矩阵的运算 实验报告

**一、实验题目解读**

1.能够实现从文件或控制台中读入相应的矩阵，可以在控制台输出显示相应的矩阵。

矩阵数据: (1)可以一个文件存放一个矩阵数据。第一行是矩阵的大小（即矩阵有几行，几列），第二行开始是矩阵元素。(2)也可以是一个文件存放多个矩阵数据（格式自定，但需要把数据文件发给助教，以便测试）。

暂不考虑修改矩阵数据的功能。

2.能够完成矩阵的一般计算（两个矩阵相加、相减、相乘）

给每个矩阵定一个名字，比如：A、B；

用矩阵名参与计算，比如：A+B

需要考虑根据矩阵运算规则，当前两个矩阵是否能参加运算。

3.能够完成矩阵的除法、求逆计算

注意不是所有矩阵可求逆。

4.能够完成矩阵的多个运算符的混合计算。

五种运算（ +，-，\*，/,| ）以及“（）”，“|” 表示矩阵求逆。

括号的优先级最高，其次是求逆，其次是乘除，最后是加减。

能够正确识别表达式，并正确计算混合运算的结果。

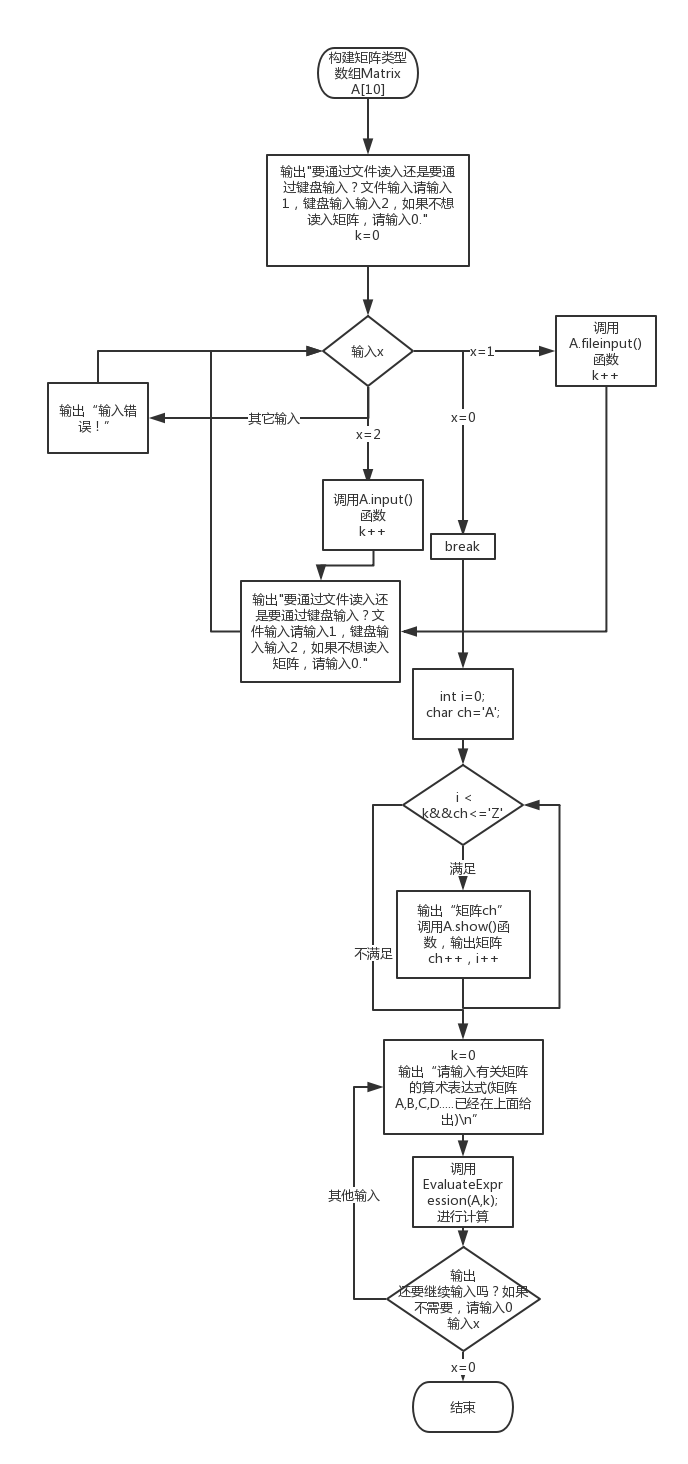
**二、需求分析和答题思路构建**

1.矩阵的运算，矩阵的信息包含行数和列数，以及矩阵本身----->能否构建一个矩阵类，重载一些运算符来进行计算？

2.矩阵的算术表达式求值------------->可以参考之前的算术表达式求值的程序，用栈来实现，只不过其中的int 变量变成了矩阵类Matrix。

3.主函数完成交互。

**三、主函数流程图**



**四、各功能函数说明及代码**

1.矩阵类Matrix的构建

private含矩阵的行数m列数n和一个二维数组，用于储存矩阵的值

public中含有两个输入函数input和fileinput，用于键盘输入和文件输入

一个输出函数show

一个求逆的函数inverse

重载的运算符+ - \* /

代码如下：

class Matrix

{

private:

int m, n;

double A[50][50];

public:

void input();

void fileinput();

void show();

Matrix operator+(Matrix &B);

Matrix operator-(Matrix &B);

Matrix operator\*(Matrix &B);

Matrix inverse();

Matrix operator/(Matrix &B);

};

input函数中，先输入矩阵行数列数，如果输入的值小于1，则提示输入错误，重新输入，再输入矩阵。代码如下

void Matrix::input()

{

for (; ;) {

cout << "请输入矩阵的行数和列数：" << endl;

cin >> m >> n;

if (m < 1 || n < 1) { cout << "输入错误，矩阵的行数和列数必须大于等于1！" << endl; continue; }

else break;

}

cout << "请输入矩阵" << endl;

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

cin >> A[i][j];

}

fileinput函数也是类似的，提示输入文件名，通过文件名打开文件。然而如果出现输入错误，直接退出。

void Matrix::fileinput()

