Objektumorientált Programozás VI.

Metódusok
Paraméterek átadása
Programozási tételek
Feladatok

Hallgatói Tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

Objektumorientált Programozás VI.

Metódusok

Paraméterek átadása

Programozási tételek

Feladatok

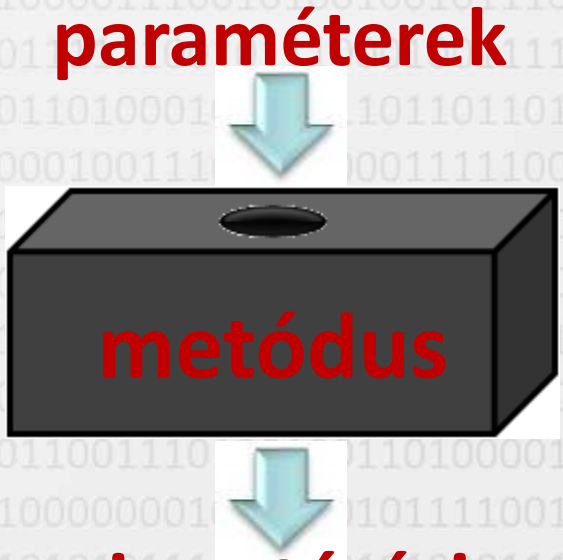
Metódusok

- A metódus egy kódblokk, amely utasítások sorozatát tartalmazza.
- A program azáltal futtatja ezeket az utasításokat, hogy "meghívja" a metódust és megszabja a szükséges paramétereit.
- C#-ban minden futtatandó utasítás egy metódusban helyezkedik el.
  - Eddigi programjainkat a Main() metódusba írtuk…
- A többször használt kódrészeket írjuk metódusba.
  - „Copy-Paste” helyett
  - Célszerű a hosszú metódusok feldarabolása az egyszerűbb értelmezés céljából

Metódusok

Séma:

visszatérési_típus metódusnév(paraméterek)
{metódustörzs}



Metódusok típusa

- A típus a visszaadott érték típusa vagy "void", ha nem adunk vissza semmit.
- Egy metódusnak legfeljebb egy visszatérési értéke van.
  - … de az lehet pl. tömb is…
- A visszatérési típus előtt állhatnak különféle módosítók.
  - Pl. static, abstract, override, new, láthatóságot jelző kulcsszavak. Részletesen később.

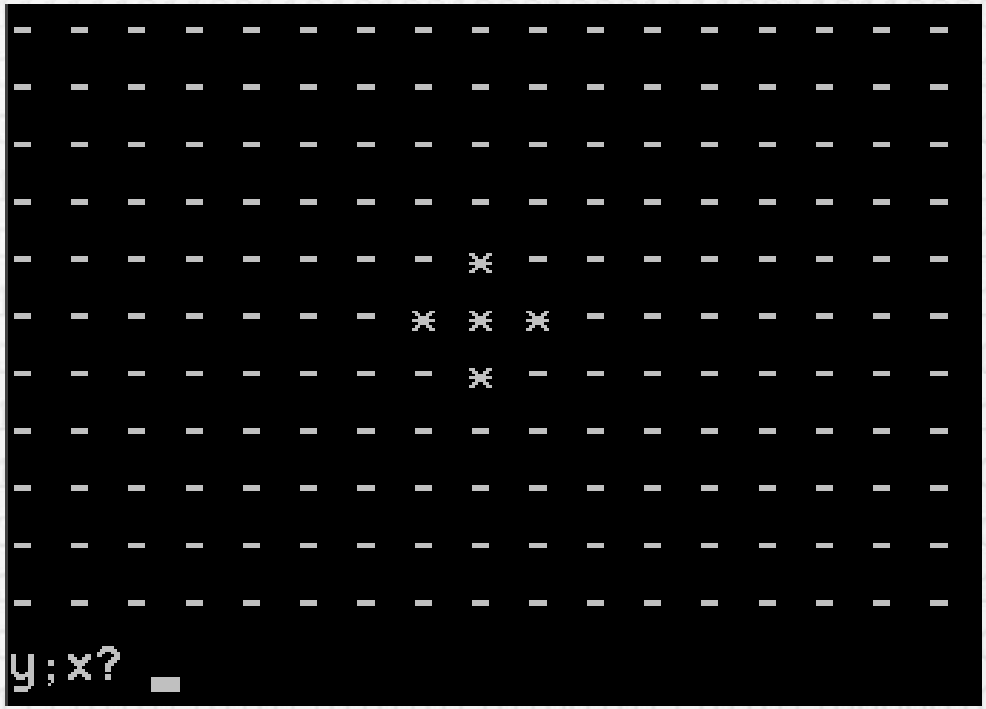
Metódustörzs

- A metódushoz tartozó utasítások, amelyek használhatják a metódusnak átadott paramétereket.
- A metódus visszatérési értékét a "return" kulcsszó után adjuk meg. Ennek hatására a metódusból azonnal visszatérünk a hívóhoz akkor is, ha még lennének további utasítások a return után.
- Ha a metódus több ágon érhet véget, akkor mindegyik ág végére szükséges return!
- Visszatérési érték nélküli (void) metódusnál – ha a program mindig a metódustörzs fizikai végénél fejeződik be – a return utasítás nem kötelező.

Változók hatóköre

- Egy blokkban deklarált változók csak a deklarálástól kezdve a blokk végéig elérhetőek.
  - Következmény: az egyik metódusban deklarált x változó nem ugyanaz, mint a másik metódusbeli x változó.
- A hívó környezet változói nem érhetőek el a metódusban.
  - Ezért szükséges a paraméter átadás és a visszatérési érték.
  - Közös adat használható: globális változók, használatuk nem javasolt

Feladat



A felhasználó minden lépésével kiválaszt egy mezőt. Ekkor a mező és annak szomszédjai az ellentettjükre módosulnak.

Használjunk paraméter nélküli metódusokat!

Objektumorientált Programozás VI.

Metódusok

Paraméterek átadása

Programozási tételek

Feladatok

Egyszerű példa

- **Feladat: téglalap területének számítása metódussal**

```
static int terület(int a, int b) // paraméterek átadása
{
    return a * b;
}
static void Main(string[] args)
{
    int egyikoldal = 5;
    int másikoldal = 7;
    Console.WriteLine("A téglalap területe: " + terület(egyikoldal, másikoldal));
    Console.ReadLine();
}
```

Metódusok paraméterei

- A paramétereknél megadandó a típus és az a név, amelyen a metódustörzsben a paraméterre hivatkozunk.
- A paraméter átadásnak kétféle módja van: érték szerinti és cím szerinti.
- **Érték szerinti**: a paraméterként megadott változóról másolat készül, a metódusban ezzel a másolattal dolgozunk. Tehát ha a metódusban megváltoztatjuk a paraméter értékét, az az eredetire nincs hatással, a hívó környezetben nem érvényesül.
- **Cím szerinti**: egy referenciát adunk át a paraméterként megadott változóra, nem készül másolat, tehát a metódusban változtatható a paraméter értéke, és ez a hívó környezetben is érvényesül.

