2 ——排序·插入排序（直接·折半·希尔）

5 ——排序·交换排序（冒泡·快速·快速递归）

8 ——排序·选择排序（简单选择·堆排序）

11——排序·归并排序

14——排序·多关键码排序·网

16——查找·AVL 平衡二叉树·调整·书

20——查找·树表·二叉排序树

24——数组·稀疏矩阵·三元组表·网·cpp

34——数组·稀疏矩阵·十字链表·网

36——数组·压缩存储·对称矩阵·三角矩阵

————

40——实验报告·八皇后问题

41——实验报告·大数相乘

43——实验报告·大一·结构体·成绩排名

46——实验报告·大一·文件·成绩排名

51——实验报告·矩阵运算

55——实验报告·迷宫问题·队列

61——实验报告·迷宫问题·栈

67——实验报告·一元多项式·文件

77——实验报告·一元多项式计算

排序·插入排序（直接·折半·希尔）

```
//
//  main.c
//  排序·插入排序（直接·折半·希尔）
//
//

#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 100  //顺序表的最大长度,假定顺序表的长度为 100
typedef int KeyType;  //假定关键字类型为整数类型
typedef int OtherType;
typedef struct
{
    KeyType key;  //关键字项
    OtherType other;  //其他项
}DataType;  //数据元素类型
typedef struct
{
    DataType r[MAXSIZE+1];  //r[0]闲置或充当前哨站
    int length;  //顺序表长度
}SqList;  //顺序表类型

SqList init_sqlist(void)
{
    SqList * L=(SqList*)malloc(sizeof(SqList));
    printf("输入串长: ");
    scanf("%d",&L->length);
    printf("输入串的元素: \n");
    for(int i=1;i<=L->length;i++)
        scanf("%d",&L->r[i].key);
    return *L;
}

void StraightInsertSort(SqList* S)
{//对顺序表 s 中的 s->r[1..length]作直接插人排序
    int i,j;
    for(i=2;i<=S->length;i++)
    {
        S->r[0]=S->r[i];  //复制到前哨站
        j=i-1;
        while(S->r[0].key < S->r[j].key)  //记录后移
        {
```

```
            S->r[j+1]=S->r[j];
            j--;
        }
        S->r[j+1]=S->r[0];  //插入到正确位置
    }
}

void BinaryInsertsort(SqList * s)
{//对顺序表 s 作折半插入排序
    int low,high,mid;
    for(int i=2;i<=s->length;i++)
    {
        s->r[0]=s->r[i];  //保存待插入元素
        low=1;
        high=i-1;  //设置初始区间
        while(low<=high)  //该循环语句完成确定插人位置
        {
            mid=(low+high)/2;
            if(s->r[0].key >= s->r[mid].key)
                low=mid+1;  //插入位置在高半区中
            else
                high=mid-1;  //插入位置在低半区中
        }
        for(int j=i-1;j>=high+1;j--)  //high+1 为插入位置
            s->r[j+1] = s->r[j];  //后移元素,留出插入空位
        s->r[high+1]=s->r[0];  //将元素插人
    }
}  //BinaryInsertSort

void ShellInsert(SqList* s,int gap)
{//一趟增量为 gap 的插人排序,gap 为步长
    int i,j;
    for(i=gap+1 ; i<=s->length ; i++)
    {
        if(s->r[i].key < s->r[i-gap].key)  //前大运行,小于时,需将r[i]插入有序表
        {
            s->r[0]=s->r[i];  //为统一算法设置监视哨，暂存
            for(j=i-gap ; j>0 && s->r[0].key<s->r[j].key ; j=j-gap)
                s->r[j+gap]=s->r[j];  //记录后移
            s->r[j+gap]=s->r[0];  //插入到正确位置
        }
        /*
        for(int k=0;k<=s->length;k++)  //分析行
            printf("%d ",s->r[k].key);
```

```
        printf("\n");
         */
    }
    //printf("\n");  //分析行
}
void ShellSort(SqList* s,int gaps[],int t)
{//按增量序列 gaps[0,1,…t-1]对顺序表 s 作希尔排序
    for(int k=0;k<t;k++)
    {
        ShellInsert(s,gaps[k]);   //一趟增量为 gaps[k]的插人排序
        /*
        for(int i=1;i<=s->length;i++)  //分析行
           printf("%d ",s->r[i].key);
        printf("\n");
         */
    }
}

int main()
{
    SqList L=init_sqlist();
    SqList S=L;
    SqList S1=L;
    int i;

    //直接插人排序
    StraightInsertSort(&L);
    printf("直接插人排序:");
    for(i=1;i<=L.length;i++)
        printf("%d ",L.r[i].key);
    printf("\n");

    //折半插入排序
    BinaryInsertsort(&S);
    printf("折半插入排序:");
    for(i=1;i<=L.length;i++)
        printf("%d ",S.r[i].key);
    printf("\n");

    //希尔排序
    int len=S1.length;
    int gaos[10];
    //int gaos[len];//vc6不支持这样写
    int t=0;
```

```
    printf("gaos[]增量(步长): ");
    for(i=0;len>1;i++)
    {
        gaos[i]=len/2;  //步长
        printf("%d ",gaos[i]);
        len/=2;
        t++;
    }
    printf("\n");
    ShellSort(&S1,gaos,t);
    printf("希尔排序: ");
    for(i=1;i<=S1.length;i++)
        printf("%d ",S1.r[i].key);
    printf("\n");
    return 0;
}
/*

 输入串长: 10
 输入串的元素:
 10
 23
 45
 31
 8
 37
 16
 67
 26
 14
 直接插人排序:8 10 14 16 23 26 31 37 45 67
 折半插入排序:8 10 14 16 23 26 31 37 45 67
 gaos[]增量(布长): 5 2 1
 希尔排序: 8 10 14 16 23 26 31 37 45 67

排序相关算法
* 插入排序:
*   1.直接插入排序   稳定
*   2.折半插入排序   稳定
*   3.希尔排序       不稳定
* 快速排序:
*   4.冒泡排序       稳定
*   5.快速排序       不稳定
* 选择排序:
*   6.简单选择排序   不稳定
*   7.堆排序         不稳定
* 归并排序:
*   8.二路归并排序   稳定

 */
```

排序·交换排序（冒泡·快速·快速递归）

```
//
//  main.c
//  排序·交换排序（冒泡·快速·快速递归）
//
//
```

```c
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 100  //顺序表的最大长度,假定顺序表的长度为 100

typedef int KeyType;  //假定关键字类型为整数类型
typedef int OtherType;
typedef struct
{
    KeyType key;  //关键字项
    OtherType other;  //其他项
}DataType;  //数据元素类型
typedef struct
{
    DataType r[MAXSIZE+1];  //r[0]闲置或充当前哨站
    int length;  //顺序表长度
}SqList;  //顺序表类型

SqList init_sqlist(void)
{
    SqList * L=(SqList*)malloc(sizeof(SqList));
    printf("输入串长: ");
    scanf("%d",&L->length);
    printf("输入串的元素: \n");
    for(int i=1;i<=L->length;i++)
        scanf("%d",&L->r[i].key);
    return *L;
}

void BubbleSort(SqList * s)
{//对顺序表 s 作冒泡排序
    int i,j;
    for(i=1;i<=s->length-1;i++)  //进行 n-1 趟排序
        for(j=2;j<=1+s->length-i;j++)
            if(s->r[j].key < s->r[j-1].key)  //S->r[j]与s->r[j-1]交换
            {
                s->r[0]=s->r[j];
                s->r[j]=s->r[j-1];
                s->r[j-1]=s->r[0];
            }
}

//一趟快速排序
int QuickSort1(SqList* s,int low,int high)
{//交换顺序表 S中子表 r[lowhigh」的记录,使轴值(支点)记录到位,并返回其所在位置
```

```
//此时,在它之前(后)的记录均不大(小)于它
    KeyType pivotkey;
    s->r[0]=s->r[low];  //以子表的第一个记录作为轴值(支点)记录
    pivotkey=s->r[low].key;  //取轴值(支点)记录关键字
    while(low<high)  //从表的两端交替地向中间扫描
    {
        while(low<high && s->r[high].key>=pivotkey)
            high--;
        s->r[low]=s->r[high];
        while(low<high && s->r[low].key<=pivotkey)
            low++;  //将比轴值(支点)记录小的交换到低端
        s->r[high]=s->r[low];  //将比轴值(支点)记录大的交换到高端
    }
    s->r[low]=s->r[0];  //轴值(支点)记录到位
    return low;  //返回轴值(支点)记录所在位置
}

//快速排序算法的递归
void QuickSort(SqList* s,int low,int high)  //递归形式的快速排序
{//对顺序表 s 中的子序列 r[low…high)作快速排序
    int pivotloc;
    if(low<high)
    {
        pivotloc=QuickSort1(s,low,high);  //将待排序序列一分为二
        QuickSort(s,low,pivotloc-1);  //对小于轴值序列实现递归排序
        QuickSort(s,pivotloc+1,high);  //对大于轴值序列实现递归排序
    }
}

int main()
{
    SqList L=init_sqlist();
    SqList S=L;
    int i;

    BubbleSort(&L);
    printf("冒泡排序: ");
    for(i=1;i<=L.length;i++)
        printf("%d ",L.r[i].key);
    printf("\n");

    QuickSort(&S, 1, S.length);
    printf("快速排序: ");
    for(i=1;i<=L.length;i++)
```

```
        printf("%d ",S.r[i].key);
    printf("\n");

}
/*

输入串长: 10
输入串的元素:
10
23
45
31
8
37
16
67
26
14
冒泡排序: 8 10 14 16 23 26 31 37 45 67
快速排序: 8 10 14 16 23 26 31 37 45 67

 */
```

排序·选择排序（简单选择·堆排序）

```
//
//  main.c
//  排序·选择排序（简单选择·堆排序）
//
//

#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 100  //顺序表的最大长度,假定顺序表的长度为 100

typedef int KeyType;  //假定关键字类型为整数类型
typedef int OtherType;
typedef struct
{
    KeyType key;  //关键字项
    OtherType other;  //其他项
}DataType;  //数据元素类型
typedef struct
```

```
{
    DataType r[MAXSIZE+1];  //r[0]闲置或充当前哨站
    int length;  //顺序表长度
}SqList;  //顺序表类型

SqList init_sqlist(void)
{
    SqList * L=(SqList*)malloc(sizeof(SqList));
    printf("输入串长: ");
    scanf("%d",&L->length);
    printf("输入串的元素: \n");
    for(int i=1;i<=L->length;i++)
        scanf("%d",&L->r[i].key);
    return *L;
}

//简单选择排序
void SelectSort(SqList * s)
{
    int i,j,t;
    DataType tmp;
    for(i=1;i<s->length;i++)  //作 S->length-1 趟选取
    {
        for(j=i+1,t=i;j<=s->length;j++)
        {//在i开始的 length-i+1条待排序记录中选关键字最小的记录*1
            if(s->r[t].key>s->r[j].key)
            t=j;  //t中存放关键字最小的记录下标
        }
        tmp=s->r[t];
        s->r[t]=s->r[i];
        s->r[i]=tmp;  //关键字最小的记录与第 i条记录交换
    }
}

//大顶堆
void HeapAdjust(SqList* s,int n,int len)
{//S->r[n…len]中的记录关键字除r[n]外均满足堆的定义,本函数将对第n个结点为根的子树筛选,使其成为大顶堆
    int j;
    DataType rc=s->r[n];  //先对操作位置上的结点数据进行保存，放置后序移动元素丢失。
    for(j=2*n;j<=len;j=j*2)  //沿关键字较大的孩子结点向下筛选
    {
        if(j<len && s->r[j].key < s->r[j+1].key)
            j++;  //如果当前结点比最大的孩子结点的值还大，跳过
        if(rc.key > s->r[j].key)  //rc应插入在位置i上
```

```
            break;
        s->r[n]=s->r[j];
        n=j;  //使n结点满足堆定义
    }
    s->r[n]=rc;
    /*
    for(j=1;j<=s->length;j++)//分析行
        printf("%d ",s->r[j].key);
    printf("\n");
    */
}

//堆排序实质上是对无序序列不断建堆和调整堆的过程
//堆排序算法
void HeapSort(SqList *s)
{
    int i;
    DataType tmp;
    for(i=s->length/2;i>0;i--)  //将r[1..length]建成堆
        HeapAdjust(s,i,s->length);
    //printf("\n");  //分析行
    for(i=s->length;i>1;i--)  //将大顶堆改成小顶堆
    {
        tmp=s->r[1];  //堆顶与堆底元素交换,将最大元素移到后面
        s->r[1]=s->r[i];
        s->r[i]=tmp;
        HeapAdjust(s,1,i-1);  //将r[1..i-1]重新调整为堆
    }
}

int main()
{
    SqList L=init_sqlist();
    SqList S=L;
    int i;

    SelectSort(&L);
    printf("简单选择排序: ");
    for(i=1;i<=L.length;i++)
        printf("%d ",L.r[i].key);
    printf("\n");

    HeapSort(&S);
    printf("堆排序-大顶堆: ");
```

```
    for(i=1;i<=S.length;i++)
        printf("%d ",S.r[i].key);
    printf("\n");

    return 0;
}
/*
                                        10
输入串长: 5                             23
输入串的元素:                           45
5                                       31
12                                      8
9                                       37
23                                      16
18                                      67
简单选择排序: 5 9 12 18 23              26
堆排序-大顶堆: 5 9 12 18 23             14
                                        */
```

