Dokumentáció

Projekt felépítése

main.c (a program főmodulja)

Ebben a fájlban mennek végbe azok a függvényhívások, mely függvényeknek a kódjai majd az alább látható két (.c) fájl tartalmaz. A *main.c*-ben főleg egy hátul tesztelő ciklus található, melyben egy több elágazással rendelkező szelekció (*switch*) van. Előbbi újra-újra előhozza nekünk a menüt, ameddig a felhasználó nem szeretne kilépni (11-es menüponttal). Míg utóbbival az adott menüpont utasítását hajtja végre a program függvényhívásokkal.

inout.c (almodul)

Nevében is található egy utalás az input és output szerepére ennek a modulnak. Főként ez a modul kommunikál a felhasználóval, itt történik meg az adatok bekérése is.

Függvényei:

- *Felszabadit()*.....Felszabadít egy strukturált típusú változó *matrix* nevű kétdimenzióstömbjét.
- *SzamBeolvas()* .....Beolvas egy számot a bemeneti hibákat kezelve.
- *Menu()* .....Kilistázza a menüpontokat, eltárol egy menüpontot.
- *feltolt()*.....Feltölt egy kétdimenziós tömböt (mátrixot).
- *kiir()*.....Fájlba ír egy strukturált típusú változó adatait, a mátrix adatait.
- *nullVektor()* .....Ellenőrzi, hogy a felhasználó egy üres mátrixot szeretne-e létrehozni: nulla oszlopos, nulla soros mátrixot.
- *BeolvasKiir()* .....Beolvas és kiír egy mátrixot.
- *olvas()*.....Fájlból visszaolvas egy mátrixot, feltölti azt elemeivel.
- *BeolvasFajl()*.....Fájlból beolvassa egy mátrixnak az adatait.

inout.h (almodul fejléc)

Tartalma az *inout.c* modul függvénydeklarációi és típusdefiníciója. Fontos, hogy csak azok a függvények deklarációi vannak itt, amelyek más *.c fájl* használatához kellenek. (Általánosan kifejtve: amelyek nem ilyenek, azokat az adott *.c fájlban STATIC* szóval vannak „címkézve” a függvény típusa előtt).

mmuvelet.c (almodul)

A mátrix műveleteket kezeli.
Függvényei:

- *feltoltottMX()* .....Eldönti egy mátrixról, hogy valaha feltöltésre került-e.
- *azonosTipusuak()*.....Eldönti, hogy két mátrix azonos méretű-e.
- *megjelenit()*.....Képernyőre ír egy mátrixot.
- *Osszeadas()* .....Összead két mátrixot.
- *Kivonas()* .....Kivon egymásból két mátrixot.
- *SzorzasValossal()* .....Beszoroz adott valós számmal egy mátrixot, majd megjeleníti azt képernyőn.
- *szorzatMatrixElem()* ....Kiszámolja egy szorzatmátrix adott sor és oszlopában lévő elemet.
- *Szorzas()* .....Összeszoroz kettő mátrixot.
- *sorOsztas()*.....Determináns számításakor leoszt egy adott sort annak a mátrixnak a főátlójában található értékkel vezéregyest képezve ott.
- *sorKivonas()* .....Determináns számításakor kinullázza a vezéregyes alatt található számokat azon sorok kivonásával.
- *vanNemNullaElem()* ....Determináns számításakor ellenőrzi, hogyha egy főátlóban nulla található, akkor létezik-e nem nulla elem annak az oszlopában.
- *sorCsere()*.....Determináns számításakor kicserél kettő sort.
- *Determinans()*.....Kiszámítja egy mátrix determinánsát.

mmuvelet.h (almodul fejléc)

Tartalma a *mmuvelet.c* modul függvénydeklarációi.

debugmalloc.h (dinamikus memóriakezelés)

Olyan függvényeket és kódokat tartalmaz, mely segíti a programozót a hibamentes és jól kezelt dinamikus memóriafoglalással kapcsolatban. Főbb függvényei, amit használ a program: *malloc()*, *free()*.

A program nem igényel szabványos C típusú nyelven kívüli függvénykönyvtárat.

