

实验报告

题目：编制一个交互式计算器，用户可以提出不同的数据处理或计算要求。

班级：理科实验班 6 班 姓名：李梓童 学号：2017202121 完成日期：2018.11.24.

一、需求分析

1、程序具有基本的交互界面，以用户和计算机对话的方式执行。即在计算器终端上显示计算器菜单之后，由用户在键盘上输入程序规定的运算指令，相应的输入数据和运算结果显示其后。

2、计算器的基本功能如下：

①使用顺序表来完成同维度向量的计算，包括加法、减法、夹角余弦值等。向量的最大维度为 50，用户需分别输入两个向量的维度和分量，若输入维度小于 0 或内存分配失败，程序作出相应提示。

②使用顺序表来完成任意一元多项式的计算，包括加法、减法、乘法、导数（包括任意阶）等。用户需要首先输入多项式的最高次项，然后根据计算机终端显示的说明依次输入各项系数和次数，完成输入后，程序会自动计算两个多项式的加法、减法、乘法和求导结果。

③使用链表来完成任意一元多项式的计算，包括加法、减法、乘法、导数（包括任意阶）等。用户需要根据计算机终端显示的说明按照大小次序输入各项系数和次数，完成输入后，程序会自动计算两个多项式的加法、减法、乘法和求导结果。

④四则运算表达式求值。操作符包括加（'+'）、减（'-'）、乘（'*'）、除（'/'）、幂（'^'）、左括号（'('）、右括号（'）'），而操作数全为整数时，使用整型计算器计算；操作数中有浮点数时，使用浮点型计算器计算。用户可根据相应的计算需求选择不同的计算器，根据计算机终端显示的说明输入表达式后，程序会给出相应的运算结果。

⑤含单变量的表达式求值，变量可以是 C/C++ 的标识符。此功能包含在定义并运行简单函数中，可以在整数型计算器中使用。比如定义： $f(x)=3+4*x$ ，然后执行 $f(5)$ ，则得到结果 23。例如下面的运行效果

```
DEF f(x)=3+4*x;  
RUN f(5);  
23
```

用户需要根据计算机终端显示的说明输入函数的定义式和运算指令，程序会自动计算给出相应的结果。

⑥保留函数定义历史，并可以运行历史函数，此功能包含在函数的调用中，在整型计算器中使用。比如已经定义了函数 $f(x)$ ，新定义函数 $g(x)$ 中调用了 f 。例如：

```
DEF f(x)=3+4*x;
```

```
DEF g(x)=3+4*f(x);  
RUN g(5);  
95
```

用户需要根据计算机终端显示的说明输入函数的定义式和运算指令，程序会自动计算给出相应的结果。

⑦操作数的各种字面常量的处理，如包含科学计数法表示的浮点数、负数、正弦、余弦值的四则运算。此项功能包含在浮点数计算器中，用户可根据计算机终端显示的说明输入相关的表达式，程序会自动计算给出相应的结果。

⑧支持矩阵的运算，如矩阵的加、减、乘、转置、逆矩阵、行列式的值。此项功能由矩阵计算器执行，用户根据总菜单指令进入矩阵计算器后，需分别输入两个矩阵的维度和具体值，程序会自动判断矩阵有关运算的可计算性给出相应的结果。

3、测试数据

①向量计算，对应总菜单输入“1”，第一个向量维度：“2”，分量：“0 1”；第二个向量维度：“2”，分量：“1 1”；

②链表实现多项式计算，对应总菜单输入“2”，第一个多项式：“1,0”“2,1”“3,3”“2,4”“0（结束输入）”，第二个多项式“3,0”“1,1”“2,2”“1,3”“0（结束输入）”。

③整数型四则运算，对应总菜单输入“3”，然后输入“2*3+3+4/2”“1（使程序继续运行）”“DEF f(x)=5*x+1”“1”“DEF g(x)=f(x)+4+x/2”“1”“RUN f(3)”“1”“RUN g(4)”“0（结束输入）”。

④矩阵计算器，对应总菜单输入“4”，然后输入第一个矩阵维度：“3 3”，输入第一个矩阵“1 2 3（换行）2 1 3（换行）3 2 1”，输入第二个矩阵维度“3 3”，输入第一个矩阵“4 5 6（换行）1 3 5（换行）1 1 2”。

对应总菜单输入“4”，然后输入第一个矩阵维度：“2 3”，输入第一个矩阵“1 2 3（换行）2 1 3”，输入第二个矩阵维度“3 3”，输入第一个矩阵“4 5 6（换行）1 3 5（换行）1 1 2”。

⑤浮点数计算器，对应总菜单输入“5”，然后输入“1e2+cos45+(-50)”。

⑥线性表实现多项式计算，对应总菜单输入“6”，输入第一个多项式最高次“4”，依次按照次数和系数的顺序输入 0,1,1（最后的 1 来保持输入状态）,1,2,1,3,3,1,4,2,0（结束输入）。第二个多项式最高次“3”，依次输入 0,3,1,1,1,2,2,1,3,1,0（结束输入）。

二、概要设计

为了实现上述程序功能，需要用到线性表、链表、栈等多种存储结构，用到的抽象数据类型有向量、多项式（链表）、多项式（线性表）、栈（四则运算使用）、矩阵。

①向量的抽象数据类型定义为：

ADT Vector{

数据对象: $D=\{a_i \mid a_i \in \text{Vectors}, i=1,2,\dots,n, n \geq 0\}$

数据关系: $R1=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, i=1,2,\dots,n-1 \}$

基本操作:

bool GetVectors(Vector &V)

操作结果: 完成向量的输入

bool AddVectors(Vector V1, Vector V2, Vector &ans)

初始条件: 向量 V1、V2 已存在

操作结果: 将 V1、V2 相加的结果赋给向量 ans

bool SubVectors(Vector V1, Vector V2, Vector &ans)

初始条件: 向量 V1、V2 已存在

操作结果: 将 V1、V2 相减的结果赋给向量 ans

double CalCos(Vector V1, Vector V2)

初始条件: 向量 V1、V2 已存在

操作结果: 计算 V1、V2 的夹角余弦值作为返回值

bool ClearVectors(Vector &V)

初始条件: 向量 V 已存在

操作结果: 清空向量 V

void PrintVectors(Vector V)

初始条件: 向量 V 已存在

操作结果: 打印向量 V

bool DestroyVectors(Vector &V)

初始条件: 向量 V 已存在

操作结果: 销毁向量 V

}

②多项式（链表）抽象数据类型定义

ADT DXS{ //多项式（链表）

数据对象: $D=\{a_i \mid a_i \in \text{Xiang}, i=1,2,\dots,n, n \geq 0\}$

数据关系: $R1=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, i=1,2,\dots,n-1 \}$

基本操作:

Xiang * InitDXS()

操作结果: 创建一个空多项式

void FreeDXS(Xiang * head)

初始条件: 多项式 head 已存在

操作结果: 释放 head 的存储空间, 头指针保留

void DesDXS(Xiang * head)

初始条件: 多项式 head 已存在

操作结果: 释放整个 head 存储空间（包括头指针）

void InputDXS(Xiang * head)

初始条件: 多项式 head 已存在

操作结果: 完成多项式 head 的输入

void OutputDXS(Xiang *head)

初始条件: 多项式 head 已存在

```

        操作结果: 打印多项式 head
Xiang *AddDXS (Xiang * X1, Xiang * X2)
    初始条件: 多项式 X1、X2 已存在
    操作结果: 实现 X1 和 X2 的相加
Xiang *SubDXS (Xiang * X1, Xiang * X2)
    初始条件: 多项式 X1、X2 已存在
    操作结果: 实现 X1 和 X2 的相减
Xiang * MultDXS(Xiang * X1, Xiang * X2)
    初始条件: 多项式 X1、X2 已存在
    操作结果: 实现 X1 和 X2 的相乘
Xiang *DerDXS(Xiang * head)
    初始条件: 多项式 head 已存在
    操作结果: 实现 head 的求导
}

```

③多项式（线性表）抽象数据类型定义

ADT Plnms{ //多项式（线性表）

数据对象: $D=\{a_i \mid a_i \in \text{Plnm}, i=1,2,\dots,n, n \geq 0\}$

数据关系: $R1=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, i=1,2,\dots,n-1 \}$

基本操作:

bool GetPlnm(Plnm &P)

操作结果: 创建一个空多项式

bool ClearPlnm(Plnm &P)

初始条件: 多项式 P 已存在

操作结果: 清空 P

bool DestroyPlnm(Plnm &P)

初始条件: 多项式 P 已存在

操作结果: 释放 P 的存储空间

bool GetPlnm(Plnm &P)

初始条件: 多项式 P 已存在

操作结果: 完成多项式 P 的输入

void PrintPlnm(Plnm P)

初始条件: 多项式 P 已存在

操作结果: 打印多项式 P

bool AddPlnms(Plnm P1, Plnm P2, Plnm &ans)

