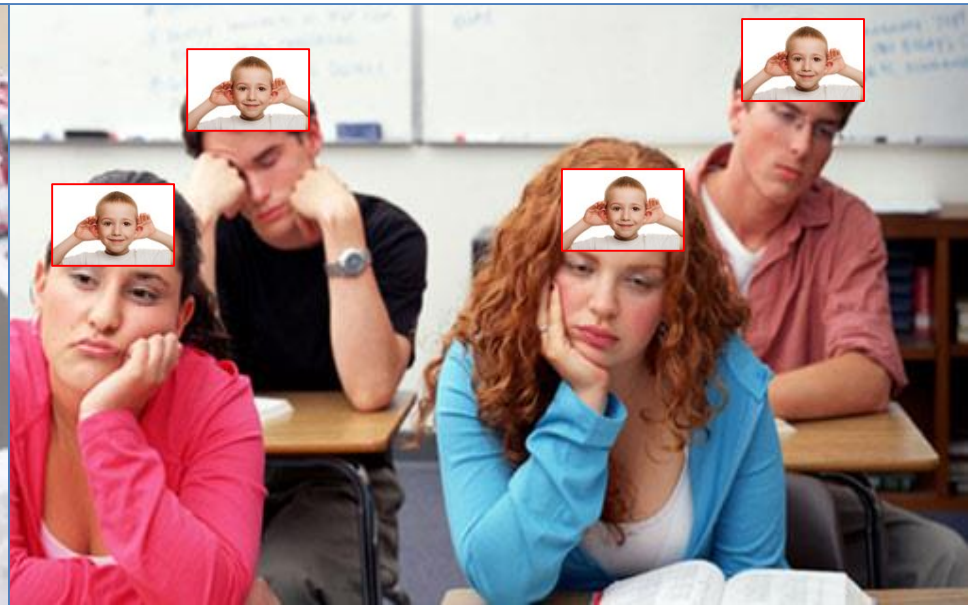
Programozás C nyelven (7. ELŐADÁS)

Sapientia EMTE

2020-21



Rekurzió - ISMÉTLÉS

Tegyük fel, hogy egy bizonyos engedélyt szeretnél kiváltani a polgármesteri hivataltól. Az első irodában közlik veled, hogy az engedély megszerzése feltételezi egy másik engedély birtoklását, amelyet egy másik irodában állítanak ki. Amikor belépsz oda, ugyanazt a választ kapod, mint az előző irodában. És ez így folytatódik addig, míg egy olyan engedélyhez nem jutsz, amelyik megszerzése már nem feltételezi egy további engedély birtoklását. Minekutána ezt kiváltottad, folytathatod a félbehagyott kísérleteidet – fordított sorrendben –, míg minden szükséges engedélyt meg nem szerzel. Végül az első irodában fogják a kezedbe adni azt az engedélyt, amiért beléptél a hivatal ajtaján.

JEGYZET, A-függelék


```
<típus> f(<a feladat paraméterei>)
{
    <típus> talca;
    if (<banalitás feltétele>){<banális eset kezelése>}
    else
    {   talca = f(<átruházott rész paraméterei>);
        <saját rész>
    }
}
```

Általános forgatókönyv bármely példány részére

A kurrens példány a feladat méretétől függően választ a rekurzív, vagy nem-rekurzív viselkedésmódok között.

1. kérdés

Hogyan vezethető vissza a feladat egy *hasonlóképpen megoldható*, de *egyszerűbb* feladatra? Az erre a kérdésre adott válasz világosan el fogja határolni a rekurzívan *átruházandó oroszlánrészt* a *felvállalt saját résztől*. Továbbá nyilvánvalóvá fogja tenni mind a fő feladat, mind az átruházott feladat paramétereit.

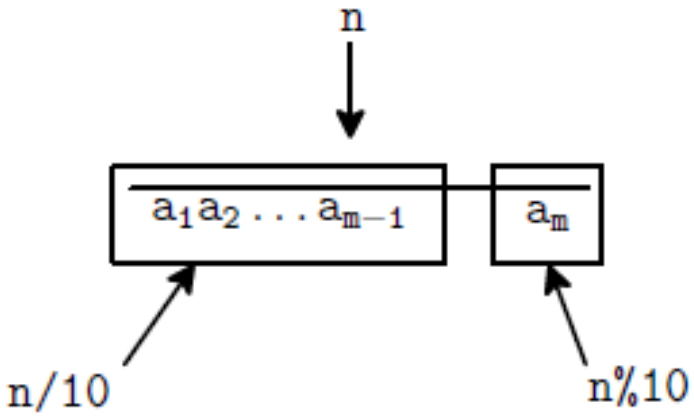
2. kérdés

Miután tálcán megkapjuk az átruházott rész eredményét, *hogyan építhető fel* ebből a teljes feladat eredménye a felvállalt saját rész megoldása által?

3. kérdés

Mikor tekintjük a feladatot annyira *banális*nak, hogy teljesen felvállaljuk a megoldását anélkül, hogy valamit is rekurzívan átruháznánk belőle?

