

华中科技大学

# 课程实验报告

课程名称： C 语言程序设计实验

专业班级： CS1708

学 号： U201714739

姓 名： 金 修 旭

指导教师： 甘 早 斌

报告日期： 2018-01-08

计算机科学与技术学院



# 目 录

<b>1</b>	<b>表达式和标准输入与输出实验 .....</b>	<b>1</b>
1.1	实验目的 .....	1
1.2	实验内容 .....	1
1.2.1	源程序改错 .....	1
1.2.2	源程序修改替换 .....	2
1.2.3	程序设计 .....	3
1.3	实验小结 .....	9
<b>2</b>	<b>流程控制实验 .....</b>	<b>11</b>
2.1	实验目的 .....	11
2.2	实验内容 .....	11
2.2.1.	源程序改错题 .....	11
2.2.2.	源程序修改替换题 .....	12
2.2.3.	编程设计题 .....	15
2.2.4.	选做题 .....	25
2.3	实验小结 .....	27
<b>3</b>	<b>函数与程序结构实验 .....</b>	<b>29</b>
3.1	实验目的 .....	29
3.2	实验内容 .....	29
3.2.1	改错题 .....	29
3.2.2	修改替换题 .....	30
3.2.3	跟踪调试题 .....	32
3.2.4	程序设计 .....	34
3.3	实验小结 .....	39
<b>4</b>	<b>编译预处理实验 .....</b>	<b>41</b>
4.1	实验目的 .....	41
4.2	实验内容 .....	41
4.2.1.	源程序改错题 .....	41

4.2.2. 源程序修改替换题 .....	42
4.2.3. 跟踪调试题 .....	47
4.2.4. 编程设计题 .....	49
4.3 实验小结 .....	52
<b>5 数组实验 .....</b>	<b>55</b>
5.1 实验目的 .....	55
5.2 实验内容 .....	55
5.2.1 源程序改错 .....	55
5.2.2 源程序修改替换 .....	57
5.2.3 跟踪调试题 .....	61
5.2.4 程序设计题 .....	64
5.2.5 选做题程序设计 .....	74
5.3 实验小结 .....	79
<b>6 指针实验 .....</b>	<b>81</b>
6.1 实验目的 .....	81
6.2 实验内容 .....	81
6.2.1. 源程序改错题 .....	81
6.2.2. 源程序修改替换题 .....	82
6.2.3. 跟踪调试题 .....	84
6.2.4. 编程设计题 .....	85
6.2.5 选做题 .....	95
(三) 指定 main 函数的参数 .....	99
6.3 实验小结 .....	100
<b>7 结构与联合实验 .....</b>	<b>103</b>
7.1 实验目的 .....	103
7.2 实验内容 .....	103
7.2.1. 表达式求值的程序验证题 .....	103
2. 源程序修改替换题 .....	105

4. 选做题 .....	127
7.3 实验小结 .....	127
<b>8 文件实验 .....</b>	<b>129</b>
8.1、实验目的 .....	129
1. 熟悉文本文件和二进制文件在磁盘中的存储方式； .....	129
2. 熟练掌握流式文件的读写方法。 .....	129
8.2 实验内容 .....	129
8.2.1. 文件类型的程序验证题 .....	129
8.2.2. 源程序修改替换题 .....	131
8.2.3. 编程设计题 .....	134
8.3 实验小结 .....	136



# 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。

```
01 #include <stdio.h>
02 #define PI 3.14159;
03 voidmain(void)
04 {
05     int f;
06     short p, k;
07     double c, r, s;
08     /* for task 1 */
```

```

09  printf("Input Fahrenheit
      1.  :");
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);
23  }

```

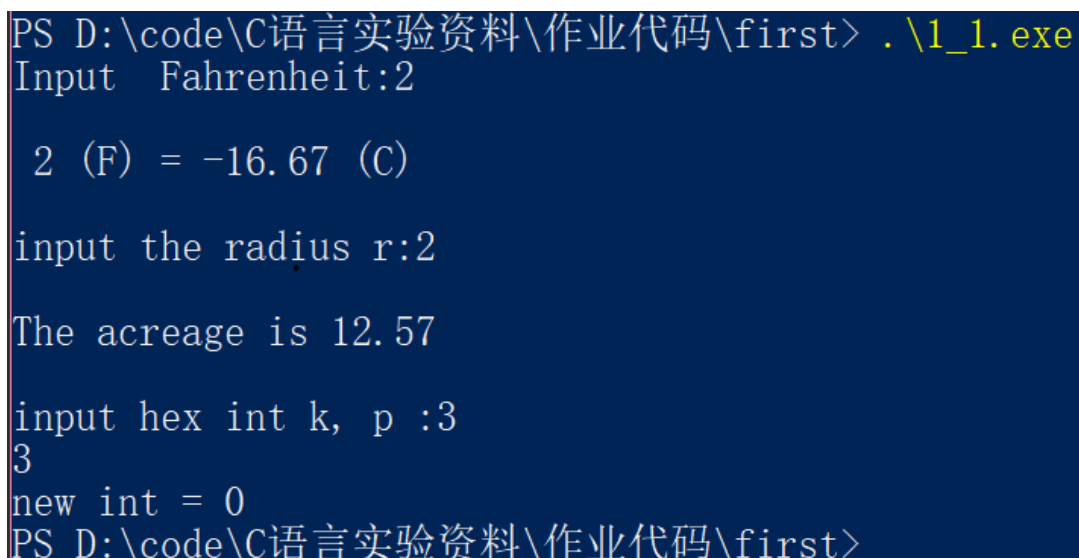


解答：

(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 3.1415926
- 2) 第 3 行的 voidmain(void)，正确形式为：int main(void)
- 3) 第 10 行缺少取地址符 scanf("%d", f)；应该为 scanf("%d", &f)；
- 4) 第 15 行%f 应该为%lf
- 5) 第 21 行中左移应改为右移，定义变量应使用 int newint...
- 6) 第 11 行中，表达式应改为 5/9.0\* (f-32)

(2) 错误修改后运行结果：



```

PS D:\code\C语言实验资料\作业代码\first> .\1_1.exe
Input Fahrenheit:2

2 (F) = -16.67 (C)

input the radius r:2

The acreage is 12.57

input hex int k, p :3
3
new int = 0
PS D:\code\C语言实验资料\作业代码\first>
    
```

图 1-1 源程序改错的运行结果

(3) 改正后源代码：

```

01 #include <stdio.h>
02 #define PI 3.14159
03 int main(void)
04 {
05     int f;
06     short p, k;
07     double c, r, s;
    
```

```

08  /* for task 1 */
09  printf("Input  Fahrenheit:");
10  scanf("%d", &f);
11  c = 5.0 / 9.0 * (f - 32.0);
12  printf("\n %d (F) = %.2f (C)\n\n", f, c);

13  /* for task 2 */
14  printf("input the radius r:");
15  scanf("%lf", &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("%hd %hd", &k, &p);
21  short newint = (p & 0xff00) + ((k & 0xff00) >> 8);
22  printf("new int = %hd \n", newint);

23  return 0;
24  }

```

### 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;

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

```

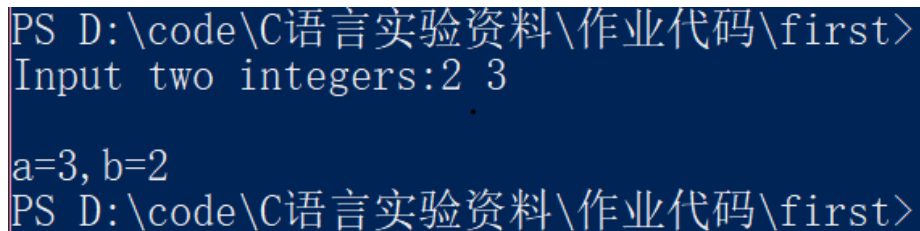
**解答：**

利用按位异或的计算性质，替换后的程序如下所示：

```

#include<stdio.h>
int main(void){
    int a, b;
    printf("Input two integers:");
    scanf("%d %d",&a,&b);
    a = a^b;
    b = a^b;
    a = a^b;
    printf("\na=%d,b=%d", a, b);
}

```



```

PS D:\code\C语言实验资料\作业代码\first>
Input two integers:2 3
a=3, b=2
PS D:\code\C语言实验资料\作业代码\first>

```

图 1-2 源程序修改替换题运行结果

### 1.2.3 程序设计

(1) 编写一个程序，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，最后输出  $c$ 。

**解答：**

1) 算法流程如图 1.1 所示。

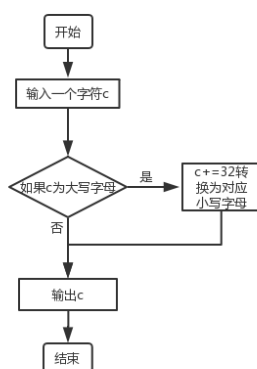


图 1-3 编程题 1 的程序流程图

## 2) 源程序清单

```

#include <stdio.h>

int main(void)
{
    char c;
    printf("Input a letter please:");
    scanf("%c", &c);
    if (c <= 90 && c >= 65)
        printf("%c", (c + 32));
    else
        printf("%c", c);
    return 0;
}
  
```

## 3) 测试

(a) 测试数据:

a A b B

(b) 对应测试数据的运行结果截图

```
Input a letter please:a
a
PS D:\code\C语言实验资料\作业代码\first> .\1_31.exe
Input a letter please:A
a
PS D:\code\C语言实验资料\作业代码\first> .\1_31.exe
Input a letter please:b
b
PS D:\code\C语言实验资料\作业代码\first> .\1_31.exe
Input a letter please:B
b
```

图 1-4 编程题 1 的运行结果截图

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

解答：

1) 解题思路：

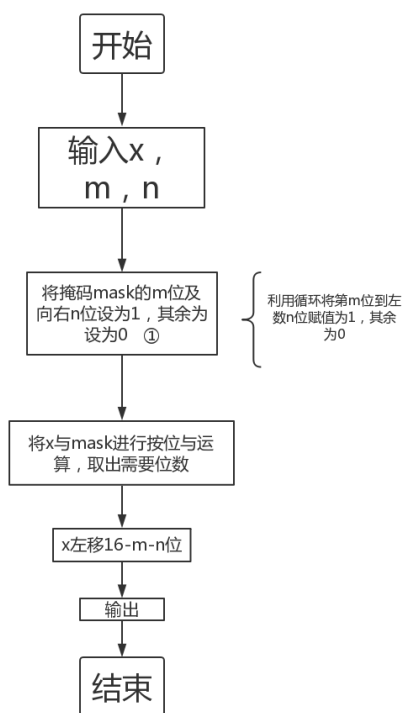


图 1-5 编程题 2 的算法流程图

2) 程序清单

```
#include <stdio.h>
#include <math.h>
```

```

int main(void){

    unsigned short x;
    int m, n;
    unsigned short mask = 0;
    int maskl[16];

    printf("please input x, m, n:");
    scanf("%hx %d %d", &x, &m, &n);

    if(m>15 || m<0 || n>16-m || n<1)
        printf("error\n");
    else{
        for(int i = 0; i <= 15; i++){           //利用循环将第 m 位到左数 n
        位
                                                //赋值为 1，其余为 0

            if(i>=m && i<(m+n))
                maskl[i] = 1;
            else
                maskl[i] = 0;
            mask = maskl[i] * pow(2.0, i) + mask;
        }

        x = x & mask;
        x <<= (15 - (m + n-1));
        printf("\n%x\n", x);
    }

    return 0;
}

```

### 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	0100 0110 1000 0000 (4680)	7	4	计算结果 1101 0000 0000 0000 即 D000
用例 2	1101 0101 1000 0011 (D583)	16	1	输入错误 (m 值超范围)
用例 3	1101 0101 1000 0011 (D583)	13	5	输入错误 (n 值超范围)

(b) 对应测试用例 1 的运行结果如图 1-6 所示。

```
PS D:\Code\C语言实验资料\作业代码\first>
please input x, m, n:4680 7 4

d000
PS D:\Code\C语言实验资料\作业代码\first>
```

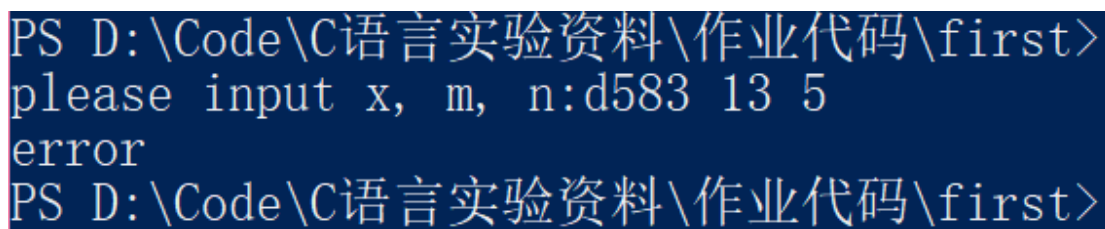
图 1-6 编程题 3 的测试用例一的运行结果

对应测试用例 2 的运行结果如图 1-7 所示。

```
PS D:\Code\C语言实验资料\作业代码\first>
please input x, m, n:d583 16 1
error
PS D:\Code\C语言实验资料\作业代码\first>
```

图 1-7 编程题 3 的测试用例二的运行结果

对应测试用例 3 的运行结果如图 1-8 所示。



```
PS D:\Code\C语言实验资料\作业代码\first>
please input x, m, n:d583 13 5
error
PS D:\Code\C语言实验资料\作业代码\first>
```

图 1-8 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），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。例如：

整形 676879571 的二进制表示是 00101000 01011000 01011100 11010011。按照 8 位一组可表示为 40 88 92 211。由于 CPU 处理数据的差异，它的顺序是颠倒的，所以最终格式为 211.92.88.40。

**解答：**

- 1) 解题思路：输入无符号整形数，并利用四组掩码提取出各需要序列，倒序后输出
- 2) 程序清单：

```
#include <stdio.h>

#include <math.h>

int main(void) {

    unsigned int mask[4] = {0xff000000, 0x00ff0000, 0x0000ff00,
0xff};
    unsigned int ipbit[4];

    unsigned int ip;
```



```

printf("please input a int ip address\n");
scanf("%d", &ip);

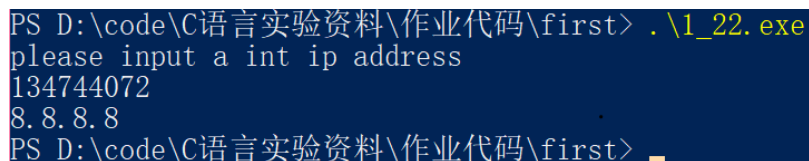
ipbit[0] = (ip & mask[0]) >> 24;
ipbit[1] = (ip & mask[1]) >> 16;
ipbit[2] = (ip & mask[2]) >> 8;
ipbit[3] = (ip & mask[3]);
printf("%u.%u.%u.%u", ipbit[3], ipbit[2], ipbit[1],
ipbit[0]);

return 0;
}

```

### 3) 测试

对应截图：



```

PS D:\code\C语言实验资料\作业代码\first> .\1_22.exe
please input a int ip address
134744072
8.8.8.8
PS D:\code\C语言实验资料\作业代码\first>

```

图 1-9 编程题 4 的测试用例的运行结果

## 1.3 实验小结

在本次实验中，第一题主要考察对 C 语言一些基本语法的理解以及标准输出函数 `scanf` 的简单用法以及对位运算的使用，题目设置为改错形式，让我深刻认识到了在编写程序时尽量保证认真，在开始编程前做好计划的重要性。任何一个细节的错误都会对整个程序造成影响。

第二题考察对已知问题的另一种思路，可使用按位异或运算的性质或简单的加减法运算达到目的，打开了思路，提醒我们在面对一个问题时，应该多思考以找到更加高效，或更加简洁，亦或是单纯的更加有趣的方式，以锻炼创新性思维。

第三题考察对于 ASCII 码表的理解及简单的选择语句使用,利用在 ASCII 码表中大小写字母编号相差 32 的特性,用简单的方法完成大小写转换。。

第四题则考察对于位运算的理解与使用,在解决第四题时,使用了不同的方式,先利用数组构造出一个掩码,由按位乘以位权的方式得到整数型掩码再进行按位与及左移运算,避免考虑右移操作时不同机器补位方式不同的问题。

第五题考察了 IP 地址的构成的常识,以及对掩码的使用。

总之,由第一次 C 语言上机实验,感受到了在简单程序设计中应该考虑到的,平时未考虑到的诸如对错误输入的处理,可移植性等问题。感受到了编程之美,体会到了当程序正确运行之后的成就感,使我开始习惯编写程序的思维。

## 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 Turbo C 2.0 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 }
11 return 0;
```

