

**课 程 实 验 报 告**

**课程名称： C语言程序设计**

**专业班级：计算机科学与技术（校交）1601班**

**学 号： U201612696**

**姓 名： 陈淏睿**

**指导教师： 甘早斌**

**报告日期： 2017.6.8**

**计算机科学与技术学院**

**目录**

[1表达式和标准输入输出实验 3](#_Toc484681458)

[1.1必做题 3](#_Toc484681459)

[1.2自设题 8](#_Toc484681460)

[1.3小结 8](#_Toc484681461)

[2流程控制实验 10](#_Toc484681462)

[2.1必做题 10](#_Toc484681463)

[2.2选做题 18](#_Toc484681464)

[2.3自设题 19](#_Toc484681465)

[2.4小结 21](#_Toc484681466)

[3函数与程序结构实验 22](#_Toc484681467)

[3.1必做题 22](#_Toc484681468)

[3.2选做题 28](#_Toc484681469)

[3.3自设题 29](#_Toc484681470)

[3.4小结 30](#_Toc484681471)

[4编译预处理实验 32](#_Toc484681472)

[4.1必做题 32](#_Toc484681473)

[4.2自设题 39](#_Toc484681474)

[4.3小结 40](#_Toc484681475)

[5数组实验 41](#_Toc484681476)

[5.1必做题 41](#_Toc484681477)

[5.2选做题 53](#_Toc484681478)

[5.3自设题 58](#_Toc484681479)

[5.4小结 60](#_Toc484681480)

[6指针实验 62](#_Toc484681481)

[6.1必做题 62](#_Toc484681482)

[6.2选做题 74](#_Toc484681483)

[6.3自设题 80](#_Toc484681484)

[6.4小结 81](#_Toc484681485)

[7结构与联合实验 83](#_Toc484681486)

[7.1必做题 83](#_Toc484681487)

[7.2选做题 94](#_Toc484681488)

[7.3自设题 94](#_Toc484681489)

[7.4小结 94](#_Toc484681490)

[8文件实验 96](#_Toc484681491)

[8.1必做题 96](#_Toc484681492)

[8.2自设题 100](#_Toc484681493)

[8.3小结 100](#_Toc484681494)

[参考文献 101](#_Toc484681495)

# 1表达式和标准输入输出实验

## 1.1必做题

⑴程序改错：1题

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

9 /\* for task 1 \*/

10 printf(“Input Fahrenheit:” ) ;

11 scanf(“%d”, f ) ;

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

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

14

15 /\* for task 2 \*/

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

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

18 s = PI \* r \* r;

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

20

21 /\* for task 3 \*/

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

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

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

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

26

27 }

**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3行的void 和 main没有分开，正确形式为：

void main( void )

3) 第11行的r未被修改值，正确形式为：

scanf(“%d”,&f ) ;

4）第12行的除法有误，正确形式为：

c = 5.0/9\*(f-32) ;

5) 第 24行的new int 没有定义，正确形式为：

（第6行）short p, k, newint;

6) 第 17行的输入类型错误，正确形式为：

scanf("%lf", &r);

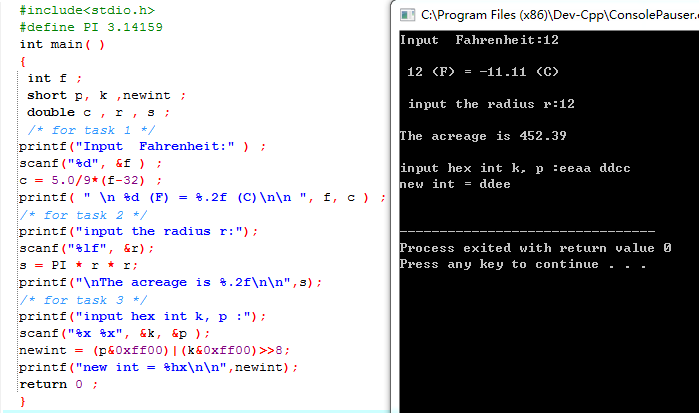
7) 第24 行的表达式有误，正确形式为：

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

8) 第25行的输出格式有误，正确形式为：

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

（2）错误修改后运行结果：



⑵程序修改替换：1题

下面的程序利用常用的中间变量法实现两数交换，请改用不用第三个变量的交换法实现。

#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：**

替换后的程序如下所示：

#include<stdio.h>

void main( )

{

int a, b, t;

printf("Input two integers:");

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

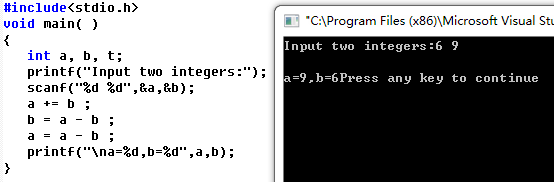
a += b ;

b = a - b ;

a = a - b ;

printf("\na=%d,b=%d",a,b);

}



⑶程序设计：2题

1.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.制作逻辑尺:

2.1 设计思路：要保留的数位在逻辑尺中对应1，其余对应0；

2.2 0左移（n-1）位，取反，即n位1；再左移m位，产生m位0；

3. 用逻辑尺筛选目标数的数位后左移，左移位数确定为16 – m – n ;

4. 输出结果，结束

2）程序清单

# include <stdio.h>

int main ()

{

int i , a , j ;

unsigned short x , m , n , mask ;

scanf ( "%d" , &a ) ;

for ( i = 0 ; i < a ; i ++ ) /\*确定数据输入组数\*/

{

mask = 1 ;

scanf ( "%hu %hu %hu" , &x , &m , &n ) ;

if ( m <= 0 || m >= 15 ) /\*判断m , n 的输入是否满足条件\*/

{

printf ( "incorrect input of m\n") ;

}

if ( n <= 1 || n >= ( 16 - m ) )

{

printf ( "incorrect input of n\n") ;

}

else

{

mask = ( ~ ( 0 << ( n - 1 ) ) ) << m ; /\*在逻辑尺将目标数位置为1\*/

x &= mask ;

x <<= 15 - m - n + 1 ; /\*筛选出的数位左移\*/

printf ( "%hu\n" , x ) ;

}

}

}

3）测试

（a） 测试数据：

叙述选择测试数据的方法。。。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 |  |
| X | M | N |
| 用例1 | 0000 0010 0010 1011（555） | 2 | 5 | 计算结果0101 0000 0000 0000 即20480 |
| 用例2 | 0000 0000 0000 0001（1） | 17 | 1 | m , n均错误 |
| 用例3 | 0001 0000 1010 0001（4257） | 11 | 15 | n错误 |

（b） 对应测试测试用例的运行结果如图1所示。

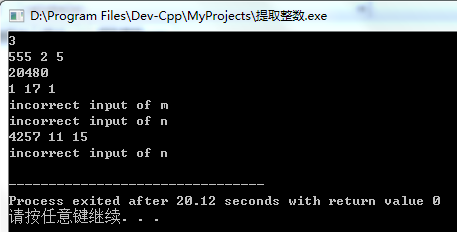


图1 编程题1的测试用例的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

2.IP地址通常是4个用句点分隔的小整数，如32.55.1.102。这些地址在机器中用无符号长整形表示。编写一个程序，以机器存储的形式读入一个32位的互联网IP地址，对其译码，然后用常见的句点分隔的4部分的形式输出。。

**解答：**

1) 解题思路：

1.输入数据数n；

2.用数组存储译码后的四个无符号整数。由于译码按字节进行，因此数组类型用unsigned char便够用。

3.由于输出低字节译码按从低到高的顺序，因此可得出原整数左移位数的数学表达式为( 4 - 1 - j ) \* 8，其中为实现可移植性，4用sizeof ( unsigned long )代替；8用字节长度CHAR\_BIT代替。

4. 右移8\*3位。同样地，8用字节长度CHAR\_BIT代替。

5. 按顺序输出数组内各数；

6．循环，直至输出组数达到n；

6.结束。

2）程序清单

# include <stdio.h>

# include <limits.h>

const int UL = sizeof ( unsigned long ) ;

int main ()

{

int i , j , n ;

unsigned long origin ;

scanf ( "%d" , &n ) ; /\*设定输入组数\*/

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

{

unsigned short s[j] = {0} ;

scanf ( "%lu" , &origin ) ;

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

{

s[j] = ( ( origin << ( UL - 1 - j ) \* CHAR\_BIT ) >> ( ( UL - 1 ) \* CHAR\_BIT ) ) ;

/\*从低到高依次筛选并保留原数的4个字节并存入数组，实现方法详见解题思路\*/

}

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

{

printf ( "%d." , s[j] ) ;

/\*原数各字节逆序输出的问题已在筛选数位时考虑，此处仅需进行顺序输出\*/

}

printf ( "%d\n" , s[3] ) ;

}

return 0 ;

} 3）测试

（a） 测试数据：

表1-1 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |  |
| Origin |
| 用例1 | 134744072 | 8.8.8.8 |
| 用例2 | 676879571 | 211.92.88.40 |
| 用例3 | 16885952 | 192.168.1.1 |
| 用例4 | 16843009 | 1.1.1.1 |  |
| 用例5 | 26975347 | 115.156.155.1 |  |

（b） 对应测试测试用例的运行结果如图2所示。

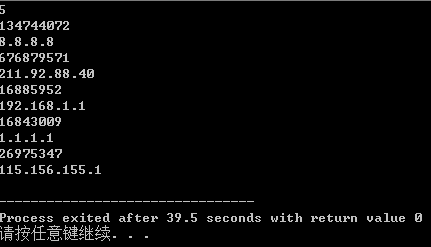


图2 编程题2的测试用例的运行结果

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。

## 1.2自设题

## 1.3小结

前面的改错题出错点虽然基础，但是最基础的地方才是最考验基本功的地方。好多处错误我虽然看了出来但却不知道或者不确定怎么改，迫不得已只能去翻书，也有像定义类型与输入类型不符的错误是我一时半会不能看出来，要在程序体不同位置一次次输出r来检查才看出来的。总而言之它暴露了我很多基础知识方面的问题，也督促我接着去看书，把基本功烂熟于心。

得益于上学期C++的训练，第二大题我很快完成，但第三大题则出了问题。提取整数中我开始用了循环体做逻辑尺，但发现在用“555 2 5“这组数测试的时候循环莫名其妙的提前终止了（目前依然在寻找原因）。但这也是好事，它逼着我使用当时还不熟悉的位运算去做逻辑尺，托这个错误的福我现在对逻辑尺也有了更熟练的运用与更深层次的理解。

本次实验中我尝试了了不同类型的开发环境，并自己动手实践了一些简单的小程序，在开发程序的过程中我发现一些小的问题往往会破坏整个程序的正常运行，而且非常不容易发现，所以我认为在我们开发程序的过程中，必须要保证好整个源码的格式，并且要十分之小心。目前我能够运用简单的函数调用，分得清数据的类型，掌握了几个循环语句的使用方法，但均不够熟练，在以后的实践过程中还会去不断提高自己，成为一个有基本职业素养的程序开发人员。

# 2流程控制实验

## 2.1必做题

⑴程序改错：1题

下面是计算s=n!的源程序，在这个源程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。例如，8！=40320。

1 #include <stdio.h>

2 void main(void)

3 {

4 int i,n,s=1;

5 printf("Please enter n:");

6 scanf("%d",n);

7 for(i=1,i<=n,i++)

8 s=s\*i;

9 printf("%d! = %d",n,s);

10 }

**解答：**

（1）错误修改：

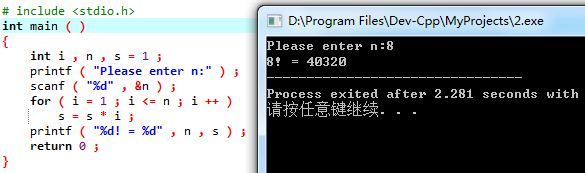
1) 第6行的n未被修改值，正确形式为：

scanf("%d",&n);

2) 第7行的循环条件分隔符号错误，正确形式为：

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

（2）错误修改后运行结果：



⑵程序修改替换：2题

1.修改1.2.1题，分别用while和do-while语句替换for语句。

**解答：**

替换后的程序如下所示：

//while语句

# include <stdio.h>

int main ( )

{

int i = 1 , n , s = 1 ;

printf ( "Please enter n:" ) ;

scanf ( "%d" , &n ) ;

while ( i <= n )

{

s \*= i ;

i ++ ;

}

printf ( "%d! = %d" , n , s ) ;

return 0 ;

}

//do-while语句

# include <stdio.h>

int main ( )

{

int i = 1 , n , s = 1 ;

printf ( "Please enter n:" ) ;

scanf ( "%d" , &n ) ;

do

{

s \*= i ;

i ++ ;

}

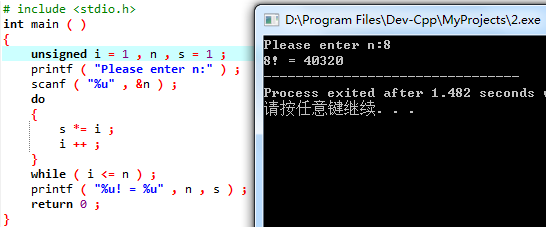
while ( i <= n ) ;

printf ( "%d! = %d" , n , s ) ;

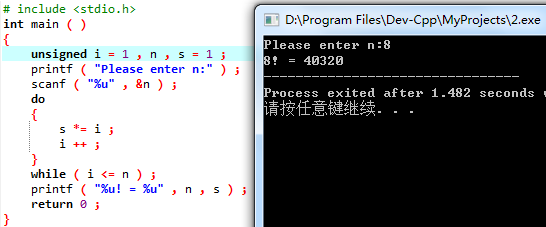
return 0 ;

}

while下:



do-while下：



2.修改第1题，输入改为“整数S”，输出改为“满足n！≥S的最小整数n”。例如输入整数40310，输出结果为n=8。

**解答：**

# include <stdio.h>

int main ( )

{

unsigned i = 1 , n , s = 1 , S ;

printf ( "Please enter S:" ) ;

scanf ( "%u" , &S ) ;

while ( s < S )

{

i ++ ;

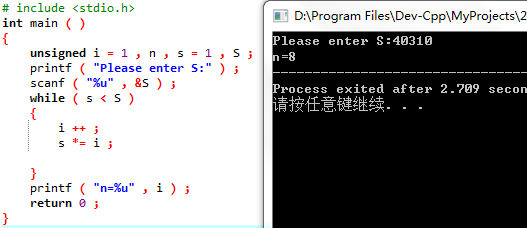
s \*= i ;

}

printf ( "n=%u" , i ) ;

return 0 ;

}



⑶程序设计：4题

1.个人所得税（实验题）

**解答：**

1) 解题思路：

1.输入薪水money；

2.对money的值进行判断，按题目所给区间设定if的条件并将money落入if；

2.1 money<1000，不交税 ；

2.2 money > 1000 && money <= 2000，超出1000的部分交%5；

2.3 money > 2000 && money <= 3000，固定交50，超出2000的部分交%10；

2.4 money > 3000 && money <= 4000，固定交150，超出3000的部分交%15；

2.5 money > 4000 && money <= 5000，固定交300，超出4000的部分交%20；

2.6 money > 5000，固定交500，超出5000的部分交%25；

3. 结束

2）程序清单

int main ()

{

double money ;

scanf ( "%lf" , &money ) ;

while ( money != 0 )

{

if ( money <= 1000 ) //不收税

printf ( "%f\n" , 0 ) ;

else if ( money > 1000 && money <= 2000 )

//1000<x<=2000 , 收%5税金

printf ( "%f\n" , ( money - 1000 ) \* .05 ) ;

else if ( money > 2000 && money <= 3000 )

//2000<x<=3000 , 收%10税金

printf ( "%f\n" , 1000 \* .05 + ( money - 2000 ) \* .1 ) ;

else if ( money > 3000 && money <= 4000 )

//3000<x<=4000 , 收%15税金

printf ( "%f\n" , 1000 \* .15 + ( money - 3000 ) \* .15 ) ;

else if ( money > 4000 && money <= 5000 )

//4000<x<=5000 , 收%20税金

printf ( "%f\n" , 1000 \* .3 + ( money - 4000 ) \* .2 ) ;

else //x>5000 , 收%25税金

printf ( "%f\n" , 1000 \* .5 + ( money - 5000 ) \* .25 ) ;

scanf ( "%lf" , &money ) ;

}

}

3）测试

（a） 测试数据：

表2-1 编程题1的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| n |
| 用例1 | 500 | 0.000000 | 0.000000 |
| 用例2 | 1500 | 25.000000 | 25.000000 |
| 用例3 | 2500 | 100.000000 | 100.000000 |
| 用例4 | 3500 | 225.000000 | 225.000000 |
| 用例5 | 4500 | 400.000000 | 400.000000 |
| 用例6 | 5500 | 625.000000 | 625.000000 |