排序·归并排序

```
//
//  main.c
//  排序·归并排序
//
//

#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 100  //顺序表的最大长度,假定顺序表的长度为 100

typedef int KeyType;  //假定关键字类型为整数类型
typedef int OtherType;
typedef struct
{
    KeyType key;  //关键字项
    OtherType other;  //其他项
}DataType;  //数据元素类型
typedef struct
{
    DataType r[MAXSIZE+1];  //r[0]闲置或充当前哨站
    int length;  //顺序表长度
}SqList;  //顺序表类型
```

```
SqList init_sqlist(void)
{
    int a[10]={10,23,45,31,8,37,16,67,26,14};
    SqList * L=(SqList*)malloc(sizeof(SqList));
    //printf("输入串长：");
    L->length=10;// 临时
    //scanf("%d",&L->length);
    //printf("输入串的元素：\n");
    for(int i=1;i<=L->length;i++)
    {
        L->r[i].key=a[i-1];  //临时
        //scanf("%d",&L->r[i].key);
    }
    return *L;
}

void Merge(DataType r[],DataType rf[],int u,int v,int t)
{//将有序的 r[u··v]和 r[v+1··t]归并为有序的 rf[u··t]
    int i,j,k;
    for(i=u,j=v+1,k=u;i<=v && j<=t;k++)  //将r中记录由小到大并到tf
    {
        if(r[i].key<=r[j].key)
        { rf[k]=r[i];i++;}  //j大给i
        else
        { rf[k]=r[j];j++;}  //i大给j
    }
    while(i<=v){rf[k++]=r[i++];}  //将剩余的 r[i··v]复制到tf
    while(j<=t){rf[k++]=r[j++];}  //将剩余的 r[j··t]复制到tf
}
//二路归并排序主要将整个序列划分为两个有序的子序列，然后将这两个子序列进行
//归并操作
void MSort(DataType p[],DataType p1[],int n,int t)
{//将 p[n··t]归并排序为 p1[n··t]
    int m;
    DataType p2[MAXSIZE+1];  //中间变量,存放部分排序结果
    if(n==t)
        p1[n]=p[n];  //P中只有一个元素，不需要进行归并操作
    else
    {
        m=(n+t)/2;  //平分待排序的序列
        MSort (p,p2,n,m);    //将 p[n··m]归并为有序的 p2[n··m],调用递归过程实现
        MSort (p,p2,m+1,t);  //将 p[m+1··t]归并为有序的 p2[m+1··t],调用递归过程实现
        Merge (p2,p1,n,m,t); //将 p2[n··m]和 p2[m+1··t]归并到 p1[n··t]
```

```
    }
}
void MergeSort(SqList * S)
{//对顺序表 S 作归并排序
    MSort(S->r,S->r,1,S->length);
}
int main()
{
    SqList L=init_sqlist();
    int i;

    MergeSort(&L);
    printf("归并排序: ");
    for(i=1;i<=L.length;i++)
        printf("%d ",L.r[i].key);
    printf("\n");
}
/*
 数组赋值方法:
 int z[5]={1,1,2,4};
 int n[10]={0};  //将n数组里的全部初始化为0
 char z[]="zhinanzhen";
 char t[6]={'s','t','u','d','e','n','t','\0'};
 char x[]="can you ";

 输入串长: 10
 输入串的元素:
 10
 23
 45
 31
 8
 37
 16
 67
 26
 14
 归并排序: 8 10 14 16 23 26 31 37 45 67
 */
```

排序·多关键码排序·网

```
//
//  main.c
//  排序·多关键码排序·网
//
//

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

#define RADIX 101 //基数，分数有 101 种可能
#define K 3 //关键字，有 3 个关键字

struct tagMark
{
    int key[K]; //有 K 个关键字
}a[8]={
    {1,2,3},
    {0,2,3},
    {5,4,6},
    {6,2,6},
    {4,4,1},
    {0,1,4},
    {60,30,6},
    {60,20,6}};

/*
 a：待排序的数组地址
 size：元素数量
 radix：基数
 k：关键字数量
 */

void LSDSort(struct tagMark *a, int size,int k)
{//最次位优先
    int i;
    struct tagMark *tmp=(struct tagMark *)malloc(sizeof(struct tagMark) *size); //待排序的
记录的数量
    while (k--)
    {
        int cnt[RADIX]={0};
        //计数
        for(i=size;i>0;)
```

```
            ++cnt[a[--i].key[k]];
        //累加cnt
        for(i=0;i<RADIX;i++)
            cnt[i+1]+=cnt[i];
        //按顺序把a[i]放入tmp中
        for(i=size;i>0;)
        {
            --i;
            tmp[--cnt[a[i].key[k]]] = a[i];
        }
        //把a[i]按照某一位排好的序，重新排序
        for(i=size;i>0;)
        {
            --i;
            a[i] = tmp[i];
        }
    }
}

int main()
{
    int i, j;
    LSDSort(a, 8, K); //对8个记录进行排序
    for(i=0;i<8;i++)
    {
        for(j=0;j<K;j++)
            printf("%d ",a[i].key[j]);
        printf("\n");
    }
    return 0;
}
/*
可能运行不了，不知道为什么
输出：
0 1 4
0 2 3
1 2 3
4 4 1
5 4 6
6 2 6
60 20 6
60 30 6
*/
```

查找·AVL 平衡二叉树·调整·书

```
//
//  main.c
//  查找·AVL平衡二叉树·调整·书
//
//

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

typedef int KeyType;
typedef struct
{
    KeyType key;
}DataType;

typedef struct AVLNode
{
    DataType elem;  //elem含有关键字域
    int bf;         //bf记录平衡因子
    struct AVLNode *lchild;
    struct AVLNode *rchild;
}AVLNode,*AVLTree;

//LL型调整代码
AVLTree LL_Rotate(AVLTree a)
{//对以a为当前结点的最小不平衡子树进行LL型调整
    AVLTree b;
    b=a->lchild;   //b指向a的左子树根结点
    a->lchild=b->rchild; //b的右子树挂接为a的左子树
    b->rchild=a;
    a->bf=b->bf=0;//调整结点的平衡因子
    return(b);
}

//RR型调整代码
AVLTree RR_Rotate(AVLTree a)
{//对以a为当前结点的最小不平衡子树进行RR型调整
    AVLTree b;
    b=a->rchild;          //b指向a的右子树根结点
    a->rchild=b->lchild;  //b的左子树挂接为a的右子树
    b->lchild=a;
    a->bf=b->bf=0;        //调整结点的平衡因子
```

```
    return b;
}

//LR型调整代码
AVLTree LR_Rotate(AVLTree a)
{   //对以a为当前结点的最小不平衡子树进行LR型调整
    AVLTree b,c;
    b=a->lchild;
    c=b->rchild;
    a->lchild=c->rchild;  // c的右子树挂接为a的左子树
    b->rchild=c->lchild;  // c的左子树挂接为b的右子树
    c->lchild=b;
    c->rchild=a;
    if (c->bf==1)        // 调整结点的平衡因子
    {
        a->bf=-1;
        b->bf=0;
    }
    else if (c->bf==-1)
    {
        a->bf=0;
        b->bf=1;
    }
    else
        a->bf=b->bf=0;
    c->bf=0;
    return(c);
}

//RL型调整代码
AVLTree RL_Rotate(AVLTree a)
{// 对以a为当前结点的最小不平衡子树进行RL型调整
    AVLTree b,c;
    b=a->rchild;
    c=b->lchild;
    a->rchild=c->lchild;      //c的左子树挂接为a的右子树
    b->lchild=c->rchild;      //c的右子树挂接为b的左子树
    c->lchild=a;             //c指向a的左子树根结点
    c->rchild=b;             //c指向b的右子树根结点
    if (c->bf==1)            //调整结点的平衡因子
    {
        a->bf=0;
        b->bf=-1;
    }
```

```
    else if (c->bf==-1)
    {
        a->bf=1;
        b->bf=0;
    }
    else
        a->bf=b->bf=0;
    c->bf=0;
    return(c);
}

//AVL平衡二叉树 查找 和 插入 算法
void AVLInsert(AVLTree *pavlt,AVLTree s)
{//将结点 s 插入到以 *pavlt 为根结点的平衡二叉排序树中
    AVLTree f,a,b,p,q;
    if(*pavlt==NULL)   //AVL 树为空
    {
        *pavlt=s;
        return;
    }
    a=*pavlt;
    f=NULL;  //指针a记录离*s最近的平衡因子不为0的结点,f指向*a的父结点
    p=*pavlt;
    q=NULL;
    while(p!=NULL)    //寻找插人结点的位置及最小不平衡子树
    {
        if(p->elem.key == s->elem.key)
            return;   //AVL树中已存在该关键字
        if(p->bf!=0)  //寻找最小不平衡子树
        {
            a=p;
            f=q;
        }
        q=p;
        if(s->elem.key < p->elem.key)
            p=p->lchild;
        else
            p=p->rchild;
    }
    if(s->elem.key < q->elem.key)
        q->lchild=s; //将结点*s插入到合适的位置上去
    else
        q->rchild=s;
    p=a;
```

```
    while(p!=s)      //插人结点后,修改相关结点的平衡因子
    {
        if(s->elem.key < p->elem.key)
        {
            p->bf++;
            p=p->lchild;
        }
        else
        {
            p->bf--;
            p=p->rchild;
        }
    }
    if(a->bf > -2 && a->bf < 2)
        return;             //插人结点后,没有破坏树的平衡性
    if(a->bf==2)
    {
        b=a->lchild;
        if(b->bf==1)         //结点插在*a的左孩子的左子树中
            p=LL_Rotate(a);        //LL 型调整
        else                //结点插在*a的左孩子的右子树中
            p=LR_Rotate(a);        //LR 型调整
    }
    else
    {
        b=a->rchild;
        if(b->bf==1)         //结点插在*a的右孩子的左子树中
            p=RL_Rotate(a);        //RL 型调整
        else                //结点插在*a的右孩子的右子树中
            p=RR_Rotate(a);        //RR 型调整
    }
    if(f==NULL)              //原*a是AVL树的根
        *pavlt=p;
    else if(f->lchild==a)    //将新子树链到原结点*a的双亲结点上
        f->lchild=p;
    else
        f->rchild=p;
}

int main()
{
    printf("看不懂怎么办~~\n");
    return 0;
}
```

查找·树表·二叉排序树

```
//
//  main.c
//  查找·树表·二叉排序树
//
//

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
typedef int KeyType;
typedef struct
{
    KeyType key;
}DataType;
typedef struct BinSTreeNode
{
    DataType  elem;   //elem含有关键字域
    struct BinSTreeNode  *lchild;
    struct BinSTreeNode  *rchild;
}*BinSTree;

//二叉排序树查找算法
BinSTree BSTreeSearch(BinSTree t ,KeyType k )
{//在根指针为t的二叉排序树中查找关键字为k的结点，若查找成功，则返回指向该结点的指针；否则返回空指针
    if (t==NULL)
        return NULL;
    if (t->elem.key==k)
        return t; //查找成功,返回值t一直不能带出，可能是递归原因
    if (t->elem.key>k)
        BSTreeSearch(t->lchild ,k); //在左子树中查找
    else
        BSTreeSearch(t->rchild ,k); //在右子树中查找
    return NULL;//返回值一直为NULL
}
//查找
BinSTree BSTreeSearch1(BinSTree t ,KeyType k )
{//在上面的查找BSTreeSearch()中，返回值t一直不能带出，可能是递归原因，所以用了BSTreeSearch1()
    if (t==NULL)
    {
        printf("\n二叉树排序未查找到\n");//查找失败
        return NULL;
    }
```

```
    if (t->elem.key==k)
    {
        printf("\n二叉树排序查找到了： %d\n",t->elem.key);//查找成功
        return t;
    }
    if (t->elem.key>k)
        BSTreeSearch1(t->lchild ,k); //在左子树中查找
    else
        BSTreeSearch1(t->rchild ,k); //在右子树中查找
    return NULL;
}

//二叉排序树插入算法
BinSTree BSTreeInsert (BinSTree *t , KeyType k)
{//在二叉排序树中插入关键字为k的结点，*t指向二叉排序树的根结点
    BinSTree r;
    if (*t==NULL)
    {
        r=(BinSTree)malloc(sizeof(struct BinSTreeNode));
        r->elem.key=k;
        r->lchild=r->rchild=NULL;
        *t=r;     //若二叉排序树为空，被插结点作为树的根结点
        return *t;
    }
    else if(k<((*t)->elem.key))
        BSTreeInsert(&((*t)->lchild),k);  //插入到p的左子树中
    else
        BSTreeInsert(&((*t)->rchild),k);  //插入到p的右子树中
    return NULL;
}

//二叉排序树的结点删除算法
int BSTreeDelete (BinSTree bt , KeyType k)
{//在二叉排序树中删除关键字为k的结点，*bt指向二叉排序树的根结点。删除成功返回1，不成功返回0
    BinSTree f,p,q,s;
    p=bt;
    f=NULL;
    while (p&& p ->elem.key!=k)
    {
        f=p;        //f为指向结点*p的双亲结点的指针
        if (p->elem.key>k)
            p=p->lchild;   //搜索左子树
        else
            p=p->rchild;   //搜索右子树
```

```
    }
    if (p==NULL)
        return 0;    //找不到待删的结点时返回
    if (p->lchild==NULL)     //待删结点的左子树为空
    {
        if (f==NULL)     //待删结点为根结点
            bt=p->rchild;
        else if (f->lchild==p)  //待删结点是其父结点的左孩子
            f->lchild=p->rchild ;
        else            //待删结点是其父结点的右孩子
            f->rchild=p->rchild;
        free (p);
    }
    else     //待删结点有左子树
    {
        q=p;
        s=p->lchild;
        while(s->rchild)   //在待删结点的左子树中查找最右下结点
        {
            q=s;
            s=s->rchild;
        }
        if (q==p)       //将最右下结点的左子树链到待删结点上
            q->lchild=s->lchild;
        else
            q->rchild=s->lchild;
        p->elem.key=s->elem.key;
        free (s) ;
        return 1;
   }
    return 0;
}

void inorder1(BinSTree t)
{//中序遍历的递归算法
    if(t)
    {
        inorder1(t->lchild);
        printf("%d ",t->elem.key);
        inorder1(t->rchild);
    }
}

int main()
```

```
{
    BinSTree t=NULL,r=NULL;
    int i,j=0;
    int a[10],b[10];

    srand((unsigned)time(NULL));
    int randnum=rand()%9+1;  //从 1 到（9+1-1）的随机数
    printf("二叉排序树要查找的数在第 %d 位\n",randnum);

    printf("二叉排序树要排序的数为：");
    for(i = 0; i<10;i++)
    {
        a[i]=rand()%90+10;  //产生随机数
        b[i]=a[i];
        printf("%d ",a[i]);
    }

    //二叉树插入，中序遍历，顺序输出
    for(i=0;i<10;i++)
    {
        if(j==0)  //只运行一次，记录根结点r
        {
            r=BSTreeInsert(&t, a[i]);
            j++;
        }
        else
            t=BSTreeInsert(&r, a[i]);  //t为插入时所在结点，没用处
        //printf("\n");  //分析行
        //inorder1(r);  //分析行
    }
    printf("\n二叉排序树排序结果为：");
    inorder1(r);

    //查找结点
    BSTreeSearch1(r, a[randnum]);

    //删除结点
    BSTreeDelete(r, a[randnum]);
    printf("二叉排序树删除 %d 为：",a[randnum]);
    inorder1(r);

    //查找结点
    BSTreeSearch1(r, a[randnum]);
    return 0;
```

```
}
/*

 二叉排序树要查找的数在第 8 位
 二叉排序树要排序的数为: 57 10 43 23 45 59 37 97 53 41
 二叉排序树排序结果为: 10 23 37 41 43 45 53 57 59 97
 二叉树排序查找到了: 53
 二叉排序树删除 53 为: 10 23 37 41 43 45 57 59 97
 二叉树排序未查找到

 */
```