{

string filename;

cout << "请输入文件名" << endl;

cin >> filename;

ifstream infile(filename);

if (!infile) {

cerr << "打开文件" << filename << "失败！"; exit(0);

}

infile >> m >> n;

if (m < 1 || n < 1) { cout << "文件中输入错误，矩阵的行数和列数必须大于等于1！" << endl; exit(0); }

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

infile >> A[i][j];

}

show函数输出矩阵信息，保留两位小数。

void Matrix::show()

{

cout << "矩阵行数：" << m << endl;

cout << "矩阵列数：" << n << endl;

cout << "矩阵：" << endl;

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++)

cout <<fixed<<setprecision(2)<< A[i][j] << '\t';

cout << endl;

}

}

重载运算符+， -，如果两个矩阵行数列数不相等，输出错误信息

Matrix Matrix::operator+(Matrix &B)

{

Matrix X;

if (m != B.m || n != B.n) { cout << "两矩阵的行数或列数不相等，无法相加！" << endl; exit(0); }

X.m = m;

X.n = n;

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

X.A[i][j] = A[i][j] + B.A[i][j];

return X;

}

Matrix Matrix::operator-(Matrix &B)

{

Matrix X;

if (m != B.m || n != B.n) { cout << "两矩阵的行数或列数不相等，无法相减！！" << endl; exit(0); }

X.m = m;

X.n = n;

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

X.A[i][j] = A[i][j] - B.A[i][j];

return X;

}

重载运算符\*

如果前一个矩阵的列数与后一个矩阵行数不相等，输出错误提示。

Matrix Matrix::operator\*(Matrix &B)

{

Matrix X; int i, j, k;

if ( n != B.m) { cout << "前一个矩阵的列数不等于后一个矩阵的行数，无法相乘！" << endl; exit(0); }

X.m = m;

X.n = B.n;

for(i=0; i<X.m; i++)

for (j = 0; j < X.n; j++) {

X.A[i][j] = 0;

for (k = 0; k < n; k++) {

X.A[i][j] += A[i][k]\*B.A[k][j];

}

}

return X;

