数据结构课程设计报告

学　　院：计算机科学与技术学院

专　　业：计算机科学与技术

姓　　名：

学　　号：

指导教师：

2021 年 12 月 22 号

一、题目一

一元多项式计算

二、算法及数据结构

通过输入两个一元多项式系数和幂次，对这两个式子进行加法减法乘法的计算，并输出结果。采用结构体、结构体指针，用链表进行存储。对于相乘运算之后产生的不同的幂次项，新开辟一个新幂次的结点，插入链表并存储，对相同幂次项相加相减，若系数加减后为 0，则把那个幂次的结点删除。

数据结构定义:

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

主要算法:

```
//相加算法
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;
        return p->next; //直接返回一个指向第一个值的指针
    }
```

//相减算法

```
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);
    return q;  //直接返回一个指向第一个值的指针
}
```

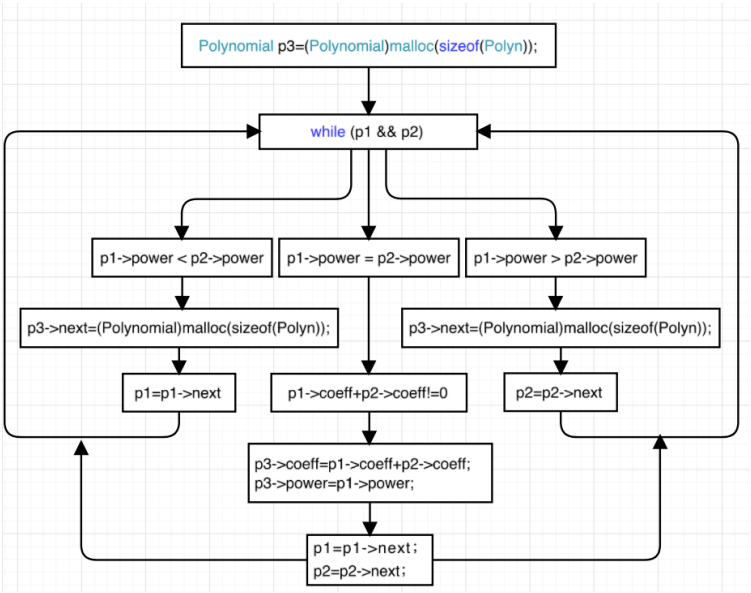
//相乘算法

```
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));//运一趟
                pn=pn->next;
                pn->coeff=p1->coeff*p2->coeff;  //系数
                pn->power=p1->power+p2->power;  //幂次
                p1=p1->next;
            }
            pn->next=NULL;
            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
                p3=AddPolyn(p3, p4->next);
            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
    }
```