Adatszerkezetek

A típusdefinícióval egy olyan strukturált típust hozunk létre, melynek a következő mezői vannak: (valós számokat tároló) *DOUBLE* típusú *kétdimenziós mátrix*, (egész számokat tároló) *INT* típusú *sor és oszlop nevű változó*. A nevek utalhatnak szerepükre. A mátrixban valós számokat fogunk tárolni, mely mátrixnak a magassága és szélessége adott a *sor* és *oszlop* nevezetű változóval.

Ha később nem kerül rá sor, fontos itt megjegyezni, hogy a mátrixunk egy dinamikusan lefoglalt kétdimenziós tömb, azaz kizárólag nála kell majd csak használni a *debugmalloc.h* függvényeit: *malloc()*, *free()*.

```
typedef struct MX{
    double **matrix;
    int sor, oszlop;
}MX;
```

- *double* ~ valós szám (pl.: 42,42)
- *int* ~ egész szám (pl.: 5)

Az adatszerkezet, struktúra neve a program során *MX*.

Függvények (feladat, param., visszatér., körülmények)

A függvények során látható azok típusai előtt egy *STATIC* „címke”, mely azt a cél szolgálja mindig, hogy csak a hozzátartozó, vagyis amiben meg lett írva .c fájl láthassa, más fájl ne.

*void Felszabadit (MX *matrix)*

Lefoglalt memóriaterületet szabadít fel. Paramétere egy strukturált típusú változóra mutató pointer, melynek *matrix* nevezetű mezőtagját, egy kétdimenziós tömbjét szabadítja fel a függvény.

*double SzamBeolvas (int min, int max, bool van_tart, char *utast1_szoveg, char *utast2_szoveg)*

Talán a program függvényei közül a legtöbb paramétert tartalmazó függvény, viszont ennek ellenére a legtöbb hibát kezelő is számbeolvasás terén. Maga a függvény tehát a **felhasználótól kér be egy számot, majd ellenőrzi, hogy a bekért szám megfelelő-e**. **Szám beolvasáskor mindig ezt a függvényt hívja meg a program**. A függvény addig fut, ameddig a felhasználó nem ad meg egy helyes számot a programnak.

Paraméterei a következők:

1. Egész szám típusú változó, mely a bekérendő szám tartományának minimumát tárolja. A bekért számnak ennél nagyobbnak kell lennie.
2. Egész szám típusú változó, mely az előbbihez hasonlóan a tartomány maximumát tárolja. A bekért számnak ennél kisebbnek kell lennie.
3. Logikai érték, mely IGAZ, ha a bekérendő számnak a fent említett számtartomány között kell mozognia. HAMIS, ha nincs megszabva, hogy milyen értékek között kell lennie a számnak.
4. Karaktertömb, mely tartalmazza a felhasználó felé továbbított elsődleges utasításokat.

5. Karaktertömb, mely tartalmazza a felhasználó felé továbbított másodlagos utasításokat. Akkor lép életbe, ha az elsődlegest nem vette figyelembe (nem teljesítette) a felhasználó.

Visszatérési értéke egy valós szám, mely átment minden feltételen a függvény futása során, tehát egy megfelelő számot ad vissza mindig.

int Menu ()

A függvény kilistázza a menüpontokat és bekér egy egész számot (menüpontot). A szám bekérésnek a hogyan kérdésre a válaszát az előző felsoroláspont tartalmazza. **Visszatérési értéke** egy megfelelő menüpont, egész szám.

*static double **feltolt (int szeles, int magas)*

A függvény **feltölt double típusú adatokkal egy kétdimenziós mátrixot** a paraméterként kapott *szeles* és *magas int* típusú változók függvényében. Fontos, hogy a mátrix, amelyet feltöltünk egy dinamikusan kezelt tömb, melyet a *malloc()* függvény segítségével foglalunk le a memóriában. **Visszatérési értéke** a feltöltött dinamikusan kezelt kétdimenziós mátrix első elemének a címe lesz, hiszen egy tömbökre mutató pointer címét adjuk vissza.

*static bool kiir(MX *paramMX, char *fajlNev)*

Kiírja fájlba a paraméterként kapott pointer által mutatott struktúra típusú mátrix adatait. A keletkező fájl neve megegyezik a paraméterként megadott fájlnévvel. Ugyanakkor egy *bool* típusú logikai **visszatérési érték**kel is rendelkezik, ami a következő célt szolgálja: Ha sikerült létrehozni a fájlt, akkor értéke *TRUE*, ellenkező esetben *FALSE*.

static bool nullVektor (int x, int y)

Ellenőrzi két számról, hogy legalább az egyikük-e nulla. A függvény segítségével kiszűrhetjük, ha a felhasználó egy üres mátrixot szeretne létrehozni.