数组·稀疏矩阵·三元组表·网·cpp

```
//
//  main.c
//  数组·稀疏矩阵·三元组表·网·cpp
//
//

#include<iostream>
#define maxtemr 100
using namespace std;

class SparseMatrix; //向前声明

class MatrixTerm//矩阵元素类
{
public:
    int row;//行
    int col;//列
    int value;//值
    friend SparseMatrix;
};

class SparseMatrix
{
private:
    int Rows;//行数
    int Cols;//列数
    int Terms;//非零项个数
    MatrixTerm smArray[maxtemr];//maxterm是常数，maxterm<<Rows*Cols
    int cpot[20];
```

```
public:
    void InitMatrix(int count);  //建立矩阵
    SparseMatrix Transpose1();   //矩阵转置算法1
    SparseMatrix Transpose2();   //矩阵转置算法2
    bool Multiply(SparseMatrix b,SparseMatrix &q);//矩阵相乘
    bool Add(SparseMatrix b,SparseMatrix &c);     //矩阵相加
    void print();   //打印三元表
    void print1();  //打印矩阵
};
int arry[6][6] = {
    {1, 0, 0, 0, 0, 0},
    {0, 2, 0, 4, 0, 0},
    {3, 0, 0, 0, 0, 4},
    {0, 0, 0, 3, 0, 3},
    {0, 4, 0, 0, 2, 0},
    {0, 0, 4, 0, 0, 1}
};
int arry1[6][6] = {
    {0, 5, 0, 0, 2, 0},
    {2, 0, 4, 0, 0, 2},
    {0, 0, 0, 0, 0, 3},
    {0, 0, 0, 2, 3, 0},
    {0, 0, 2, 0, 0, 0},
    {0, 2, 0, 4, 0, 0}
};
int arry2[6][6] = {
    {0, 2, 0, 2, 0, 3},
    {1, 0, 4, 0, 0, 1},
    {0, 4, 0, 0, 0, 0},
    {0, 0, 2, 0, 4, 0},
    {3, 0, 0, 0, 1, 0},
    {0, 0, 0, 2, 0, 0}
};
void SparseMatrix::InitMatrix(int count)
{
    Terms=0;
    int r,c,s=0,val,m;
    //cout<<"输入要创建的矩阵的行数和列数：";
    //cin>>r>>c;
    r=6;  //临时，简化输入
    c=6;  //临时，简化输入
    Rows=r;
    Cols=c;
    //cout<<"请输入矩阵中各元素的值，请输入"<<r*c<<"个整数"<<endl;
```

```
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            //cin>>val;
            if(count==0) //临时，简化输入
                val=arry[i][j];
            else if(count==1)
                val=arry1[i][j];
            else
                val=arry2[i][j];

            if(val!=0)
            {
                smArray[s].row=i;
                smArray[s].col=j;
                smArray[s].value=val;
                s++;
            }
        }
    }

    Terms=s;
    int *num=new int[r];

    for(m=0;m<Rows;m++)
        num[m]=0;

    for(m=0;m<Terms;m++)
        ++num[smArray[m].row];

    cpot[0]=0;

    for(m=1;m<Rows;m++)
        cpot[m]=cpot[m-1]+num[m-1];

}

SparseMatrix SparseMatrix::Transpose1()
{
    SparseMatrix b;
    b.Rows=Cols;
    b.Cols=Rows;
    b.Terms=Terms;
```

```
    if(b.Terms)
    {
        int currentB=0;
        for(int c=0;c<Cols;c++)
            for(int i=0;i<Terms;i++)
                if(smArray[i].col==c)//找到c列中的元素
                {
                    b.smArray[currentB].row=c;
                    b.smArray[currentB].col=smArray[i].row;
                    b.smArray[currentB].value=smArray[i].value;
                    currentB++;
                }
    }
    return b;
}

SparseMatrix SparseMatrix::Transpose2()
{
    SparseMatrix b;
    b.Rows=Cols;
    b.Cols=Rows;
    b.Terms=Terms;

    int *num;
    int *cpot;
    int c;
    num=new int[Cols];
    cpot=new int[Cols];

    if(b.Terms)
    {
        for(c=0;c<Cols;c++)
            num[c]=0;

        for(c=0;c<b.Terms;c++)
            ++num[smArray[c].col];
        cpot[0]=0;

        for(c=1;c<Cols;c++)
            cpot[c]=cpot[c-1]+num[c-1];

        for(int p=0;p<b.Terms;p++)
        {
            int j=smArray[p].col;
```

```
            int q=cpot[j];
            b.smArray[q].row=smArray[p].col;
            b.smArray[q].col=smArray[p].row;
            b.smArray[q].value=smArray[p].value;
            ++cpot[j];
        }
    }
    return b;
}


bool SparseMatrix::Multiply(SparseMatrix b,SparseMatrix &q)
{
    /*
    返回两个矩阵的乘积，用C来装
    采用与第二种转置的算法一样的做法，给每个矩阵增加一个数组，用来记录每一行第一个非零值在
    在矩阵smArray[]中的位置，这个数组就是cpot[],有这样结构的稀疏矩阵叫做行逻辑连接顺序表
    计算过程如下：
    1.    由于矩阵a和Q的行数相等并且C语言以行为主序进行存储，所以以a进行逐行的扫描。
    2.    使Q的此行逻辑表的序号等于其非零元个数Q.terms+1，以表示其行的首个元素的序号。
    3.    从行中找到a的非零元，并以它的列值为b的行号，对b进行行的扫描，若存在，
          则依次计算它们，并把其值累加到一个以b中这个对应非零元的列值为序号的临时数组q_sum[q_col]中。
    4.    在a的当前行完成扫描后，将q_sum[q_col]不为0的值，压入到Q矩阵的三元组，累加Q.Terms
    */

    int t1,t2,q_col;
    int *q_sum;//用于存放结果的临时数组

    if(this->Cols!=b.Rows)
        return false;

    q.Rows=this->Rows;
    q.Cols=b.Cols;
    q.Terms=0;
    q_sum=new int[b.Cols];
    if(this->Terms*b.Terms!=0)
    {
        for(int a_row=0;a_row<this->Rows;a_row++)//以a的每一行为标准，进行运算
        {
            for(int i=0;i<b.Cols;i++)
                q_sum[i]=0;

            q.cpot[a_row]=q.Terms;
```

```
            if(a_row<this->Rows-1)
                t1=this->cpot[a_row+1];
            else
                t1=this->Terms;

            for(int p=this->cpot[a_row];p<t1;p++)
            {
                int b_row=this->smArray[p].col;

                if(b_row<b.Rows-1)
                    t2=b.cpot[b_row+1];
                else
                    t2=b.Terms;

                for(int q=b.cpot[b_row];q<t2;q++)
                {
                    q_col=b.smArray[q].col;
                    q_sum[q_col]+=this->smArray[p].value*b.smArray[q].value;
                }
            }

            for(q_col=0;q_col<q.Cols;q_col++)
                if(q_sum[q_col])
                {
                    q.smArray[q.Terms].row=a_row;
                    q.smArray[q.Terms].col=q_col;
                    q.smArray[q.Terms].value=q_sum[q_col];
                    ++q.Terms;
                }
        }
    }
    return true;
}

bool SparseMatrix::Add(SparseMatrix b,SparseMatrix &c)
{
    /*
    计算过程是这样的，一次扫描两个矩阵a,b的smArray[]中的每个数据，
    比较两个数据，如果行号相同，则比较列号，若列号不想等则将列号小的存入矩阵c
    的smArray[]中，若列号相等，则计算，计算完非零，再存入c中；
    如果a的当前项行号小于b的当前项行号，那么就将a的项存入c，反之，存b
    */
    if(this->Rows!=b.Rows||this->Cols!=b.Cols)
        return false;
```

```
int sum=0;
int p=0;//指向a的当前项
int t=0;//指向b的当前项
int k=0;//指向c的当前项
while( p<this->Terms && t<b.Terms)
{
    if(this->smArray[p].row==b.smArray[t].row)
    {
        if(this->smArray[p].col==b.smArray[t].col)
        {
            sum=this->smArray[p].value+b.smArray[t].value;
            if(sum)
            {
                c.smArray[k].row=b.smArray[t].row;
                c.smArray[k].col=b.smArray[t].col;
                c.smArray[k].value=sum;
                k++;p++;t++;
            }
        }
        else if(this->smArray[p].col<b.smArray[t].col)
        {
            c.smArray[k].row=this->smArray[p].row;
            c.smArray[k].col=this->smArray[p].col;
            c.smArray[k].value=this->smArray[p].value;
            k++;p++;
        }
        else
        {
            c.smArray[k].row=b.smArray[t].row;
            c.smArray[k].col=b.smArray[t].col;
            c.smArray[k].value=b.smArray[t].value;
            k++;t++;
        }
    }
    else if(this->smArray[p].row<b.smArray[t].row)
    {
        c.smArray[k].row=this->smArray[p].row;
        c.smArray[k].col=this->smArray[p].col;
        c.smArray[k].value=this->smArray[p].value;
        k++;p++;
    }
    else
    {
        c.smArray[k].row=b.smArray[t].row;
```

```
            c.smArray[k].col=b.smArray[t].col;
            c.smArray[k].value=b.smArray[t].value;
            k++;t++;
        }
    }
    while(p<this->Terms)
    {
        c.smArray[k].row=this->smArray[p].row;
        c.smArray[k].col=this->smArray[p].col;
        c.smArray[k].value=this->smArray[p].value;
        k++;p++;
    }
    while(t<b.Terms)
    {
        c.smArray[k].row=b.smArray[t].row;
        c.smArray[k].col=b.smArray[t].col;
        c.smArray[k].value=b.smArray[t].value;
        k++;t++;
    }
    c.Rows=b.Rows;
    c.Cols=b.Cols;
    c.Terms=k;
    return true;
}

void SparseMatrix::print()//输出三元表
{
    int p=Terms;
    for(int i=0;i<p;i++)
        cout<<"smArray["<<smArray[i].row<<"]["<<smArray[i].col<<"]=
"<<smArray[i].value<<endl;
    cout<<endl;
}
void SparseMatrix::print1()//输出矩阵
{
    int m=0;
    for(int i=0;i<6;i++)
    {
        for(int j=0;j<6;j++)
        {
            if(smArray[m].row==i&&smArray[m].col==j)
            {
                cout<<smArray[m].value<<" ";
                m++;
```

```
            }
            else
                cout<<"0 ";
        }
        cout<<endl;
    }
    cout<<endl;
}

int main()
{
    SparseMatrix M,Z,T,Y,K,L;

    cout<<"矩阵 M 为: "<<endl;
    M.InitMatrix(0);
    M.print();
    M.print1();

    cout<<"矩阵 M 的转置为: "<<endl;
    K=M.Transpose2();
    //Z=M.Transpose1();
    K.print();
    K.print1();

    cout<<"矩阵 Z 为: "<<endl;
    Z.InitMatrix(1);
    Z.print();
    Z.print1();
    if(M.Multiply(T,Y))
    {
        cout<<"矩阵 T 与矩阵 M 相乘为: "<<endl;
        Y.print();
        Y.print1();
    }

    cout<<"矩阵 T 为: "<<endl;
    T.InitMatrix(2);
    T.print();
    T.print1();
    if(M.Add(T,L))
    {
        cout<<"矩阵 M 和矩阵 T 相加为: "<<endl;
        L.print();
        L.print1();
```

```
    }
    return 0;
}
/*
 矩阵 M 为:
 smArray[0][0]= 1
 smArray[1][1]= 2
 smArray[1][3]= 4
 smArray[2][0]= 3
 smArray[2][5]= 4
 smArray[3][3]= 3
 smArray[3][5]= 3
 smArray[4][1]= 4
 smArray[4][4]= 2
 smArray[5][2]= 4
 smArray[5][5]= 1

 1 0 0 0 0 0
 0 2 0 4 0 0
 3 0 0 0 0 4
 0 0 0 3 0 3
 0 4 0 0 2 0
 0 0 4 0 0 1

 矩阵 M 的转置为:
 smArray[0][0]= 1
 smArray[0][2]= 3
 smArray[1][1]= 2
 smArray[1][4]= 4
 smArray[2][5]= 4
 smArray[3][1]= 4
 smArray[3][3]= 3
 smArray[4][4]= 2
 smArray[5][2]= 4
 smArray[5][3]= 3
 smArray[5][5]= 1

 1 0 3 0 0 0
 0 2 0 0 4 0
 0 0 0 0 0 4
 0 4 0 3 0 0
 0 0 0 0 2 0
 0 0 4 3 0 1

 矩阵 Z 为:
 smArray[0][1]= 5
 smArray[0][4]= 2
 smArray[1][0]= 2
 smArray[1][2]= 4
 smArray[1][5]= 2
 smArray[2][5]= 3
 smArray[3][3]= 2
 smArray[3][4]= 3
 smArray[4][2]= 2
 smArray[5][1]= 2
 smArray[5][3]= 4

 0 5 0 0 2 0
 2 0 4 0 0 2
 0 0 0 0 0 3
 0 0 0 2 3 0
 0 0 2 0 0 0
 0 2 0 4 0 0

 矩阵 T 为:
 smArray[0][1]= 2
 smArray[0][3]= 2
 smArray[0][5]= 3
 smArray[1][0]= 1
 smArray[1][2]= 4
 smArray[1][5]= 1
 smArray[2][1]= 4
 smArray[3][2]= 2
 smArray[3][4]= 4
 smArray[4][0]= 3
 smArray[4][4]= 1
 smArray[5][3]= 2

 0 2 0 2 0 3
 1 0 4 0 0 1

 0 4 0 0 0 0
 0 0 2 0 4 0
 3 0 0 0 1 0
 0 0 0 2 0 0

 矩阵 M 和矩阵 T 相加为:
 smArray[0][0]= 1
 smArray[0][1]= 2
 smArray[0][3]= 2
 smArray[0][5]= 3
 smArray[1][0]= 1
 smArray[1][1]= 2
 smArray[1][2]= 4
 smArray[1][3]= 4
 smArray[1][5]= 1
 smArray[2][0]= 3
 smArray[2][1]= 4
 smArray[2][5]= 4
 smArray[3][2]= 2
 smArray[3][3]= 3
 smArray[3][4]= 4
 smArray[3][5]= 3
 smArray[4][0]= 3
 smArray[4][1]= 4
 smArray[4][4]= 3
 smArray[5][2]= 4
 smArray[5][3]= 2
 smArray[5][5]= 1

 1 2 0 2 0 3
 1 2 4 4 0 1
 3 4 0 0 0 4
 0 0 2 3 4 3
 3 4 0 0 3 0
 0 0 4 2 0 1
 */
```