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.空格处理（实验题）

**解答：**

1) 解题思路：

1.准备工作：i,n控制循环组数，字符数组长度200：大开小用；getchar()隔开字符组数与字符文本；

2.空格处理：

2.1 首先flag置为0，此时当前字符可以输出，通过后续flag 1/0的变化决定当前字符是否输出，1：是/0：否；

2.2 用count存输出字符总数作为检查（尤其是字符串末尾有一串空格的情况）；

2.3 当前字符非空格时flag取默认值，同时指针从空格移向非空格时将flag改回默认值；

2.4 当前字符为首字符空格或字符间首个空格时，由于此时flag仍为0，此空格可正常输出，但flag将在输出工作后改为1；

2.5 flag为1时，若当前字符非空格则修改flag，否则不输出；

3. 指针指向空字符时一轮输出工作结束，进行下一轮输入；

4. 结束

2）程序清单

int main ()

{

unsigned i , j , n , count , flag ;

char sen[200] ;

scanf ( "%u" , &n ) ;

getchar () ;

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

{

flag = count = 0 ; //由flag判断当前状态是否输出字符

fgets ( sen , 200 , stdin ) ;

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

{

if ( sen[j] != ' ' ) flag = 0 ;

//由空格切进字符时改flag，继续输出

if ( flag ) continue ;

//在一连串空格中不进行输出

if ( ! ( flag ) )

{

putchar ( sen[j] ) ; //当前为非空格字符时正常输出

count ++ ; //count用于协助检查输出

if ( sen[j] == ' ' )

flag = 1 ;

//由非空字符切入空格时改flag,由对后续字符的判断结果决定flag是否改变

}

}

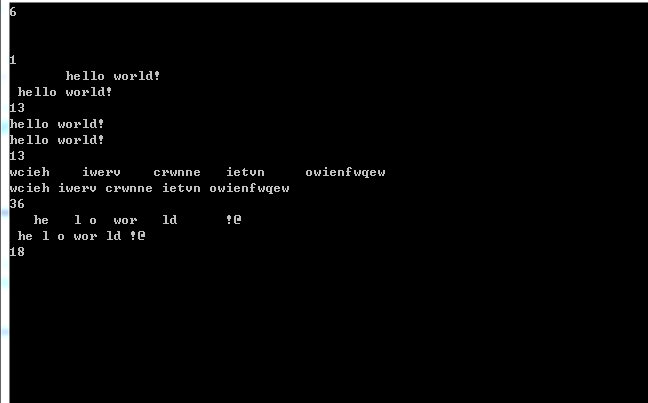
printf ( "\n%d" , count ) ;

putchar ( '\n' ) ;

}

}

3）测试数据及相应的运行结果截图



说明上述的运行结果与理论分析吻合，验证了程序的正确性。

3.打印如下杨辉三角形。

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

每个数据值可以由组合计算（表示第i行第j列位置的值），而的计算如下：

 (i=0,1,2,…)

 (j=0,1,2,3,…,i)

本程序中为了打印出金字塔效果，要注意空格的数目。一位数之间是3个空格，两位数之间有2个空格，3位数之间只有一个空格，程序编制过程中要注意区分。

**解答：**

1) 解题思路：

1.由于行数在13以内才能让杨辉三角满足题目格式，因此二维数组定义为13\*13的方阵；

2.生成杨辉三角：

2.1 将矩阵的第一列和主对角线上元素置为1；

2.2 从第三行开始生成杨辉三角的其他数，即此位置在方阵中正上与左上元素之和；

3.按需输出杨辉三角：

3.1 确定输出行数；

3.2 每列输出数据的个数等于当前行的行数；

4. 结束

2）源程序清单

# include <stdio.h>

int main ()

{

short i , j , n , blank , tri[13][13] = {0} ;

//数组行数在13以内时才可能按照题目要求格式输出

for ( i = 0 ; i < 13 ; i ++ )

{

tri[i][0] = tri[i][i] = 1 ;

//初始化杨辉三角的两腰，即矩阵的最左列及主对角线

}

for ( i = 2 ; i < 13 ; i ++ )

{

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

//以杨辉三角的两腰为基础生成整个杨辉三角，方便取用

{

tri[i][j] = tri[i-1][j-1] + tri[i-1][j] ; //杨辉三角递推公式

}

}

scanf ( "%hd" , &n ) ;

while ( n != 0 )

{

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

{

blank = 2 \* ( n - 1 - i ) ;

/\*观察输出格式，可发现数据输出行宽为4，杨辉三角中数据隔一行对齐一次，故行缩进差为2\*/

for ( j = 1 ; j <= blank ; j ++ ) //调节空格输出等腰三角形

putchar ( ' ' ) ;

for ( j = 0 ; j <= i ; j ++ ) //按序输出杨辉三角的前n行

printf ( "%-4hd" , tri[i][j] ) ;

putchar ( '\n' ) ;

}

putchar ( '\n' ) ;

scanf ( "%hd" , &n ) ;

}

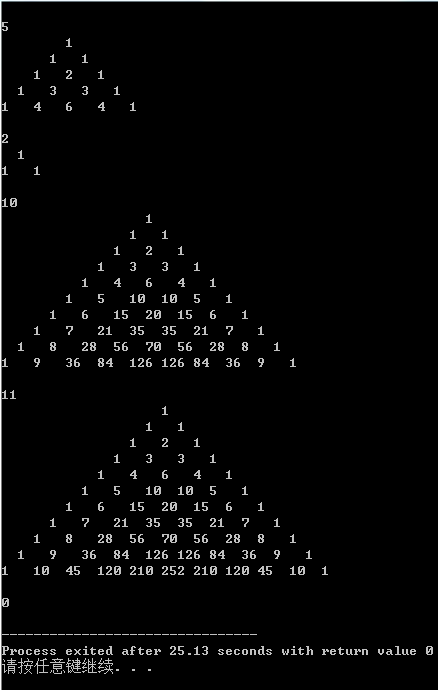
}

3）测试

（a） 测试数据：

5，2，10，11，0

（b） 对应测试数据的运行结果截图



说明上述的运行结果与理论分析吻合，验证了程序的正确性。

4.编写一个程序，将用户输入的任意正整数逆转，例如，输入1234，输出4321。

**解答：**

1) 解题思路：

1.题目要对每一次输入的数据进行判断，因此需要在在while语句之前附加一次输入

2.提取并输出各位数.

2.1 首先n % 10，提取个位数并输出；

2.2 n /= 10保证n%10每次提取的是原数据从小到大的各个数位；

2.3 n=0时逆序输出工作结束，进行下一次输入；

3. 输入为0时程序结束；

2）程序清单

# include <stdio.h>

# include <stdlib.h>

int main ()

{

unsigned long long n ;

//用unsigned long long装下绝对值尽可能大的输入数据

scanf ( "%llu" , &n ) ;

while ( n != 0 )

{

while ( n != 0 )

{

printf ( "%d" , n % 10 ) ; n /= 10 ; //提取个位数并输出

}

putchar ( '\n' ) ; scanf ( "%llu" , &n ) ;

}

}

3）测试

（a） 测试数据：

数据类型为unsigned long long时可正常存储的最大数据为2^64-1

表2-2 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 |
| n |
| 用例1 | 2^64-1 | 61,615,590,737,044,764,481 | 61,615,590,737,044,764,481 |
| 用例2 | 2^64 | return 0 | return 0 |
| 用例3 | 2^64+1 | 1（overfloat） | 1 |

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。

## 2.2选做题

程序设计：1题

求方程的近似根（实验题，选做题）

**解答：**

1) 解题思路：

1.按题目设定原函数与导函数；由于牛顿迭代法得到的方程近似根与迭代初值无关，因此x1可任意赋值；

2.显然循环终止条件为迭代出的两数之差小于给定误差1e-6，用x1和给定公式算出x2的值，若两数之差不满足终止条件则将x2赋给x1，开始下一轮循环；

3. 循环中止，输出x2，结束。

2）程序清单

# include <stdio.h>

int main ( )

{

double x1 = 0.0 , x2 , ori , dif ;

x2 = x1 + 1 ;

while ( fabs ( x1 - x2 ) > 1e-6 )

{

x1 = x2 ;

ori = 3 \* pow ( x1 , 3 ) - 4\* x1 \* x1 - 5 \* x1 + 13 ;

dif = 9 \* x1 \* x1 - 8 \* x1 - 5 ;

x2 = x1 - ori / dif ;

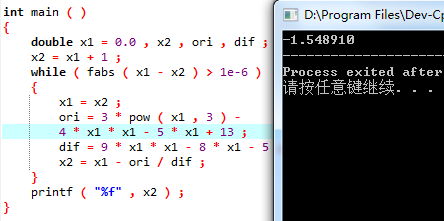
}

printf ( "%f" , x2 ) ;

}

3）测试

运行结果截图：



说明上述的运行结果与理论分析吻合，验证了程序的正确性。

## 2.3自设题

**（1）** 自设实验题目：鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一。百钱买百鸡，问鸡翁，母，雏各几何？

**（2）** 实验目的：通过设计实验程序，理解多重循环中计算时间随循环层数幂增长的规律，并明白通过算法优化可有效缩短运行时间。。

**（3）**实验程序：

#include <stdio.h>

int main()

//原始算法——三重循环

#include <stdio.h>

{

short i , j , k ;

printf ( " 公鸡 母鸡 小鸡\n" ) ;

for ( i = 0 ; i <= 20 ; i ++ )

for ( j = 0 ; j <= 33 ; j ++ )

for ( k = 0 ; k <= 300 ; k ++ )

if ( ( i + j + k == 100 ) && ( 5 \* i + 3 \* j + k / 3 == 100 ) && ( k % 3 == 0 ) )

printf ( "%6hd%6hd%6hd\n" , i , j , k ) ;

return 0 ;

}

//改进算法——二重循环

int main ( )

{

short i , j , k ;

printf ( " 公鸡 母鸡 小鸡\n" ) ;

for ( i = 0 ; i <= 20 ; i ++ )

{

for ( j = 0 ; j <= 33 ; j ++ )

{

k = 100 - i - j ;

if ( ( 5 \* i + 3 \* j + k / 3 == 100 ) && ( k % 3 == 0 ) )

printf ( "%6hd%6hd%6hd\n" , i , j , k ) ;

}

}

return 0 ;

}

**（4）**实验用例：

原始算法运行时间：





改进算法运行时间：





**（5）**实验结论：理论上讲，原始算法中循环体将执行20\*33\*300=198000次，改进算法中只需执行20\*33=660次，然而运行时间的缩短体现的并不明显。此为程序较小，对算法的优化未能体现充分，同时二重循环比三重循环多了几步计算的缘故。在更大更复杂的程序中，程序执行时间的减少将更为明显。

## 2.4小结

本次实验我进一步熟悉了Dev的操作。了解了在写程序的时候容易犯的一些小错误，在以后的编程道路上还得处处小心，具体的，发现scanf函数里面不能输入float型变量，不然会出错。另外进一步熟练了函数的调用过程，但总体来说，写程序的速度还是颇慢，以后亟待提高。

源程序改错与修改替换的训练与上学期C++的知识储备让我熟悉了三种循环体的使用方法与情境，但目前我的运用还较为死板，对for循环的三个语句的构造还停留在单语句简单程度，应在之后多加练习，熟练掌握多个起始条件、多个自增步骤、判断条件复杂化与多样化、语句空置等for语句的活用情况。对于while语句与do-while语句，我认为while更贴近我们的语言逻辑，do-while一样“先斩后奏”的情况在日常生活中出现较少，但所谓存在即合理，do-while的众多实用情景需要通过今后的编程一点点去摸索。

空格处理可以说是困住我时间最长的一道题。首先是由于一开始自己一定程度上被题目吓住了，产生了一定的害怕心理，其次是因为在构思的时候并没有明确而完整的算法体系，想到一点就直接往上写，导致后来自己也想不清楚自己的程序究竟是怎样运行的。所以说磨刀不误砍柴工，再动手敲代码之前一定要先在脑子里勾勒出清晰的算法架构，写起来才不至于产生“自己也不太明白自己在写什么”的感觉。同时通过这道题我也对用printf检查程序运行流程的方法运用的更为娴熟，但这也只是不熟悉单步调试时的不得已之策，还是应尽快掌握这一方法才能在检查程序时更为高效。

杨辉三角是对我而言是另外一个坑。算法上没遇上什么大问题，但是由于没有对样例输出格式仔细观察，导致我一直以为可以用左输出处理而忽略了两种输出方式下数与数之间空格的不同之处。有的时候文字里没有写出来的的要求是被体现在样例输出里的，今后一定要对其仔细观察，不可马马虎虎就跳过去从而遗漏很多细节。

数字倒置一题其实可以用后面的数组知识解决，如此便解决了数据类型值域受限的问题；更可以用链表知识处理，解决了数组处理时数组长度必须预先决定的问题。但对于处理的数据类型较简单、绝对值较小时，后两种方法未免有一些牛刀杀鸡的感觉，因此按位提取在这里其实已经够用。

# 3函数与程序结构实验

## 3.1必做题

⑴程序改错：1题

下面是计算s=1!+2!+3!+…+n!的源程序，在这个源程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。

1 #include "stdio.h"

2 void main(void)

3 {

4 int k;

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

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

7 }

8 long sum\_fac(int n)

9 {

10 long s=0;

11 int i;

12 long fac;

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

14 fac\*=i;

15 s+=fac;

16 return s;

17 }

**解答：**

（1）错误修改：

1) 第8行开始的函数没有声明，正确形式为：

（第1行与第2行之间插入）long sum\_fac ( int n ) ;

2) 第12行的fac未初始化，正确形式为：

long fac = 1;

3) 第14,15行的语句未装入一个循环节，正确形式为：

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

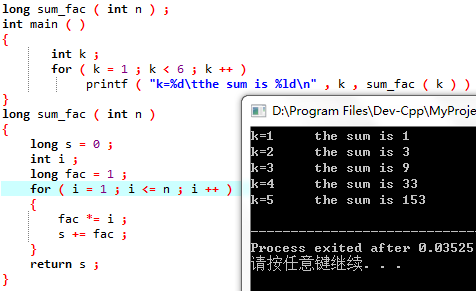
{

fac\*=i;

s+=fac;

}

（2）错误修改后运行结果：



⑵程序修改替换：2题

1.修改第1题中sum\_fac函数，使其计算量最小。

**解答：**

#include<stdio.h>

long sum\_fac ( int n )

{

static long s = 0 , fac = 1 ;

fac \*= n ;

s += fac ;

return s ;

}

int main ( void )

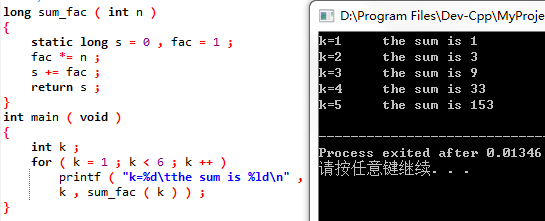
{

int k ;

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

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

}



2.修改第1题中sum\_fac函数，计算。

**解答：**

long myfac ( int n )

{

if ( n == 0 || n == 1 )

return 1 ;

else

return n \* myfac ( n - 1 ) ;

}

double sum\_fac ( int n )

{

if ( n == 0 || n == 1 )

return 1 ;

else

return 1 / ( double )( n \* myfac ( n - 1 ) ) + sum\_fac ( n - 1 ) ;

}

int main ( )

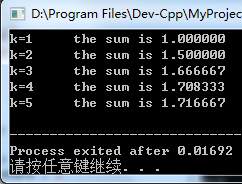
{

int k ;

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

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

}



⑶跟踪调试：4题

计算fabonacci数列前n项和的程序如下：

其中，long sum=0,\*p=&sum;声明p为长整型指针并用&sum取出sum的地址对p初始化。\*p表示引用p所指的变量（\*p即sum）。

void main(void)

{

int i,k;

long sum=0,\*p=&sum;

scanf("%d",&k);

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

sum+=fabonacci(i);

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

}

}

long fabonacci(int n)

{

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

return 1;

else

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

}

单步执行程序，观察p,i,sum,n值。

1. 刚执行完scanf("%d",&k);语句，p,i值是多少？

p:0x16（随机分配地址）；i:2;

1. 从fabonacci函数返回后光条停留在哪个语句上？

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

1. 进入fabonacci函数，watch窗口显示的是什么？

Function arguments:n—2; Locals:（无）；

（4）当i=3，从调用fabonacci函数到返回，n值如何变化？

n=3 => n=2 => n=1 ;

⑷程序设计：2题

1.编程让用户输入两个整数，计算两个数的最大公约数并且输出之（要求用递归函数实现求最大公约数）。同时以单步方式执行该程序，观察递归过程。

**解答：**

1) 解题思路：

1. 先期判定输入的a,b是否满足条件，只有循环条件成立时才进行求公约数计算;

2.辗转相减求最大公约数；

2.1 递归终止条件：两数中较大的可整除较小的，满足条件时返回较小值，此即为原来的两数的最大公约数；

2.2 不满足条件时返回b与两数差的绝对值。这是考虑到若返回a则a>b时return maxdiv ( b , abs ( a - b ) ) 是死循环；

3. 继续输入a和b，进行循环条件的下一次判断；

4. a为0时跳出循环，程序结束;

2）源程序清单

int maxdiv ( int a , int b )

{

if ( a % b != 0 )

return maxdiv ( b , abs ( a - b ) ) ;

//从递归第三层开始稳定输出a,b中较小的数及两数差的绝对值

return b ;

}

int main ()

{

int a , b ;

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

while ( a != 0 )

{

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

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

}

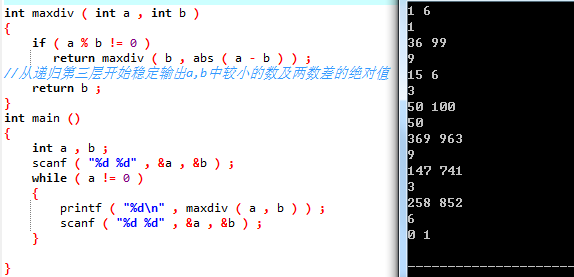
}

3）测试

（a） 测试数据：

1&6 36&99 15&6 50&100 369&963 147&741 258&852 0&2

（b） 对应测试数据的运行结果截图



2.编程验证歌德巴赫猜想：一个大于等于4的偶数都是两个素数之和。

编写一个程序证明对于在符号常量BEGIN和END之间的偶数这一猜测成立。例如，如果BEGIN为10，END为20，程序的输出应为：

GOLDBACH'S CONJECTURE:

Every even number n>=4 is the sum of two primes.