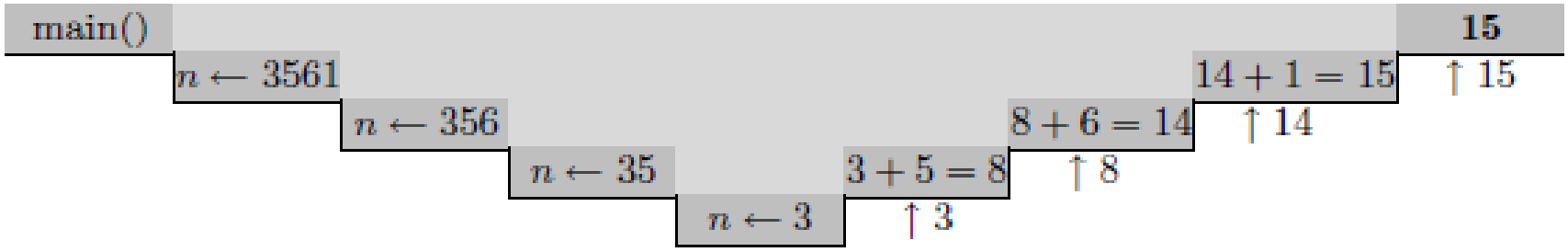
JEGYZET, A-függelék




```
int szamjegy_osszeg( int n){
    if (n > 9){
        return n%10 + szamjegy_osszeg(n/10);
    }
    else {return n;}
}
```

A `talca` változó használata nem kötelező

szamjegy_osszeg(3561)



JEGYZET, A-függelék


```
void rek_elj (<paraméter_lista>)
{
  <lokális definiálások>
  a zóna
  if(<feltétel>)
  {
    b zóna
    rek_elj (<paraméter_lista>);
    B zóna
  }
  else
    { X zóna }
  A zóna
}
```

Hányszor, milyen sorrendben hajtódnak végre az **a,A,b,B,X** zónákba írt utasítások?

$$\underset{\mathrm{I}}{a_1 f_1 b_1}\ \underset{\mathrm{I}}{a_2 f_2 b_2}\ \ldots\ \underset{\mathrm{I}}{a_{n-1} f_{n-1} b_{n-1}}\ \underset{\mathrm{H}}{a_n f_n X_n A_n}\ B_{n-1} A_{n-1}\ \ldots\ B_2 A_2\ B_1 A_1$$

JEGYZET, A-függelék

3. feladat. Írjunk rekurzív eljárást, amely karaktereket olvas be vakon, addig, amíg `'*'` karakter olvasott, a képernyőre pedig a beolvasás fordított sorrendjében írja ki őket:

Például, ha a bemenet: `ABCD*`

Kimenet:

a) `*DCBA`

b) `DCBA`

c) `ABCDDCBA`

d) `ABCD**DCBA`

e) `ABCD*DCBA`

JEGYZET, A-függelék

Például, ha a bemenet: ABCD*
Kimenet:

a) *DCBA
b) DCBA
c) ABCDDCBA
d) ABCD**DCBA
e) ABCD*DCBA

```
void eljA()
{
    char c; /* mindenik átjárásnak meglesz a saját c-je */
    c=getch(); /* beolvasás vakon az a zónában */
    if (c!='*') eljA;
    putchar(c); /* kiírás az A zónában */
}
```

```
void eljB()
{
    char c; /* mindenik átjárásnak meglesz a saját c-je */
    c=getch(); /* beolvasás vakon az a zónában */
    if (c!='*')
        {
          eljB;
          putchar(c); /* kiírás a B zónában */
        }
}
```

()

```
void eljC()
{
    char c; /* mindenik átjárásnak meglesz a saját c-je */
    c=getch(); /* beolvasás vakon az a zónában */
    if (c!='*')
        {
          putchar(c); /* kiírás a b zónában */
          eljC;
          putchar(c); /* kiírás a B zónában */
        }
}
```

```
void eljD()
{
    char c; /* mindenik átjárásnak meglesz a saját c-je */
    c=getch(); /* beolvasás vakon az a zónában */
    putchar(c); /* kiírás az a zónában*/
    if (c!='*') eljD;
    putchar(c); /* kiírás az A zónában */
}
```

```
void eljE()
{
    char c; /* mindenik átjárásnak meglesz a saját c-je */
    c=getch(); /* beolvasás vakon az a zónában */
    if (c!='*')
        {
          putchar(c); /* kiírás a b zónában */
          eljE;
          putchar(c); /* kiírás a B zónában */
        }
    else putchar(c); /* kiírás az X zónában */
}
```

```
bool tartalmaz_5ost( int n){
    if (n > 0){
        bool talca = tartalmaz_5ost(n/10);
        return (n%10 == 5) || talca;
    }
    else {return false;}
}
```

Hányszor hívódnak meg az n=78453 esetben?

```
bool tartalmaz_5ost( int n){
    if (n > 0){
        if (n%10 == 5) {return true;}
        return tartalmaz_5ost(n/10);
    }
    else {return false;}
}
```

Hányszor hívódnak meg az n=78483 esetben?

```
bool tartalmaz_5ost( int n){
    if (n > 0){
        return (n%10 == 5) || tartalmaz_5ost(n/10);
    }
    else {return false;}
}
```