初始条件: 多项式 P1、P2 已存在

操作结果: 实现 P1 和 P2 的相加

bool SubPlnms(Plnm P1, Plnm P2, Plnm &ans)

初始条件: 多项式 P1、P2 已存在

操作结果: 实现 P1 和 P2 的相减

bool MulPlnms(Plnm P1, Plnm P2, Plnm &ans)

初始条件: 多项式 P1、P2 已存在

操作结果: 实现 P1 和 P2 的相乘, 结果放入 ans 中

bool DePlnms(Plnm &P)

初始条件：多项式 P 已存在
 操作结果：实现 P 的求导
 }

④栈抽象数据类型定义

ADT Stack{ //栈（四则运算）

数据对象： $D=\{a_i \mid a_i \in \text{dataSet}, i=1,2,\dots,n, n \geq 0\}$

数据关系： $R1=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, i=1,2,\dots,n-1 \}$

基本操作：

void initStack(Stack *sta)

操作结果：初始化栈 sta

void EmptyStack(Stack &sta)

初始条件：栈 sta 已存在

操作结果：清空栈 sta

bool Push(Stack &sta, int e)

初始条件：栈 sta 已存在

操作结果：把元素 e 压入栈

bool Pop(Stack &sta, int &e)

初始条件：栈 sta 已存在

操作结果：弹出栈顶元素至 e

}

⑤矩阵抽象数据类型定义

ADT Matrixes{ //矩阵

数据对象： $D=\{a_i \mid a_i \in \text{Matrix}, i=1,2,\dots,n, n \geq 0\}$

数据关系： $R1=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, i=1,2,\dots,n-1 \}$

基本操作：

void matrix_output(float dis[][25],int m,int n)

初始条件：矩阵 dis 已存在

操作结果：打印 $m \times n$ 维的矩阵 dis

void matrixadd(float a[][25],float b[][25],int ma,int na,int mb,int nb)

初始条件： $ma \times na$ 的矩阵 a、 $mb \times nb$ 的矩阵 b 已存在

操作结果：实现矩阵 a 和 b 的相加

void matrixsub(float a[][25],float b[][25],int ma,int na,int mb,int nb)

初始条件： $ma \times na$ 的矩阵 a、 $mb \times nb$ 的矩阵 b 已存在

操作结果：实现矩阵 a 和 b 的相减

void matrixmultiply(float a[25][25],float b[25][25],int ma,int na,int mb,int nb)

初始条件： $ma \times na$ 的矩阵 a、 $mb \times nb$ 的矩阵 b 已存在

操作结果：实现矩阵 a 和 b 的相乘

int determinant(float a[25][25],int n)

初始条件： $n \times n$ 的矩阵 a 已存在

操作结果：计算 a 的行列式

void inverse(float a[25][25],int m,int n)

初始条件： $m \times n$ 的矩阵 a 已存在

操作结果：计算 a 的逆矩阵

```
void cofactor(float num[25][25],int f)
    初始条件：f*f 矩阵 num 已存在
    操作结果：计算 num 的伴随矩阵
void transpose_input(float a[][25],int m,int n)
    初始条件：m*n 的矩阵 a 已存在
    操作结果：计算 a 的转置并输出
}
```

本程序包含 7 个模块：

1) 主程序模块

```
int main()
{
    初始化;
    While(命令 !=退出)
    {
        接受命令;
        处理命令;
    }
}
```

2) 向量模块

3) 多项式（链表）模块

4) 整数型四则运算计算器模块

5) 矩阵模块

6) 浮点数四则运算计算器模块

7) 多项式（线性表）模块

各模块之间关系为：

主程序调用向量模块、多项式（链表）模块、整数型四则运算计算器模块、矩阵模块、浮点数四则运算计算器模块、多项式（线性表）模块。

三、详细设计

1、向量模块

```
typedef struct
{
    int * component; //向量的分量
    int length; //向量的维度
}Vector;
```

```
bool GetVectors(Vector &V)
```

```

{
    int i; //循环变量
    printf("维度:\n");
    scanf("%d",&V.length);

    if(V.length<0)
    {
        printf("error!");
        return 1;
    }
    else{
        V.component = (int *) malloc ( V.length * sizeof(int) );
    }

    if(V.component==0){
        printf("内存分配失败.\n");
        exit(1);
    }

    printf("分量:\n");
    for(i=0;i<V.length;i++)
    {
        cin >> V.component[i];
    }

    return 0;
}

bool AddVectors( Vector V1, Vector V2, Vector &ans )
{
    if(V1.length!=V2.length){
        printf("维度不同，不可计算。 \n");
        return 1;
    }
    ans.length=V1.length;
    ans.component=( int * )malloc( ans.length*sizeof(int) ); //分配存储空间

    int i;

    for(i=0;i<ans.length;i++) //按次序相加
    {
        ans.component[i]=V1.component[i]+V2.component[i];
    }
}

```

```

        return 0;
    }

//向量相减
bool SubVectors( Vector V1, Vector V2, Vector &ans )
{
    if(V1.length!=V2.length){
        printf("维度不同，不可计算。 \n");
        return 1;
    }
    ans.length=V1.length;
    ans.component=( int * )malloc( ans.length*sizeof(int) );

    int i;

    for(i=0;i<ans.length;i++) //按次序相减
    {
        ans.component[i]=V1.component[i]-V2.component[i];
    }

    return 0;
}

double CalCos( Vector V1, Vector V2 )
{
    double ans;

    //两向量维度不等则报错
    if(V1.length!=V2.length)
    {
        printf("error!");
        return 1;
    }

    //初始化
    double nume=0;
    double domi1=0,domi2=0,domi=0;
    int i;

    for(i=0;i<V1.length;i++)
    {
        nume+=V1.component[i]*V2.component[i]; //计算向量的点乘
        domi1+=V1.component[i]*V1.component[i]; //计算向量的模的平方
        domi2+=V2.component[i]*V2.component[i]; //计算向量的模的平方
    }
}

```



```

    }

    domi1=sqrt(domi1); //计算向量的模
    domi2=sqrt(domi2); //计算向量的模

    return nume/(domi1*domi2); //点乘除以模的乘积
}

```

```

//清空向量
bool ClearVectors( Vector &V )
{
    V.length=0;
    return 0;
}

```

```

//打印向量
void PrintVectors( Vector V )
{
    cout<<"向量 :";

    int i;
    for(i=0;i<V.length;i++) //i 小于向量的维度
    {
        cout<<V.component[i]<<' ';
    }
    cout<<endl;

    return;
}

```

```

//销毁向量
bool DestroyVectors( Vector &V )
{
    if (V.component) free(V.component);
    return 0;
}

```

```

int vector_all()
{
    Vector V1,V2,ans;

    printf("\n-----向量计算器-----\n");
    printf("第一个向量:\n");
    GetVectors(V1);

```

```

    printf("第二个向量:\n");
    GetVectors(V2);
    printf("\n 加法结果 :  \n");
    AddVectors( V1, V2, ans );
    PrintVectors( ans );
    printf("\n 减法结果 :  \n");
    SubVectors( V1, V2, ans );
    PrintVectors( ans );
    printf("\n 余弦值 : %.3f\n", CalCos( V1,V2 ));
    return 0;
}

```

2、多项式（链表）模块

```

//数据类型
typedef struct Xiang{
    int Xishu; //系数
    int Cishu; //次数
    struct Xiang * next; //下一项的指针
}Xiang;

//初始化多项式
Xiang * InitDXS()
{
    Xiang * head;
    head = (Xiang *)malloc(sizeof(Xiang)); //分配存储空间

    if (head==NULL) return NULL;

    head->next=NULL;
    return head;
}

//释放多项式
void FreeDXS(Xiang * head)
{
    Xiang * Tmp1, *Tmp2;

    Tmp1 = head->next;

    while(Tmp1 != NULL){
        Tmp2 = Tmp1->next;
        free(Tmp1); //依次释放内存
    }
}

```

```

        Tmp1 = Tmp2; //cursor 后移
    }
    head->next = NULL;//设为空指针
}

//销毁多项式
void DesDXS(Xiang * head)
{
    FreeDXS(head);
    free(head);//释放头指针
}

//输入多项式
void InputDXS(Xiang * head)
{
    int Xishu,Cishu;
    Xiang * end, * xNew;

    end = head; //初始化

    while(1){
        printf("请按照“系数,指数”格式、按递增顺序输入,最后输入 0 结束输入:\n");
        scanf("%d,%d", &Xishu, &Cishu);
        if (Xishu!= 0){
            xNew = (Xiang *)malloc(sizeof(Xiang));
            if (xNew == NULL)
                exit(-1);

            xNew->Xishu = Xishu;
            xNew->Cishu = Cishu;
            end->next = xNew;
            end = xNew;//从尾部建立链表
        }
        else{
            break;
        }
    }
    end->next = NULL;//设置尾指针
}

void OutputDXS(Xiang *head)
{
    Xiang * Tmp;