数组·稀疏矩阵·十字链表·网

```
//
//  main.c
//  数组·稀疏矩阵·十字链表·网
//
//

#include<stdio.h>
#include<stdlib.h>
typedef struct OLNode
{
    int i,j,e;      //矩阵三元组i代表行 j代表列 e代表当前位置的数据
    struct OLNode *right,*down; //指针域 右指针 下指针
}OLNode, *OLink;

typedef struct
{
    OLink *rhead,*chead; //行和列链表头指针
    int mu,nu,tu;       //矩阵的行数,列数和非零元的个数
}CrossList;

CrossList CreateMatrix_OL(CrossList M)
{
    int m, n, t;
    int i, j, e;
    OLNode *p, *q;
    printf("输入矩阵的行数、列数和非0元素个数: \n");
    scanf("%d%d%d", &m, &n, &t);
    M.mu = m;
    M.nu = n;
    M.tu = t;
    M.rhead = (OLink*)malloc((m + 1) * sizeof(OLink));
    M.chead = (OLink*)malloc((n + 1) * sizeof(OLink));
    if(!M.rhead||!M.chead)
    {
        printf("初始化矩阵失败");
        exit(0);
    }
    for (i = 1; i <= m; i++)
        M.rhead[i] = NULL;
    for (j = 1; j <= n; j++)
        M.chead[j] = NULL;
    printf("输入三元组:\n");
```

```
    for(scanf("%d%d%d", &i, &j, &e);0!=i;scanf("%d%d%d", &i, &j, &e))
    {//当i=0时不再输入
        p=(OLNode*)malloc(sizeof(OLNode));
        if(!p)
        {
            printf("初始化三元组失败");
            exit(0);
        }
        p->i = i;
        p->j = j;
        p->e = e;
        //链接到行的指定位置
        if (NULL == M.rhead[i] || M.rhead[i]->j>j)
        {
            p->right = M.rhead[i];
            M.rhead[i] = p;
        }
        else
        {
            for (q = M.rhead[i]; (q->right) && q->right->j<j;q=q->right);
            p->right = q->right;
            q->right = p;
        }
        //链接到列的指定位置
        if (NULL==M.chead[j]||M.chead[j]->i>i)
        {
            p->down = M.chead[j];
            M.chead[j] = p;
        }
        else
        {
            for (q = M.chead[j];(q->down)&&q->down->i<i;q=q->down);
            p->down = q->down;
            q->down = p;
        }
    }
    return M;
}
void display(CrossList M)
{
    for (int i = 1; i <= M.nu; i++)
    {
        if (NULL != M.chead[i])
        {
```

```
            OLink p = M.chead[i];
            while (NULL != p)
            {
                printf("%d  %d  %d\n", p->i, p->j, p->e);
                p = p->down;
            }
        }
    }
}
int main()
{
    CrossList M;
    M.rhead = NULL;
    M.chead = NULL;
    M = CreateMatrix_OL(M);
    printf("输出矩阵M:\n");
    display(M);
    return 0;
}
/*
 我也不知道这程序是怎么跑的？？？！！！

 输入矩阵的行数、列数和非0元素个数:
 3 3 3
 输入三元组:
 2 2 3
 2 3 4
 3 2 5
 0 0 0
 输出矩阵M:
 2  2  3
 3  2  5
 2  3  4
 */
```

数组·压缩存储·对称矩阵·三角矩阵

```
//
//  main.c
//  数组·压缩存储·对称矩阵·三角矩阵
//
//
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define r 6//列
#define c 6//行

int *Compress(int arr_before[][c])
{//压缩存储
    int *arr_after = (int *)malloc(sizeof(int)*((c*(c+1))/2));
    int row=1;//二维数组行标
    int a=0;//一维数组下标
    for(int m=0;m<r;m++)
    {
        for(int n=0;n<row;n++)
        {
            arr_after[a] = arr_before[m][n];
            a++;
        }
        row++;//下一行
    }
    return arr_after;//返回一维数组
}
int arr_show(int *arr_after,int m,int n)
{//压缩矩阵输出
    assert(m>=0&&m<r&&n>=0&&n<c);//断言，当条件不成立，直接退出函数
    if(m>=n)//下三角
        return arr_after[(m*(m+1))/2+n];
    else//上三角
        return arr_after[(n*(n+1))/2+m];
}
int *Compress1(int arr_before[][c])
{//压缩存储
    int *arr_after = (int *)malloc(sizeof(int)*((c*(c+1))/2));
    int row=1;//二维数组行标
    int a=0;//一维数组下标
    for(int m=0;m<r;m++)
    {
        for(int n=0;n<row;n++)
        {
            arr_after[a] = arr_before[m][n];
            a++;
        }
        row++;//下一行
```

```
    }
    arr_after[(c*(c+1))/2]=0;//上三角的值
    return arr_after;//返回一维数组
}
int arr_show1(int *arr_after,int m,int n)
{//压缩矩阵输出
    assert(m>=0&&m<r&&n>=0&&n<c);//断言，当条件不成立，直接退出函数
    if(m>=n)//下三角
        return arr_after[(m*(m+1))/2+n];
    else//上三角
        return arr_after[(c*(c+1))/2];
}

int main()
{
    int m;
    //三角矩阵
    int arr_before[r][c] = {
        {1, 0, 0, 0, 0, 0},
        {2, 2, 0, 0, 0, 0},
        {3, 3, 3, 0, 0, 0},
        {4, 4, 4, 3, 0, 0},
        {5, 4, 4, 3, 2, 0},
        {6, 5, 4, 3, 2, 1}
    };//上三角都为0
    int *arr_after=Compress1(arr_before);
    for(m=0;m<r;++m)
    {
        for(int n=0;n<c;n++)
            printf( "%d ",arr_show1(arr_after,m,n));
        printf("\n");
    }
    printf("\n");

    //对称矩阵
    int arr_before1[r][c] = {
        {1, 2, 3, 4, 5, 6},
        {2, 2, 3, 4, 4, 5},
        {3, 3, 3, 4, 4, 4},
        {4, 4, 4, 3, 3, 3},
        {5, 4, 4, 3, 2, 2},
        {6, 5, 4, 3, 2, 1}
    };
    int *arr_after1=Compress(arr_before1);
```

```
    for(m=0;m<r;++m)
    {
        for(int n=0;n<c;n++)
            printf( "%d ",arr_show(arr_after1,m,n));
        putchar('\n');
    }
    return 0;
}
/*
 1 2 3 4 5 6
 2 2 3 4 4 5
 3 3 3 4 4 4
 4 4 4 3 3 3
 5 4 4 3 2 2
 6 5 4 3 2 1
 */
```

实验报告·八皇后问题

```
//
//  main.c
//  实验报告·八皇后问题
//
//

#include <stdio.h>

int count = 0;
int queen[8][8]={0};

int find(int row,int col)
{
    int i,k;
    for(i=0;i<8;i++)  // 判断列方向
        if(queen[i][col]==1)
            return 0;
    for(i=row,k=col;i>=0&&k>=0;i--,k--)  // 判断左对角线(\)
        if(queen[i][k]==1)
            return 0;
    for(i=row,k=col;i>=0&&k<8;i--,k++ )  // 判断右对角线(/)
        if(queen[i][k]==1)
            return 0;
    return 1;
}

void show(void)  //打印结果
{
    int row,col;
    printf("第 %d 种\n", count+1);
    for(row=0;row<8;row++)
    {
        for(col=0;col<8;col++)
            printf("%d ",queen[row][col]);
        printf("\n");
    }
    printf("\n");
}

void Eight_Queen(int row)
{
    int col;
```

```
    if(row==8)    //如果遍历完八行都找到放置皇后的位置则打印
    {
        show();    //打印八皇后的解
        count++;
        return ;
    }
    for(col=0;col<8;col++)   //回溯，递归
    {
        if(find(row,col))    //判断是否危险
        {
            queen[row][col]=1;
            Eight_Queen(row+1);
            queen[row][col]=0;    //清零
        }
    }
}

int main()
{
    Eight_Queen(0);
    return 0;
}
//一共92种
```

实验报告·大数相乘

```
//
//  main.c
//  实验报告·大数相乘
//
//

#include <stdio.h>
#include <string.h>
#define MaxSize 100

void mult_largenum(char A[], char B[])
{
    int i,j,num;
    int x=0;
    int m=(int)strlen(A);//（int）为类型强制转化，strlen(A)本为 char型 强制转为 int型
    int n=(int)strlen(B);
    //char result[m+n+1];//在vc6上不给这样写
```

```
    char result[MaxSize]; //直接给个大树组
    for(i=0;i<m+n;i++)  //初始化
        result[i]='0';

    for(i=m-1;i>=0;i--)
    {
        for(j=n-1;j>=0;j--)
        {
            num=(A[j]-48)*(B[i]-48);
            if(num<10)  //相同位次相乘 <10 的
            {
                result[x]=result[x]+num;
                if(result[x]-48>9)  //本位达到 10，下一位进 1
                {
                    result[x+1]++;  //下一位 进位
                    result[x]=result[x]-10;  //处理 本位
                }
            }
            else    //相同位次相乘 >10 的
            {
                result[x]=num%10+result[x];
                if(result[x]-48>9)  //本位达到 10，下一位进 1
                {
                    result[x+1]++;  //下一位 进位
                    result[x]=result[x]-10;  //处理 本位
                }
                result[x+1]=num/10+result[x+1];
                if(result[x+1]-48>9)  //本位达到 10，下一位进 1
                {
                    result[x+2]++;  //下一位 进位
                    result[x+1]=result[x+1]-10;   //处理 本位
                }
            }
            x++;
        }
        x=x-n+1;  //返回到 上一周期位次 的 下一位
    }
    printf("最终结果为：\n");
    for(i=m+n-1;i>=0;i--)
        printf("%c",result[i]);
    printf("\n");
}

int main()
```

```
{
    char A[MaxSize]="99999999999999999999999999";
    char B[MaxSize]="99999999999999999999999999";
    /*
    printf("输入第一个乘数: ");
    scanf("%s",A);
    printf("输入第二个乘数: ");
    scanf("%s",B);
    */
    mult_largenum(A, B);
    return 0;
}
/*
输出:
 最终结果为:
 99999999999999999999999998000000000000000000000000001
A和B的位数最好要一样
不然会输出"烫烫烫烫"是因为vc会自动把未用函数的局部变量空间按字节全部初始化为CC
而定义的字符数组空间不够，所以就占用了那部分不属于自己的空间
因为输出char*是要遇到空字符才停止的，所以便把那些垃圾都打出来了
即是说在最后没有字符串结束标志'\0'
*/
```