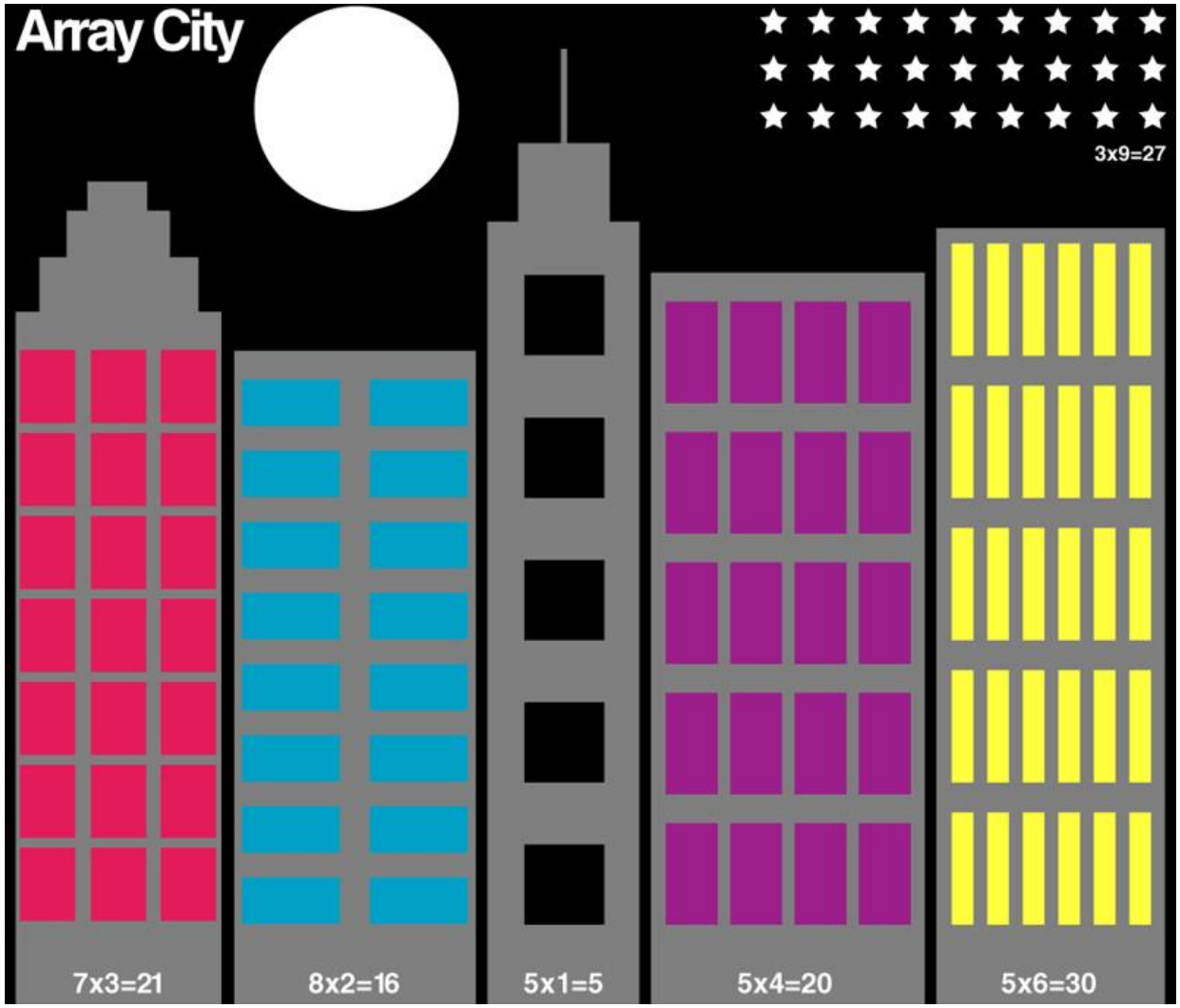
Ha az első operandus IGAZ, akkor nem hívódik meg a függvény. MIÉRT?


```
int min_szamjegy( int n){
    if (n > 9){
        int talca = min_szamjegy(n/10);
        return (n%10 < talca) ? (n%10) : (talca);
    }
    else {return n;}
}
```

```
int min_szamjegy( int n){
    if (n > 9){
        if (n%10 < min_szamjegy(n/10)){
            return n%10;
        }
        else{
            return min_szamjegy(n/10);
        }
    }
    else {return n;}
}
```

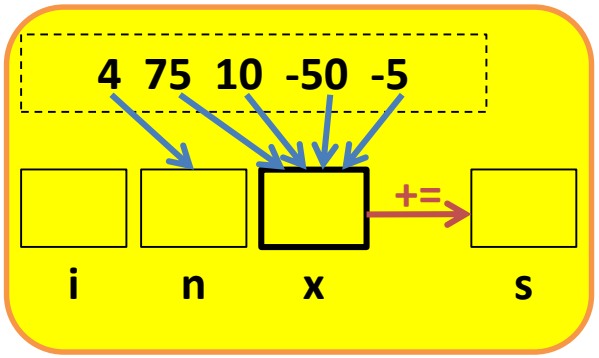
Miért kifejezetten ajánlott, jelen esetben, a `talca` változó használata?