```

```

    Tmp = head->next;

    if (Tmp == NULL){
        printf("\n");
        return;
    }

    printf("%dx^%d", Tmp->Xishu, Tmp->Cishu); //按照系数^次数的格式打印
    Tmp = Tmp->next; //cursor 后移

    while(Tmp != NULL){
        printf("%+dx^%d", Tmp->Xishu, Tmp->Cishu);
        Tmp = Tmp->next;
    }
    printf("\n");
}

//表达式相加
Xiang *AddDXS (Xiang * X1, Xiang * X2)
{
    int tmp;
    Xiang * ans, *xNew, *end;    //结果用
    Xiang * Tmp1, *Tmp2;        //运算用

    Tmp1 = X1->next;
    Tmp2 = X2->next;
    ans = InitDXS();
    end = ans;
    //将两个多项式扫描放入结果链表中
    while(Tmp1 != NULL && Tmp2 != NULL){
        if (Tmp1->Cishu == Tmp2->Cishu){
            tmp = Tmp1->Xishu + Tmp2->Xishu;
            if ( tmp == 0){ //若系数为 0， 则跳过
                Tmp1 = Tmp1->next;
                Tmp2 = Tmp2->next;
                continue;
            }

            xNew = (Xiang *)malloc(sizeof(Xiang));
            if ( xNew == NULL ) exit(1);

            xNew->Xishu = tmp;
            xNew->Cishu = Tmp1->Cishu;
            end->next = xNew; //从尾部建立链表

```

```

        xNew->next = NULL;
        end = xNew;

        Tmp1 = Tmp1->next;
        Tmp2 = Tmp2->next;
    }//如果表达式 1 指数小于 2
    else if(Tmp1->Cishu < Tmp2->Cishu){

        xNew = (Xiang *)malloc(sizeof(Xiang));
        if ( xNew == NULL) exit(1);

        *xNew = *Tmp1;
        xNew->next = NULL;
        end->next = xNew;
        end = xNew;

        Tmp1 = Tmp1->next;
    }//如果表达式 1 指数大于 2
    else{

        xNew = (Xiang *)malloc(sizeof(Xiang));
        if ( xNew == NULL) exit(1);

        *xNew = *Tmp2;
        xNew->next = NULL;
        end->next = xNew;
        end = xNew;
        Tmp2 = Tmp2->next;
    }
}

//将剩余的未扫描项放入结果链表中
while(Tmp1 != NULL){
    xNew = (Xiang *)malloc(sizeof(Xiang));
    if ( xNew == NULL) exit(1);

    *xNew = *Tmp1; //尾部建立链表
    xNew->next = NULL;
    end->next = xNew;
    end = xNew;
    Tmp1 = Tmp1->next;
}

while(Tmp2!= NULL){

```

```

        xNew = (Xiang *)malloc(sizeof(Xiang));
        if ( xNew == NULL) exit(1);

        *xNew = *Tmp2; //尾部建立链表
        xNew->next = NULL;
        end->next = xNew;
        end = xNew;
        Tmp2 = Tmp2->next;
    }
    return ans;
}

//多项式相减
Xiang *SubDXS (Xiang * X1, Xiang * X2)
{
    int tmp;
    Xiang * ans, *xNew, *end;
    Xiang * Tmp1, *Tmp2;

    Tmp1 = X1->next;
    Tmp2 = X2->next;
    ans = InitDXS();
    end = ans;
    //将两个多项式扫描放入结果链表中
    while(Tmp1 != NULL && Tmp2 != NULL){
        if (Tmp1->Cishu == Tmp2->Cishu){
            tmp = Tmp1->Xishu - Tmp2->Xishu;
            if ( tmp == 0){
                Tmp1 = Tmp1->next;
                Tmp2 = Tmp2->next;
                continue;
            }

            xNew = (Xiang *)malloc(sizeof(Xiang));
            if ( xNew == NULL ) exit(1);

            xNew->Xishu = tmp;
            xNew->Cishu = Tmp1->Cishu;
            end->next = xNew;
            xNew->next = NULL;
            end = xNew;

            Tmp1 = Tmp1->next;
            Tmp2 = Tmp2->next;
        }
    }
}

```

```

} //如果表达式 1 指数小于 2
else if(Tmp1->Cishu < Tmp2->Cishu){

    xNew = (Xiang *)malloc(sizeof(Xiang));
    if ( xNew == NULL) exit(1);

    xNew->Xishu = Tmp1->Xishu;
    xNew->Cishu = Tmp1->Cishu;
    xNew->next = NULL;
    end->next = xNew;
    end = xNew;

    Tmp1 = Tmp1->next;
} //如果表达式 1 指数大于 2
else{

    xNew = (Xiang *)malloc(sizeof(Xiang));
    if ( xNew == NULL) exit(1);

    xNew->Xishu = -Tmp2->Xishu;
    xNew->Cishu = Tmp2->Cishu;
    xNew->next = NULL;
    end->next = xNew;
    end = xNew;
    Tmp2 = Tmp2->next;
}
}

//将剩余的未扫描项放入结果链表中
while(Tmp1 != NULL){
    xNew = (Xiang *)malloc(sizeof(Xiang));
    if ( xNew == NULL) exit(1);

    xNew->Xishu = Tmp1->Xishu;
    xNew->Cishu = Tmp1->Cishu;
    xNew->next = NULL;
    end->next = xNew;
    end = xNew;
    Tmp1 = Tmp1->next;
}

while(Tmp2 != NULL){
    xNew = (Xiang *)malloc(sizeof(Xiang));
    if ( xNew == NULL) exit(1);

```

```

        xNew->Xishu = -Tmp2->Xishu;
        xNew->Cishu = Tmp2->Cishu;
        xNew->next = NULL;
        end->next = xNew;
        end = xNew;
        Tmp2 = Tmp2->next;
    }

    return ans;
}

//乘法
Xiang * MultDXS(Xiang * X1, Xiang * X2)
{
    Xiang *ans, *Tmp1, *Tmp2, *Mult1, *Mult2, *xNew, *end;

    Mult1 = InitDXS(); //乘法运算时的中间变量
    Mult2 = InitDXS();
    Tmp1 = X1->next;

    while(Tmp1 != NULL){
        FreeDXS(Mult1);
        end = Mult1;
        Tmp2 = X2->next;

        while(Tmp2 != NULL){
            xNew = (Xiang *)malloc(sizeof(Xiang));
            if ( xNew == NULL) exit(1);

            xNew->Xishu = Tmp1->Xishu * Tmp2->Xishu; //系数相乘
            xNew->Cishu = Tmp1->Cishu + Tmp2->Cishu; //指数相加
            xNew->next = NULL;
            end->next = xNew; //从尾部放入链表
            end = xNew;
            Tmp2 = Tmp2->next;
        }

        ans = AddDXS(Mult1, Mult2); //得到的多项式相加
        FreeDXS(Mult2);

        Mult2->next = ans->next;
        Tmp1 = Tmp1->next;
    }
}

```



```

        DesDXS(Mult1);
        free(Mult2);

        return ans;
    }

//求导
Xiang *DerDXS(Xiang * head)
{

    Xiang *Tmp = head->next;
    Xiang *xNew, *X1, *end;

    X1 = InitDXS();
    end = X1;

    while(Tmp != NULL){

        if (Tmp->Cishu == 0){
            Tmp = Tmp->next;
            continue;
        }

        xNew = (Xiang *)malloc(sizeof(Xiang));
        if ( xNew == NULL) exit(1);

        xNew->Xishu = Tmp->Xishu * Tmp->Cishu; //次数与系数相乘
        xNew->Cishu = Tmp->Cishu - 1; //次数减 1

        end->next = xNew;
        xNew->next = NULL;
        end = xNew; //尾部建立链表
        Tmp = Tmp->next;
    }
    return X1;
}

int DXS_all()
{

    Xiang *X1, *X2, *ans;
    X1 = InitDXS();
    InputDXS(X1);

```

```

X2 = InitDXS();
InputDXS(X2);
printf("表达式 1: \n");
OutputDXS(X1);
printf("第二个表达式: \n");
printf("表达式 2: \n");
OutputDXS(X2);

ans = AddDXS(X1, X2);
printf("表达式 1 + 表达式 2 = ");
OutputDXS(ans);
DesDXS(ans);

ans = SubDXS(X1, X2);
printf("表达式 1 - 表达式 2 = ");
OutputDXS(ans);
DesDXS(ans);

ans = DerDXS(X1);
printf("表达式 1 求导 = ");
OutputDXS(ans);
DesDXS(ans);

ans = MultDXS(X1, X2);
printf("表达式 1 * 表达式 2 = ");
OutputDXS(ans);

DesDXS(ans);
DesDXS(X1);
DesDXS(X2);

return 0;
}

```

3、多项式（线性表）模块

```

typedef struct
{
    int * Xishu;
    int ZuiGaoCi;
}Plnm;

//输入多项式
bool GetPlnm(Plnm &P)

