# Mátrixok, mutatók

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020 szeptember 28.

# Mátrix tárolása vektorban

Adott egy $M \times N$ méretű mátrix, amit tárolni szeretnénk. Ehhez gyakran kényelmes vektorokat és mutatókat használni.

Adattípus definiálásához szükséges:

- két változó a méret tárolására: `cols`, `rows`
- egy $M \times N$ `double` tárolására elegendő memóriaterület
- a mátrixot tároló memória területre mutató `double *mat` pointer
- indexeljük a mátrix sorait az `i`, oszlopait a `j` változókkal

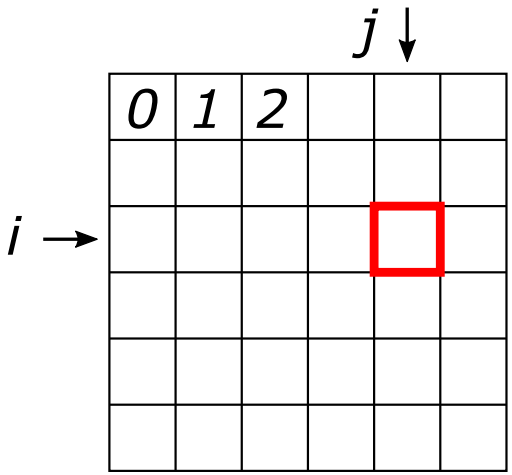
# Mátrix tárolása vektorban

Adott egy $M \times N$ méretű mátrix, amit tárolni szeretnénk. Ehhez gyakran kényelmes vektorokat és mutatókat használni.

Adattípus definiálásához szükséges:

- két változó a méret tárolására: `cols`, `rows`
- egy $M \times N$ `double` tárolására elegendő memóriaterület
- a mátrixot tároló memória területre mutató `double *mat` pointer
- indexeljük a mátrix sorait az `i`, oszlopait a `j` változókkal

Mit kell `mat[ ... ]` indexelésekor a zárójelek közé írni?



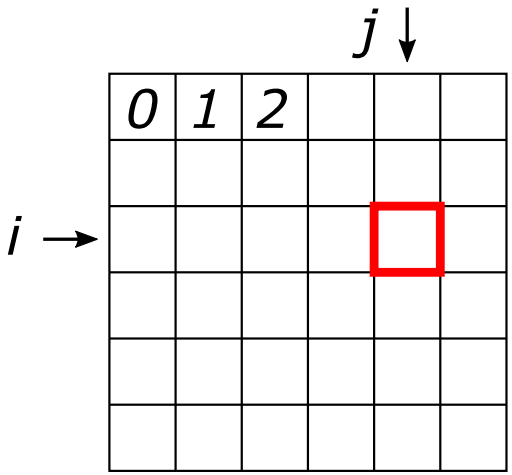
# Mátrix tárolása vektorban

Adott egy $M \times N$ méretű mátrix, amit tárolni szeretnénk. Ehhez gyakran kényelmes vektorokat és mutatókat használni.

Adattípus definiálásához szükséges:

- két változó a méret tárolására: `cols`, `rows`
- egy $M \times N$ `double` tárolására elegendő memóriaterület
- a mátrixot tároló memória területre mutató `double *mat` pointer
- indexeljük a mátrix sorait az `i`, oszlopait a `j` változókkal

Mit kell `mat[ ... ]` indexelésekor a zárójelek közé írni?




A mátrix soronkénti (row-major) tárolása esetén az i,j elem:

```
m[i * cols + j]
```

# Pédafüggvények mátrixok kezelésére

A vektorok kezelésére használt függvényekhez hasonlóan:

```
1  double *alloc_matrix(int cols, int rows) { //memoria foglalo fuggveny
2    double *matr = (double*)malloc(cols * rows * sizeof(double));
3    if (matr == 0) {
4      printf("Memory allocation error.\n");
5      exit(-1);
6    }
7    return matr;
8  }
9
10 void read_matrix(FILE* f, double *m, int cols, int rows) { //matrix elemeit beolvaso fuggveny
11   for (int i = 0; i < rows; i++) {
12     for (int j = 0; j < cols; j++) {
13       fscanf(f, "%lf", &m[i * cols + j]);  //egyszeru beolvasas fscanf-fel
14     }
15   }
16 }
17
18 void write_matrix(FILE* f, double  *m, int cols, int rows) { //matrix kiirasa
19   for (int i = 0; i < rows; i++) {
20     for (int j = 0; j < cols; j++) {
21       fprintf(f, "%f ", m[i * cols + j]);
22     }
23     fprintf(f, "\n");
24   }
25 }
```

# Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

# Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor legfeljebb `maxstrl` karaktert olvasunk be soronként

# Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor legfeljebb `maxstrl` karaktert olvasunk be soronként
- ha az első karakter '#' akkor azt a sort írja ki

# Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor legfeljebb `maxstrl` karaktert olvasunk be soronként
- ha az első karakter '#' akkor azt a sort írja ki
- ha a sor nem üres és van még hely a lefoglalt adatvektorban, akkor alakítsa a stringet `float` számmá

# Mátrix beolvasása szöveges fájlból soronként

A példaprogram:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


double *alloc_vec(int n){
  double *v = (double *)malloc(n * sizeof(double));
  if (v == 0) { printf(" Not enough memory\n");
  exit(-1);}
 return v ;
 }


int main(int argc, char *argv[])
 {
  int maxstrl=300;
  int vektindx=0;
  int maxdim=200;
  char str[maxstrl];

  double *matr=alloc_vec(maxdim);

  FILE *finp = fopen(argv[1],"r");

   if(finp == NULL) {
      printf("Error opening input file");
      return(-1);}
   while (fgets(str,maxstrl,finp)!=NULL )
   {
     if (str[0]=='#')
     {printf("%s",str);}
     else if (strlen(str)>1)
     {
      if (vektindx>maxdim)
      {
       printf("Max vector size reached\n");
       exit(-1);
      }
      else {
      matr[vektindx]=atof(str);
      vektindx++;
      }
     }
   }

   free(matr);
  fclose(finp);

  return 0;
}
```

# Ismeretlen nagyságú adatsor beolvasása vektorba

Mi történik, ha nem tudjuk előre, hogy mekkora lesz a beolvasott mátrix mérete ?

# Ismeretlen nagyságú adatsor beolvasása vektorba

Mi történik, ha nem tudjuk előre, hogy mekkora lesz a beolvasott mátrix mérete ?

Dinamikusan foglalt memória terület nagysága a program futása során változtatható **realloc** függvénnyel

```
1
2   double *dynvect=(double *)malloc(vect_size); //dinamikus adatfoglalas  egy init_size
3                                                //meretu memoriablokkra
4
5   // egy act_size valtozoban taroljuk, hogy mennyi adat  van pillanatnyilag
6   // a lefoglalt memoriablokkban
7
8    if (act_size < vect_size)
9      {
10     // ha van meg hely az eredetileg lefoglalt memoriateruleten
11     // akkor beolvasott adat elhelyezese a dynvect-be
12
13     act_size++;  //feljegyezzuk, hogy mennyi memoria teruletet hasznaltunk el
14     }
15   else
16   {
17     dynvect=(double *)realloc(dynvect,new_vect_size); // kiterjesztjuk a lefoglalt memoriatombot
18
19     vect_size=new_vect_size;
20
21     // beolvasott adat elhelyezese a dynvect-be
22
23     act_size++;  //feljegyezzuk, hogy mennyi memoria teruletet hasznaltunk el
24   }
```

- vigyázzunk, ha a **realloc** függvényt nem a main() függvényből hívjuk!