
A C programozási nyelv III. Pointerek és tömbök.

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

Mutató (pointer) fogalma

A mutató olyan változó, amely egy másik objektum címét tartalmazza.

Mutató deklarációja:

tipusnév* azonosító

Egy objektum címe megkapható az & operátorral.

A mutató által megcímzett objektum értékére a

***pointerazonosító**

kifejezéssel hivatkozhatunk, amely **balérték!**

Formális példa

```
int a;  
int b;  
int* pa;  
pa = &a; /* A "pa"-ban az "a" címe */  
b = *pa; /* b=a értékadás */  
*pa = 0; /* a=0 értékadás */  
(*pa)++; /* a++ */  
/* *pa++ nem a fenti, hanem *(p++)!  
*/
```

Címaritmetika 1.

Mivel a mutató is változó, így

- értéket kaphat,
- műveletek végezhetők rajta.
- A műveletek definíciója figyelembe veszi azt a tényt, hogy a mutató címet tárol
- A műveletek eredményét befolyásolja az, hogy a mutató milyen típusra mutat.

Címaritmetika 2.

Megengedett műveletek:

- mutató és egész összeadása, kivonása
- mutató inkrementálása, dekrementálása

Eredménye újabb cím, amely ugyanolyan típusú objektumra mutat. Kiszámítása:

típus* p;

int n;

esetén a **p+n** értékét úgy határozzuk meg, hogy a **p** értékéhez az **n * sizeof(típus)** eltolást hozzáadjuk.

Címaritmetika 3.

Megengedett műveletek (folytatás):

- két mutató kivonása

Eredménye egész, a két cím között elhelyezkedő, adott típusú elemek száma. Csak egy tömbön belüli mutatók esetén értelmes az eredmény.

- mutatók összehasonlítása

Eredménye egész, és csak speciális esetekben értelmezhető

Címaritmetika 4.

Megengedett műveletek (folytatás):

- mutatónak "0" érték adása
- mutató összehasonlítása 0-val
A "0" érvénytelen cím, ezzel jelezhetjük, hogy a pointer értéke még beállítatlan
- mutató indexelése - lásd később!
- mutatók közötti értékadás

Nem azonos alaptípusú mutatók között általában értelmetlen. Egyéb veszélyei is vannak (példák később.)

Mutatók és tömbök 1.

A C- ben egy tömb azonosítóját a fordító mindig a tömb első elemét megcímző (konstans, és értékkel rendelkező) mutatóként kezeli.

Következmények:

- egy tömb elemei indexeléssel és mutatóaritmetikával egyaránt elérhetők
- a mutatók is indexelhetők
- tömb esetén a kezdőcímet kapja meg a függvény

Példák:

Mutatók és tömbök 2.

Legyen a deklarációs részben az alábbi sor:

```
int *pa, a[10], i;
```

és tételezzük fel, hogy végrehajtódott az alábbi:

```
pa = &a[0];
```

Mutatók és tömbök 3.

Kifejezés	Vele egyenértékű	Jelentés
<code>pa=&a[0]</code>	<code>pa = a</code>	A pa mutató az a tömb első elemére mutat
<code>a[i]</code>	<code>*(pa+i)</code> <code>*(a+i)</code> <code>pa[i]</code>	Hivatkozás az a tömb i indexű elemére
<code>&a[i]</code>	<code>pa+i</code> <code>a+i</code>	Az a tömb i indexű elemének címe

Karakterlánc és mutató

A string-állandó karaktertömb, és megengedett egy karaktertömböt karakterláncsal inicializálni.

Ugyanezen okból megengedett egy **char*** mutató és egy string közötti értékadás, mert ez két mutató közötti értékadást jelent. Például:

```
char *string;
```

```
string="Ez egy szoveg"
```

használható, és ez után például ***(string+3)**

vagy **string[3]** egyenlő **'e'**-vel

Mutató tömb, mutatót címző mutató

Mivel a mutató is változó, így

- tömbökbe foglalható,
- címezheti mutató.

