

VÁLTOZÓK

C# NYELVEN

Változók (variable)

A változó egy névvel azonosított memória terület.

- A program végrehajtásakor az adatokat változókbán tároljuk.
- A változókat első használat előtt **deklarálni** kell:
 - Név
 - Típus } megadása
- **Definíció:** olyan deklaráció, amely helyfoglalással jár
 - Deklaráció értékadással, pl. `int a=5;`
 - Deklaráció példányosítással, pl. `int[] t=new int[10];`
(Megjegyzés: példányosításkor értékadás is történik)

Változók további jellemzői

- Hatókör vagy érvényességi kör
- Élettartam
- Láthatóság

Változók hatóköre (scope), élettartama (lifetime)

Változók hatóköre:

- A forráskódnak az a része, ahol hivatkozhatunk a változóra, azaz használhatjuk a nevét anélkül, hogy a fordítóprogram hibaüzenetet adna.
- **A változó hatókörét a deklaráció helye határozza meg.**

Változók élettartama:

- A futási időnek az a része, amikor a változó megtalálható a memóriában.
- A helyfoglalás a hatókörbe lépéskor történik.

Alapelv a deklarálás helyére:

A változóinkat

- minél **később**,
- minél **szűkebb** programblokkban deklaráljuk.

Ezzel segíthetünk a fordítóprogramnak a memória-kihasználás optimalizálásában.

A változók hatókör szerint lehetnek:

➤ Globális változók → nincs a C#-ban!

➤ **Lokális változók**

↳ blokkok (pl. for, if stb)

↳ metódusok

} lokális változói

➤ **Osztályok/struktúrák tagváltozói (mezői)**

Deklarálás helye és hatókör, élettartam

Deklarálás helye	Hatókör	Élettartam
Blokk (lokális változók)	A deklaráció helyétől a befoglaló blokk végéig	A metódus végéig
Metódus (lokális változók)	A deklaráció helyétől a metódus blokkjának végéig	A metódus végéig
Class/Struct (tagváltozók)	A teljes osztály/struct, függetlenül a deklaráció helyétől. Nem statikus tagváltozók csak nem statikus metódusokban használhatók.	?

Metódusok blokkjaira igaz, hogy egyazon blokkon belül nem lehet két azonos nevű változót deklarálni.

Láthatóság

Egy osztályon/struktúrán belül a metódusoknak lehet ugyanolyan nevű lokális változója, mint az osztály valamelyik tagváltozója.

Ebben az esetben a metóduson belül a lokális változó „eltakarja” a tagváltozót, ezért az nem látható.

A változó láthatósági köre:

a forráskódnak az a része, ahol nem takarja el a változót egy ugyanolyan nevű, szűkebb hatókörrel rendelkező változó.