Metódusok paraméterei

- **Az érték szerinti paraméter átadás az alapértelmezett.**
  - Referencia típusú változóknál az érték szerinti paraméter átadás azt jelenti, hogy a változóban tárolt cím másolódik le és adódik át, azaz ha a metódusban módosítjuk a referencia által hivatkozott objektumot, annak a hatása kívül is látszik.
- **Ha érték típusú paramétert szeretnénk cím szerint átadni (hogy a metódusban megváltoztatható legyen az értéke), arra a *ref* vagy az *out* kulcsszó használatával van lehetőség.**
- **DE: két fajta változótípus van: érték és referenciatípus → 4 fajta paraméterátadási mód**

Értéktípusok érték szerinti paraméterátadása

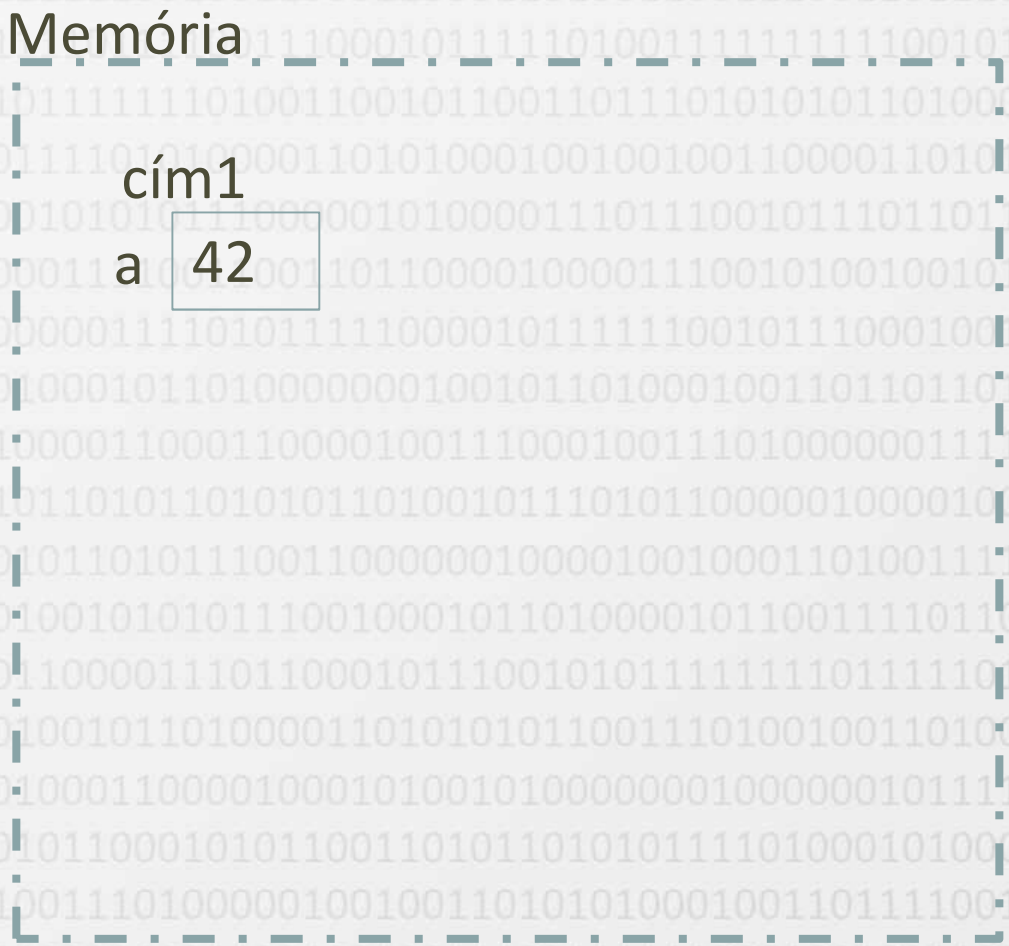
```
static void novel(int bemenet)
{
    bemenet++;
}
static void csere(int elso, int masodik)
{
    int temp = elso;
    elso = masodik;
    masodik = temp;
}
```

Értéktípusok érték szerinti paraméterátadása

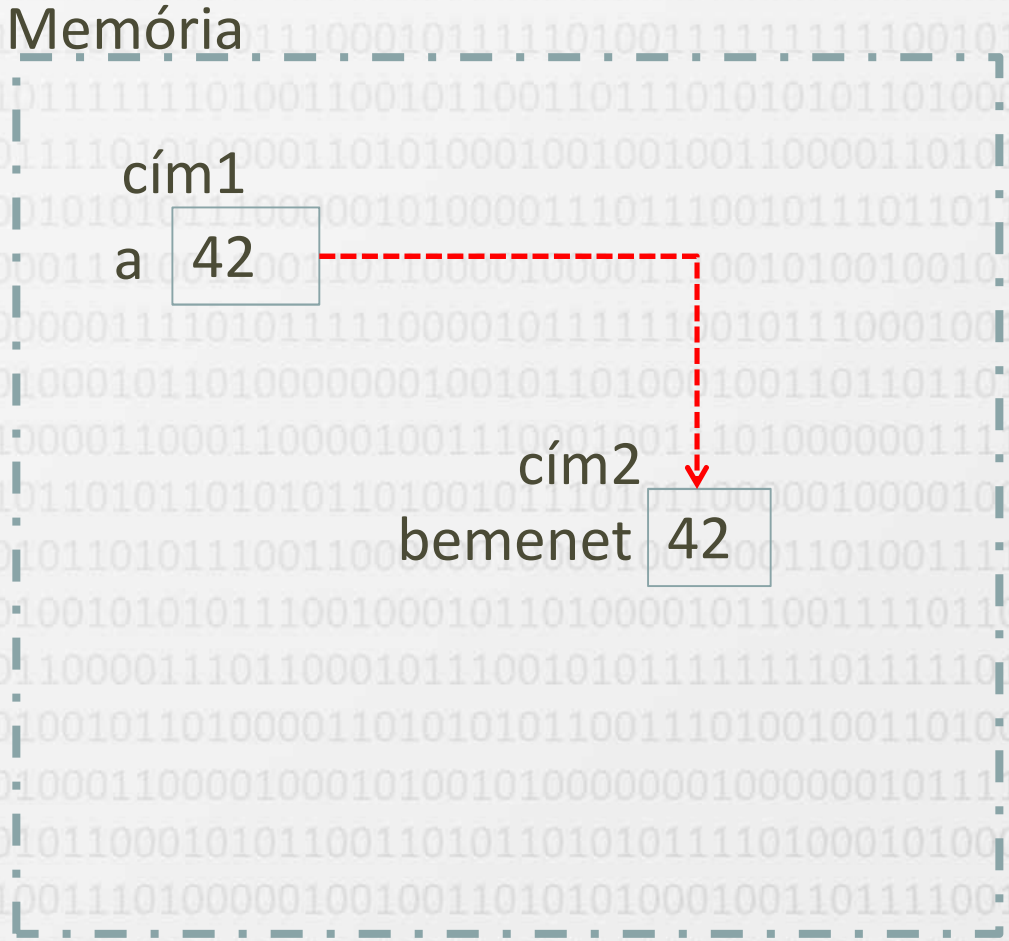
```
static void Main(string[] args)
{
    int a = 42, b = 23;
    novel(a);
    csere(a, b);
}
```

- A híváskor a két változó értékéről másolat képződik, és a másolat kerül át paraméterként a hívott eljárásokba
- A hívott eljárásokban a paraméterek módosítása semmilyen hatással nincs ezen változók értékére