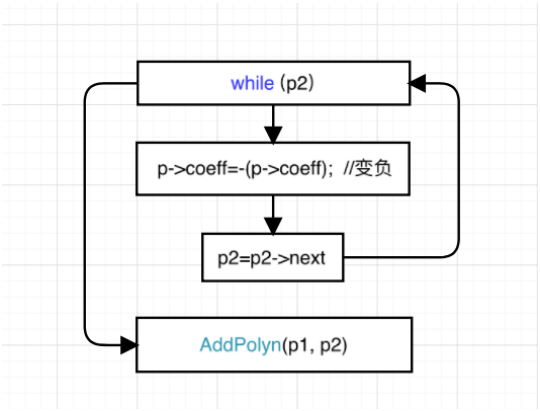
三、思路图

根据需求分析，可以把这个系统的设计分为：

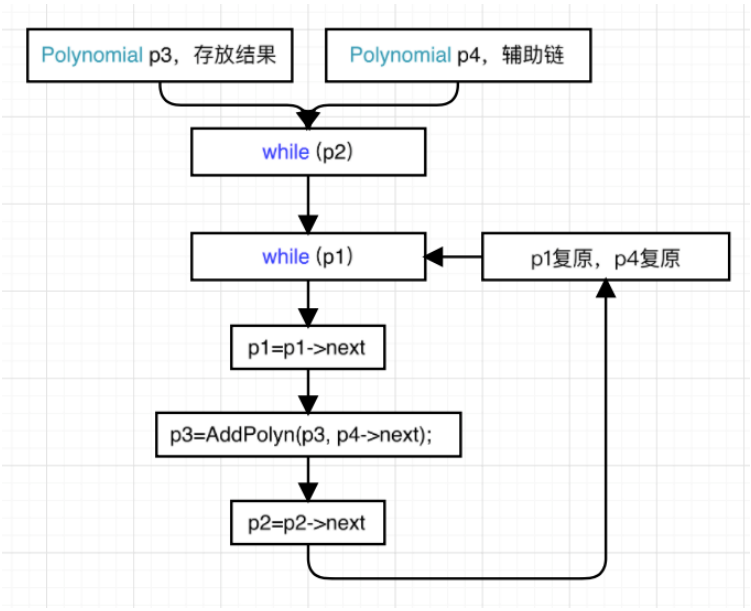
1、相加：


Polynomial p3=(Polynomial)malloc(sizeof(Polyn));
while (p1 && p2)
p1->power < p2->power
p1->power = p2->power
p1->power > p2->power
p3->next=(Polynomial)malloc(sizeof(Polyn));
p3->next=(Polynomial)malloc(sizeof(Polyn));
p1=p1->next
p1->coeff+p2->coeff!=0
p2=p2->next
p3->coeff=p1->coeff+p2->coeff;
p3->power=p1->power;
p1=p1->next;
p2=p2->next;

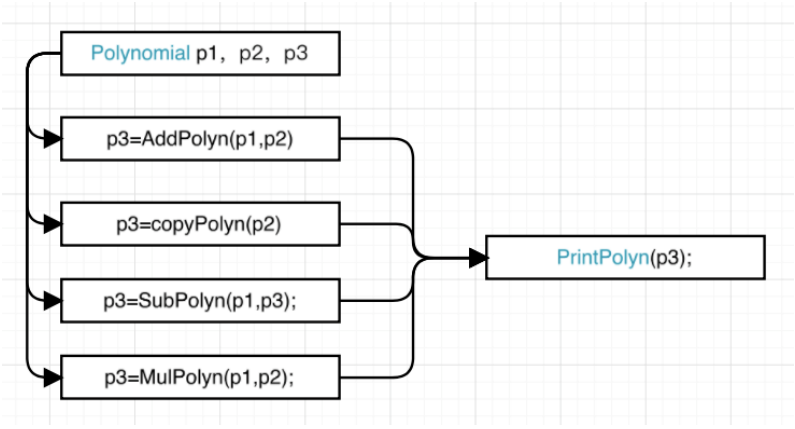
2、相减


while (p2)
p->coeff=-(p->coeff); //变负
p2=p2->next
AddPolyn(p1, p2)

3、相乘


Polynomial p3，存放结果
Polynomial p4，辅助链
while (p2)
while (p1)
p1复原，p4复原
p1=p1->next
p3=AddPolyn(p3, p4->next);
p2=p2->next

系统功能模块图如下所示



四、源程序

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Polynomial
{
    int coeff;   //系数
    int power;   //幂次
    struct Polynomial *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 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;
    return p1->next;  //直接返回一个指向第一个值的指针
}
void PrintPolyn(Polynomial p)
{
    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;
    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);
    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
            pn=pn->next;
            pn->coeff=p1->coeff*p2->coeff;  //系数
            pn->power=p1->power+p2->power;  //幂次
            p1=p1->next;
        }
        pn->next=NULL;
        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
            p3=AddPolyn(p3, p4->next);
        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 p3 = NULL;  //辅助结点