```

```

{
    int i,j;
    printf("请输入该多项式的最高次项:\n");
    scanf("%d",&P.ZuiGaoCi);

    if(P.ZuiGaoCi<0)
    {
        printf("error!");
        return 1;
    }
    else{
        P.Xishu = (int *) malloc ( (P.ZuiGaoCi+1) * sizeof(int) );
        for(i=0;i<=P.ZuiGaoCi;i++) P.Xishu[i]=0;
    }

    j=1;
    while(j)
    {
        printf("请输入次数: \n");
        scanf("%d",&i);
        if(i<0 || i>P.ZuiGaoCi)
        {
            printf("error!\n");
            continue;
        }
        else
        {
            printf("请输入系数: \n");
            cin>>P.Xishu[i];
        }

        printf("若已完成多项式设置, 请输入 0, 若要继续输入多项式, 请输入 1.\n");
        scanf("%d",&j);
    }

    return 0;
}

//多项式相加
bool AddPlnms( Plnm P1, Plnm P2, Plnm &ans )
{
    int i;
    ans.ZuiGaoCi=(P1.ZuiGaoCi>=P2.ZuiGaoCi)?P1.ZuiGaoCi:P2.ZuiGaoCi; //结果的最高次为两者中较大的最高次

```

```

ans.Xishu=( int * )malloc( (ans.ZuiGaoCi+1) * sizeof(int) );
for(i=0;i<=ans.ZuiGaoCi;i++) ans.Xishu[i]=0; //初始化

int lmin=(P1.ZuiGaoCi<=P2.ZuiGaoCi)?P1.ZuiGaoCi:P2.ZuiGaoCi;

for(i=0;i<=lmin;i++)
{
    ans.Xishu[i]=P1.Xishu[i]+P2.Xishu[i]; //相加
}

//当有一方还有剩余的时候，把剩余的项加入结果中
if(P1.ZuiGaoCi<=P2.ZuiGaoCi)
{
    for(i=lmin+1;i<=P2.ZuiGaoCi;i++)
    {
        ans.Xishu[i]=P2.Xishu[i];
    }
    return 0;
}
else
{
    for(i=lmin+1;i<=P1.ZuiGaoCi;i++)
    {
        ans.Xishu[i]=P1.Xishu[i];
    }
    return 0;
}
}

//多项式相减
bool SubPlnms( Plnm P1, Plnm P2, Plnm &ans )
{
    int i;
    ans.ZuiGaoCi=(P1.ZuiGaoCi>=P2.ZuiGaoCi)?P1.ZuiGaoCi:P2.ZuiGaoCi; //取较大的最高次
    ans.Xishu=( int * )malloc( (ans.ZuiGaoCi+1) * sizeof(int) );
    for(i=0;i<=ans.ZuiGaoCi;i++) ans.Xishu[i]=0; //初始化

    int lmin=(P1.ZuiGaoCi<=P2.ZuiGaoCi)?P1.ZuiGaoCi:P2.ZuiGaoCi;

    for(i=0;i<=lmin;i++)
    {
        ans.Xishu[i]=P1.Xishu[i]-P2.Xishu[i]; //相减
    }
}

```

```

//有一方有剩余的时候，添加到结果中去
if(P1.ZuiGaoCi<=P2.ZuiGaoCi)
{
    for(i=lmin+1;i<=P2.ZuiGaoCi;i++)
    {
        ans.Xishu[i]=-P2.Xishu[i];
    }
    return 0;
}
else
{
    for(i=lmin+1;i<=P1.ZuiGaoCi;i++)
    {
        ans.Xishu[i]=-P1.Xishu[i];
    }
    return 0;
}
}

```

//求导

```

bool DePlnms( Plnm &P )
{
    int i;
    for(i=0;i<=P.ZuiGaoCi;i++)
    {
        if(i==0&&P.Xishu[0]!=0) P.Xishu[0]=0; //处理第一项

        if(P.Xishu[i]!=0)
        {
            P.Xishu[i-1]=P.Xishu[i]*i;
            P.Xishu[i]=0;
        }
    }

    P.ZuiGaoCi--;//最高次减 1

    return 0;
}

```

//相乘

```

bool MulPlnms( Plnm P1, Plnm P2, Plnm &ans )
{
    int i,j;

```

```

ans.ZuiGaoCi=P1.ZuiGaoCi+P2.ZuiGaoCi; //取两个多项式最高次之和作为结果的最高次
ans.Xishu=( int * )malloc( (ans.ZuiGaoCi+1) * sizeof(int) );
for(i=0;i<=ans.ZuiGaoCi;i++) ans.Xishu[i]=0; //初始化

for(i=0;i<=P1.ZuiGaoCi;i++)
{
    if(P1.Xishu[i]!=0)
    {
        for(j=0;j<=P2.ZuiGaoCi;j++)
        {
            ans.Xishu[i+j]+=P1.Xishu[i]*P2.Xishu[j]; //依次相乘后累加
        }
    }
    else continue;
}

return 0;
}

//清空多项式
bool ClearPlnm( Plnm &P )
{
    P.ZuiGaoCi=-1;
    return 0;
}

//打印多项式
void PrintPlnm( Plnm P )
{
    int i;
    for(i=0;i<=P.ZuiGaoCi;i++)
    {
        if(P.Xishu[i]!=0) cout<<P.Xishu[i]<<"x^"<<i;
        if(i>0&&i!=P.ZuiGaoCi&&P.Xishu[i+1]>0) cout<<'+';
    }
    cout<<endl;
    return;
}

bool DestroyPlnm( Plnm &P )
{
    if (P.Xishu) free(P.Xishu);
    return 0;
}

```

```

int DXS_sql_all()
{
    Plnm P1,P2,ans;
    int control=1;

    printf("-----多项式简易计算器-----\n");
    printf("请输入第一个多项式:\n");
    GetPlnm(P1);
    printf("请输入第二个多项式:\n");
    GetPlnm(P2);

    printf("加法: \n");
    AddPlnms( P1, P2, ans );
    PrintPlnm( ans );

    printf("减法: \n");
    SubPlnms( P1, P2, ans );
    PrintPlnm( ans );

    printf("乘法: \n");
    MulPlnms( P1, P2, ans );
    PrintPlnm( ans );

    printf("第一个多项式求导: \n");
    DePlnms( P1 );
    PrintPlnm( P1 );

    printf("第二个多项式求导: \n");
    DePlnms( P2 );
    PrintPlnm( P2 );

    return 0;
}

```

4、整数四则运算计算器模块

```

const int MAXSIZE=30;
int x=0;//单变量值
int ShuChu=0; //控制输出与否
int YouXianJi[8]={0,0,2,1,0,1,0,2};//运算符优先级
int HaShHao=0;
char HanShuMing[10][5];
char HanShuTi[10][MAXSIZE];

```

```

//top 存储栈顶是第几个元素，栈底即为 data[0]
typedef struct
{
    int data[MAXSIZE];
    int top;
} Stack;

Stack sta, sta2;

int HouJiSuan(Stack &sta, int *Hou, int length);
void ZhongtoHou(Stack &sta, char *Zhong, int *Hou, int *length);

//初始栈
void initStack(Stack *sta)
{
    int i;

    for (i=0; i<MAXSIZE; i++)
    {
        sta->data[i] = 0; //所有值设为 0
    }
    sta->top = -1;
}

//清空栈
void EmptyStack(Stack &sta)
{
    int i;

    for(i=0; i<sta.top; i++)
        sta.data[i] = 0; //数值设为 0

    sta.top=-1; //栈顶设为-1

    return;
}

//弹出栈
bool Pop(Stack &sta, int &e)
{
    if(sta.top==0) return 1;

```



```

        e=sta.data[sta.top];
        sta.top--;

        return 0;
    }

//入栈
bool Push(Stack &sta, int e)
{

    sta.data[sta.top++]=e;

    return 0;
}

//如果输入表达式的开头为“DEF”
int Def(char *Zhong)
{
    ShuChu=0;
    int i=4;
    int HanShu=0;

    //第一次出现左括号，开始读取函数名
    while(Zhong[i]!='(')
    {
        HanShuMing[HaShHao][HanShu]=Zhong[i];
        i++;
        HanShu++;
    }

    HanShu=0;

    while(Zhong[i-1]!='=') i++;

    //出现等于号之后，开始读取函数体
    while(Zhong[i]!='0')
    {
        HanShuTi[HaShHao][HanShu]=Zhong[i];
        i++;
        HanShu++;
    }

    //函数的序号加 1
    HaShHao++;
}

```

```

    printf("  你  输  入  的  函  数  名  为  :   %s   ,   函  数  体
为: %s\n",HanShuMing[HaShHao-1],HanShuTi[HaShHao-1]);
    return 0;
}

