

**课 程 实 验 报 告**

**课程名称： C语言程序设计**

**专业班级： 计算机1804**

**学 号： U201814604**

**姓 名： 黄俊淇**

**指导教师： 胡龙**

**报告日期：**

**计算机科学与技术学院**

**目录**

**[1表达式和标准输入输出实验 1](#_Toc404837920)**

**[1.1实验目的 1](#_Toc404837921)**

**[1.2实验内容 1](#_Toc404837923)**

**[1.3小结 1](#_Toc404837923)**

**[2流程控制实验 2](#_Toc404837924)**

**[2.1必做题 2](#_Toc404837925)**

**[2.2选做题 2](#_Toc404837926)**

**[2.3小结 2](#_Toc404837927)**

**[3函数与程序结构实验 3](#_Toc404837929)**

**[3.1实验目的 3](#_Toc404837930)**

**[3.2实验内容 3](#_Toc404837931)**

**[3.3小结 3](#_Toc404837932)**

**[4编译预处理实验 4](#_Toc404837934)**

**[4.1实验目的 4](#_Toc404837935)**

**[4.2实验内容 4](#_Toc404837936)**

**[4.3小结 4](#_Toc404837937)**

**[5数组实验 5](#_Toc404837938)**

**[5.1实验目的 5](#_Toc404837939)**

**[5.2实验内容及要求 5](#_Toc404837940)**

**[5.3小结 5](#_Toc404837941)**

**[6指针实验 6](#_Toc404837943)**

**[6.1实验目的 6](#_Toc404837944)**

**[6.2实验内容及要求 6](#_Toc404837945)**

**[6.3小结 6](#_Toc404837946)**

**[7结构与联合实验 7](#_Toc404837948)**

**[7.1实验目的 7](#_Toc404837949)**

**[7.2实验内容及要求 7](#_Toc404837950)**

**[7.3小结 7](#_Toc404837952)**

**[8文件实验 8](#_Toc404837953)**

**[8.1实验目的 8](#_Toc404837954)**

**[8.2实验内容及要求 8](#_Toc404837955)**

**[8.3小结 8](#_Toc404837956)**

**[参考文献 9](#_Toc404837957)**

# 1表达式和标准输入输出实验

# 1.1实验目的

掌握c语言的表达式和标准输入和输出

# 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2.1节中源程序1和源程序2的实验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。

#include<stdio.h>

#define PI 3.14159/\*分号，预处理不需要分号\*/

int main( void )/\*int与main之间有空格\*/

{

int f,newint ;/\*变量定义\*/

short p, k ;

double c , r , s ;

/\* for task 1 \*/

printf("Input Fahrenheit:" ) ;

scanf("%d",& f ) ;/\*取址\*/

c = 5.0/9\*(f-32) ;/\*整型变浮点\*/

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

/\* for task 2 \*/

printf("input the radius r:");

scanf("%lf", &r);

s = PI \* r \* r;

printf("\nThe acreage is %.2lf\n\n",s);/\*lf精确度\*/

/\* for task 3 \*/

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

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

newint = (p&0xff00)|((k&0xff00)>>8);/\*左移改为右移\*/

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

return 0;getchar();getchar();

}

## 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。

**分析原程序**

使用中间变量t作为a的值暂时存放的单元，然后将b值赋给a值，再将在t单元下的a值赋给b值。t=a;a=b;b=t;为交换语句。

替换：

#include<stdio.h>

int main( )

{

int a, b;

printf("Input two integers:");

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

a=a-b;

b=a+b;

a=b-a;

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

return 0;getchar();getchar();

}

流程图：利用加减法，如图1.1

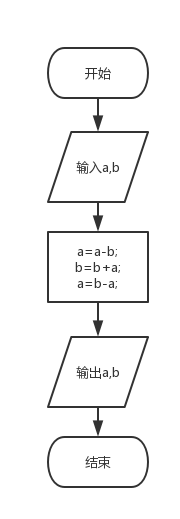


图1.1程序替换流程图

测试：输入a=12，b=21，测试结果如图1.2

F6549951CDD5ADC3B11A4FAFF65568F9

图1.2替换程序测试结果

## 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。

代码

#include<stdio.h>

int main()

{

char c;

scanf("%c",&c);

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

printf("%c",c+32);

else

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

return 0;getchar();getchar();

}

流程图：c如果是大写字母，用c的字符码加上32就变成小写如图1.3

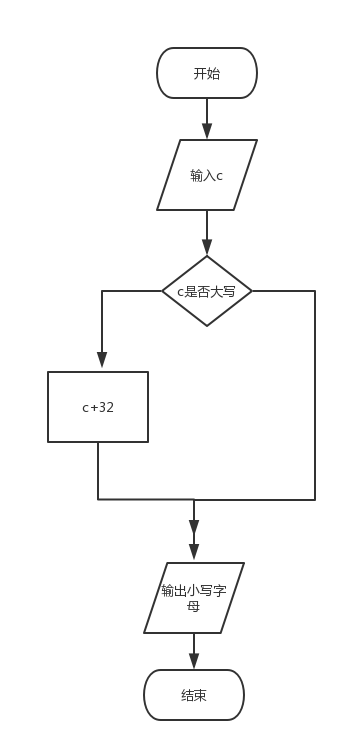


图1.3输出小写字母流程图

测试：输入大写字母A，结果图1.4

BFC24D6035340EB9D78E9B48D1BBEE9E

图1.4输出小写字母测试结果

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

代码

#include<stdio.h>

int main()

{

unsigned short x,m,n,mask;

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

mask=(~((~0)<<n))<<m;

printf("\n");

x=(x&mask)<<(16-m-n);

printf("%d",x);

return 0;getchar();getchar();

}

流程图：先取出n个0，取反得到n个1，向左移m位即做好逻辑尺，用逻辑尺与x进行位运算，再向左移（16-m-n）位即对齐如图1.5

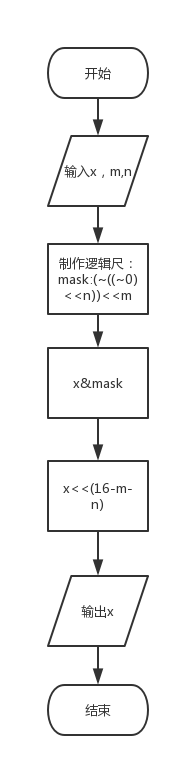


图1.5程序设计流程图

测试：输入3 0 15，结果如图1.6

D541BC42-D20D-475D-8BA2-865EC8642296

图1.6程序设计测试结果

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676879571二进制表示就是：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，所有最终格式为211.92.88.40。

代码

#include<stdio.h>

int main()

{

unsigned long x,a,b,c,d;

scanf("%ld",&x);

a=(x&0xff000000)>>24;

b=(x&0x00ff0000)>>16;

c=(x&0x0000ff00)>>8;

d=(x&0x000000ff);

printf("%d.%d.%d.%d",d,c,b,a);

return 0;getchar();getchar();

}

流程图：分别取出8位一组的高低字节如图1.7

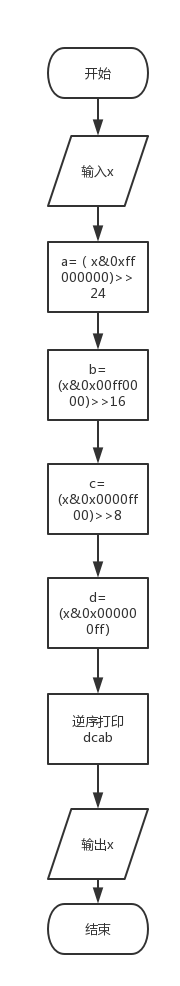


图1.7输出域名流程图

测试：输入676879571，测试结果如图1.8

77253697-C86D-4B73-BDC4-CBF2CDA6422F

图1.8输出域名测试结果

# 1.3小结

在这个实验中，我学到了对于short类型的数据应使用%hd而不是%d不然程序运行会出错。除此之外，还有注释对于c语言的必要性，很多时候难以记住庞大的程序中每一个细节，故要利用注释，不仅增强可读性，更方便程序员自己进行检查。

# 2流程控制实验

## **2.1必做题**

⑴程序改错：下面是计算s=n！的源程序，在这个源程序中存在若干个语法和逻辑错误。要求在计算机上对这个例子源程序进行调试修改，使之能够正确完成任务。例如，8！=40320.

#include <stdio.h>

**int** main( )

{

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

    printf("Please enter n:");

    scanf("%d",&n);*/\*取值\*/*

**for**(i=1;i<=n;i++)*/\*分号隔开\*/*

        s=s\*i;

    printf("%d!=%d",n,s);

**return** 0;

}*/\*花括号在外面\*/*

⑵程序修改替换：

1、修改上述计算s=n！的源程序，用while替换for语句

#include <stdio.h>

**int** main()

{

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

    i=1;

    printf("please enter n:\n");

    scanf("%d",&n);

**while**(i<=n)

    {

        s=s\*i;

        i++;

    }

    printf("%d!=%d",n,s);

**return** 0;

}

2、用do-while语句替换for语句

#include <stdio.h>

**int** main()

{

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

    printf("please enter n:\n");

    scanf("%d",&n);

    i=1;

**do**

    {

s=s\*i;

        i++;

    }

**while**(i<=n);

    printf("%d!=%d",n,s);

**return** 0;

}

流程图如图1.1

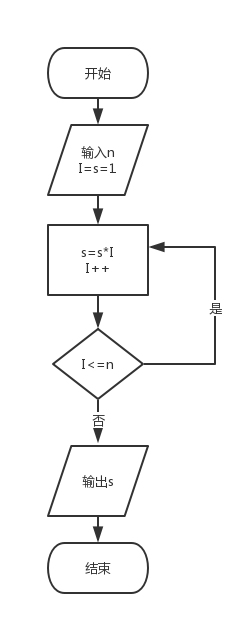


图1.1n！计算

测试结果如图1.2

5C298DB8C9084DE969CD877C6F6689B1

图1.2n！计算测试结果

1. 修改上述计算s=n！的源程序，要求输入改为“整数S”，输出改为“满足n！>=S的小整数n”。例如，输入整数40310，输出结果为n=8.