Példa: 100 darab char típusú mutatót tároló tömb:

```
char* sor[100];
```

A tömb azonosítója is mutató, ezért

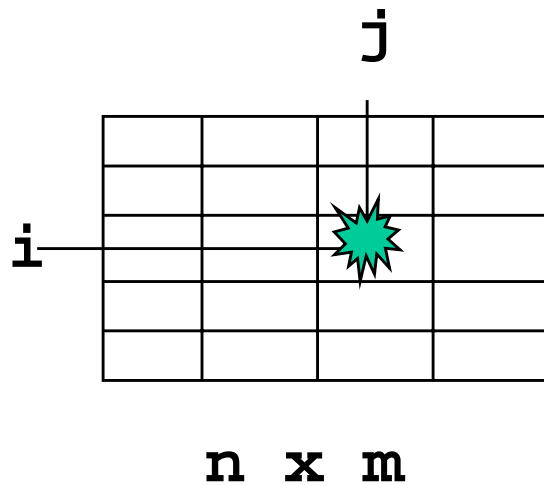
- a **sor** is az
- egy **char*** mutatót (a **sor[0]**-át) címez meg
- tehát típusa **char****

Többsdimenziós tömbök

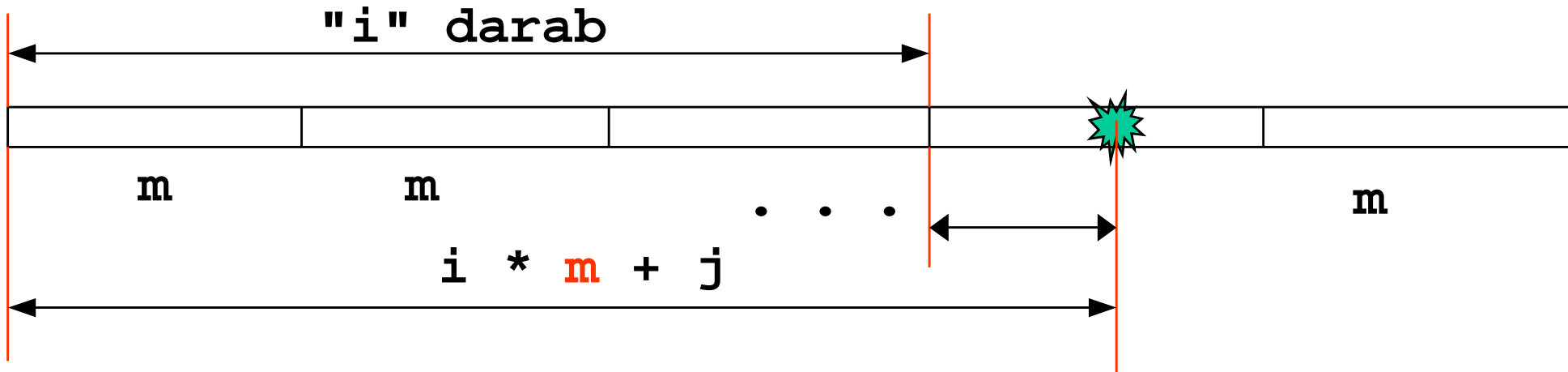
Kétdimenziós tömb:

- egydimenziós tömb, amelynek minden eleme tömb
- `int a[10][5]` jelentése `(int a[10])[5]`, azaz olyan 5 elemű tömb, amelynek minden eleme egy 10 elemű egész tömb
- `a` típusa `int (*a)[5]` (mert olyan mutató, ami 5 elemű, egész tömböket tartalmazó tömböt címez)
- következmény: a C tömbök elemei sorfolytonosan tárolódnak a memóriában

Többsdimenziós tömbök tárolása



**Egy tömbelem helyének
kiszámításához szükséges
az oszlopok száma!!!**



Többsdimenziós tömb és mutatótömb 1.

Kétdimenziós `int a[10][10]` Helyfoglalás:
tömb 100 db egész

Mutatótömb `int *b[10]` 10 darab cím

Mivel `b[5]` `int*` típusú mutató, indexelhető, tehát `b[5][5]` írható. Jelentése:

"a `b[5]` által megcímzett helytől számított 5. `int` elem"

Következmény: csak akkor értelmes, ha `b[5]`
beállított!!!