实验报告·大一·结构体·成绩排名

```
//
//  main.c
//  实验报告·大一·结构体·成绩排名
//
//

#include <stdio.h>
#define num 5
struct student
{
    int number;
    char name[10];
    int math,Chinese,English;
    float average;
}stu[num];

void singlemax(struct student stu[])
{
```

```
    int i,m,c,e,j=0,k=0,l=0;
    m=stu[0].math;
    c=stu[0].Chinese;
    e=stu[0].English;
    for(i=0;i<num;i++)
    {
        if(m<stu[i].math)
        {
            j=i;
            m=stu[i].math;
        }
        if(e<stu[i].English)
        {
            k=i;
            e=stu[i].English;
        }
        if(c<stu[i].Chinese)
        {
            l=i;
            c=stu[i].Chinese;
        }
    }
    printf("\n数学成绩最高:%s  学号:%d  数学成
绩:%d\n",stu[j].name,stu[j].number,stu[j].math);
    printf("语文成绩最高:%s  学号:%d  语文成
绩:%d\n",stu[k].name,stu[k].number,stu[k].English);
    printf("英语成绩最高:%s  学号:%d  英语成
绩:%d\n",stu[l].name,stu[l].number,stu[l].Chinese);
}

void allmax(struct student stu[])
{
    int i,t=0;
    float a;
    a=stu[0].average;
    for(i=0;i<num;i++)
    {
        if(a<stu[i].average)
        {
            t=i;
            a=stu[i].average;
        }
    }
    printf("\n平均成绩最高:%s  学号:%d  平均成
```

```
绩:%.2lf\n\n",stu[t].name,stu[t].number,stu[t].average);
}

void sort(struct student stu[])
{
    struct student t;
    int i,j;
    float a;
    a=stu[0].average;
    for(i=0;i<num-1;i++)
    {
        for(j=i+1;j<num;j++)
        {
            if(stu[i].average<stu[j].average)
            {
                t=stu[i];
                stu[i]=stu[j];
                stu[j]=t;
            }
        }
    }
    printf("%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","Chinese","English","average");
    for(i=0;i<num;i++)
        printf("%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].number,stu[i].name,stu[i].math,stu[i].Chinese,stu[i].English,stu[i].aver
age);
}

int main()
{
    int i,j;
    for(i=0,j=1;i<num;i++,j++)
    {
        printf("输入第 %d 位学生学号: ",j);
        scanf("%d",&stu[i].number);
        printf("输入第 %d 位学生姓名: ",j);
        getchar();
        gets(stu[i].name);
        printf("输入第 %d 位学生 数学 语文 英语 的成绩: ",j);
        scanf("%d %d %d",&stu[i].math,&stu[i].Chinese,&stu[i].English);
        stu[i].average=(stu[i].math+stu[i].Chinese+stu[i].English)/3.0;
    }
    singlemax(stu);
```

```
    allmax(stu);
    sort(stu);
    return 0;
}
/*
 输入第 1 位学生学号: 1001
 输入第 1 位学生姓名: banban
 输入第 1 位学生 数学 语文 英语 的成绩: 120 121 122
 输入第 2 位学生学号: 1002
 输入第 2 位学生姓名: huahua
 输入第 2 位学生 数学 语文 英语 的成绩: 112 132 110
 输入第 3 位学生学号: 1003
 输入第 3 位学生姓名: yueyue
 输入第 3 位学生 数学 语文 英语 的成绩: 132 113 127
 输入第 4 位学生学号: 1004
 输入第 4 位学生姓名: caicai
 输入第 4 位学生 数学 语文 英语 的成绩: 131 125 132
 输入第 5 位学生学号: 1005
 输入第 5 位学生姓名: mei
 输入第 5 位学生 数学 语文 英语 的成绩: 142 137 141

 数学成绩最高:mei  学号:1005  数学成绩:142
 语文成绩最高:mei  学号:1005  语文成绩:141
 英语成绩最高:mei  学号:1005  英语成绩:137

 平均成绩最高:mei  学号:1005  平均成绩:140.00

 number  name  math  Chinese  English  average
 1005    mei   142   137      141      140.00
 1004    caicai131   125      132      129.33
 1003    yueyue132   113      127      124.00
 1001    banban120   121      122      121.00
 1002    huahua112   132      110      118.00
 */
```

实验报告·大一·文件·成绩排名

```
//
//  main.c
//  实验报告·大一·文件·成绩排名
//
//
```

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

typedef struct student {
    int num;
    int math;
    int English;
    int Chinese;
    double average;
    char name[20];
}Stu;

FILE *fp1,*fp2;
Stu stu[N];

void input(void)
{
    Stu stu[N];
    int i;
    printf("请依次输入10个同学的信息：学号 姓名 数学 英语 语文 成绩:\n");
    for(i=0;i<N;i++)
    {
        scanf("%d %s %d %d %d",
&stu[i].num,stu[i].name,&stu[i].math,&stu[i].English,&stu[i].Chinese);
        stu[i].average=(stu[i].math+stu[i].English+stu[i].Chinese)/3.0;
    }
    if((fp1=fopen("stud.txt","w"))==NULL)
    {
        printf("文件打开失败!\n");
        exit(0);
    }
    for(i=0;i<N;i++)
        if(fwrite(&stu[i],sizeof(struct student),1,fp1)!=1)
            printf("信息写入失败!\n");
    fprintf(fp1,"%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","English","Chinese","average");
    for(i=0;i<N;i++)
        fprintf(fp1,"%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].num,stu[i].name,stu[i].math,stu[i].English,stu[i].Chinese,stu[i].average
);
    fclose(fp1);
}
```

```
void output(void)
{
    int i;
    if((fp1=fopen("stud.txt","r"))==NULL)
    {
        printf("文件打开失败\n");
        exit(0);
    }
    for(i=0;i<N;i++)
        if(fread(&stu[i],sizeof(struct student),1,fp1)!=1)
        {
            if(feof(fp1))break;
                printf("FILE read error!\n");
        }
    fprintf(fp1,"排序前\n%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","English","Chinese","average");
    for(i=0;i<N;i++)
        fprintf(fp1,"%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].num,stu[i].name,stu[i].math,stu[i].English,stu[i].Chinese,stu[i].average
);
    fclose(fp1);

    printf("排序前\n%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","English","Chinese","average");
    for(i=0;i<N;i++)
        printf("%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].num,stu[i].name,stu[i].math,stu[i].English,stu[i].Chinese,stu[i].average
);
    printf("\n");
}

void sort(void)
{
    struct student t;
    int i,a,j;
    if((fp1=fopen("stud.txt","r")) == NULL)
    {
        printf("文件打开失败\n");
        exit(0);
    }
    for(i=0;i<N;i++)
    {
        if(fread(&stu[i],sizeof(struct student),1,fp1)!=1)
        {
```

```
            if(feof(fp1))break;
                printf("信息写入失败!\n");
        }
    }
    fclose(fp1);
    a=stu[0].average;
    for(i=0;i<N;i++)
    {
        for(j=i+1;j<N;j++)
        {
            if(stu[i].average<stu[j].average)
            {
                t=stu[i];
                stu[i]=stu[j];
                stu[j]=t;
            }
        }
    }
    if ((fp2=fopen("studsort.txt","w"))==NULL)
    {
        printf("文件打开失败\n");
        exit(0);
    }
    fprintf(fp2,"排序后\n%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","English","Chinese","average");
    for(i=0;i<N;i++)
        fprintf(fp2,"%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].num,stu[i].name,stu[i].math,stu[i].English,stu[i].Chinese,stu[i].average
);
    fclose(fp2);

    printf("排序后\n%-8s%-6s%-6s%-9s%-9s%-
9s\n","number","name","math","English","Chinese","average");
    for(i=0;i<N;i++)
        printf("%-8d%-6s%-6d%-9d%-9d%-
9.2lf\n",stu[i].num,stu[i].name,stu[i].math,stu[i].English,stu[i].Chinese,stu[i].average
);
    printf("\n");
}

int main()
{
    input();
    output();
```

```
    sort();

    return 0;
}

/*

101 a 89 95 92
102 b 96 87 78
103 c 70 98 93
104 d 79 86 78
105 e 93 90 78
106 f 80 90 79
107 g 88 75 83
108 h 81 91 73
109 i 92 79 87
110 j 91 82 80

 请依次输入10个同学的信息：学号 姓名 数学 英语 语文 成绩：
 101 a 89 95 92
 102 b 96 87 78
 103 c 70 98 93
 104 d 79 86 78
 105 e 93 90 78
 106 f 80 90 79
 107 g 88 75 83
 108 h 81 91 73
 109 i 92 79 87
 110 j 91 82 80
 排序前
 number  name  math  English  Chinese  average
 101     a     89    95       92       92.00
 102     b     96    87       78       87.00
 103     c     70    98       93       87.00
 104     d     79    86       78       81.00
 105     e     93    90       78       87.00
 106     f     80    90       79       83.00
 107     g     88    75       83       82.00
 108     h     81    91       73       81.67
 109     i     92    79       87       86.00
 110     j     91    82       80       84.33

 排序后
 number  name  math  English  Chinese  average
```

```
101    a    89    95    92    92.00
102    b    96    87    78    87.00
103    c    70    98    93    87.00
105    e    93    90    78    87.00
109    i    92    79    87    86.00
110    j    91    82    80    84.33
106    f    80    90    79    83.00
107    g    88    75    83    82.00
108    h    81    91    73    81.67
104    d    79    86    78    81.00
*/
```

实验报告·矩阵运算

```
//
//  main.c
//  实验报告·矩阵运算
//
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXSIZE 100

typedef struct
{
    int line,row;       //line为行,row为列
    int data[MAXSIZE];
}Matrix;

//矩阵初始化，赋值
Matrix initMatrix(void)
{
    Matrix *matrix=(Matrix *)malloc(sizeof(Matrix));
    printf("输入矩阵的行数和列数：");
    scanf("%d %d",&matrix->line,&matrix->row);
    printf("输入矩阵元素：");
    for(int i=0;i<matrix->line*matrix->row;i++)
        scanf("%d",&matrix->data[i]);
    return *matrix;
}
//打印矩阵
```

```
void PrintMatrix(Matrix *matrix)
{
    for(int i=0 ; i<matrix->line*matrix->row ; i++)
    {
        printf("%4d", matrix->data[i]);
        if((i+1)%matrix->row == 0)
            printf("\n");
    }
}
//矩阵加减法
Matrix jisuanMatrix(Matrix *matrix1,Matrix *matrix2,int muo)
{
    Matrix *matrix3=(Matrix *)malloc(sizeof(Matrix));
    matrix3->line=matrix1->line;
    matrix3->row=matrix1->row;

    for(int i=0 ; i < matrix1->line*matrix1->row ; i++)
    {
        if(muo==0)  //输0 相加
            matrix3->data[i]=matrix1->data[i]+matrix2->data[i];
        else       //输1 相减
            matrix3->data[i]=matrix1->data[i]-matrix2->data[i];
    }
    PrintMatrix(matrix3);
    return *matrix3;
}
//矩阵乘法
Matrix mulMatrix(Matrix *matrix1,Matrix *matrix2)
{
    Matrix *matrix3=(Matrix *)malloc(sizeof(Matrix));
    matrix3->line=matrix1->line;
    matrix3->row=matrix2->row;
    int m=0,n=0;  //m为数组2增量服务，n为数组1增量服务
    for(int i=0 ; i < matrix1->line*matrix2->row ; i++)  //大循环，i为新数组下标
    {
        matrix3->data[i]=0;  //在赋值新数组元素前，初始化每个元素为0
        int k=0;
        if(m == matrix2->row)  //每当数组2的一列乘完，m归零，数组1换下一行
        {
            m=0;
            n+=matrix1->row;
        }
        for(int j=0; j<matrix1->row ; j++)
        {
```