解答：

(1) 错误修改:

1) 第 2 行的 `void main(void)` 有误, 正确形式为:

```
int main(void)
```

2) 第 4 行的 `int s`; 存在漏洞, 当 `n` 比较大时, `int` 型的位数不够, `longlong` 型可以更好容纳, 正确形式为:

```
int i,n; long long s=1;
```

3) 第 6 行的 `n` 前缺少取地址符, 正确形式为:

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

4) 第 7 行的标点符号错误, 应把逗号改为分号, 正确形式为:

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

5) 第 9 行的 `printf` 应对应为:

```
printf("%d! = %.0lf",n,s);
```

6) 第 9, 10 行右花括号与 `return` 语句应交换;

(2) 错误修改后运行结果: 如图 2-1 所示



```
Please enter n:8
8! = 40320
PS D:\Code\C语言实验资料\作业代码\second>
```

图 2-1 改错题的测试的运行结果

### 2.2.2. 源程序修改替换题

(1) 修改第 1 题, 分别用 `while` 和 `do-while` 语句替换 `for` 语句。

```
#include <stdio.h>
```

```
void main(void)
```

```
{
```

```
int i,n,s=1;
```

```

    printf("Please enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        s=s*i;
    printf("%d! = %d",n,s);
    return 0;
}

```

**解答：**

```
#include <stdio.h>
```

```

int main(void){
    int i, n, s=1;

    printf("Please enter n:");
    scanf("%d", &n);

```

```

    #if 0
        i = 1;
        while(i<=n){
            s *= i;
            i++;
        }
    #endif

```

```

    #if 1
        i = 1;
        do{
            s *= i;

```

```

        i++;
    }while(i<=n);
#endif

    printf("%d! = %d", n, s);
    return 0;
}

```

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

**解答：**

替换后的程序如下所示：

```

#include <stdio.h>

int main(void){
    int i, n = 1, s=1, su = 1;

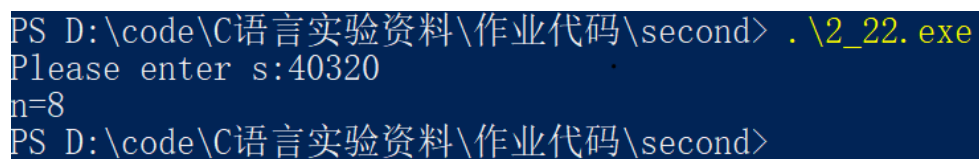
    printf("Please enter s:");
    scanf("%d", &s);

    for(n = 1; su < s; n++){
        su *= n;
    }

    printf("n=%d", n-1);
    return 0;
}

```

替换后运行结果如图 2-2 所示：



```

PS D:\code\C语言实验资料\作业代码\second> .\2_22.exe
Please enter s:40320
n=8
PS D:\code\C语言实验资料\作业代码\second>

```

图 2-2 替换题的测试的运行结果

### 2.2.3. 编程设计题

编写并上机调试运行能实现以下功能的程序。

- (1) 假设工资税金按以下方法计算： $x < 1000$  元，不收取税金； $1000 \leq x < 2000$ ，收取 5% 的税金； $2000 \leq x < 3000$ ，收取 10% 的税金； $3000 \leq x < 4000$ ，收取 15% 的税金； $4000 \leq x < 5000$ ，收取 20% 的税金； $x \geq 5000$ ，收取 25% 的税金。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：**

A) if 语句

1) 解题思路：

运用 if 构造条件选择， $x < 1000$ ，不收税金； $1000 \leq x < 2000$ ，收 5% 税金； $2000 \leq x < 3000$ ，收 10% 税金； $3000 \leq x < 4000$ ，收 15% 税金； $4000 \leq x < 5000$ ，收 20% 税金； $x \geq 5000$ ，收 25% 税金。

2) 源程序清单

```
#include<stdio.h>
```

```
int main(void){
```

```
    double x;
```

```
    int i;
```

```
    scanf("%lf", &x);
```

```
    #if 1
```

```
        if(x < 1000)
```

```

        printf("No Tax\n");

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

        printf("Tax is %lf\n", 0.05*(x-1000));

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

        printf("Tax is %lf\n", 0.1*(x-2000)+50);

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

        printf("Tax is %lf\n", 0.15*(x-3000)+150);

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

        printf("Tax is %lf\n", 0.2*(x-4000)+300);

    else if(x > 5000)

        printf("Tax is %lf\n", 0.25*(x-5000)+500);

#endif

    return 0;

}


```

### 3) 测试

(a) 测试数据:  $x=4500$

(b) 对应测试数据的运行结果截图

对应测试的运行结果如图 2-3 所示。



```

4500
Tax is 400.000000
PS D:\Code\C语言实验资料\作业代码\second>

```

图 2-3 编程题 1A 的测试运行结果

### B) switch 语句

#### 1) 解题思路:



运用 switch 构造条件选择,  $x < 1000$ , 不收税金;  $1000 \leq x < 2000$ , 收 5%税金;  $2000 \leq x < 3000$ , 收 10%税金;  $3000 \leq x < 4000$ , 收 15%税金;  $4000 \leq x < 5000$ , 收 20%税金;  $x \geq 5000$ , 收 25%税金。

## 2) 源程序清单

```
#include<stdio.h>
```

```
int main(void){
```

```
    double x;
```

```
    int i;
```

```
    scanf("%lf", &x);
```

```
    if(x < 1000)
```

```
        i = 1;
```

```
    else if(x >= 1000 && x < 2000)
```

```
        i = 2;
```

```
    else if(x >= 2000 && x < 3000)
```

```
        i = 3;
```

```
    else if(x >= 3000 && x < 4000)
```

```
        i = 4;
```

```
    else if(x >= 4000 && x < 5000)
```

```
        i = 5;
```

```
    else if(x > 5000)
```

```
        i = 6;
```

```

switch(i){

    case 1:

        printf("No Tax\n");

        break;

    case 2:

        printf("Tax is %lf\n", 0.05*(x-1000));

        break;

    case 3:

        printf("Tax is %lf\n", 0.1*(x-2000)+50);

        break;

    case 4:

        printf("Tax is %lf\n", 0.15*(x-3000)+150);

        break;

    case 5:

        printf("Tax is %lf\n", 0.2*(x-4000)+300);

        break;

    case 6:

        printf("Tax is %lf\n", 0.25*(x-5000)+500);

        break;

}

return 0;

}

```

### 3) 测试

(a) 测试数据：如表 2-1 所示

**表 2-1-2 编程题 1B 的测试数据**

测试用例	程序输入	理论结果	运行结果
用例 1	1000	0.000000	0.000000
用例 2	2000	50.000000	50.000000
用例 3	3000	150.000000	150.000000
用例 4	10000	1750.000000	1750.000000

(b) 对应测试数据的运行结果截图

对应测试的运行结果如图 2-4 所示。

```

1000
Tax is 0.000000
PS D:\code\C语言实验资料\作业代码\second> .\2_31.exe
2000
Tax is 50.000000
PS D:\code\C语言实验资料\作业代码\second> .\2_31.exe
3000
Tax is 150.000000
PS D:\code\C语言实验资料\作业代码\second> .\2_31.exe
10000
Tax is 1750.000000
PS D:\code\C语言实验资料\作业代码\second>
    
```

**图 2-4 编程题 1B 的测试运行结果**

(2) 编写一个程序,将输入的一行字符复制到输出,复制过程中将一个以上的空格字符用一个空格代替。

**解答：**

1) 算法流程如图 2-5 所示

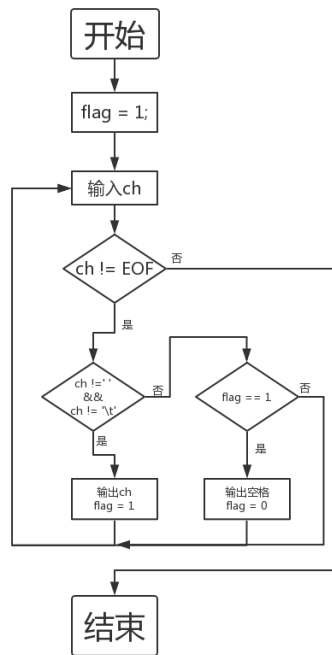


图 2-5 编程题 2 的算法流程图

## 2) 源程序清单

```
#include<stdio.h>
```

```
#include<string.h>
```

```
int main(void){
```

```
    char ch;
```

```
    int flag = 1;
```

```
    while((ch = getchar()) != EOF){
```

```
        if(ch != ' ' && ch != '\t'){
```

```
            putchar(ch);
```

```
            flag = 1;
```

### 3) 测试

```
dajsfja asldkfaldkfasjalkds dsfafs
dajsfja asldkfaldkfasjalkds dsfafs
dsfa sfd a
dsfa sfd a
a
a
    adsfafaf
adsfafaf
^Z
PS D:\Code\C语言实验资料\作业代码\second>
```

图 2-6 编程题 2 的测试用例 1 的运行结果

(3) 打印如下杨辉三角形。

---

21

```

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  $C_i^j$  计算（表示第  $i$  行第  $j$  列位置的值），而  $C_i^j$  的计算如下：

$$C_i^0 = 1 \quad (i=0,1,2,\dots)$$

$$C_i^j = C_i^{j-1} * (i - j + 1) / j \quad (j=0,1,2,3,\dots,i)$$

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) 算法思路：

利用函数递归调用求出杨辉三角每个数据的值，循环输出，在 `printf` 函数中使用 “%3d ” 控制每一项数据间间隔空格数要求与金字塔效果

#### 2) 源程序清单

```

#include<stdio.h>

int pascalT(int i, int j);

//该函数输出杨辉三角中的第 i 行第 j 个，从 0 开始计数

int main(void){

    for(int i = 0, k = 18; i < 10; i++, k-=2){

        for (int l = k; l >= 0; l--)

            putchar(' ');

        for (int j = 0; j < 10; j++){

```

```

        if (pascalT(i, j))

            printf("%3d ", pascalT(i, j));

        else

            printf(" ");

    }

    putchar('\n');

}

return 0;

}

int pascalT(int i, int j){

    if(j == 0)

        return 1;

    else

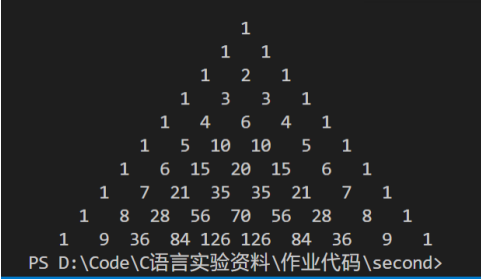
        return pascalT(i, j-1)*(i-j+1)/j;//使用递归

}

```

### 3)测试

对应测试数据的运行结果截图如图 2-7 所示



```

      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 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
PS D:\Code\C语言实验资料\作业代码\second>

```

图 2-7 编程题 3 的测试的运行结果

(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) 算法流程如图 2-8 所示

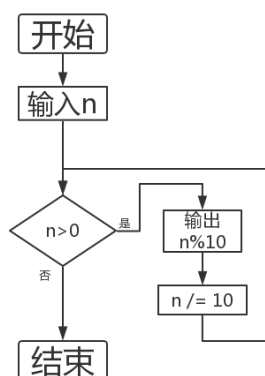


图 2-8 编程题 4 的程序流程图

2) 源程序清单

```

#include<stdio.h>

int main(void){

    int n;
    scanf("%d", &n);
    while(n > 0){
        printf("%d", n % 10);
        n /= 10;
    }
    return 0;
}
    
```

3)测试



对应测试运行结果如图 2-9 所示。

```
mpccode runner v1.12.0 j
132211
112231
PS D:\Code\C语言实验资料\作业代码\second>
```

图 2-9 编程题 4 的测试运行结果

#### 2.2.4. 选做题

编写并上机调试运行能实现以下功能的程序。

编写一个程序，用牛顿迭代法求方程  $f(x) = 3x^3 - 4x^2 - 5x + 13 = 0$  满足精度  $e=10^{-6}$  的一个近似根，并在屏幕上输出所求近似根。

**解答：**

1) 算法流程如图 2-10 所示

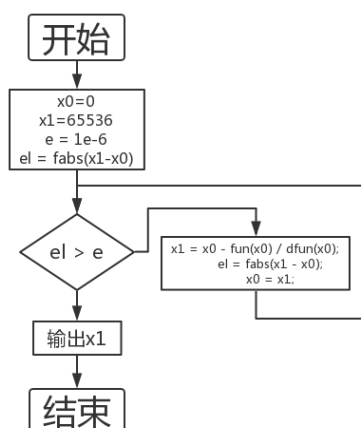


图 2-10 选做题的程序流程图

2) 源程序清单

```
#include<stdio.h>

#include<stdlib.h>

#include<math.h>
```

```
double fun(double x);

double dfun(double x);


int main(void){

    double x0=0.0, x1= 65536.0;

    double e = 1e-6;

    double el;


    el = fabs(x1 - x0);
    while ( el > e){

        x1 = x0 - fun(x0) / dfun(x0);

        el = fabs(x1 - x0);

        x0 = x1;

    }


    printf("%lf", x1);

    return 0;

}


double fun(double x){

    return 3 * x * x * x - 4 * x * x - 5 * x + 13;

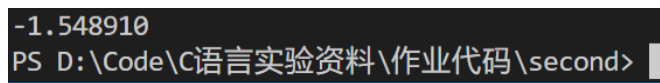
}


double dfun(double x){
```

```
return 4 * x * x - 8 * x - 5;

}
```

3) 对应测试数据的运行结果截图如图 2-11 所示



```
-1.548910
PS D:\Code\C语言实验资料\作业代码\second>
```

图 2-11 选做题的测试的运行结果

## 2.3 实验小结

第二次 C 语言上机实验，主要熟悉了复合语句、if 语句、switch 语句的使用，以及 for、while、do-while 三种基本的循环控制语句的使用，并了解了转移语句与标号语句。

### (1) 改错实验

进一步提醒我，对于一些语法的细节需要格外注意，有时一个打字错误带来的 bug 需要耗费大量精力去找出。同时，需要对数据类型的范围及精度有认识和概念。同时，学习使用了 for 语句

### (2) 程序修改替换实验

此题让我感受到用不同语句完成相同任务，以及已知一算法，对其求反的思想，同时与第一题改错结合，练习使用了三种循环语句，感受到了三种语句的利弊。

### (3) 程序设计实验

工资税一题，了解到了税款的算法与 switch 语句的使用。处理空格的题目中，利用 flag 变量记录当前字符状态以决定是否需要输出单个空格，十分有效，可以迁移运用到其他程序设计中。杨辉三角题首先让我感受到了我国古代数学家的智慧，其次，再次感受到了递归的威力，将普通的迭代算法换成了简洁的递归写法。正整数逆转中，继续熟悉循环的利用。

### (4) 选做题

此题首先了解了牛顿迭代法求解方程近似根的思想，正如其名，

利用循环迭代求解是十分自然的思路，在实际编程中，使用到了库函数 `fabs()`。

## 3 函数与程序结构实验

### 3.1 实验目的

- (1) 熟悉和掌握函数的定义、声明；函数调用与参数传递方法；以及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) 熟悉多文件编译技术。

### 3.2 实验内容

#### 3.2.1 改错题

下面是计算  $s=1!+2!+3!+\dots+n!$  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。

```
#include "stdio.h"

void main(void)
{
    int k;
    for(k=1;k<6;k++)
        printf("k=%d\tthe sum is %ld\n",k,sum_fac(k));
}

long sum_fac(int n)
{
    long s=0;
    int i;
    long fac;
    for(i=1;i<=n;i++)
        fac*=i;
```