}

https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D83/sign=ba0522104036acaf5de09bff7dd9d4b7/9345d688d43f8794d5282907d91b0ef41ad53ac7.jpg那么，矩阵应该如何求逆呢？这里，我们使用“如果A可逆，则 ”这个公式，为了判断矩阵是否可逆，我们先使用一个计算行列式的函数getdet函数。

在这个函数中，我们使用了递归的方法计算行列式。getdet函数代码如下

double getdet(double A[50][50], const int n)

{

double temp[50][50];

int i, j, k;

if (n == 1)

return A[0][0];

double det = 0;

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

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

temp[i][j] = 0.0;

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

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

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

temp[i][k] = A[i + 1][(k >= j) ? k + 1 : k];

double t = getdet(temp, n - 1);

if (j % 2 == 0)

det += A[0][j] \* t;

else

det -= A[0][j] \* t;

}

return det;

}

我们知道，一个矩阵的伴随矩阵是它各元素的代数余子式所形成的矩阵的转置，所以，我们可以用类似的方法求得这个矩阵的伴随矩阵，定义函数getAa

代码如下

void getAa(double A[50][50], int n, double Aa[50][50])

{

if (n == 1) Aa[0][0] = 1;

double temp[50][50];

int i, j, k, t;

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

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

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

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

temp[k][t] = A[(k >= i) ? k + 1 : k][(t >= j) ? t + 1 : t];

Aa[j][i] = getdet(temp, n - 1);

if ((i + j) % 2 == 1)

Aa[j][i] = -Aa[j][i];

}

}

在求逆的函数中，我们就可以调用这两个函数进行计算。如果矩阵不是方阵，输出错误信息，如果矩阵不可逆，也输出错误信息。

Matrix Matrix::inverse()

{

if (m != n) cout << "矩阵不是方阵，不可逆！" << endl;

if (getdet(A, n) == 0) cout << "矩阵不可逆！" << endl;

int i, j;

double Aa[50][50]; Matrix X;

X.m = X.n = n;

double det = getdet(A, n);

getAa(A, n, Aa);

for (i = 0; i<X.m; i++)

for (j = 0; j < X.n; j++)

X.A[i][j] = Aa[i][j] / det;

return X;

}

重载运算符/时，我们可以把A/B看成是A\*B的逆，于是又可以调用getdet和getAa两个函数进行计算

Matrix Matrix::operator/(Matrix &B)

{

Matrix b,X;

int i, j, k;

if (n != B.m) { cout << "第一个矩阵的列数不等于第二个矩阵的行数或列数，无法相除！" << endl; exit(0); }

X.m = m;

X.n = B.n;

b=B.inverse();

for (i = 0; i<X.m; i++)

for (j = 0; j < X.n; j++) {

X.A[i][j] = 0;

for (k = 0; k < n; k++) {

X.A[i][j] += A[i][k] \* b.A[k][j];

}

}

return X;

}

这样矩阵的类就建立起来了。

2.矩阵的算术表达式运算

我们选择使用栈进行计算，先建立两个栈，一个用来存放char类型的操作符，另一个用来存放矩阵Matrix。定义与它们相关的函数pop（弹出栈顶元素）, push（压入栈中）, top（获得栈顶元素的值）等等

各个函数代码如下

tatic int sp = 0;

static int sq = 0;

Matrix val[MAXVAL];

char opr[MAXVAL];

void pushnum(Matrix f)

{

if (sq<MAXVAL)

val[sq++] = f;

else

cout << "错误：栈已满1" << endl;

}

void pushopr(char ch)

{

if (sp<MAXVAL)

opr[sp++] = ch;

else

cout << "错误，栈已满" << endl;

}

Matrix popnum()

{

if (sq>0) {

return val[--sq];

}

else {

cout << "错误，栈已空" << endl;

}

}

Matrix topnum()

{

if (sq>0)

return val[sq - 1];

else {

cout << "错误，栈已空1" << endl;

}

}

char popopr()

{

if (sp>0)

return opr[--sp];

else {

cout << "错误，栈已空" << endl;

return 0.0;

}

}

char topopr()

{

if (sp>0) {

return opr[sp - 1];

}

else {

cout << "错误，栈已空" << endl;

return 0.0;