#include <stdio.h>

**int** main()

{

**int** S,i,s=1;

    scanf("%d",&S);

    i=1;

**while**(s<=S)

    {

        s=s\*i;

        i++;

    }

    printf("n=%d",i-1);

**return** 0;

}

流程图如图1.3

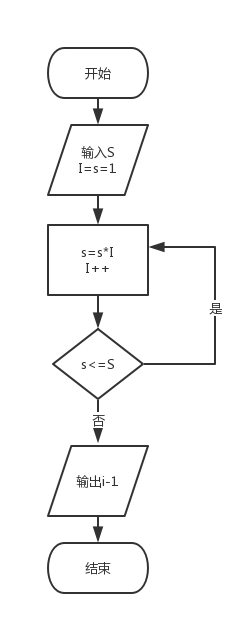


图1.3计算S的最小阶乘

测试结果如图1.4

3885619EFE1AFE3D696029C2F7156851

图1.4计算S的最小阶乘测试结果

⑶程序设计：

1、假设工资税金按以下方法计算：x ＜ 1000元，不收取税金；1000 ≤ x ＜ 2000，收取5%的税金；2000 ≤ x ＜ 3000，收取10%的税金；3000 ≤ x ＜ 4000，收取15%的税金；4000 ≤ x ＜ 5000，收取20%的税金；x＞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。

if语句：

#include <stdio.h>

**int** main()

{

**double** x;

    scanf("%lf",&x);

**if**(x<1000)

        printf("0");

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

        printf("%lf",x\*0.05);

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

        printf("%lf",0.1\*x);

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

        printf("%lf",0.15\*x);

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

        printf("%lf",0.2\*x);

**else** **if**(x>=5000)

        printf("%lf",0.25\*x);

**return** 0;

}

switch语句：

#include <stdio.h>

**int** main()

{

**int** y;

**double** x;

    scanf("%lf",&x);

    y=(**int**)(x/1000);

**switch**(y)

    {

**case** 0:

            printf("0");

**break**;

**case** 1:

            printf("%lf",0.05\*x);

**break**;

**case** 2:

            printf("%lf",0.1\*x);

**break**;

**case** 3:

            printf("%lf",0.15\*x);

**break**;

**case** 4:

            printf("%lf",0.2\*x);

**break**;

**default**:

            printf("%lf",0.25\*x);

    }

**return** 0;

}

流程图如图1.5

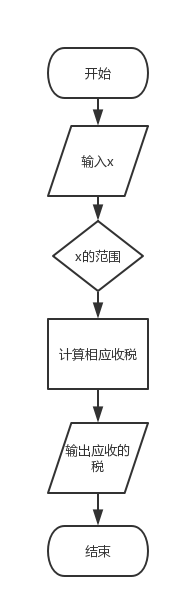


图1.5分段计算收税

测试结果如图1.6

3C92E306FED8932217DCFA3117DDF5A4

图1.6分段计算收税

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。

代码：

#include <stdio.h>

**int** main( )

{

**int** c;

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

    {

**if**(c==' ')

        {

            putchar(c);

**while**((c=getchar())==' ')

            {

**continue**;

            }

        }

        putchar(c);

    }

**return** 0;

}

流程图如图1.7

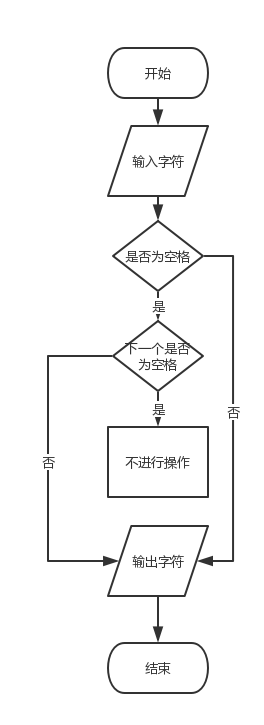


图1.7合并空格

测试结果如图1.8

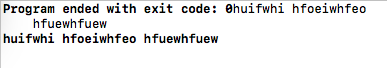


图1.8合并空格测试结果

3、编写一个程序，打印如下的杨辉三角形。

1 /\*第0行 \*/

1 1 /\*第1行 \*/

1 2 1 /\*第2行 \*/

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

每个数据值可以由组合计算（表示第i行第j列位置的值），而的计算如下：

 (i=0,1,2,…)

 (j=0,1,2,3,…,i)

说明：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。

代码：

#include<stdio.h>

**int** main()

{

**int** i,j,a[10][10];*/\*i是行，j是列\*/*

**int** n=11;

**for**(i=0;i<=9;i++)

**for**(j=0;j<=9;j++)

        {

            a[i][0]=1;

            a[i][i]=1;

        }

**for**(i=2;i<=9;i++)

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

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

**for**(i=0;i<=9;i++)

    {

        printf("%\*c",2\*n,' ');

**for**(j=0;j<=i;j++)

        printf("%5d",a[i][j]);

        printf("\n");

        n--;

    }

**return** 0;

    }

流程图如图1.9

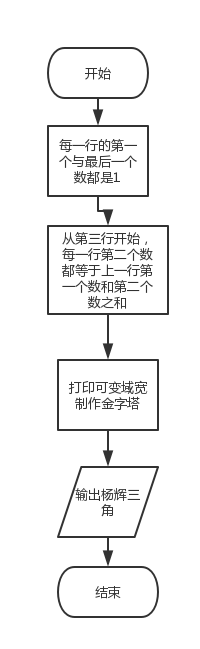


图1.9杨辉三角

测试结果如图2.0

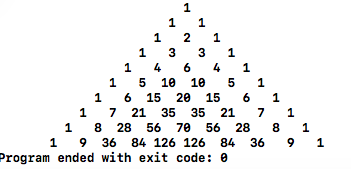


图2.0杨辉三角测试结果

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。

代码：

#include <stdio.h>

**int** main(**void**)

{

**int** x;

    scanf("%d", &x);

**do**

       printf("%d",x%10);

**while**((x/=10)!=0);

    printf("\n");

**return** 0;

}

流程图如图2.1

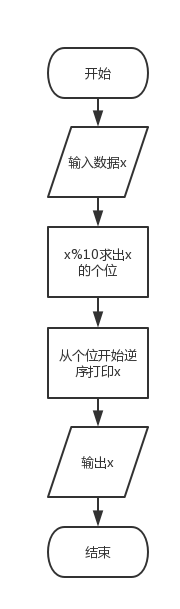


图2.1逆序输出

测试结果如图2.2

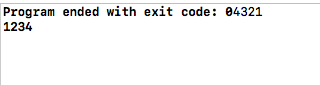


图2.2逆序输出测试结果

## 2.2选做题

程序设计：编写一个程序，用牛顿迭代法求方程满足精度e=10-6的一个近似根，并在屏幕上输出所求近似根。牛顿迭代法求方程近似根的迭代公式为：



其中,是函数f(x) 的导函数。牛顿迭代法首先任意设定的一个实数来作为近似根的迭代初值x0，然后用迭代公式计算下一个近似根x1。如此继续迭代计算x2, x3, …, xn, 直到，此时值xn即为所求的近似根。

代码：

#include <stdio.h>

#include<math.h>

#define EPS 1e-6

**int** main( )

{

**double** x,d;

    printf("input initial root:\n");

    scanf("%lf",&x);

**do**{

        d=-((((3\*x-4)\*x)-5)\*x+13)/((9\*x-8)\*x-5);

        x+=d;

    }**while**(fabs(d)>EPS);

    printf("x=%f\n",x);

**return** 0;

}

流程图如图2.3

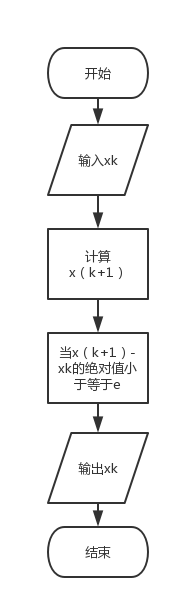


图2.3牛顿迭代方程

测试结果如图2.4

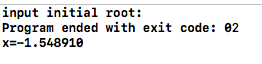


图2.4牛顿迭代方程测试结果

## 2.3小结

本次实验中，充分体会到了三种循环结构的不同之处，while可读性较强，for比较简洁而且灵活，do-while适合于至少循环一次的循环体。其次，在杨辉三角的实验中，对于二维数组的运用的熟练程度进一步加深，感受到了数组会比普通算法更为简单。除此之外，杨辉三角的金字塔利用了可变域宽，增加了对图像的修饰手段。最后就是牛顿迭代方程利用c语言进行编写后，可以利用计算机高效地算出高阶方程的近似根，体现了计算机强大的运算能力。

# 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．源程序改错题

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

#include <stdio.h>

**long** sum\_fac(**int** n);*/\*声明函数\*/*

**int** main(**void**)

{

**int** k;

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

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

**return** 0;

}*/\*花括号\*/*

**long** sum\_fac(**int** n)

{

**long** s=0;

**int** i;

**long** fac=1;*/\*赋值\*/*

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

    {            */\*花括号\*/*

        fac\*=i;

        s+=fac;

    }

**return** s;

}

### 3.2.2．源程序修改替换题

（1）修改第1题中sum\_fac函数，使其计算量最小。

#include <stdio.h>

**long** sum\_fac(**int** n);

**static** **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));

**return** 0;

}

**long** sum\_fac(**int** n)

{

**static** **long** s=0;

    fac\*=n;

     s+=fac;

**return** s;

}

流程图如图1.1

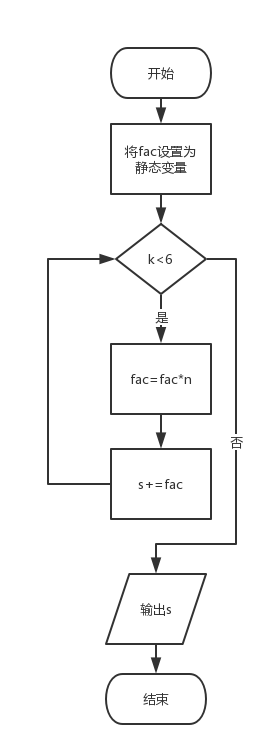


图1.1静态变量

测试结果如图1.2

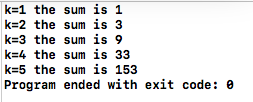


图1.2静态变量测试结果

1. 修改第1题中sum\_fac函数，计算。

#include <stdio.h>

**float** sum\_fac(**int** n);

**int** main(**void**)

{

**int** k;

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

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

**return** 0;

}

**float** sum\_fac(**int** n)

{

**float** s=0;

**int** i;

**float** fac=1;

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

    {

        fac\*=i;

        s+=(1/fac);

