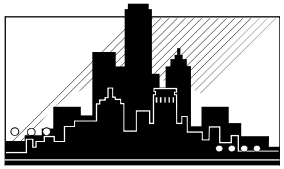
# Dinamikus tömbök, kétdimenziós tömbök

- *Vektorok létrehozása futásidőben, dinamikusan zajló helyfoglalással*
- *Kétdimenziós tömbök*
- *Mintapélda mátrix-vektor szorzására*
- *Kapcsolat a mátrixok indexes és mutatós hivatkozása között*
- *Változó elemszámú tömbök dinamikus helyfoglalással*



## *Vektorok létrehozása futásidőben, dinamikusan zajló helyfoglalással*

*Egy tömb lehet ideiglenes, ha a program futási idejének csak egy részében létezik, azaz foglalja a memóriát. A blokkokban definiált tömbök csak a blokkban élnek, azaz a programnak a blokkból való kilépésekor automatikusan felszabadulnak. Problémát okozhat azonban, hogy a számukra a veremben rendelkezésre álló memóriahely korlátozott. Kikerülhetjük dinamikus memóriakezeléssel ezt a problémát, ha a nagyméretű tömböknek mi magunk foglalunk le helyet a vermen kívüli területen, és használat után a tömbök által foglalt memóriahelyet felszabadítjuk.*

*A dinamikus vektordefiniálás két lépése a mutatódefiniálás és memóriafoglalás (allokálás):*

```
<típus> * <mutató>;
  ...
<mutató> = (<típus>*) calloc(<elemszám>, sizeof(<típus>));
```

Pl.:

```
float * vektmut ;
vektmut = (float*) calloc(200, sizeof(float));
```

*A* calloc() *függvény a lefoglalt területet nullákkal fel is tölti. Ha nincs elég hely, a függvényérték 0.*

*Példa a lefoglalt vektor használatára:*

```
vektmut [19] = 3*14.56 ; printf("%f",vektmut [19]);
```

*Dinamikusan lefoglalt vektor helyének felszabadítása:*

```
free(<mutató>);
```

Pl.: `free ( vektmut );`

- ***Kétdimenziós tömbök, vagy másnéven mátrixok definiálása:***

<típus> <azonosító>[<sorok_sz>] [<oszlopok_sz>] ;

*Pl.:* **int** *beepules*[4][3];

| | | |
|---|---|---|
| 25 | 32 | 45 |
| 18 | 23 | 32 |
| 1 | 2 | 3 |
| 1 | 1 | 2 |

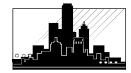
*A mátrixok definiálhatók kezdőértékmegadással (inicializálás) is.*

*Pl.:*

```
int beepules[4][3] = { 25, 32, 45,
                       18, 23, 32,
                        1,  2,  3,
                        1,  1,  2 };
```

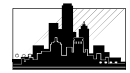
*A mátrixok feltöltése, feldolgozása, kiíratása elvégezhető egymásba-ágyazott kettős* **for** *ciklussal.*

## Mintapélda mátrix-vektor szorzására:



| Alkatrészfélesége | | Termékféleségek | | | | Rendelés | | | Alkartész-igény | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Daráló | Hajtómű | Varrógép | | | | | |
| | | Csavar | 25 | 32 | 45 | Daráló | 8 | db | 932 | db |
| | | Alátét | 18 | 23 | 32 | Hajtómű | 6 | db | 666 | db |
| | | Tengely | 1 | 2 | 3 | Varrógép | 12 | db | 56 | db |
| | | Ház | 1 | 1 | 2 | | | | 38 | db |

*Az alkatrészféleségek fenti beépülési mátrixának*
*és a rendelés vektorának ismeretében határozzuk meg*
*az alkatrészigény-vektort!*



```
#include <stdio.h>
int beepules[ 4 ][ 3 ], rendeles[ 3 ], alkigeny[ 4 ];
main()
{
 int i , j ;
 printf("Alkatrészigény számítás\n\n");
 printf("Adja meg a beépülési mátrixot : \n");
 for ( i = 0 ; i < 4 ; i++)
 {
   for ( j = 0 ; j < 3 ; j++)
    {
     printf("\nA %d. alkatrészből a %d. termékbe",  i +1, j +1);
     printf(" beépülö darabszám= " );
     scanf("%d", &beepules[ i ][ j ] );
     }
 }
```

```
    printf("\n\nAdja meg a rendelést:\n" );

    for ( j = 0; j < 3; j++)
    {
        printf("\naz %d. termékböl (db)= ", j + 1 );
        scanf("%d", &rendeles[ j ] );
    }

    for ( i = 0 ; i < 4; i++)
    {
      for ( alkigeny[ i ] = 0, j = 0; j < 3; j++)
      {
        alkigeny[ i ] += beepules[ i ][ j ] * rendeles[ j ];
      }
    }

    for ( i = 0; i < 4; i++)
    {
        printf("\nA %d. alkatrészböl %d db kell.", i +1, alkigeny[ i ] );
    }
}
```

*A fenti példában is alkalmazhattuk volna a tömbök pointeres megfelelőit az alábbiak szerint:*

*beepules* [ *i* ][ *j* ] *helyett*
* (*beepules*[ *i* ] + *j* ) , *vagy* * ( * (*beepules*+*i* ) + j ),

*rendeles*[ *j* ] *helyett* * (*rendeles* + *j* ) ,

*alkigeny*[ *i* ] *helyett* * (*alkigeny* + *i* ).