```
            matrix3->data[i]+=matrix1->data[j+n]*matrix2->data[k+m];
            k+=matrix2->row;  //矩阵2同列换行乘
        }
        m++;
    }
    PrintMatrix(matrix3);
    return *matrix3;
}
//矩阵置换
Matrix TransMatrix(Matrix *matrix)
{
    Matrix *matrix1=(Matrix *)malloc(sizeof(Matrix));
    matrix1->line=matrix->row;
    matrix1->row=matrix->line;
    int m=0,n=0,k=0;
    for(int i=0;i<matrix1->row*matrix1->line;i++)   //大循环，i为新数组下标
    {
        if(n==matrix->line)  //原矩阵的一列全部放置到矩阵1中的一行，执行
        {
            k=0;
            m++;
            n=0;
        }
        matrix1->data[i]=matrix->data[k+m];
        k+=matrix->row;
        n++;
    }
    PrintMatrix(matrix1);
    return *matrix1;
}


int main()
{
    Matrix matrix1 = initMatrix();
    Matrix matrix2 = initMatrix();

    printf("矩阵1为：\n");
    PrintMatrix(&matrix1);
    printf("矩阵2为：\n");
    PrintMatrix(&matrix2);

    if(matrix1.row==matrix2.row && matrix1.line==matrix2.line)
    {
```

```
        printf("矩阵一 和 矩阵二 相加为: \n");
        jisuanMatrix(&matrix1,&matrix2, 0);

        printf("矩阵一 和 矩阵二 相减为: \n");
        jisuanMatrix(&matrix1,&matrix2, 1);
    }
    else
        printf("两个矩阵的行数或列数不对应，无法计算加减！\n");
    if(matrix1.row==matrix2.line)
    {
        printf("矩阵一 和 矩阵二 相乘为: \n");
        mulMatrix(&matrix1, &matrix2);
    }
    else
        printf("两个矩阵的行数或列数不对应，无法计算乘法！\n");
    printf("矩阵一 的置换为: \n");
    TransMatrix(&matrix1);
    return 0;
}
/*

 3 3   1 2 3 4 5 6 7 8 9
 3 3   9 8 7 6 5 4 3 2 1

 3 4   1 2 3 4 5 6 7 8 9 10 11 12
 4 3   12 11 10 9 8 7 6 5 4 3 2 1

 输入矩阵的行数和列数: 3 3
 输入矩阵元素: 1 2 3 4 5 6 7 8 9
 输入矩阵的行数和列数: 3 3
 输入矩阵元素: 9 8 7 6 5 4 3 2 1
 矩阵1为:
    1   2   3
    4   5   6
    7   8   9
 矩阵2为:
    9   8   7
    6   5   4
    3   2   1
 矩阵一 和 矩阵二 相加为:
   10  10  10
   10  10  10
   10  10  10
 矩阵一 和 矩阵二 相减为:
```

```
  -8  -6  -4
  -2   0   2
   4   6   8
矩阵一 和 矩阵二 相乘为:
  30  24  18
  84  69  54
 138 114  90
矩阵一 的置换为:
   1   4   7
   2   5   8
   3   6   9

输入矩阵的行数和列数: 3 4
输入矩阵元素: 1 2 3 4 5 6 7 8 9 10 11 12
输入矩阵的行数和列数: 4 3
输入矩阵元素: 12 11 10 9 8 7 6 5 4 3 2 1
矩阵1为:
   1   2   3   4
   5   6   7   8
   9  10  11  12
矩阵2为:
  12  11  10
   9   8   7
   6   5   4
   3   2   1
矩阵一 和 矩阵二 相乘为:
  48  42  36
 129 114  99
 210 186 162
矩阵一 的置换为:
   1   5   9
   2   6  10
   3   7  11
   4   8  12
*/
```

实验报告·迷宫问题·队列

```
//
//  main.c
//  实验报告·迷宫问题·队列
//
//
```

```c
#include<stdio.h>
#include<stdlib.h>
#define m 6   /* 迷宫的实际行 */
#define n 8   /* 迷宫的实际列 */
int maze[m+2][n+2];
int maze_before[m][n]={
    {0 ,1 ,1 ,1 ,0 ,1 ,1 ,1},
    {0 ,0 ,0 ,0 ,1 ,1 ,1 ,1},
    {0 ,1 ,0 ,0 ,0 ,0 ,0 ,1},
    {0 ,1 ,1 ,1 ,0 ,0 ,1 ,1},
    {1 ,0 ,0 ,1 ,1 ,0 ,0 ,0},
    {0 ,1 ,1 ,0 ,0 ,1 ,1 ,0},
};
typedef struct
{
    int x,y;
}item;
item move1[4]={{0,1},{1,0},{0,-1},{-1,0}}; //方向，0右移，1是下移，2是左移，3是上移
//关于数据结构的问题
typedef struct
{
    int x,y,d;//横纵坐标及方向
}DataType ;

typedef struct Node
{
    DataType data;
    struct Node *next;
}QNode,*PQNode;//队列中结点的类型

typedef struct
{
    QNode *front;//队头指针
    QNode *rear;//队尾指针
}LinkQueue,*PLinkQueue;//链队列

//创建迷宫
void creat1(void)
{
    int i,j;
    for(i=0;i<=n+1;i++)
    {
        maze[0][i]=1;
```

```
        maze[m+1][i]=1;
    }  //i表示列，将迷宫的上下两行设为栅栏
    for(j=1;j<=m+1;j++)
    {
        maze[j][0]=1;
        maze[j][n+1]=1;
    }//j表示行，将迷宫的左右两类设为栅栏

    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
           // scanf("%d",&maze[i][j]);//将迷宫内的数值设置
            maze[i][j]=maze_before[i-1][j-1];
}
//输出迷宫
void printe(void)
{
    int i,j;
    for(i=0;i<=m+1;i++)
    {
        for(j=0;j<=n+1;j++)
            printf("%d  ",maze[i][j]);
        printf("\n");
    }
}
//初始化队列（未设头结点）
PLinkQueue Init_LinkQueue(void)
{
    PLinkQueue Q;
    Q=(PLinkQueue)malloc(sizeof(LinkQueue));
    if(Q)
    {
        Q->front=Q->rear=NULL;
    }
    return Q;
}
//判断队列是否为空，空返回1
int Empty_LinkQueue(PLinkQueue Q)
{
    if(Q&&Q->front==NULL&&Q->rear==NULL)
        return 1;
    else
        return 0;
}
//数据进队列，在队尾插入
```

```
int In_LinkQueue(PLinkQueue Q,DataType tmp)
{
    PQNode p;
    p=(PQNode)malloc(sizeof(QNode));
    if(!p)
    {
        printf("内存溢出\n");
        return 0;
    }
    p->data.x=tmp.x;
    p->data.y=tmp.y;
    p->data.d=tmp.d;
    //printf("[%d %d %d]\n",p->data.x,p->data.y,p->data.d); //分析行
    p->next=NULL;
    if(Empty_LinkQueue(Q))
        Q->rear=Q->front=p;
    else
    {
        Q->rear->next=p;
        Q->rear=p;
    }
    return 1;
}
//将队尾的结点删除，通过建立一个新的链表间接删除
void Out_LinkQueue(PLinkQueue Q,DataType *tmp)
{
    tmp->x=Q->rear->data.x;
    tmp->y=Q->rear->data.y;
    tmp->d=Q->rear->data.d;
    //printf("(%d %d %d)\n",tmp->x,tmp->y,tmp->d);  //分析行
    PLinkQueue s=Init_LinkQueue();//初始化一个新队列
    PQNode q;
    q=Q->front;
    if(q->next==NULL)//仅一个结点
    {
        Q->front=Q->rear;
        //printf("@"); //分析行
        free(q);
    }
    else
    {
        while(q->next->next!=NULL)//从循环中找到倒数第二个结点
        {//不只一个结点
            s->rear=q;//把收索到的结点放入新队列的的队尾
```

```
            q=q->next;
        }
        free(q->next);
        q->next=NULL;
        s->front=Q->front;
        s->rear->next=q;
        s->rear=q;
    }
    Q->front=s->front;
    Q->rear=s->rear;//调整结束之后，将队列恢复成t的新队列
}
void Out_LinkQueue1(PLinkQueue Q,DataType *tmp)
{//出队
    tmp->x=Q->front->data.x;
    tmp->y=Q->front->data.y;
    tmp->d=Q->front->data.d;
    PQNode p;
    p=Q->front;
    Q->front=Q->front->next;
    free(p);
    if(!Q->front)
        Q->rear=NULL;
}
//销毁队列
void Destroy_LinkQueue(PLinkQueue *Q)
{
    PQNode p;
    if(*Q)
    {
        while((*Q)->front)
        {
            p=(*Q)->front;
            (*Q)->front=(*Q)->front->next;
            free(p);
        }
        free(*Q);
    }
    *Q=NULL;
}
//寻找路径的过程
int path(int maze[][n+2],item move1[],int x0,int y0)//迷宫，移动方向的数组，起始的位置
{
    PLinkQueue S;
    DataType temp;
```

```
int x,y,d,i,j;
temp.x=x0;
temp.y=y0;
temp.d=-1;
S=Init_LinkQueue();
if(!S)
{
    printf("初始化失败\n");
    return 0;
}
In_LinkQueue(S,temp);
while(!Empty_LinkQueue(S))
{
    //printf("出1");  //分析行
    Out_LinkQueue(S,&temp);
    x=temp.x;
    y=temp.y;
    d=temp.d+1;
    while(d<4)
    {
        i=x+move1[d].x;
        j=y+move1[d].y;
        if(maze[i][j]==0)
        {
            temp.x=x;
            temp.y=y;
            temp.d=d;
            //printf("进");  //分析行
            In_LinkQueue(S,temp);
            x=i;y=j;maze[i][j]=-1;
            if(x==m&&y==n)
            {
                printf("迷宫路径为: ");
                while(!Empty_LinkQueue(S))
                {
                    Out_LinkQueue1(S,&temp);
                    printf("(%d,%d)->",temp.x,temp.y);
                }
                printf("(%d,%d)\n",m,n);
                Destroy_LinkQueue(&S);
                return 1;
            }
            else
                d=0;
```

```
            }
            else
                d++;
        }
    }
    Destroy_LinkQueue(&S);
    printf("未找到通路");
    return 0;
}
//主函数
int main()
{
    creat1();
    printf("所建造的迷宫为: \n");
    printe();
    printf("从起点(1,1)到终点(%d,%d)寻找路径的情况如下:\n",m,n);
    path(maze,move1,1,1);
    return 0;
}

/*

 所建造的迷宫为:
 1  1  1  1  1  1  1  1  1  1
 1  0  1  1  1  0  1  1  1  1
 1  0  0  0  0  1  1  1  1  1
 1  0  1  0  0  0  0  0  1  1
 1  0  1  1  1  0  0  1  1  1
 1  1  0  0  1  1  0  0  0  1
 1  0  1  1  0  0  1  1  0  1
 1  1  1  1  1  1  1  1  1  1
 从起点(1,1)到终点(6,8)寻找路径的情况如下:
 迷宫路径为:
(1,1)->(2,1)->(2,2)->(2,3)->(2,4)->(3,4)->(3,5)->(3,6)->(4,6)->(5,6)->(5,7)->(5,8)->(6,8
)
 */
```

实验报告·迷宫问题·栈

```
//
//  main.c
//  实验报告·迷宫问题·栈
//
```

```
//

#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 200
#define Row 10  //行
#define Col 10  //列

typedef struct Position
{
    int x;
    int y;
}Position;

typedef struct
{
    Position arry[MAXSIZE];
    int top;
}seqstack,*pseqstack;

typedef struct
{
    int map[Row][Col];
}Maze,*PMaze;

pseqstack start(void)
{
    pseqstack s;
    s=(pseqstack)malloc(sizeof(seqstack));
    if(s)
        s->top=-1;
    return s;
}

int empty(pseqstack s)
{
    if(s->top==-1)
        return 1;
    else
        return 0;
}

int push(pseqstack s,Position x)
{
```

```
    if(s->top==MAXSIZE-1)
        return 0;
    else
    {
        s->top++;
        s->arry[s->top]=x;
        return 1;
    }
}

void pop(pseqstack s)
{
    if(empty(s))
        return;
    else
        s->top--;
}

Position gettop(pseqstack s)
{
    if(empty(s))
        exit(0);
    else
        return s->arry[s->top];
}
void print_Maze(PMaze m)  //打印迷宫
{
    for(int i=0;i<Row;i++)
    {
        for(int j=0;j<Col;j++)
            printf("%d ",m->map[i][j]);
        printf("\n");
    }
    printf("\n");
}
void init_Maze(PMaze m,int map[Row][Col])  //初始化迷宫
{
    for(int i=0;i<Row;i++)
        for(int j=0;j<Col;j++)
            m->map[i][j]=map[i][j];
}
int Judge_Enter(Position en)  //判断入口是否合法
{
    if((en.x>=0 || en.x<Row) && (en.y>=0 || en.y<Col))
```

```
        return 1;
    return 0;
}
int Judge_Exit(Position set,Position en)  //判断是否为出口
{
    if((set.x==0||set.y==0||set.x==Col-1||set.y==Row-1)&&(set.x!=en.x||set.y!=en.y))
        return 1;
    return 0;
}
int Judge_Pase(PMaze m,Position set)  //判断是否可走
{
    if(m->map[set.x][set.y]==1)
        return 1;
    return 0;
}
void Tranlate(PMaze m)
{
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            if(m->map[i][j]==1||m->map[i][j]==3)
                m->map[i][j]=0;
            if(m->map[i][j]==2)
                m->map[i][j]=1;
        }
    }
}
void Start_Maze(PMaze m,Position en)
{
    pseqstack s=start();
    push(s, en);
    while (!empty(s))
    {
        Position set=gettop(s);
        Position next;
        m->map[set.x][set.y]=2;   //当前位置变为2
        //print_Maze(m);   //展示每一步
        if(Judge_Exit(set, en))
        {
            Tranlate(m);
            print_Maze(m);      //打印迷宫
            return;
        }
```