}

}

在计算中，需要使用到如下几个辅助的函数

1）判断是否是操作符的函数IsOperator(char c)

bool IsOperator(char c) //判断c是否为7种运算符之一

{

switch (c)

{

case '+':

case '-':

case '\*':

case '/':

case '(':

case ')':

case '#':

case '\n':

return true;

default:

return false;

}

}

2)判断运算符优先级的函数char Precede(char t1, char t2) 输出三种字符 “<””>” “=”

char Precede(char t1, char t2)

{ //判断t1,t2两符号的优先关系('#'用'#'代替)

char f;

switch (t2)

{

case '+':

case '-':

if (t1 == '(' || t1 == '#')

f = '<'; //t1<t2

else

f = '>'; //t1>t2

break;

case '\*':

case '/':

if (t1 == '\*' || t1 == '/' || t1 == ')')

f = '>'; //t1>t2

else

f = '<'; //t1<t2

break;

case '(':

if (t1 == ')')

{

cout << "括号不匹配" << endl;

exit(0);

}

else

f = '<'; //t1<t2

break;

case ')':

switch (t1)

{

case '(': f = '='; //t1=t2

break;

case '#':

cout << "缺乏左括号" << endl;

exit(0);

default: f = '>'; //t1>t2

}

break;

case '\n':

switch (t1)

{

case '#':

f = '='; //t1=t2

break;

case '(':

cout << "缺乏右括号" << endl;

exit(0);

default: f = '>'; //t1>t2

}

}

return f;

}

3）用于执行操作数的函数Matrix Operate(Matrix A, char oper, Matrix B)

Matrix Operate(Matrix A, char oper, Matrix B)

{ //做运算a theta b,返回运算结果

Matrix X;

switch (oper)

{

case '+':

X = A + B; return X;

case '-':

X = A - B; return X;

case '\*':

X = A\*B; return A\*B;

default:

break;

}

X = A / B;

return A/B;

}

于是，我们可以定义一个用于计算的函数，Matrix EvaluateExpression(Matrix A[10],int k)，在这个函数中，我们需要调用前面的三个函数

Matrix EvaluateExpression(Matrix A[10],int k)

{

//设置操作数栈和操作符栈

Matrix a, b, X, M;

char x;

char c;

if (k == 0) {

c = getchar();

}

pushopr('#');

c = getchar();

x = topopr();

while (c != '\n' || x != '#')

{

if (IsOperator(c))

{

switch (Precede(x, c))

{

case'<':

pushopr(c);

c = getchar();

break;

case'=':

x = topopr();

popopr();

c = getchar();

break;

case'>':

x = topopr();

popopr();

b = topnum();

popnum();

a = topnum();

popnum();

pushnum(Operate(a, x, b));

}

}

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

{

X = A[c - 'A'];

pushnum(X);

c = getchar();

}

else if (c == '|')

{

a = topnum();

popnum();

b = a.inverse();

pushnum(b);

c = getchar();

}

else

{

cout << "出现非法字符" << endl;

exit(0);

}

x = topopr();

}

M = topnum();

M.show();

popnum();

return M;

}

**五、编程中遇到的问题及解决办法**

1.在编写矩阵求逆的程序片段时，我原本是想要重载运算符|，却发现编译不通过，经过上网查找发现|是一个二元运算符，并不能重载为一元运算符。所以我放弃使用了重载的方法，改为一个类成员函数inverse，在进行表达式计算时，把|与其他的运算符区别开来，如果找到|，直接进行独立的求逆计算。

2.在调试过程中，发现有时，EvaluateExpression无法正确运行，第一次运行时正常的，之后会出现栈已空的情况，经检查，发现这个函数的前面出现了两个c=getchar()，即从输入缓冲区连续读了两次。然而，如果删除一个，那么第一次的输入又不正常了。经过分析，发现这是因为之前的输入给输入缓冲区留下了残留的字符，那么第一调用时必须把那个字符跳过，才能正常读取，然而在后来的调用中，输入缓冲区没有残留字符，就不需要再跳过了。于是我在函数参数中添加了一个控制的k，当k=0时（第一次调用）就执行两次getchar（），k!=0时只执行一次。