```

//输入表达式的开头为“RUN”

```
int Run(char *Zhong,int func) //func: 0,内层函数;1:最外层函数
```

```

{
    if(func) ShuChu=1;

    int  length,ans;
    int  Hou[MAXSIZE];
    char Zhong2[MAXSIZE];
    int i=4;
    int j=0;
    x=0;
    char tmp[5];

    for(j=0;j<5;j++)
    {
        tmp[j]=0;//初始化 tmp
    }
    j=0;

    while(Zhong[i]!='(')
    {
        tmp[j]=Zhong[i];//读取函数名
        i++;
        j++;
    }

    for(j=0;j<HaShHao;j++)
    {
        if(strcmp(tmp,HanShuMing[j])==0) {
            printf("找到目标函数。 \n");
            break;
        }
        //搜索函数名
    }

    if(j>=HaShHao&&func)
    {
        printf("没有找到该函数\n");
    }
}

```

```

        ShuChu=0;
        return -1;
    }

    i++;

    //读取自变量的值
    while(Zhong[i]!='')
    {
        x=x*10+(Zhong[i]-'0');
        i++;
    }

    //把函数体拷贝到中缀表达式中，然后开始解析中缀表达式
    strcpy(Zhong2,HanShuTi[j]);

    initStack(&sta);
    ZhongtoHou(sta, Zhong2, Hou, &length); //解析中缀表达式
    initStack(&sta);
    ans=HouJiSuan(sta, Hou, length);

    if(ShuChu) printf("计算结果为 : %d\n",ans);
    return ans;
}

//判断分支
void DEForRUN(char *Zhong)
{
    int i;
    for (i=0;i<strlen(Zhong);)
    {
        //DEF 为输入函数的标志，把函数名和函数体分别存在两个二维数组里
        if(Zhong[i]=='D'&&Zhong[i+1]=='E'&&Zhong[i+2]=='F')
        {
            Def(Zhong);
            return;
        }

        //RUN 为运行函数的标志，通过用 HaShHao 来搜索函数名，锁定相应的函数
        else if(Zhong[i]=='R'&&Zhong[i+1]=='U'&&Zhong[i+2]=='N')
        {
            Run(Zhong,1);
            return;
        }
    }
}

```

```

//如果既不是 DEF 也不是 RUN，就是普通的四则运算表达式计算
else{
    int length,ans;
    int Hou[MAXSIZE];

    initStack(&sta);
    ZhongtoHou(sta, Zhong, Hou, &length); //解析中缀表达式
    initStack(&sta);
    ans=HouJiSuan(sta, Hou, length); //计算后缀表达式

    printf("计算结果为 :%d\n",ans);
    return;
}
}
}

```

//中缀表达式到后缀表达式

```

void ZhongtoHou(Stack &sta, char *Zhong, int *Hou, int *length)
{
    int i;
    int b = 0;
    int j = 0;
    int k = 0;
    int priority = 0;
    char BianL[10]; //存储第一个变量
    char BianL2[10]; //存储第二个变量
    int DiYiCi=0; //判断是否第一次出现变量
    int bi=0;
    int HanShu=0; //在循环中做下标使用，作用相当于 i, j
    int length2;
    int Hou2[MAXSIZE];

    for (i=0;i<strlen(Zhong);)
    {
        //读出自变量时，自动代入变量的值。
        if (!DiYiCi&&((Zhong[i] >= 'A' && Zhong[i] <= 'Z')||
            (Zhong[i] >= 'a' && Zhong[i] <= 'z')||
            (Zhong[i] == '_' )))
        {
            while ((Zhong[i] >= 'A' && Zhong[i] <= 'Z')||
                (Zhong[i] >= 'a' && Zhong[i] <= 'z')||
                (Zhong[i] >= '0' && Zhong[i] <= '9')||
                (Zhong[i] == '_' ))

```

```

    {
        BianL[bi] = Zhong[i];
        i++;
        bi++;
    }

    BianL[bi]=0;
    bi=0;

    for(k=0;k<HaShHao;k++)
    {
        if(strcmp(BianL,HanShuMing[k])==0) //搜索函数名
        {
            initStack(&sta2);
            ZhongtoHou(sta2, HanShuTi[k], Hou2, &length2);
            //如果搜索到内层函数 ， 则先计算内层函数
            initStack(&sta2);
            Hou[j]=HouJiSuan(sta2, Hou2, length2);
            printf("found another:%d", Hou[j]);
            j++;
            i+=3;
            break;
        }
    }
    //把自变量的值代入后缀表达式

    if(k>=HaShHao) //如果没有搜索到内层函数，则按照单变量处理
    {
        Hou[j]=x;
        j++;
        DiYiCi=1;
    }

    continue;
}

if (DiYiCi&&((Zhong[i] >= 'A' && Zhong[i] <= 'Z') ||
    (Zhong[i] >= 'a' && Zhong[i] <= 'z') ||
    (Zhong[i] == '_' )))
{
    while ((Zhong[i] >= 'A' && Zhong[i] <= 'Z') ||
        (Zhong[i] >= 'a' && Zhong[i] <= 'z') ||
        (Zhong[i] >= '0' && Zhong[i] <= '9') ||
        (Zhong[i] == '_' ))

```

```

    {
        BianL2[bi] = Zhong[i];
        i++;
        bi++;
    }
    BianL2[bi]=0;
    bi=0;

    if(strcmp(BianL,BianL2)!=0){
        printf("变量名不一致！ ");
        exit(1);
    }

    Hou[j]=x;
    j++;

    continue;
}

//读到数字，解析入后缀表达式
if (Zhong[i] >= '0' && Zhong[i] <= '9')
{
    b = 0;
    while (Zhong[i] >= '0' && Zhong[i] <= '9')
    {
        b = b * 10 + (Zhong[i] - '0');//把字符转化为数字
        i++;
    }
    Hou[j] = b;
    j++;
    continue;
}

if (Zhong[i] == 41)//读到右括号，没遇到左括号时一直弹出
{
    while (sta.data[sta.top] != 40)
    {
        Hou[j] = sta.data[sta.top];
        sta.data[sta.top] = 0;

        sta.top--;

        j++;
    }
}

```

```

    }
    sta.data[sta.top] = 0;
    sta.top--;

    priority = YouXianJi[sta.data[sta.top] % 10]; //更新优先级

    i++;
    continue;
}
//左括号
if (Zhong[i] == 40)
{
    sta.top++;
    sta.data[sta.top] = Zhong[i];

    priority = YouXianJi[sta.data[sta.top] % 10];
    i++;
    continue;
}

if (Zhong[i] >= 42 && Zhong[i] <= 47)
{
    if (priority >= YouXianJi[Zhong[i] % 10])
    {
        //如果入栈元素优先级低于栈顶元素，则弹出
        while (priority >= YouXianJi[Zhong[i] % 10] && sta.data[sta.top] != 40)
        {
            Hou[j] = sta.data[sta.top];
            sta.data[sta.top] = 0;
            sta.top--;
            priority = YouXianJi[sta.data[sta.top] % 10];

            j++;
        }

        sta.top++;

        sta.data[sta.top] = Zhong[i];
        priority = YouXianJi[sta.data[sta.top] % 10]; //更新优先级
        i++;
    }
    else

```

```

    {
        //负数的情况，“(-”
        if (Zhong[i] == 45 && sta.data[sta.top] == 40)
        {
            b = 0;
            while (Zhong[i+1] >= '0' && Zhong[i+1] <= '9')
            {
                b = b * 10 + (Zhong[i+1] - '0');
                i++;
            }
            Hou[j] = b * -1;
            sta.data[sta.top] = 0;//出栈
            sta.top--;
            j++;
            i += 2;
            priority = YouXianJi[sta.data[sta.top] % 10];//更新优先级

            continue;
        }

        sta.top++;
        sta.data[sta.top] = Zhong[i];
        priority = YouXianJi[sta.data[sta.top] % 10];
        i++;
    }
}

//弹出剩余元素
while (sta.top != -1)
{
    Hou[j] = sta.data[sta.top];
    sta.top--;
    j++;
}

/*  for( i=0 ; i<j ; i++)
    {
        printf("~%d ",Hou[i]);
    }
    printf("\n");
*/
*length=j;//更新长度