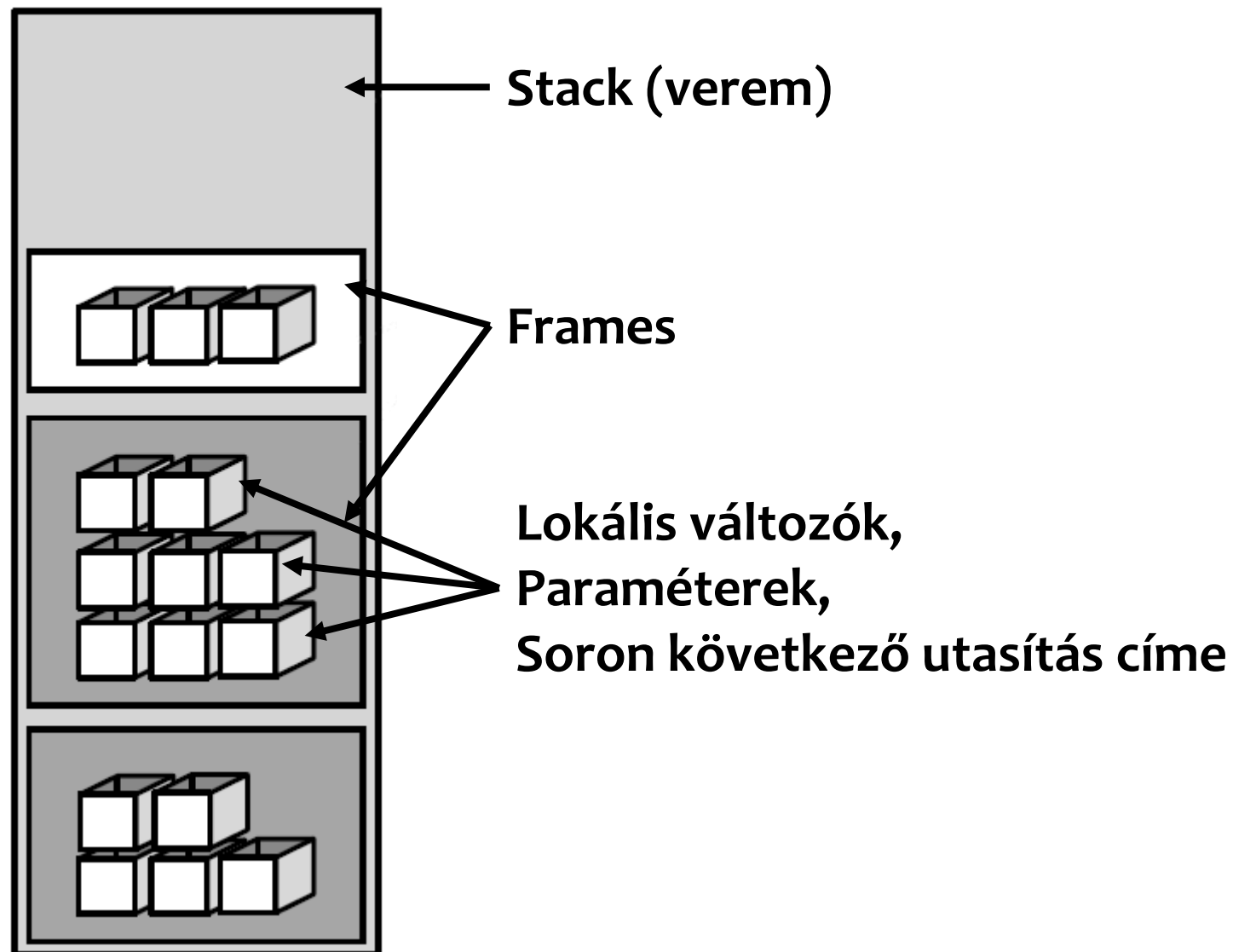
Változók tárolása a memóriában

- ⇒ Program indítása
- ⇒ Operációs rendszer kioszt egy memóriaterületet a programnak
- ⇒ A program több részre osztja fel a kiosztott memóriaterületet, melyeket kül. célra használ
 - ⇒ A két legnagyobb terület, ami adattárolásra szolgál:
 1. **Stack (verem)**
 2. **Heap (halom)**