    }

**return** s;}

流程图如图1.3

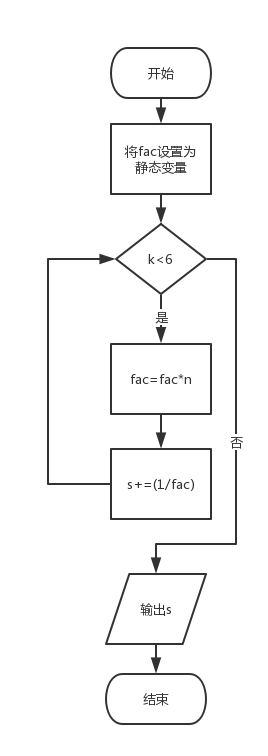


图1.3（1/n！）计算

测试结果如图1.4

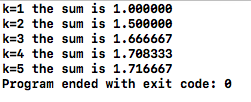
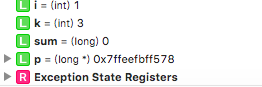


图1.4（1/n！）计算测试结果

### 3.2.3．跟踪调试题

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

1. 刚执行完scanf("%d",&k);语句，p,i值是多少？



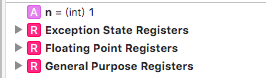
1. 从fabonacci函数返回后光条停留在哪个语句上？

sum+=fabonacci(i);

1. 进入fabonacci函数，watch窗口显示的是什么？



（4）当i=3，从调用fabonacci函数到返回，n值如何变化？







#include <stdio.h>

int main(void)

{

int i,k;

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

scanf("%d",&k);

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

sum+=fabonacci(i);

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

}

return 0;

}

long fabonacci(int n)

{

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

return 1;

else

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

}

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）。

### 3.2.4．程序设计题

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

#include <stdio.h>

**int** maxgongyueshu(**int** a,**int** b)

{

**int** r;

    r=a%b;

**if**(r==0)

**return** b;

**else**

        printf("%d\n",r);

**return** maxgongyueshu(b,r);

}

**int** main(**void**)

{

**int** a,b;

    scanf("%d%d",&a,&b);

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

 }

流程图如图1.5

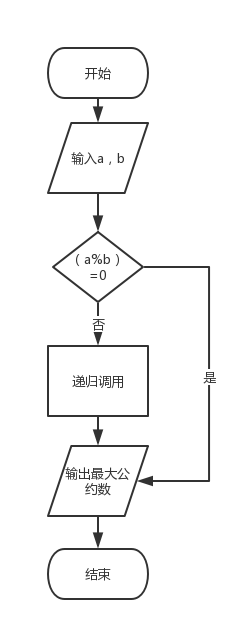


图1.5最大公约数

测试结果如图1.6

7CA24D9280B66B986CDD38A84272CE7D

图1.6最大公约数测试结果

1. 算术级数是后项比前项大的常数序列。假设第1项为a，相邻两项的差为d，则第k项为tk = a + (k-1)d。编写包含3个形参a、d和k的函数，根据给定的a和d，返回第k项的值。编写程序提示用户输入a和d，然后打印级数的前100项，5个一行。若产生整数上溢，则提前终止程序。

#include <stdio.h>

#define max 2147483647

#include<math.h>

**void** canshu(**int** a,**int** d,**int** k);

**int** k=1;

**int** main()

{

**int** a,d;

    printf("please input 2 numbers:\n");

    scanf("%d %d",&a,&d);

    canshu(a,d,k);

}

**void** canshu(**int** a,**int** d,**int** k)

{

**int** t=0;

**for**(;k<=100;k++)

    {

**if**((a+(k-1)\*d)>(a+k\*d))*/\*整数上溢判断条件\*/*

**break**;

        t=a+(k-1)\*d;

        printf("%d\t",t);

**if**(!(k%5))

            printf("\n");

    }

}

流程图如图1.7

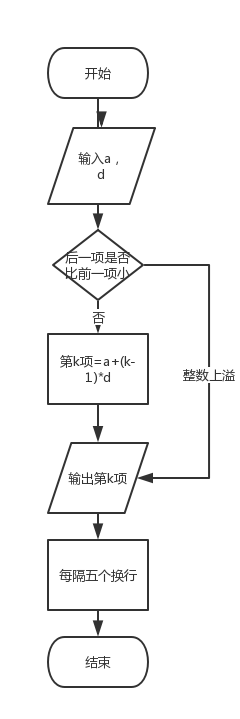


图1.7等差数列及整数上溢

若整数不上溢测试结果如图1.8

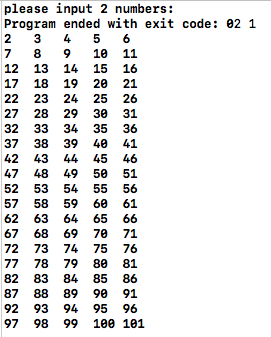


图1.8等差数列不上溢

若整数上溢测试结果如图1.9

7093E1EB386A429FCD74B98721DA8ECB

图1.9整数上溢测试结果

（3）写一个函数gotbaha验证哥德巴赫猜想：一个大于等于4的偶数可以表示为两个素数的和，例如4=2+2，6=3+3，8=3+5，...，在函数gotbaha中再调用函数isprime，isprime函数的作用是判别一个数是否为素数。在函数gotbaha中输出以下形式的结果（以函数调用gotbaha（34）为例）：

34=3+31

在main函数中调用函数gotbaha验证在符号常量BEGIN和END之间的偶数这一猜测成立。例如，如果BEGIN为10，END为20，程序的输出应为：

GOLDBACH'S CONJECTURE:

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

10=3+7

12=5+7

……

20=3+17

#include <stdio.h>

**void** gotbaha(**int** BEGIN );

**int** isprime(**int** i)

{

**int** a;

**if** (i==2) **return** 1;

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

**if**(i%a==0)

**break**;

**if**(a>=i)

**return** 1;

**else**

**return** 0;

}

**void** gotbaha(**int** BEGIN)

{

**int** y;

**for**(y=2;y<=BEGIN/2;y++)

    {

**if**((isprime(y)==1)&&(isprime(BEGIN-y)==1))

        {

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

**break**;

        }

    }

}

**int** main()

{

**int** END;

**int** BEGIN;

    scanf("%d",&BEGIN);

    scanf("%d",&END);

**while**(BEGIN!=0)

    {

**if**((BEGIN%2)!=0)

**for**(;BEGIN<=(END-1);BEGIN++,BEGIN++)

            gotbaha((BEGIN+1));

**else**

**for**(;BEGIN<=END;BEGIN++,BEGIN++)

                gotbaha(BEGIN);

         printf("\n");

         scanf("%d",&BEGIN);

        scanf("%d",&END);

    }

**return** 0;

}

流程图如图2.1

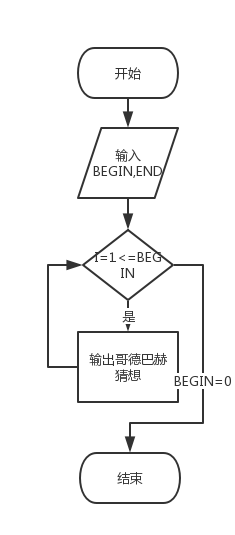


图2.1哥德巴赫猜想

测试结果如图2.2

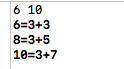


图2.2哥德巴赫猜想

### 3.2.5．选做题

假设一个C程序由file1.c和file2.c两个源文件及一个file.h头文件组成，file1.c、file2.c和file.h的内容分别如下所述。试编辑该多文件C程序，并编译和链接。然后运行生成的可执行文件。

源文件file1.c的内容为：

#include "file.h"

int x,y; /\* 外部变量的定义性说明 \*/

char ch; /\* 外部变量的定义性说明 \*/

int main(void)

{

x=10;

y=20;

ch=getchar( );

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

func1( );

return 0;

}

源文件file2.c的内容为：

#include "file.h"

void func1(void)

{

x++;

y++;

ch++;

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

}

头文件file.h的内容为：

#include <stdio.h>

extern int x,y; /\* 外部变量的引用性说明 \*/

extern char ch; /\* 外部变量的引用性说明 \*/

void func1(void); /\* func1函数原型 \*/

测试结果如图2.3

03B406A0419EF8610E9F8979330A7F58

图2.3多源文件

## 3.3小结

本次实验中，静态变量的适当使用可以极大的减少函数中的乘法运算，直接调用存在静态区的数值可以极大地减少程序计算量。在跟踪调试中，利用断点和条件断点清楚的了解了循环体中及递归调用中变量的变化过程，也掌握了调试程序的方法。在整数上溢中，如何判定整数上溢的条件对程序是极其必要的，如果出现整数上溢，程序会有极大的bug，因此在递增数列中需要考虑整数上溢。在多源文件中，可以将一个程序分成几个板块，对于运行和检查都有极大的便利。以及外部变量的使用会使文件同一个参数的使用不会发生错误。

# 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实验内容及要求

### 4.2.1．源程序改错

下面是用宏来计算平方差和交换两数的源程序。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源程序进行调试修改，使之能够正确完成计算平方差和交换两数的任务。

#include<stdio.h>

#define SUM  (a+b)

#define DIF  (a-b)

#define SWAP(a,b)  t=a,a=b,b=t*/\*z缺少中间变量\*/*

**int** main(**void**)

{

**int** a,b,t;*/\*变量a的声明\*/*

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

    scanf("%d,%d", &a,&b);*/\*逗号\*/*

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

    SWAP(a,b);

    printf("\nNow a=%d,b=%d\n",a,b);