TÖMBÖK



Szám-sorozat *egy* változóban

Adott egy n elemű számsorozat, számítsuk ki az elemek összegét.




```
int i,n,x,s=0;
scanf("%i", &n);
for ( i=1 ; i<=n ; ++i ){
    scanf("%i",&x); s += x;
}
printf("s= %i", s);
```

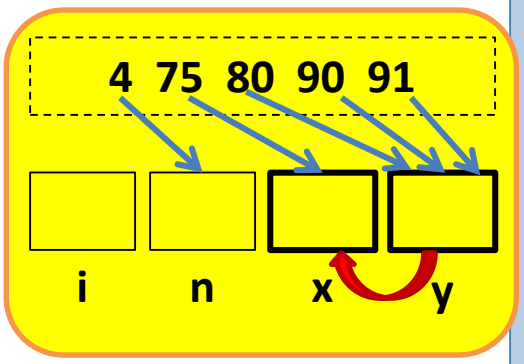
```
4
75
10
-50
-5
s = 30_
```

WATCH

x	s
?	0
75	75
10	85
-50	35
-5	30

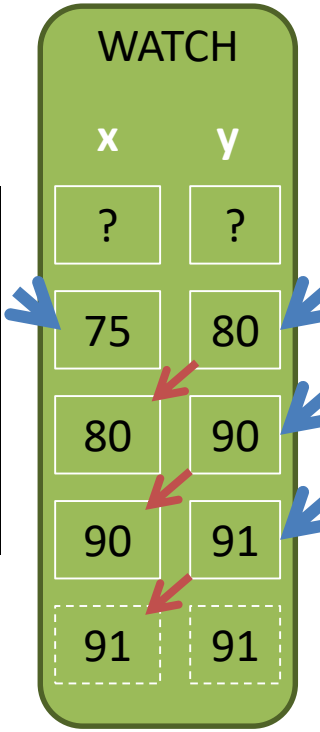
Szám-sorozat *két* változóban

Adott egy n (n>1) elemű számsorozat, ellenőrizzük, hogy növekvő-e.



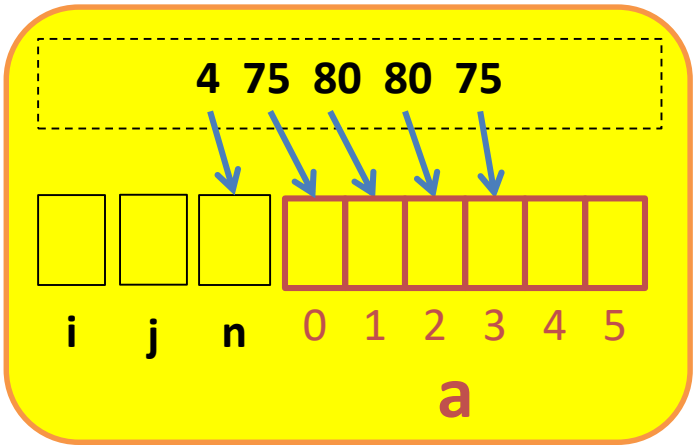

```
int i,n,x,y;
scanf("%i", &n);
scanf("%i", &x);
for ( i=2 ; i<=n ; ++i ){
    scanf("%i", &y);
    if(y < x) {break;}
    x = y;
}
if (i<=n) { printf("NEM"); }
else { printf("IGEN"); }
```

```
4
75
80
90
91
IGEN_
```



Szám-sorozat *tömb* változóban

Adott egy n (n>1) elemű számsorozat, ellenőrizzük, hogy tükörsorozat-e.




```
int i,j,n,a[6];
scanf("%i", &n);
for ( i=0 ; i<n ; ++i ){
   scanf("%i", &a[i]);
}
for ( i=0,j=n-1 ; i<j ; ++i,--j ){
   if(a[i] != a[j]) {break;}
}
if (i<j) { printf("NEM"); }
else { printf("IGEN"); }
```

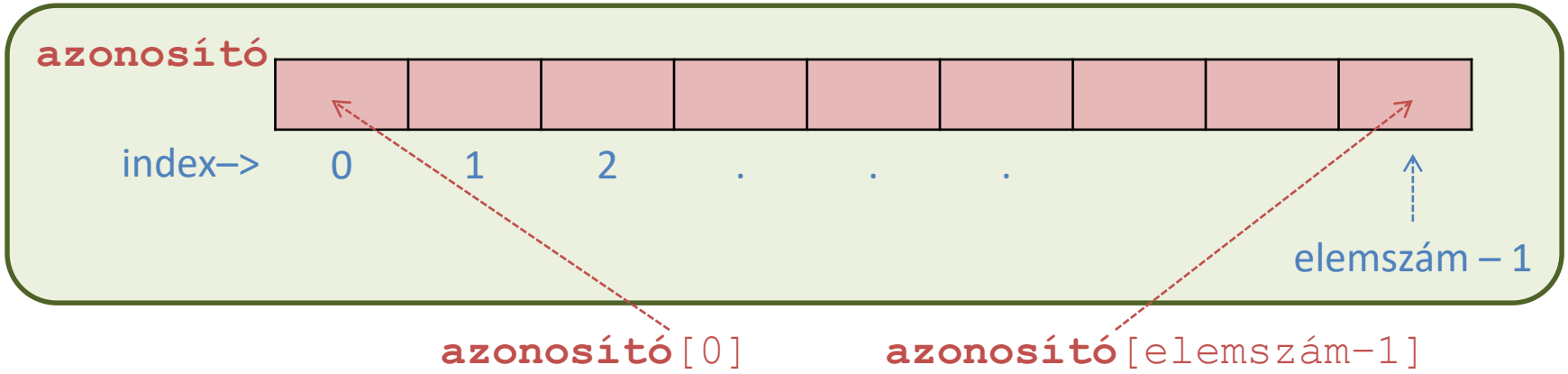
```
4
75
80
80
75
IGEN_
```

TÖMB – típus

Definiálás:

Egész típusú

```
<elemtípus> <azonosító>[<elemszám>];
```



Példák:

```
int a[100]; //elemei a[0], a[1], …, a[99]
double x[10], y, z[50]; // sizeof(x)=?
short b[5]={11,22,33,44,55}; //b[0]=?, b[4]=?
long c[1000]={0}, d[50]={1}; //c[999]=?, d[49]=?
char s[5]={'a','b','c','d','e','f'}; //???
int w[]={1,2,3,4,5,6,7,8,9,10}; // sizeof(w)=?
```

Számsorozat beolvasása egydimenziós tömbbe

```
int main(){
    freopen("szamsor.txt", "r", stdin);
    int n, a[100];
    scanf("%i", &n);
    for(int i = 0 ; i < n ; ++i){  // generálja i-ben az
        scanf("%i", &a[i]);        // 0,1,...,n-1 index-sort
    }
    ooo
    return 0;
}
```

C99
```
int n;
scanf("%i",&n);
int a[n];
```

C99
i csak a for-on belül ismert

szamsor.txt

```
6
44 5 13 7 -10 11
```

44	5	13	7	-10	11	?	...	?
0	1	2	3	4	5	6		99

Számsorozat kiírása egydimenziós tömbből

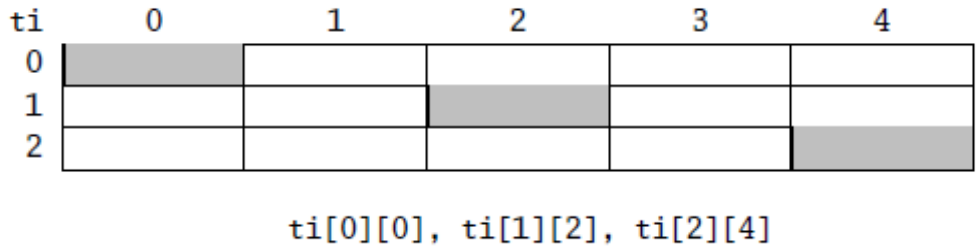
```
int main(){
   ooo
   for(int i = 0 ; i < n ; ++i){
      printf("%i ", a[i]);
   }
   ooo
   return 0;
}
```

Kétdimenziós tömbök

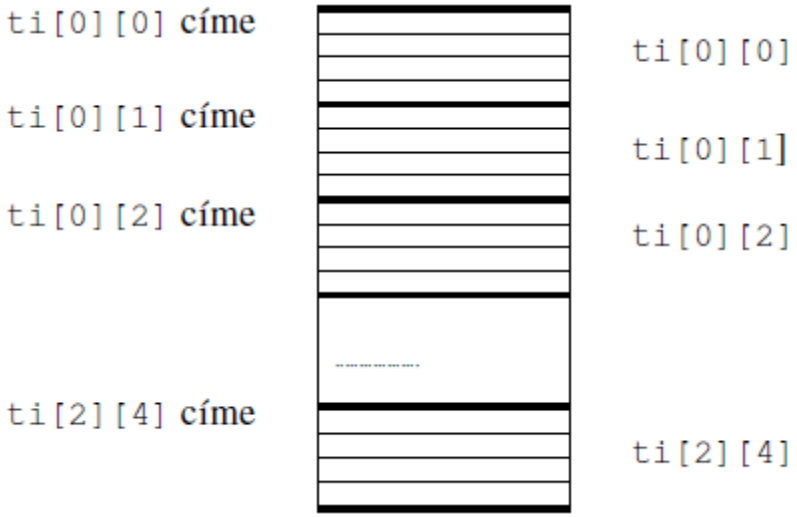
```
<elem_típus> <név>[<sorok_száma>][<oszlopok_száma>];
```

```
char tc[20][20];
int ti[3][5];
long double tld[10][10];
```

```
int a[2][3]={1, 3, -5, 0, 77, -13};
int b[][3]={1, 3, -5, 0, 77, -13, 9, 1, 34};
```



ti[0][0], ti[1][2], ti[2][4]




```
<elem_típus> <név>[<méret_1>][<méret_2>]...[<méret_n>];
```

```
<név>[<index_1>][<index_2>]...[<index_n>]
```

Mátrix beolvasása kétdimenziós tömbbe

matrix.txt

```
2 3
44 5 13
7 -10 11
```

```
int main(){
    freopen("matrix.txt", "r", stdin);
    int n, m, b[100][100];
    scanf("%i%i", &n, &m);
    for(int i = 0 ; i < n ; ++i){          // generálja (i,j)-ben az
        for(int j = 0 ; j < m ; ++j){      // (0,0),(0,1),…,(0,m-1 ),(1,0),…,(n-1,m-1)
            scanf("%i", &b[i][j]);         // indexpár-sort
        }
    }

    ooo

    return 0;
}
```

	0	1	2	3	…	99
0	44	5	13	?	…	?
1	7	-10	11	?	…	?
2	?	?	?	?	…	?
…	…					
99	?	?	?	?	…	?

