Mutatók, dinamikus helyfoglalás

Egy memóriacímet tartalmazó változót **mutatónak (pointer)** nevezzük.

1. Mutató definiálása

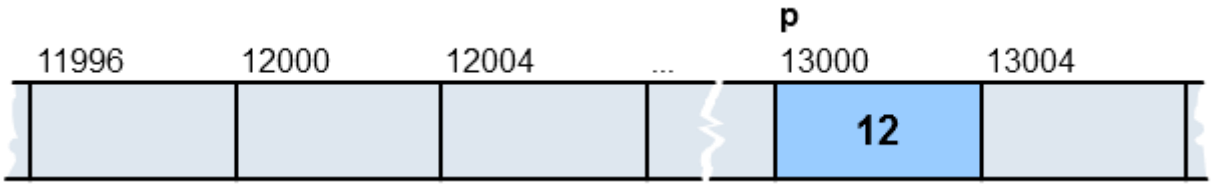
- Ha a mutató egész szám típust tartalmazó memóriaterületre fog mutatni: **int *poin1**
- Ha a mutató valós szám típust tartalmazó memóriaterületre fog mutatni: **double *poin2**
- Általános, típus nélküli mutató (ilyent mutatót akkor definiálunk, ha nem ismerjük a megcímzendő adat típusát): **void *poin3**

2. Egész szám típusra mutató pointer

Az alábbi programsor létrehoz egy egész szám típusú változót a memóriában és beállítja annak értékét 12-re:

```
int p = 12;
```

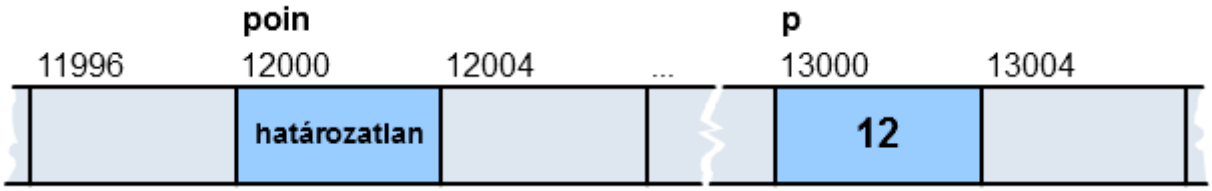
Az egész szám tárolására 4 bájt szükséges. Tegyük fel, hogy ennek a változónak a helyét a 13000-es memóriacímtől foglalta le a program. Ekkor ennek a változónak a memóriában való elhelyezkedését így képzelhetjük el:



Az alábbi programsor létrehoz egy olyan mutatót, amely egész szám típusú változó memóriacímét fogja tartalmazni. A létrehozáskor ennek tartalma határozatlan, ezt később be kell állítanunk!

```
int *poin;
```

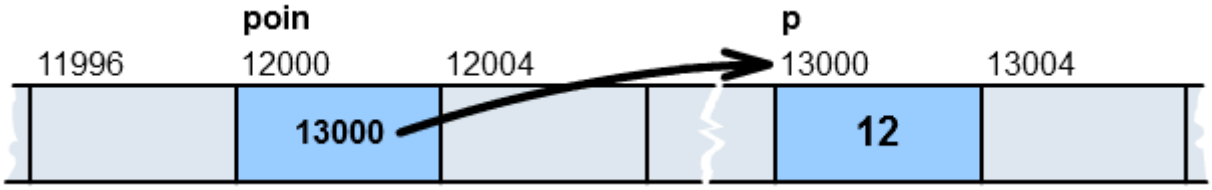
A memóriacím tárolására is 4 bájt szükséges. Tegyük fel, hogy ennek a mutatónak a helyét a 12000-es memóriacímtől foglalta le a program. Ekkor ennek a mutatónak a memóriában való elhelyezkedését így képzelhetjük el:



Mivel egy változó memóriacímét az **&** segítségével határozhatjuk meg (hasonlóan, ahogy azt a **scanf** függvénynél tettük), a mutatónkhoz a memóriacímet így rendelhetjük hozzá:

```
poin = &p;
```