```



```
}
```

```
//计算后缀表达式
```

```
int HouJiSuan(Stack &sta, int *Hou, int length)
```

```
{
```

```
    int i,j,ans;
```

```
    ans=0;
```

```
    for (i=0;i<length;i++)
```

```
    {
```

```
        //解析后缀表达式里的运算符，如果是，则对 ans 做相应的操作并压栈
```

```
        switch (Hou[i])
```

```
        {
```

```
            case 42:
```

```
                ans=sta.data[sta.top-1]*sta.data[sta.top];
```

```
                sta.top-=1;
```

```
                sta.data[sta.top]=ans;
```

```
                break;
```

```
            case 43:
```

```
                ans=sta.data[sta.top-1]+sta.data[sta.top];
```

```
                sta.top-=1;
```

```
                sta.data[sta.top]=ans;
```

```
                break;
```

```
            case 45:
```

```
                ans=sta.data[sta.top-1]-sta.data[sta.top];
```

```
                sta.top-=1;
```

```
                sta.data[sta.top]=ans;
```

```
                break;
```

```
            case 47:
```

```
                ans=sta.data[sta.top-1]/sta.data[sta.top];
```

```
                sta.top-=1;
```

```
                sta.data[sta.top]=ans;
```

```
                break;
```

```
            default://如果不是运算符，把后缀表达式的值压栈
```

```
                sta.top++;
```

```
                sta.data[sta.top]=Hou[i];
```

```
                break;
```

```
        }
```

```
    }
```

```
    return ans;
```

```
}
```

```
int SiZeYunSuan_func_all()
```

```
{
```

```

char Zhong[MAXSIZE],Zhong2[MAXSIZE];
int k2=1;//k 控制结束与否

printf("使用细则： \n");
printf("本计算器为整型计算器，支持函数定义功能。 \n");
printf("你可以使用 ‘DEF f(x)=5*x+1’ ‘DEF g(x)=f(x)+4+x/2’ ‘RUN g(4)’ 这样的语句来运行函数\n");
printf("输入： \n");
getchar();

while(k2)
{
    gets(Zhong);//获取中缀表达式
    strcpy(Zhong2,Zhong);

    DEForRUN(Zhong);
    printf("继续，输入 1； 离开，输入 0： \n");
    scanf("%d",&k2);
    getchar();
}

return 0;
}

```

5、矩阵计算器模块

```

void matrix_output(float dis[][25],int m,int n);
int determinant(float a[25][25],int n);
void cofactor(float num[25][25],int f);
void transpose(float num[25][25],float fac[25][25],int r);

void transpose_input(float a[][25],int m,int n) //转置计算
{
    int i,j;
    float a2[25][25];
    for (i=0;i<n;i++)
    {
        for (j=0;j<m;j++)
        {
            a2[i][j]=a[j][i]; //行列对调
        }
    }
    printf("矩阵的转置为:\n");
}

```

```

        matrix_output(a2,n,m);
    }

    //行列式计算
void determinant_input(float a[25][25],int m, int n)
{
    if(m!=n){
        printf("非方阵，无行列式。 \n");
        return;
    }
    int d;
    int i,j;
    d=determinant(a,n); //计算行列式
    printf("行列式为  %d \n",d);
}

//矩阵相加
void matrixadd(float a[][25],float b[][25],int ma,int na,int mb,int nb)
{
    if(ma!=mb || na!=nb){
        printf("维数不同，无法加减\n");
        return;
    }
    int i,j;
    float ans[25][25];
    for(i=0;i<ma;i++)
    {
        for(j=0;j<na;j++)
            ans[i][j]=a[i][j] +b[i][j];
    }
    printf("矩阵相加  \n");

    matrix_output(ans,ma,na);
}

//矩阵相减
void matrixsub(float a[][25],float b[][25],int ma,int na,int mb,int nb) //矩阵相减
{
    if(ma!=mb || na!=nb){
        printf("维数不同，无法加减\n");
        return;
    }
    int i,j;
    float ans[25][25]; //最终结果

```

```

        for(i=0;i<ma;i++)
        {
            for(j=0;j<na;j++)
                ans[i][j]=a[i][j] - b[i][j];
        }
        printf("矩阵相减 \n");

        matrix_output(ans,ma,na);
    }

```

//打印矩阵

```

void matrix_output(float dis[][25],int m,int n)
{
    int i,j;
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%.3f ",dis[i][j]);
        }
        printf("\n");
    }
}

```

//矩阵求逆：矩阵的逆矩阵等于原矩阵的伴随矩阵除以行列式

void inverse(float a[25][25],int m,int n) //矩阵求逆

```

{
    int k,d;
    k=n;
    if(m!=n){
        printf("非方阵，无法求行列式，无法求逆。");
        return;
    }
    d=determinant(a,k);
    printf("行列式为 %d",d);
    if (d==0)
        printf("\n 逆矩阵不存在\n");
    else
        cofactor(a,k); //利用余子式计算逆矩阵
    printf("计算完成!\n");

    return;
}

```

//计算行列式：按行展开，递归求解

```
int determinant(float a[25][25],int k)
{
    int s=1,det=0;
    float b[25][25];
    int i,j,m,n,c;
    if (k==1)
    {
        return (a[0][0]);//一阶方阵的行列式为其唯一元素
    }
    else
    {
        det=0;
        for (c=0;c<k;c++) //按照第一行展开
        {
            m=0;
            n=0;
            for (i=0;i<k;i++)
            {
                for (j=0;j<k;j++)
                {
                    //构造余子式
                    b[i][j]=0;//初步赋值
                    if (i != 0 && j != c) //划去第一行锁定元素的行和列
                    {
                        b[m][n]=a[i][j];
                        if (n<(k-2))
                            n++;
                        else
                        {
                            n=0;
                            m++;//换行
                        }
                    }
                }
            }

            det=det + s * (a[0][c] * determinant(b,k-1)); //递归函数
            s=-1 * s;
        }

        return (det);
    }
}
```

```
}
```

```
//求解伴随矩阵
```

```
void cofactor(float num[25][25],int f)
```

```
{
```

```
float b[25][25],fac[25][25]; //b: 存放伴随矩阵每个值对应的余子式, fac: 存放伴随矩阵
```

```
int p,q,m,n,i,j;
```

```
for (q=0;q<f;q++)
```

```
{
```

```
for (p=0;p<f;p++)
```

```
{
```

```
m=0;
```

```
n=0;
```

```
for (i=0;i<f;i++)
```

```
{
```

```
for (j=0;j<f;j++)
```

```
{
```

```
if (i != q && j != p)
```

```
{
```

```
b[m][n]=num[i][j]; //构造余子式
```

```
if (n<(f-2))
```

```
n++;
```

```
else
```

```
{
```

```
n=0;
```

```
m++;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
fac[q][p]=pow(-1,q + p) * determinant(b,f-1); //计算伴随矩阵对应分量的值
```

```
}
```

```
}
```

```
transpose(num,fac,f);
```

```
}
```

```
//计算最终的逆矩阵
```

```
void transpose(float num[25][25],float fac[25][25],int r)
```

```
{
```

```
int i,j;
```

```
float b[25][25],inverse[25][25],d;
```

```
//inverse 存放逆矩阵,b 存放伴随矩阵的转置
```

```
//计算伴随矩阵的转置
```

```

for (i=0;i<r;i++)
{
    for (j=0;j<r;j++)
    {
        b[i][j]=fac[j][i];
    }
}
d=determinant(num,r);

for (i=0;i<r;i++)
{
    for (j=0;j<r;j++)
    {
        inverse[i][j]=b[i][j] / d; //逆矩阵对应的值等于伴随矩阵转置的值除以行列式
    }
}
printf("\n 逆矩阵为 : \n");

matrix_output(inverse,r,r);

}

void matrixmultiply(float a[25][25],float b[25][25],int ma,int na,int mb,int nb)
{
    if(na!=mb){
        printf("维数不符合乘法要求。 \n");
        return;
    }
    float multi[25][25];

    int i,j,k;
    for (i=0;i<ma;i++)
    {
        for(j=0;j<nb;j++)
        {
            multi[i][j]=0;
            for(k=0;k<nb;k++)
                multi[i][j]=multi[i][j]+(a[i][k]*b[k][j]);
        }
    }

    printf("矩阵相乘: \n");
    matrix_output(multi,ma,nb);
}

```

```

int matrix_all()
{
    float _a[25][25], _b[25][25];
    int _ma, _na, _mb, _nb, i, j;
    printf("-----矩阵计算器-----\n");
    printf("请输入两个矩阵，该计算器将会计算两矩阵的加法、减法，\n 以及第一个矩阵
的行列式、转置和逆矩阵\n");
    printf("输入第一个矩阵的维度（行*列）\n");
    scanf("%d %d", &_ma, &_na);
    printf("输入第一个矩阵\n");
    for(i=0; i<_ma; i++)
    {
        for(j=0; j<_na; j++)
            scanf("%f", &_a[i][j]);
    }

    printf("输入第二个矩阵的维度（行*列）\n");
    scanf("%d %d", &_mb, &_nb);
    printf("输入第二个矩阵\n");
    for(i=0; i<_mb; i++)
    {
        for(j=0; j<_nb; j++)
            scanf("%f", &_b[i][j]);
    }

    transpose_input(_a, _ma, _na);
    determinant_input(_a, _ma, _na);
    matrixadd(_a, _b, _ma, _na, _mb, _nb);
    matrixsub(_a, _b, _ma, _na, _mb, _nb);
    matrixmultiply(_a, _b, _ma, _na, _mb, _nb);
    inverse(_a, _ma, _na);

    return 0;
}

```

6、浮点数四则运算计算器模块

```

#define PI 3.1415926

//top 存储栈顶是第几个元素，栈底即为 data[0]
typedef struct
{
    float  dataf[MAXSIZE];