## Kapcsolat a mátrixok indexes és mutatós hivatkozása között

*Indexes formában definiált mátrixok tartalma egyaránt elérhető indexekkel, vagy mutatókkal. A mutatókkal, pointeraritmetikával történő elérés gyorsabb, mert nem tartalmazza az indexes alak fordító általi átírásakor használt*

**( (<elemtípus>*) <matrixnev> +*

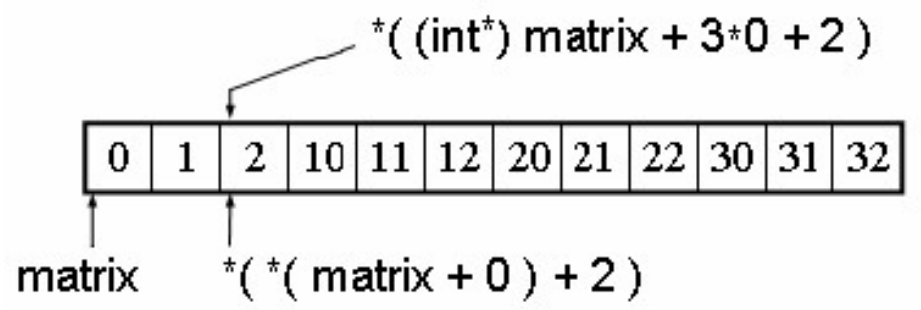
*<oszlopok_száma> * <sorindex> + <oszlopindex>)*

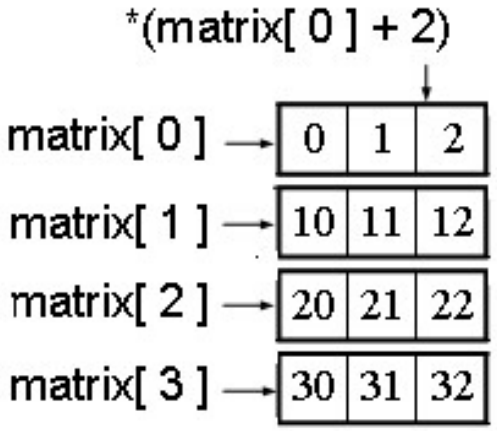
*kifejezésben levő szorzást.*

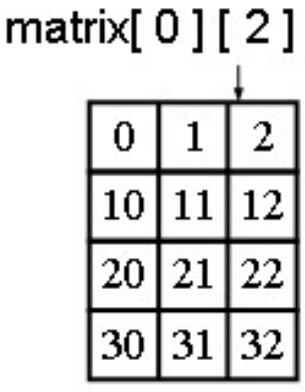
*A memóriában sorfolytonosan elhelyezkedő tömb-elemek tekinthetők a sorokból, mint vektorelemekből álló vektor, vagy a szokásos sor-oszlop tagolású mátrix alakban is:*

*( (**int***) *matrix* + 3*0 + 2 )

Pl.: **int** *matrix* [ 4 ] [ 3 ];