**return** 0;

}

### 4.2.2．源程序修改替换

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

(1) 对这个例子程序进行调试修改，使之能够正确完成指定任务。

(2) 用带参数的宏替换函数max，来实现求最大数的功能。

#include<stdio.h>

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

**float** sum(**float** x,**float** y);*/\*声明函数\*/*

**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;

}

**float** sum(**float** x, **float** y)

{

**return** x+y;

}

max函数流程图如图1.1，sum函数流程图如图1.2

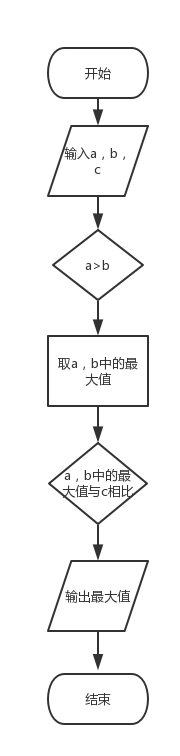


图1.1max函数流程图

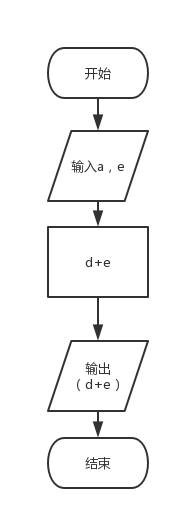


图1.2sum函数流程图

测试结果如图1.3

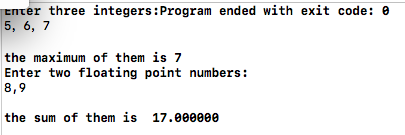


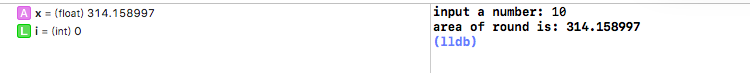
图1.3max，sum函数测试结果

### 4.2.3．跟踪调试程序

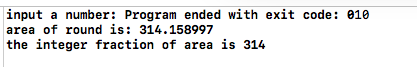
下面程序的功能是利用R计算圆的面积s，以及面积s的整数部分。现要求：

(1) 修改程序，使程序编译通过且能运行。

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



1. 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。



#include<stdio.h>

#include <assert.h>*/\*头文件\*/*

#define  R

**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);

   assert((s-s\_integer)<=1.0);*/\*大于号改小于号\*/*

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

#endif

**return** 0;

}

**int** integer\_fraction(**float** x)

{

**int** i=x;

**return** i;

}

### 4.2.4．程序设计题

(1) 已知三角形的面积是，其中，a,b,c为三角形的三边。定义两个带参数的宏，一个用来求s，另一个用来求area,试编写一程序，用带参数的宏来计算三角形的面积。

#include <stdio.h>

#include <math.h>

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

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

**int** main( )

{

**float** a,b,c,s,area;

    scanf("%f%f%f",&a,&b,&c);

    s=S(a,b,c);

    printf("s=%f\n",s);

    area=AREA(a,b,c,s);

    printf("area=%f",area);

}

流程图如图1.4

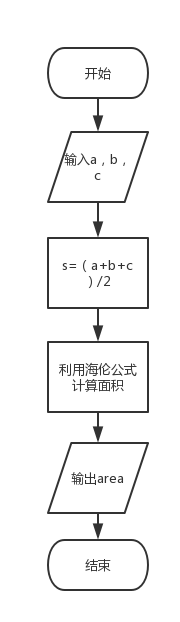


图1.4海伦公式流程图

海伦公式测试结果如图1.5

29CA4F16-AD30-4C35-BFC4-5C3676A942FF

图1.5海伦公式测试结果

1. 使用条件编译方法编写一程序,其功能要求是,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。

#include <stdio.h>

#define CHANGE 0

**int** main()

{

**char** s;

    scanf("%c",&s);

**while**(s!=EOF)

    {

#if CHANGE==1

**if**(s<='Z'&&s>='A')

    {  s+=32;

        putchar(s);}

**else**   **if**(s<='z'&&s>='a')

    {    s=s-32;

        putchar(s);}

#elif CHANGE==0

    putchar(s);

#endif

        scanf("%c",&s);

    }

}

流程图如图1.5

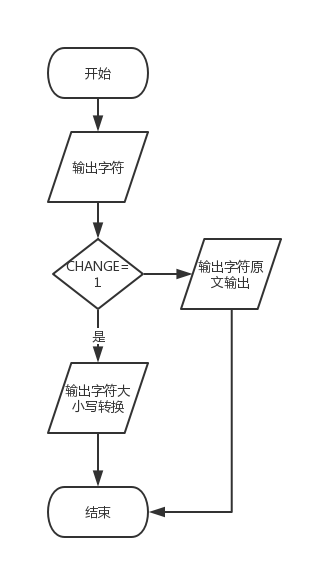


图1.6宏定义命令控制大小写流程图

测试结果如图1.7,1.8

14FC13873D2573A9F3BD5C9A06031C7F

图1.7CHANGE==1测试结果

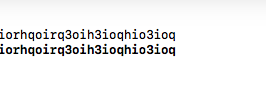


图1.8CHANGE==0测试结果

## 4.3小结

**本次实验中，大部分程序的输入函数scanf（）中的%d都以逗号隔开，因此在键盘输入数据的时候也应该用逗号隔开而不是用空格，否则会导致数据的错误。本章主要介绍了宏定义的使用，其大大方便了我们对程序进行测试，不需要一行一行地打printf，而只用在预处理时改变预处理的数据，且这种方法不会对多余程序进行编译。在跟踪测试中，如果数据上溢同样会造成断言错误而引起程序停止，因此在程序中应该对整数上溢进行限制。**

# 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实验内容及要求

### 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。

#include<stdio.h>

void sort(int [],int);

int main(void)

{

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

int i;

sort(a,10);

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

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

printf("\n");

return 0;

}

void sort(int b[], int n)

{

int i,j, t;

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

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

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

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

}

1. 函数在开头应声明
2. sort(a[0],10);

错误原因：应是从第0号元素开始

改正：sort（a，10）；

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

错误原因：升序排列

改正：if(b[j]>b[j+1])

### 5.2.2. 源程序完善、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，直到圈中只剩下一个人为止。请在源程序中的下划线处填写合适的代码来完善该程序。

#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接着报，形成一个圈 \*/

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

if(j)

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

;

}

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

printf(“%6d”, b[i]);

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

return 0;

}

完善代码：

#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)/\*i-1就是最后一个人\*/

j = 0; /\* 最后一个人报数后第一个接着报，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;

}

流程图如图1.1

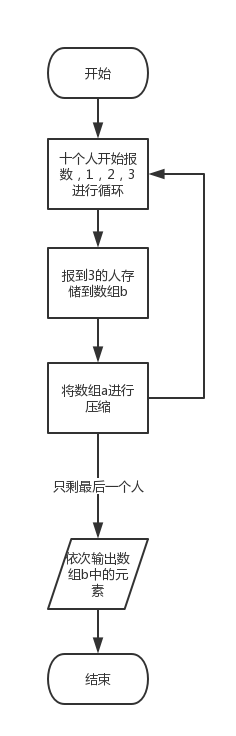


图1.1程序完善流程图

测试结果如图1.2



图1.2程序完善测试结果

(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。

代码：

#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-1; i++) /\* 对圈中人按顺序编号1—M \*/

a[i] = i + 1;

a[9]=10;

for(i = M, j = 0; i > 0; 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(a[++j-1]==0) k--;

if(j>M-1) j=0;

}/\* 最后一个人报数后第一个接着报，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)

a[i-1]=0;

else

a[j-1]=0;

}

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

/\* 按次序输出出圈人的编号 \*/

{ printf("%3d", b[i]);

}

return 0;

}

流程图如图1.3

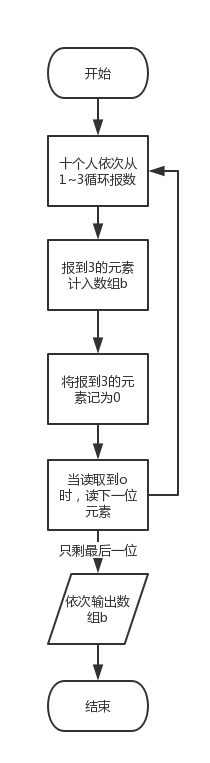


图1.3标记法流程图

测试结果如图1.4



图1.4标记法测试结果

### 5.2.3．跟踪调试源程序

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？

i的值如图1.5

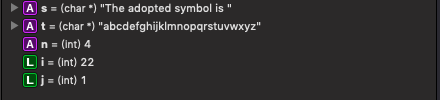


图1.5i的值

s，t如图1.6

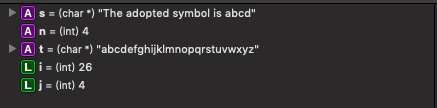


图1.6s，t

(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。

错误原因：当读到串尾时又自增了一次，因此串尾依然在，无法插入

改正： while(s[i]!='\0')

{

i++;

}

输出结果如图1.7

52F98AFA60F19BE09B417F0161C45641

图1.7输出结果

**源程序：**

#include<stdio.h>

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

int main(void)

{

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

strncat(a, b, 4);

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

return 0;

}

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

{

int i = 0, j;

while(s[i++]) ;

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

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

s[i] = '\0';

}

### 5.2.4．程序设计

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

代码：

#include <stdio.h>

int main()

{

int a[4][4],b[4][4];

int i,j;

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

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

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

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

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

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

printf("原矩阵:\n");

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

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

printf("%4d",a[i][j]);

printf("\n");

}

printf("转置矩阵：\n");

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

{

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

printf("%4d",b[j][i]);

printf("\n");

}

}

流程图如图1.8

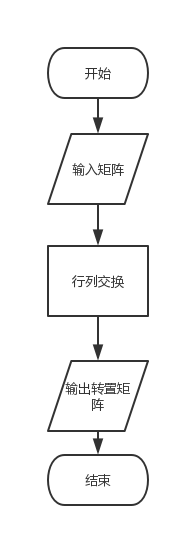


图1.8矩阵流程图

测试结果如图1.9

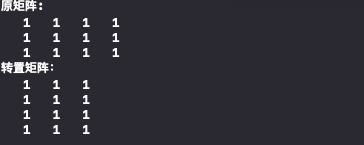


图1.9矩阵测试结果

1. 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。

代码：

#include <stdio.h>

int main()

{

int s[33];

int mask,i;

int n,x;

scanf("%d",&x);

n=sizeof(int)\*8;

mask=1<<(n-1);

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

{

s[i-1]=(!(x&mask)?'0':'1');

x<<=1;

}

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

printf("%c",s[i-1]);

}

流程图如图2.0

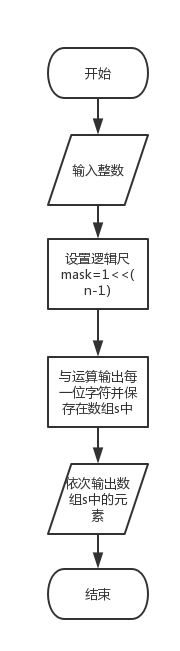


图2.0输出整数二进制表示流程图

测试结果如图2.1

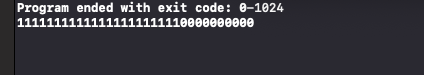


图2.1输出整数二进制测试结果

1. 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!”。

代码：

#include <stdio.h>

#include <string.h>

int search(float a[], int x, int );

void bubble\_sort(float score[], int index[], int m);

int main()