```
s+=fac;

return s;
}
```

**解答：**

(1) 错误修改：

1) 未在主函数前声明 `sum_fac(int n)` 函数正确形式为：

```
long sum_fac(int n);
```

2) 第 2 行的 `voidmain( void )` 有误，正确形式为：

```
int main( void )
```

3) 第 10 行的 `fac` 未定义，正确形式为：

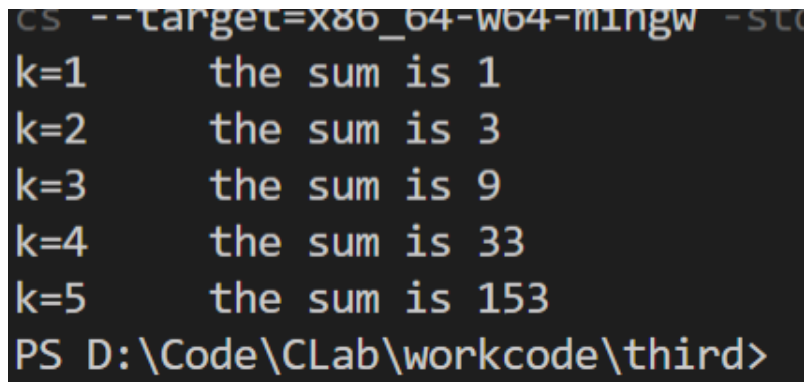
```
long fac=1;
```

4) 第 11 行的 `for` 循环需要修改，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如图 3-1 所示：



```
cs --target=x86_64-w64-mingw -std
k=1 the sum is 1
k=2 the sum is 3
k=3 the sum is 9
k=4 the sum is 33
k=5 the sum is 153
PS D:\Code\CLab\workcode\third>
```

图 3-1 改错题的测试的运行结果

### 3.2.2 修改替换题

(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);

long s=0;

long fac = 1;


int main(void){

    int k;

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

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

}

```

```

long sum_fac(int n){

    fac*=n;

    s+=fac;

    return s;

}

```

(2) 修改第 1 题中 sum\_fac 函数，计算  $s = 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$ 。

```
#include <stdio.h>
```

```

double sum_fac(int n);

double s=0;

double fac = 1;


int main(void){

```

```

int k;
for(k=1;k<6;k++)
    printf("k=%d\tthe sum is %lf\n",k,sum_fac(k));
}

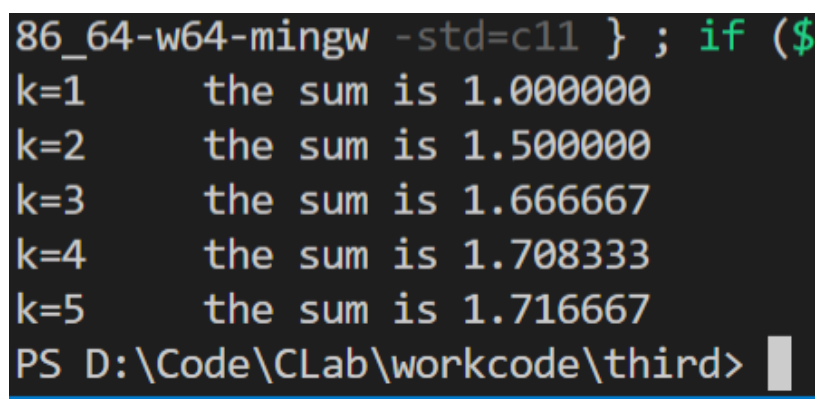
double sum_fac(int n){

    fac *= n;
    s += 1/fac;

    return s;
}

```

替换后运行结果如图 3-2 所示：



```

86_64-w64-mingw -std=c11 } ; if ($)
k=1      the sum is 1.000000
k=2      the sum is 1.500000
k=3      the sum is 1.666667
k=4      the sum is 1.708333
k=5      the sum is 1.716667
PS D:\Code\CLab\workcode\third>

```

图 3-2 替换题的测试的运行结果

### 3.2.3 跟踪调试题

下面是计算 fibonacci 数列前 n 项和的程序，要求单步执行程序，观察 p,i,sum,n 值。

源程序：

```

1  #include <stdio.h>

2  long fibonacci(int);

```



```

3  int main(void)

4  {

5  int i,k;

6  long sum=0,*p=&sum; /* 声明 p 为长整型指针, 指向 sum */

7  scanf("%d",&k);

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

9  sum+=fabonacci(i);

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

11 }

12 return 0;

13 }

14 long fabonacci(int n)

15 {

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

17 return 1;

18 else

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

20 }

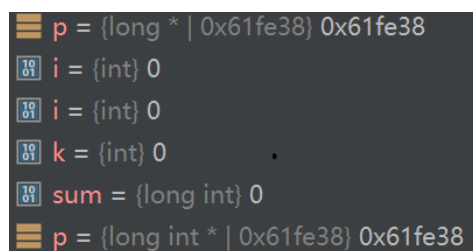
```

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)。

(1) 刚执行完 scanf("%d",&k);语句, p,i 值是多少?

答: 执行完该语句, 进入 for 循环前, p 值为 0x61fe38, i 为 0.//此处

有疑问，i, k 均为 0，可能由于 gcc 编译器的优化选项引起。



```

p = {long * | 0x61fe38} 0x61fe38
i = {int} 0
i = {int} 0
k = {int} 0
sum = {long int} 0
p = {long int * | 0x61fe38} 0x61fe38
    
```

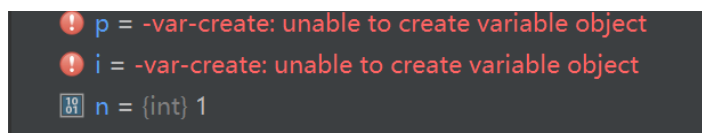
图 3-3 跟踪调试题的测试结果（1）

（2）从 fibonacci 函数返回后光条停留在哪个语句上？

答：从 fibonacci 函数返回后光条停留在 `printf("i=%d\tthe sum is %ld\n",i,*p);`上

（3）进入 fibonacci 函数，watch 窗口显示的是什么？

答：watch 窗口显示了 n。



```

! p = -var-create: unable to create variable object
! i = -var-create: unable to create variable object
n = {int} 1
    
```

图 3-4 跟踪调试题的测试结果（3）

（4）当 i=3，从调用 fibonacci 函数到返回，n 值如何变化？

答：从 3 到 2 到 1，然后返回。

### 3.2.4 程序设计

（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：

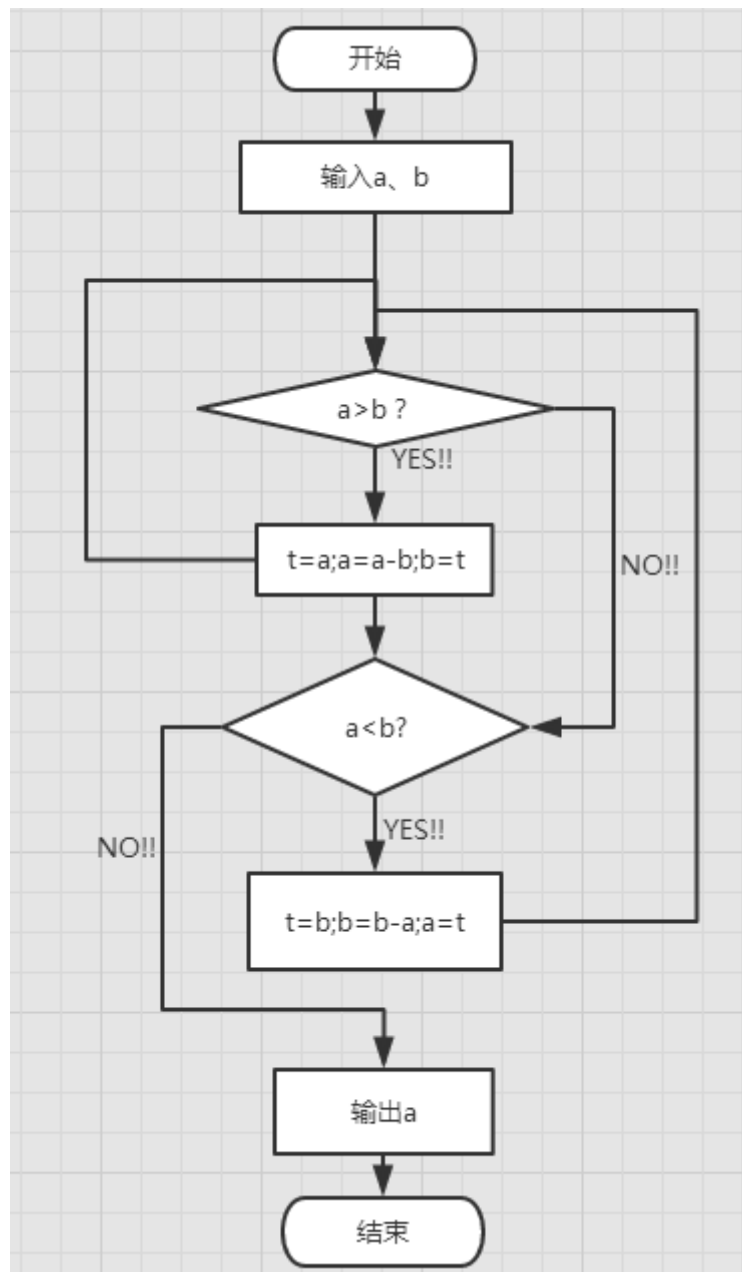


图 3-5 编程题 1 的算法流程图

## 2) 源程序清单

```
#include <stdio.h>
```

```
int gcd(int x, int y);
```

```
int main(void)
```

```
{
```

```

int x, y;

scanf("%d%d", &x, &y);

printf("The gcd of %d and %d is %d\n", x, y, gcd(x, y));

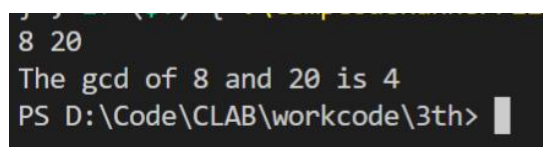
return 0;
}

int gcd(int x, int y)
{
    if (x > y)
        return gcd(x - y, y);
    else if (x < y)
        return gcd(x, y - x);
    else
        return x;
}

```

### 3) 测试

对应测试数据的运行结果截图



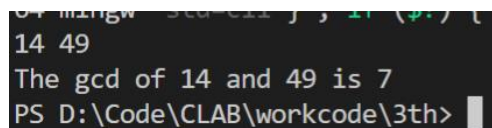
```

8 20
The gcd of 8 and 20 is 4
PS D:\Code\CLAB\workcode\3th>

```

图 3-6 编程题 1 的测试运行结果 (1)

对应测试用例 2 的运行结果如图 1-3-3 所示。



```

14 49
The gcd of 14 and 49 is 7
PS D:\Code\CLAB\workcode\3th>

```

图 3-7 编程题 1 的测试运行结果 (2)

(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  $n \geq 4$  is the sum of two primes.

10=3+7

12=5+7

.....

20=3+17

解答：

1) 解题思路：

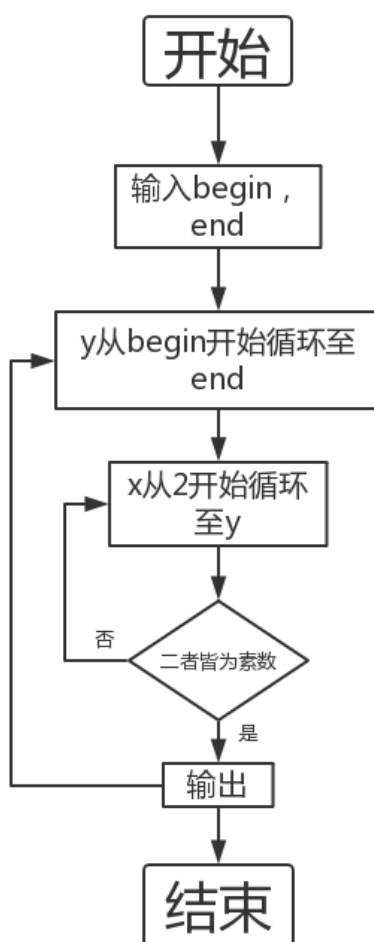


图 3-8 编程题 2 的算法流程图

## 2) 程序清单

```
#include<stdio.h>
#include<math.h>
#define BEGIN 4
#define END 20

int IsPrime(int n);

int main(void){
    int i, j, k = 0;
    printf("GOLDBACH'S CONJECTURE:\nEvery even number n>=4 is the sum of
two primes.\n");
    for (j = BEGIN; j <= END; j += 2){
        for (i = 2; i < j; i++){
            if(IsPrime(i) && IsPrime(j-i))
                printf("%d = %d + %d\n", j, i, j - i);
            k++;
            if(!(k%5))
                break;
        }
    }
    return 0;
}

int IsPrime(int n){
    int i;
    if(n == 1)
        return 0;

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

```

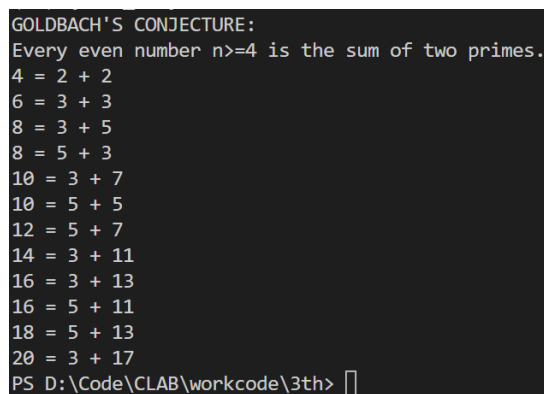
        if(n%i == 0)
            return 0;

    return 1;
}

```

### 3) 测试

对应测试的运行结果如图 3-9 所示。



```

GOLDBACH'S CONJECTURE:
Every even number n>=4 is the sum of two primes.
4 = 2 + 2
6 = 3 + 3
8 = 3 + 5
8 = 5 + 3
10 = 3 + 7
10 = 5 + 5
12 = 5 + 7
14 = 3 + 11
16 = 3 + 13
16 = 5 + 11
18 = 5 + 13
20 = 3 + 17
PS D:\Code\CLAB\workcode\3th>

```

图 3-9 编程题 2 的测试运行结果

## 3.3 实验小结

第三次 C 语言上机实验，主要掌握并熟悉了函数的定义、声明、函数调用与参数传递方法以及函数返回值类型的定义和返回值使用、不同存储类型变量的使用以及熟悉多文件编译技术。

### (1) 改错实验

在改错实验中，主要学习到了对函数的基本使用。对一些基本概念有了一定的了解。认识到了函数在结构化程序设计中的重要意义。函数可以把大的计算过程分解成若干个小的任务。程序设计人员可以以函数为基础，进一步编写程序，而无须重复编写相同代码。同时，一个设计优秀的函数还可以将程序中不需了解或可不关心的计算过程隐藏，做到“黑箱”效果，使程序结构更加清晰。

### (2) 程序修改替换实验

在修改替换题中，第一题开始我使用全局变量，将每次的阶乘以及阶乘和都用到了下一次的计算中。经过助教学长点拨，后来更换成为了局部静态变量，也意识到了随意使用全局变量的缺陷。第二题继承第一题方式，略作修改完成。

### **(3) 跟踪调试题**

此题分别使用 gdb 与 CLion 中的调试器完成，gdb 由于 gcc 优化选项问题无法查看到两变量值。CLion 中，p 显示出了 sum 在变量中的地址，而代码中原来未初始化的变量值全为 0，考虑也许是编译器优化问题。做完此题，感受到了现代编译器对代码的优化技术之深奥与强大。

### **(4) 程序设计实验**

第一题在使用递归技术的同时，领会到了更相减损术的思想，感受到了中国古代数学研究的美丽，同时通过观察递归中函数调用栈的变化，对递归有了进一步了解。第二，三题合并为一题完成。此题起初并未考虑到一个偶数可能有多种拆分方式的问题，在后续网上提交作业系统中加以改正。

### **(5) 选做**

通过此题，掌握到了利用 gcc 与 CLion 进行多文件编译的方式。

通过此次 C 语言上机实验，我感到编程是需要不断练习，才可以得到进步的。



## 4 编译预处理实验

### 4.1 实验目的

- (1) 掌握文件包含、宏定义、条件编译、assert 宏的使用；
- (2) 练习带参数的宏定义、条件编译的使用；
- (3) 练习 assert 宏的使用；
- (4) 使用 Turbo C 2.0 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。

### 4.2 实验内容

#### 4.2.1. 源程序改错题

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);

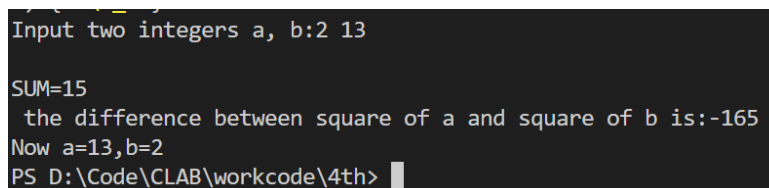
13 }
    
```