然而后来添加交互设计的时候我们发现，因为我们在主函数中有一个

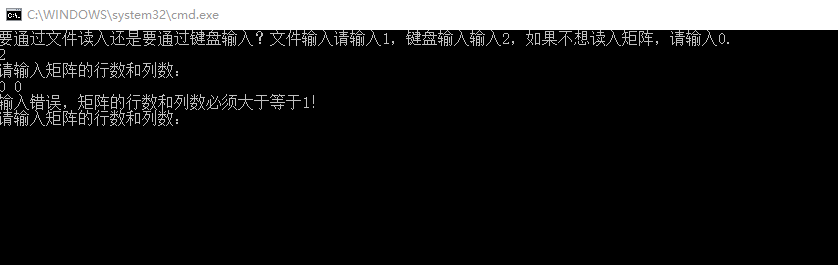
cout << "还要继续输入吗？如果不需要，请输入0" << endl;

cin >> x;

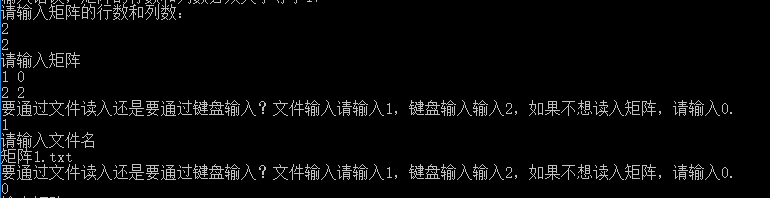
这个也会产生缓冲区残留，所以其实后来的调用也需要用一个getchar来跳过，所以再添加k来控制其实就没有必要了。

当然，更简便的方法是在前面的输入结束后加一个cin.clear（），清除缓冲区。

**六、程序调试过程**



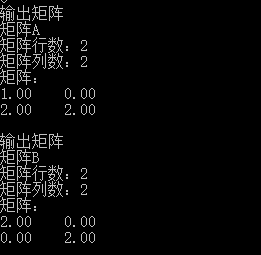
**输入2键盘输入，如果输入行数或列数小于1，提示错误信息，重新输入。**



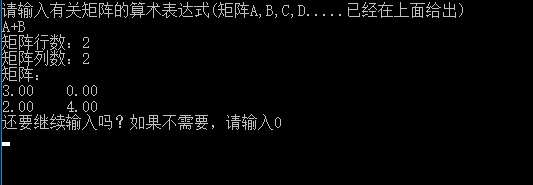
**输入矩阵，**

**输入1时文件输入，再键入文件名。**

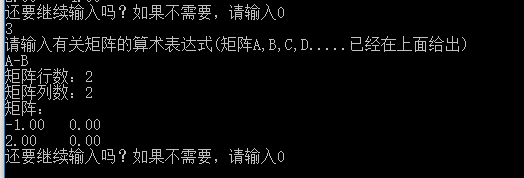
**不想输入时输入0**



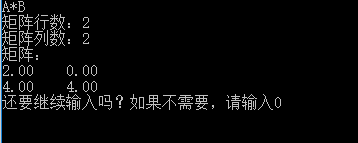
**得到矩阵A、B为刚才输入的两个矩阵**



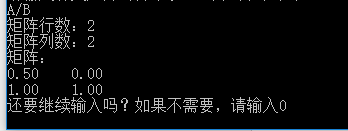
**算术表达式计算A+B**



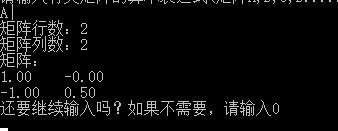
**继续输入 计算A-B**



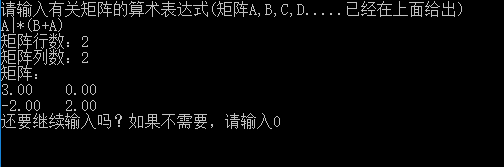
**计算A\*B**



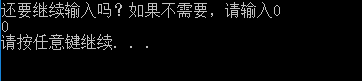
**计算A/B**



**计算A|**



**计算A|\*(B+A)**



**输入0，结束计算**