Ez után a **poin** mutató a **p** változó memóriacímét, tehát a mi példánkban a 13000-es címet fogja tartalmazni:



Ezek után a **p** változó értékét kiírhatjuk a változó nevével, de akár a **poin** mutatónk segítségével is:

```
printf("%d ", p);
printf("%d ", *poin);
```

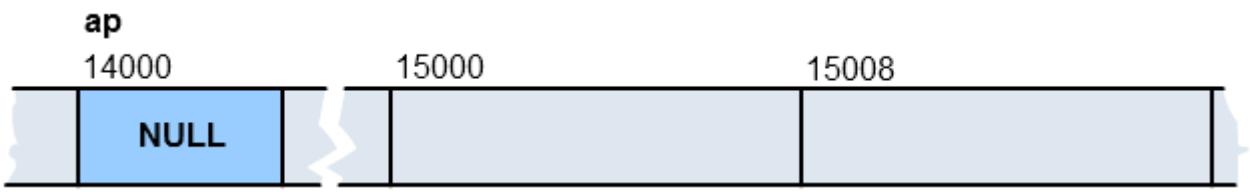
A fenti két sor ugyanazt a számot, a 12-t fogja kiírni. A ***poin** kifejezés a megcímzett adatot jelenti (a mi esetünkben a **p** változó tartalmát, tehát a 12-t).

3. Dinamikus helyfoglalás

A mutatók segítségével könnyen lefoglalható bármilyen méretű memóriaterület. A memóriafoglalás a **malloc()** függvény segítségével végezhető el, amely az **<stdlib.h>** könyvtárban található.

Az alábbi programsor létrehoz egy mutatót, amely egy double típus memóriacímét fogja tartalmazni. Mivel ez a mutató egyelőre még nem tartalmaz memóriacímet, ezért NULL értéket rendelünk hozzá:

```
double *ap = NULL;
```

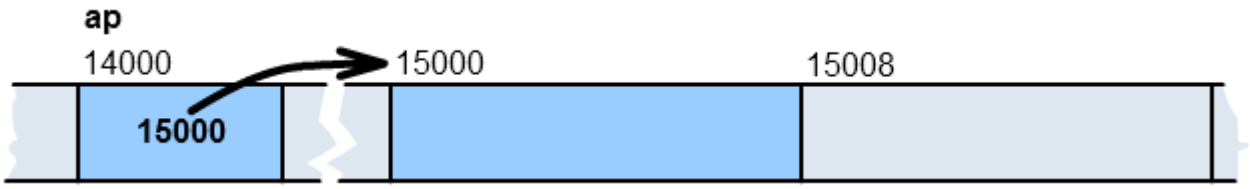


Ez után az **malloc()** függvénnyel lefoglalunk akkora memóriaterületet, amekkora egy **double** típusú változó tárolására szükséges.

A szükséges memóriaterület méretét a **sizeof()** függvénnyel határozzuk meg (ez a mi esetünkben 8-at ad vissza, mivel a double típus mérete 8 bájt).

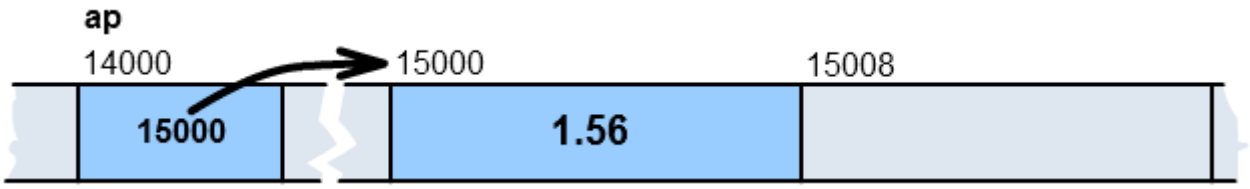
A malloc() függvény egy általános (**void***) mutatót ad vissza, ezért ezt **double*** típussá alakítjuk mielőtt hozzárendeljül a mutatónkhoz.

```
ap = (double*)malloc(sizeof(double));
```