{

int n,x,s;

printf("input:\n");

scanf("%d",&n);

float score[n];

char name[n][40];

int i;

int together[n];

printf("u want input %d students' names and scores:\n", n);

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

{

scanf("%s %f", name[i], &score[i]);

}

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

{

together[i] = i;

}

bubble\_sort(score, together, n);

printf("name\tscore\n");

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

{

printf("%s\t%.2f\n", name[together[i]], score[together[i]]);

}

printf("input the score u want：\n");

scanf("%d", &x);

s = search(score, x, n);

if(s != 0)

printf("%s\t\t%.2f\n", name[s], score[s]);

else

printf("not found!");

return 0;

}

void bubble\_sort(float score[], int together[], int m)

{

int i, j, t;

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

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

{

if (score[together[j]] < score[together[j+1]])

{

t = together[j];

together[j] = together[j+1];

together[j+1] = t;

}

}

}

int search(float a[], int x, int p)

{

int head= 0, back = p -1, mid;

while (head<= back)

{

mid=(head+ back) / 2;

if(x < a[mid])

back = mid-1;

else if(x > a[mid])

head= mid+1;

else

return mid;

}

return 0;

}

流程图如图2.2

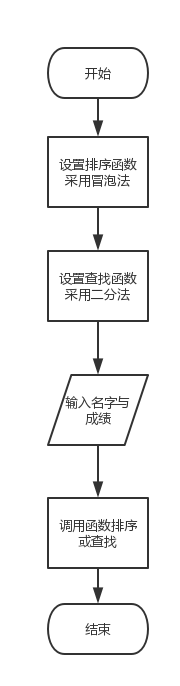


图2.2名字成绩排序及查找流程图

测试结果如图2.3

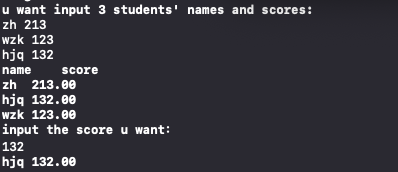


图2.3名字成绩排序及查找测试结果

### 5.2.5．选做题

(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，编写main测试strnins的正确性。

代码：

#include <stdio.h>

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

int main()

{

char s[100],t[100];

int n;

scanf("%s",s);

scanf("%s",t);

scanf("%d",&n);

mystrnins(s,t,n);

printf("%s",s);

}

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

{

int i=0,j=0,a;

while(s[i++]);

while(t[j++]);

for(a=i;a>=n;a--)/\*从最后一位开始移安全，从第一位开始移有可能覆盖\*/

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

i=0;

while(i<j-1)

{

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

i++;

}

}

流程图如图2.4

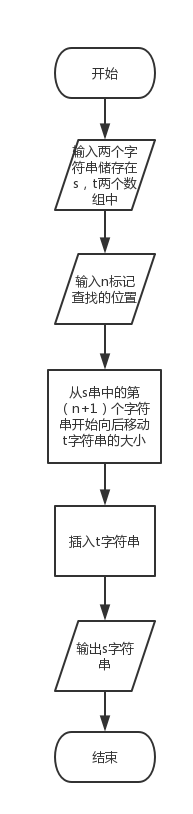


图2.4插入字符串流程图

测试结果如图2.5

5A3D7259E1092FFB9CAACFFC6128BD26

图2.5插入字符串测试结果

1. 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。

代码：

#include<stdio.h>

#define line 8

void queen(int i,int j);

int check(int i,int j);

int chess[line][line];

int answer=0;

int main()

{

queen(0,0);

printf("there are %d answer\n",answer);

return 0;

}

void queen(int i,int j)

{

if(j>=line)

{

return ;

}

if(check(i,j)==1)

{

chess[i][j]=1;

if(i==line-1)

{

answer++;

}

else

{

queen(i+1,0);

}

}

chess[i][j]=0;

queen(i,j+1);

}

int check(int i,int j)

{

int k;

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

{

if(chess[i][k]==1) return 0;

}

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

{

if(chess[k][j]==1) return 0;

}

for(k=-line;k<=line;k++)

{//两对角线

if(i+k>=0&&i+k<line&&j+k>=0&&j+k<line)//从左上到右下对角线

if(chess[i+k][j+k]==1) return 0;

if(i-k>=0&&i-k<line&&j+k>=0&&j+k<line)//从左下到右上对角线

if(chess[i-k][j+k]==1) return 0;

}

return 1;

}

流程图如图2.6

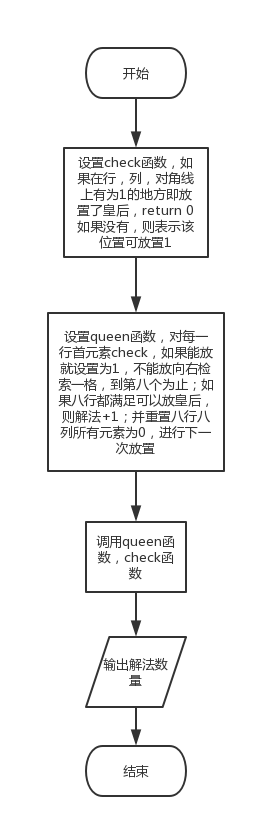


图2.6八皇后流程图

测试结果如图2.7

B39520A6EEB5969937167B91F45C89AA

图2.7八皇后测试结果

## 5.3小结

本次实验中大量运用了数组的输入和输出。在我所掌握的知识中，数组的用处有：存储某一长度的字符串；对于函数无法改变形参的量时，可调用数组，利用地址来实现。数组相比于之前所学更加灵活，且能更为简便的处理某一问题。同样的，在本次实验还有冒泡排序法以及二分法的应用，可以发现二分法有其弊端，当出现两个相同数据时只能输出一组，无法多组输出，对此我仍未想到好的办法来解决这个问题。而对于插入字符串的实验，关键问题是为什么要从最后一个元素开始向后移动，理由是如果从第一个开始移动，假如t字符串长度过小，则会导致数据的覆盖，造成实验错误。在八皇后问题中，对于每一行的检索至关重要，而最重要的就是设置好边界，并对于每一种解法之后要对棋盘进行初始化，否则会导致后一种解法出现严重偏差。

# 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 实验内容及要求

### 6.2.1．源程序改错

下面的源程序中是否存在错误？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源程序进行调试修改，使之能够正确执行。

#include<stdio.h>

int main(void)

{

float \*p;

scanf("%f",p);

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

return 0;

}

错误原因：指针没有初始化

改正：float n,\*p=&n;

### 6.2.2．源程序完善、修改、替换

(1) 下面源程序的功能是,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。请在程序中的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 a[80],b[80],c[160],\*result=c;

int choice,i;

do{

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

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

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

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

scanf("%d",&choice);

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

switch(choice){

case 1:

p=strcpy;

break;

case 2:

p=strcat;

break;

case 3:

goto down;

}

getchar();

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

i=0;

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

i=0;

result= (a,b);

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

down:

return 0;

}

(2) 请上机运行第（1）题程序，使之能按要求输出下面结果（注：（输入）表示该数据是键盘输入数据）：

1 copy string.

2 connect string.

3 exit.

input a number (1-3) please!

2 （输入）

input the first string please!

the more you learn, （输入）

input the second string please!

the more you get. （输入）

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

完善后代码如下：

#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;

while((a[i]=getchar())!='\n')

{

i++;

}

a[i]='\0';

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

i=0;

while((b[i]=getchar())!='\n')

{

i++;

}

b[i]='\0';

result=p(a,b);

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

down:

return 0;

}

运行结果如图1.1

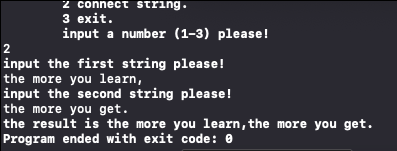


图1.1程序完善运行结果

### 6.2.3．跟踪调试源程序

请按下面的要求对所给源程序进行操作，并回答问题和排除错误。

1. 单步执行源程序。进入strcpy时,watch窗口中s为何值？返回main时, watch窗口中s为何值？

进入函数s的值如图1.2

返回main是s的值如图1.3

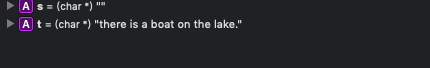


图1.2进入函数s的值

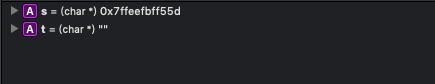


图1.3返回main时s的值

(2) 排除源程序中的错误，使程序输出结果为： there is a boat on the lake.

#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)

{

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

;

return (s);

}

正确代码：

#include<stdio.h>

char \*mystrcpy(char \*,char \*);

int main(void)

{

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

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

return 0;

}

char \*mystrcpy(char \*s,char \*t)/\*返回指针\*/

{

char\*p;

p=s;

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

;

return p;

}

运行结果如图1.4

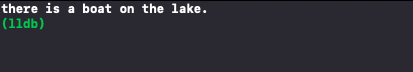


图1.4跟踪测试运行结果

### 6.2.4．程序设计

(1) 已知一个长整型变量占4个字节，其中每个字节又分成高4位和低4位。试编写一个程序,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位并以十六进制数字字符的形式进行显示。

代码如下：

#include <stdio.h>

int main()

{

long n;

scanf("%ld",&n);

char \*p=(char\*)&n;

char a[4][2];

int i,j;

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

{

a[i][1]=(\*p>>4)&0x0f;

a[i][0]=(\*p)&0x0f;

p++;

}

for(i=3;i>=0;i--)

{

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

{

if((a[i][j]>=0)&&(a[i][j]<=9))

a[i][j]+='0';

if((a[i][j]>=10)&&(a[i][j]<=15))

a[i][j]=a[i][j]-10+'a'-'0';

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

}

printf("\n");

}

return 0;

}

流程图如图1.5

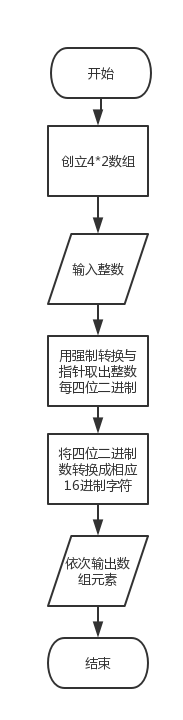


图1.5取出每四位对应16进制字符

测试结果如图1.6

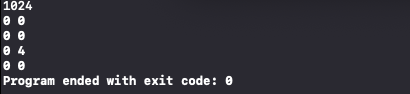


图1.6取出每四位对应的16进制字符测试结果

(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。试编写一个函数，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。