Paraméterei kettő egész szám, amelyeket vizsgálunk. **Visszatérési értéke** egy logika IGAZ, ha legalább az egyik szám nulla, és HAMIS, ha egyik szám sem nulla.

*void BeolvasKiir(MX *matrix, char *fajlNev)*

A függvény során bekérjük a felhasználótól az adott mátrix sor és oszlop számait, majd itt **meghívjuk azokat a függvényeket, amelyek a mátrix feltöltéséért és kiíratásáért felelnek** (lásd *feltolt()* és *kiir()*). Lényeges kód itt az, hogy a *feltolt()* nevezetű függvényben dinamikusan lefoglalt kétdimenziós tömböt felszabadítsuk, hiszen már a menüpont során (1-es/2-es) nem dolgozunk vele. Így meghívásra kerül a *Felszabadit()* függvény, amely felszabadítja nekünk a memóriában a dinamikusan lefoglalt területet. **Paraméterei** egy mátrixra mutató pointer és a létrehozandó fájl neve karaktertömbként.

*static double **olvas (char *fajlNev, int *sor, int *oszlop)*

Egy darab fájlból beolvassa egy mátrix adatait. **Paraméterei** egy karaktertömb, amely tartalmazza a fájl nevét és kettő egész számokra mutató pointer. Utóbbiakra azért lesz szükségünk, hogy a fájl első sorából kiolvasott mátrix sorainak és oszlopainak a számát a függvény paraméter listáján keresztül pointer segítségével visszaadjuk.

Visszatérési értéke egy valós számokat tároló kétdimenziós tömb, mely tartalmazza a fájlból beolvasott elemek értékeit.

*void BeolvasFajl(MX *matrix, char *fajlNev)*

Fájlból beolvas és feltölt egy darab mátrixot. A függvény során igazából egyszer meghívásra kerül az előző felsorolásban leírt függvény. Használatára egy másik fájlban kerül majd sor, ez a *main.c* lesz. Itt szeretnénk visszaolvasni a mátrixot. Minden egyes mátrix művelet előtt meghívjuk ezt a függvényt (ha kell, kétszer mindkét mátrixra), hogy a legutóbb eltárolt (fájlba kiírt) mátrixo(ka)t visszaolvassuk és eltároljuk dinamikusan. A mátrix műveletek befejezése után kerül meghívásra a *Felszabadit()* nevezetű függvény. **Paraméterei** egy mátrixra mutató pointer és a létrehozandó fájl neve karaktertömbként.

---

static bool feltoltottMX(MX matrix)

Eldönti a paraméterként kapott strukturált mátrixról, hogy valaha feltöltésre került-e. Ez azt jelenti, hogy fájlból megpróbálja beolvasni a mátrixot előtte egy függvénnyel, és, ha az a fájl nem is létezik, magyarán a felhasználó még egyszer sem indította el a programot, akkor biztos, hogy feltöltetlen.

Visszatérési értéke logikai IGAZ, ha feltöltésre került.

static bool azonosTipusuak(MX matrixA, MX matrixB)

Eldönti a paraméterként kapott strukturált mátrixokról, hogy azok sorainak és oszlopainak száma megegyezik-e. **Visszatérési értéke** logikai IGAZ, ha megegyeznek.

static void megjelenit(MX matrix)

Megjelenít egy paraméterként kapott strukturált mátrixot. Képernyőre írja annak az adatait: sorainak és oszlopainak számát, a mátrix értékeit mátrix-alakban.

void Osszeadas(MX matrixA, MX matrixB)

Összeadja a paraméterként kapott strukturált mátrixokat. A műveletet megelőzi pár hibakezelés is. Történetesen, ellenőrzi, hogy feltöltött mátrixokkal dolgozunk-e (*feltoltottMX(…)*) és az összeadás miatt ellenőrizni kell, hogy a mátrixok mérete azonos-e (*azonosTipusuak(…)*). Az eredményt képernyőn megjeleníti (*megjelenit(…)*).

void Kivonas(MX matrixA, MX matrixB)

Kivonja egymásból a paraméterként kapott strukturált mátrixokat. A műveletet az előzőhez hasonlóan hibakezelés előzi meg. Az eredményt képernyőre írja.

void SzorzasValossal(MX matrix)

Beszorozz egy bizonyos mátrixot egy bizonyos valós számmal. Paraméterei között meg kell adni egy struktúra típusú változót, mely mátrix elemeit szorozzuk egy paraméterként kapott valós számmal, melyet a felhasználótól olvasunk be (*SzamBeolvas(…)*). A függvény továbbá meg is jeleníti az eredményt. Hibakezelést is végez, ellenőrzi, hogy van-e feltöltött mátrix.