Különböző dolgokat tárol, különböző módon

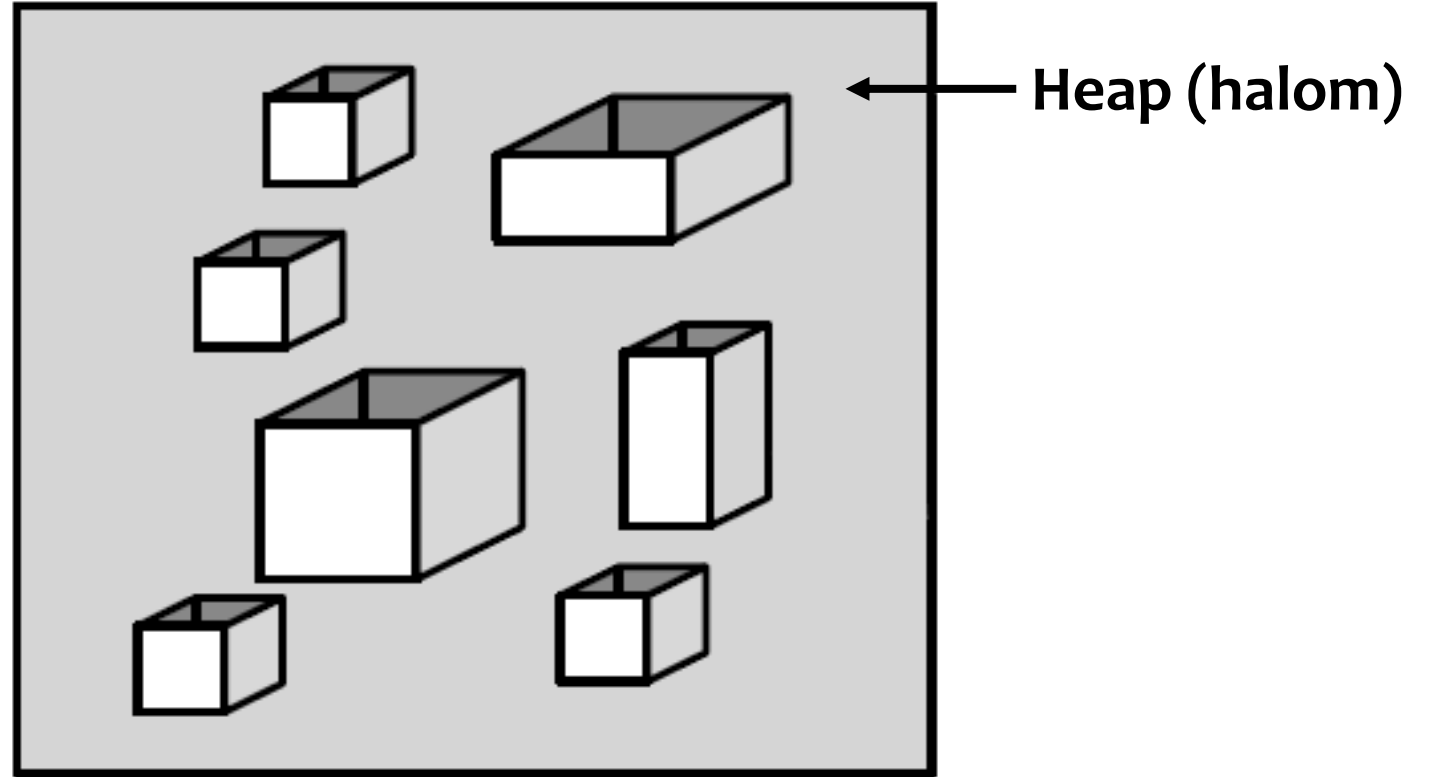
Stack (verem)

- ⇒ Lokális változók (metódusok blokkjában és paramétereként deklarált változók)
- ⇒ Egy metódus összes lokális változója egy frame-be kerül
- ⇒ A program aktuális állapotának nyomkövetése
- ⇒ Csak a felső keret elérhető
- ⇒ Amikor visszatérünk egy eljárásból, akkor a hozzá tartozó frame lekerül a verem tetejéről



Heap (halom)

- ⇒ Class tagváltozói
- ⇒ A benne levő dolgok bármikor elérhetők
- ⇒ Nincs semmilyen logikai elrendezés („kupac”)
- ⇒ Többszálú programok esetén minden szálnak lesz egy saját stack-je, de ugyanazon a heap-en osztoznak



Memória menedzselés

A nem használt memória területek felszabadítása:

- ⇒ Stack esetén a metódus végén automatikus
- ⇒ **Heap** esetén a .NET Framework kezeli a **Garbage Collector** („szemétgyűjtő”) segítségével.

Érték és referencia típusok

A C#-ban a változók két nagy csoportja:

⇒ Érték típusok:

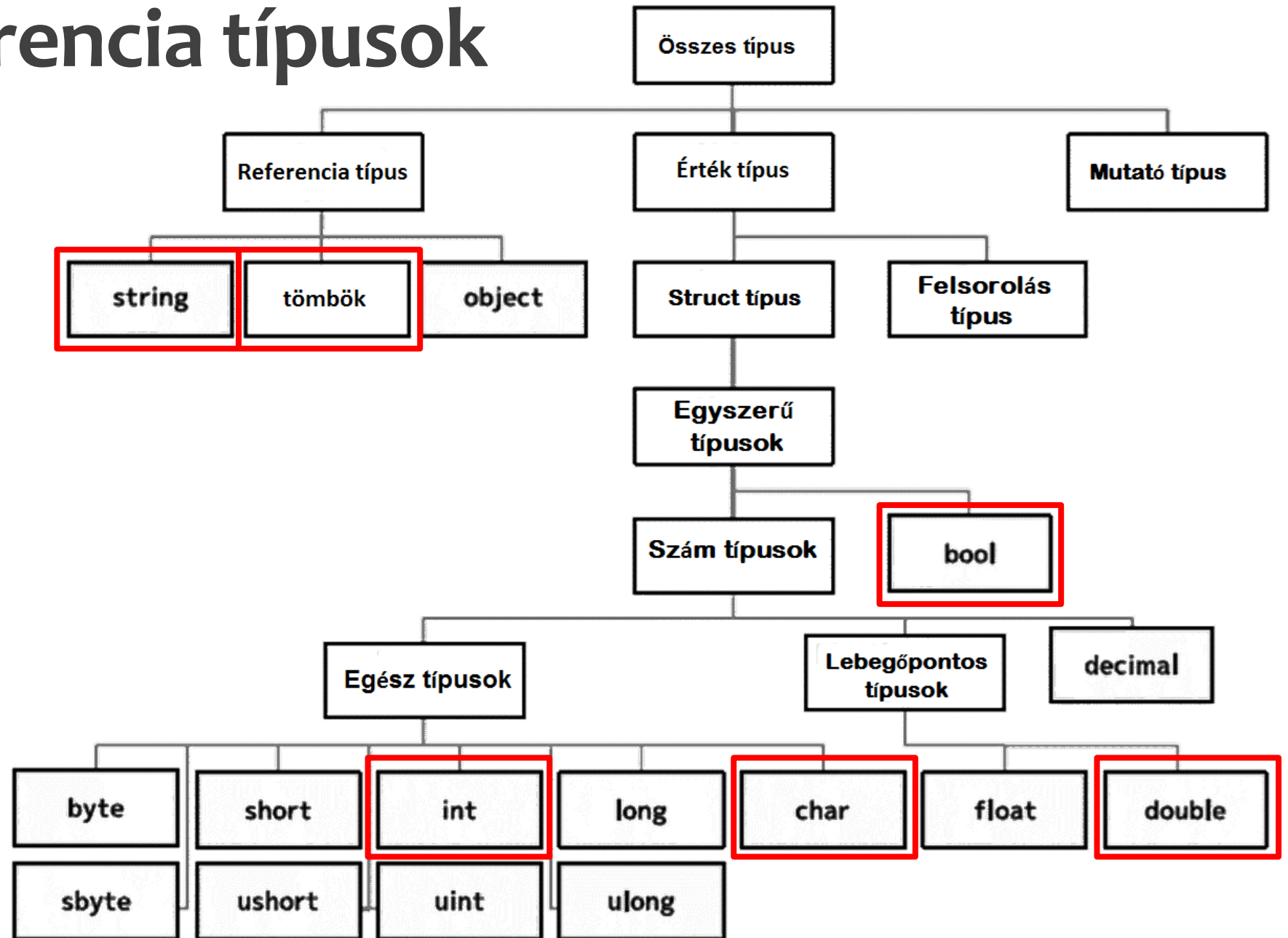
↳ Ahol a változó van, ott van a tartalma. (stack vagy heap)

⇒ Referencia típusok:

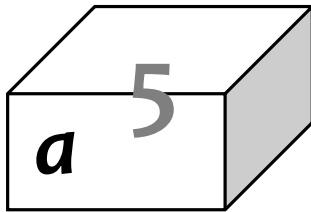
↳ Kétszeres tárolás:

↳ A változó csak egy referenciát tartalmaz (stack vagy heap), ami az „igazi” tartalom memóriacíme, ami a heap-ben van

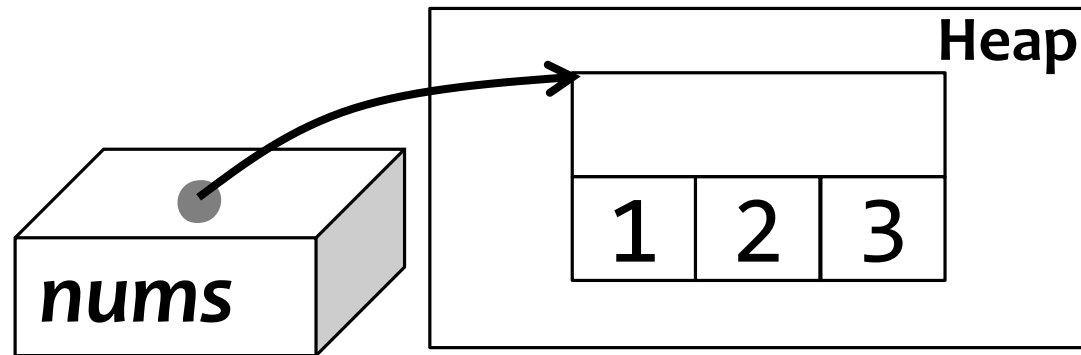
Érték és referencia típusok



Példák:

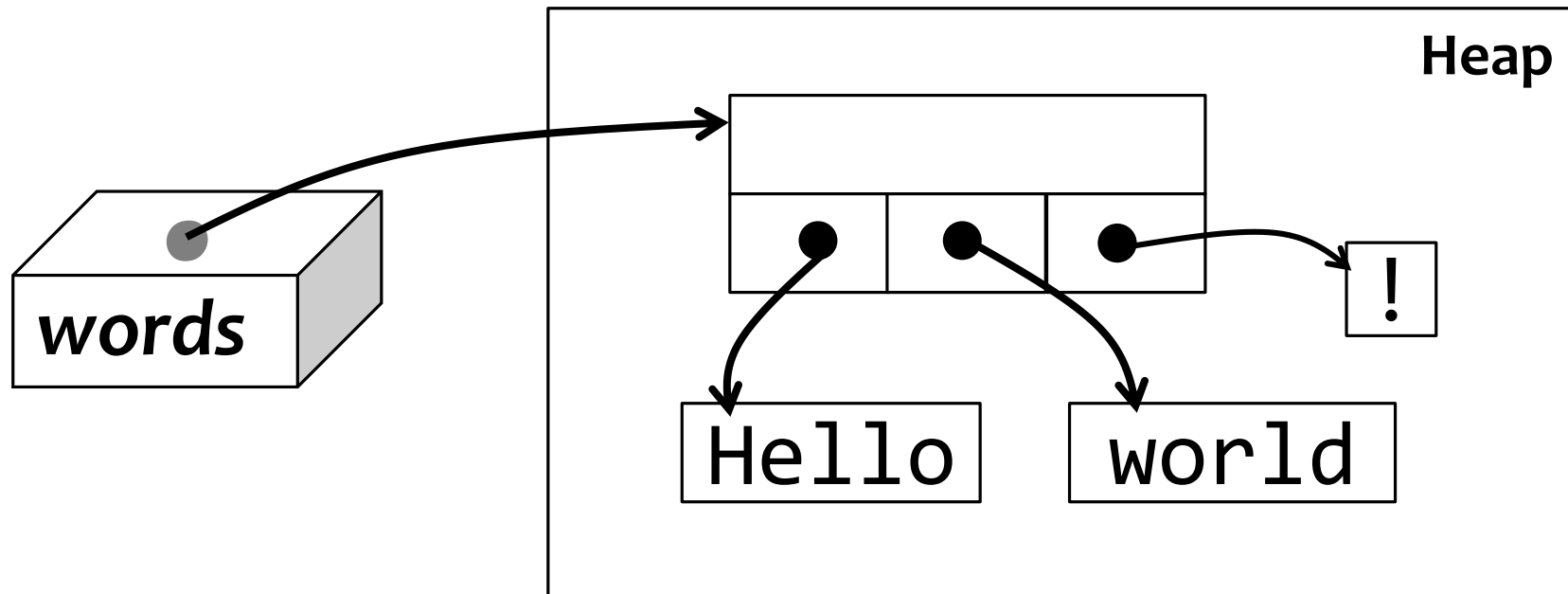
`int a=5;` 

`int[] nums=new int[3] {2,3,4};`



Példák:

```
string[] words=new string[3] {"Hello","world","!"};
```



Érték típus és referencia típus értékadásnál:

```
int x=3;
```

```
int y=a;
```

```
y++;
```

```
Console.WriteLine(x); //3
```

```
Console.WriteLine(y); //4
```

Nincs kapcsolat az x és y változó között.

Érték típus és referencia típus értékadásnál:

```
int[] x=new int[] {1,2,3};  
int[] y=a;  
y[0]=11;  
Console.WriteLine(x[0]); //11  
X[2]=33  
Console.WriteLine(y[2]); //33
```

Az x és y változó ugyanarra a memóriaterületre mutat, ugyanannak a tartalomnak alias nevei.