Értéktípusok érték szerinti paraméterátadása



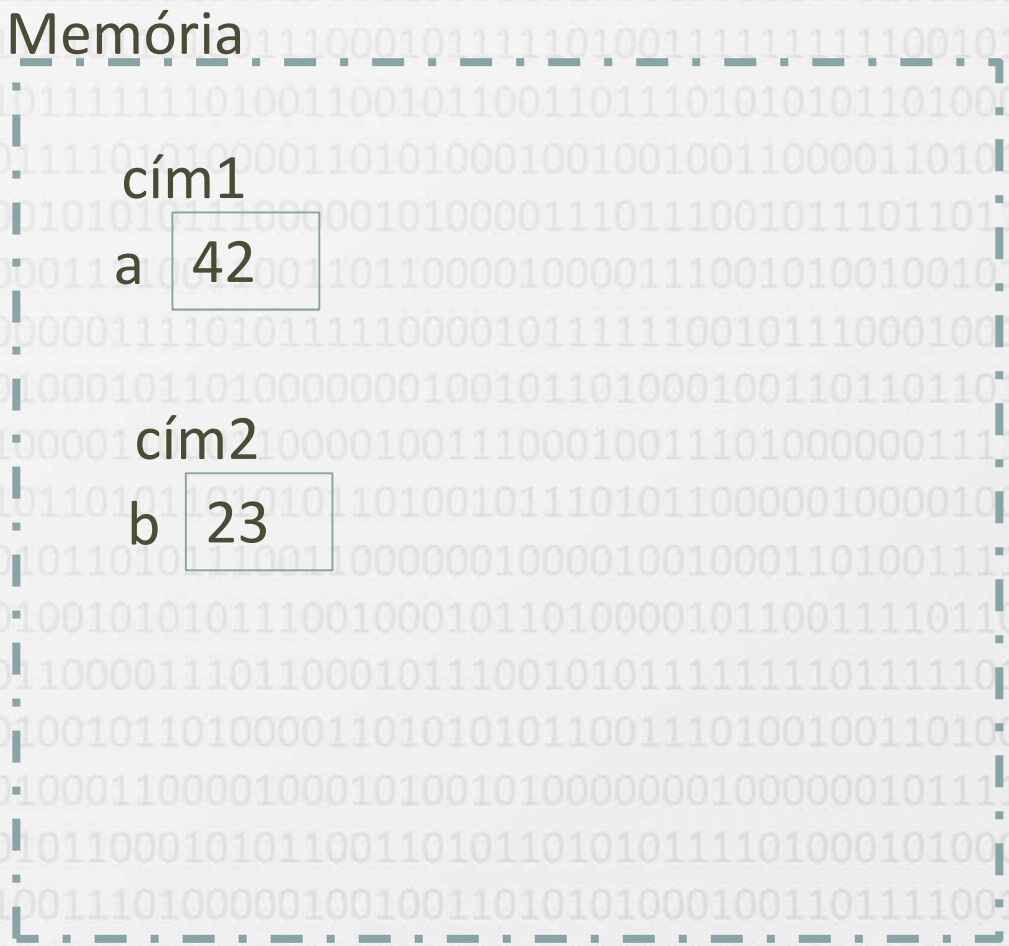
Értéktípusok érték szerinti paraméterátadása




```
int a=42;
novel(a);
```

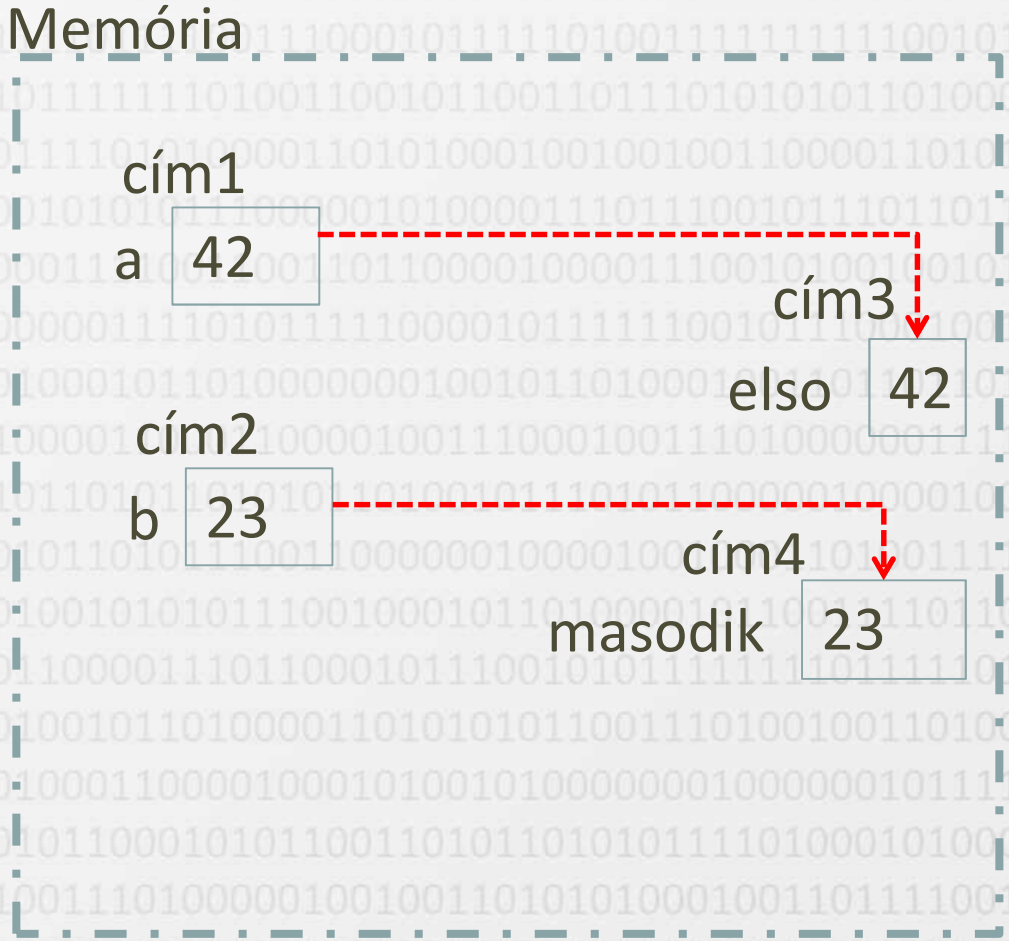
Értéktípusok érték szerinti paraméterátadása