10=3+7

12=5+7

……

20=3+17

**解答：**

1) 解题思路：

1. 输出判定区间START和END ;

2.对START和END判断，START若为奇数则+1；END若为偶数则-1；

3. 对START和END间的每个偶数进行算子判断，若算子均为质数则输出算式；

4. 进入算子判断函数，flag作为标志变量方法为若其对2到其平方根的任一整数取余为0则其非质数，进入下一循环。

5.返回（非）flag值 ;

6.结束 ;

2）程序清单

char prime ( long n )

{

char flag = 1 ; long i = 2 ;

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

if ( n % i == 0 ) //若非质数, flag=0 ，相应地main中if语句判断为假

return ! ( flag ) ;

return flag ;

}

int main ()

{

long a , b , i , j , BEGIN , END ;

scanf ( "%ld %ld" , &BEGIN , &END ) ;

while ( BEGIN != 0 )

{

if ( BEGIN < 6 || END < 6 || END < BEGIN ) //非法输入干预

{ printf ( "error!\n" ) ; }

else

{

if ( BEGIN % 2 )

a = BEGIN + 1 ;

else

a = BEGIN ; // 设定循环初值（偶数）

if ( END % 2 )

b = END - 1 ;

else

b = END ; // 设定循环终值（偶数）

for ( i = a ; i <= b ; i += 2 )

for ( j = 2 ; j <= i / 2 ; j ++ )

if ( prime ( j ) && prime ( i - j ) ) //判断两个算子是否均为质数

{

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

break ;

}

}

putchar ( '\n' ) ;

scanf ( "%ld %ld" , &BEGIN , &END ) ;

}

}

3）测试

对应测试用例的运行结果如图1-2所示。

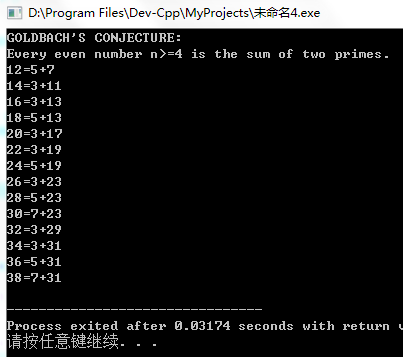


图3-2 编程题2的测试用例的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

## 3.2选做题

程序设计：1题

设file1.c如下：

#include <stdlib.h>

#include "HEADER.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 ;

}

file2.c如下：

#include "HEADER.h"

void func1(void)

{

x++;

y++;

ch++;

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

}

HEADER.h如下：

#ifndef HEADER\_H\_INCLUDED

#define HEADER\_H\_INCLUDED

#include <stdio.h>

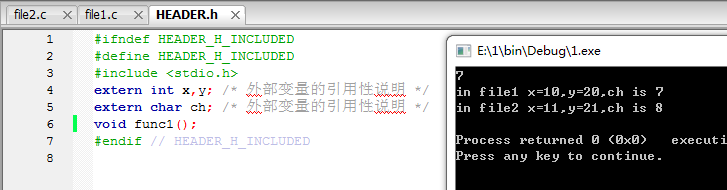
extern int x,y; /\* 外部变量的引用性说明 \*/

extern char ch; /\* 外部变量的引用性说明 \*/

void func1();

#endif // HEADER\_H\_INCLUDED

试用TCC进行多文件编译和链接。然后在DOS环境下运行生成的可执行文件。



## 3.3自设题

自设实验题目：定义三个函数，分别求两个整数，三个整数及三个浮点数之和

**（2）** 实验目的：初步熟悉函数重载特点及原理

**（3）**实验程序：

#include <stdio.h>

int sum ( int a , int b ) //int型，双参数

{

return a + b ;

}

int sum ( int a , int b , int c ) //int型，三参数

{

return a + b + c ;

}

double sum ( double a , double b , double c ) //double型，三参数

{

return a + b + c ;

}

int main ()

{

int a , b , c ;

double x , y , z ;

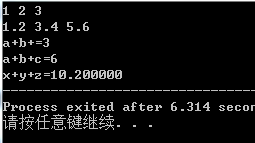
scanf ( "%d %d %d" , &a , &b , &c ) ;

scanf ( "%lf %lf %lf" , &x , &y , &z ) ;

printf ( "a+b+=%d\na+b+c=%d\nx+y+z=%f" , sum ( a , b ) , sum ( a , b , c ) , sum ( x , y , z ) ) ;

}

**（4）**实验用例：a=1;b=2;c=3;x=1.2;y=3.4;z=5.6;



**（5）**实验结论：函数重载可减少函数名数量并提高其可读性。但应注意的是要实现重载至少要求形参个数或类型有区别以避免函数调用二义性；编译器将根据不同参数类型和个数产生调用匹配。

## 3.4小结

递归虽然曾经在C++中接触过，但也只是接触而已。这次是第一次自己大规模频繁运用递归，虽然每次都要在上面卡一会壳但后来也都顺利过关。要正确应用并熟练掌握递归，重要的还是要对程序的具体实现方法与递归的算法、传值特点、计算顺序有深刻的了解。同时，递归虽然算法结构紧凑、清晰、可读性强、代码简洁，但其运行效率较低，计算时间较长，占用的存储空间也尤其多。因此在运用递归写程序时要注意对程序本身是否有计算效率的要求以及内存是否能够为递归运算准备足够多的空间。

本次在程序设计题上我没有遇到什么大的障碍，唯一值得注意的一点是递归求最大公约数中递归值应是a和abs ( a - b )而非b和abs ( a - b )。这样的小错误虽说很容易改，但以后在团队开发中不容易查出来的恰恰是这种小bug。所以在今后设计程序时一定要一开始想清楚，不要等到最后让攒起来的一堆小错把自己搞晕了。

单步调试是之前没接触过的新技法。熟悉操作花了一段时间，最后虽然调试成功了但是熟练度还是欠佳，尤其是题目没有要求的Step out 和Run to curser 还显生疏。回去还是要在自己编程的时候多加练习。单步调试可以有效提高程序debug的速度，但前提是对其使用方法有深刻了解，同时要注意的是编译器不同，单步调试的显示风格、方式也有所不同，比如Dev-C++在观察字符数组时会附带显示其首地址，但Codeblocks便不会。因此还应该熟练掌握在watch窗口中显示不同内容（值或地址）的方法以备不时之需。

在程序体较小且函数关系简单时，函数体提前于主函数不失为好方法，可省去函数的声明从而在一定程度上减小代码量。但在面对函数较多、相互调用关系复杂时，将被调函数一股脑堆在前面不仅会造成阅读程序时的不便，更会在函数之间有相互调用时被绕的晕头转向或出现函数未声明的情况。因此在此情况下应将被调函数统一放在主函数后并在前面加上声明，同时声明顺序与被调函数书写顺序最好能够保持一致。

多文件编译虽是第一次接触但入门还算简单，只是在目前程序体还较小，此方法没什么付诸实践的机会。要真正掌握还是要自己找程序写分块文件。

# 4编译预处理实验

## 4.1必做题

⑴程序改错：1题

1.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语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。

1 #include "stdio.h"

2 #define SUM a+b

3 #define DIF a-b

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

5 void main

6 {

7 int b, t;

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

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

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

11 SWAP(a,b);

12 Printf("\nNow a=%d,b=%d\n",a,b);

13 }

**解答：**

（1）错误修改：

1) 第7行的变量定义错误，正确形式为：

int b, a;

2) 第2行的无参宏定义形式错误，正确形式为：

#define SUM （a+b）

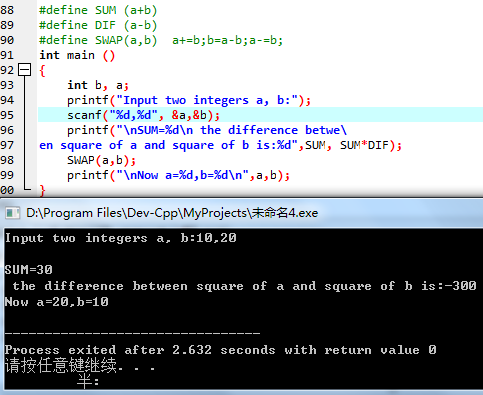
3) 第3行的无参宏定义形式错误，正确形式为：

#define DIF （a-b）

2) 第4行的带参宏定义算法错误，正确形式为：

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

（2）错误修改后运行结果：



⑵程序修改替换：2题

下面是用函数实现求三个数中最大数、计算两数之和的程序，在这个源程序中存在若干语法和逻辑错误。

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；

2）用带参数的宏替换函数max，来实现求最大数的功能。

void main(void)

{

int a, b, c;

float d, e;

printf("Enter three integers:");

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

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

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

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

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

}

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

{

int t;

if (x>y)

t=x;

else

t=y;

if (t<z)

t=z;

return t;

}

float sum(float x, float y)

{

return x+y;

}

**解答：**

替换后的程序如下所示：

#include<stdio.h>

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

float sum ( float x , float y )

{

return x + y ;

}

int main(void)

{

int a , b , c ;

float d , e ;

printf ( "Enter three integers:" ) ;

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

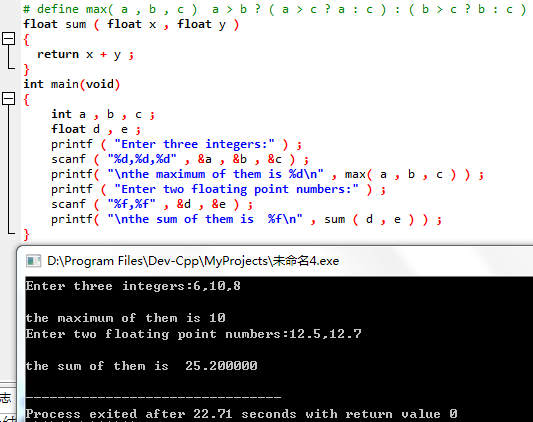
printf( "\nthe maximum of them is %d\n" , max( a , b , c ) ) ;

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

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

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

}



⑶跟踪调试：3题

下面程序利用R计算圆的面积s，以及面积s的整数部分。

#define R

void 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);

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

assert((s-s\_integer)<1.0);

#endif

}

int integer\_fraction(float x)

{

int i=x;

return i;

}

1）修改程序，使程序编译通过且能运行；

# include <assert.h> //添加

# include <stdio.h>

# define R

int integer\_fraction ( float x )

{

int i = x ;

return i ;

}

int main ( )

{

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 ) ;

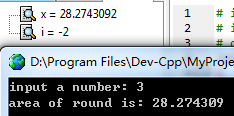
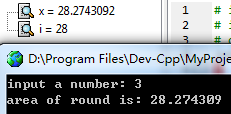
printf ( "the integer fraction of area is %d\n" , s\_integer ) ;

assert ( ( s - s\_integer ) < 1.0 ) ;

#endif

}

2）单步执行。进入函数decimal\_fraction时watch窗口中x为何值？在返回main时, watch窗口中i为何值？

进入：返回：

3）排除错误，使程序能正确输出面积s值的整数部分，不会输出错误信息assertion failed。（见1）中程序）

⑷程序设计：2题

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

**解答：**

1) 解题思路：

1.输入三角形三边长a，b，c

2.实现循环条件：输入不为文件尾

3.半周长与面积的宏定义实现

2.1 带参宏定义半周长，参数为三角形三边长，其值应用变量接收，否则需要在面积的宏定义中包含半周长的宏定义，徒增计算次数且宏定义时代码过长

2.2 带参宏定义面积，参数为三边长及接收的半周长，其值可直接输出

4. 输入并进行下一轮循环的判断

5. 结束

2）源程序清单

# include <stdio.h>

# define S( a , b , c ) ( a + b + c ) / 2

//带参宏定义S为三角形半周长

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

//带参宏定义AREA为海伦公式求得的三角形面积

int main ()

{

double a , b , c , s ;

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

//double类型在输入格式串中记为%lf

{

s = S( a , b , c ) ;

//用变量接住宏定义求得的半周长，并作为面积宏定义的参数

printf ( "%d %f\n" , ( int ) s , AREA( a , b , c , s ) ) ;

//按输出要求对double型的s强制转换，同时AREA的值可直接格式输出

}

}

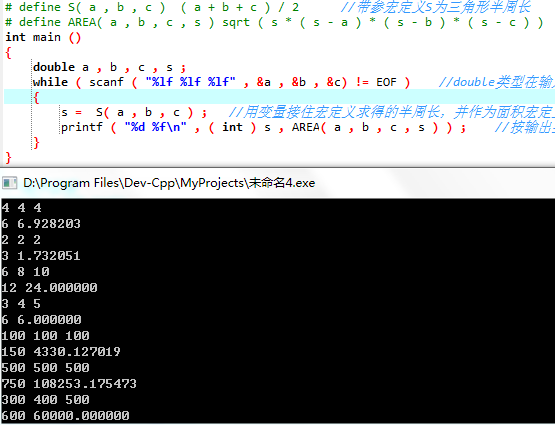
3）测试

（a） 测试数据：

4&4&4 2&2&2 6&8&10 3&4&5

100&100&100 500&500&500 300&400&500

（b） 对应测试数据的运行结果截图



2.用条件编译方法来编写程序。输入一行电报文字，可以任选两种输出：一为原文输出；二为变换字母的大小写（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：

1.输入n，确定测试字符串组数；

2.fgets进行字符串输入；

2.条件编译格式.