代码如下：

#include <stdio.h>

void fun(char \*p[],char a[][80],int n);

int main()

{

int n;

scanf("%d",&n);

getchar();

char \*p[n];

char a[n][80];

int i;

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

p[i]=a[i];

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

gets(a[i]);

fun(p,a,n);

return 0;

}

void fun(char \*p[],char a[][80],int n)

{

int i,j,k;

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

{

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

{

if(a[i][j]==' '||a[i][j]=='\t')

while(a[i][j+1]==' '||a[i][j+1]=='\t')

{ for(k=j+1;a[i][k]!='\0';k++)

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

}

}

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

{

if(a[i][0]=='\0'||a[i][0]==' '||a[i][0]=='\t')

continue;

else

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

}

}

流程图如图1.7

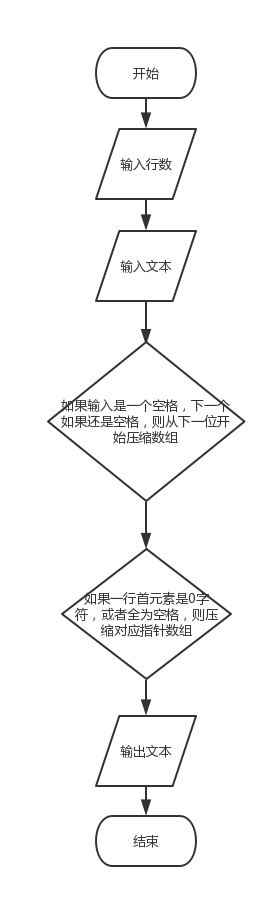


图1.7删除多余空格且不输出空行测试结果

测试结果如图1.8

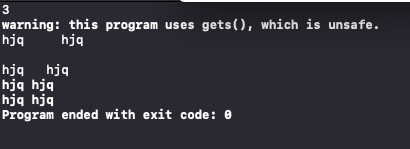


图1.8删除多余空格且不输出空行测试结果

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

代码如下：

#include <stdio.h>

#include<string.h>

void up\_bubble\_sort(int a[],int n);

void down\_bubble\_sort(int a[],int n);

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

{

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

int a[100];

int n,i;

scanf("%d",&n);

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

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

if((strcmp(argv[1],"-d")==0))

p=down\_bubble\_sort;

else

p=up\_bubble\_sort;

p(a,n);

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

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

return 0;

}

void up\_bubble\_sort(int a[],int n)

{

int i,j,t;

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

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

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

{t=a[j];a[j]=a[j+1];a[j+1]=t;}

}

void down\_bubble\_sort(int a[],int n)

{

int i,j,t;

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

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

if(a[j]<a[j+1])

{t=a[j];a[j]=a[j+1];a[j+1]=t;}

}

流程图如图1.9

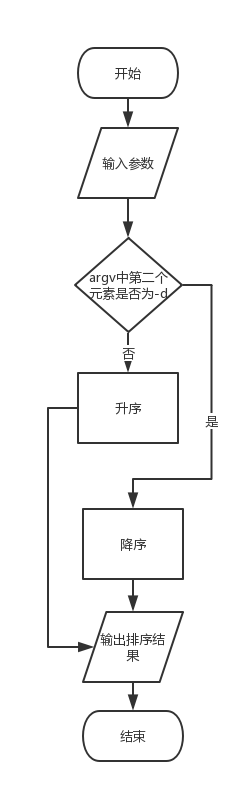


图1.9命令行实现排序

测试结果如图2.0

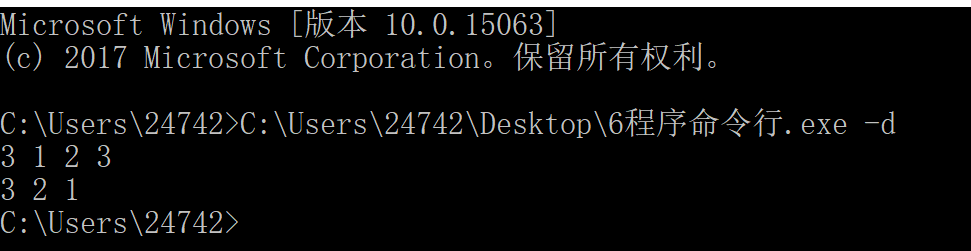


图2.0命令行实现排序测试结果

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

(a) 计算每个学生各门课程平均成绩；

(b) 计算全班每门课程的平均成绩；

(c) 分别统计低于全班各门课程平均成绩的人数；

(d) 分别统计全班各门课程不及格的人数和90分以上（含90分）的人数。

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

代码如下：

#include <stdio.h>

#define N 3

#define M 2

int average1(int (\*a)[M]);

void average2(int (\*a)[M]);

void low1(int (\*a)[M]);

void low2(int (\*a)[M]);

int main()

{

int a[N][M];

int i,j;

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

{

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

{

scanf("%d",&\*(\*(a+i)+j));

}

}

average1(a);

average2(a);

low1(a);

low2(a);

}

int average1(int (\*a)[M])

{

int i,j,average=0,sum=0;

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

{

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

{

sum+=\*(\*(a+i)+j);

}

average=sum/M;

sum=0;

printf("average of %d student is %d\n",i,average);

}

return average;

}

int sum1=0;

void average2(int (\*a)[M])

{

int i,j,average=0,sum=0;

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

{

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

{

sum+=\*(\*(a+i)+j);

}

average=sum/N;

sum=0;

printf("average of %d course is %d\n",j,average);

if(\*(\*(a+i)+j)<average)

sum1++;

}

}

void low1(int (\*a)[M])

{

int j;

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

{

printf("there are %d student lower than the average of %d course\n",sum1,j);

}

}

void low2(int (\*a)[M])

{

int i,j,low=0,high=0;

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

{

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

{

if(\*(\*(a+i)+j)<60)

low++;

if(\*(\*(a+i)+j)>90)

high++;

}

printf("there are %d student failed in %d course\n",low,j);

printf("there are %d student successed in %d course\n",high,j);

low=0;

high=0;

}

}

流程图如图2.1

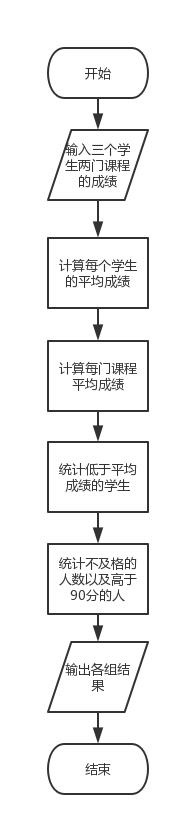


图2.1指针操作流程图

测试结果如图2.2

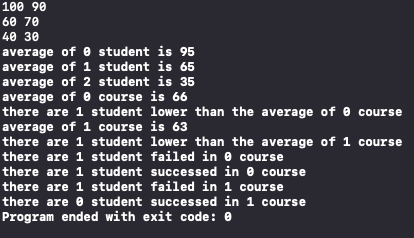


图2.2 调用函数指针操作

### 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。

代码如下：

#include <stdio.h>

#include <ctype.h>

#define N 20

#define M 10

int main()

{

int x[M+N+1],y[M+N+1],z[M+N+2];

int i,carry=0,flag=0;

for(i=0;i<=19;i++)

{

x[i]=getchar();

x[i]-='0';

}

x[20]=getchar();

for(i=21;i<=30;i++)

{

x[i]=getchar();

x[i]-='0';

}

getchar();

for(i=0;i<=19;i++)

{

y[i]=getchar();

y[i]-='0';

}

y[20]=getchar();

for(i=21;i<=30;i++)

{

y[i]=getchar();

y[i]-='0';

}

for(i=31;i>=22;i--)

{

z[i]=x[i-1]+y[i-1]+carry;

if(z[i]>=10)

carry=1;

else

carry=0;

z[i]%=10;

z[i]+='0';

}

for(i=20;i>=1;i--)

{

z[i]=x[i-1]+y[i-1]+carry;

if(z[i]>=10)

carry=1;

else

carry=0;

z[i]%=10;

z[i]+='0';

}

z[0]=carry+'0';

z[21]='.';

for(i=0;i<=31;i++)

{

if(flag==0&&z[i]!='0') flag=1;

if(flag==1) putchar(z[i]);

}

putchar('\n');

return 0;

}

流程图如图2.3

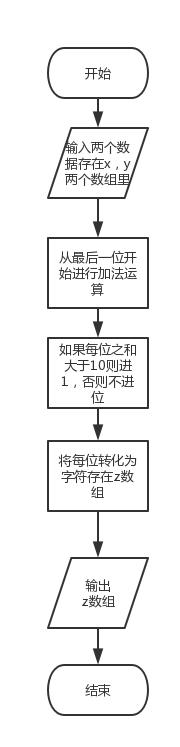


图2.3超长整数加法运算

测试结果如图2.4

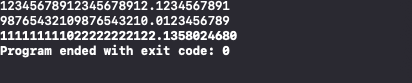


图2.4超长整数运算

(2) 编写一个使用复杂声明“char \*(\*p[2])(const char \*,const char \*);”的程序。

提示：p中元素可为strstr等函数名。

代码如下：

#include <stdio.h>

#include <string.h>

char \*fun1(const char\*,const char\*);

char \*fun2(const char\*,const char\*);

int main()

{

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

char a[2],b[2];

int i;

for(i=0;i<2;i++)

{

a[i]=getchar();

}

getchar();

for(i=0;i<2;i++)

{

b[i]=getchar();

}

p[0]=fun1;

p[1]=fun2;

int n;

printf("input 0 or 1\n");

scanf("%d",&n);

printf("%c",\*p[n](a,b));

return 0;

}

char \*fun1(const char\*a,const char\*b)

{

char\*k;

k=a;

return k;

}

char \*fun2(const char\*a,const char\*b)

{

char\*k;

k=b;

return k;