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

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

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

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

    return 0;
    }
```

五、运行结果截图

```
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: 4
输入多项式的系数和对应幂次:
8 1
 2 4
 11 7
 -9 8
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
```

六、小结

因为我在解决一元多项式问题中，使用了链表的方式建立的一元多项式，所以程序的空间是动态的生成的，而且且链表可以灵活地添加或删除结点，所以使得程序得到简化。但是出现的语法问题主要在于子函数和变量的定义，降序排序，关键字和函数名称的书写，以及一些库函数的规范使用，这些问题均可以根据编译器的警告提示，对应的将其解决。

一、题目二

矩阵的运算

二、算法及数据结构

输入两个不同矩阵，对这两个矩阵进行加、减、相乘、置换处理。采用结构体、结构体数组存储矩阵，用一维数组处理运算。加减之前对矩阵进行判断，若两个矩阵的行数相同，列数相同，那么可以进行加减，否则不可以；乘法之前也需判断，若第一个矩阵列数和第二个矩阵行数相同，则可以相乘，否则不可以。

数据结构定义：

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

主要算法:

```
//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;
```

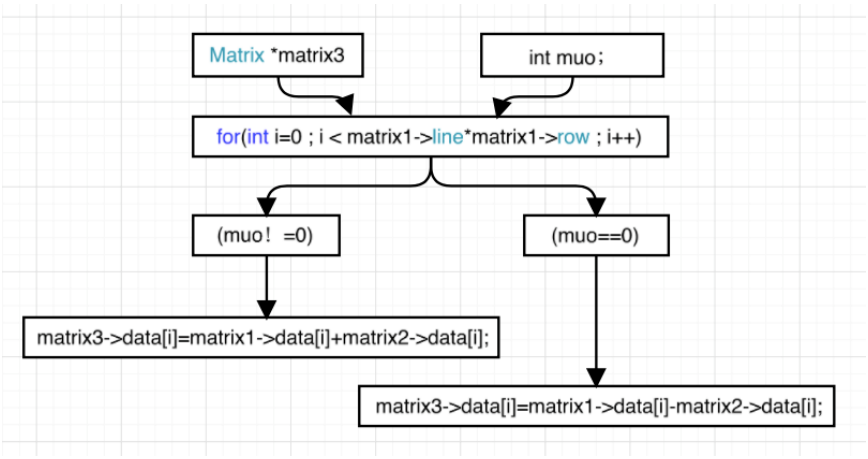
```
    int m=0,n=0;  //m为数组2增量服务，n为数组1增量服务
    for(int i=0 ; i < matrix1->line*matrix2->row ; i++)//大循环，i为新数组下标
    {
        matrix3->data[i]=0;  //在赋值新数组元素前，初始化每个元素为0
        int k=0;
        if(m == matrix2->row)  //每当数组2的一列乘完，m归零，数组1换下一行
        {
            m=0;
            n+=matrix1->row;
 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