*static double szorzatMatrixElem(int j, double *sorA, MX B)*

Kiszámolja egy adott sor és oszlop indexű mátrixszorzat elemét. A sor indexre azért nincsen szükségünk, hiszen a függvény **második paramétere** elvárja, hogy az elsőtényezős mátrix adott sorában lévő elemeket adjuk oda egy egydimenziós valós számokat tároló tömbként. Míg **harmadik paramétertől (**másodiktényezős mátrixtól**)** sajnos nem lehet elvárni. (Utóbbi oka, hogy nem oszlop vektorokkal dolgozik a program, hanem sor.) Így tehát az **első paramétere** egy egész szám, mely a mátrixszorzat oszlop indexét tárolja.
Visszatérési értéke pedig egy valós szám, ez a kiszámolt egy darab érték.

void Szorzas(MX matrixA, MX matrixB)

Összeszoroz kettő paraméterként kapott strukturált mátrixot a paraméter listában megadott sorrendben. (Hiszen AxB nem egyenlő BxA.)

*void sorOsztas(MX *matrix, int adottSor)*

Paraméterként kapott strukturált **mátrix egy egész sorát leosztja** a főátlóban található értékkel vezéregyest képezve, majd a (keletkezett) **módosított mátrix címét visszaadja** a paraméterben kapottnak. **Második paramétere** jellemzi, hogy éppen melyik sort kell majd leosztani.

*void sorKivonas(MX *matrix, int adottSor)*

Paraméterként kapott strukturált mátrixban **a paraméterben megadott adott sor alatt levő sorokkal kivonást végez** a következő képlet szerint: $a_{i,j} = a_{i,j} + (-1) * a_{i,t} * a_{t,j}$. Ahol **ai,j** az éppen aktuális sor és oszlopban található elem (ez mindenképp vezéregyes alatt található sor), **ai,t** az aktuális sorban a vezéregyes oszlopában található elem és **at,j** a vezéregyes sorában éppen aktuális oszlopában található elem. A sor kivonások után a függvény meghívás helyén a **módosított strukturált mátrixot kapjuk vissza cím szerint**.

*bool vanNemNullaElem(Mx matrix, int adottSor, int *nemNullaSor)*

Paraméterként kapott strukturált mátrixban **ellenőrzi, hogy a paraméterként kapott főátló oszlopában található-e nem nulla elem**, ha igen, akkor azt cím szerint visszaadja a harmadik paraméterben megadott változónak. (Megtévesztő lehet az elnevezés, de mivel mindig egy négyzetes mátrixról van szó, így a főátlóbéli elem oszlopindexe mindig egyenlő a sorindexével: *adottSor*). A függvény **visszatérési értéke** logikai IGAZ, ha található nullától különböző érték az adott oszlopban. Továbbá, ha volt ilyen elem, annak a sor indexét cím szerint odaadja a **harmadik paraméternek**.

*void sorCsere(MX *matrix, int adottSor, int nemNullaSor)*

Sor cserét hajt végre. A **paraméterként** kapott strukturált mátrix címe alapján sor cserét hajt végre a második és harmadik paraméterben található sorindex alapján. A művelet után a függvény hívása helyén **visszatéríti** a módosított mátrixot cím szerint paraméterként.

void Determinans(MX matrix)

Kiszámolja a paraméterként kapott mátrix determinánsának értékét. A függvény részben a Bevezetés a számításelméletbe c. tárgy hivatalos jegyzetében található pszeudokód alapján készült. Továbbá itt kerülnek meghívásra azok a függvények, amelyek egy sor cseréért, kivonásáért, leosztásáért felelnek, továbbá az is, mely ellenőrzi, hogy van-e nullától különböző elem adott főátló oszlopában. Az eredményt képernyőre írja.