### 解答:

#### (1) 错误修改:

- A) 将宏定义中后面的表达式加上括号,以防止在第 10 行中出现优先级混乱的问题导致错误。
- B) 第四行, 修改为 `#define SWAP(a, b, t) t=a, a=b, b=t。`
- C) 第五行, 修改为 `int main(void)。`
- D) 第九行, 去掉两 `%d` 之间的逗号。
- E) 第十一行改为 `SWAP(a, b, t);`

#### (2) 错误修改后的运行和结果:



```

Input two integers a, b: 2 13

SUM=15
the difference between square of a and square of b is:-165
Now a=13,b=2
PS D:\Code\CLAB\workcode\4th>
    
```

图 4-1 改错题错误修改后的运行结果

### 4.2.2. 源程序修改替换题

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。

```

1 void main(void)

2 {

3     int a, b, c;
    
```

```

4    float d, e;

5    printf("Enter three integers:");

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

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

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

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

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

11   return  0;

12 }

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

14 {

15     int t;

16     if (x>y)

17         t=x;

18     else

19         t=y;

20     if (t<z)

21         t=z;

22     return t;

23 }

24 float sum(float x, float y)

```

```

25 {
26     return x+y;
27 }

```

(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。

修改后代码：

```

#include <stdio.h>

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

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

float sum(float x, float 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));
    return 0;
}

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;
}

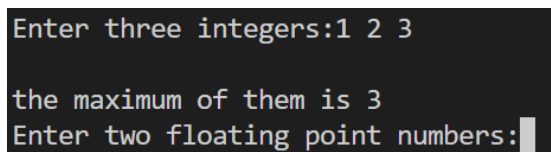
```

```

float sum(float x, float y)
{
    return x + y;
}

```

运行结果：



```

Enter three integers:1 2 3
the maximum of them is 3
Enter two floating point numbers:

```

图 4-2 源程序修改替换后的运行结果

(2) 用带参数的宏替换函数 `max`，来实现求最大数的功能。

修改后代码：

```

#include <stdio.h>

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

int max(int x, int y, int z);
float sum(float x, float 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", MAXC(a, b, c));

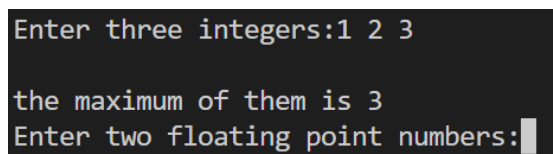
printf("Enter two floating point numbers:");
scanf("%f%f", &d, &e);
printf("\nthe sum of them is  %f\n", sum(d, e));
return 0;
}

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;
}

```

运行结果:



```

Enter three integers:1 2 3
the maximum of them is 3
Enter two floating point numbers:

```

图 4-3 源程序用宏定义修改替换后的运行结果

#### 4.2.3. 跟踪调试题

下面程序利用 R 计算圆的面积 s，以及面积 s 的整数部分。

```
1 #define R
2 void main(void)
3 {
4     float r, s;
5     int s_integer=0;
6     printf("input a number: ");
7     scanf("%f",&r);
8     #ifdef R
9         s=3.14159*r*r;
10        printf("area of round is: %f\n",s);
11        s_integer= integer_fraction(s);
12        printf("the integer fraction of area is %d\n", s_integer);
13        assert((s-s_integer)<1.0);
14    #endif
15 }
16 int integer_fraction(float x)
17 {
18     int i=x;
19     return i;
```

20 }

1) 修改程序，使程序编译通过且能运行；

修改后代码：

```
#include<stdio.h>

#include<assert.h>

#define R 1

int integer_fraction(float x);

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);
    printf("the integer fraction of area is %d\n", s_integer);
    assert((s - s_integer) < 1.0);
#endif

    return 0;
}

int integer_fraction(float x)
{
```



```
int i = x;
return i;
}
```

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？

当输入值为 3 时：

进入函数 `decimal_fraction` 时 `watch` 窗口中 `x` 为：

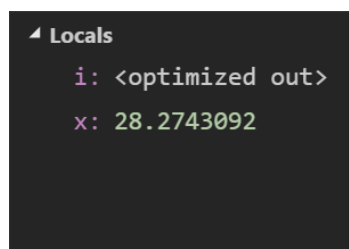


图 4-4 跟踪调试测试结果 (1)

在返回 `main` 时, `watch` 窗口中 `i` 为：

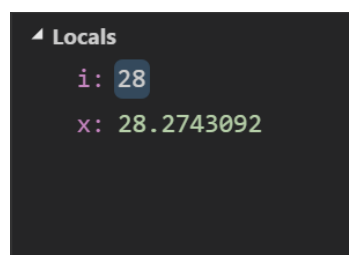


图 4-5 跟踪调试测试结果 (2)

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`。

运行结果：

```
input a number: 3
area of round is: 28.274309
the integer fraction of area is 28
PS D:\Code\CLAB\workcode\4th>
```

图 4-6 跟踪调试运行结果

#### 4.2.4. 编程设计题

(1) 三角形的面积是  $area = \sqrt{s(s-a)(s-b)(s-c)}$ ，其中  $s = (a+b+c)/2$ ，

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：定义两个带参数宏 S，AREA。进行计算

2) 程序清单

```
#include<stdio.h>
#include<math.h>

#define S(a, b, c) (((a)+(b)+(c))/2)
#define AREA(S, a, b, c) sqrt((S)*((S)-a)*((S)-b)*((S)-c))

int main(void){

    double s, a, b, c, area;

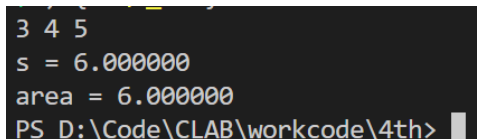
    scanf("%lf%lf%lf", &a, &b, &c);

    s = S(a, b, c);
    area = AREA(s, a, b, c);
    printf("s = %lf\narea = %lf", s, area);

    return 0;
}
```

3) 测试

对应测试的运行结果如图 4-8 所示。



```
3 4 5
s = 6.000000
area = 6.000000
PS D:\Code\CLAB\workcode\4th>
```

图 4-8 编程题 1 的测试运行结果

(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：

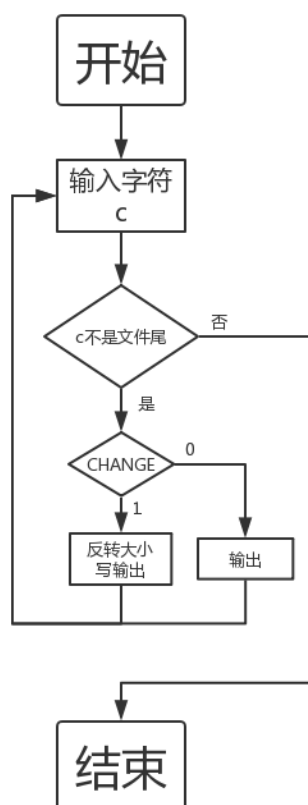


图 4-9 编程题 2 的算法流程图

2) 程序清单

```

#include<stdio.h>
#define CHANGE 1

int main(void){

```

```

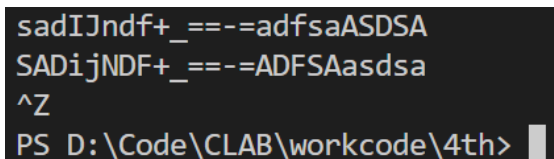
char c;
while((c = getchar()) != EOF){
#if CHANGE
    if(c >= 'A' && c <= 'Z')
        printf("%c", c + 32);
    else if(c >= 'a' && c <= 'z')
        printf("%c", c - 32);
    else
        putchar(c);
#else
    printf("%c", c);
#endif
}
return 0;
}

```

### 3) 测试

对应测试的运行结果如图 4-10、4-11 所示。

**#define CHANGE 1:**



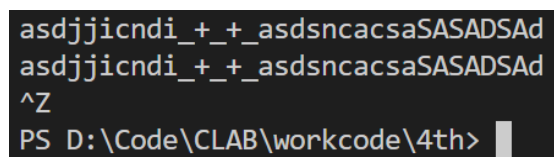
```

sadIjNdf+_==--adfsaASDSA
SADiJNDF+_==--ADFSaasdsa
^Z
PS D:\Code\CLAB\workcode\4th>

```

图 4-10 编程题 2 的测试运行结果 (1)

**#define CHANGE 0:**



```

asdjjicndi_+_asdsncacsaSASADSAd
asdjjicndi_+_asdsncacsaSASADSAd
^Z
PS D:\Code\CLAB\workcode\4th>

```

图 4-11 编程题 2 的测试运行结果 (2)

## 4.3 实验小结

第四次 C 语言上机实验，主要掌握并熟悉了宏定义和条件编译的使用并练

习了 assert 宏的使用，同时温故了上节课学习的调试并学会了单步执行、设置断点、观察变量值等方法。

### **(1) 改错实验**

改错题首先继续让我认识到了宏定义的不当使用，可能会造成一些奇怪的错误。这些错误进一步告诉我们，在使用宏定义这一技术时，应注意到的一系列问题。加深了我对预处理器与编译器之间的关系，也领会到了宏定义通过预处理器进行替换的方式，受益匪浅。而且，通过比对带参数宏以及函数的差别，认识到了二者在运行效率以及所需程序运行空间上的差别，大致上学会了如何选择是使用带参数宏还是函数来完成任务。

### **(2) 程序修改替换实验**

在此题中继续领会到了带参数宏的使用上的便捷与方便，与需要更多行的函数相比较更加省时省力。但同时，需要保证在宏定义中所使用的表达式的正确性。

### **(3) 跟踪调试题**

实际体会到了条件编译的使用方法，在实际的工程开发中，条件编译常用于避免头文件重复，或使得代码在不同环境下都可以正常编译，有更好的跨平台性，对于初学者来说可学习最基本的条件编译。同时，复习了使用调试器的方法。

### **(4) 程序设计实验**

在三角形面积的题目中，我将题目所需要的两个函数(半周长和三角形面积)都用宏定义的形式构造。仅仅两行简单的代码，就成功替换了多行的函数。同时，助教检查时提醒我，以后最好注意如果输入的值是负数等不合理的情况，要返回给用户其输入错误的信息。这样的良好习惯，在我们以后工作时开发软件，可以给用户带来更好的体验。

在电文转换的实验题中，按照题目要求对其进行了编写。使用条件编译，可以在类似情况下，省去对暂时不需要的代码进行编译的过程，节省编译时间与程序占用空间，同时提高效率。



## 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

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。

```
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 行函数调用时数组错误，正确形式为：

```
sort(a,10);
```

2) 第 18 行程序逻辑错误，正确形式为：

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

3) sort 函数未声明，正确形式为：

```
void sort(int b[], int n);
```

(2) 错误修改后运行结果：

预测：将数组 a 中整数按升序排序后输出，为

3 5 13 17 23 27 32 39 43 55

结果如图所示：



```
PS D:\Code> cd "d:\Code\CLAB\workcode\5th\" ; if ($?) { clang
g -Og -static-libgcc -fcolor-diagnostics --target=x86_64-w64-mingw32
) { .\5_1 }
3      5      13      17      23      27      32      39      43      55
PS D:\Code\CLAB\workcode\5th>
```

图 5-1 源程序改错题 1 的测试运行结果

### 5.2.2 源程序修改替换

- (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。

```
1  #include<stdio.h>

2  #define M 10

3  #define N 3

4  int main(void)

5  {

6  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
   */

7  int i, j, k;

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

9      a[i] = i + 1;

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

11 /* i 表示圈中人个数，初始为 M 个，剩 1 个人时结束循环；

12 j 表示当前报数人的位置 */

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

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

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

```

        b */

16        if(j)

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

18                                ;

19 }

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

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

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

23        return 0;

24 }

```

### 解答：

完善程序

```

#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? 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;
}

```

测试运行结果：应该将圈中人编号按出圈顺序输出，推测为：3 6 9 2 7 1 8 5 10 4

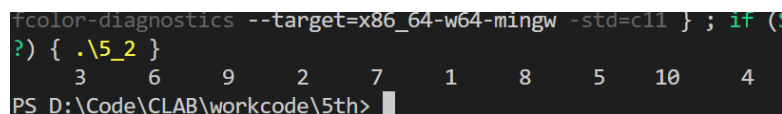


图 5-2 源程序修改替换题完善后的测试运行结果

- (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；

**解答：**

修改替换后的程序如下所示：

```

#include <stdio.h>

#define M 10

```

```

#define N 3

#define OUT -1


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 count = M;

    int luckydog;

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

    int m = 0, j = 0; //m 与下文 k 皆用于标识 a 数组下标

    while(count){
        int k = 1;

        while(k != N || a[m%M]==OUT)//模运算可达到使下标达到最
大后自动从头开始

            if(a[m++%M]!=OUT)
                k++;
    }
}

```

```

        b[j++] = a[m % M];

        a[m % M] = OUT;

        if(count == 1)

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

                if(b[i] != OUT)

                    luckydog = b[i];

        count--;

    }

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

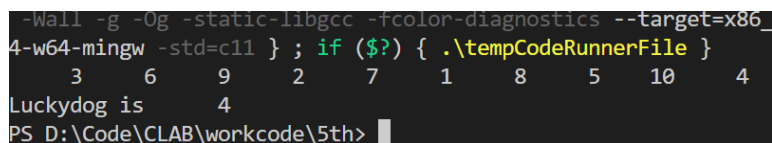
        printf("%6d", b[i]);

    printf("\nLuckydog is %6d\n", luckydog); /* 输出圈中最后一个人的编号 */

    return 0;

}

```



```

-Wall -g -Og -static-libgcc -fcolor-diagnostics --target=x86_
4-w64-mingw -std=c11 } ; if ($?) { .\tempCodeRunnerFile }
    3    6    9    2    7    1    8    5   10    4
Luckydog is      4
PS D:\Code\CLAB\workcode\5th>

```

图 5-3 源程序修改替换题修改替换后的测试运行结果

### 5.2.3 跟踪调试题

在下面所给的源程序中，函数 `strncat(s,t,n)` 本来应该将字符数组 `t` 的前 `n` 个字

符连接到字符数组 s 中字符串的尾部。但函数 `strncat` 在定义时代码有误，不能实现上述功能。请按下面的要求进行操作，并回答问题和排除错误。

- (1) 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？
- (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。

```

1  #include<stdio.h>

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

3  int main(void)

4  {

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

6      strncat(a, b, 4);

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

8      return 0;

9  }

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

11 {

12     int i = 0, j;

13     while(s[i++]);

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

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

```
16         s[i] = '\0';

17     }
```

**解答：**

- (1) 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？

答：`i` 的值为 23

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？

答：`s` 地址和 `t` 地址

- (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。

答：当 `while(s[i++])` ;执行完后，`i=23` 不为 22，`s` 的末尾为字符串结束标记，若 `i` 为 23，在字符串结束标记后没有意义。

修改后的程序为：

```
#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;

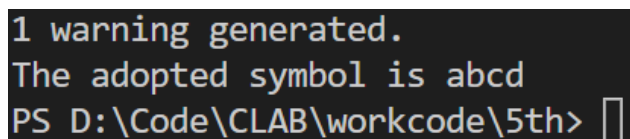
    while(s[i]) i++;