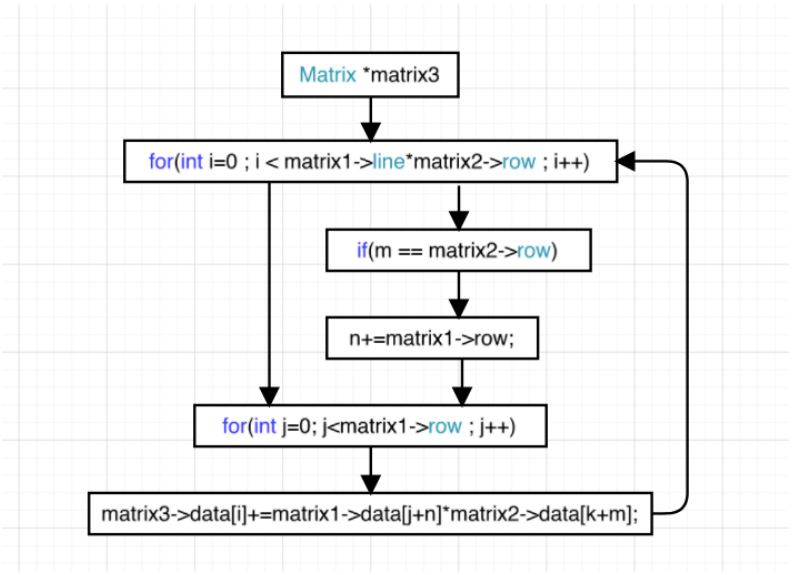
三、思路图

根据需求分析，可以把这个系统的设计分为：

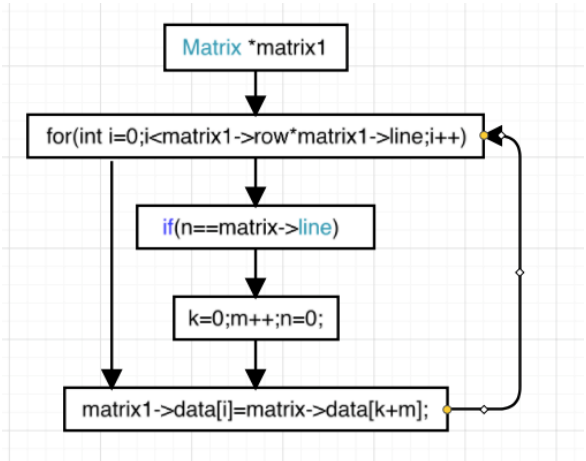
1、相加，相减



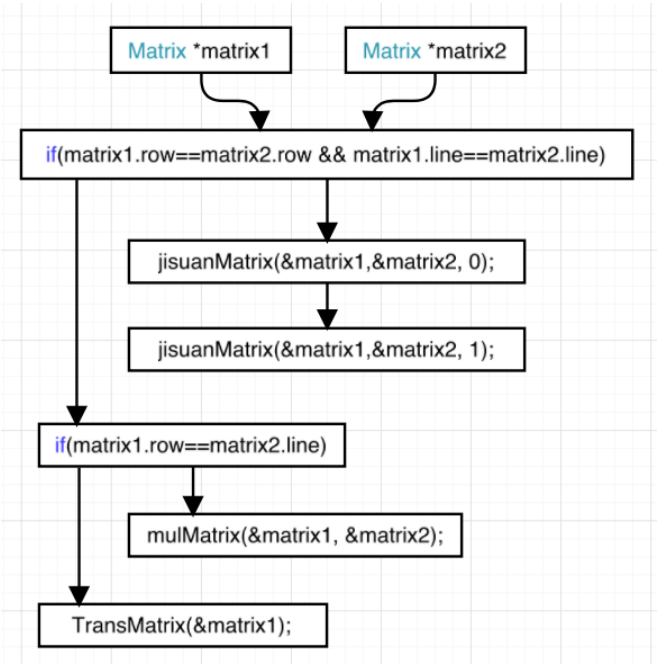
2、相乘



3、置换



系统功能模块图如下所示



四、源程序

```
#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++)
    {
```

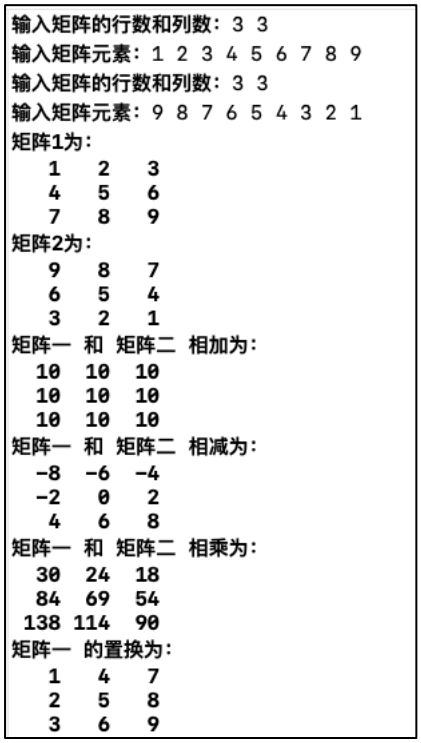
```
            printf("%4d", matrix->data[i]);
            if((i+1)%matrix->row == 0)
                printf("\n");
        }
}
//矩阵加减法
Matrix jisuanMatrix(Matrix *matrix1,Matrix *matrix2,int muo)
{
    Matrix *matrix3=(Matrix *)malloc(sizeof(Matrix));
    matrix3->line=matrix1->line;
    matrix3->row=matrix1->row;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

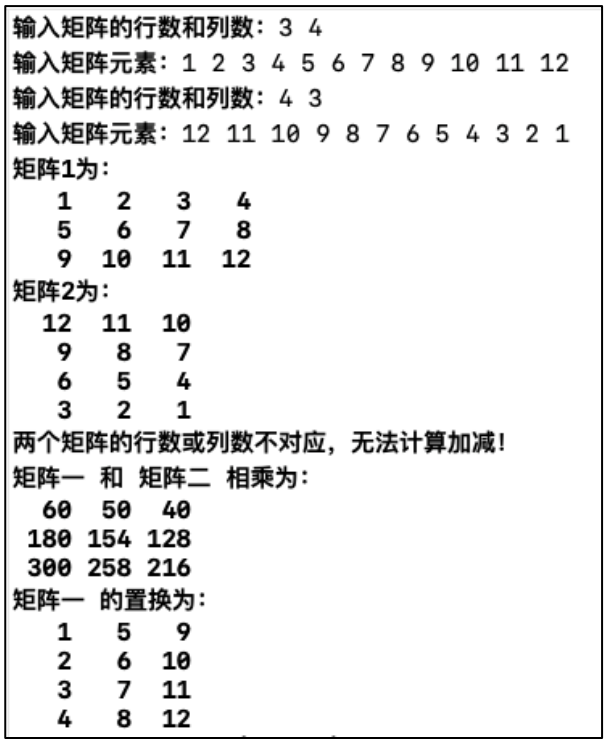
```
    PrintMatrix(matrix3);
