


Dokumentation für den Benutzer

Mit diesem Programm kann man die verschiedenen Matrixoperationen und die verschiedenen Probleme, die die Matrixoperationen benutzen, leichtmachen. Dieses Programm besteht aus drei Teile. Die unterschiedlichen Teile sind die verschiedenen Funktionen, die in diesem Programm verwirklichen werden. Die Bedienung des Programmes ist einfach.

Erstes Teil ist eine einfache Klasse, die nur Matrixoperationen schnell durchführen kann. Die Hauptfunktionen sind: Multiplizierung, Addieren, Subtrahieren mit Matrix oder eine Zahl, Transponierung. Die Matrix wird von einer Datei gelesen. Leider muss der Benutzer das folgendes Text in den Code schreiben, aber es ist nicht so schwierig:

```
ifstream file2;
ifstream file3;
file2.open("matrix1.txt");
file3.open("matrix2.txt");
```

Eine richtige Datei sieht so aus:

 matrix1 - Jegyzetömb

Fájl Szerkesztés Formátum Nézet Súgó

```
4
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

„ifstream“ ist nötig, wenn wir die Datei öffnen möchten. Wenn wir aus mehreren Datei Matrizen auslesen möchten, dann müssen wir eine andere ifstream dazu schreiben, zum Beispiel: „ifstream file4; file4.open(„matrix3.txt“)“. Es ist wichtig, dieses Syntaxis einzuhalten. Wenn man der Name der Datei falsch schreibt, ist es kein großes Problem, das Programm kann es handeln und wird es terminieren. Diese Regel

auch gültig für andere Funktionen. Wenn wir diese Datei eingegeben haben, darf man auf „Local Windows Debugger“ klicken:



Bei den Operationen der Matrizen ist es möglich kleines Fehler zu machen. Zum Beispiel bei der Multiplizierung zwei Matrizen. Nehmen wir 2 Matrizen, A und B. Wenn die Anzahl der Spalte von A nicht gleich mit der Anzahl der Zeile von B ist, dann ist es wegen Definition nicht möglich, diese Operation zu lösen. In diesem Fall wird durch das Programm ausgeschrieben, dass die Bedingungen der Matrix Multiplikation nicht erfüllt sind. Es ist kein großes Problem, denn das Programm kann es handeln, es wird einfach terminieren.

Mit „cout<< „name“;“ können wir die Matrix ausschreiben, die das Programm ausgerechnet hat. Die andere Teile des Programms sind nicht so wichtig für den Benutzer(try-catch usw.).

Beispiel Programm mit der Ausgabe:

```
int main() {
    try {
        ifstream file2;
        ifstream file3;
        file2.open("matrix1.txt");
        file3.open("matrix2.txt");

        matrixop<double> g2(file2);
        matrixop<double> g3(file3);
        g2 = g2 * g3;
        cout << g2;
    }
    catch (const exception &e) {
```

```
84 96 108 120
212 248 284 320
340 400 460 520
468 552 636 720
```

Das zweite Teil des Programms kann ein Gleichungssystem lösen. Die Matrix wird auch aus einer Datei gelesen aber hier muss man darauf achten, dass die Matrixform den „b“ Vektor auch enthalten muss, also die Matrix hat n unbekannte aber n+1 Spalte! Es ist sehr wichtig, weil ohne diese plus Spalte wird das Programm nicht laufen. Die einzelne Spalte sind die Unbekannte und die einzelne Zeile sind die Gleichungen. Natürlich ist es möglich mehr Gleichung als Unbekannte eingeben und umgekehrt aber wenn es mehr Unbekannte als Gleichung gibt, muss man die einzelnen Lösungen eingeben. (Das Programm wird diese Lösungen in Betracht nehmen und dann die Lösung ausrechnen.)

Diese Matrixform und eine gültige Datei sieht so aus:

$$x_1 \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix} + x_2 \begin{pmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{pmatrix} + \dots + x_n \begin{pmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

gleichung - Jegyzetömb

Fájl	Szerkesztés	Formátum	Nézet	Súgó
4				
4				
0 2 8 9 6				
3 14 2 5 5				
7 8 3 6 4				
12 56 8 8 2				

Dieses Programm kann ganz viele Probleme lösen, bei einem Gleichungssystem ist es möglich ,triviale Zeile zu enthalten.(während der Gauss-Elimination bildet sich eine Zeile mit 0 Elemente heraus). Wenn es mehrere Gleichungen als Unbekannten gibt, wird das Programm die Zeilen umtauschen, andernfalls ist das Gleichungssystem nicht lösbar. So sieht eine gültige Code und die Lösungen aus:

```
int main() {
    try {
        ifstream file;
        file.open("gleichung.txt");
        gleichung<double> g(file);
        g.loesen();
        cout << g;
    }
}
```

```
0 2 8 9 6
3 14 2 5 5
7 8 3 6 4
12 56 8 8 2

3 14 2 5 5
0 2 8 9 6
0 0 97 105.333 66.3333
0 0 0 -12 -18

Die Loesungen:
x[1]: -0.343643
x[2]: 0.0300687
x[3]: -0.945017
x[4]: 1.5
```

Wir bekommen 2 Matrizen, erste ist die originelle Matrix, zweite ist diese Matrix nach der Gauss-Elimination und die Lösungen.

Drittes Teil des Programms ist eine Vektor Klasse. Dieses Programmteil kann die verschiedenen Transformationen mit einem Vektor durchführen. Jede Klasse gehört 2 Vektor: der originelle Vektor und der abgebildete Vektor. Diese Transformationsmöglichkeiten sind: Rotieren um x,y,z Achse, Projizieren oder Spiegelung um xy,xz,yz Ebene, Streckung(x,y,z). Bei der Rotation muss man ein Grad auch eingeben, für andere muss man nur die Funktion rufen. Es ist nicht egal, ob wir erstens Rotieren oder Spiegeln und umgekehrt, die abgebildete Vektoren werden nicht gleich. Im Hintergrund wird dieses Programmteil mit Matrixmultiplikation gelöst.

So sieht eine gültiges Kode und die Ausgabe aus:

```
try {
    Vector<double> v(1, 2, 3);
    v.Rotx(30);
    cout << v;
    v.Spiegxy();
    cout << v;
}
```

Originelle Vector:
(1,2,3)
Abgebildete Vector:
(1,0.233006,3.59801)
Originelle Vector:
(1,0.233006,3.59801)
Abgebildete Vector:
(2,2.23301,-0.598014)

Im ganzen Programm können wir eingeben, mit welchen Zahlen das Programm arbeiten dürfen. Ganze Zahlen, Bruchzahlen, long Zahlen, long long Zahlen, unsigned Zahl (≥ 0) usw. Diese Zahlenformen können wir mit <..> auswählen. Bei der Ganze Zahlen: <int>, Bruchzahlen: <double>, unsigned Zahlen: <unsigned> usw. Es ist auch ein wichtiges Teil des Programms. Es ist nicht egal welches Zahltyp wir eingeben. Das Unterschied möchte ich mit einem Beispiel demonstrieren. Dieses Programm wird mit den ganzen Zahlen rechnen, und sieht die drittes (Vektor) Teil in diesem Fall so aus:

```
try {
    Vector<int> v(1, 2, 3);
    v.Rotx(30);
    cout << v;
    v.Spiegxy();
    cout << v;
}
```

Originelle Vector:
(1,2,3)
Abgebildete Vector:
(1,0,0)
Originelle Vector:
(1,0,0)
Abgebildete Vector:
(2,2,3)

Warum ist der Unterschied so groß? Das Programm wird die Bruchzahlen abrunden. also 0,95 wird 0 und 1,2 ist 1. In dem dritten Programmteil ist es besonders wichtig, keine int (ganze Zahlen) zu benutzen, weil es mit cosinus und sinus arbeitet und in den meisten Fällen werden diese Zahlen <1 und das Programm wird diese auf 0 abrunden.

Während des Programms ist es möglich, etwas falsch zu machen.

Diese Fehlertypen sind:

- Indexierung Fehler(bei Matrix):

Error: Falsche Indexierung

- Datei Fehler(nicht gültige Datei):

Error: Nicht valid File

- Etwas Bedingung nicht erfüllt wird (bei Matrixmultiplikation):

Error: Die Bedingungen der Matrix Multiplizierung werden nicht erfuehlt!