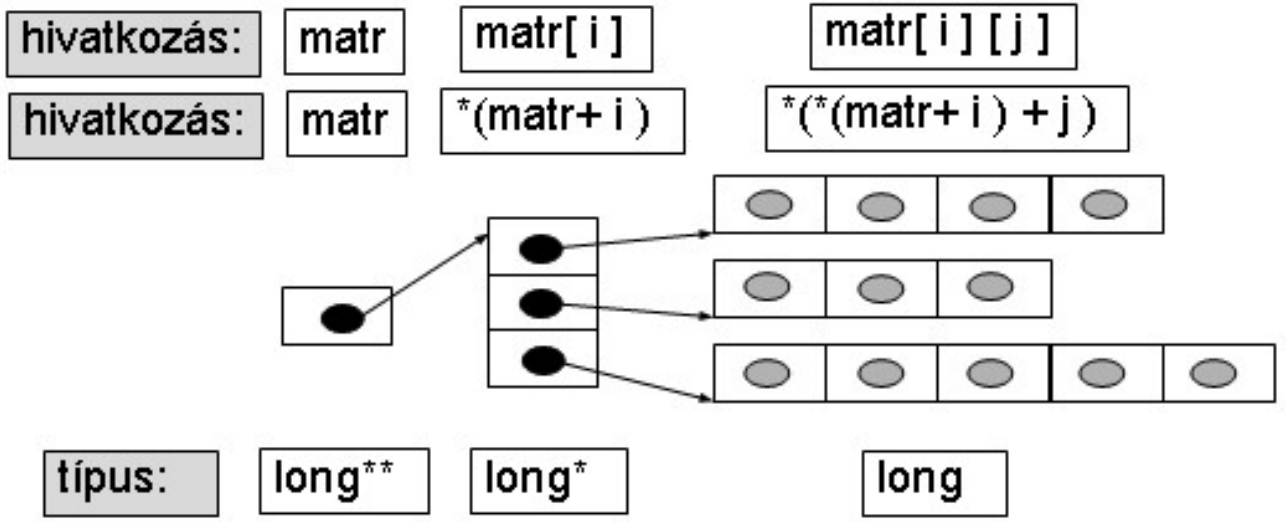



```
#include <stdio.h>

int matrix[ 4 ][ 3 ]   = {0,1,2, 10,11,12, 20,21,22, 30,31,32};

main()
{
    int i, j;
    printf("Kapcsolat a matrixok indexes és mutatos formaja kozott\n\n");
    printf("Az int matrix[4][3] definicio haromfele tipusu és \n");
    printf(" meretu (helyfoglalas byte-okban) kifejezest is \n");
    printf(" meghataroz egyszerre: matrix[ ][ ], matrix[ ] és matrix.\n");
    printf("\nMiutan a definialas soran mar meghatározasra kerult \n");
    printf(" es eltarolodott a meretuk,\n");
    printf(" egyarant hasznalhatjuk az indexes es a mutatos hivatkozast, \n");
    printf(" a kovetkezo egyenlosegeknek megfeleloen:\n");
```

```
    printf("\nsizeof( matrix[0][0] )= %d = sizeof( **matrix ) = %d",
          sizeof( matrix[0][0] ), sizeof( **matrix ) );    // 4  4

    printf("\nsizeof( matrix[0] )=  %d = sizeof( *matrix )  = %d",
          sizeof( matrix[0] ), sizeof( *matrix ) );        //  12  12

    printf("\nsizeof( matrix )=  %d = sizeof( matrix )   = %d",
          sizeof( matrix ), sizeof( matrix ) );            // 48  48

    printf("\n\nFontos, hogy a pointeraritmetikai kifejezesekben a \n");
    printf(" definialaskor eltarolt meretek adjak az elemmeretet, \n");
    printf(" azaz pl: a matrix[ i ][ j ] elemet pointeresen a \n");
    printf("  *(*(matrix + i ) + j )  adja, amely \n");
    printf(" i db  *matrix  tipusú elemmel (i db sorral), és \n");
    printf(" j db  **matrix  tipusú elemmel leptet a matrixban.\n\n");
    printf(" Pl.: matrix[1][2] = %d = *( *( matrix + 1 ) + 2 ) = %d",
          matrix[ 1 ][ 2 ], *(*(matrix+1)+2) );
}
```

## Változó elemszámú tömbök dinamikus helyfoglalással

*Amennyiben csak futás közben derül ki egy adatszerkezetről, hogy hány sora, és soronként hány oszlopa (eleme) van, dinamikusan kell helyet lefoglalnunk az elemek számára. Csak az egy calloc függvénnyel foglalt memóriahelyek tárolnak sorfolytonos elemeket, egyébként azok sorfolytonossága nem garantált.*
*Az adatszerkezet ábrája a következő lehet:*



*A fenti adatszerkezetet létrehozó program:*

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
main()
{
   long** matr ;            // egyetlen mutatót definiálunk csak
   int sor, oszl[20], i, j ;  // meg néhány segédváltozót
   clrscr();
   printf("A programban foglalunk helyet \n");
   printf("megadhato szamu sornak, és \n");
   printf("soronkent megadhato szamu elemnek \n\n");
   printf("Sorok száma = ");
   scanf("%d", &sor );
   matr = (long**) calloc( sor, sizeof(long*) );
```

```
for (i = 0 ; i < sor ; i++)
 {
    printf("\nA %d. sor elemeinek száma= ", i );
    scanf("%d", &oszl[ i ] );
    matr[ i ] = (long*) calloc(oszl[ i ], sizeof(long) );
    for (j = 0; j < oszl[ i ]; j++)
    {
       printf("A %d. sor %d. eleme =", i , j );
       scanf("%ld",&matr[ i ][ j ] );
    }
 }
 for (i = 0; i < sor; i++)                    // Kiíratás
 {
    for (j = 0; j < oszl[ i ]; j++)   printf("%8ld ", matr[ i ][ j ] );
    printf("\n");
 }
 for (i = 0; i < sor; i++)  free(matr[i]); // tárolóhelyek felszabadítása
 free(matr);                                  // mutatóvektor felszabadítása
 getch();
}
```