    return *matrix3;
}
//矩阵置换
Matrix TransMatrix(Matrix *matrix)
{
    Matrix *matrix1=(Matrix *)malloc(sizeof(Matrix));
    matrix1->line=matrix->row;
    matrix1->row=matrix->line;
    int m=0,n=0,k=0;
    for(int i=0;i<matrix1->row*matrix1->line;i++)   //大循环，i为新数组下标
    {
        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;
}
```

五、运行结果截图





六、小结

我在乘法函数的编写过程中也遇到了大量的问题，由于要同时比较多个关键字，而且设计中涉及了数组和链表的综合运用，导致反复修改了很长的时间才完成了一个相乘的设计。这也是本程序中一个不完美的地方。我在设计函数的时候由于考虑不够充分就直接编写程序，走了很多弯路，不得不停下来仔细研究算法，后来发现由于前边的加法函数完全适用于减法，可见算法的重要性不低于程序本身。在输入矩阵时，按顺序将元素输入链表，一维处理比二维的都复杂很多，所以仍需要学习，提升自己。

一、题目三

迷宫求解

二、算法及数据结构

先画出迷宫二维数组的图案，再通过循环，将迷宫数组输入到结构体中，开始走迷宫，开始输入入口，若入口不正确，则一直输入；首先入口进栈，按“上、左、下、右“的顺序找“1”的路径，为“1”的坐标进栈，若为“0”则路不通，退出栈顶元素再寻找，若有不同的出口则路线不唯一。关键在于顺序栈使用，不断进栈和出栈来寻找出口。

数据结构定义:

```
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;
```

主要算法:

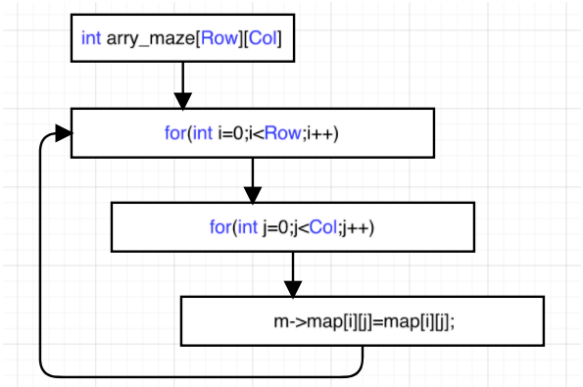
```
int Judge_Enter(Position en)  //判断是否为入口
{
    if((en.x>=0 || en.x<Row) && (en.y>=0 || en.y<Col))
        return 1;
    return 0;
}
int Judge_Exit(Position set,Position en)  //判断是否为出口
{
    if((set.x==0||set.y==0||set.x==Col-1||set.y==Row-1)&&(set.x!=en.x||set.y!=en.y))
        return 1;
    return 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 Start_Maze(PMaze m,Position en) //走迷宫开始
{
    pseqstack s=start();
    push(s, en);  //入口进栈
    while (!empty(s))
    {
        Position set=gettop(s);   //取栈顶
```

```
        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))
            return;
        else
        {
            next=set;
            next.x-=1;   //上，x代表行数
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            next=set;
            next.y-=1;   //左，y代表列数
            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;  //要出栈之前，元素变为3
            pop(s);  //出栈
        }
    }
    printf("迷宫不存在通路\n");
}
```

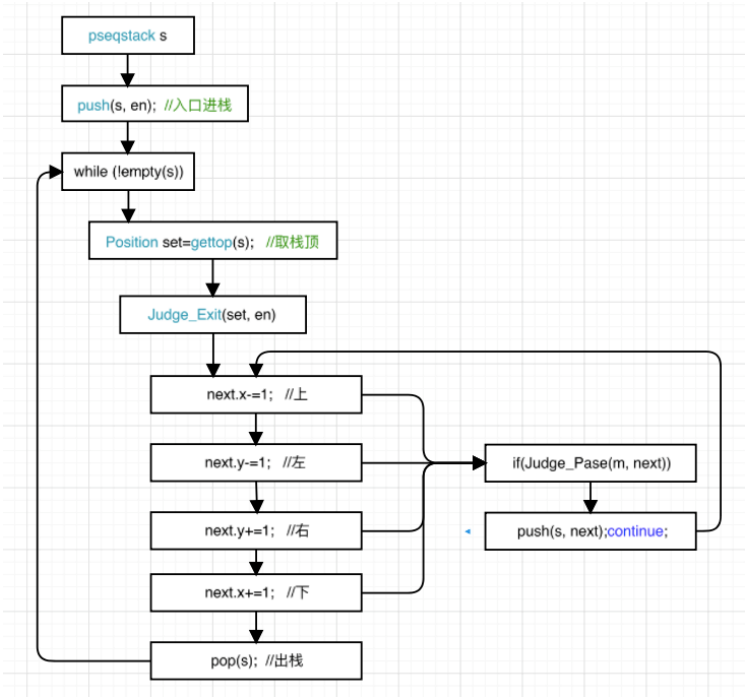
三、思路图

根据需求分析，可以把这个系统的设计分为：

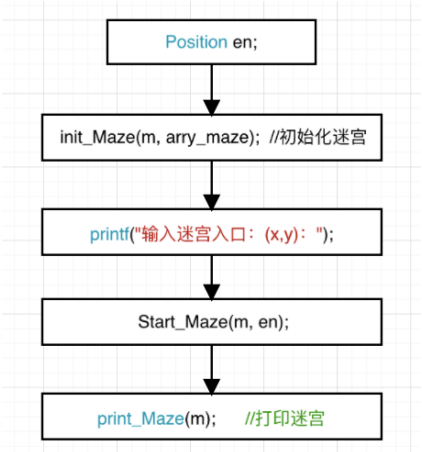
1、建图



2、走迷宫



系统功能模块图如下所示



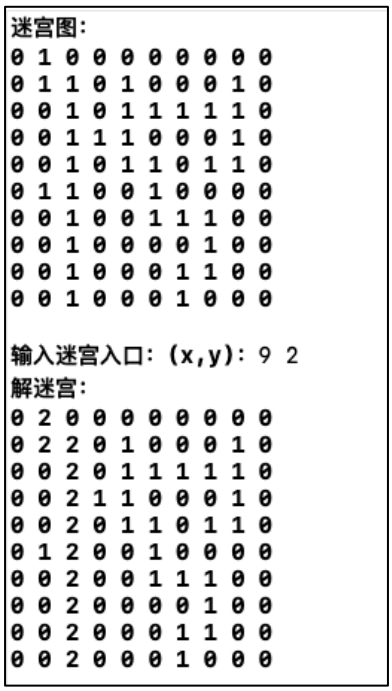
四、源程序