}

流程图如图2.5

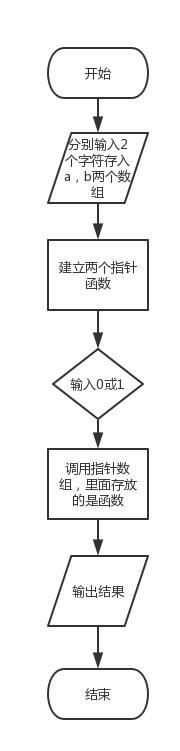


图2.5复杂声明应用

测试结果如图2.6

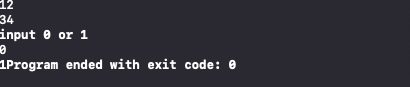


图2.6复杂声明应用

## 6.3小结

在本次实验中，可以看到悬挂指针每次运行的值都不一样，可以想象它会对原有数据进行巨大的破坏。在完善程序中，学到了指针函数返回的应该是一个指针类型的值。此外，还学到了利用指针进行位运算以及指针函数，指针数组的综合应用。最后就是利用字符数组进行超长整数运算，其实本质上跟平常对超长整数做加法运算一样，对每一位进行运算，然后判断是否需要进位，加上进位数，最后对最高位进行判断运算，输出即可完成超长整数运算。

# 7结构与联合实验

7.1 实验目的

（1）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）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 实验内容及要求

### 7.2.1．表达式求值的程序验证

设有说明：

char u[]="UVWXYZ";

char v[]="xyz";

struct T{

int x;

char c;

char \*t;

}a[]={{11,ˊAˊ,u},{100, ˊBˊ,v}},\*p=a;

请先自己计算表7.1中表达式的值，然后编写程序并运行来加以验证。(各表达式相互无关)

代码如下：

#include <stdio.h>

int main()

{

char u[]="UVWXYZ";

char v[]="xyz";

struct T{

int x;

char c;

char \*t;

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

printf("(++p)->x=%d\n",(++p)->x);

p=a;

p++;

printf("p->c=%c\n",p->c);

p=a;

printf("\*p++->t=%c ",\*p++->t);

printf("\*p->t=%c\n",\*p->t);

p=a;

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

p=a;

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

p->t=u;

p=a;

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

return 0;

}

运行结果如图1.1



图1.1表达式值运行结果

7.1 表达式值的计算

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

### 7.2.2．源程序修改替换

下面所给源程序的功能是：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。请完成以下工作：

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。

错误：create\_list函数中形参为二级指针

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

1. 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。

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 (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(&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");

return 0;

}

void create\_list(struct s\_list \*\*headp,int \*p) /\*修改成两级指针\*/

{

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

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 \*/

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

new->next=loc\_head;

loc\_head=new;

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

}

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 \*/

}

运行结果如图1.2

C0C7C84BF6B2D5CD366475C6C320F2CA

图1.2后进先出运行结果

**源程序：**

#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);

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(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");

return 0;

}

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 \*/

}

### 7.2.3．程序设计

(1) 编写一个程序,实现以下功能：定义一个字段结构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，各字段的优先级依次降低，即bit0的优先级最高，bit7的优先级最低。同时设计8个函数，将这8个函数的名字存入一个函数指针数组p\_fun。如果biti(i=0,1,...,7)为1，则调用p\_fun[i]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。例如，8个函数中的第0个函数可简单设计为：

void f0( )

{

printf(“the function 0 is called!\n”);

}

分析：声明字段结构，因为优先级的要求，设置联合结构，最后调用函数

代码如下：

#include <stdio.h>

struct bits{

unsigned char bit0:1,bit1:1,bit2:1,bit3:1,bit4:1,bit5:1,bit6:1,bit7:1,bit8:1;

}a;

void f0()

{

printf("the function 0 is called!\n");

}

void f1()

{

printf("the function 1 is called!\n");

}

void f2()

{

printf("the function 2 is called!\n");

}

void f3()

{

printf("the function 3 is called!\n");

}

void f4()

{

printf("the function 4 is called!\n");

}

void f5()

{

printf("the function 5 is called!\n");

}

void f6()

{

printf("the function 6 is called!\n");

}

void f7()

{

printf("the function 7 is called!\n");

}

union bit{

unsigned char b;

struct bits a;

}c;

int main()

{

void (\*p\_fun[8])(void);

scanf("%c",&c.b);

p\_fun[0]=f0;

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

if(c.a.bit0)

p\_fun[0]();

if(c.a.bit1)

p\_fun[1]();

if(c.a.bit2)

p\_fun[2]();

if(c.a.bit3)

p\_fun[3]();

if(c.a.bit4)

p\_fun[4]();

if(c.a.bit5)

p\_fun[5]();

if(c.a.bit6)

p\_fun[6]();

if(c.a.bit7)

p\_fun[7]();

return 0;

}

运行结果如图1.3

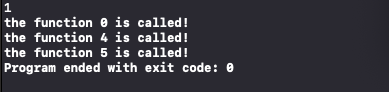


图1.3字段结构运行结果

(2) 假设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,试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。

代码如下：

#include <stdio.h>

#include <stdlib.h>

struct score

{

char name[20];

int number;

float English;

float Physic;

float Math;

float Clanguage;

struct score \*next;

};

void output(struct score\*headp);

void change(struct score \*head);

void output2(struct score\*head);

int main()

{

struct score \* head=NULL,\*tail;

int i;

int n;

printf("please input the number of students you want\n");

scanf("%d",&n);

head=(struct score\*)malloc(sizeof(struct score));

scanf("%s",head->name);

scanf("%d",&head->number);

scanf("%f",&head->English);

scanf("%f",&head->Physic);

scanf("%f",&head->Math);

scanf("%f",&head->Clanguage);

tail=head;

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

{

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

tail=tail->next;

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

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

scanf("%f",&tail->English);

scanf("%f",&tail->Physic);

scanf("%f",&tail->Math);

scanf("%f",&tail->Clanguage);

}

tail->next=NULL;

change(head);

output(head);

output2(head);

}

void output(struct score\*head)

{

struct score \*p=head;

while(p)

{

printf("%s\n",p->name);

printf("%d\n",p->number);

printf("%f\n",p->English);

printf("%f\n",p->Physic);

printf("%f\n",p->Math);

printf("%f\n",p->Clanguage);

p=p->next;

}

}

void change(struct score \*head)

{

int n,i=1;

struct score \*p=head;

while(i<=3)

{

printf("please input a number to choose the data you want to change\n");

printf("1:name\n");

printf("2:number\n");

printf("3:English score\n");

printf("4:Physic score\n");

printf("5:Math score\n");

printf("6:Clanguage score\n");

printf("7:no change!\n");

scanf("%d",&n);

switch(n)

{

case 1:scanf("%s",p->name);

break;

case 2:scanf("%d",&p->number);

break;

case 3:scanf("%f",&p->English);

break;

case 4:scanf("%f",&p->Physic);

break;

case 5:scanf("%f",&p->Math);

break;

case 6:scanf("%f",&p->Clanguage);

break;

default:break;

}

p=p->next;

i++;

}

}

void output2(struct score\*head)

{

struct score\*p=head;

int i=1;

while(i<=3)

{

printf("sum is %f\n",(p->Clanguage+p->English+p->Math+p->Physic));

printf("average is %f\n",(p->Clanguage+p->English+p->Math+p->Physic)/4);

p=p->next;

i++;

}

}

流程图如图1.4

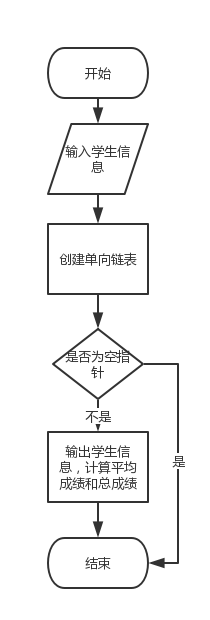
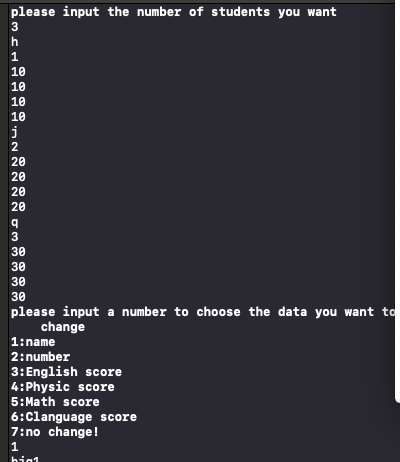


图1.4学生成绩处理流程图

运行结果如图1.5



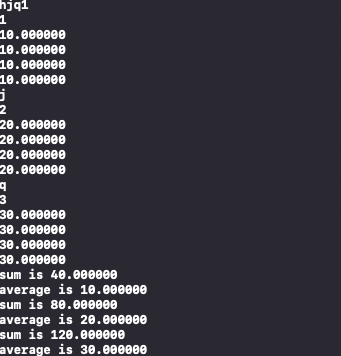
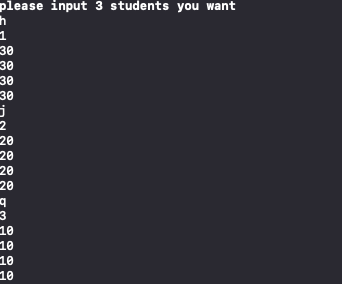


图1.5学生成绩处理运行结果

### 7.2.4.选做题

**以下两题统一输入如图1.8**

****

**图1.8统一输入**

(1) 对上述程序设计题中第（2）题的程序，增加按照平均成绩进行升序排序的函数，试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。

函数代码如下：

void sort(struct score \*head)

{

int i,j;

struct score \*p=head,\*k[3];

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

{

p->average=(p->Clanguage+p->English+p->Math+p->Physic)/4;

k[i]=p;

p=p->next;

}

char temp[20];

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

for(j=0;j<2;j++)

{

if((k[j]->average)>(k[j+1]->average))

{

strcpy(temp,k[j]->name);

strcpy(k[j]->name,k[j+1]->name);

strcpy(k[j+1]->name,temp);

t=k[j]->average;

k[j]->average=k[j+1]->average;

k[j+1]->average=t;

}

}

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

printf("%s\n",k[i]->name);

}

运行结果如图1.6

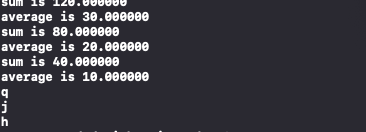


图1.6交换数据域运行结果

1. 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。

代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct score

{

char name[20];

int number;

float English;

float Physic;

float Math;

float Clanguage;

float average;

struct score \*next;

};

void output(struct score\*headp);

void change(struct score \*head);

void output2(struct score\*head);

void sort(struct score \*head);

int main()

{

struct score \* head=NULL,\*tail;

int i;

printf("please input 3 students you want\n");

head=(struct score\*)malloc(sizeof(struct score));

scanf("%s",head->name);

scanf("%d",&head->number);

scanf("%f",&head->English);

scanf("%f",&head->Physic);

scanf("%f",&head->Math);

scanf("%f",&head->Clanguage);

tail=head;

for(i=1;i<3;i++)