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

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

    s[i] = '\0';
}

```



```

1 warning generated.
The adopted symbol is abcd
PS D:\Code\CLAB\workcode\5th>

```

图 5-4 调试题的测试的运行结果

#### 5.2.4 程序设计题

- (1) 编写一个程序,从键盘读取数据, 对一个 3x4 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:

算法流程如图所示。



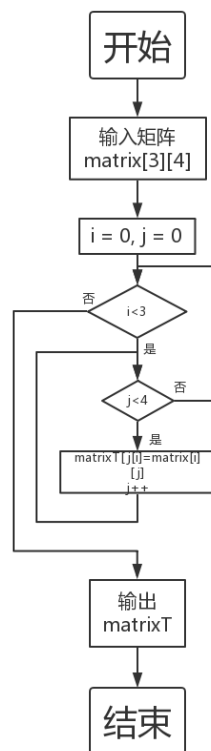


图 5-5 编程题 1 的程序流程图

## 2) 源程序清单

```
#include <stdio.h>
```

```
int main(void){
```

```
    int matrix[3][4] = {};
```

```
    int matrixT[4][3] = {};
```

```
    for (int i = 0; i < 3; i++){
```

```
        for (int j = 0; j < 4; j++){
```

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

```

    }

}

for (int i = 0; i < 3; i++)

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

        matrixT[j][i] = matrix[i][j];

for (int i = 0; i < 3; i++){

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

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

    printf("\n");

}

printf("*****\n");

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

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

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

    printf("\n");

}

    return 0;

}

```

### 3) 测试结果

```

1 2 3 4 5 6 7 8 9 10 11 12
    1      2      3      4
    5      6      7      8
    9     10     11     12
*****
    1      5      9
    2      6     10
    3      7     11
    4      8     12
PS D:\Code\CLAB\workcode\5th>

```

图 5-6 编程题 1 的运行结果

- (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：

程序流程图：

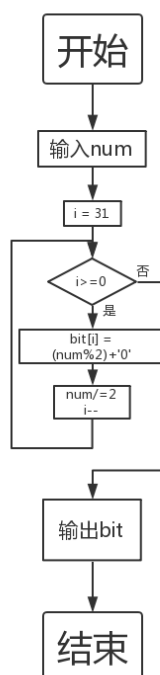


图 5-7 编程题 2 的程序流程图

2) 程序清单

```
#include <stdio.h>

int main(void){

    int num;

    char bit[33] = {};

    bit[32] = '\0';

    scanf("%d", &num);

    for (int i = 31; i >= 0; i--)

    {

        bit[i] = (num % 2) + '0';

        num /= 2;

    }

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

    return 0;

}
```

3) 测试

预测结果：输入 8 得到 00000000 00000000 00000000 00001000

[illegible]

图 5-8 编程题 2 的测试运行结果

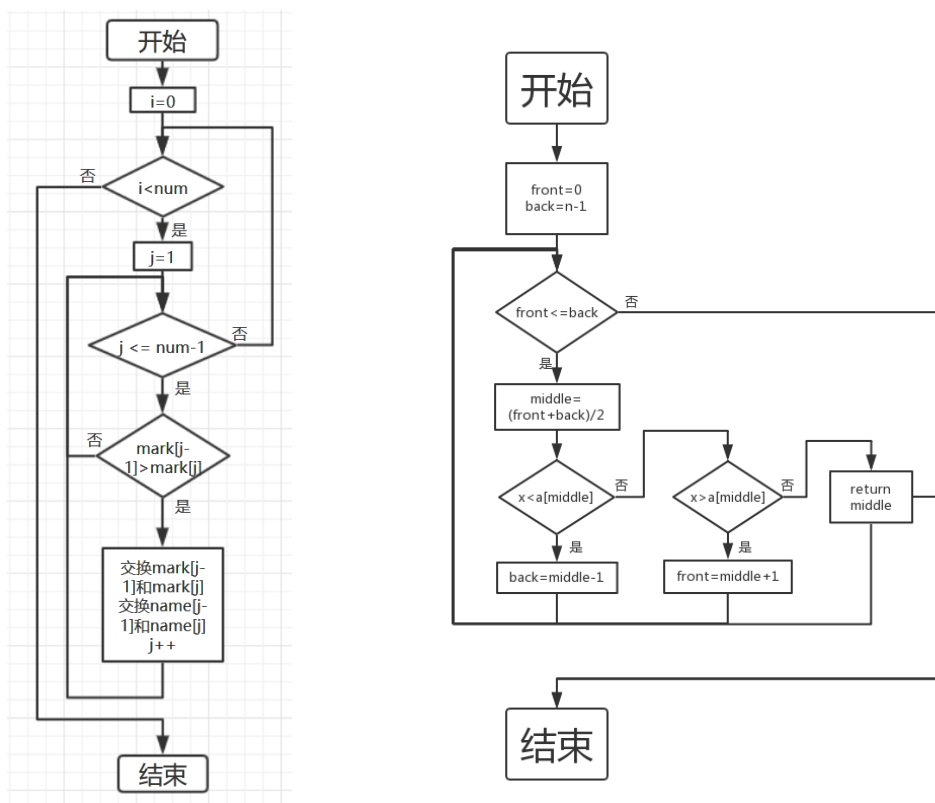
说明上述的运行结果与理论分析吻合,验证了程序的正确性。

- (3) 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:

分别利用一个字符数组和一个整数数组存储学生姓名和成绩，利用冒泡排序排列成绩的同时对姓名进行排列，最后同时输出即可得到排序后成绩表。利用有序表，进行二分查找。



A. 冒泡排序流程图

B. 二分法查找流程图

图 5-9 编程题 3 的程序流程图

## 2) 程序清单

```

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAXSTU 100

void sort(char name[][20], int mark[], int num);

void swap(int *a, int *b);

void swaps(char *a, char *b);

int BinarySearch(int a[], int x, int n);
    
```

```

int main(void){

    char name[MAXSTU][20];

    int mark[MAXSTU];

    int num;

    char option = 0;

    int n;

    int searchmark;

    int index;


    printf("Please input name and grade like this\nxiaoming 59\n");

    for (num = 0; scanf("%s %d", name[num], &mark[num]) != EOF;
num++)

        ;


    sort(name, mark, num-1);


    do{

        option = 0;

        fflush(stdin);

        printf("*****\nwhat do you want?\n");

        printf("1.print list\n2.search\nq to quit\n*****\n");

        option = getchar();
    
```

```

        if(option == '1')

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

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

        else if(option == '2'){

            printf("input a mark\n");

            scanf("%d", &searchmark);

            index = BinarySearch(mark, searchmark, n);

            if(index != -1)

                printf("find  %d  is  %s  grade\n",  mark[index],

name[index]);

            else

                printf("not found\n");

        }

    } while (option != 'q');

    return 0;

}

```

```

void sort(char name[][20], int mark[], int num){

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

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

            if(mark[j - 1] > mark[j]){

                swap(&mark[j - 1], &mark[j]);

```



```

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

    }

}

```

```

void swap(int *a, int *b){

    int temp = *a;

    *a = *b;

    *b = temp;

}

```

```

void swaps(char *a, char *b){

    char *temp = malloc(sizeof(char*));

    strcpy(temp, a);

    strcpy(a, b);

    strcpy(b, temp);

}

```

```

int BinarySearch(int a[], int x, int n){

    int front = 0, back = n - 1, middle;

    while(front<=back){

        middle = (front + back) / 2;

        if(x < a[middle])

            back = middle - 1;

        else if(x>a[middle])

```

```

        front = middle + 1;

    else

        return middle;

    }

    return -1;

}

```

### 3) 测试结果

程序运行结果如图所示。

```

Please input name and grade like this
xiaoming 59
a 59
b 20
c 12
^Z

what do you want?
1.print list
2.search
q to quit
*****
1
c 12
b 20
a 59

what do you want?
1.print list
2.search
q to quit
*****
2
input a mark
20
find 20 is b grade
*****
what do you want?
1.print list
2.search
q to quit
*****
q

```

图 5-10 编程题 3 的测试运行结果

### 5.2.5 选做题程序设计

编写并上机调试运行能实现以下功能的函数和程序。

(1) 编写函数 `strnins(s, t, n)`, 其功能是: 可将字符数组 `t` 中的字符串插入到字符数组 `s` 中字符串的第 `n` 个字符的后面。

1) 解题思路: 首先定位至字符数组 `s` 中第 `n` 个字符, 将其后的所有字符均向后移动字符数组 `t` 长度位, 再将 `t` 中字符插入空位中, 此算法简单易行但时间复杂度高。

2) 程序代码:

```
#include<stdio.h>
```

```
#include<string.h>

void strnins(char s[], char t[], int n);

int main(void) {
    char s[1000] = {}, t[1000] = {};
    int n;

    scanf("%s %s %d", s, t, &n);

    strnins(s, t, n);

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

    return 0;
}

void strnins(char s[], char t[], int n) {
    char temp;
    for (int i = strlen(t); i; i--)
        for (int i = strlen(s); i > n-1 ; i--)
            s[i + 1] = s[i];

    for (int i = 0; t[i]; i++) {
        s[n + i] = t[i];
    }
}
```

3) 运行结果截图:

```
123456789 abcd 5
12345abcd6789
PS D:\Code\CLAB\workcode\5th>
```

图 5-11 选做题 1 的测试运行结果

(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置 8 个皇后，任意两个皇后不能位于同一行、同一列或同一斜线（正斜线或反斜线）上，并输出所有可能的放法。

1) 解题思路：此题使用递归实现回溯法，由于未系统学习算法知识，并未独立完成。利用一个一维数组表示棋盘，每一个数组元素中数字表示该行皇后的列编号，问题转化为全排列生成问题。利用数学知识可使用行列编号来完成皇后间是否可以互相攻击的判断标准： $x0 == x1$  或  $x0 - y0 == x1 - y1$  即为攻击到。

2) 程序清单：

```
#include<stdio.h>

#define TRUE 1
#define FALSE 0
#define MAX 8

void print(char queen[]);
int judge(int index);

static char queen[MAX] = {};
static int counter = 0;

int main(void){

    judge(0);

    printf("there are totally %d solutions\n", counter);
```

```

        return 0;
    }

void print(char queen[]){
    printf("The %dth solution:\n*****\n", counter);
    for (int i = 0; i < 8; i++){
        for (int j = 0; j < queen[i]; j++)
            printf("# ");

        printf("Q ");

        for (int j = queen[i] + 1; j < 8; j++)
            printf("# ");

        printf("\n");
    }
    printf("*****\n");
}

int judge(int index){

    if(index == MAX){
        counter++;
        print(queen);
        return TRUE;
    }

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

```

```

int flag = TRUE;

queen[index] = i;//将 index 行的后移动到第 i 列位置,
(index, i)

for (int j = 0; j < index; j++){//j 是行数
    if(queen[j] == queen[index] ||
        (index - j) == (queen[index] - queen[j]) ||
        (j - index) == (queen[index] - queen[j])){
        flag = FALSE;//判断是否有必要递归
        break;
    }
}

if(flag)
    judge(index + 1);
}
}

```

3) 运行截图：（节选）

```

The 90th solution:
*****
# # # # # # # Q
# Q # # # # # #
# # # # Q # # #
# # Q # # # # #
Q # # # # # # #
# # # # # # Q #
# # # Q # # # #
# # # # # Q # #
*****

```

```

The 92th solution:
*****
# # # # # # # Q
# # # Q # # # #
Q # # # # # # #
# # Q # # # # #
# # # # # Q # #
# Q # # # # # #
# # # # # # Q #
# # # # Q # # #
*****

```

```

there are totally 92 solutions

```

图 5-12 选做题 2 的测试运行结果

## 5.3 实验小结

第五次 C 语言上机实验，主要学习并熟练了数组的使用，同时学习了二分查找算法和选择法排序算法的思想。

### (1) 改错实验

此题首先加深了我对排序算法的印象，其次，使我学会了如何向函数传递一个数组，在函数形参列表中声明的数组，在函数体内部，实际上通过指针运算完成，因此传值时应该传入指针常量，即数组首地址。同时我意识到，在函数内部用 `sizeof` 运算符无法得到数组的长度，会引起难以发现的错误。

### (2) 程序修改替换实验

约瑟夫环问题首先让我感受到，如果没有使用清晰的变量名称，没有写上必要的注释，阅读程序代码将会十分困难。在完成语句填空以及改写后，老师告诉我约瑟夫环还可用循环链表进行处理，十分便捷。这让我意识到了需要学习的内容还很多，还需继续努力。

### (3) 跟踪调试题

在继续学习使用调试器的同时，也让我理解了为何对于边界要格外注意的问题

### (4) 程序设计实验

第一题向我介绍了一个新的数学概念，同时加强了我对二维数组中行和列的理解，也让我认识到，数组这一概念实际上可以用于模拟数学中 N 维空间坐标系的概念。

第二题重温进制转换的同时，进一步熟悉了对字符数组的使用。

第三题首先使我熟悉了二分查找的原理以及使用细节，也另外编写程序比对了遍历查找与二分查找的速度差异，认识到，大多数情况下，最直接的算法都不会是最优的，需要经过思考才可以提高效率。然后，在对排序时对字符数组的交换上出现了一些困难，在使用函数完成交换字符数组这一步时，由于对函数形参中数组与指针的关系以及一些概念的模糊，导致耗费了大量时间，在重新复习了相关知识，加上老师讲解之后，完成了

这一工作。另外，在这题中，设计了一个简单的命令行界面下的 UI，虽然简陋，但是也感受到了一个好的 UI 的作用。这其中为了解决循环输出菜单的问题，学习使用了标准库函数 `fflush()`，了解了一些输入输出流的深入内容。

#### (5) 选做题

第一个选做题，选择了最易想到的方式。将字符数组 `s` 中第 `n` 位往后的数组元素全部后移直到空位足够容纳字符数组 `t`，再将字符数组 `t` 填入空位中。经过助教提醒，我意识到这种方式效率极低，想到在之前二分查找处得到的启示，对此题要求的函数功能实现的算法进行了改进，用足够长的数组来存储数组 `s` 的后半段，再将三个数组按顺序连接，避免了耗时的数组元素移动操作。

第二个选做题，遇到了著名的数学问题——八皇后问题。起初利用二维数组模拟棋盘遍历查找的方式实现，但是首先循环嵌套相当多层，代码看起来混乱、难以理解。且这种方法耗时极高，在写报告时计算得到的结果，这种全部遍历的算法，一共有近 45 亿种可能的结果，需要从这其中找到所有解相当耗费时间，而且花费了大量时间生成与判断毫无价值的数据。因此，在咨询老师及查询资料后，了解到了什么是递归回溯算法，可以在已知无法继续摆放任何棋子之后，回到上一步操作继续寻找下一个安全位置的组合，可以节省大量的时间。同时使用一维数组来模拟棋盘，本身也排除了大量无用组合，将数据范围缩减至大概八万种，比起之前的范围，可以说是非常优秀的算法了。经过这一题，我继续意识到自己还需要学习很多知识，单方面地认为计算机运算速度很快，可以不加考虑地使用暴力求解的方式是很懒惰，很不负责的。同时激起了我对学习算法的热情，现在很期待将来的数据结构课程以及算法课程。

**总结：**此次上机实验题很有难度，也十分有趣，想说的话已经在各题目小结中说完，希望我自己可以在以后的学习中保持热情。



## 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。

```
#include "stdio.h"

void main(void)
{
    float *p;
    scanf("%f",p);
    printf("%f\n",*p);
}
```

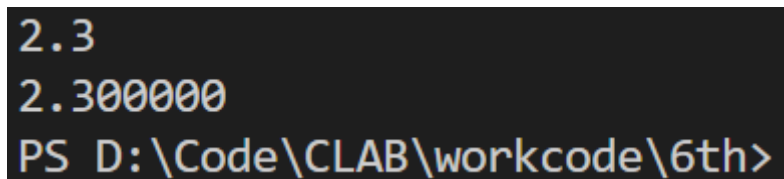
**解答：**

(1) 错误修改：

第 4 行使用了未初始化的变量 p，正确形式为：