2.1 main函数前无参宏定义设定CHANGE值；

2.2 if-else结构用条件编译替换，注意作为判定条件的常量表达式应在预编译时已确定取值

3. 条件编译结构；

3.1 CHANGE==0 =>原文输出；

3.2 CHANGE==1 =>扫描字符串的同时转换字母的大小写（若当前字符为字母），之后输出

4. 进入下一次循环

5. 结束

2）程序清单

#include<stdio.h>

# define CHANGE 0

int main ()

{

unsigned short i , j , n ;

char a[500] , c ;

scanf ( "%hu " , &n ) ;

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

{

fgets ( a , 500 , stdin ) ;

c = a[0] ;

#if ( ! CHANGE ) //CHANGE==0

printf ( "%s" , a ) ;

#else //CHANGE==1

{

for ( j = 0 ; a[j] != 0 ; j ++ )

//对数组各元素进行筛选，若为字母则进行大小写翻转

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

a[j] = a[j] -'a' + 'A' ; //小写转大写

else if ( a[j] >= 'A' && a[j] <='Z' )

a[j] = a[j] -'A' + 'a' ; //大写转小写

printf ( "%s" , a ) ; //字符串格式输出

}

#endif

}"

}

3）测试

（a） 测试数据：

"abcd!@#$efg";

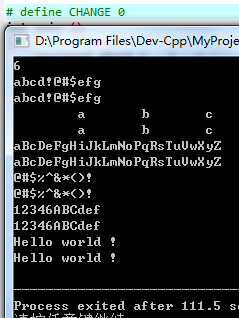
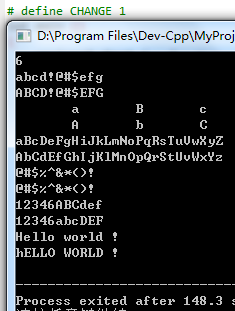
" a b c"; "

"aBcDeFgHiJkLmNoPqRsTuVwXyZ";

"@#$%^&\*()!";

"Hello world !";

（b） 对应测试数据的运行结果截图



## 4.2自设题

**（1）** 自设实验题目：多语句宏定义

**（2）** 实验目的：通过设计实验程序，理解宏定义内语句可以有复数个，从而使宏定义在运用时更加向轻巧型函数靠拢。

**（3）**实验程序：

#include "stdio.h"

# define mul( s1 , s2 , s3 , v ) s1 = k \* w ; s2 = k \* h ; s3 = w \* h ; v = w \* k \* h ;

int main ()

{

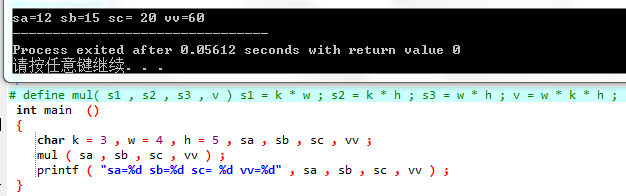
char k = 3 , w = 4 , h = 5 , sa , sb , sc , vv ;

mul ( sa , sb , sc , vv ) ;

printf ( "sa=%d sb=%d sc= %d vv=%d" , sa , sb , sc , vv ) ;

}

**（4）**实验用例：



**（5）**实验结论：对于与语句少且功能简单的函数完全可以用多语句宏定义代替，从而减少参数传递次数与失误可能。

## 4.3小结

编译预处理这一章看起来较为简单，我想这是由于其知识点较为浅显且书写简单，因此应用部分不成问题。但其中include部分其实大有名堂，高级编程语言简洁之处在于系统提供的大量头文件与库函数，而头文件与库函数的实现又是一块可以深挖的部分。同时，编译预处理还同前面的多文件编译以及存储类型等知识相关联，今后我们会经常根据开发的环境与要求编写自己的头文件，变量与函数的跨文件使用也是常有的事，这时候包括include在内的各种编译预处理才会充分显现出其重要性。

在运用宏定义时，绝对不能因为嫌麻烦就去省括号，源程序改错已经说明了编译器只做宏代换，不对代换后的语句优先处理。定义时费一些精力加括号是为了debug时不再被其带来的逻辑错误牵扯时间与精力。

由于开始对条件编译不熟悉，因此在定义CHANGE时并没有意识到#if / #elif 后的常量表达式在预编译时就要有确定值而不能等到运行时依靠外部输入确定，同时 #endif 在条件结构中并没有对应语句，在用条件编译时要注意不能遗漏。

由于assert断言是第一次用，不可避免的要重新扫一遍书。对于这种独立知识点还是要平时有意识去使用，如此才能早日掌握。assert断言配合条件编译与单步调试可提高debug效率，不同点在于单步调试更便于检查变量，而assert断言与条件编译更适合检查表达式与输出问题，三种debug方式的灵活运用需要在今后的编程中不断反复才能掌握。

# 5数组实验

## 5.1必做题

⑴程序改错：1题

下面是用来将数组a中元素按升序排序后输出的源程序。分析源程序中存在的问题，并对源程序进行修改，使之能够正确完成任务。

源程序

1 #include<stdio.h>

2 int main(void)

3 {

4 int a[10] = {27, 13, 5, 32, 23, 3, 17, 43, 55, 39};

5 void sort(int [],int);

6 int i;

7 sort(a[0],10);

8 for(i = 0; i < 10; i++)

9 printf("%6d",a[i]);

10 printf("\n");

11 return 0;

12 }

13 void sort(int b[], int n)

14 {

15 int i, j, t;

16 for (i = 0; i < n - 1; i++)

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

18 if(b[j] < b[j+1])

19 t = b[j], b[j] = b[j+1], b[j+1] = t;

20 }

**解答：**

（1）错误修改：

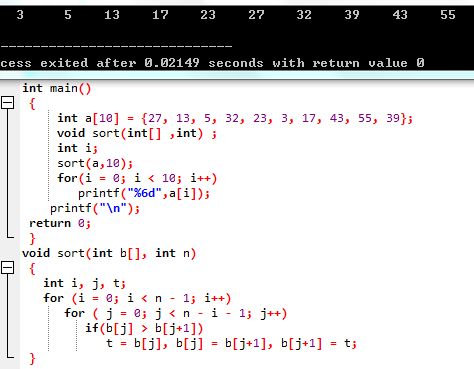
1) 第7行的数组传值形式错误，正确形式为：

sort(a,10);

2) 第18行的函数交换条件错误导致实际按降序排序，正确形式为：

if(b[j] > b[j+1])

（2）错误修改后运行结果：



⑵程序修改替换：2题

1.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。请在源程序中的下划线处填写合适的代码来完善该程序。

源程序：

#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：**

#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 ? a[j-1] : a[i-1] ; // 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

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

}

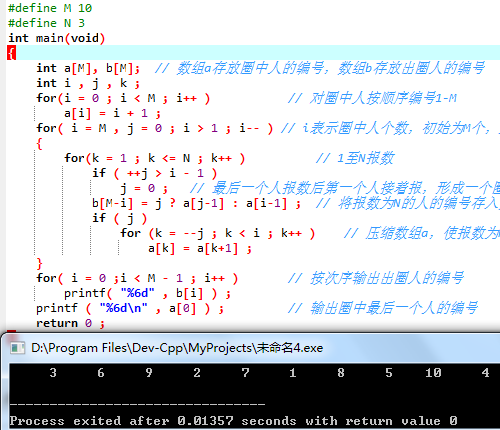
for( i = 0 ;i < M - 1 ; i++ ) // 按次序输出出圈人的编号

printf( "%6d" , b[i] ) ;

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

return 0 ;

}



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。因此，请采用做标记的办法修改（1）中的程序，并使修改后的程序与（1）中的程序具有相同的功能。

#include <stdio.h>

#define M 10

#define N 3

int main()

{

int a[M] = {0} , b[M] ;

int i , j , k ;

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

{

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

{

if ( ++ j > M - 1 )

j = 0 ;

if ( ( a[9] == 1 && j == 0 ) || ( a [j-1] == 1 && j != 0 ) )

k -= 1 ;

}

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

a [j-1] = 1 ;

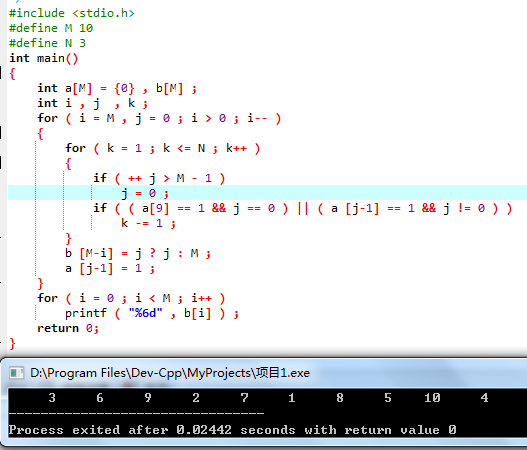
}

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

printf ( "%6d" , b[i] ) ;

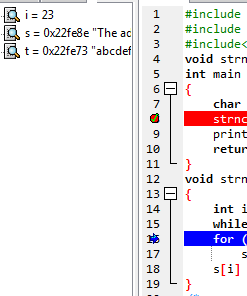
return 0;

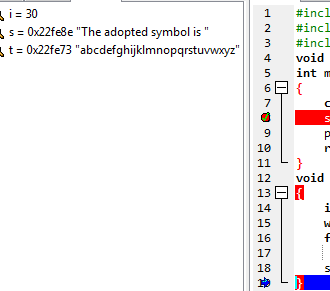
}



⑶跟踪调试：2题

单步执行源程序。进入函数strncat后观察表达式s、t和i。当光条落在for语句所在行时，i为何值？当光条落在strncat函数块结束标记（右花括号 }）所在行时, s、t分别为何值？

进入时

结束时

（2）分析函数出错的原因，排除错误，使函数正确实现功能，最后写出程序的输出结果。

源程序：

#include<stdio.h>

void strncat(char [],char [],int);

int main(void)

{

char a[50]="The adopted symbol is ",b[27]="abcdefghijklmnopqrstuvwxyz";

strncat(a, b, 4);

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

return 0;

}

void strncat(char s[],char t[], int n)

{

int i = 0, j;

while(s[i++]) ;

for(j = 0; j < n && t[j];)

s[i++] = t[j++];

s[i] = '\0';

}

修改后：

#include <stdio.h>

#include <stdlib.h>

void mystrncat ( char[] , char[] , int ) ;

int main ( void )

{

char a[50] = "The adopted symbol is " , b[27] = "abcdefghijklmnopqrstuvwxyz" ;

mystrncat ( a , b , 4 ) ;

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

return 0 ;

}

void mystrncat ( char s[] , char t[] , int n )

{

int i = 0 , j ;

while ( s[i++] ) ;

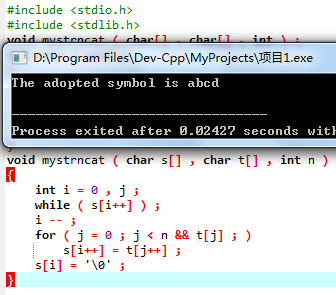
i -- ;

for ( j = 0 ; j < n && t[j] ; )

s[i++] = t[j++] ;

s[i] = '\0' ;

}



⑷程序设计：3题

1.编写一个程序,从键盘读取数据，对一个3×4矩阵进行赋值，求其转置矩阵，然后输出原矩阵和转置矩阵。

**解答：**

1) 解题思路：

1. 定义矩阵a[3][4]；

2. 循环录入矩阵各元素；

3. 输出原矩阵，外循环对行操作，内循环对列操作；

4. 输出转置矩阵，外循环对列操作，内循环对行操作；

5. 结束

2）源程序清单

int main ()

{

int a[3][4] ;

char i , j ;

for ( i = 0 ; i < 3 ; i ++ ) //录入矩阵

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

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

for ( i = 0 ; i < 3 ; i ++ ) //输出行

{

for ( j = 0 ; j < 4 ; j ++ ) //输出列

printf ( "%5d" , a[i][j] ) ;

putchar ( '\n' ) ;

}

putchar ( '\n' ) ;

for ( j = 0 ; j < 4 ; j ++ ) //输出行（原矩阵的列）

{

for ( i = 0 ; i < 3 ; i ++ ) //输出列（原矩阵的行）

printf ( "%5d" , a[i][j] ) ;

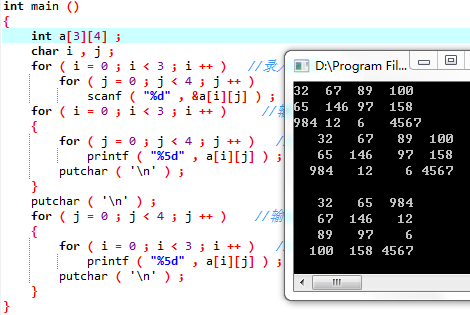
putchar ( '\n' ) ;

}

}

3）测试

对应测试数据的运行结果截图



2.编写一个程序, 其功能要求是：输入一个整数，将它在内存中二进制表示的每一位转换成为对应的数字字符，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) 解题思路：

1. 定义INT为当前机器int型长度，定义逻辑尺mask=1（0x01）；

2. 输入整数个数n；

3. 循环输入各整数ori[0]~ori[n-1]；

4. 数字转字符数组；

4.1 思路：每个待处理数据ori[i]的二进制表示在相应字符数组b[i]内按相同顺序存放，即对每个b[i]从串尾开始存；

4.2 用逻辑尺0x01筛出当前数字的最低位，存入b[i][INT-1]；

4.3 逻辑尺0x01不动，a[i]型左移1位；

4.4 进入下一重循环，晒出的数据存入低一位；

4.5 将当前字节取得的low和high输出

4.6 格式控制

5. 二进制字符数组输出，用计数器控制每八位空一格，且最后一次不空格；

6. 重复4，5直至所有数据处理完成；

7. 结束

2）程序清单

#include<stdio.h>

# include <limits.h>

//# define DEBUG

# define INT ( CHAR\_BIT \* sizeof ( int ) )

//整型二进制长度由机器字长及对int的定义决定

int main ()

{

int ori[100] , mask = 1 ; //mask用于提取原整数各二进制位

char bi[100][INT] , i , count , j ;

unsigned short n ;

# ifdef DEBUG //检查机器字长

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

# endif

scanf ( "%hu" , &n ) ;

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

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

# ifdef DEBUG //检查数据是否输入成功

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

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

# endif

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

{

count = 0 ;

for ( j = INT - 1 ; j >= 0 ; j -- )

{

bi[i][j] = ori[i] & mask ; //逻辑尺筛出二进制最低位

ori[i] >>= 1 ; //原整数左移一位，为下一轮筛选做准备

} //字符数组将原整数的二进制表示正向存储

for ( j = 0 ; j < INT ; j ++ ) //顺序输出字符数组

{

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

count ++ ;

if ( ! ( count % 8 ) ) //调整格式，方便检查输出

putchar ( ' ' ) ;

}

putchar ( '\n' ) ;

}

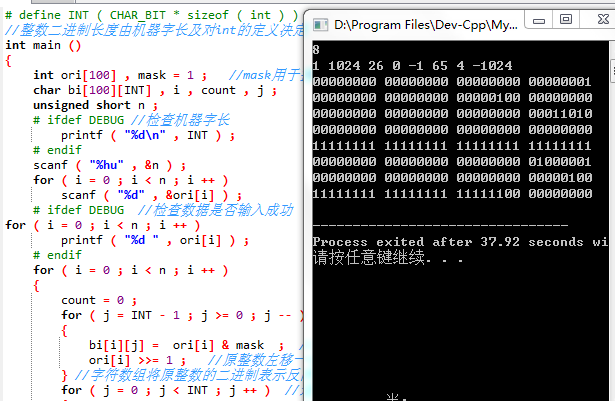
}

3）测试

（a） 测试数据：

n=8; ori[100]={1,1024,26,0,-1,65,4,-1024}

（b） 对应测试数据的运行结果截图



3.编写一个程序, 其功能要求是：输入n个学生的姓名和C语言课程的成绩，将成绩按从高到低的次序排序，姓名同时作相应调整，输出排序后学生的姓名和C语言课程的成绩。然后，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，输出该成绩同学的姓名和C语言课程的成绩；否则输出提示“not found!”。

**解答：**

1) 解题思路：

1.输入成绩组数n；

2.循环输入学生姓名及相应成绩，此时需注意字符输入与数字输入之间的转换；

3.将姓名与成绩两数组传入函数，在函数中用冒泡排序按降序整理学生成绩及相应姓名；

4. 循环输出整理好的学生姓名及成绩；

5. 输入待查成绩组数m;

6. 循环输入待查成绩；

7. 在循环中将每个待查成绩与成绩数组传入第二个函数；

8. 在函数中按二分法查找对应成绩，若找到则返回其下标，未找到返回-1（或下标值域外的任意值）；

9. 若找到待查成绩则输出相应学生姓名及成绩，未找到则输出“Not Found！”；

10. 结束

2）源程序清单

void line ( char name[][20] , int score[] , int n )

{

char cpy[20] ;

int t , i , j ;

for ( i = 0 ; i < n ; i ++ ) //冒泡排序，降序整理

{

\*cpy = '\0' ; //数组初始化

for ( j = n - 1 ; j > i ; j -- )

{

if ( score[j] > score[j-1] )

{

t = score[j] ; //交换成绩

score[j] = score[j-1] ;

score[j-1] = t ;

strcpy ( cpy , name[j] ) ; //交换名字

strcpy ( name[j] , name[j-1] ) ;

strcpy ( name[j-1] , cpy ) ;

}

}

}

}

int bin ( int score[] , int fin , int n ) //二分查找

{

int down = 0 , up = n - 1 , mid ;

while ( down <= up )

{

mid = ( up + down ) / 2 ;

if ( fin < score[mid] ) //在后半区查找

down = mid + 1 ;

else if ( fin > score[mid] ) //在前半区查找

up = mid - 1 ;

else //返回目标值

return mid ;

}

return -1 ; //返回值不能为find数组的下标

}

int main ()

{

char name[100][20] , mid ;

int score[100] , find[100] , i , j , m , n ;

scanf ( "%d " , &n ) ; //输入成绩与名字组数

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

{

j = 0 ;

while ( ( name[i][j] = getchar() ) != ' ' ) //输入名字

j ++ ;

name[i][j] = '\0' ; //将字符数组变为字符串以便整体操作

scanf ( "%d" , &score[i] ) ; //输入对应成绩

getchar () ;

}

line ( name , score , n ) ; //成绩与名字排序

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

{

printf ( "%-20s %d\n" , name[i] , score[i] ) ;