Mátrix kiírása kétdimenziós tömbből

```
int main(){
    ooo
    for(int i = 0 ; i < n ; ++i){
        for(int j = 0 ; j < m ; ++j){
            printf("%i ", b[i][j]);
        }
        printf("\n");
    }
    ooo
    return 0;
}
```

	0	1	2	3	…	99
0	44	5	13	?	…	?
1	7	-10	11	?	…	?
2	?	?	?	?	…	?
…	…					
99	?	?	?	?	…	?

```
44 5 13
7 -10 11

_
```

Mátrix bejárása oszloponként

```
int main(){
    ooo
    for(int j = 0 ; j < m ; ++j){
        for(int i = 0 ; i < n ; ++i){
            printf("%4i", b[i][j]);
        }
        printf("\n");
    }
    ooo
    return 0;
}
```

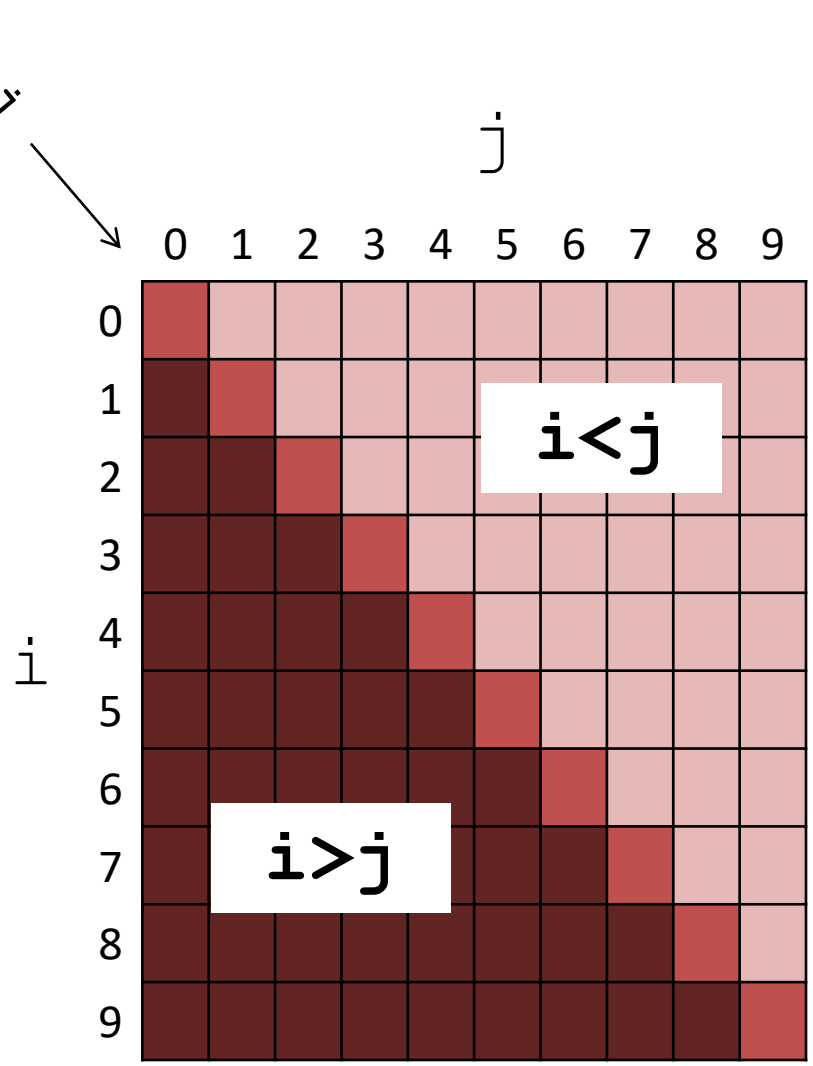
	0	1	2	3	…	99
0	44	5	13	?	…	?
1	7	-10	11	?	…	?
2	?	?	?	?	…	?
…	…					
99	?	?	?	?	…	?

```
  44   7
   5 -10
  13  11

_
```

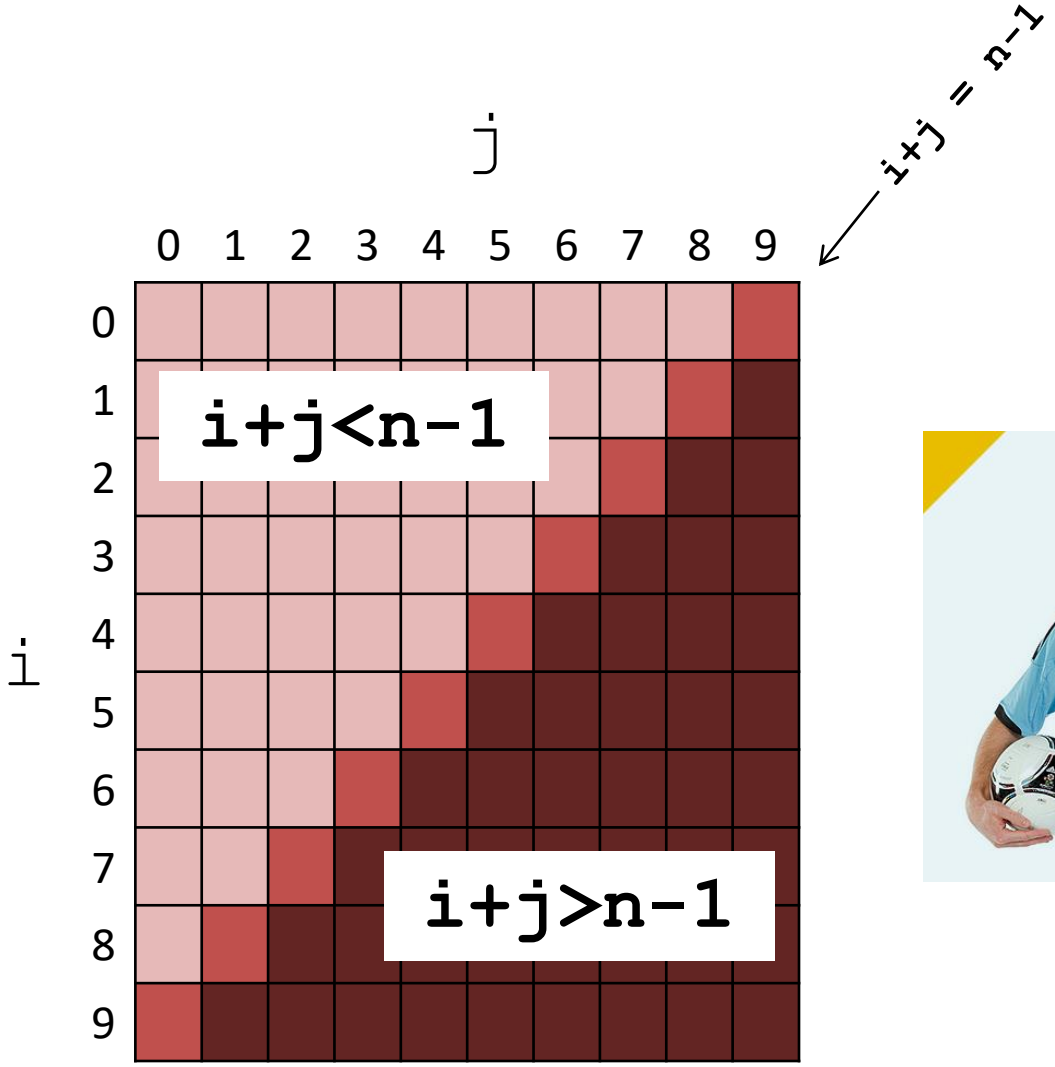
Mozogjunk otthonosan négyzetes mátrixokban