```
#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)
{//创建顺序栈
    pseqstack s;
    s=(pseqstack)malloc(sizeof(seqstack));
    if(s)
        s->top=-1;
    return s;
}
int empty(pseqstack s)
{//判断栈是否为空
    if(s->top==-1)
        return 1;
    else
        return 0;
}
int push(pseqstack s,Position x)
{//栈顶插入新元素x
    if(s->top==MAXSIZE-1)
        return 0;//栈满无法入栈
    else
    {
        s->top++;
        s->arry[s->top]=x;
        return 1;
    }
}
void pop(pseqstack s)
{//删除栈顶元素
    if(empty(s))
        return;//栈空不能出栈
    else
        s->top--;
}
Position gettop(pseqstack s)
{//取出栈顶元素
    if(empty(s))
        exit(0);//栈空
    else
        return s->arry[s->top];
}
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];
}
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");
}
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 Start_Maze(PMaze m,Position en)
{
    pseqstack s=start();
    push(s, en);  //入口进栈
    while (!empty(s))
    {
        Position set=gettop(s);   //取栈顶
        Position next;
        m->map[set.x][set.y]=2;//当前变为2
        //print_Maze(m);   //展示每一步
        if(Judge_Exit(set, en))
            return;
        else
        {
            next=set;
            next.x-=1;   //上，x代表行数
            if(Judge_Pase(m, next))
            {
                push(s, next);
                continue;
            }
            next=set;
            next.y-=1;   //左，y代表列数
            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;//要出栈
之前，元素变为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, 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 }
    };
    printf("迷宫图：\n");
    init_Maze(m, arry_maze);  //初始化迷宫
    print_Maze(m);            //打印迷宫
    do{
        printf("输入迷宫入口：(x,y)：");
        scanf("%d %d",&en.x,&en.y);
    }while(!Judge_Enter(en));
```

```
    printf("解迷宫：\n");
    Start_Maze(m, en);  //开始迷宫
    print_Maze(m);      //打印迷宫
    return 0;
}
```

五、运行结果截图



六、小结

在这次数据结构课程设计中，关于栈的问题主要就是入栈与出栈。栈的特性就是 LIFO（先进后出）。其次在迷宫的操纵就是深度优先搜索找出一条路径。在根据这条路径找到所有的路径。从这次数据结构中我学习到了关于数据结构的一些对于栈的操作及对于图的深度优先搜索的方法，在与老师交流的过程中发现许多问题：列如这张地图在处理过程中怎样先把图中不需要走的路径先给堵上。在对于全零的地图处理上，由于按照现实的观点一个人在走迷宫过程中是不可能将并行的两条通路看作是一条通路所以我和老师交流摒弃自己的四宫格遍历方法，实现老师的九宫格方案对地图进行处理。其次在与老师交流过程中，改变自己许多编程上的陋习加上结合自己所学的知识点，无论是从编程风格还是从编程思路上都是对自己的不小的进步。

在思考问题上，始终保持的将代码能够尽可能的简单，并且做到一个函数只做一个功能尽可能将程序的耦合性能够降到最低，能过够便于其他人阅读自己的代码。