}

putchar ( '\n' ) ;

scanf ( "%d" , &m ) ; //输入待查成绩数

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

{

scanf ( "%d" , &find[i] ) ; //输入待查成绩

}

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

{

mid = bin ( score , find[i] , n ) ; //成绩查找

if ( mid != -1 )

printf( "%-20s %d\n" , name[mid] , score [mid] ) ; //输出目标成绩对应信息

else

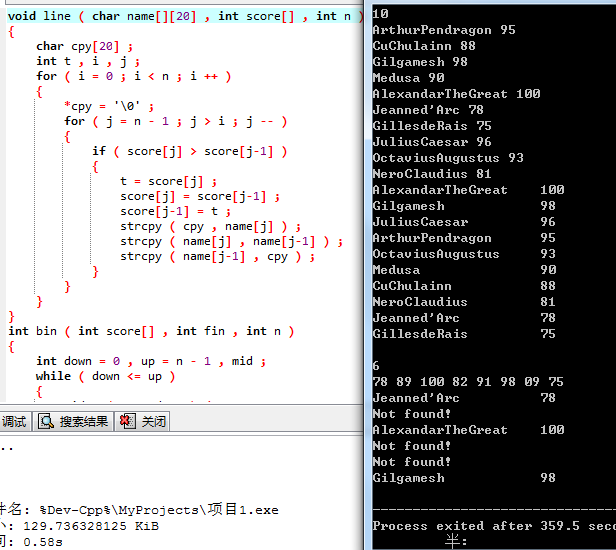
printf( "Not found!\n" ) ; //输出报错

}

}

3）测试

对应测试数据的运行结果截图



## 5.2选做题

程序设计：2题

1.编写函数strnins(s,t,n),其功能为可将字符串t插入字符串s的第n个字符的后面。

**解答：**

1) 解题思路：

1. 输入数组组数time；

2. 读入两个字符串及插入位置；

3. 获取两字符串长度ls和lt；

4. 字符串s从第n个元素开始向后错lt位；

5. 将字符串t插入字符串s内的空档；

6. 输出新字符串长度；

7. 进入下一重循环；

8. 结束

2）程序清单

#include<stdio.h>

# include <string.h>

char i ;

void strnins ( char s[] , char t[] , unsigned short n )

{

unsigned ls , lt ;

char i ,j ;

ls = strlen ( s ) ;

lt = strlen ( t ) ; //获取字符串s，t的长度

s[ls-1] = t[lt-1]= '\0' ; //s,t最后一个元素用0代替

for ( i = ls - 2 ; i >= n ; i -- ) //字符串s从第n个元素开始向后错lt位

s[i+lt-1] = s[i] ;

for ( i = n , j = 0 ; t[j] != '\0' ; i ++ , j ++ ) //将字符串t插入字符串s内的空档

s[i] = t[j] ;

s[ls+lt-2]= '\0' ; //新字符串以'\0'结尾

}

int main ()

{

unsigned short n , time ;

char s[500] , t[500] , blank[500] = "\0" ;

scanf ( "%hu" , &time ) ; //测试数据组数

getchar () ;

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

{

fgets ( s , 500 , stdin ) ;

fgets ( t , 500 , stdin ) ;

scanf ( "%hu" , &n ) ; //字符串t插入位置

getchar () ;

strnins ( s , t , n ) ;

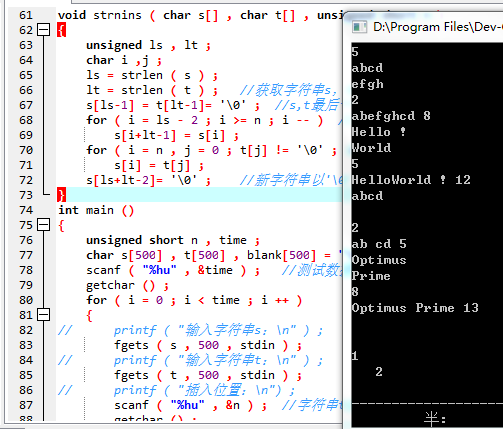
printf ( "%s %u\n" , s , strlen(s) ) ;

}

}

3）测试

对应测试数据的运行结果截图



2.编写一个实现八皇后问题的程序, 即在8\*8方格国际象棋盘上放8个皇后，任意两个皇后不能位于同一行/列/斜线（正或反）上，并输出所有可能的方法。

**解答：**

1) 解题思路：

1. 初始矩阵全部置0；

2. 从第一行开始落子，并以其作为递归基础；

3. 在第二行按规则寻找合适落子点，找到则对应矩阵元素改为1；

4. 按此思路向下递归，若中途碰到无合适落子点的情况则上一行元素重新置0；

5. 在第八行成功落子后输出按输出行矩阵中值为1的元素；

6. 第一行落子向后一格，按类似方法递归；；

7. 结束

2）程序清单

#include<stdio.h>

# include <limits.h>

#define N 8

\_Bool matrix[N + 1][N + 1] = {0} ;

\_Bool IsLegal ( \_Bool matrix[N + 1][N + 1] , const int i , const int j )

{

int m , n ;

// 判断前面的i-1个棋子与matrix[i][j]是否冲突，i为1时合法

for ( m = 1 ; m <= i - 1 ; ++m )

for ( n = 1 ; n <= N ; ++n ) // 核验之前n行的落子，实际每一行只有一个棋子

if ( matrix[m][n] == 1 )

if ( n == j || abs ( i - m ) == abs ( j - n ) ) // 在同一列/斜线上

return false ;

return true ;

}

void Print ( \_Bool matrix[N + 1][N + 1] )

{

static int count = 1 ;

int i , j ;

printf( "Case %d:\n" , count++ ) ;

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

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

if ( matrix[i][j] == 1 )

printf( "%d " , j ) ; //行号由输出顺序给出，故只需给出列号

putchar('\n') ;

}

void Trial ( const int i )

{

// 进入本函数时，在N\*N的棋盘前i-1行已放置了互不攻击的i-1个棋子

// 现从第i行起继续为后续棋子选择合适位置

int j ;

if ( i > N ) // 输出一组可行放法

Print ( matrix ) ;

else

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

{

matrix[i][j] = 1 ; //从当前行第一列开始寻找落子点

if ( IsLegal ( matrix , i , j ) )

Trial ( i + 1 ) ; //当前行成功后进入下一行

matrix[i][j] = 0 ;

}

}

int main ( void )

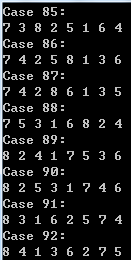
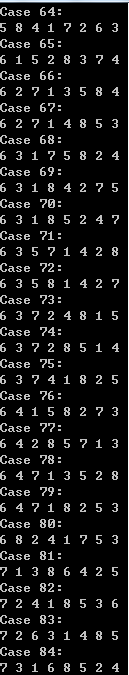
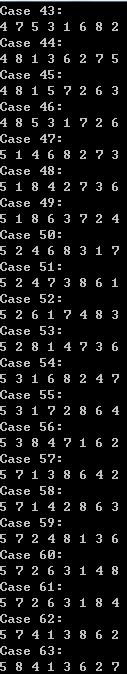
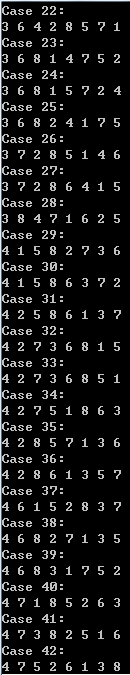
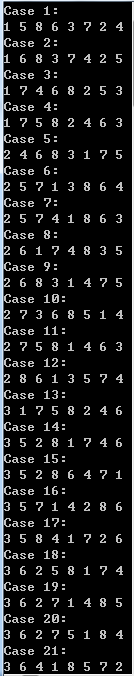
{

Trial ( 1 ) ;

return 0 ;

} 3）测试

对应测试数据的运行结果截图



## 5.3自设题

**（1）** 自设实验题目：计算n的阶乘。

**（2）** 实验目的：通过设计实验程序，熟悉将大整数用字符进行处理的思想方法。

**（3）** 算法步骤：

1. 定义长为100的字符数组array存放阶乘结果；

2. 循环处理输入数据n；

3. 初始化数组array为0；

4. 串头array[0]置为‘1’，此为阶乘基础；

5. 阶乘；

5.1 sc负责存放进位值；

5.2 从i=2开始,数组最低位乘i加上上一次的进位（第一次为0）；

5.3 进位改为sum / 10；

5.4 sum对10取余，新结果存入array[j]；

5.5 重复5.2~5.4，直到完成串尾的运算；

5.6 重复5.2~5.5，直到i取到设定的n值；

6. 长整数在array内逆序存放，因此从高到低输出数组array各元素；

7. 重复3~6，直至n读入文件尾；

8. 结束；

**（4）**实验程序：

#include "stdio.h"

int main ()

{

const unsigned short MAX=100 ;

char i , j , array[MAX];

unsigned short sum , sc , n ;

printf ( "n=" ) ;

while ( scanf ( "%hu" , &n ) != EOF ) //文件尾测试输出

{

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

\* ( array + i ) = 0 ; //每次将数组初始化

\* array = 1 ; //设定阶乘基础

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

{

sc = 0 ; //负责进位

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

{

sum = \* ( array + j ) \* i + sc ;

// 每位上数值与当前被累乘的数相乘，同时加上上一次的进位

sc = sum / 10 ; //存放进位数值

\* ( array + j ) = sum % 10 ;

//将余数存入对应数组元素

}

}

printf ( "%hu!=" , n ) ;

for ( i = MAX - 1 ; i >= 0 ; i -- ) //保存开始输出结果的位置

if ( \* ( array + i ) != 0 )

break ;

for ( j = i ; j >= 0 ; j -- ) //输出结果

printf ( "%u" , \* ( array + j ) ) ;

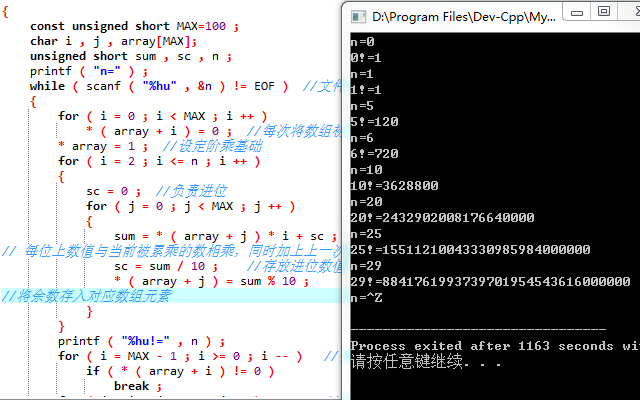
putchar ( '\n' ) ;

printf ( "n=" ) ;

}

}

**（5）**实验用例：0 1 5 6 10 25 29



**（6）**实验结论：超大整数的处理不适合再用数字变量，而应借助字符数组处理

## 5.4小结

可以发现从本次开始程序体渐渐开始变得庞大且功能逐渐复杂，以后无论是做项目还是工作中这都是不可逆转的趋势。这就要求我们选择适合自己的编程思维，无论是每写一个函数紧接着就去实现的顺序架构还是先写出程序主体关注具体函数实现方式的逐层架构，我们都应该提前理清思路，在编程时手要快心要细，节约构造时间与检查时间，从而逐渐提高自己的工作效率。

数组排序时升序与降序的切换极为容易，只需想清楚后项与前项相比大还是小的问题即可，因此虽存在混淆的可能，但只需修改控制条件便可改过来。

约瑟夫问题虽然在上学期接触过，但奈何其实在抽象，这次解决时又费了一番工夫。约瑟夫问题中循环体较为抽象且多次考察自增自减运算符的优先级问题。面对此类问题时首先应沉着冷静，不要被陌生问题所吓倒，其次耐心分析程序的语义，还可以将其落实到自己的笔下，通过一步一步写写画画获得对代码更为直观的感受，长此以往方可在遇见陌生问题时增强对代码的参悟与具象化的能力。

配合将自增自减嵌入下标可有效地减少代码量，但同时也需要对前缀后缀的自增自减的优先级与操作顺序有深刻的把握。尤其要注意分析在数组尾部进行自增操作时下标是否越界，若越界千万不可忘记补上一次自减，否则会出现逻辑错误，并且由于错误本身藏在逻辑之中较为隐蔽，在回来debug时寻找起来极为不便。

矩阵的转置实现起来十分简单，但其提供了一种二维/高维数组非按行操作的途径，但应注意嵌套循环结构的书写顺序以及下标顺序，切莫自己把自己绕了进去。从本次开始，我有意识的在debug时运用了刚刚学过的条件编译，按#ifdef DEBUG的注释与否进行控制，操作十分简便。

查找成绩看似程序庞大复杂，实则两个功能都是对已讲算法的复习。只需注意在编程时详尽理解每一步算法的意义并注意不要产生“笔误”。程序体越庞大，其中隐藏的笔误便会越难找。最后，不能仅仅满足于用书上的算法解决问题，有时间还应该尝试将算法进行替换。

# 6指针实验

## 6.1必做题

⑴程序改错：1题

下面程序是否存在错误？如果存在，原因是什么？如果存在错误，要求在计算机上对这个例子程序进行调试修改，使之能够正确执行。

1 #include "stdio.h"

2 void main(void)

3 {

4 float \*p;

5 scanf("%f",p);

6 printf("%f\n",\*p);

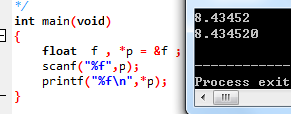
7 }

**解答：**

（1）错误修改：

1) 第4行的指针悬空，正确形式为：

float f , \*p = &f ;

（2）错误修改后运行结果：

⑵程序修改替换：2题

1.下面的程序通过函数指针和菜单选择来调用字符串拷贝函数或字符串连接函数，请在下划线处填写合适的表达式、语句、或代码片段来完善该程序。

#include "stdio.h"

#include "string.h"

void main(void)

{

char a[80],b[80],c[160],\*result=c;

int choice,i;

do{

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

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

printf("\t\t3 exit.\n");

printf("\t\tinput a number (1-3) please!\n");

scanf("%d",&choice);

}while(choice<1 || choice>5);

switch(choice){

case 1:

p=strcpy;

break;

case 2:

p=strcat;

break;

case 3:

goto down;

}

getchar();

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

i=0;

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

i=0;

result= (a,b);

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

down:

;

}

2.为了使程序不受scanf、getchar、gets等函数输入后回车符的影响，请修改第（1）题程序，按要求输出下面结果：（（输入）表示该数据是键盘输入数据）

1 copy string.

2 connect string.

3 exit.

input a number (1-3) please!

2 （输入）

input the first string please!

the more you learn, （输入）

input the second string please!

the more you get. （输入）

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

**解答：**

完善后的程序如下所示：

#include <stdio.h>

#include <string.h>

void main(void)

{

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

char a[80],b[80],c[160],\*result=c;

int choice,i;

do{

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

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

printf("\t\t3 exit.\n");

printf("\t\tinput a number (1-3) please!\n");

scanf("%d",&choice);

}while(choice<1 || choice>5);

switch(choice){

case 1:

p=strcpy;

break;

case 2:

p=strcat;

break;

case 3:

goto down;

}

getchar();

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

i=0;

fgets(a,80,stdin);

a[strlen(a)-1] = '\0' ;

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

i=0;

fgets(b,80,stdin);

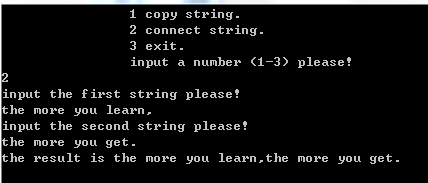
result= p(a,b);

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

down:

;

}



⑶跟踪调试：2题

#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？

进入函数时C:\Users\q\AppData\Roaming\Tencent\Users\479101295\QQ\WinTemp\RichOle\S4TH]H1`AW~}%[`SI6~JD]C.png

返回main时，无s的值。

（2）排除错误，使程序输出结果为：

there is a boat on the lake.