Értéktípusok érték szerinti paraméterátadása



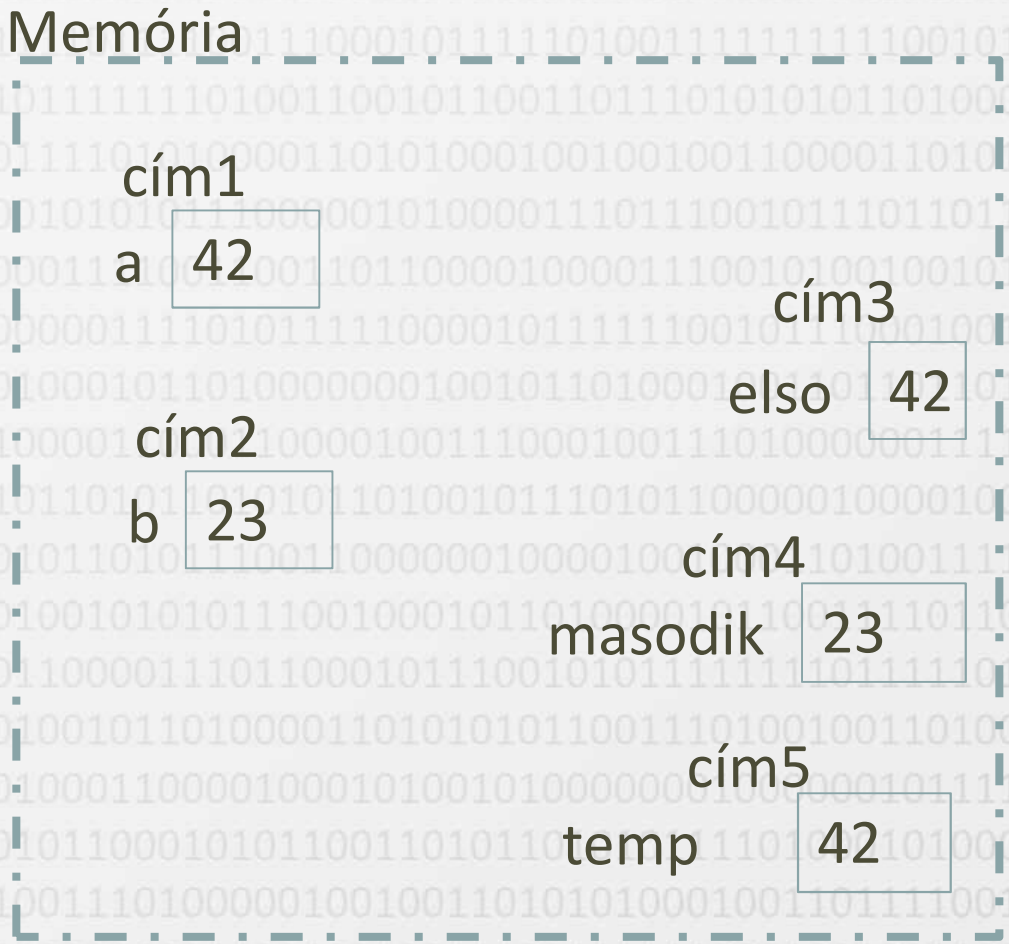

```
int a=42, b=23;
```

Értéktípusok érték szerinti paraméterátadása



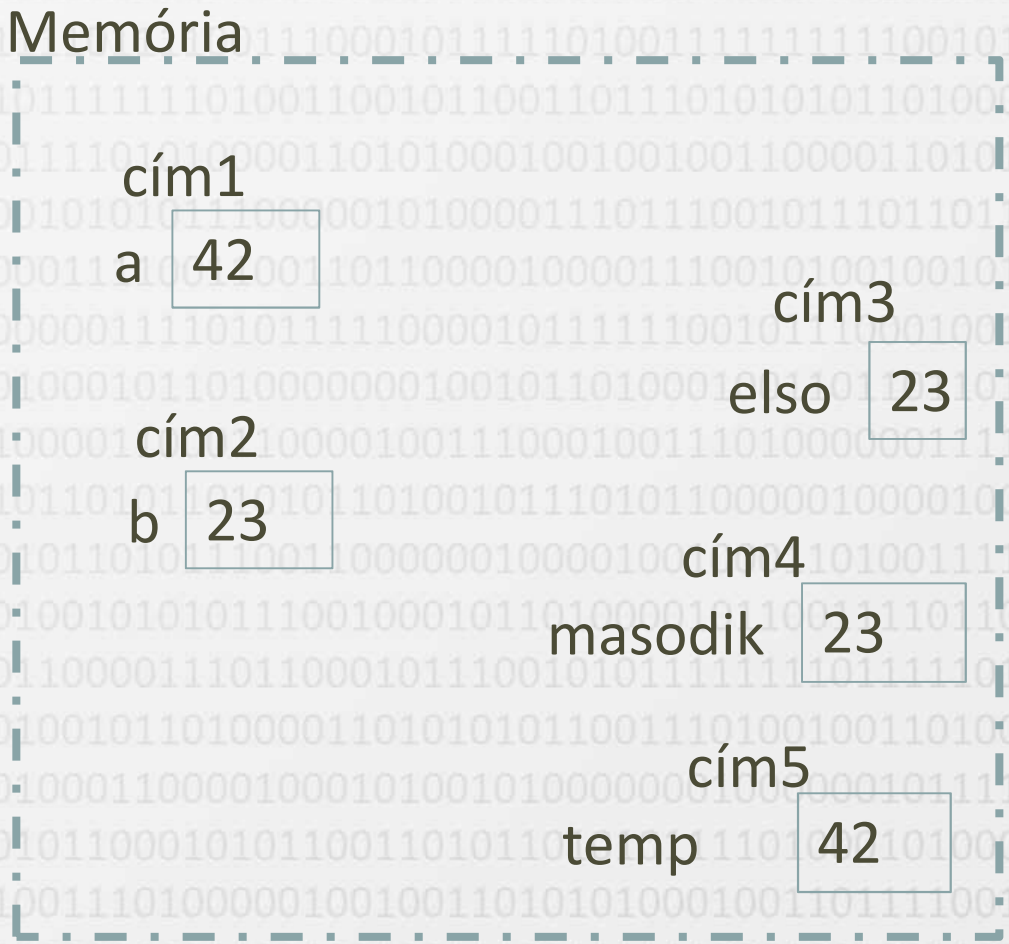

```
int a=42, b=23;
csere(a,b);
```

Értéktípusok érték szerinti paraméterátadása



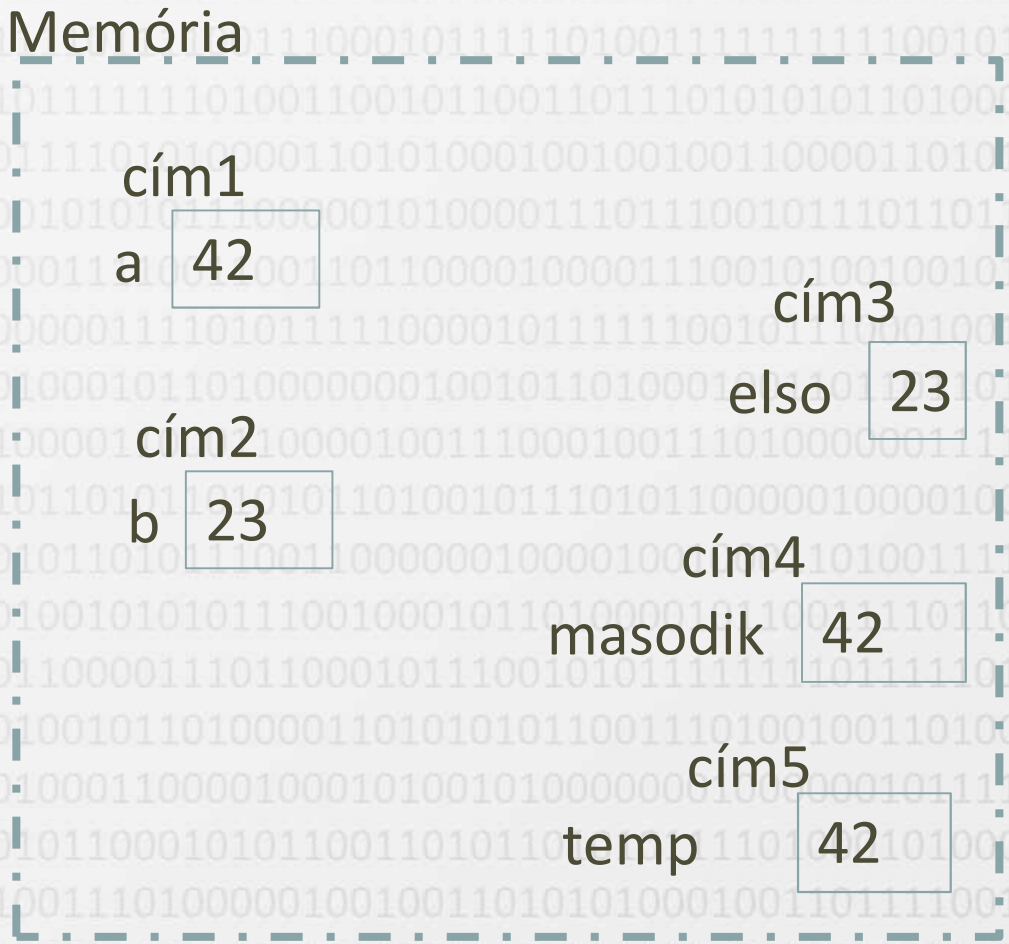

```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
```