```
float a,*p=&a;
```

(2) 错误修改后运行结果如图 6-1 所示：



```
2.3
2.300000
PS D:\Code\CLAB\workcode\6th>
```

图 6-1 改错题的测试的运行结果

### 6.2.2. 源程序修改替换题

(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>
int main(void)
{
    char *(*p)(char *, const char *);
    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;
```

```

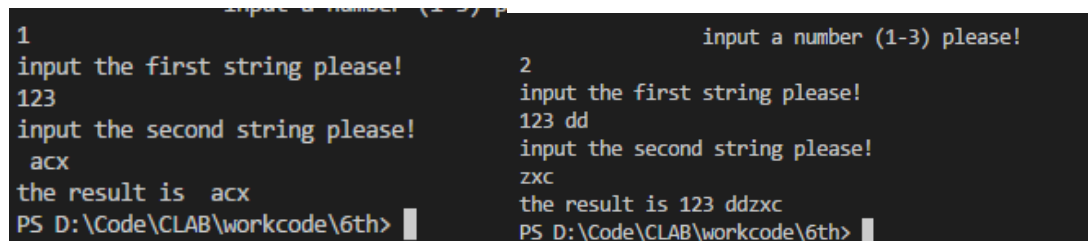
        break;
    case 2:
        p = strcat;
        break;
    case 3:
        goto down;
}
getchar();
printf("input the first string please!\n");
i = 0;
gets(a);
printf("input the second string please!\n");
i = 0;
gets(b);
result = p(a, b);
printf("the result is %s\n", result);

down:
    ;

return 0;
}

```

运行结果:



```

1 input a number (1-3) please!
input the first string please!
123
input the second string please!
acx
the result is acx
PS D:\Code\CLAB\workcode\6th>

2 input a number (1-3) please!
input the first string please!
123 dd
input the second string please!
zxc
the result is 123 ddzxc
PS D:\Code\CLAB\workcode\6th>

```

图 6-2 替换题（1）的测试的运行结果

### 6.2.3. 跟踪调试题

```
#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？



图 6-4 跟踪调试题测试结果 (1)

返回 main 时, watch 窗口中 s 为何值？



图 6-5 跟踪调试题测试结果 (2)

(2) 排除错误，使程序输出结果为：

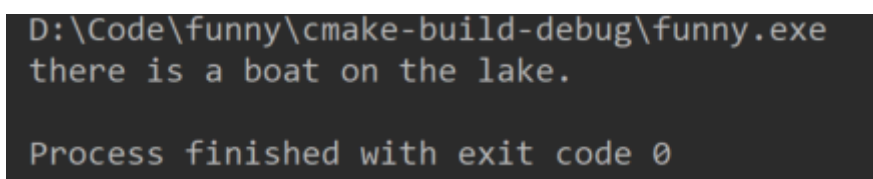
there is a boat on the lake.

程序代码：

```
#include "stdio.h"
```

```
char *strcpy(char *,char *);
int main(void)
{
    char a[20],b[60]="there is a boat on the lake.";
    printf("%s\n",strcpy(a,b));
    return 0;
}
char *strcpy(char *s,char *t)
{
    char *a = s;
    while(*s++=*t++);
    return (a);
}
```

运行结果：



```
D:\Code\funny\cmake-build-debug\funny.exe
there is a boat on the lake.

Process finished with exit code 0
```

图 6-6 跟踪调试题测试运行结果

#### 6.2.4. 编程设计题

(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。

1) 解题思路：

利用指向 `char` 型的指针取出长整型变量中的单个字节，再分别取出高低 4 位。

2) 源程序清单

```
#include<stdio.h>
```

```
int main(void){
```

```

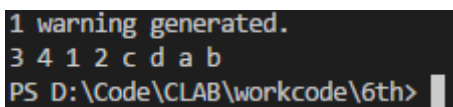
long int l = 0xabcd1234;
char *c = (char *)&l;
char ch[8];
for (int i = 0; i < 8; i+=2){
    ch[i] = (*c & 0xf0) >> 4;
    ch[i + 1] = *c & 0xf;
    *(c++);
}

for (int i = 0; i < 8; i++){
    if(ch[i] >= 0 && ch[i] <= 9)
        printf("%c ", ch[i] + '0');
    else if(ch[i] >= 0xa && ch[i] <= 0xf)
        printf("%c ", ch[i] + 'a' - 10);
}

return 0;
}

```

### 3) 测试



```

1 warning generated.
3 4 1 2 c d a b
PS D:\Code\CLAB\workcode\6th>

```

图 6-7 编程题 1 的测试运行结果

- (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：**

- 2) 算法流程如图所示

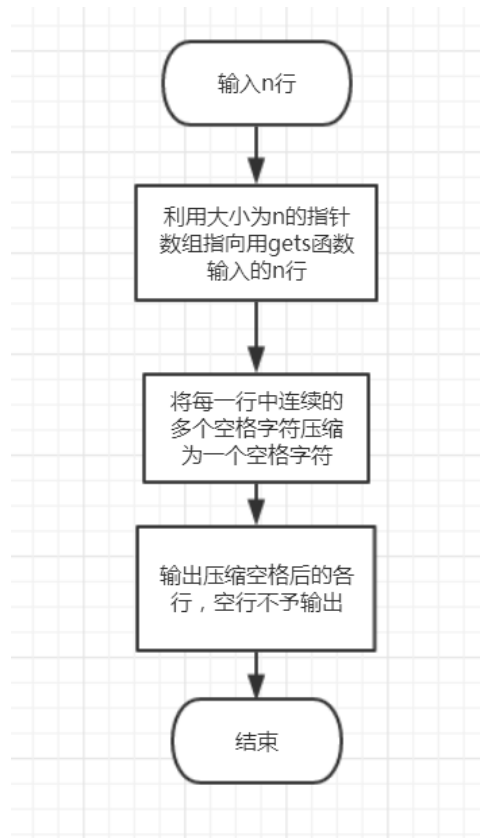


图 6-8 编程题 2 的程序流程图

### 3) 源程序清单

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void fvc(char *a);

int main(void){

    char a[100];

    while(gets(a))
    
```

```

        fvc(a);

    return 0;
}

void fvc(char *a){

    char *ch;

    int flag = 1;

    if(strlen(a) == 0)

        return;

    for (ch = a; *ch; ++ch){

        if (*ch == '\n')

            return;

        else if (*ch != ' ' && *ch != '\t' && *ch != '\n'){

            putchar(*ch);

            flag = 1;

        }

        else if (flag == 1){

            putchar(' ');

            flag = 0;

        }

    }

```



```

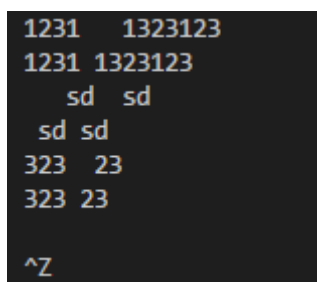
    }

    putchar('\n');

}

```

### 3)测试



```

1231 1323123
1231 1323123
sd sd
sd sd
323 23
323 23
^Z

```

图 6-9 编程题 2 的测试运行结果

(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：

- 计算每个学生各门课程平均成绩；
- 计算全班每门课程的平均成绩；
- 分别统计低于全班各门课程平均成绩的人数；
- 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) 算法流程如图所示

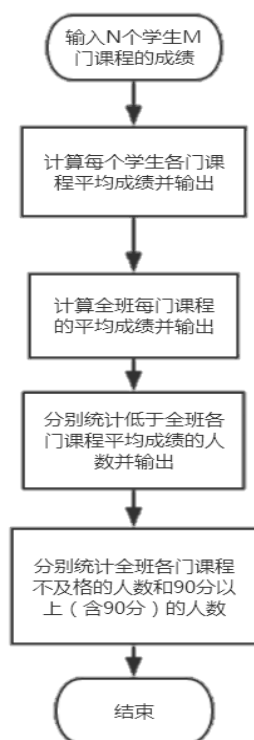


图 6-10 编程题 3 的程序流程图

## 2) 源程序清单

```

#include<stdio.h>

#define N 30

#define M 6

#define MAXNAME 20

double StuAverage(double (*grade)[M], int stunum, int subsum);

double SubAverage(double (*grade)[M], int stusum, int subnum);

int Lower(double (*grade)[M], int stusum, int subnum);

int Failed(double (*grade)[M], int stusum, int subnum);

int Excel(double (*grade)[M], int stusum, int subnum);
    
```

```

int main(void){

    double grade[N][M] = {}; //N 为最大学生数，M 为最大科目数

    char subname[M][MAXNAME] = {};

    int subsum = 1, stusum = 1;

    int subnum, stunum;

    do{

        printf("\nPlease input the amount of subject and students:(0 to
quit)");

        scanf("%d%d", &subsum, &stusum);

        if(subsum==0)

            break;

        printf("Input the names of those subs\n");

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

            scanf("%s", subname[i]);

        }

        printf("Input the grade of every one\n");

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

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

                scanf("%lf", &grade[i][j]);
    }
}

```

```

printf("Whose Average?\n");

scanf("%d", &stunum);

printf("this classmate's average is%lf\n", StuAverage(grade,
stunum, subsum));


printf("Which Subject\n");

scanf("%d", &subnum);

printf("the SubAverage is %lf\n", SubAverage(grade, stusum,
subnum));

printf("there is %d student(s) lower than the subaverage.\n",
Lower(grade, stusum, subnum));

printf("And %d student(s) failed, %d student(s) gets a A.\n",
Failed(grade, stusum, subnum), Excel(grade, stusum, subnum));


} while (subsum);


return 0;

}

```

```

double StuAverage(double (*grade)[M], int stunum, int subsum){

double result, sum = 0;

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

sum += (*(grade+stunum-1)+i);

result = sum / (double)subsum;

```

```

        return result;

    }

double SubAverage(double (*grade)[M], int stusum, int subnum){

    double result, sum = 0;

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

        sum += (*(grade+i)+subnum-1);

    result = sum / (double)stusum;

    return result;

}

int Lower(double (*grade)[M], int stusum, int subnum){

    double Average = SubAverage(grade, stusum, subnum);

    int result = 0;

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

        if(*(grade+i)+subnum-1 <= Average)

            result++;

    return result;

}

int Failed(double (*grade)[M], int stusum, int subnum){

```

```

int result = 0;

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

    if (*(*(grade+i)+subnum-1) < 60)

        result++;

return result;
}

int Excel(double (*grade)[M], int stusum, int subnum){

    int result = 0;

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

        if (*(*(grade+i)+subnum-1) >= 90)

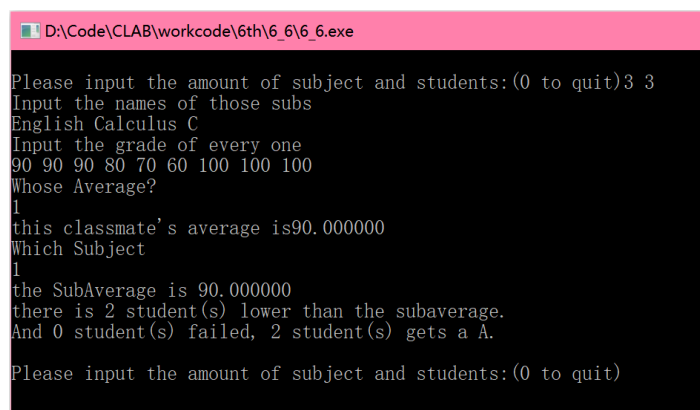
            result++;

    return result;

}

```

### 3)测试



```

D:\Code\CLAB\workcode\6th\6_6\6_6.exe
Please input the amount of subject and students:(0 to quit)3 3
Input the names of those subs
English Calculus C
Input the grade of every one
90 90 90 80 70 60 100 100 100
Whose Average?
1
this classmate's average is90.000000
Which Subject
1
the SubAverage is 90.000000
there is 2 student(s) lower than the subaverage.
And 0 student(s) failed, 2 student(s) gets a A.
Please input the amount of subject and students:(0 to quit)

```

图 6-11 编程题 3 的测试运行结果

### 6.2.5 选做题

(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

1) 解题思路:

利用字符串存储输入的大数以及运算结果, 按位对齐之后相加, 并记录是否进位。

2) 程序清单:

```
#include<stdio.h>

#include<string.h>

void reverse(char *string, int length);

int main(void) {

    char num1[50] = {}, num2[50] = {}, sum[55] = {};
    int length1, length2, length3;
    int acc = 0;
    int i, bit;

    printf("please input a addition expression(^z to quit)");

    while(scanf("%s %s", num1, num2) != EOF) {

        for (int j = 0; j < 55; j++)
            sum[j] = 0;

        length1 = strlen(num1);
```

```

length2 = strlen(num2);
reverse(num1, length1-1);
reverse(num2, length2-1);

//printf("%s %s", num1, num2);
for (i = 0; i < length1 && i < length2; i++){
    if(num1[i] == '.'){
        sum[i] = '.';
    }
    else{
        bit = num1[i] + num2[i] - 2 * '0' + acc;
        sum[i] = bit % 10 + '0';

        if(bit<10)
            acc = 0;
        else
            acc = 1;
    }
}

if(i < length1)
    for (; i < length1; i++){
        bit = num1[i] - '0' + acc;
        sum[i] = bit % 10 + '0';
        if(bit<10)
            acc = 0;
        else

```



```

        acc = 1;
    }

    if(i < length2)
        for (; i < length2; i++){
            bit = num2[i] - '0' + acc;
            sum[i] = bit % 10 + '0';
            if(bit<10)
                acc = 0;
            else
                acc = 1;
        }

    if(acc == 1)
        sum[i++] = '1';

    sum[i] = '\0';

    length3 = strlen(sum);
    reverse(sum, length3 - 1);

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

}

```

```

    return 0;
}

void reverse(char *string, int length){
    for (int i = 0; i <= length; i++, length--){
        char tempc = string[i];
        string[i] = string[length];
        string[length] = tempc;
    }
}

```

### 3) 运行结果:

经 Windows 自带计算器验算结果正确

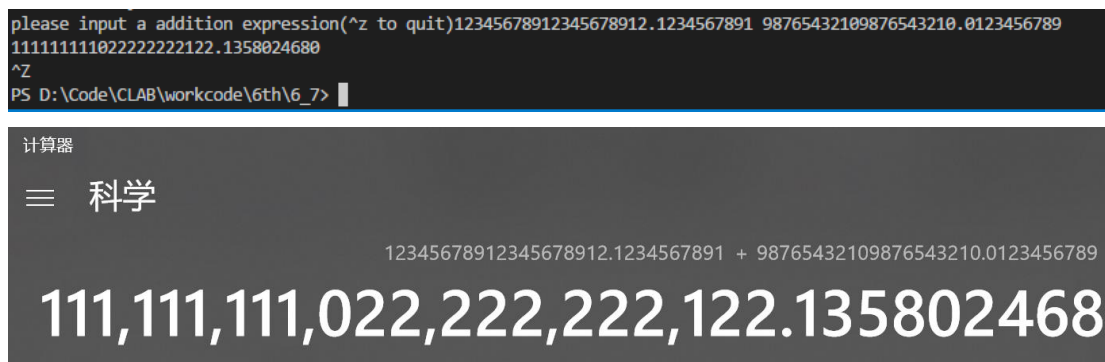


图 6-12 选做题 1 的测试运行结果

(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。

#### 1) 程序思路:

利用复杂声明, 定义一个函数指针数组, 分别指向 `strcpy` 与 `strstr` 两个字符处理函数并通过函数指针调用函数。

#### 2) 程序清单

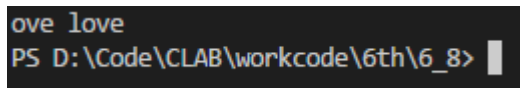
```
#include <stdio.h>
```

```
#include <string.h>

int main(int argc, char *argv[]) {
    char a[6] = "1love2";
    char b[5] = "love";
    char *(*p[2])(const char *, const char *) = {strcpy,
    strstr};

    printf("%s %s", p[1](a, b), p[0](a, b));
    return 0;
}
```

3) 运行结果:

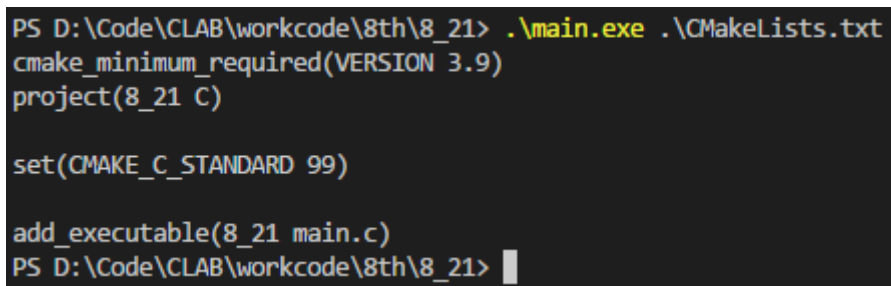


```
ove love
PS D:\Code\CLAB\workcode\6th\6_8>
```