```



```

        int top;
    } Stackf;

//初始栈
void initStackf(Stackf *sta)
{
    int i;

    for (i=0;i<MAXSIZE;i++)
    {
        sta->dataf[i] = 0;
    }
    sta->top = -1;
}

//中缀表达式转化为后缀表达式
void ZhongtoHouf(Stackf &sta, char *Zhong, float *Hou, int *length)
{
    int i;
    int b = 0;
    int j = 0;
    int bi = 0; //bi 记录 buff 中的位数
    int priority = 0; //优先级
    char buff[10]; //读取数字段时用作暂存数组

    for (i=0;i<strlen(Zhong);)
    {
        if (Zhong[i] >= '0' && Zhong[i] <= '9')
        {
            b = 0;

            while ((Zhong[i] >= '0' && Zhong[i] <= '9') || Zhong[i]=='.')
            {
                buff[bi] = Zhong[i];
                b = b * 10 + (Zhong[i] - '0'); //若不是浮点数运算，用这行代码解析即可
                i++;
                bi++;
            }
            buff[bi]=0;

            sscanf(buff, "%f", &Hou[j]); //把字符数组转化为浮点数

```

```

//只支持个位数的指数，因为只往后读了 1 位。
if(Zhong[i]=='e')
{
    i++;
    Hou[j]*=pow(10,Zhong[i]-'0');
    i++;
}

memset(buff,0,bi);
bi=0; //bi 置零
j++;
continue;
}

//考虑余弦值
if (Zhong[i] == 'c' && Zhong[i+1] == 'o' && Zhong[i+2] == 's')
{
    i+=3;
    b = 0;

    while ((Zhong[i] >= '0' && Zhong[i] <= '9') || Zhong[i]=='.')
    {
        b = b * 10 + (Zhong[i] - '0'); //若不是浮点数运算，用这行代码解析即可
        i++;
    }

    Hou[j]=cos((b/180.0)*PI);
    j++;
    continue;
}

//考虑正弦值
if (Zhong[i] == 's' && Zhong[i+1] == 'i' && Zhong[i+2] == 'n')
{
    i+=3;
    b = 0;

    while ((Zhong[i] >= '0' && Zhong[i] <= '9') || Zhong[i]=='.')
    {
        b = b * 10 + (Zhong[i] - '0'); //若不是浮点数运算，用这行代码解析即可
        i++;
    }

    Hou[j]=sin((b/180.0)*PI);

```

```

        j++;
        continue;
    }

    if (Zhong[i] == 41) //遇到右括号
    {
        while (sta.dataf[sta.top] != 40) //没遇到左括号之前一直弹出
        {
            Hou[j] = sta.dataf[sta.top];
            sta.dataf[sta.top] = 0;

            sta.top--;

            j++;
        }
        sta.dataf[sta.top] = 0;
        sta.top--;

        priority = YouXianJi(((int)sta.dataf[sta.top]) % 10); //更新优先级

        i++;
        continue;
    }

    //左括号
    if (Zhong[i] == 40)
    {
        sta.top++;
        sta.dataf[sta.top] = Zhong[i]; //左括号压栈

        priority = YouXianJi(((int)sta.dataf[sta.top]) % 10); //更新优先级
        i++;
        continue;
    }

    if (Zhong[i] >= 42 && Zhong[i] <= 47)
    {
        if (priority >= YouXianJi(Zhong[i] % 10))
        {
            while (priority >= YouXianJi(Zhong[i] % 10) && sta.dataf[sta.top] != 40)
                //优先级小的运算符入栈
            {
                Hou[j] = sta.dataf[sta.top];

```

```

        sta.dataf[sta.top] = 0;
        sta.top--; //弹出到后缀表达式
        priority = YouXianJi(((int)sta.dataf[sta.top] )% 10);

        j++;

    }

    //一般情况下的压栈
    sta.top++;
    sta.dataf[sta.top] = Zhong[i];
    priority = YouXianJi(((int)sta.dataf[sta.top] )% 10);
    i++;
}
else
{
    //负数的情况：“(- ”
    if (Zhong[i] == 45 && sta.dataf[sta.top] == 40)
    {
        b = 0;
        while ((Zhong[i+1] >= '0' && Zhong[i+1] <= '9') | | Zhong[i+1]=='('.')
        {
            buff[bi] = Zhong[i+1];
            i++;
            bi++;
        }
        bi=0;
        sscanf(buff, "%f", &Hou[j]); //解析 buff
        Hou[j] = Hou[j] * -1;
        j++;

        sta.dataf[sta.top] = 0;
        sta.top--;
        i += 2;
        priority = YouXianJi(((int)sta.dataf[sta.top] )% 10);

        continue;
    }

    sta.top++;
    sta.dataf[sta.top] = Zhong[i];
    priority = YouXianJi(((int)sta.dataf[sta.top])%10);
    i++;
}

```

```

        }
    }
}

//弹出剩余的元素
while (sta.top != -1)
{
    Hou[j] = sta.dataf[sta.top];
    sta.top--;
    j++;
}

*length=j;
}

//后缀表达式的计算
float HouJiSuanf(Stackf &sta, float *Hou, int length)
{
    int i,j;
    float ans;
    ans=0;

    for (i=0;i<length;i++)
    {
        float tmp=Hou[i]; //计算后缀表达式时保存 (int) Hou[i]之前的浮点值。
        switch ((int)Hou[i])
        {
            case 42:
                ans=sta.dataf[sta.top-1]*sta.dataf[sta.top]; //对 ans 进行相应处理后压栈
                sta.top-=1;
                sta.dataf[sta.top]=ans;
                break;
            case 43:
                ans=sta.dataf[sta.top-1]+sta.dataf[sta.top];
                sta.top-=1;
                sta.dataf[sta.top]=ans;
                break;
            case 45:
                ans=sta.dataf[sta.top-1]-sta.dataf[sta.top];
                sta.top-=1;
                sta.dataf[sta.top]=ans;
                break;
            case 47:
                ans=sta.dataf[sta.top-1]/sta.dataf[sta.top];

```

```

        sta.top-=1;
        sta.dataf[sta.top]=ans;
        break;
    default:
        sta.top++;
        sta.dataf[sta.top]=tmp; //采用原始值
        break;
    }
}
return ans;
}

```

```
int SiZeYunSuan_float_all()
```

```

{
    Stackf sta;
    int length;
    float ans;           //接受后缀表达式转换的结果
    float Hou[MAXSIZE]; //存储后缀表达式
    char Zhong[MAXSIZE]; //存储中缀表达式

    printf("-----浮点数四则运算计算器-----\n");
    printf("你可以使用形如 1e1、cos45、(-1) 此类表达式\n");
    printf("输入四则运算表达式（负数添加括号）：");
    scanf("%s",Zhong);
    initStackf(&sta);
    ZhongtoHouf(sta,Zhong,Hou,&length); //后缀到中缀转换

    initStackf(&sta);
    ans=HouJiSuanf(sta,Hou,length);
    printf("%.2f\n",ans);
    return 0;
}

```

```
void menu()
```

```

{
    printf("\n-----CALCULATOR BY 2017202121-----\n");
    printf("1 向量计算（加、减、求余弦）\n");
    printf("2 链表实现多项式计算（加、减、乘、求导）\n");
    printf("3 整数型四则运算计算器\n");
    printf("4 矩阵计算器(加、减、乘、求逆、转置、求行列式)\n");
    printf("5 浮点数四则运算计算器\n");
    printf("6 线性表实现多项式计算（加、减、乘、求导）\n");
    printf("0 退出\n");
}

```

```

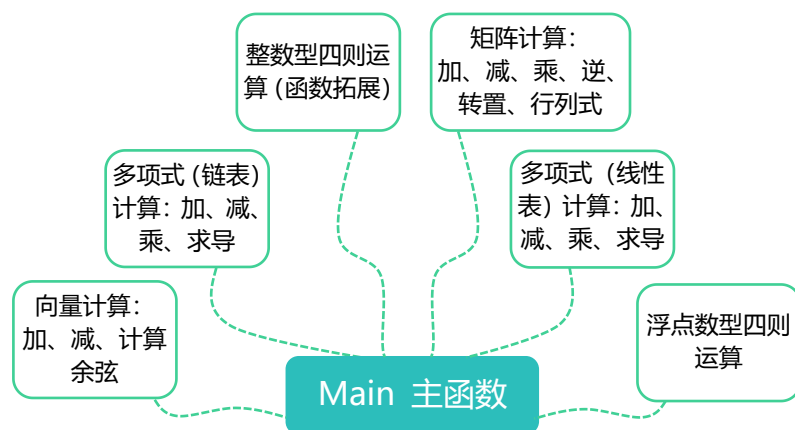
int main()
{
    int k=8;//控制进程

    while(k)
    {
        menu();
        scanf("%d",&k);

        if(k==1) vector_all();
        else if(k==2) DXS_all();
        else if(k==3) SiZeYunSuan_func_all();
        else if(k==4) matrix_all();
        else if(k==5) SiZeYunSuan_float_all();
        else if(k==6) DXS_sql_all();
        else if(k==0) break;
        else{
            printf("输入值无效。 \n");
            k=1;
        }
    }
}