Értéktípusok érték szerinti paraméterátadása




```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
        elso=masodik;
```

Értéktípusok érték szerinti paraméterátadása




```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
        elso=masodik;
        masodik=temp;
```

Értéktípusok cím szerinti paraméterátadása

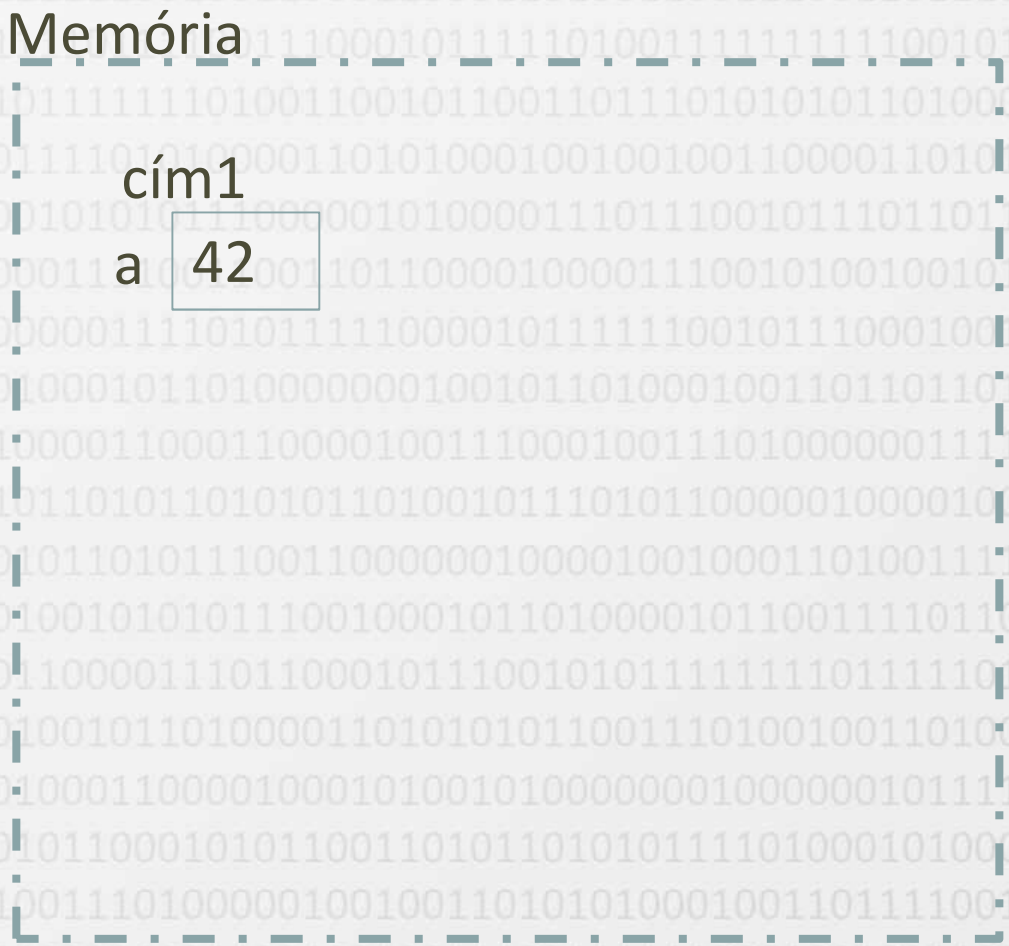
```
static void novel(ref int bemenet)
{
    bemenet++;
}
static void csere(ref int elso, ref int masodik)
{
    int temp = elso;
    elso = masodik;
    masodik = temp;
}
```

Értéktípusok cím szerinti paraméterátadása

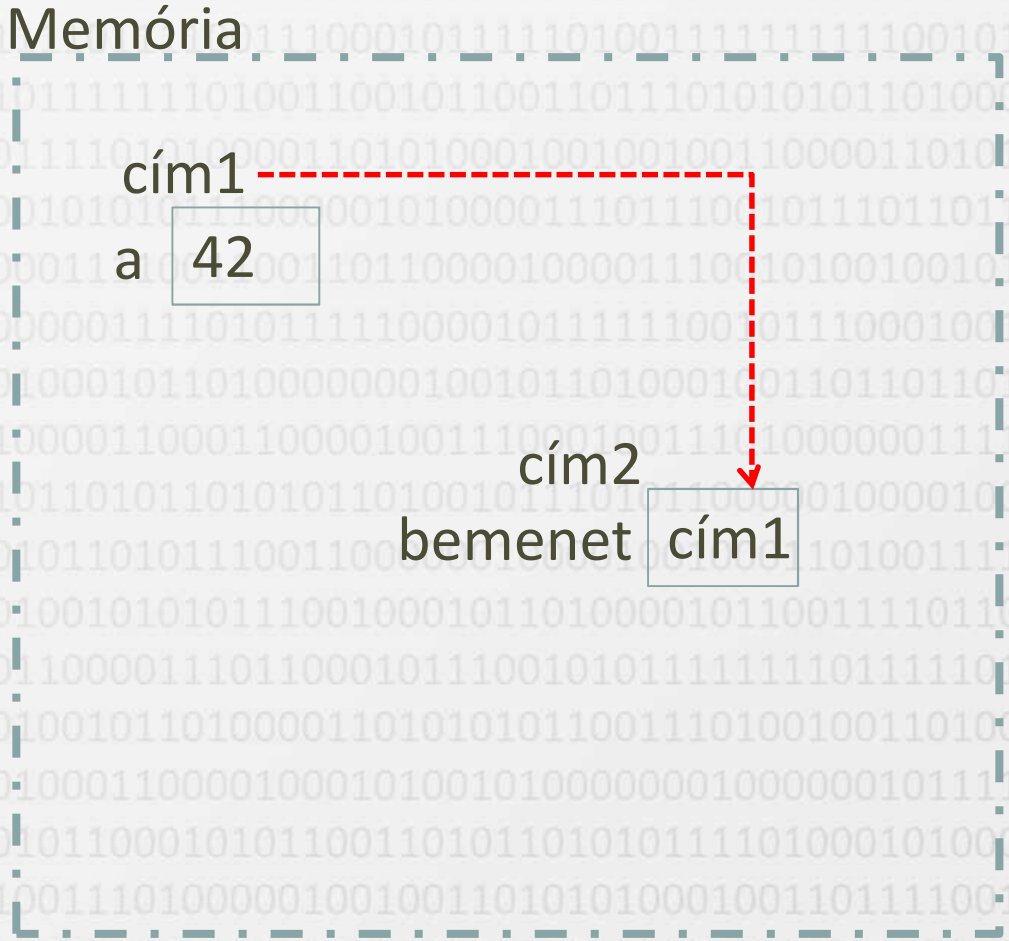
```
static void Main(string[] args)
{
    int a = 42, b = 23;
    novel(ref a);
    csere(ref a, ref b);
}
```

- A híváskor a két változó CÍMÉRŐL másolat képződik, és a CÍM kerül át paraméterként a hívott eljárásokba
- A hívott eljárásokban a paraméterek módosítása gyakorlatilag ezen változók módosítását jelenti