Többsdimenziós tömb és mutatótömb 2.

Példa:

```
char s[3][10];
```

```
char *st[3];
```

```
    s[0] = "Első";
```

```
    s[1] = "Második";
```

```
    s[2] = "Harmadik";
```

Az **s** tömb 30 byte-nyi helyet foglal le, és minden string hossza korlátozott 10- re.

Az **s[1][5]** az **i** betűt jelenti.

Többsdimenziós tömb és mutatótömb 3.

```
st[0] = "Első";
```

```
st[1] = "Második";
```

```
st[2] = "Harmadik";
```

Az `st[3]` 3 címnek szükséges helyet foglal el

Ehhez még hozzáadódik a karaktersorozatok tárolására szükséges hely (de csak a feltétlenül szükséges)

Tetszőleges hosszúságú stringek tárolhatók.

Az `st[1][5]` most is az `i` betűt jelenti.

Mutató, mint függvény argumentum 1.

- Függvény argumentuma egydimenziós tömb
 - a tömbazonosító mutató \Rightarrow a függvény a tömb első elemének címét kapja meg.
 - A deklaráció formájától függetlenül a tömbelemekre mutatókkal vagy indexeléssel is hivatkozhatunk.
 - átadható egy tömb tetszőleges elemének címe is

Példa:

Formális paraméter

```
int a[]
```

```
int *a
```

Hivatkozás

```
a[i] vagy *(a+i)
```

```
a[i] vagy *(a+i)
```

Mutató, mint függvény argumentum 2.

- Függvény argumentuma többdimenziós tömb
 - A tömbelemek helyének kiszámításához kellene a függvénynek a dimenziók határai (az első kivételével)

Példák:

```
int a[10][5]
```

```
int a[][5]
```

```
int (*a)[5]
```

- Az algoritmus számára külön paraméterekben kell megadni a tényleges dimenzió-határokat
- **Hibalehetőség:** ellentmondó paraméterezés!!!



Mutató, mint függvény argumentum 3.

Példa:

```
double atlag(int mat[][10], int n, int m)
{
    double sum = 0;
    int i,j;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            sum += matr[i][j];
    return sum - n * m;
}
```

Mutató, mint függvény argumentum 4.

Hívás:

```
main() {  
    int a[5][10];  
    int n, m, i, j;  
    double atlagos;  
    n = 3; m = 8; szam = 0;  
    for (i=0; i<n; i++)  
        for (j=0; j<m; j++) a[i][j] = szam++;  
    atlagos = atlag(a, n, m); }
```

Mutató, mint függvény argumentum 5.

Hibalehetőségek:

- $n > 5$ vagy $m > 10$
 - index túllépést okoz
- az aktuális paraméter tömb második dimenziójának elemszáma nem 10
 - hibás számítást eredményez a tömbelemek helyének kiszámításánál

Mutató, mint függvény argumentum 6.

- Függvény argumentuma egy változó mutatója
 - így indirekt hivatkozással megváltoztatható a függvény argumentumának értéke.

Példa: két argumentum értékének cseréje:

```
void csere (int* x,int* y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Mutató, mint függvény argumentum 7.

– A függvény hívása:

```
main( )
```

```
{
```

```
int a,b;
```

```
•
```

```
•
```

```
csere (&a,&b);
```

```
•
```

```
•
```

```
}
```



Mutató, mint függvény argumentum 8.

Összefoglalva: egy formális paraméter két okból lehet egyszerű mutató

- egydimenziós tömböt vesz át a függvény
 - az aktuális paraméter egy tömb azonosítója vagy valamelyik elemének címe lehet
 - a formális paraméter tömb elemeire indexeléssel vagy mutató aritmetikával hivatkozunk
- egy változó értékét akarja megváltoztatni a fgv.
 - az aktuális paraméter egy balérték címe lehet
 - a formális paraméter értékére indirekcióval hivatkozunk