mtx.c

```
Matrix A = read("./a.mtx");
Matrix B = read("./b.mtx");

printf("Determinant of A: %.3f", det(A));

Matrix mult = multiply(A, B);
write(mult, "./mult.mtx");
```

mtx egy könyvtár C nyelvben ami könnyűvé teszi mátrixok írását olvasását és az azokon végzett egyszerű és kevésbé egyszerű műveleteket. A könyvtár próbálja követni a **MATLAB**-ben mátrixoknál használt konvenciókat.

A projekt fejlesztésének folyamata és eredménye végigkövethető a program **GitHub** oldalán.

Típusok

Values

A **`Values`** típus egy dinamikusan lefoglalt 2 dimenziós tömb, ami **double**-ket tartalmaz (jövőben lehetőség nyílhat akár más típusok használata is). Ezzel lehet tárolni a mátrix és tömb értékeit.

Matrix

Ez a típus a lényeges típus, amit a legtöbb függvény használ.

```
struct Matrix {
  int n;       // A sorok száma a mátrixba
  int m;       // A oszlopok száma a mátrixba
  Values vals; // A mátrix értékei
}
```

Row

Ez típus használandó amikor egy sort akarunk megkapni, vagy azon műveleteket végezni.

```
struct Row {
  int n;       // A sor mérete
  Values vals; // A sor értékei
}
```

Olvasás

Lehet mátrixokat olvasni fájlokból, de egy fájl csak egy mátrix adatait tartalmazhatja. Például egy 3-szor 3-as mátrix az **`a.mtx`** (a .mtx nem kötelező csak könnyebbé teszi a dolgokat) tartalma így néz ki:

```
[2 3 4; 2 1 0; 0 0 1]
```

Olvasni fájlból mátrixota **read** függvénnyel lehet.

```
Matrix a = read(char *fajl_eleresi_ut);
```

Írás

Mátrixokat fájlba írni ugyanabban a formátumban a **write** függvénnyel lehet.

```
void write(Matrix A, char *fajlnev);
```

Függvények

create

Visszatér egy **0-ra** inicializált **n** soros **m** oszlopos mátrix-szal.

```
Matrix create(int n, int m);
```

identity

Visszatér egy **n** soros **m** oszlopos **identitás** mátrix-szal.

```
Matrix identity(int n, int m);
```

add

B mátrixot hozzáadja az **A** mátrixhoz. Ha az összeadás nem lehetséges, akkor semmi sem történik.

```
Matrix add(Matrix A, Matrix B);
```

scale

Megszorozza **lambda** skalárral az **A** mátrixot.

```
void scale(Matrix A, double lambda);
```

multiply

Megszorozza az **A** mátrixot **B** mátrixszal és visszatér az eredménnyel.

```
Matrix multiply(Matrix A, Matrix B);
```

exchange

Kicseréli az **A** mátrix **i.** és **j.** sorát.

```
void exchange(Matrix *A, int i, int j);
```

row

Visszatér **A** mátrix **i.** sorával.

```
Row row(Matrix A, int i);
```

scale_row

Megszorozza **lambda** skalárral **r** sort.

```
void scale_row(Row r, double lambda);
```

add_row

Hozzáadja **r** sort az **A** mátrix **i.** sorához.

```
void add_row(Row r, int i, Matrix A);
```

rref

Az **A** mátrixot **reduced row echelon form**-ba teszi.

```
void rref(Matrix A);
```

rank

Visszatér az **A** mátrix **rang**jával.

```
int rank(Matrix A);
```

dim

Visszatér az **A** mátrix **dimenziójával**.

```
int dim(Matrix A);
```

transpose

Transzponálja az **A** mátrixot.

```
void transpose(Matrix A);
```

det

Kiszámolja az **A** mátrix **determináns**át.

```
double det(Matrix A);
```

dup

Duplikálja az **A** mátrixot.

```
Matrix dup(Matrix A);
```

Ütemterv

Ezek a további fontos függvényeket a jövőben elkészítendőek.

inv

Kiszámolja az **A** mátrix inverzét.

```
Matrix inv(Matrix A);
```

eig

Kiszámolja az **A** mátrix sajátértékeit.

```
double[] eig(Matrix A);
```