```

7、函数的调用关系图反映了各模块的层次结构



四、调试分析

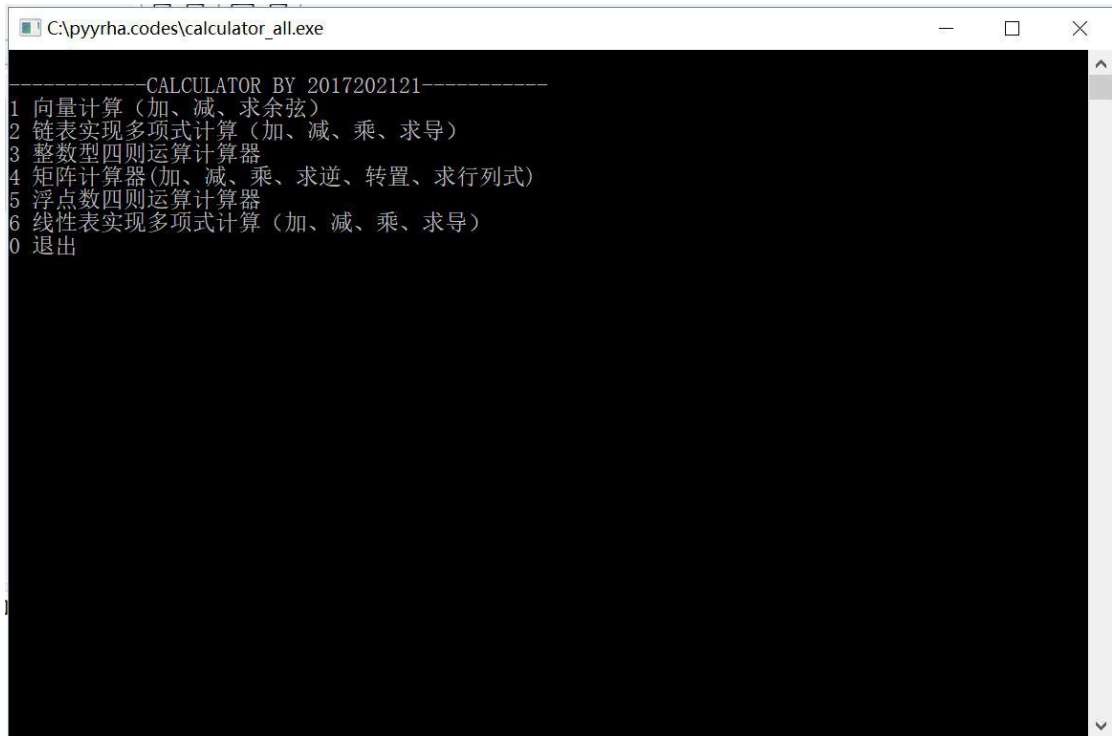
- 1、对各参数输入的检查不足，如果用户输入不符合要求的值，则不能保证会报错并跳转至菜单页面。
- 2、时空分析：采用的都是最经典的存储结构和算法，故时空消耗较大。

3、本程序模块划分较为合理，呈总分结构。

五、用户手册

1、本程序运行环境为 DOS 操作系统，执行文件为 calculator_all.exe

2、进入运行程序后即显示文本方式的交互界面：



```
-----CALCULATOR BY 2017202121-----
1 向量计算 (加、减、求余弦)
2 链表实现多项式计算 (加、减、乘、求导)
3 整数型四则运算计算器
4 矩阵计算器 (加、减、乘、求逆、转置、求行列式)
5 浮点数四则运算计算器
6 线性表实现多项式计算 (加、减、乘、求导)
0 退出
```

3、键入相应的数字，即会跳转到相应的模块。

4、用户根据模块的要求输入指令，程序会自动计算给出相应结果。

六、测试结果

1、向量计算器


```
-----向量计算器-----  
第一个向量：  
维度：  
2  
分量：  
0 1  
第二个向量：  
维度：  
2  
分量：  
1 1  
  
加法结果：  
向量：1 2  
  
减法结果：  
向量：-1 0  
  
余弦值：0.707
```

2、多项式计算器（链表）

```

-----CALCULATOR BY 2017202121-----
1 向量计算(加、减、求余弦)
2 链表实现多项式计算(加、减、乘、求导)
3 整数型四则运算计算器
4 矩阵计算器(加、减、乘、求逆、转置、求行列式)
5 浮点数四则运算计算器
6 线性表实现多项式计算(加、减、乘、求导)
0 退出
2
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
1,0
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
2,1
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
3,3
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
2,4
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
0
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
3,0
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
1,1
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
2,2
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
1,3
请按照“系数,指数”格式、按递增顺序输入,最后输入0结束输入:
0
表达式1:
1x^0+2x^1+3x^3+2x^4
第二个表达式:
表达式2:
3x^0+1x^1+2x^2+1x^3
表达式1 + 表达式2 = 4x^0+3x^1+2x^2+4x^3+2x^4
表达式1 - 表达式2 = -2x^0+1x^1-2x^2+2x^3+2x^4
表达式1求导 = 2x^0+9x^2+8x^3
表达式1 * 表达式2 = 3x^0+7x^1+4x^2+14x^3+11x^4+8x^5+7x^6+2x^7

```

3、整数型四则运算计算器

```

-----CALCULATOR BY 2017202121-----
1 向量计算（加、减、求余弦）
2 链表实现多项式计算（加、减、乘、求导）
3 整数型四则运算计算器
4 矩阵计算器(加、减、乘、求逆、转置、求行列式)
5 浮点数四则运算计算器
6 线性表实现多项式计算（加、减、乘、求导）
0 退出
3
使用细则：
本计算器为整型计算器，支持函数定义功能。
你可以使用‘DEF f(x)=5*x+1’ ‘DEF g(x)=f(x)+4+x/2’ ‘RUN g(4)’ 这样的语句来运行函数
输入：
2*3+3+4/2
计算结果为：11
继续，输入1；离开，输入0：
1
DEF f(x)=5*x+1
你输入的函数名为：f，函数体为：5*x+1
继续，输入1；离开，输入0：
1
DEF g(x)=f(x)+4+x/2
你输入的函数名为：g，函数体为：f(x)+4+x/2
继续，输入1；离开，输入0：
1
RUN f(3)
找到目标函数。
计算结果为：16
继续，输入1；离开，输入0：
1
RUN g(4)
找到目标函数。
计算结果为：27
继续，输入1；离开，输入0：
0

```

4、矩阵计算器

```

-----矩阵计算器-----
请输入两个矩阵，该计算器将会计算两矩阵的加法、减法，
以及第一个矩阵的行列式、转置和逆矩阵
输入第一个矩阵的维度（行*列）
3 3
输入第一个矩阵
1 2 3
2 1 3
3 2 1
输入第二个矩阵的维度（行*列）
3 3
输入第二个矩阵
4 5 6
1 3 5
1 1 2
矩阵的转置为：
1.000 2.000 3.000
2.000 1.000 2.000
3.000 3.000 1.000
行列式为 12
矩阵相加
5.000 7.000 9.000
3.000 4.000 8.000
4.000 3.000 3.000
矩阵相减
-3.000 -3.000 -3.000
1.000 -2.000 -2.000
2.000 1.000 -1.000
矩阵相乘：
9.000 14.000 22.000
12.000 16.000 23.000
15.000 22.000 30.000
行列式为 12
逆矩阵为：
-0.417 0.333 0.250
0.583 -0.667 0.250
0.083 0.333 -0.250
计算完成!

```

5、浮点数四则运算计算器

```

1 向量计算（加、减、求余弦）
2 链表实现多项式计算（加、减、乘、求导）
3 整数型四则运算计算器
4 矩阵计算器（加、减、乘、求逆、转置、求行列式）
5 浮点数四则运算计算器
6 线性表实现多项式计算（加、减、乘、求导）
0 退出
5
-----浮点数四则运算计算器-----
你可以使用形如1e1、cos45、（-1）此类表达式
输入四则运算表达式（负数添加括号）： 1e2+cos45+(-50)
50.71

```

6、多项式（线性表）计算

加法:

$$4x^0 + 3x^1 + 2x^2 + 4x^3 + 2x^4$$

减法:

$$-2x^0 + 1x^1 - 2x^2 + 2x^3 - 2x^4$$

乘法:

$$3x^0 + 7x^1 + 4x^2 + 14x^3 + 11x^4 + 8x^5 + 7x^6 + 2x^7$$

第一个多项式求导:

$$2x^0 + 9x^2 + 8x^3$$

第二个多项式求导:

$$1x^0 + 4x^1 + 3x^2$$