

## 8.3 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。

通过本次实验，我较深入地学习了C语言的文件操作，特别是对二进制文件的读写掌握的更加深刻。先前的数据结构课程设计中，由于文件读入的原因出了不少差错，进了不少的坑，现在通过文件类型的程序验证题从理论上更深入的理解了二进制文件的读写。也通过源程序修改替换题和程序设计题1掌握了文件类型，和输入输出重定向进行文件读写的实现方式。也在程序设计题1中使用文件读写解决了一定的实际问题。

参考文献

[1] 曹计昌,卢萍,李开. C语言程序设计,北京： 科学出版社,2013

[2] 李开,卢萍,曹计昌. C语言实验与课程设计, 北京：科学出版社,2011