főátlón / alatta / felette



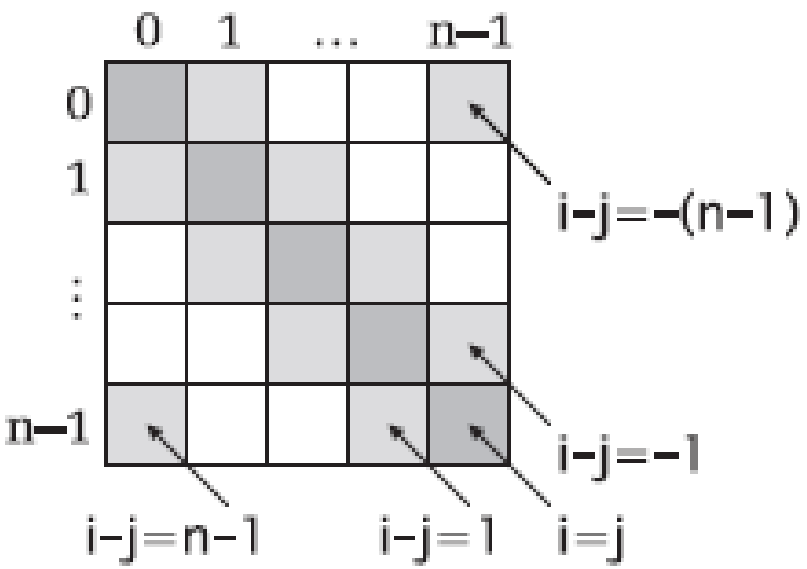
Mozogjunk otthonosan négyzetes mátrixokban