Értéktípusok cím szerinti paraméterátadása

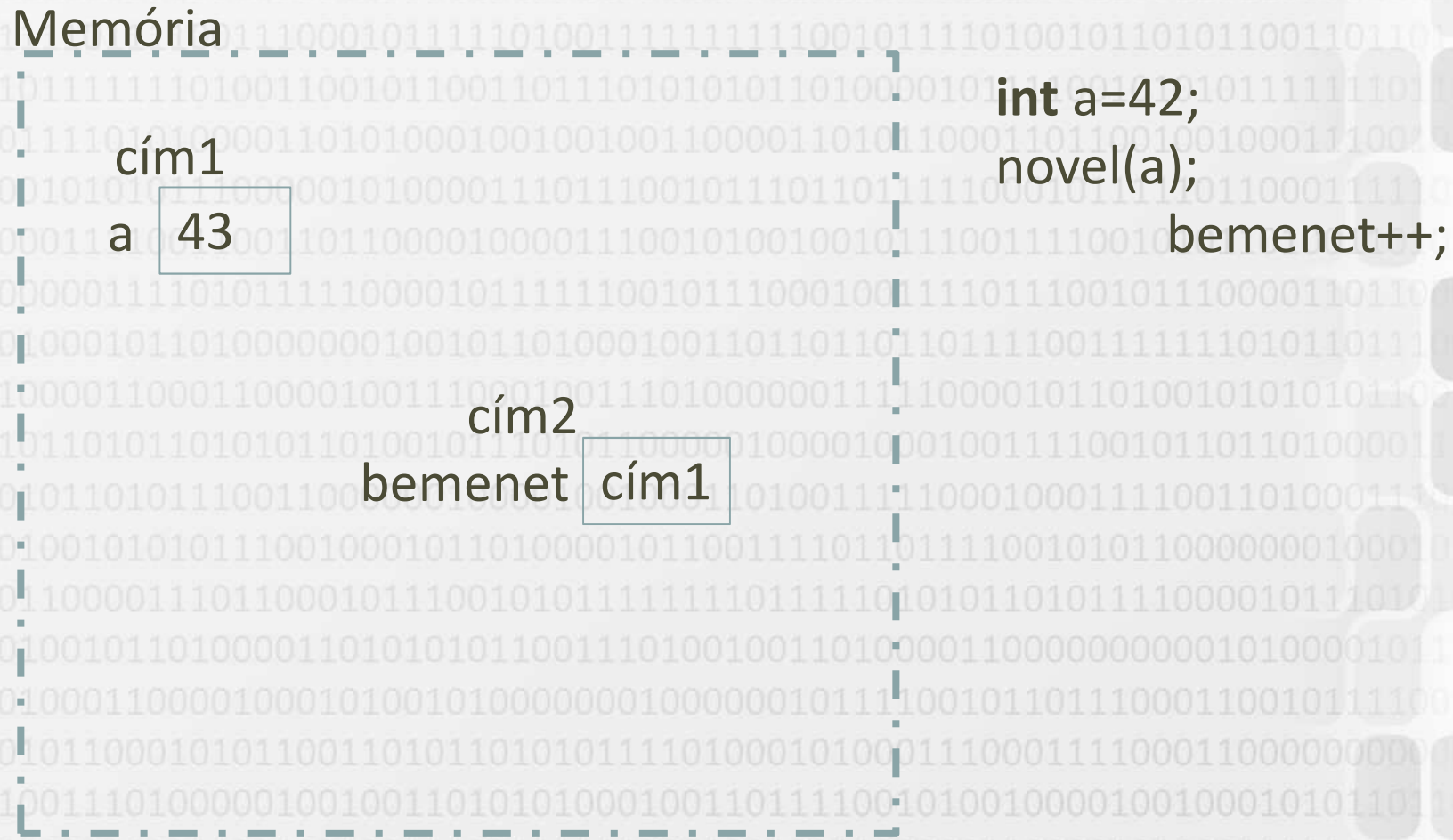


Értéktípusok cím szerinti paraméterátadása

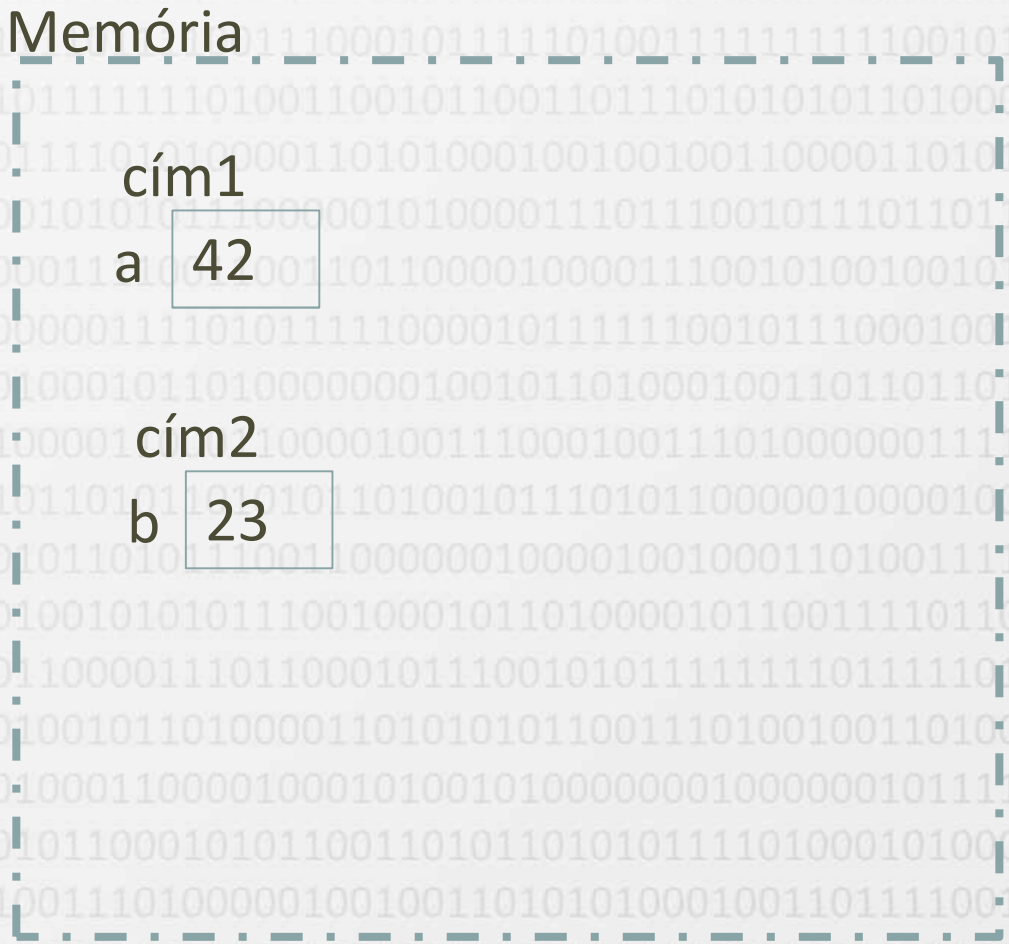



```
int a=42;
novel(a);