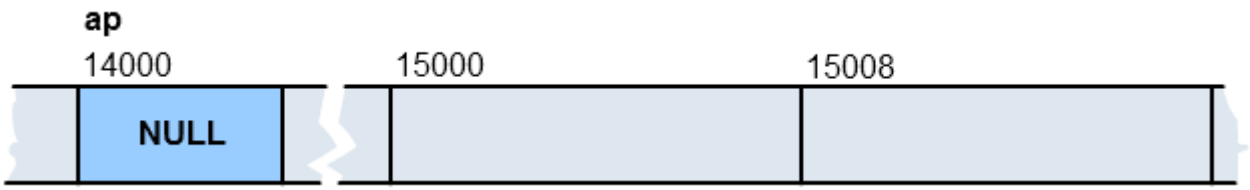
Ha az **ap** tartalma nem **NULL**, akkor a memóriafoglalás sikerült. Ebben az esetben értéket adunk a mutatóval kijelölt tárolónak:

```
if (ap != NULL) {
  *ap = 1.56;
}
```



Ha később már nincs többé szükségünk a lefoglalt területre, azt a **free()** függvény segítségével szabadíthatjuk fel (ez a függvény szintén az **<stdlib.h>** könyvtárban található). A mutatóban levő cím azonban nem törlődik, a **NULL** értékadással állítjuk be hogy nem mutat sehova.

```
free(ap);
ap = NULL;
```

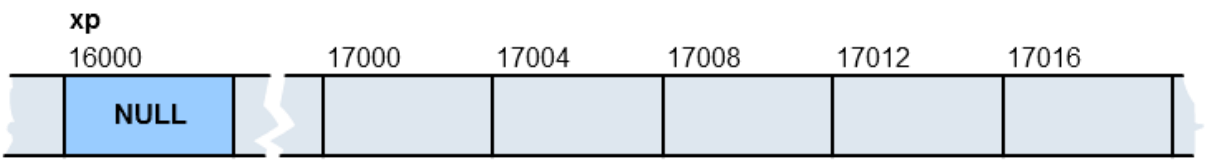


4. Tömb dinamikus létrehozása

A pointerek segítségével a tömbök dinamikusan létrehozhatók, számukra a kívánt nagyságú terület az **malloc()** függvénnyel foglalható le.

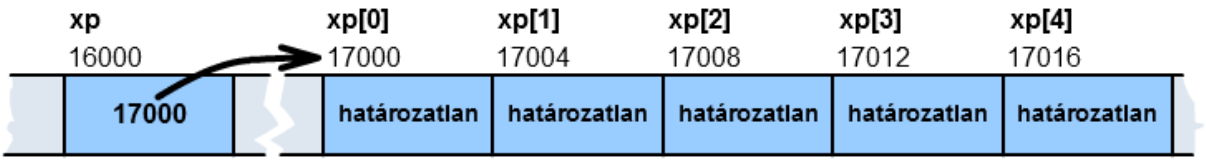
Egész szám típusú tárolóra mutató pointer létrehozása:

```
int *xp = NULL;
```



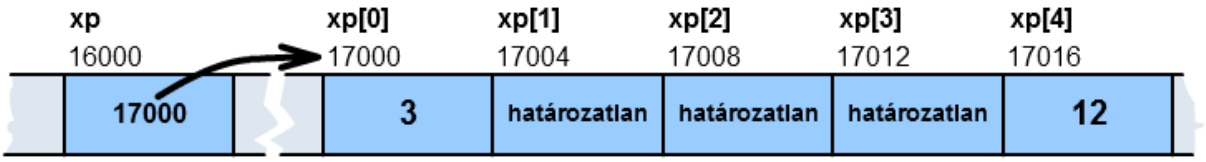
Memóriafoglalás 5 elemű tömb számára:

```
xp = (int*)malloc(5 * sizeof(int));
```



Ha a memóriafoglalás sikerült, értéket adhatunk a tömb elemeinek:

```
if (xp != NULL) {
  xp[0] = 3;
  xp[4] = 12;
}
```



Ha később már nincs többé szükségünk a lefoglalt területre, azt a **free()** függvény segítségével szabadíthatjuk fel. A mutatóban levő cím azonban nem törlődik, a **NULL** értékadással állítjuk be hogy nem mutat sehova.

```
free(xp);
xp = NULL;
```