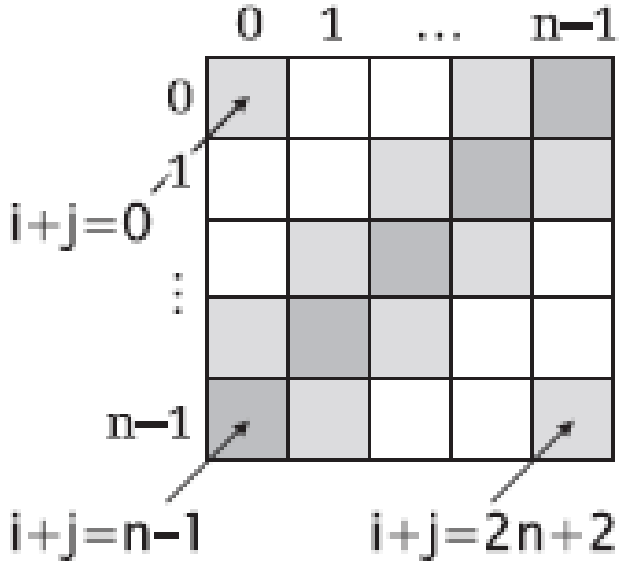
mellékátlón / alatta / felette



Mozogjunk otthonosan négyzetes mátrixokban

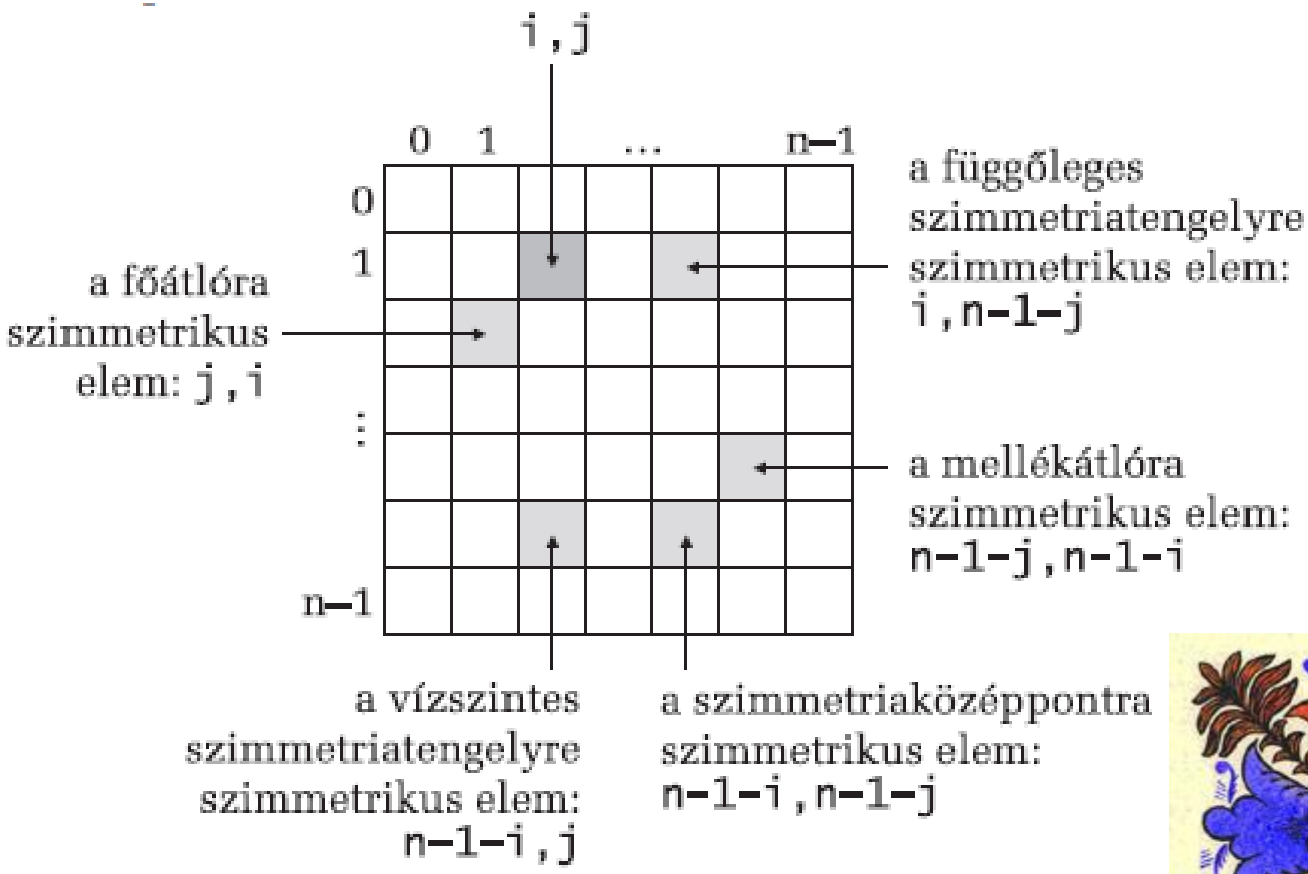
főátlóval/mellékátlóval *párhuzamos* átlókon





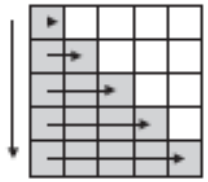
Mozogjunk otthonosan négyzetes mátrixokban

szimmetriapontok

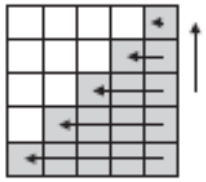


Mozogjunk otthonosan négyzetes mátrixokban

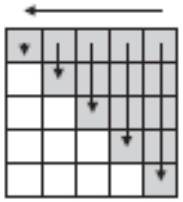
háromszögekben sétafikálva



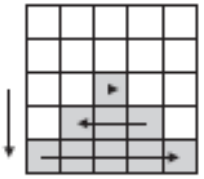
```
for(i=0; i<n; i++)
    for(j=0; j<=i; j++)
```



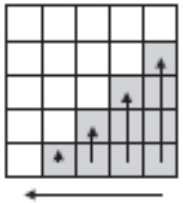
```
for(i=n-1; i>=0; i--)
    for(j=n-1; j>=n-1-i; j--)
```



```
for(j=n-1; j>=0; j--)
    for(i=0; i<=j; i++)
```



```
for(i=n/2; i<=n-1; i++)
    for(i%2?j=i:j=n-1-i;
        j>=n-1-i && j<=i;
        i%2?j--:j++)
```



```
for(j=n-1; j>0; j--)
    for(i=n-1; i>n-1-j; i--)
```

Összefoglalás

- 1-dimenziós tömbök (számsorok tárolására)
  - **<elemtípus> <név>[<elemszám>];**
    - `int a[100], b[1000]={0}, c[]={1,2,3};`
    - `for(int i=0 ; i<n ; ++i){… a[i] …}`

- 2-dimenziós tömbök (mátrixok tárolására)
  - **<elemtípus> <név>[<sorszám>][<oszlopszám>];**
    - `long x[50][100], y[][3]={1,2,3,4,5,6};`
    - ```
      for(int i=0 ; i<n ; ++i){
        for(int j=0 ; j<m ; ++j){
          … x[i][j] …
        }
      }
      ```


      ```