**解答：**

#include <stdio.h>

#include <stdlib.h>

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

int main()

{

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

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

return 0;

}

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

{

char \*cp = s;

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

\*s='\0';

return(cp);

}

（3）选做：由于watch窗口中只显示s所指串的值，不显示s中存储的地址值，怎样才能观察到s值的变化呢？

在strcpy函数的循环中printf出s的地址，在while语句行添加断点或改用Dev-C++

⑷程序设计：4题

1.一个长整型变量占4个字节，其中每个字节又分成高4位和低4位。试从该长整型变量的高字节开始，依次取出每个字节的高4位和低4位并以数字字符的形式进行显示。

**解答：**

1) 解题思路：

1.输入待处理数据个数n;

2.循环录入待处理数据a[0]~a[n-1];

3.指针指向长为4的字符数组的尾部（最高位）;

4.构造逻辑尺0xff，将long形数据分别左移0,8,16,24位并用逻辑尺筛出以获得长度为4的char数组；

5.从a[3]（原数据最高位）开始处理二进制数据。

4.1 用逻辑尺0x0f筛出char型的低四位，存入变量low

4.2 逻辑尺0x0f不动，a[i]型左移四位；

4.3 用逻辑尺筛出char型的高四位，存入变量high

4.4 将当前字节取得的low和high输出

4.5 格式控制

6. 指针左移，开始下一重循环；

7. 结束

2）源程序清单

#include <stdio.h>

void litoc(long a)

{

char ch[4],\*p,low,high,i;

p=&ch[3]; // p指向字符数组最高位

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

ch[i]=(a>>(CHAR\_BIT\*i))&0xff; //将long型a的二进制打散进大小为4的字符数组，从long的低位到高位依次存放

}

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

{

low=(\*p)&0x0f; //对每个字节取低四位

if(low<10)

low=low+'0'; //0~9处理

else

low=low+'A'-10; //A~F处理

high=(\*p>>4)&0x0f; //对每个字节取高四位

if (high<10)

high=high+'0'; //0~9处理

else

high=high+'A'-10; //A~F处理

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

if(i!=3) {

putchar(' ') ; //格式控制，最后一个字符后无空格输出

}

p--; //指针后移

}

putchar('\n');

}

int main()

{

long a[50];

unsigned short n, i;

scanf("%hu", &n) ; //待处理数据个数

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

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

}

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

litoc(a[i]);

}

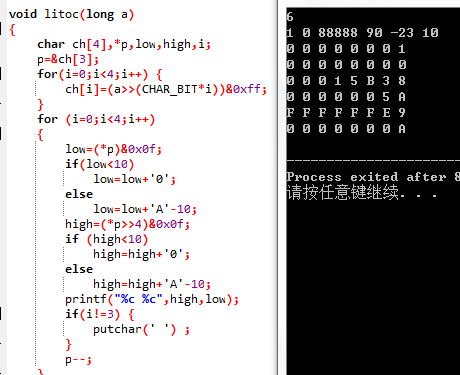
}

3）测试

（a） 测试数据：

1 0 88888 90 -23 10

（b） 对应测试数据的运行结果截图



2.利用大小为n的指针数组指向用gets函数输入的n行，每行不超过80个字符。编写一个函数，它将每一行中连续的多个空格字符压缩为一个空格字符。在调用函数中输出压缩空格后的各行，空行不予输出。

**解答：**

1) 解题思路：

1输入字符串行数n，n为0时退出；

2定义二位字符数组 a[n][N]；

3.fgets函数循环输入各字符串，同时相应地字符指针指向当前字符串；

4.fgets函数循环输入各字符串；

5.字符串末位元素由'\n'置换为'\0'

6.字符串空格处理

6.1 首先对每个字符串，筛查其中的连续空格并记录第二个空格（若存在）的下标k、t；

6.2 循环进行下标后滚，找到其后的第一个非空字符并记录下标k；注意由于自增写在下标内故应先额外做一步k--；

6.3 用非空字符之后的部分字符串从第二个空格开始向前覆盖.

6.4 遍历字符串直至串尾.

6.5 进入下个字符串并按上述步骤处理.

7. 循环检查各字符串长度，若为非空行则正常输出，否则进行屏蔽；

8. 结束

2）程序清单

#define N 80

void change (char \*p[],int n);

int main(void)

{

int i,j,n;

while(scanf("%d",&n)) { //输入字符串行数

if (n == 0) //终止条件

break ;

n=n+1;

char a[n][N],\*p[n];

for(i=0;i<n;i++){ //输入各字符串

fgets(a[i],N,stdin);

p[i]=a[i];

}

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

{

j=strlen(a[i]) ; //字符串末尾置为'\0'

a[i][j-1]='\0';

}

change(p,n);

}

}

void change(char \*p[],int n)

{

int i,j,k,t,l;

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

for(j=0;p[i][j]!='\0';j++){

if(p[i][j]==' '&&p[i][j+1]==' '){ //吞空格

k=j+1; //仅保留一个空格，故记录第二个空格的位置

while(p[i][k++]==' ') ; //记录空格结束位置

k--;

t=j+1;

while(p[i][t++]=p[i][k++]); //非空格字符前移

}

}

if(l=strlen(p[i])>0) //吞掉空行

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

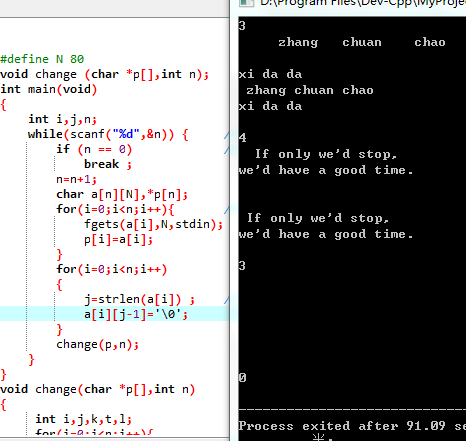
}

putchar('\n'); //格式控制

}

3）测试

对应测试数据的运行结果截图



3.设某个班有N个学生，每个学生修了M门课程（用#define定义N、M）。输入M门课程的名称，然后依次输入N个学生中每个学生所修的M门课程的成绩并且都存放到相应的数组中。编写下列函数：

a.计算每个学生各门课程平均成绩；

b.计算全班每门课程的平均成绩；

c.分别统计低于全班各门课程平均成绩的人数；

d.分别统计全班各门课程不及格的人数和90分以上（含90分）的人数。

在调用函数中输出上面各函数的计算结果。（要求都用指针操作，不得使用下标操作。）

**解答：**

1) 解题思路：

1. 在同一行内用循环读字符的方式输入各课程名称，注意吞掉分隔符并串尾附'\0'；

2. 用循环读字符的方式输入各课程名称，遇' '或'\n'终止，注意串尾附'\0'；

3. scanf循环当前学生对应的各科成绩；

4. std\_ave函数内计算每个学生的平均成绩，同时输出计算结果；

5. 求每门课程的平均成绩，注意外循环按列（学科）操作，内循环按行（学生）操作；

6. below\_ave函数内循环统计低于全班各门课程平均成绩的人数；

6.1 传递参数score, course, subave；.

6.2 定义计数数组count[5]存低于对应学科平均成绩的人数；.

6.3 按列（学科）操作，将各学生对应学科成绩与平均成绩相比较，若学生成绩较低则计数一次；

6.4 输出当前学科计数结果num[i]；.

6.5 进入下一重循环；.

7. fail函数内统计全班各门课程不及格的人数；

7.1 传递参数score, course；.

7.2 剩余操作同6.2~6.5，但判断条件改为学生成绩是否低于60；

8. perfect函数内统计90分以上（含90分）的人数，操作同7.1~7.2，但判断条件改为学生成绩是否高于90；

9. 结束

2）程序清单

#include <stdio.h>

#define N 5

#define M 5

void std\_ave(int \*s, char \*name, double stuave)

{

int sum = 0;

int i;

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

sum += \*(s+i);

stuave = sum / 5.0;

printf("Average score of %s is %.2lf\n", name, stuave);

}

void below\_ave(int s[][M], char sub[][20], double \*subave)

{

int i, j;

int num[5] = { 0 };

for (j = 0; j < N; j++) {

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

if ((\*(\*(s + i) + j)) < \*(subave + j))

(\*(num + j)) ++;

}

}

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

printf("Number of students lower than avg of %s is %d\n", \*(sub + i), \*(num + i));

}

void fail(int s[][M], char sub[][20])

{

int count = 0;

int i, j;

for (j = 0; j <M; j++) {

count = 0;

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

if (\*(\*(s + i) + j) < 60) //分数低于60则计数

count++;

}

printf("Number of students %s fail is %d\n", \*(sub + j), count);

}

}

void perfect(int s[][M], char sub[][20])

{

int count = 0;

int i, j;

for (j = 0; j < M; j++) {

count = 0;

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

if (\*(\*(s + i) + j) >= 90) //分数高于90则计数

count++;

}

printf("Number of students %s perfect is %d\n", \*(sub + j), count);

}

}

int getname(char \*s, int n)

{

char c;

int i = 0;

for (i = 0; i < n - 1 && ((c = getchar()) != EOF) && c != '\n'; i++) //若第n个字符为\n i=n

\*(s + i) = c;

if (c == '\n') {

\*(s + i) = '\0'; //串尾置'\0'

i++;

}

return i - 1;

}

int main()

{

char course[N][20], name[N][10];

char c;

int i, j;

int score[N][M];

double stuave[M], subave[N];

int temp = 0;

for (i = 0; i < N; i++) //课程名称

{

j = 0;

while ((c = getchar()) != ' '&&c != '\n') {

\*(\*(course + i) +j) = c;

j++;

}

if (c == ' ' || c == '\n') {

\*(\*(course + i) + j) = '\0';

}

}

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

getname(\*(name + i), 10); //学生姓名

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

scanf("%d", &score[i][j]); //学生成绩

getchar();

}

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

std\_ave(\*(score + i), \*(name + i), \*(stuave + i)); //每个学生的平均成绩

for (j = 0; j < N; j++) {

temp = 0;

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

temp += \*(\*(score + i) + j); //每门课程的平均成绩

}

\*(subave + j) = temp / 5.0;

printf("Average score of %s is %.2lf\n", \*(course + j), \*(subave + j));

}

below\_ave(score, course, subave); //低于全班各门课程平均成绩的人数

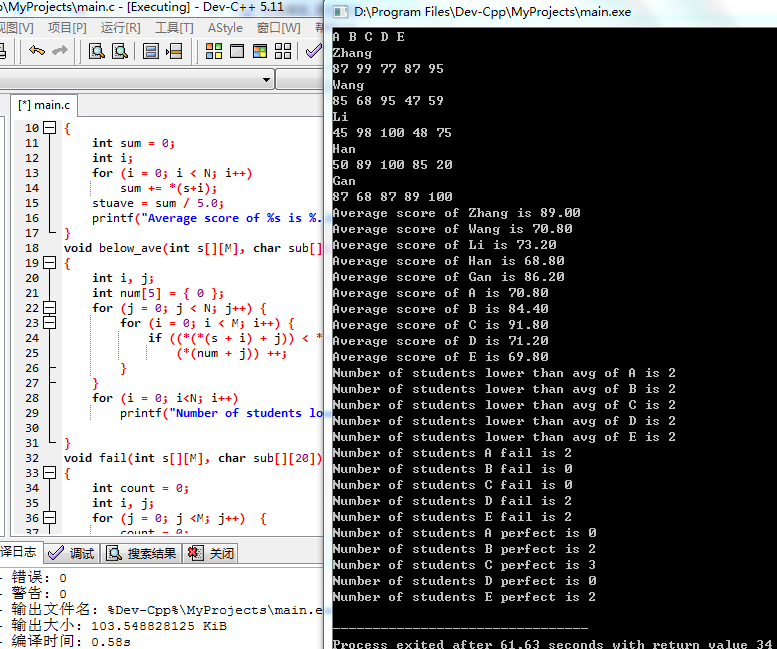
fail(score, course); //各门课程不及格的人数

perfect(score, course); //90分以上（含90分）的人数

}

3）测试

对应测试数据的运行结果截图



4.编写一个程序,输入n个整数，排序后输出。排序的原则由命令行可选参数-d决定，有参数-d时按递减顺序排序，否则按递增顺序排序。要求将排序算法定义成函数，利用指向函数的指针使该函数实现递增或递减排序。

**解答：**

1) 解题思路：

1调用系统环境，输入程序地址（及命令行参数），由此判断排序顺序，具体见题目；

2.若参数个数或第二个参数不为“-d”，输出报错信息并弹出；

3.输入数据个数n；

4.循环录入数组各元素a[0]~a[n-1]；

6.将数组、元素个数、标记值传入sort函数进行排序；

7.用选择排序整理数组，排序方式由是否有“-d”决定的标记值而决定；

8. 输出排序后的数组；

9. 结束

2）程序清单

#include<stdio.h>

void sort(int a[], unsigned short x, char flag)

{

int temp ;

char i, j;

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

{

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

{

if(flag ? (a[j] < a[j+1] ) : (a[j] > a[j+1])) //判断条件为命令行参数决定的flag值

{

temp = a[j] ;

a[j] = a[j+1] ;

a[j+1] = temp ;

}

}

}

}

int main(int argc,char \*argv[]) //main参数由系统给出

{

unsigned short n ;

char j = 1, i;

if(argc > 2) //参数个数不符

{

printf("Error!\n");

return -1 ;

}

else if(argc == 2) //参数内容不符

{

if(\*argv[1] != '-' || \*(argv[1] + 1) != 'd')

printf("Error!\n");

return -1 ;

}

else j = 0 ;

int array[50] ;

scanf("%hu", &n) ; //确定数据组数

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

scanf("%d", &array[i]); //输入数组

sort(array, n, j); //分类排序

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

{

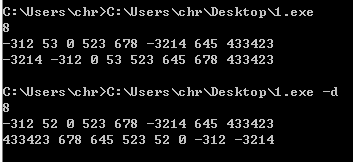
printf("%d ",array[i]);

}

}

3）测试

对应测试数据的运行结果截图



## 6.2选做题

程序设计：2题

1.设有N位整数和M位小数（N=20，M=10）的数据a,b。编程计算a+b并输出结果。

如：12345678912345678912.1234567891 + 98765432109876543210.0123456789

**解答：**

1) 解题思路：

1. 定义整数部分长度M与小数部分长度N（注：此处与题目要求相反）；

2 定义a, b, c三个超长整数的整数部分与小数部分字符数组，长度为M/N+2是由于一位存放'\0'，c的另一位存放进位，a, b的另一位存放读入时后缀回车符；

3. 输入a,的整数部分；

3.1 fgets函数读入a的整数部分；

3.2 用获取其长度lia,串尾回车符用'\0'代替；

3.3 将长整数在数组内贴高位对齐以便于后续计算；

3.4 遍历指针找到数字部分的最高位；

3.5 字符串未填满的部分（对整数部分而言为高位）用'0'填充便于后续计算；

4. 输入a,的小数部分；

4.1 fgets函数读入a的小数部分；

4.2 用获取其长度lna；

4.3 字符串未填满的部分（对小数部分而言为低位）用'0'填充便于后续计算；

5. 输入b的正数部分，方法同3.1-3.5，长度存入lib；

6. 输入b的小数部分，方法同4.1-4.3，长度存入lnb；

7. 定义DEBUG下，输入统一格式后的超长整数a, b便于检查；

8. a, b小数部分相加（adnum函数实现）；

8.1 lna, lnb中较大的一个作为numc的串长，记为lnc；

8.2 和的小数部分数组numc串尾置'\0'；

8.3 从lnc给出的最低位开始将numa, numb相应位置的数字转成整形，与上一次的save（第一次save为0）相加，结果存入sum；

8.4 判断sum与10的大小关系，若小于10，则sum转为字符数字存入numc的相应数位，否则先做sum-10再执行后续操作，进位存入save；

8.5 至小数部分最高位结束，剩余的进位save传回main函数；

8.6 定义DEBUG下，输出c的小数部分便于检查；

9. a, b整数部分相加并调整格式（adint函数实现）；

9.1 adnum函数传回的进位同a, b的整数部分一起传入adint；

9.2 lia, lib中较大的一个作为intc的可能串长（考虑进位），记为lic；

9.3 和的整数部分数组intc串尾置'\0'；

9.4 a, b的整数逐位相加方法同8.3-8.4

9.5 至整数部分最高位（lic）结束；

9.6 最高位（串头）改为最后一次save的符号化，同时也是考虑a, b至少有一个整数部分占满M位的特殊情况；

9.7 定义DEBUG下，输出c的整数部分便于检查；

9.8 格式处理；

9.8.1 指针指向numc的最高位（靠近串头）；

9.8.3 字符串整体左移（包括'\0'），使c的整数部分在intc内左对齐存放；

9.9 定义DEBUG下，输出c格式处理后的整数部分便于检查；

10. 分别输出超长整数c的整数部分与小数部分；

11. 结束

2）程序清单

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define M 20

#define N 10

//#define DEBUG

unsigned short lia, lib, lna, lnb, lic, lnc ;

void get(char \*inta, char \*intb, char \*numa, char \*numb) /\*输入浮点数a,b并统一存放格式\*/

{

char i, \*p ;

printf("input the integar part of a :") ;

fgets(inta, M+2, stdin) ;

lia = strlen(inta)-1 ; /\*获取整数部分长度\*/

\*(inta + M) = '\0' ; /\*末位置'\0'\*/

/\*数字字符串在数组内右对齐\*/

for (i = lia - 1, p = inta + M - 1; i > -1; i --, \*p --) {

\*p = \*(inta + i) ;

}

/\*非数字部分用0填充\*/

for(i = 0; i < M - lia; i ++) {

\*(inta + i) = '0' ;

}

printf("input the numeric part of a :") ;

fgets(numa, N+2, stdin) ;

lna = strlen(numa)-1 ; /\*获取小数部分长度\*/

for (i = lna; i < N; i ++) { /\*非数字部分用0填充\*/

\*(numa + i) = '0' ;

}

\*(numa + N) = '\0' ;

printf("input the integar part of b :") ;

fgets(intb, M+2, stdin) ;

lib = strlen(intb)-1 ;