```
        else
        {
            next=set;
            next.x-=1;   //上
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            next=set;
            next.y-=1;   //左
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            next=set;
            next.y+=1;   //右
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            next=set;
            next.x+=1;   //下
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            m->map[set.x][set.y]=3;
            pop(s);  //出栈
        }
    }
    printf("迷宫不存在通路\n");
}
int main()
{
    PMaze m=(PMaze)malloc(sizeof(Maze));
    Position en;

    int arry_maze[Row][Col]={
    { 0, 1, 0, 0, 0, 0 ,0 ,0 ,0 ,0 },
    { 0, 1, 1, 0, 1, 0 ,0 ,0 ,1 ,0 },
```

```
    { 0, 0, 1, 0, 1, 1 ,1 ,1 ,1 ,0 },
    { 0, 0, 1, 1, 1, 0 ,0 ,0 ,1 ,0 },
    { 0, 0, 1, 0, 1, 1 ,0 ,1 ,1 ,0 },
    { 0, 1, 1, 0, 0, 1 ,0 ,0 ,0 ,0 },
    { 0, 0, 1, 0, 0, 1 ,1 ,1 ,0 ,0 },
    { 0, 0, 1, 0, 0, 0 ,0 ,1 ,0 ,0 },
    { 0, 0, 1, 0, 0, 0 ,1 ,1 ,0 ,0 },
    { 0, 0, 1, 0, 0, 0 ,1 ,0 ,0 ,0 }
    };
    int arry_maze1[Row][Col]={
    { 0, 1, 0, 0, 0, 0 ,0 ,0 ,0 ,0 },
    { 0, 1, 1, 0, 1, 0 ,0 ,0 ,1 ,0 },
    { 0, 0, 1, 0, 1, 1 ,1 ,1 ,1 ,0 },
    { 0, 0, 1, 1, 1, 0 ,0 ,0 ,1 ,0 },
    { 0, 0, 1, 0, 1, 1 ,0 ,1 ,1 ,0 },
    { 0, 1, 1, 0, 0, 1 ,0 ,0 ,0 ,0 },
    { 0, 0, 1, 0, 0, 1 ,1 ,1 ,0 ,0 },
    { 0, 0, 1, 0, 0, 0 ,0 ,1 ,0 ,0 },
    { 0, 0, 1, 0, 0, 0 ,1 ,1 ,0 ,0 },
    { 0, 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 }
    };

    int a[3]={9,9,0};
    int b[3]={2,6,1};
    printf("迷宫图一：\n");
    for(int i=0;i<3;i++)
    {
        init_Maze(m, arry_maze);
        if(i==0) print_Maze(m);
        en.x=a[i];
        en.y=b[i];
        printf("解迷宫一%d：入口：(%d,%d)\n",i+1,en.y,en.x);
        Start_Maze(m, en);
    }
    printf("迷宫图二：\n");
    init_Maze(m, arry_maze1);
    print_Maze(m);
    en.x=0;
    en.y=1;
    printf("解迷宫二：入口：(%d,%d)\n",en.y,en.x);
    Start_Maze(m, en);
    return 0;
}
```

```
/*
 迷宫图一:
 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 1 0 0 0 1 0
 0 0 1 0 1 1 1 1 1 0
 0 0 1 1 1 0 0 0 1 0
 0 0 1 0 1 1 0 1 1 0
 0 1 1 0 0 1 0 0 0 0
 0 0 1 0 0 1 1 1 0 0
 0 0 1 0 0 0 0 1 0 0
 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 0 0 1 0 0 0

 解迷宫一1: 入口: (2,9)
 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0

 解迷宫一2: 入口: (6,9)
 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 1 1 0 0 0 0 0
 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 1 1 1 0 0
 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 1 0 0
 0 0 0 0 0 0 1 0 0 0

 解迷宫一3: 入口: (1,0)
 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 1 1 1 0 0 0 0 0
 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 1 1 1 0 0
 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 1 0 0
 0 0 0 0 0 0 1 0 0 0

 迷宫图二:
 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 1 0 0 0 1 0
 0 0 1 0 1 1 1 1 1 0
 0 0 1 1 1 0 0 0 1 0
 0 0 1 0 1 1 0 1 1 0
 0 1 1 0 0 1 0 0 0 0
 0 0 1 0 0 1 1 1 0 0
 0 0 1 0 0 0 0 1 0 0
 0 0 1 0 0 0 1 1 0 0
 0 0 0 0 0 0 0 0 0 0

 解迷宫二: 入口: (1,0)
 迷宫不存在通路
 */
```

实验报告·一元多项式·文件

```
//
//  main.c
//  实验报告·一元多项式·文件
//
//

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

```c
typedef struct Polyn
{
    int coeff;   //系数
    int power;   //幂次
    struct Polyn *next;
}Polyn,*Polynomial;

FILE *fp,*fp1;

int a[10];
int b[10];
int f;

int load(int t)//从文件读取数据
{
    FILE *fp;
    int i;
    int f;
    if(t==0)
    {
        if((fp=fopen("stud11.txt","rb"))==NULL)
        {
            printf("cannot open file\n");
            exit(0);
        }
        f=7;
    }
    else
    {
        if((fp=fopen("stud22.txt","rb"))==NULL)
        {
            printf("cannot open file\n");
            exit(0);
        }
        f=8;
    }
    for(i=0;i<f;i++)
    {
        fscanf(fp,"%d ",&a[i]);
        //printf("%d ",a[i]);   //分析行
    }
    rewind(fp);//文件指针返回到开头
    fscanf(fp,"%*[^\n]%*c");//换到下一行
    printf("\n");
```

```
    for(i=0;i<f;i++)
    {
        fscanf(fp,"%d ",&b[i]);
        //printf("%d ",b[i]);   //分析行
    }
    /*****************************************************
        %*       :    是“跳过”
        [^\n]          :    字符串的分隔符是"\n", 中括号里可以写 分隔符 表
        %*[^\n]        :    跳过 \n 前的所有字符串。
        %*c            :    是“跳过”行尾 的 换行符
    *****************************************************/
    fclose(fp);
    return f;
}

Polynomial InitPolyn1(int f)
{
    //int a[4]={7,3,9,5};   //数据组
    //int b[4]={0,1,8,17};
    //int a[4]={1,-1,1,2};
    //int b[4]={0,1,2,3};
    Polynomial p=(Polynomial)malloc(sizeof(Polyn));
    Polynomial q=p;  //保存头结点
    //printf("输入多项式的系数和对应幂次: \n");
    for(int i=0;i<4;i++)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        p->coeff=a[i];
        p->power=b[i];
    }
    p->next=NULL;
    return q->next;  //直接返回一个指向第一个值的指针
}
Polynomial InitPolyn2(int f)
{
    //int a[3]={8,22,-9};  //数据组
    //int b[3]={1,7,8};
    //int a[3]={-1,1,-1};
    //int b[3]={0,1,3};
    Polynomial p=(Polynomial)malloc(sizeof(Polyn));
    Polynomial q=p;  //保存头结点
    //printf("输入多项式的系数和对应幂次: \n");
    //printf("<%d>",f);
```

```
    for(int i=4;i<f;i++)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        p->coeff=a[i];
        p->power=b[i];
    }
    p->next=NULL;
    return q->next;  //直接返回一个指向第一个值的指针
}

void PrintPolyn1(Polynomial p)
{//输出1看的更加清晰，2可能行数会少一些
    int count=0;  //判断是否为第一个
    while (p)
    {
        if(p->power==0)
        {  //幂次为0
            printf("%d ",p->coeff);  //零次幂第一次
            count++;
            p=p->next;
        }
        if(p->power==1)
        {  //幂次为1
            if(p->coeff == 1)
            {
                if(count==0)
                {
                    printf("x ");  //一次幂第一次
                    count++;
                }
                else
                    printf("+x ");
            }
            else if (p->coeff == -1)
                printf("-x ");
            else
            {
                if(count==0)
                {
                    printf("%dx ",p->coeff);  //一次幂第一次
                    count++;
                }
                else
```

```
                printf("%+dx ",p->coeff);
            }
            p = p->next;
        }
        else
        {  //更高幂次
            if(p->coeff == 1)
            {
                if(count==0)
                {
                    printf("x^%d ",p->power);  //高次幂第一次
                    count++;
                }
                else
                    printf("+x^%d ",p->power);
            }
            else if (p->coeff == -1)
                printf("-x^%d ",p->power);
            else
            {
                if(count==0)
                {
                    printf("%dx^%d ",p->coeff,p->power);  //高次幂第一次
                    count++;
                }else
                    printf("%+dx^%d ",p->coeff,p->power);
            }
            p = p->next;
        }
    }
    printf("\n");
}
Polynomial copyPolyn(Polynomial p2)
{
    Polynomial p1=(Polynomial)malloc(sizeof(Polyn));
    Polynomial p=p1;
    while (p2)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        p->coeff=p2->coeff;
        p->power=p2->power;
        p2=p2->next;
    }
```

```
    p->next=NULL;
    //PrintPolyn1(p1->next);   //分析行
    return p1->next;  //直接返回一个指向第一个值的指针
}

void PrintPolyn2(Polynomial p)
{//输出和1一样，只是化简一些
    int count=0;  //判断是否为第一个
    while (p)
    {
        if(p->power==0)  //幂次为0
        {
            printf("%d ",p->coeff);  //零次幂 第一次
            count++;
            p=p->next;
        }
        if(p->power==1)  //幂次为1
        {
            if(p->coeff == 1)
            {
                if(count==0) {printf("x ");count++;}  //一次幂 第一次
                else         printf("+x ");
            }
            else if (p->coeff == -1) printf("-x ");
            else
            {
                if(count==0) {printf("%dx ",p->coeff);count++;}  //一次幂 第一次
                else         printf("%+dx ",p->coeff);
            }
            p = p->next;
        }
        else           //更高幂次
        {
            if(p->coeff == 1)
            {
                if(count==0) {printf("x^%d ",p->power);count++;}  //高次幂 第一次
                else         printf("+x^%d ",p->power);
            }
            else if (p->coeff == -1) printf("-x^%d ",p->power);
            else
            {
                if(count==0) {printf("%dx^%d ",p->coeff,p->power);count++;}  //高次幂 第一次
                else         printf("%+dx^%d ",p->coeff,p->power);
            }
```

```
            p = p->next;
        }
    }
    printf("\n");
}
//加法
Polynomial AddPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p3=(Polynomial)malloc(sizeof(Polyn));
    Polynomial p=p3;  //保存头结点
    while (p1 && p2)
    {
        if(p1->power < p2->power)
        {
            p3->next=(Polynomial)malloc(sizeof(Polyn));
            p3=p3->next;
            p3->coeff=p1->coeff;
            p3->power=p1->power;
            p1=p1->next;
        }
        else if(p1->power > p2->power)
        {
            p3->next=(Polynomial)malloc(sizeof(Polyn));
            p3=p3->next;
            p3->coeff=p2->coeff;
            p3->power=p2->power;
            p2=p2->next;
        }
        else
        {
            if(p1->coeff+p2->coeff!=0)
            {
                p3->next=(Polynomial)malloc(sizeof(Polyn));
                p3=p3->next;
                p3->coeff=p1->coeff+p2->coeff;
                p3->power=p1->power;
            }
            p1=p1->next;
            p2=p2->next;
        }
    }
    while(p1)
    {
        p3->next=(Polynomial)malloc(sizeof(Polyn));
```

```
        p3=p3->next;
        p3->coeff=p1->coeff;
        p3->power=p1->power;
        p1=p1->next;
    }
    while(p2)
    {
        p3->next=(Polynomial)malloc(sizeof(Polyn));
        p3=p3->next;
        p3->coeff=p2->coeff;
        p3->power=p2->power;
        p2=p2->next;
    }
    p3->next=NULL;
    //PrintPolyn1(p->next);  //分析行
    return p->next;  //直接返回一个指向第一个值的指针
}
//减法
Polynomial SubPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p=p2,q=NULL;
    while (p)
    {
        p->coeff=-(p->coeff);  //变负
        p=p->next;
    }
    q=AddPolyn(p1, p2);
    //PrintPolyn1(p);   //分析行
    return q;  //直接返回一个指向第一个值的指针
}
//乘法
Polynomial MulPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p3=(Polynomial)malloc(sizeof(Polyn));  //结果域
    Polynomial p4=(Polynomial)malloc(sizeof(Polyn));  //辅助域
    Polynomial p=p1;   //保存头结点
    Polynomial pn=p4;  //保存头结点
    int i=0;
    while(p2)
    {
        pn=p4;  //p4重来
        p1=p;  //p1重来
        while(p1)
        {
```

```
            if(i==0)
                pn->next=(Polynomial)malloc(sizeof(Polyn));  //运行一趟
            //printf("(%d %d)\n",pn->coeff,pn->power);  //分析行
            pn=pn->next;
            pn->coeff=p1->coeff*p2->coeff;  //系数
            pn->power=p1->power+p2->power;  //幂次
            p1=p1->next;
            //printf("[%d %d]\n",pn->coeff,pn->power);  //分析行
        }
        pn->next=NULL;
        //printf("/");
        //PrintPolyn1(pn); //分析行
        //printf("//");
        if(i==0)
            p3=copyPolyn(p4->next);  //拷贝，p3变为指向第一个值的指针，运行一趟
        else
        {
            //PrintPolyn1(pn);  //分析行
            p3=AddPolyn(p3, p4->next);
        }
        //printf("///");
        p2=p2->next;  //p2向后移一位
        i++;
    }
    return p3;   //直接返回一个指向第一个值的指针
}

int main()
{
    for(int i=0;i<2;i++)
    {
        f=load(i);
        Polynomial p1=InitPolyn1(f);  //p1指向第一个值的指针，分析代码运行时可以使用这行
        Polynomial p2=InitPolyn2(f);  //p2指向第一个值的指针，分析代码运行时可以使用这行
        Polynomial p3 = NULL;  //辅助结点

        printf("A(x)=");
        PrintPolyn1(p1);  //打印从指向第一个值的指针打印

        printf("B(x)=");
        PrintPolyn1(p2);

        printf("1.多项式相加\n");
        printf("C(x)=");
```

```
        p3=AddPolyn(p1,p2);
        PrintPolyn1(p3);

        printf("2.多项式相减\n");
        printf("D(x)=");
        p3=copyPolyn(p2);  //拷贝一份，防止被覆盖掉。拷贝从指向第一个值的指针开始拷贝
        //PrintPolyn1(p3);    //分析行
        p3=SubPolyn(p1,p3);
        PrintPolyn1(p3);

        printf("3.多项式相乘\n");
        printf("E(x)=");
        p3=MulPolyn(p1,p2);
        PrintPolyn1(p3);
    }

    return 0;
}
/*

 A(x)=1 -x +x^2 +2x^3
 B(x)=-1 +x -x^3
 1.多项式相加
 C(x)=x^2 +x^3
 2.多项式相减
 D(x)=2 -2x +x^2 +3x^3
 3.多项式相乘
 E(x)=-1 +2x -2x^2 -2x^3 +3x^4 -x^5 -2x^6

 A(x)=7 +3x +9x^8 +5x^17
 B(x)=8x +2x^4 +11x^7 -9x^8
 1.多项式相加
 C(x)=7 +11x +2x^4 +11x^7 +5x^17
 2.多项式相减
 D(x)=7 -5x -2x^4 -11x^7 +18x^8 +5x^17
 3.多项式相乘
 E(x)=56x +24x^2 +14x^4 +6x^5 +77x^7 -30x^8 +45x^9 +18x^12 +99x^15 -81x^16 +40x^18
+10x^21 +55x^24 -45x^25

 */
```