图 6-13 选做题 2 的测试运行结果

### (三) 指定 main 函数的参数

- 1) 此题未使用原题中介绍方法，在命令行中直接传入参数
- 2) 程序可显示以参数为文件名的文本文件的内容
- 3) 结果:



```
PS D:\Code\CLAB\workcode\8th\8_21> .\main.exe .\CMakeLists.txt
cmake_minimum_required(VERSION 3.9)
project(8_21 C)

set(CMAKE_C_STANDARD 99)

add_executable(8_21 main.c)
PS D:\Code\CLAB\workcode\8th\8_21>
```

图 6-14 选做题 3 的测试运行结果

## 6.3 实验小结

第六次 C 语言上机实验，主要学习并熟练了指针的说明、赋值与使用。掌握了用指针引用数组的元素与指向数组的指针、字符数组字符串、指针数组及字符指针数组、指针函数与函数指针以及带有参数的 main 函数的用法。

### (1) 改错实验

改错实验还原了我们这种初学者在使用指针时最常见的错误，故意设置了一个悬挂指针并进行了引用和解引用，访问到了可能非法的内存地址，在将其修改之后，我了解到了悬挂指针的问题关键所在。如果随意地使用指针，轻则程序崩溃，重则可能引发一下潜在的问题，导致出现极难发现的 BUG。

### (2) 程序修改替换实验

此题考察了函数指针的运用，运用 gets 函数进行输入。此时我对原题中设置的变量 i 的存在产生了疑问，经思考后得知题目本意可能是利用循环以及 scanf 函数配合，避免 gets 函数带来的内存安全问题。同时我对指针函数以及函数指针的概念有了正确的认知，一个本质上是返回指针的函数，另一个是指向一个函数的指针变量，二者根本不同，在实际使用之后，明白了这一概念。

### (3) 跟踪调试题

此题解题过程可谓一波三折。起初编译运行时，此题并无错误，虽有 warning，但是可以正常输出，便并没有处理此题。在助教检查作业时，我突然发现，此题在单步运行时，并不能跳转到文件中定义的 strcpy 函数。这时我怀疑，编译器的自动优化功能忽视了与标准库函数同名的函数，调用了标准库函数。在给题中函数重新命名后，验证了猜想。随后发现了在 while 循环中，对指针 s 与 t 进行的自增操作会使两个指针最终都指向字符串的末尾，因此无法正常输出。在适当扩大字符数组 a 的大小使其可以容纳字符串，并在定义的函数中新加入指针保存字符串首地址并返回他之后，此题得解。

### (4) 程序设计实验

第一题，主要让我明白了为何指针也有类型之分。通过这题，我认识到对指针进行操作时，实际上考虑其指向的变量的数据类型是十分重要的。指针本身只存储内存地址，但是需正确处理所指向变量，就需要知道应该从所指向地址向后几位以获得完整的数据，指针与整数加减法，也依赖此进行，数组的实现，也需要得知数据类型以及其占据的内存大小。

第二题，熟悉了指针数组的使用以及处理，指针数组与普通数组实际上并无差别，熟悉了对指针数组的操作。

第三题，计算学生成绩的各项数据，此题利用二维数组，计算学生平均成绩，可用行来存储每个学生的各项成绩，列自然成为各科目的每人成绩，再在嵌套循环中选择下标处理即可。此题主要让我意识到了模块化的重要，编写的四个函数可以很方便的得到再利用。同时，对于这类略有实用性的程序，在助教提醒下，还应该做好 UI 的设计，以方便使用与测试。

#### (5) 选做题

此次选做题中，第一题利用了我之前曾编写过的超大整数相加的程序修改而来，加上了小数点判断后便可正常工作，利用字符数组逐位相加并进位。第二个选做题主要还是在熟悉函数指针数组的使用，但是原题中略有错误，strcmp 函数并不能作为题目要求中指针指向的对象。Strcmp 函数返回整型数值，以表示参数中两个字符串的排序关系。第三题对 main 函数的参数的使用，使用了命令行进行完成，没有使用 IDE 的功能。

总之，此次 C 语言上机实验，让我领会到了指针的各种用途，对指针不再惧怕，在之后的学习与工作中，应该有勇气面对指针的使用。



## 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, 'A', u}, {100, 'B', 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	<code>(++p)-&gt;x</code>	100	100
2	<code>p++, p-&gt;c</code>	B	B
3	<code>*p++-&gt;t, *p-&gt;t</code>	x	x
4	<code>*(++p)-&gt;t</code>	x	x
5	<code>*++p-&gt;t</code>	V	V

6	++*p->t	V	V
---	---------	---	---

实现程序代码：

```
#include<stdio.h>
```

```
int main(void)
```

```
{
```

```
    char *A, *B, *C, *D;
```

```
    char u[]="UVWXYZ";
```

```
    char v[]="xyz";
```

```
    struct T{
```

```
        int x;
```

```
        char c;
```

```
        char *t;
```

```
    }a[]={{{11,'A',u},{100,'B',v}},*p=a;
```

```
    A=u;B=v;C=a[0].t;D=a[1].t;
```

```
    printf("1.(++p)->x\t%d\n",(++p)->x);
```

```
    p=a;*u=*A;*v=*B;a[0].t=C;a[1].t=D;
```

```
    printf("2.p++,p->c\t%c\n",(p++,p->c));
```

```
    p=a;*u=*A;*v=*B;a[0].t=C;a[1].t=D;
```

```
    printf("3.*p++->t,*p->t\t%c\n",(*p++->t,*p->t));
```

```
    p=a;*u=*A;*v=*B;a[0].t=C;a[1].t=D;
```

```
    printf("4.*(++p)->t\t%c\n",(*(++p)->t));
```

```
    p=a;*u=*A;*v=*B;a[0].t=C;a[1].t=D;
```

```
    printf("5.*++p->t\t%c\n",(*++p->t));
```

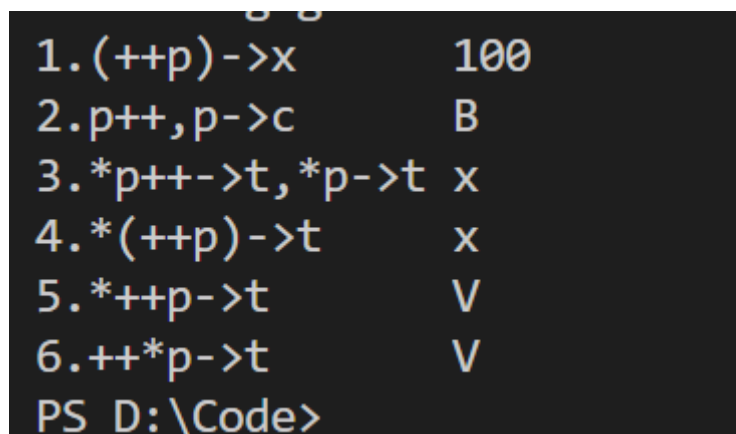


```
p=a;*u=*A;*v=*B;a[0].t=C;a[1].t=D;

printf("6.++*p->t\t%c\n",(++*p->t));

return 0;
}
```

运行结果：



```
1. (++p)->x      100
2. p++, p->c      B
3. *p++->t, *p->t x
4. *(++p)->t     x
5. *++p->t        V
6. ++*p->t        V
PS D:\Code>
```

图 7-1 程序验证题的测试的运行结果

## 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。

（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 */
}

```

head 本应用来保存链表的首地址，而源代码中函数传入的是 head 的副本，应改为传址引用。

**解答：**

修改后代码如下：

```

#include <stdio.h>
#include <stdlib.h>

struct s_list{
    int data;          /* 数据域 */
    struct s_list *next; /* 指针域 */
} ;

void create_list_v1(struct s_list **headp, int *p);
void create_list_v2(struct s_list **headp, int *p);

int 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_v2(&head, s);      /* 创建新链表 头指针地址传递
给内部函数的指针*/

    p=head;                        /*遍历指针 p 指向链头 */
    while(p) {
        printf("%d|", 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");

    return 0;
}

void create_list_v1(struct s_list **headp,int *p){//此处纠错，双重指
针，指向 main 函数中指针 head

    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 */

```

---

```

}

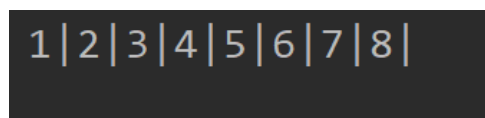
void create_list_v2(struct s_list **headp, int *p) { // 后进先出表，即栈
    struct s_list * loc_head=NULL, *tail;

    if (p[0]==0)                                /* 相当于*p==0 */
        ;
    else {                                        /* tail 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;
        tail=loc_head;                         /* loc_head 指向第一个结点 */

        while(*p) {
            tail=(struct s_list *)malloc(sizeof(struct s_list));
            tail->next=loc_head;
            loc_head=tail;
            tail->data=*p++;
        }
        //tail->next=NULL;
    }
    *headp=loc_head;
}

```

替换后运行结果如下



```
1|2|3|4|5|6|7|8|
```

图 7-2 替换题（1）的测试的运行结果

(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：**

程序代码如下：

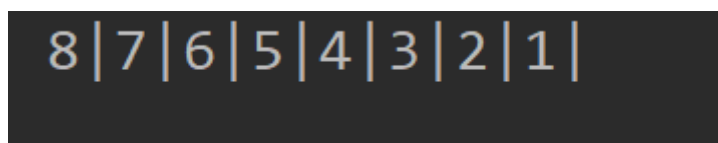


图 7-3 替换题（2）的测试的运行结果

### 3. 编程设计题

(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` ( $i=0,1,2,\dots,7$ ) 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：**

#### 3) 算法思路：

此题对于函数的定义的提示有误，在解题时加以改正，输出了当前函数的编号以及传入函数的参数。同时按照题意，应该使用无符号短整形而不仅仅是无符号整形数据。此题主要利用了字段结构的特性，对一个无符号短整形进行按位操作，利用函数指针数组指向八个函数，方便操作。

#### 4) 源程序清单

```
#include <stdio.h>

typedef struct bits{

    unsigned short bit0 : 1;

    unsigned short bit1 : 1;

    unsigned short bit2 : 1;

    unsigned short bit3 : 1;

    unsigned short bit4 : 1;

    unsigned short bit5 : 1;

    unsigned short bit6 : 1;

    unsigned short bit7 : 1;

} BITS;

void f0(BITS b);

void f1(BITS b);

void f2(BITS b);

void f3(BITS b);

void f4(BITS b);

void f5(BITS b);

void f6(BITS b);

void f7(BITS b);
```

```

int main() {

    BITS bit= {1, 1, 1, 1, 0, 1, 0, 1};

    void(*p[8])(struct bits b);

    p[0] = f0;
    p[1] = f1;
    p[2] = f2;
    p[3] = f3;
    p[4] = f4;
    p[5] = f5;
    p[6] = f6;
    p[7] = f7;

    if (bit.bit0)  p[0](bit);
    if (bit.bit1)  p[1](bit);
    if (bit.bit2)  p[2](bit);
    if (bit.bit3)  p[3](bit);
    if (bit.bit4)  p[4](bit);
    if (bit.bit5)  p[5](bit);
    if (bit.bit6)  p[6](bit);
    if (bit.bit7)  p[7](bit);

    return 0;
}

```



```
}
```

```
void f0(BITS b){
    printf("the function 0 is called!The argv is %x\n", b);
}
```

```
void f1(BITS b){
    printf("the function 1 is called!The argv is %x\n", b);
}
```

```
void f2(BITS b){
    printf("the function 2 is called!The argv is %x\n", b);
}
```

```
void f3(BITS b){
    printf("the function 3 is called!The argv is %x\n", b);
}
```

```
void f4(BITS b){
    printf("the function 4 is called!The argv is %x\n", b);
}
```

```
void f5(BITS b){
    printf("the function 5 is called!The argv is %x\n", b);
```

```

    }

    void f6(BITS b){

        printf("the function 6 is called!The argv is %x\n", b);

    }

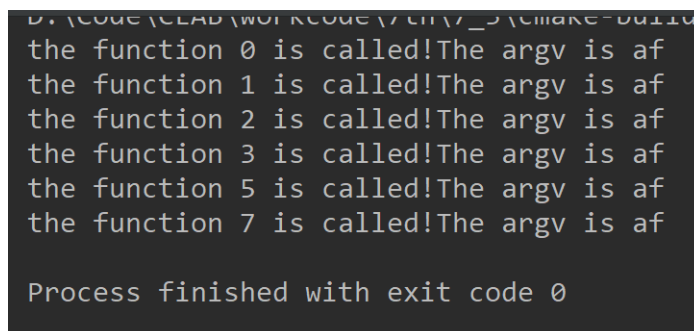
    void f7(BITS b){

        printf("the function 7 is called!The argv is %x\n", b);

    }

```

## 5) 测试



```

D:\CODE\CLAB\WORKCODE\7_1\7_5\make-build
the function 0 is called!The argv is af
the function 1 is called!The argv is af
the function 2 is called!The argv is af
the function 3 is called!The argv is af
the function 5 is called!The argv is af
the function 7 is called!The argv is af

Process finished with exit code 0

```

图 7-4 编程题 1 的测试运行结果

(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。

**解答：**

## 1) 算法思路：

对于必做的程序设计题部分，首先建立一个带头结点的单链表，在创建链表时，就将学生信息输入。利用遍历链表查找学号的方式查找指定学生并修改其指定数据。自此：必做题部分基本完成。

此题难度主要集中于选做部分要求对指针域进行操作进行排序的部分。由于在创建链表时已经添加了空的头结点，指针操作时略微简化，在更改了创建链表的函数之后，创建成双链表，可以进一步简化操作。

## 2) 源程序清单

## 1. main.c

```
#include "LinkedList.h"

int main(void){

    StuInfoList Class8Info = malloc(sizeof(StuInfoList));

    Position head = malloc(sizeof(Position));

    StuInfoList sortedList = malloc(sizeof(struct StuNode));

    int StuSum;

    int StuNum;

    printf("How many people in Class:");

    scanf("%d", &StuSum);
```

```
head = newStu(StuSum);
```

```
printf("Who's data need to be change? 0 to skip");
```

```
scanf("%d", &StuNum);
```

```
if(StuNum)
```

```
    Change(head, StuNum);
```

```
StuAverage(head, StuSum);
```

```
TraversalOutput(Class8Info, head);
```

```
sortedList = PtrSort(head);
```

```
TraversalOutput(Class8Info, sortedList);
```

```
return 0;
```

```
}
```

## 2. LinkedList.h

```
//
```

```
// Created by Think on 2017/12/27.
```

```
//
```

```
#ifndef INC_7_4_LINKEDLIST_H
```

```
#define INC_7_4_LINKEDLIST_H
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
struct StuNode{ //定义成绩单中链表结点
```

```
    struct StuNode *prev;
```

```
    int StuNum;
```

```
    char StuName[20];
```

```
    int Calculus;
```

```
    int English;
```

```
    int Physics;
```

```
    int CPL;
```

```
    double Average;
```

```
    struct StuNode *next;
```

```
};
```

```
typedef struct StuNode *PtrToNode; //类型定义指向结点的指针
```

```
typedef PtrToNode StuInfoList; //用于列表
```

```
typedef PtrToNode Position; //用于位置指示
```

```
Position FindStu(int StuNum, StuInfoList L); //寻找特定学生函数原型
```

```
StuInfoList newStu(int StuSum); //添加新学生
```

Position FindPre(Position head, Position target); // 寻找前一个结点

int IsEmpty(StuInfoList L);

int IsLast(Position P, StuInfoList L);

void TraversalOutput(StuInfoList L, Position Head);

void PrintNode(Position P);

void Change(Position P, int StuNum);

void StuAverage(Position P, int StuSum);

void Delete(StuInfoList L, Position target);

StuInfoList SelSort(StuInfoList L);

StuInfoList PtrSort(StuInfoList L);

#endif //INC\_7\_4\_LINKEDLIST\_H

### 3. Function.c

//

// Created by jyxx on 2017/12/27.

//

#include "LinkedList.h"

Position FindStu(int StuNum, StuInfoList L){

Position P;

P = L->next;