\*(intb + M) = '\0' ;

for (i = lib - 1, p = intb + M - 1; i > -1; i --, \*p --) {

\*p = \*(intb +i) ;

}

for(i = 0; i < M - lib; i ++) {

intb[i] = '0' ;

}

printf("input the numeric part of b :") ;

fgets(numb, N+2, stdin) ;

lnb = strlen(numb)-1 ;

for (i = lnb; i < N; i ++) {

\*(numb + i) = '0' ;

}

\*(numb + N) = '\0' ;

# ifdef DEBUG

printf("%s.%s\n%s.%s\n", inta, numa, intb, numb) ;

# endif

}

char adnum(char \*numa, char\* numb, char \*numc)

{

char save = 0, i, sum ;

lnc = lna > lnb ? lna : lnb ; /\*获取待加数字串长\*/

\*(numc + lnc) = '\0' ; /\*串尾置'\0'\*/

for(i = lnc - 1; i >= 0 ; i --) {

sum = \*(numa + i) - '0' + \*(numb + i) - '0' + save ; /\*字符串同位数字转换成数字再相加\*/

if(sum >= 10) { /\*将sum的个位数存入sumc中相同的位置，进位赋给save\*/

\*(numc + i) = sum - 10 + '0' ;

save = 1 ;

}

else {

\*(numc + i) = sum + '0' ;

save = 0 ;

}

}

# ifdef DEBUG

printf("%s", numc) ;

# endif

return save ; /\*将小数部分最高位的进位返回\*/

}

void adint(char \*inta, char\* intb, char \*intc, char get)

{

char save = get, i, sum, \*p, add ;

lic = lia > lib ? lia : lib ; /\*获取待加数字串长\*/

\*(intc + M + 1) = '\0' ; /\* 串尾（最后一个元素)置'\0' \*/

for(i = M - 1; i >= 0 ; i --) { /\*操作与小数部分相加时相同\*/

sum = \*(inta + i) - '0' + \*(intb + i) - '0' + save ;

if(sum >= 10) {

\*(intc + i + 1) = sum - 10 + '0' ;

save = 1 ;

}

else {

\*(intc + i + 1) = sum + '0' ;

save = 0 ;

}

\*intc = save + '0' ;

if(i == M + 1 - lic) { /\*获取数字串部分最后的进位\*/

add = save ;

}

}

# ifdef DEBUG

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

# endif

p = &intc[M+1-(lic+add)] ; /\*格式处理，吞掉数字串高位占位的0\*/

for (i = 0; p < intc + M + 2; i ++, p ++) {

\*(intc + i) = \*p ;

}

# ifdef DEBUG

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

# endif

}

int main()

{

/\*int存放整数部分，num存放实数部分\*/

char inta[M+2], numa[N+2], intb[M+2], numb[N+2], intc[M+2], numc[N+2], i, j, extra ;

i = 0 ;

get(inta, intb, numa, numb) ; /\*输入浮点数a,b并统一存放格式\*/

extra = adnum(numa, numb, numc) ; /\*小数部分相加，返回进位\*/

adint(inta, intb, intc, extra) ; /\*整数部分相加\*/

printf("the sum of a and b is %s.%s\n", intc, numc) ;

}

3）测试

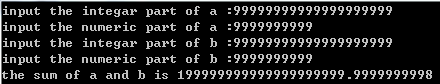
（a） 测试数据：

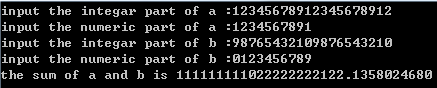
①a=99999999999999999999.9999999999; b=99999999999999999999.9999999999

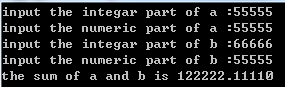
②a=12345678912345678912.1234567891; b=98765432109876543210.0123456789

③a=55555.55555; b=66666.55555

（b） 对应测试数据的运行结果截图







2.编写使用复杂声明char \*(\*p[2])(const char \*,const char \*);的程序。

提示：p中元素可为strcmp、strstr等函数名。

**解答：**

1) 解题思路：

1. 定义字符串比较函数tostrcmp，判断子串函数tostrcat，两函数均返回对应的库函数的指针，注意头文件<string.h>的strcmp返回值为int型，因此需用 (char \*)做强制转换；

2. 声明函数指针数组char \*(\*p[2])(const char \*,const char \*)并分别初始化为tostrcmp与tostrcat；

3. 用fgets输入两字符串，注意要清掉串尾的换行符，以防在比较时造成干扰；

4. 输入调用函数的选择n；

5.用（char \*）re存被调函数返回的指针值；

6. switch结构输出信息；

6.1 n为0，调用tostrcmp,根据返回的re值判断两串是否相同并决定输出信息；

6.2 n为1，调用tostrstr,根据返回的re值判断b是否为a的子串并决定输出信息；

7. 结束

2）程序清单

#include <stdio.h>

#include <string.h>

char \*tostrcmp(const char \*a, const char \*b) /\*调用系统的strcmp\*/

{

return (char \*)strcmp(a,b) ;

}

char \*tostrstr(const char \*s, const char \*t) /\*调用系统的strstr\*/

{

return strstr(s,t) ;

}

char \*(\*p[2])(const char \*, const char \*) ;

int main()

{

char a[100], b[100], \*re ;

unsigned short n ;

p[0] = tostrcmp;

p[1] = tostrstr;

fgets(a, 50, stdin) ;

fgets(b, 50, stdin) ;

a[strlen(a)-1] = b[strlen(b)-1] = 0 ; /\*消掉串尾换行符\*/

scanf("%hu", &n) ; /\*函数调用选择\*/

re = p[n](a, b) ;

switch(n) {

case 0:

if(!re) {

printf("a and b are the same!\n") ;

}

else {

printf("a and b are different!\n") ;

}

break ;

case 1:

if(re)

printf("b is included in a\n") ;

else

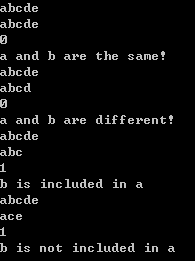
printf("b is not included in a\n") ;

}

}

3）测试

对应测试数据的运行结果截图



## 6.3自设题

**（1）** 自设实验题目：函数指针作为函数参数实现四则运算

**（2）** 实验目的：通过设计实验程序练习函数指针的使用。

**（3）** 算法说明：

1.设计目录函数float mathfunc(float(\*p)(float, float), float para1, float para2) ；注意要将函数声明与变量作为参数，且返回值为加减乘除中某一函数；

2.按四则运算设计add（加）,min（减），tim（乘），div（除）四函数；

3.定义两float型变量a , b，注意因为有除法存在因此均不能为0；

4.分别输出四则运算结果，注意函数书写对应目录函数的格式；

5.结束

**（4）** 实验程序：

/\*函数名作为参数进行传递\*/

float mathfunc(float(\*p)(float, float), float para1, float para2) {

return(\*p)(para1, para2) ; /\*返回函数\*/

}

float add(float f1, float f2) {

return f1 + f2 ;

}

float min(float f1, float f2) {

return f1 - f2 ;

}

float tim(float f1, float f2) {

return f1 \* f2 ;

}

float div(float f1, float f2) {

return f1 / f2 ;

}

int main () {

float a = 1.5, b = 2.5 ;

printf("a+b=%f\n", mathfunc(add, a, b)) ;

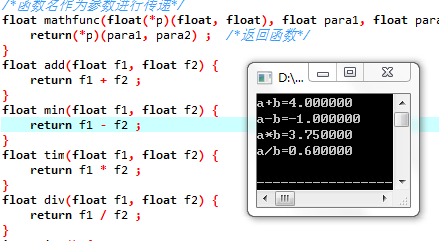
printf("a-b=%f\n", mathfunc(min, a, b)) ;

printf("a\*b=%f\n", mathfunc(tim, a, b)) ;

printf("a/b=%f\n", mathfunc(div, a, b)) ;

}

**（5）**实验用例：a=1.5, b=2.5



**（6）**实验结论：函数名称作为参数传递提供了另一条函数重载的思路。

## 6.4小结

本次可明显感觉到任务开始变难了，而这种“难”是由两方面引起的。一是这次的试验赶上了13,14周的三门考试，复习的压力会让人想尽快从平时的作业中脱身准备考试；二是知识点本身难度加大，首先指针的诸般应用在第一次上手时显得较困难，在编程时有更大几率出错且一时半会找错误的位置和原因；其次本次实验代码量较大，再加上可能出现的错误百出的情况容易让人产生烦躁感，甚至形成正反馈。

最基础的指针定义声明也是容易出岔子的。要注意声明同时初始化与先声明再初始化的语句写法不同，相差一个指针说明符“\*”。本次的空格处理又进一步，再多行语句同时存储的同时增加了屏蔽空白行的要求。此处有两种选择：一是在内存中直接删除这一空白行并让后续语段行依次前移，二是判断当前行是空白行时不予输出。显然后一种实现起来容易不少，而且在程序体基本不小消耗内存的情况下也完全可行，今后编程时要注意这种方法。

这一次卡住我的是录入成绩以及选做题的两道题。

成绩录入中涉及到字符串与数字的混合输入，这就涉及到对字符串逐字循环输入、scanf函数输入以及fgets函数输入的理解与区分。当然很不凑巧的是由于之前没有对这几种输入方法区分清楚，因此在数据录入环节我便耽误了不少时间。当然经过不断的试错与比较，这个问题终于得到了解决。此外本次函数参数传值的多样性大幅提高，也有数组指针表示的集中运用，单体代码量更是又上一个台阶，想心平气和地把一个个问题搞清楚也是十分考验功力的。

计算器除了本身的代码量，由于要用字符串存数字也存在着输入时区分数字和其他字符的问题，此外还有各种各样的类似进位、字符转数字做加法、数位对齐等各种细节。其实单纯完成功能并不需要很多代码，但更进一步的格式要求例如删除字符串中占位的‘0’以及小数点后进位后末位零的去留（本次程序中暂无此功能）才是拉高代码量的“罪魁祸首”。函数指针的声明则是单纯难在其正确声明与运用上，程序本身则难度不高。

鉴于指针在今后的编程中可谓是常客，熟练掌握并运用指针的若干知识十分必要，但光是这一次实验的题是远远不够的，还是需要课下多垒代码。

# 7结构与联合实验

## 7.1必做题

⑴表达式求值与程序验证：1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)

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **表达式** | **计算值** | **验证值** |
| 1 | (++p)->x | 100 | 100 |
| 2 | p++,p->c | (地址),B | 00403010,A |
| 3 | \*p++->t,\*p->t | U, x | U, U |
| 4 | \*(++p)->t | x | x |
| 5 | \*++p->t | V | V |
| 6 | ++\*p->t | V | V |

⑵程序修改替换：2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。

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。

源程序如下：

#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 \*/

}

**解答：**

原程序无输出结果，替换后的程序如下所示：

struct s\_list{

int data; /\* 数据域 \*/

struct s\_list \*next; /\* 指针域 \*/

} ;

struct s\_list \*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 \*/

p = 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");

}

struct s\_list \*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 \*/

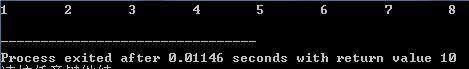
}

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

return headp ;

}

程序运行结果如下，程序正确。



（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：**

替换后的程序如下所示：

struct s\_list{

int data;

struct s\_list \*next;

};

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

int main()

{

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

int s[]={1,2,3,4,5,6,7,8,0};

creat\_list(&head,s);

p=head;

while(p){

printf("%d\t",p->data);

p=p->next; }

printf("\n");

return 0;

}

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

{

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

struct s\_list\*temp;

if(p[0]==0) ;

else {

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

loc\_head->data=\*p++;

tail=loc\_head;

while(\*p){

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

temp->next=loc\_head;

loc\_head=temp;

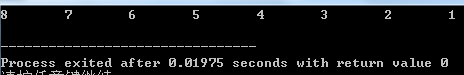
loc\_head->data = \*p++; }

tail->next=NULL; }

\*headp=loc\_head;

}

程序运行结果如下，程序正确。



⑶程序设计：2题

1.设计一个字段结构struct bits，它将一个8位无符号字节从最低位向最高位声明为8个字段，各字段依次为bit0, bit1, …, bit7，且bit0的优先级最高。同时设计8个函数，第i个函数以biti(i=0,1,2,…,7)为参数，并且在函数体内输出biti的值。将8个函数的名字存入一个函数指针数组p\_fun。如果bit0为1，调用p\_fun[0]指向的函数。如果struct bits中有多位为1，则根据优先级从高到低依次调用函数指针数组p\_fun中相应元素指向的函数。8个函数中的第0个函数可以设计为：

void f0(struct bits b)

{

Printf(“the function %d is called!\n”,b);

}

**解答：**

1) 解题思路：

1.设计字段结构struct ISR\_BITS，内含unsigned类型bit0-bit7（各占1bit）与rsv8（对齐unsigned short 的高位）并声明变量bit；

2.设计联合结构union ISR\_REG，内含unsigned short 型变量a及struct ISR\_BITS型变量bit；

3.定义void型函数f0-f8用于输出信息；

4.定义长为8的函数指针数组并逐一赋值为f0-f8；

5.输入待处理数据个数n；

6.循环输入各待处理数据；

7.对每个数据进行处理；

7.1 将数组存入联合结构的unsigned short型；

7.2 用struct结构访问并处理union内的数据；

7.3 判断bit0-bit8是否为1，若为1则进入对应输出函数；

8.结束

2）源程序清单

struct ISR\_BITS { //字段结构

unsigned bit0: 1, bit1: 1, bit2: 1, bit3: 1, bit4: 1, bit5: 1, bit6: 1, bit7: 1, rsv: 8 ;

}bit;

union ISR\_REG { //字段结构间接读入

unsigned short a ;

struct ISR\_BITS bit ;

}temp;

//输出函数组

void f0(struct ISR\_BITS b) {

printf("the 1 function %d is called!\n", b.bit0); }

void f1(struct ISR\_BITS b) {

printf("the 2 function %d is called!\n", b.bit1);}

void f2(struct ISR\_BITS b) {

printf("the 3 function %d is called!\n", b.bit2); }

void f3(struct ISR\_BITS b){

printf("the 4 function %d is called!\n", b.bit3);}

void f4(struct ISR\_BITS b){

printf("the 5 function %d is called!\n", b.bit4);}

void f5(struct ISR\_BITS b){

printf("the 6 function %d is called!\n", b.bit5);}

void f6(struct ISR\_BITS b){

printf("the 7 function %d is called!\n", b.bit6);}

void f7(struct ISR\_BITS b){

printf("the 8 function %d is called!\n", b.bit7);}

int main()

{

void (\*p\_fun[8])(struct ISR\_BITS b);

unsigned short n, num[50] ;

char i ;

scanf("%hu", &n) ;

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

scanf("%hu", &num[i]) ;

}

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

printf("%hu:\n", num[i]) ;

temp.a = num[i] ;

p\_fun[0] = f0; //函数指针赋值

p\_fun[1] = f1;

p\_fun[2] = f2;

p\_fun[3] = f3;

p\_fun[4] = f4;

p\_fun[5] = f5;

p\_fun[6] = f6;

p\_fun[7] = f7;

if(temp.bit.bit0) p\_fun[0](temp.bit); //函数指针间访

if(temp.bit.bit1) p\_fun[1](temp.bit);

if(temp.bit.bit2) p\_fun[2](temp.bit);

if(temp.bit.bit3) p\_fun[3](temp.bit);

if(temp.bit.bit4) p\_fun[4](temp.bit);

if(temp.bit.bit5) p\_fun[5](temp.bit);

if(temp.bit.bit6) p\_fun[6](temp.bit);

if(temp.bit.bit7) p\_fun[7](temp.bit);

putchar('\n') ;

}

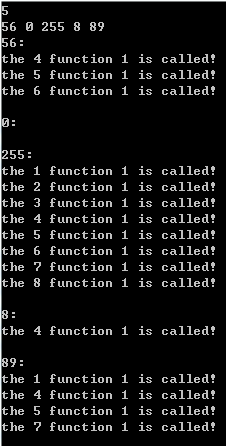
}

3）测试

（a） 测试数据：

56； 0； 255； 8； 89

（b） 对应测试数据的运行结果截图



2.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。用函数编程实现下列功能：

(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。

（与选做题第一，二题合做）

**解答：**

1) 解题思路：

1.定义学生信息结构struct student，包含num[15]（学号），name[20]（姓名），eng（英语成绩），cal（数学成绩），phy（物理成绩），cpp（C语言成绩），sum（总成绩），avg（平均成绩），\*next（自引用），同时创建对应遍历指针\*head,\*tail，指向链表头；

2.定义成绩修改结构struct alter，包含num15（学号），sub[8]（待修改科目），score（新成绩），\*next（自引用），同时创建对应遍历指针\*fore,\*aft，指向链表头；