实验报告·一元多项式计算

banban https://github.com/dream4789/Computer-learning-resources.git

```c
//
//  main.c
//  实验报告·一元多项式计算
//
//

#include <stdio.h>
#include <stdlib.h>
typedef struct Polyn
{
    int coeff;   //系数
    int power;   //幂次
    struct Polyn *next;
}Polyn,*Polynomial;

Polynomial InitPolyn(void)
{
    int num;
    Polynomial p=(Polynomial)malloc(sizeof(Polyn));
    Polynomial q=p;  //保存头结点
    printf("输入多项式的项数: ");
    scanf("%d",&num);
    printf("输入多项式的系数和对应幂次: \n");
    for(int i=0;i<num;i++)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        scanf("%d %d",&p->coeff,&p->power);
    }
    p->next=NULL;
    return q->next;  //直接返回一个指向第一个值的指针
}

Polynomial InitPolyn1(void)
{
    //int a[4]={7,3,9,5};   //数据组1
    //int b[4]={0,1,8,17}; //数据组2
    int a[4]={1,-1,1,2};
    int b[4]={0,1,2,3};
    Polynomial p=(Polynomial)malloc(sizeof(Polyn));
    Polynomial q=p;  //保存头结点
    for(int i=0;i<4;i++)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
```

```
        p=p->next;
        p->coeff=a[i];
        p->power=b[i];
    }
    p->next=NULL;
    return q->next;  //直接返回一个指向第一个值的指针
}
Polynomial InitPolyn2(void)
{
    //int a[3]={8,22,-9};  //数据组1
    //int b[3]={1,7,8};
    int a[3]={-1,1,-1}; //数据组2
    int b[3]={0,1,3};
    Polynomial p=(Polynomial)malloc(sizeof(Polyn));
    Polynomial q=p;  //保存头结点
    for(int i=0;i<3;i++)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        p->coeff=a[i];
        p->power=b[i];
    }
    p->next=NULL;
    return q->next;  //直接返回一个指向第一个值的指针
}

void PrintPolyn1(Polynomial p)
{//输出1看的更加清晰，2可能行数会少一些
    int count=0;  //判断是否为第一个
    while (p)
    {
        if(p->power==0)
        {  //幂次为0
            printf("%d ",p->coeff);  //零次幂第一次
            count++;
            p=p->next;
        }
        if(p->power==1)
        {  //幂次为1
            if(p->coeff == 1)
            {
                if(count==0)
                {
                    printf("x ");  //一次幂第一次
```

```
                count++;
            }
            else
                printf("+x ");
        }
        else if (p->coeff == -1)
            printf("-x ");
        else
        {
            if(count==0)
            {
                printf("%dx ",p->coeff);  //一次幂第一次
                count++;
            }
            else
                printf("%+dx ",p->coeff);
        }
        p = p->next;
    }
    else
    {  //更高幂次
        if(p->coeff == 1)
        {
            if(count==0)
            {
                printf("x^%d ",p->power);  //高次幂第一次
                count++;
            }
            else
                printf("+x^%d ",p->power);
        }
        else if (p->coeff == -1)
            printf("-x^%d ",p->power);
        else
        {
            if(count==0)
            {
                printf("%dx^%d ",p->coeff,p->power);  //高次幂第一次
                count++;
            }else
                printf("%+dx^%d ",p->coeff,p->power);
        }
        p = p->next;
    }
```

```
    }
    printf("\n");
}
Polynomial copyPolyn(Polynomial p2)
{
    Polynomial p1=(Polynomial)malloc(sizeof(Polyn));
    Polynomial p=p1;
    while (p2)
    {
        p->next=(Polynomial)malloc(sizeof(Polyn));
        p=p->next;
        p->coeff=p2->coeff;
        p->power=p2->power;
        p2=p2->next;
    }
    p->next=NULL;
    //PrintPolyn1(p1->next);   //分析行
    return p1->next;  //直接返回一个指向第一个值的指针
}

void PrintPolyn2(Polynomial p)
{//输出和1一样，只是化简一些
    int count=0;  //判断是否为第一个
    while (p)
    {
        if(p->power==0)  //幂次为0
        {
            printf("%d ",p->coeff);  //零次幂 第一次
            count++;
            p=p->next;
        }
        if(p->power==1)  //幂次为1
        {
            if(p->coeff == 1)
            {
                if(count==0) {printf("x ");count++;}  //一次幂 第一次
                else         printf("+x ");
            }
            else if (p->coeff == -1) printf("-x ");
            else
            {
                if(count==0) {printf("%dx ",p->coeff);count++;}  //一次幂 第一次
                else         printf("%+dx ",p->coeff);
            }
```

```
            p = p->next;
        }
        else           //更高幂次
        {
            if(p->coeff == 1)
            {
                if(count==0) {printf("x^%d ",p->power);count++;}  //高次幂 第一次
                else         printf("+x^%d ",p->power);
            }
            else if (p->coeff == -1) printf("-x^%d ",p->power);
            else
            {
                if(count==0) {printf("%dx^%d ",p->coeff,p->power);count++;}  //高次幂 第一次
                else         printf("%+dx^%d ",p->coeff,p->power);
            }
            p = p->next;
        }
    }
    printf("\n");
}
//加法
Polynomial AddPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p3=(Polynomial)malloc(sizeof(Polyn));
    Polynomial p=p3;  //保存头结点
    while (p1 && p2)
    {
        if(p1->power < p2->power)
        {
            p3->next=(Polynomial)malloc(sizeof(Polyn));
            p3=p3->next;
            p3->coeff=p1->coeff;
            p3->power=p1->power;
            p1=p1->next;
        }
        else if(p1->power > p2->power)
        {
            p3->next=(Polynomial)malloc(sizeof(Polyn));
            p3=p3->next;
            p3->coeff=p2->coeff;
            p3->power=p2->power;
            p2=p2->next;
        }
        else
```

```
        {
            if(p1->coeff+p2->coeff!=0)
            {
                p3->next=(Polynomial)malloc(sizeof(Polyn));
                p3=p3->next;
                p3->coeff=p1->coeff+p2->coeff;
                p3->power=p1->power;
            }
            p1=p1->next;
            p2=p2->next;
        }
    }
    while(p1)
    {
        p3->next=(Polynomial)malloc(sizeof(Polyn));
        p3=p3->next;
        p3->coeff=p1->coeff;
        p3->power=p1->power;
        p1=p1->next;
    }
    while(p2)
    {
        p3->next=(Polynomial)malloc(sizeof(Polyn));
        p3=p3->next;
        p3->coeff=p2->coeff;
        p3->power=p2->power;
        p2=p2->next;
    }
    p3->next=NULL;
    //PrintPolyn1(p->next);  //分析行
    return p->next;  //直接返回一个指向第一个值的指针
}
//减法
Polynomial SubPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p=p2,q=NULL;
    while (p)
    {
        p->coeff=-(p->coeff);  //变负
        p=p->next;
    }
    q=AddPolyn(p1, p2);
    //PrintPolyn1(p);   //分析行
    return q;  //直接返回一个指向第一个值的指针
```

```
}
//乘法
Polynomial MulPolyn(Polynomial p1,Polynomial p2)
{
    Polynomial p3=(Polynomial)malloc(sizeof(Polyn));  //结果域
    Polynomial p4=(Polynomial)malloc(sizeof(Polyn));  //辅助域
    Polynomial p=p1;   //保存头结点
    Polynomial pn=p4;  //保存头结点
    int i=0;
    while(p2)
    {
        pn=p4;  //p4重来
        p1=p;  //p1重来
        while(p1)
        {
            if(i==0)
                pn->next=(Polynomial)malloc(sizeof(Polyn));  //运行一趟
            //printf("(%d %d)\n",pn->coeff,pn->power);  //分析行
            pn=pn->next;
            pn->coeff=p1->coeff*p2->coeff;  //系数
            pn->power=p1->power+p2->power;  //幂次
            p1=p1->next;
            //printf("[%d %d]\n",pn->coeff,pn->power);  //分析行
        }
        pn->next=NULL;
        //printf("/");
        //PrintPolyn1(pn);
        //printf("//");
        if(i==0)
            p3=copyPolyn(p4->next);  //拷贝，p3变为指向第一个值的指针，运行一趟
        else
        {
            //PrintPolyn1(pn);  //分析行
            p3=AddPolyn(p3, p4->next);
        }
        //printf("///");
        p2=p2->next;  //p2向后移一位
        i++;
    }
    return p3;   //直接返回一个指向第一个值的指针
}

int main()
{
```

```
    printf("按升幂输入多项式A(x),B(x)的系数\n");
    //Polynomial p1=InitPolyn();  //p1指向第一个值的指针
    //Polynomial p2=InitPolyn();  //p2指向第一个值的指针

    Polynomial p1=InitPolyn1();  //p1指向第一个值的指针，简化输入
    Polynomial p2=InitPolyn2();  //p2指向第一个值的指针，简化输入
    Polynomial p3 = NULL;  //辅助结点

    printf("A(x)=");
    PrintPolyn1(p1);  //打印从指向第一个值的指针打印

    printf("B(x)=");
    PrintPolyn1(p2);

    printf("1.多项式相加\n");
    printf("C(x)=");
    p3=AddPolyn(p1,p2);
    PrintPolyn1(p3);

    printf("2.多项式相减\n");
    printf("D(x)=");
    p3=copyPolyn(p2);  //拷贝一份，防止被覆盖掉。拷贝从指向第一个值的指针开始拷贝
    //PrintPolyn1(p3);    //分析行
    p3=SubPolyn(p1,p3);
    PrintPolyn1(p3);

    printf("3.多项式相乘\n");
    printf("E(x)=");
    p3=MulPolyn(p1,p2);
    PrintPolyn1(p3);

    return 0;
}
/*
数据1:
4
1 0
-1 1
1 2
2 3

3
-1 0
1 1
```

```
-1 3
```
数据2:
```
4
7 0
3 1
9 8
5 17

3
8 1
22 7
-9 8
```
以下是输出:

```
按升幂输入多项式A(x),B(x)的系数
输入多项式的项数: 4
输入多项式的系数和对应幂次:
1 0
-1 1
1 2
2 3
输入多项式的项数: 3
输入多项式的系数和对应幂次:
-1 0
1 1
-1 3
A(x)=1 -x +x^2 +2x^3
B(x)=-1 +x -x^3
1.多项式相加
C(x)=x^2 +x^3
2.多项式相减
D(x)=2 -2x +x^2 +3x^3
3.多项式相乘
E(x)=-1 +2x -2x^2 -2x^3 +3x^4 -x^5 -2x^6

按升幂输入多项式A(x),B(x)的系数
输入多项式的项数: 4
输入多项式的系数和对应幂次:
7 0
3 1
9 8
5 17
输入多项式的项数: 3
```

```
输入多项式的系数和对应幂次:
8 1
22 7
-9 8
A(x)=7 +3x +9x^8 +5x^17
B(x)=8x +22x^7 -9x^8
1.多项式相加
C(x)=7 +11x +22x^7 +5x^17
2.多项式相减
D(x)=7 -5x -22x^7 +18x^8 +5x^17
3.多项式相乘
E(x)=56x +24x^2 +154x^7 +3x^8 +45x^9 +198x^15 -81x^16 +40x^18 +110x^24 -45x^25

*/
```