```

Értéktípusok cím szerinti paraméterátadása

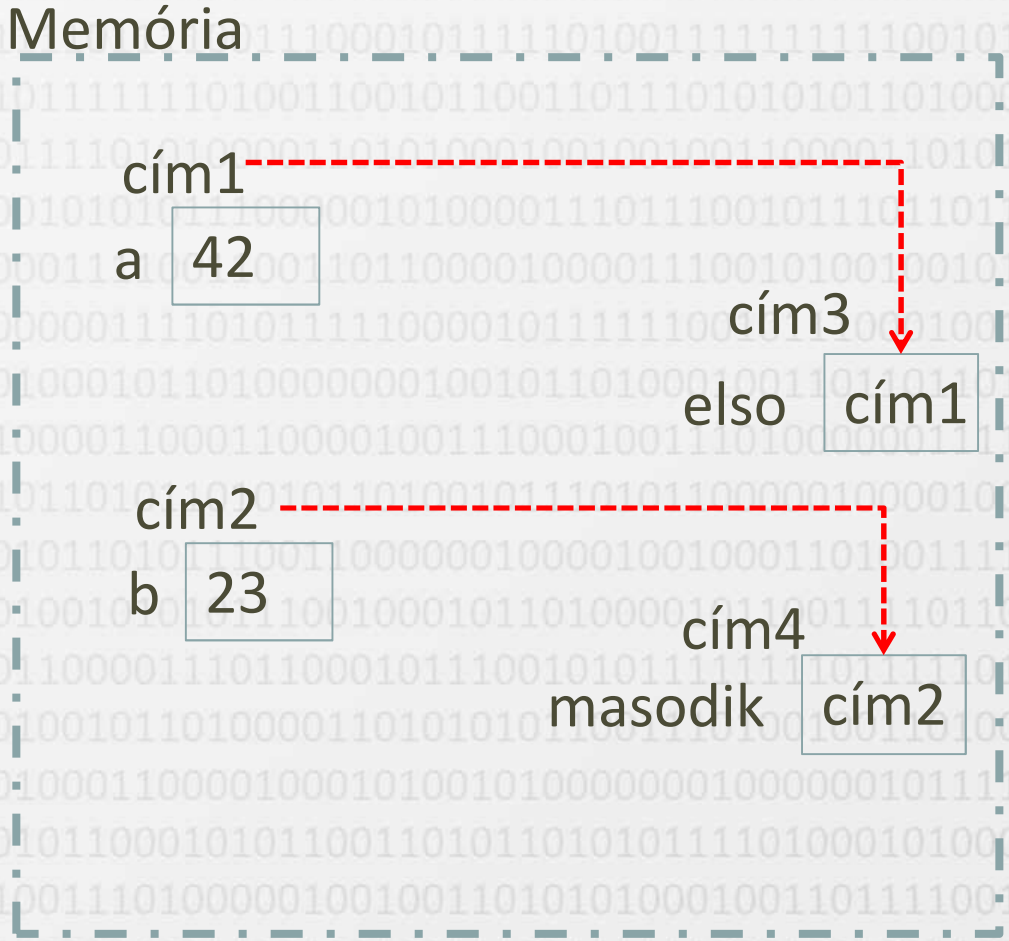


Értéktípusok cím szerinti paraméterátadása



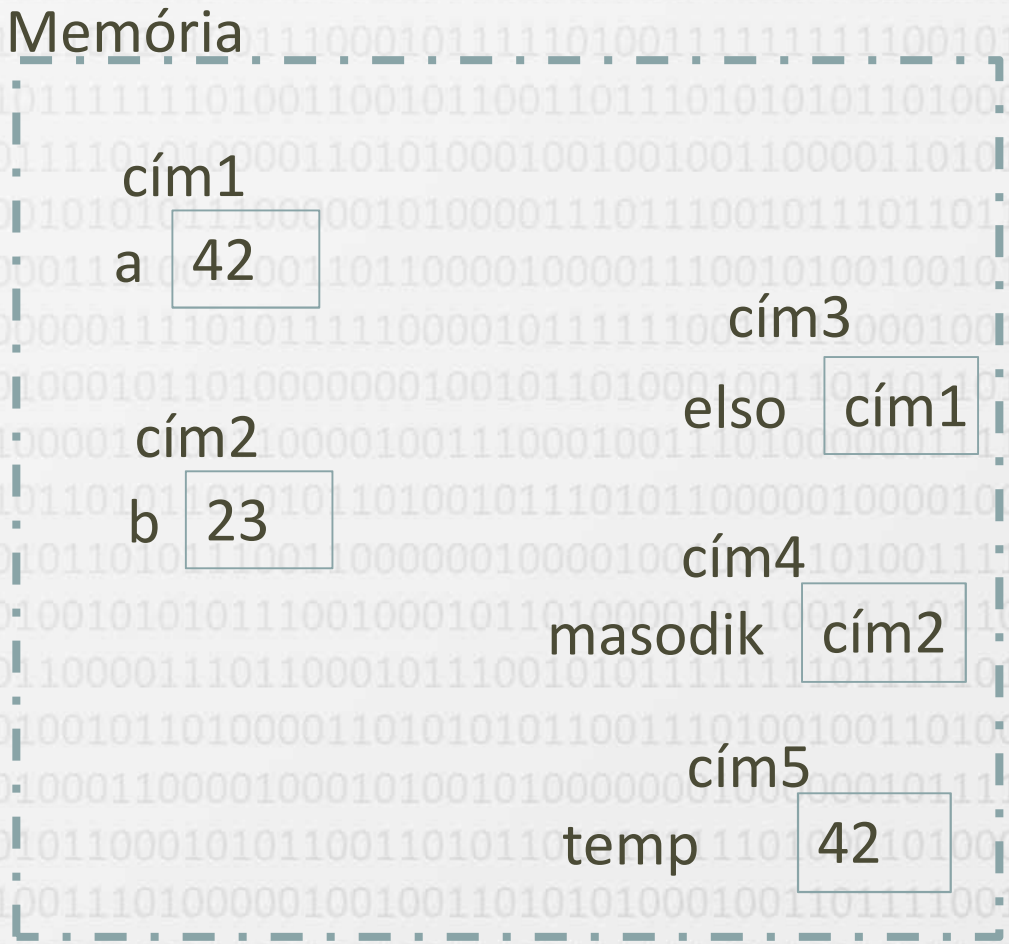
int a=42, b=23;