3.定义表头结构struct head，包含id[3]（ID），name[5]（name），eng[8]（English）,cal[5]（Maths），phy[8]（Physics），c[2]（C），sum[4]（SUM），avg[4]（AVG）；

4.定义记录条数n，；

5.输入记录条数n；

6.循环录入各条struct student记录

6.1 scanf函数输入结构内各成员，注意吞掉回车符；

6.2 向后创建一节新链表，地点为当前链表节自引用指针所指；

6.3 新链表节的自引用指针置空；

7.输出表头，包含学号、姓名、各科成绩；

8循环输出各条记录，同时遍历指针指向下一节链表，注意退出循环后加一空行；

9.输入待修改科目数m；

10.循环录入各条struct alter记录，方法及注意事项同6.1-6.3；

11.遍历指针tail,aft回指各自对应的链表头；

12.将新成绩记录与原纪录循环比对并录入；

12.1 将遍历指针当前所指新成绩，struct student头指针与表头指针\*head传给比对函数amend(struct student \*idv, struct alter \*stu, struct title \*head);

12.2 将struct alter的学号与struct student的学号逐一比对，若匹配则进入12.3，不匹配则进入12.5 ；

12.3 将struct alter的sub与struct title的各科名称逐一比对，若匹配则进入12.4，不匹配则进入12.5 ；

12.4 用struct alter的新成绩替换struct student的相应的旧成绩（通过struct title成员名与struct student成员名的一一对应实现）；

12.5. 比对/修改结束，退出函数；

12.6. 遍历指针后移；

12.7 重复12.1-12.5直至遍历指针指空；

13.同7；

14.遍历指针tail回指对应的链表头；

15.统计成绩；

15.1 tail与表头指针h传给成绩统计函数stat；

15.2 输出表头，包含学号、姓名、总成绩、平均成绩；

15.3 循环求各个学生的总成绩与平均成绩并输出，按6.2，6.3向后递推；

16.输出更新后的各条记录，注意附带表头及格式控制；

17.同14；

18-0.按平均成绩排序（交换指针域）；

18-1.按平均成绩排序（交换指针域）；

19. 同15.2；

20. 循环输出排序后各学生信息，包含学号、姓名、平均成绩，按6.2，6.3向后递推；；

21. 结束

2）程序清单

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct student { //学号+姓名+各科成绩+总成绩+平均成绩

char num[15], name[20] ;

float eng, cal, phy, cpp, sum, avg;

struct student \*next ;

};

struct alter { //学号+待修改科目+新成绩

char num[15], sub[8] ;

float score ;

struct alter \*next ;

};

struct title { //表头

char id[3], name[5], eng[8], cal[5], phy[8], c[2], sum[4], avg[4] ;

}head={"ID", "Name", "English", "Math", "Physics", "C", "SUM", "AVG"}, \*h = &head ;

unsigned short n ;

//按学号与科目修改成绩

void amend(struct student \*idv, struct alter \*stu, struct title \*head)

{

while(idv) {

if(!strcmp(idv->num, stu->num)) { //学号判断

if(!strcmp(head->eng, stu->sub)) { //科目判断

idv->eng = stu->score ;

}

else if (!strcmp(head->cal, stu->sub)) {

idv->cal = stu->score ;

}

else if (!strcmp(head->phy, stu->sub)) {

idv->phy = stu->score ;

}

else if (!strcmp(head->c, stu->sub)) {

idv->cpp = stu->score ;

}

}

idv = idv->next ;

}

}

void stat(struct student \*stu, struct title \*head) //统计总成绩与平均成绩

{

struct student \*save = stu ;

printf("\nSumAndAvg:\n%-15s%-20s%-10s%-10s\n", head->id, head->name, head->sum, head->avg) ;

while(stu->next) {

stu->sum = stu->cal + stu->cpp + stu->eng + stu->phy ;

stu->avg = stu->sum / 4 ;

printf("%-15s%-20s%-10.2f%-10.2f\n", stu->num, stu->name, stu->sum, stu->avg) ;

stu = stu->next ;

}

}

void sort(struct student \*stu, struct title \*head) //按平均成绩排序（交换数据域）

{

struct student \*p1 = stu, \*p2 ;

char i, j, temp[20] ="";

unsigned short len = 0 ;

printf("\nSort:\n%-15s%-20s%-10s\n", head->id, head->name, head->avg) ;

float t ;

while(p1) {

len ++ ;

p1 = p1->next ;

}

for(i = 0, p1 = stu; i < len - 1; i ++, p1=p1->next ) {

for(j = i + 1, p2 = p1->next; j < len - 1; j ++, p2 = p2->next) {

if(p1->avg > p2->avg) {

t = p1->avg ; p1->avg = p2->avg ; p2->avg = t ;

strcpy(temp, p1->num) ; strcpy(p1->num, p2->num) ; strcpy(p2->num, temp) ;

strcpy(temp, p1->name) ; strcpy(p1->name, p2->name) ;strcpy(p2->name, temp) ;

}

}

}

}

void sort(struct student \*\*stu, struct title \*head) //按平均成绩排序（交换指针域）

{

struct student \*prior1, \*prior2, \*p1, \*p2, \*t ;

char i = 0 ;

printf("\nSort:\n%-15s%-20s%-10s\n", head->id, head->name, head->avg) ;

p1 = (struct student \*)malloc(sizeof(struct student)) ;

p1->next = \*stu ;

(\*stu) = prior1 = p1 ;

for(p1 = prior1->next; p1->next !=NULL; prior1 = p1, p1 = p1->next ) {

for(p2 = p1->next, prior2 = p1; p2 !=NULL; prior2 = p2, p2 = p2->next) {

if(p1->avg > p2->avg) {

t = p2->next ;

prior1->next = p2 ;

prior2->next = p1 ;

p2->next = p1->next ;

p1->next = t ;

t = p1 ;

p1 = p2 ;

p2 = t ;

}

}

}

p1 = (\*stu) ;

(\*stu) = (\*stu)->next ;

free(p1) ;

}

int main ()

{

char i, j ;

unsigned short m ;

struct student ori, \*head = &ori, \*tail = &ori;

scanf("%hu", &n) ;

for(i = 0; i < n; i ++) { //输入记录

scanf("%s %s", tail->num, tail->name) ;

scanf("%f %f %f %f", &tail->eng, &tail->cal, &tail->phy, &tail->cpp) ;

getchar() ;

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

tail = tail->next ;

}

tail->next = NULL ;

printf("%-15s%-20s%-10s%-10s%-10s%-10s\n", h->id, h->name, h->eng, h->cal, h->phy, h->c) ;

tail = &ori ;

while(tail->next) {

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n", tail->num, tail->name, tail->eng, tail->cal, tail->phy, tail->cpp) ;

tail = tail->next ;

}

putchar('\n') ;

scanf("%hu", &m) ;

struct alter top, \*fore = &top, \*aft = &top ;

for(i = 0; i < m; i ++) { //输入新成绩

scanf("%s %s", aft->num, aft->sub) ;

scanf("%f", &aft->score) ;

getchar() ;

aft->next = (struct alter \*)malloc(sizeof(struct alter)) ;

aft = aft->next ;

aft->next = NULL ;

}

aft = &top ;

tail = &ori ;

while(aft) {

tail = &ori ;

amend(tail, aft, h) ;

aft = aft->next ;

}

printf("Alter:\n%-15s%-20s%-10s%-10s%-10s%-10s\n", h->id, h->name, h->eng, h->cal, h->phy, h->c) ;

while(tail->next) {

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n", tail->num, tail->name, tail->eng, tail->cal, tail->phy, tail->cpp) ;

tail = tail->next ;

}

tail = &ori ;

stat(tail, h) ;

tail = &ori ;

sort(tail, h) ;

while(tail->next) {

printf("%-15s%-20s%-10.2f\n", tail->num, tail->name, tail->avg) ;

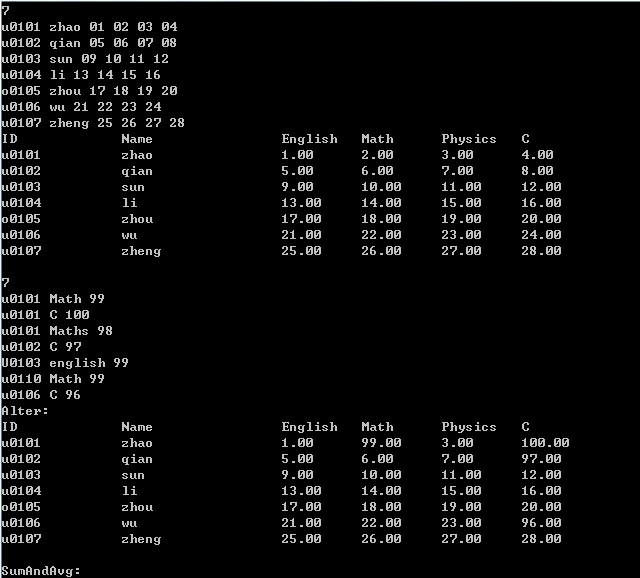
tail = tail->next ;

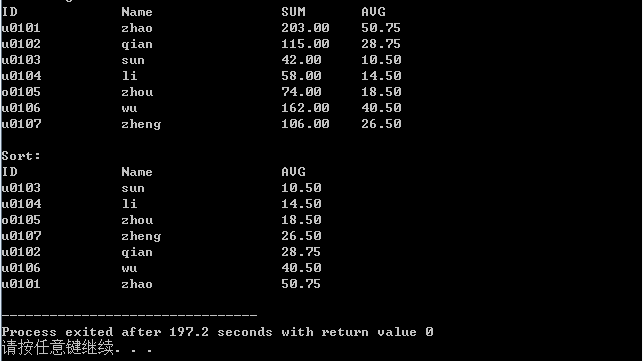
}

}

3）测试

（b） 对应测试数据的运行结果截图





## 7.2选做题

程序设计：3题

1.对编程设计题第（2）题的程序，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。（与程序设计第二题合并）

2.对选做题第（1）题，进一步写出用交换结点指针域的方法升序排序的函数。

（与程序设计第二题合做）

3.采用双向链表重做编程设计题第（2）题。

NULL

## 7.3自设题

## 7.4小结

个人感觉本次实验整体上来讲要比指针简单。一是因为结构联合部分整体比指针好上手一些，二是前面学的指针到了现在已经有了相当程度的消化，而且这一部分对指针的运用也没有那么深入。

前面的求值验证部分是结构的基本内容，但其中也有渗透复杂表达式的意义，需要进一步加深对优先级结合性表的理解记忆，程序修改替换则只要理解了指向自身的指针也能举一反三出来，这两个部分可以说没什么难度。

字段结构一题第一个难点就是其理解。重点在于声明一个字段结构以后地址里存的值是一定的，区别只在于通过何种方式去访问地址中的内容，且还应记住字段结构中先声明的变量对应低字节，后声明变量的对应高字节。按字节访问的多样性也是字段结构本身最大的精妙之处。此外本题还涉及了函数指针数组，对于未来得及多加练习的我来说是个不大不小的障碍，好在最后顺利解决。

有了前面较为丰富的写大程序的经验，再加上一波考试已经过去，面对这次的成绩处理综合体我终于能够沉下心来。让人头疼的字符串与数字混合输入问题在上次的实验中已经得到解决，剥开结构类型的外衣之后里面的其实就是基本的输入输出、参数传递、数组排序。要注意的是在交换数据域排序时其实只需要交换将要输出的部分即可，但坏处是打乱了人与成绩的一一对应关系，使得后续操作变得较为困难。因此更保险的方法是交换全部数据域，或者用更为复杂的交换指针域。

链表是我们所接触的第二种数据结构（第一种为数组），但课内对其最广泛的应用集中在了单向链表上，所以若是想掌握得较为深入，还是要自己去尝试双向链表、十字交叉列表的运用。最后要注意的是结构是一种数据类型，不可同C++中的类相混淆。

# 8文件实验

## 8.1必做题

⑴文件类型的程序验证：1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);

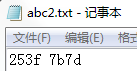
fclose(fp2);

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。

fp1:253f7b7d; fp2:253f 7b7d; 屏幕:?%}{ (换行) 253f 7b7d

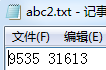
fp2:屏幕:

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

屏幕:

a,b写入fp1时只写入了两个short型的低字节部分，即3f和7d（十六进制），因此屏幕上输出fp1时只输出了?}

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

fp2:屏幕:

a,b两数被转成十进制后再写入fp2并印在屏幕上

⑵程序修改替换：2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

1. 源程序中存在什么样的逻辑错误（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。

#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;

}

**解答：**

替换后的程序如下所示：

#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 \*/

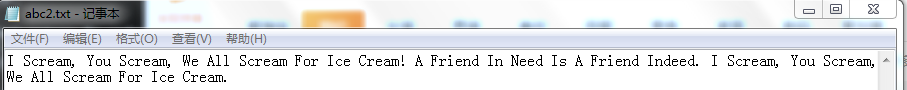
putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

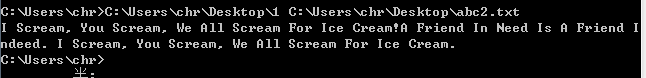
return 0;

}

abc2.txt



cmd



（2）用输入输出重定向freopen改写上述源程序中的main函数。

**解答：**

替换后的程序如下所示：

#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=freopen(argv[1],"r",stdin))==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 \*/

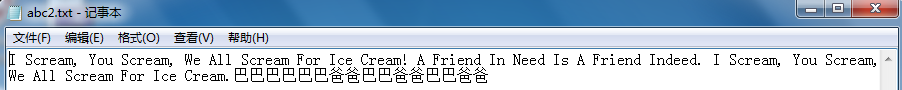
putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

return 0;

}

abc2.txt



cmd



⑶程序设计：1题

从键盘输入一行英文句子，将每个单词的首字母换成大写字母，然后输出到一个磁盘文件“test”中保存。

**解答：**

1）解题思路

1.定义文件指针\*fout,通过其以更新（"a"）方式打开输出用文本文件;

2.从标准输入设备读入一行语句，语句带有空格的特点决定了用fgets读入字符串较为适宜，但需注意串尾置'\0'；

3.从第二个字符开始用循环体判断，且只需判断单词首字母，即若s[i] == ' ' && s[i+1] >= 'a' && s[i+1] <= 'z'，则将s[i+1] 改为大写；

4.单独判断s[0]，若s[0]为小写字母，则将其转成大写 ；

5.在文件夹内输出修改后的字符串；

6.在标准输出设备上输出修改后的字符串；

7.结束

2）源程序清单

#include <stdio.h>

#include <string.h>

int main ()

{

char s[100], i ;

FILE \*fout ;

fout = fopen("C:\\Users\\chr\\Desktop\\abc2.txt", "a") ;

fgets(s, 100, stdin) ;

s[strlen(s) - 1] = '\0' ;

if(s[0] >= 'a' && s[0] <= 'z') {

s[0] = s[0] - 'a' + 'A' ;

}

for(i = 0; s[i+1] != '\0'; i ++ ) {

if(s[i] == ' ' && s[i+1] >= 'a' && s[i+1] <= 'z') {

s[i+1] = s[i+1] - 'a' + 'A' ;

}

}

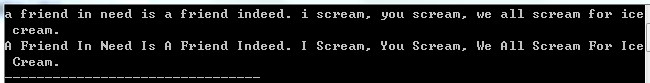
fputs(s, fout) ;

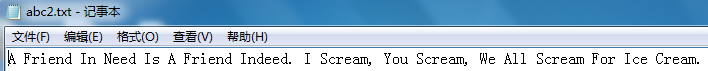
printf("%s", s) ;

}

3）测试

对应测试数据的运行结果截图





## 8.2自设题

## 8.3小结

由于本章实验偏重文件I/O端口的使用，因此程序设计部分的内容上手相对较为简单，但难点在于大量需要理解及掌握的文件相关函数。不巧的是题目中其实只涉及到了教材前半部分的内容，对于后半部分还是要自己下去找题多加练习。由于自己上一次实验的命令行参数部分其实并未完全掌握，因此本次使用时稍微遇到了一些麻烦，但最后总算顺利解决，对于命令行参数的理解使用也更为熟悉，其中最为重要的一点是目标可执行文件的地址固定占用了第一个字符串，我之间就是没有搞清这点才在命令行参数上耽误了不少时间。

在之前的编程中我单纯的认为要是让程序易于理解只需要在字词之间一律加上空格就可以，但上一周我才看到了C语言编程有着自己的格式规范，其中具体到注释风格、代码空格要求、长行拆分与空行要求。统一的编程风格规范可以提高代码的可读性，有利于程序员本人或其他程序员在最短时间内彻底理解代码。因此从本次作业开始，我的源代码开始想变成规范中逐步靠近，不过由于这部分工作是在程序运行正常后统一进行的，因此对完成任务的效率并未造成很大的影响。

## 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