数据结构课程设计

一、课程性质、目的与任务

通过课程设计的锻炼使学生进一步加强对所学知识的理解和掌握，培养学生利用各种数据结构(如线性表、栈、队列、树和图)分析问题和解决问题的能力，提高学生算法设计与分析的能力。

数据结构课程设计不同于一般上机实验，它强调设计性和综合性，难度和分量都比较大。本章给出课程设计的基本要求并通过几个范例详细说明课程设计各个步骤的具体实现。

二、教学内容、基本要求及学时分配

课程内容	教学要求	重点（☆）	难点（△）	学时安排	实验学时	上机学时	备注
问题描述				2			
问题分析及解决方案框架确定			△	2			
数据结构定义		☆		2			
详细设计和编码		☆		20		20	
调试程序				4		4	
撰写报告				2			

（教学基本要求：A-熟练掌握；B-掌握；C-了解）

三、课程设计报告基本要求

设计报告一般要以固定规格的纸张(如 A4)书写或打印并装订，字迹及图表要清楚、工整、规范。内容主要包括下面几个方面：

(1) 问题描述
(2) 设计思路 (数学模型的选择)
(3) 数据结构定义
(4) 系统功能模块介绍
(5) 程序清单
(6) 运行与调试分析等

四、实验项目

1．一元多项式计算

(1) 问题描述

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。

(2) 设计思路 (数学模型的选择)

采用结构体、结构体指针，用链表插入或删除相乘运算之后产生的不同的幂次项，对相同幂次项相加相减。

(3) 数据结构定义

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

(4) 系统功能模块介绍

相加、相减、相乘

(5) 程序清单

```
#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;
    }
```

(6) 运行与调试分析等

```
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
```

2. 矩阵的运算

(1) 问题描述

通过输入两个不同矩阵，对这两个矩阵进行加、减、相乘、置换处理。

(2) 设计思路（数学模型的选择）

采用结构体、结构体数组存储矩阵，用一维数组处理运算。

(3) 数据结构定义

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

(4) 系统功能模块介绍

矩阵相加、相减、相乘、置换

(5) 程序清单

```
#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)
{
    if(matrix1->row==matrix2->row && matrix1->line==matrix2->line)
    {
        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;
    }
    else
    {
        printf("两个矩阵的行数或列数不一样，无法计算！\n");
        exit(0);
    }
}
//矩阵乘法
Matrix mulMatrix(Matrix *matrix1,Matrix *matrix2)
{
    if(matrix1->row==matrix2->line)
    {
        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;
    }
    else
```

```
    {
        printf("两个矩阵的行数或列数不对应，无法计算！\n");
        exit(0);
    }
}
//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);
```

```
    }
    if(matrix1.row==matrix2.line)
    {
        printf("矩阵一 和 矩阵二 相乘为：\n");
        mulMatrix(&matrix1, &matrix2);
    }
    printf("矩阵一 的置换为：\n");
    TransMatrix(&matrix1);
    return 0;
    }
```

(6) 运行与调试分析等

```
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
两个矩阵的行数或列数不对应，无法计算加减!
矩阵一 和 矩阵二 相乘为:
  60  50  40
 180 154 128
 300 258 216
矩阵一 的置换为:
   1   5   9
   2   6  10
   3   7  11
   4   8  12
```

3．迷宫求解

(1) 问题描述

画出迷宫二维数组图案，通过找“1”的路径，来找出口，路线不唯一

(2) 设计思路（数学模型的选择）

采用顺序栈，不断进栈和出栈，找出口

(3) 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;
```

(4) 程序清单

```
#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");
}
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 }
    };
    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;
    }
```

(5) 运行与调试分析等