{

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

tail=tail->next;

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

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

scanf("%f",&tail->English);

scanf("%f",&tail->Physic);

scanf("%f",&tail->Math);

scanf("%f",&tail->Clanguage);

}

tail->next=NULL;

change(head);

output(head);

output2(head);

sort(head);

return 0;

}

void output(struct score\*head)

{

struct score \*p=head;

while(p)

{

printf("%s\n",p->name);

printf("%d\n",p->number);

printf("%f\n",p->English);

printf("%f\n",p->Physic);

printf("%f\n",p->Math);

printf("%f\n",p->Clanguage);

p=p->next;

}

}

void change(struct score \*head)

{

int n,i=1;

struct score \*p=head;

while(i<=3)

{

printf("please input a number to choose the data you want to change\n");

printf("1:name\n");

printf("2:number\n");

printf("3:English score\n");

printf("4:Physic score\n");

printf("5:Math score\n");

printf("6:Clanguage score\n");

printf("7:no change!\n");

scanf("%d",&n);

switch(n)

{

case 1:scanf("%s",p->name);

break;

case 2:scanf("%d",&p->number);

break;

case 3:scanf("%f",&p->English);

break;

case 4:scanf("%f",&p->Physic);

break;

case 5:scanf("%f",&p->Math);

break;

case 6:scanf("%f",&p->Clanguage);

break;

default:break;

}

p=p->next;

i++;

}

}

void output2(struct score\*head)

{

struct score\*p=head;

int i=1;

while(i<=3)

{

printf("sum is %f\n",(p->Clanguage+p->English+p->Math+p->Physic));

printf("average is %f\n",(p->Clanguage+p->English+p->Math+p->Physic)/4);

p=p->next;

i++;

}

}

void sort(struct score \*head)

{

int i,j;

struct score \*p=head,\*k[3],\*temp=NULL;

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

{

p->average=(p->Clanguage+p->English+p->Math+p->Physic)/4;

k[i]=p;

p=p->next;

}

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

for(j=0;j<2;j++)

{

if((k[j]->average)>(k[j+1]->average))

{

k[j]->next=k[j+1]->next;

k[j+1]->next=k[j];

temp=k[j];

k[j]=k[j+1];

k[j+1]=temp;

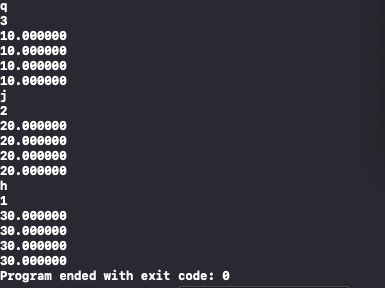
}

}

output(k[0]);

}

运行结果如图1.7



1. 采用双向链表重做编程设计题中的第（2）题。

代码如下：

#include <stdio.h>

#include <stdlib.h>

struct score

{

char name[20];

int number;

float English;

float Physic;

float Math;

float Clanguage;

struct score \*prior;

struct score \*next;

};

void output(struct score\*headp);

void change(struct score \*head);

void output2(struct score\*head);

int main()

{

struct score \* head=NULL,\*tail;

int i;

int n;

printf("please input the number of students you want\n");

scanf("%d",&n);

head=(struct score\*)malloc(sizeof(struct score));

scanf("%s",head->name);

scanf("%d",&head->number);

scanf("%f",&head->English);

scanf("%f",&head->Physic);

scanf("%f",&head->Math);

scanf("%f",&head->Clanguage);

tail=head;

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

{

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

tail->next->prior=tail;

tail=tail->next;

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

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

scanf("%f",&tail->English);

scanf("%f",&tail->Physic);

scanf("%f",&tail->Math);

scanf("%f",&tail->Clanguage);

}

tail->next=NULL;

change(head);

output(head);

output2(head);

}

void output(struct score\*head)

{

struct score \*p=head;

while(p)

{

printf("%s\n",p->name);

printf("%d\n",p->number);

printf("%f\n",p->English);

printf("%f\n",p->Physic);

printf("%f\n",p->Math);

printf("%f\n",p->Clanguage);

p=p->next;

}

}

void change(struct score \*head)

{

int n,i=1;

struct score \*p=head;

while(i<=3)

{

printf("please input a number to choose the data you want to change\n");

printf("1:name\n");

printf("2:number\n");

printf("3:English score\n");

printf("4:Physic score\n");

printf("5:Math score\n");

printf("6:Clanguage score\n");

printf("7:no change!\n");

scanf("%d",&n);

switch(n)

{

case 1:scanf("%s",p->name);

break;

case 2:scanf("%d",&p->number);

break;

case 3:scanf("%f",&p->English);

break;

case 4:scanf("%f",&p->Physic);

break;

case 5:scanf("%f",&p->Math);

break;

case 6:scanf("%f",&p->Clanguage);

break;

default:break;

}

p=p->next;

i++;

}

}

void output2(struct score\*head)

{

struct score\*p=head;

int i=1;

while(i<=3)

{

printf("sum is %f\n",(p->Clanguage+p->English+p->Math+p->Physic));

printf("average is %f\n",(p->Clanguage+p->English+p->Math+p->Physic)/4);

p=p->next;

i++;

}

}

运行结果同程序设计第（2）题

## 7.3小结

本次实验中大量使用链表，相比于数组，链表的创建更加麻烦，而且难以查找，无序。因此，在程序设计中，我通过建立一个结构指针数组来存储各个结点的地址，代替了遍历，可以看到，在不需要插入或删除结点的问题中，数组是优先选择的。此外，在表达式求值中，掌握了链表中运算符的优先级顺序。而在程序替换中，如果要对头指针进行操作，那么函数的形参就应该是二级指针，通过二级指针一次解引用来实现头指针的修改。

# 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 实验题目及要求

### 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');

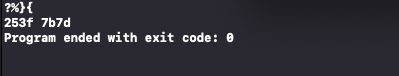
fclose(fp1);

fclose(fp2);

return 0;

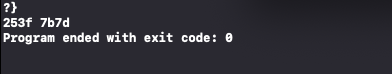
}

1. 请思考程序的输出结果，然后通过上机运行来加以验证。



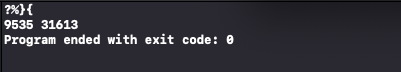
1. 将两处sizeof(short)均改为sizeof(char)结果有什么不同，为什么？

原因：改变之后，读入方式为只读一个字节，而不是同时读入两个字节，因此输出时只输出存储的第一个字节



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

原因：存储方式变为了十进制，因此输出时按十进制输出



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

#include<stdio.h>

#include<stdlib.h>

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

{

char ch;

FILE \*fp;

if(argc!=2){

printf("Arguments error!\n");

exit(-1);

}

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

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

exit(-1);

}

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

putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

return 0;

}

1. 用输入输出重定向freopen改写main函数。
2. 代码如下：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

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

{

char ch;

FILE \*fp;

if (argc != 3) {

printf("Arguments error!\n");

exit(-1);

}

if (strcmp(argv[1], "type") != 0)

{

printf("%s is not a right command", argv[1]);

exit(-1);

}

if ((fp = fopen(argv[2], "r")) == NULL) { /\* fp 指向 filename \*/

printf("Can't open %s file!\n", argv[2]);

exit(-1);

}

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

putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

return 0;

}

运行结果如图1.1

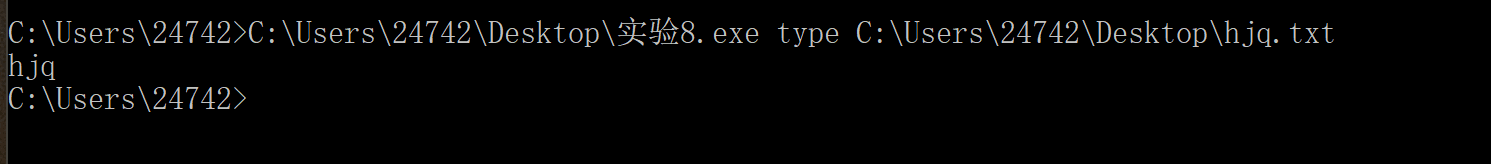


图1.1文件命令行输入运行结果

（2）

代码如下：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

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

{

char ch;

FILE \*fp;

if (argc != 3) {

printf("Arguments error!\n");

exit(-1);

}

if (strcmp(argv[1], "type") != 0)

{

printf("%s is not a right command", argv[1]);

exit(-1);

}

if ((fp = freopen(argv[2], "r",stdin)) == NULL) { /\* fp 指向 filename \*/

printf("Can't open %s file!\n", argv[2]);

exit(-1);

}

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

putchar(ch); /\* 向显示器中写字符 \*/

fclose(fp); /\* 关闭filename \*/

return 0;

}

### 8.2.3．编程设计题

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

代码如下：

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*fp;

char a[40];

int i;

fp=(fopen("test.txt","w"));

gets(a);

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

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

{

if((a[i]>'a'&&a[i]<'z')&&a[i-1]==' ')

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

}

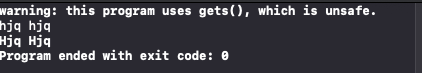
puts(a);

fputc(a,fp);

fclose(fp);

}

运行结果如图1.2



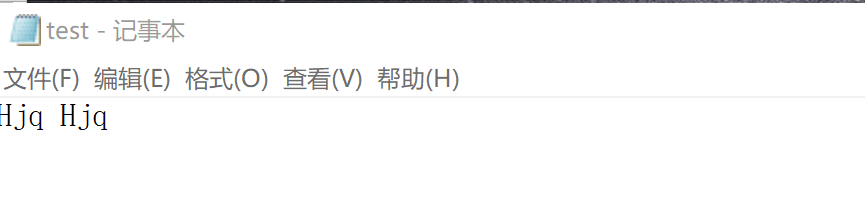


图1.2文件读写运行结果

## 8.3小结

本次实验中，是对文件进行了直接的操作，如读写文件，并将其从编译器里转移到文件里，不再与以前一样，关了编译器，所有数据都会消失，这次的数据即使关闭编译器，它依然存在。此外，通过直接对文件的操作，可以更改输入流，输出流，可以依据个人习惯来操作电脑而不依靠编译器默认的输入输出流。

## 参考文献

[1]曹计昌,卢萍,李开. C语言程序设计,北京：科学出版社,2013

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