```

while(P != NULL && P->StuNum != StuNum)

    P = P->next;

return P;
}

```

```

StuInfoList newStu(int StuSum){

    Position head, tail;

    head = tail =(Position)malloc(sizeof(struct StuNode));

    //head->prev = NULL;

    tail->prev = NULL;

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

        tail->next    =    (Position)malloc(sizeof(struct
StuNode));

        tail->next->prev = tail;

        tail = tail->next;

        printf("Input the student's Num:");

        scanf("%d", &tail->StuNum);

        printf("Input his/her name:");

        scanf("%s", tail->StuName);

        printf("Input the grade of 4 subject:\nEnglish
Calculus Physics CPL\n");

```

```
scanf("%d%d%d%d", &tail->English,
&tail->Calculus, &tail->Physics, &tail->CPL);
```

```
}
```

```
tail->next = NULL;
```

```
return head;
```

```
}
```

```
int IsEmpty(StuInfoList L){
```

```
    return L->next == NULL;
```

```
}
```

```
int IsLast(Position P, StuInfoList L){
```

```
    return P->next == NULL;
```

```
}
```

```
void TraversalOutput(StuInfoList L, Position Head){
```

```
    Position prev = Head;
```

```
    while(!IsLast(prev, L)){
```

```
        prev = prev->next;
```

```
        PrintNode(prev);
```

```
}
```



```
    prev->next = NULL;

}
```

```
void PrintNode(Position P){

    Position prev = P;

    printf("*****\n");

    printf("Name:%s\t", prev->StuName);

    printf("StuNum:%d\n", prev->StuNum);

    printf("English:%d\nCalculus:%d\nPhysics:%d\nCPL:
%d\n", prev->English, prev->Calculus, prev->Physics,
prev->CPL);

    printf("The Average:%.2lf\n", prev->Average);

    printf("*****\n");

}
```

```
void Change(Position P, int StuNum){

    Position prev = FindStu(StuNum, P);

    int option;

    printf("What do you want to change:\n");

    printf("1.English\t2.Calculus\n3.Physics\t4.CPL\n");

    scanf("%d", &option);

    switch(option){

        case 1:

            printf("please input a new mark:");
```

```

        scanf("%d", &prev->English);

        break;

    case 2:

        printf("please input a new mark:");

        scanf("%d", &prev->Calculus);

        break;

    case 3:

        printf("please input a new mark:");

        scanf("%d", &prev->Physics);

        break;

    case 4:

        printf("please input a new mark:");

        scanf("%d", &prev->CPL);

        break;

    }

}

void StuAverage(Position P, int StuSum){

    Position prev = P;

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

        prev->Average                                     =
prev->CPL+prev->Physics+prev->Calculus+prev->English;

        prev->Average = prev->Average / 4;

        prev = prev->next;
    }
}

```

```

    }

}

StuInfoList SelSort(StuInfoList L){

    StuInfoList p, q, small;

    double temp;

    int tempi;

    char *temps = malloc(sizeof(char)*20);

    for(p = L->next; p->next != NULL; p = p->next){

        small = p;

        for(q = p->next; q; q = q->next){

            if(q->Average > small->Average)

                small = q;

        }

        if(small != p){

            temp = p->Average;

            p->Average = small->Average;

            small->Average = temp;

            temp = p->StuNum;

            p->StuNum = small->StuNum;

```

```
small->StuNum = temp;
```

```
tempi = p->English;
```

```
p->English = small->English;
```

```
small->English = tempi;
```

```
tempi = p->Calculus;
```

```
p->Calculus = small->Calculus;
```

```
small->Calculus = tempi;
```

```
tempi = p->Physics;
```

```
p->Physics = small->Physics;
```

```
small->Physics = tempi;
```

```
temp = p->CPL;
```

```
p->CPL = small->CPL;
```

```
small->CPL = temp;
```

```
strcpy(temps, p->StuName);
```

```
strcpy(p->StuName, small->StuName);
```

```
strcpy(small->StuName, temps);
```

```
}
```

```
}
```

```

        return L;
    }

```

```

StuInfoList PtrSort(StuInfoList L){
    Position p, q;
    Position temp;
    for(p = L->next; p->next != NULL; p = p->next){
        for(q = p->next; q->next; q = q->next){
            if(q->Average > p->Average){
                temp = p->prev;
                p->prev = q->prev;
                q->prev = temp;
                temp = p->next;
                p->next = q->next;
                q->next = temp;
            }
        }
    }
    return L;
}

```

```

Position FindPre(Position head, Position target){
    Position p = head;

```

```
        while(p->next != NULL && p->next != target)

            p = p->next;

        return p;
    }

void Delete(StuInfoList L, Position target){

    Position p, temp;

    p = FindPre(L, target);

    if(!IsLast(p, L)){

        temp = p->next;

        p->next = temp->next;

    }

}
```

### 3) 测试

```

How many people in Class:3
Input the student's Num:1
Input his/her name:zhang
Input the grade of 4 subject:
English Calculus Physics CPL
90 80 90 90
Input the student's Num:2
Input his/her name:li
Input the grade of 4 subject:
English Calculus Physics CPL
100 80 80 40
Input the student's Num:3
Input his/her name:zhao
Input the grade of 4 subject:
English Calculus Physics CPL
80 90 90 100
Who's data need to be change? 0 to skip2
What do you want to change:
1.English      2.Calculus
3.Physics      4.CPL
4
please input a new mark:80
*****
Name:zhang      StuNum:1
English:90
Calculus:80
Physics:90
CPL:90
The Average:87.50
*****
*****
Name:li StuNum:2
English:100
Calculus:80
Physics:80
CPL:80
The Average:85.00
*****
*****
Name:zhao      StuNum:3
English:80
Calculus:90
Physics:90
CPL:100
The Average:90.00
*****

```

图 7-5 编程题 2 的测试运行结果

#### 4. 选做题

- (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。
- (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。
- (3) 采用双向链表重做编程设计题第 (2) 题。

解答: 已与程序设计第二题结合完成, 代码与思路见上题解答。

### 7.3 实验小结

第七次 C 语言上机实验, 主要学习并熟练了结构的说明和引用、结构的指针、结构数组、函数中使用结构的方法、动态储存分配函数的用法、自引用结构、单向链表的创建、遍历、结点的增删、查找以及字段结构和联合的用法。

#### (1) 验证题

此题首先考察了对于运算符优先级以及结合性的熟悉程度, 其次考察了对结构指针的用法的掌握。虽然在实际编程中不会用到类似的表达方式, 但是对于我们理解与掌握基本的语法是很有好处的。

#### (2) 程序修改替换实验

此题向我介绍了最基本的数据结构之一: 链表。在熟悉与掌握了如何构建

一个简单的单链表，同时利用尾插法和头插法达到不同的效果之后，我理解了设计结构这一聚合数据类型的意义。作为抽象数据结构的一部分，一个结构体既可以储存数据，亦可以包含与前后两个结点之间的关系，方便抽象数据结构的形成。同时，对于结构指针以及动态内存分配的用法与好处也有了更深层的理解与认识。

### **(3) 程序设计实验**

在第一题中熟悉了对于字段结构的操作之后，来到第二题。此题对于初学者来说，难度略大。首先要正确地理解结构体与结构指针，结构体与链表本身的区别。其次，要认识到在单链表中，所有操作都应从一个不变的头结点开始进行。并且要熟悉结构指针的使用。掌握了以上两点之后，我们便可开始设计各个函数，以完成不同的功能。在创建链表的函数中，一个低级的动态分配的错误耽误了很多时间，由于并没有为结构指针分配正确大小的地址，导致程序在运行时出现了很奇怪的错误，在请教老师后，发现了这一错误。同时老师在审阅我的代码的同时指出，我在无意识间创建了一个带头结点的单链表，可以为很多对于单链表的操作做很多的简化。在查阅资料，了解了头结点等概念后，我完善了对这个带头结点的链表的设计。这在选做题中，也发挥了很大的作用

### **(4) 选做题**

有了成功解决必做题的经验，我有了充足的信心。利用选择排序解决了通过交换数据域来对链表进行排序的题目之后，我立刻从繁琐的代码实现中了解到了寻找新的方法的重要性。因此，进行交换指针域排序的函数的编写。由于此前已经创建了带头结点的链表，对于头结点的处理方便许多，但是前后一共涉及四个结点的指针操作仍然显得繁琐而复杂。在最后学习了双向链表的基础知识之后，我终于找到了实现链表排序的略微简洁的方式。同时意识到，对于简洁的行事方式的追求，也许是人们进行各种研究的动力。



## 8 文件实验

### 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;
}

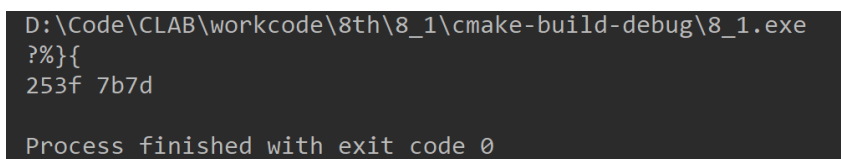
```

(1) 请思考程序的输出结果，然后通过上机运行来加以验证。

第一行将输出十六进制 25 3f 7b 7d ASCII 码值对应的字符:}%{

第二行将直接输出 253f 7b7d

运行结果:



```

D:\Code\CLAB\workcode\8th\8_1\cmake-build-debug\8_1.exe
}%{%
253f 7b7d

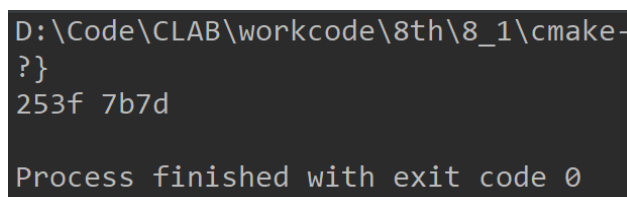
Process finished with exit code 0

```

图 8-1 验证题的测试的运行结果 (1)

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

运行结果:



```

D:\Code\CLAB\workcode\8th\8_1\cmake-build-debug\8_1.exe
}%{%
253f 7b7d

Process finished with exit code 0

```

图 8-2 验证题的测试的运行结果 (2)

第一行只打印出 2 个字符, 为之前的第一和第三个

因为两次写入 char 长度的内容对应 2 个字符

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

输出内容变成十进制。

```
D:\Code\CLAB\workcode\8th\8_1\cmake-1
?%}{
9535 31613|

Process finished with exit code 0
```

图 8-3 验证题的测试的运行结果 (3)

### 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

1  #include<stdio.h>
2  #include<stdlib.h>
3  int main(int argc, char* argv[])
4  {
5      char ch;
6      FILE *fp;
7      if(argc!=2){
8          printf("Arguments error!\n");
9          exit(-1);
10     }
11     if((fp=fopen(argv[1],"r"))==NULL){          /* fp 指向 filename
*/
12         printf("Can't open %s file!\n",argv[1]);
13         exit(-1);
14     }
15
16     while(ch=fgetc(fp)!=EOF)                    /* 从 filename 中读字符 */
17         putchar(ch);                             /* 向显示器中写字符 */
18     fclose(fp);                                  /* 关闭 filename */
19     return 0;
```

20 }

- (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。

错误：while(ch=fgetc(fp)!=EOF)

修改：while((ch=fgetc(fp))!=EOF)

修改后代码：

```
#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 错在
此处*/

        putchar(ch);

    /* 向显示器中写字符 */

    fclose(fp);
```

```
    return 0;
}
```

运行结果：

读入的文本文件为：

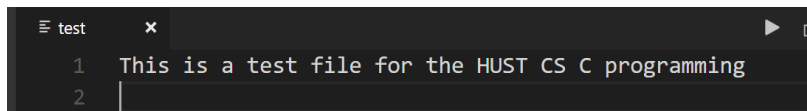


图 8-4 修改替换题读入文本截图

结果为：

```
PS D:\Code\CLAB\workcode\8th\8_2\cmake-build-debug> .\8_2.exe test
This is a test file for the HUST CS C programming
PS D:\Code\CLAB\workcode\8th\8_2\cmake-build-debug>
```

图 8-5 修改替换题（1）的测试运行结果

(2) 用输入输出重定向 `freopen` 改写上述源程序中的 `main` 函数。

程序代码：

```
#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
*/
        fprintf(stderr,"Can't open %s file!\n",argv[1]);
    }
}
```

```

        exit(-1);
    }

    while((ch=getchar())!=EOF)        /* 从 filename 中读字符 错在
    此处*/

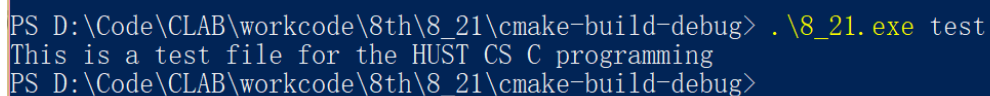
        putchar(ch);
    /* 向显示器中写字符 */

    fclose(stdout);

    return 0;
}

```

运行结果：



```

PS D:\Code\CLAB\workcode\8th\8_21\cmake-build-debug> .\8_21.exe test
This is a test file for the HUST CS C programming
PS D:\Code\CLAB\workcode\8th\8_21\cmake-build-debug>

```

图 8-6 修改替换题（2）的测试运行结果

### 8.2.3. 编程设计题

（1）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) 算法思路：

利用

2) 源程序清单

```

#include <stdio.h>

#include <stdlib.h>

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

    freopen("test", "w", stdout);

```

```

if(argc<2){
    fprintf(stderr, "arguments error!\n");
    exit(-1);
}

for(int i = 1; i < argc; i++)
    if(*argv[i] >= 'a' && *argv[i] <= 'z')
        (*argv[i]) -= 32;

for(int i = 1; i < argc; i++)
    printf("%s ", argv[i]);

fclose(stdout);

return 0;
}

```

### 3) 测试

```

PS D:\Code\CLAB\workcode\8th\8_3\cmake-build-debug> .\8_3.exe This is a statements
PS D:\Code\CLAB\workcode\8th\8_3\cmake-build-debug> cat test
This Is A Statements
PS D:\Code\CLAB\workcode\8th\8_3\cmake-build-debug>

```

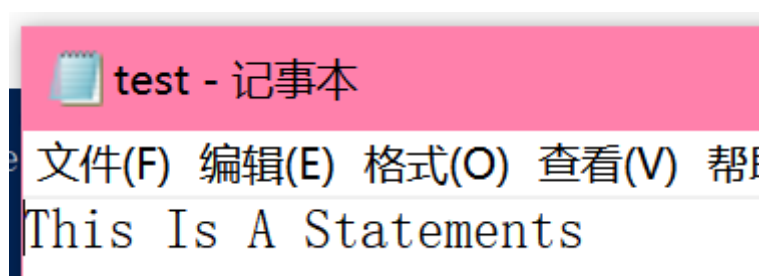


图 8-8 编程题 1 的测试运行结果

### 8.3 实验小结

此次实验让我学用了 C 语言中对文件进行操作的库函数。同时认识到了对文件进行处理的重要性与优劣之处。对文件进行操作是一个基本操作，属于必须学会的内容。经过这次实验，我了解到了基本的文件处理方法。

#### (1) 验证题

验证题展示了两种不同的文件类型在操作时的不同之处，经过解答验证题，我对 C 语言中两种文件类型的区别有了认识与了解。

#### (2) 程序修改替换实验

此题再次向我展示了基础之重要性。提醒我应对一些常用运算符的优先级以及结合性做到熟记。同时运用重定向函数的替换题，也使我对文件流这一概念有了更深刻的认识，对于 C 语言程序员来说，无论是键盘，屏幕，打印机，磁盘上的文件等输入输出设备或方式都被抽象成为文件，其抽象之后性质基本相同，因此可以利用重定向这一方式进行替换。这无疑带来了极大的便利，但是对其操作应该谨慎以防带来预料不到的错误。

#### (3) 程序设计实验

程序设计题中利用 main 函数参数读入字符串并处理单词首字母。同时继续熟悉文件的操作。