Értéktípusok cím szerinti paraméterátadása



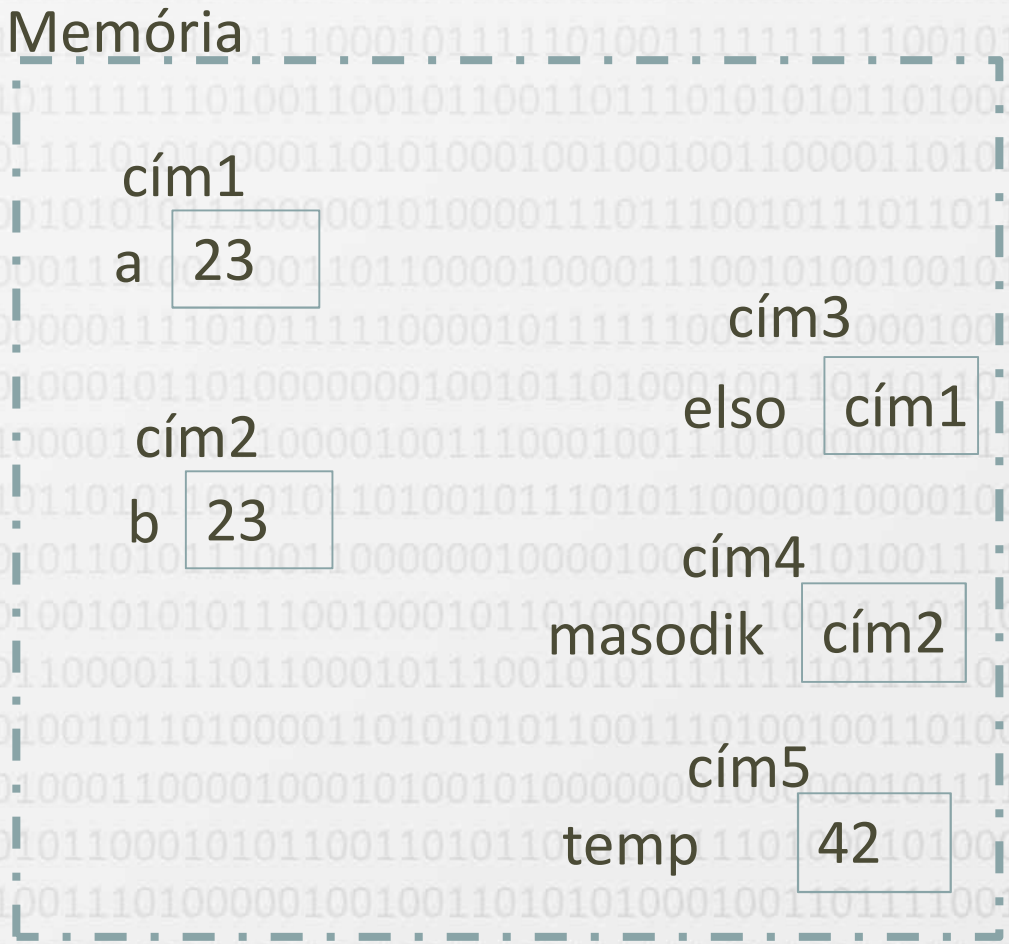

```
int a=42, b=23;
csere(a,b);
```

Értéktípusok cím szerinti paraméterátadása



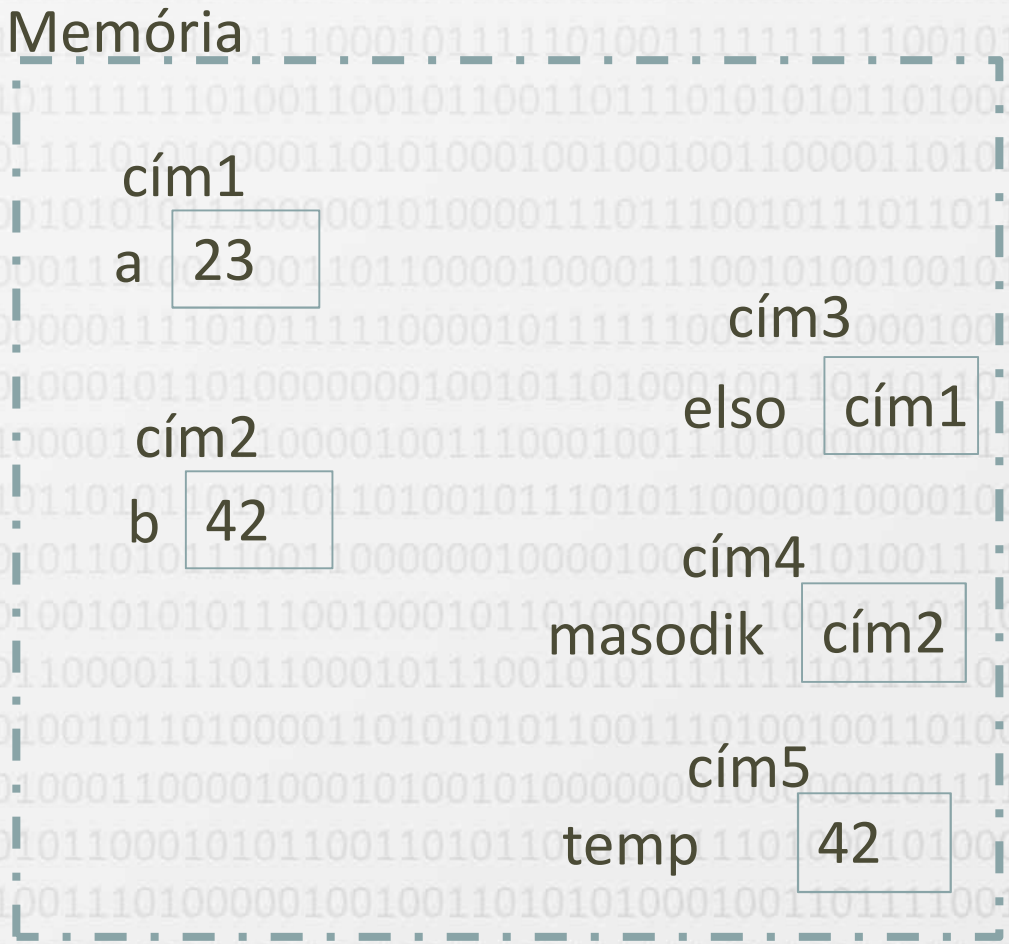

```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
```

Értéktípusok cím szerinti paraméterátadása




```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
        elso=masodik;
```

Értéktípusok cím szerinti paraméterátadása




```
int a=42, b=23;
csere(a,b);
        int temp;
        temp=elso;
        elso=masodik;
        masodik=temp;
```

Ref.típusok érték szerinti paraméterátadása

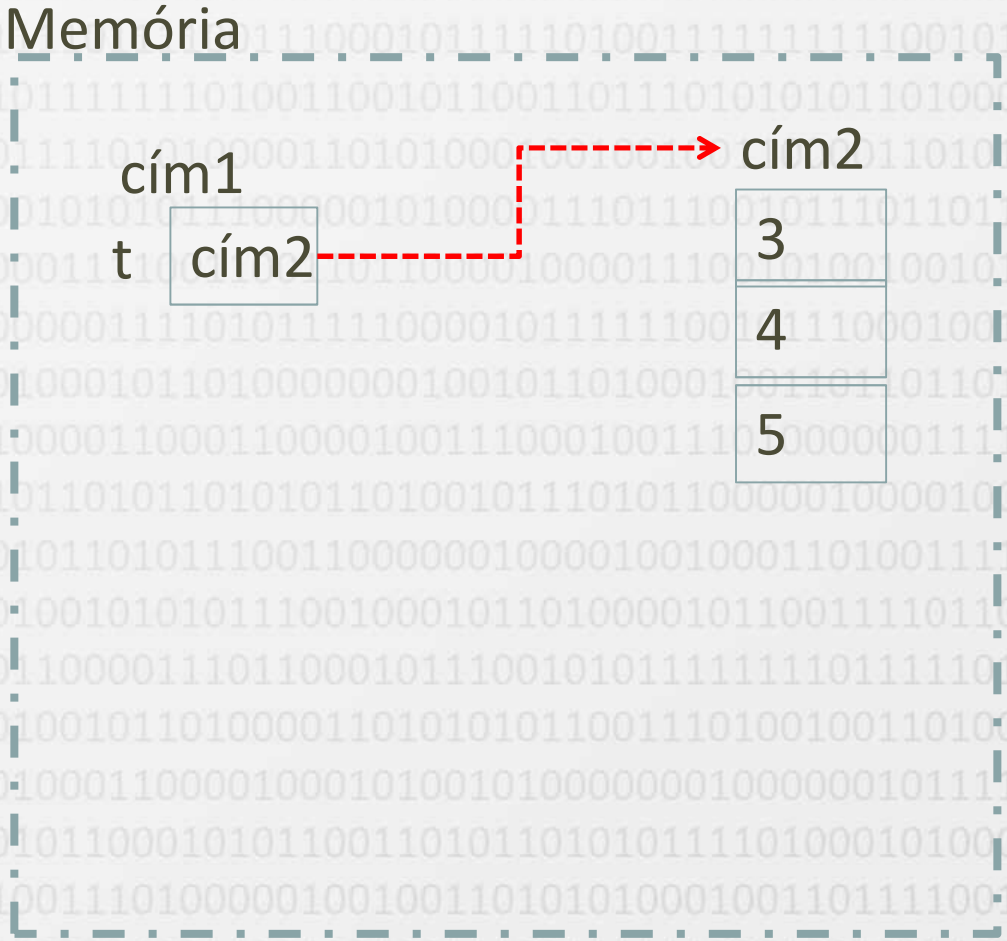
```
static void tombkez1(int[] arr)
{
    arr[1] = 42;
}
static void tombkez2(int[] arr)
{
    arr = new int[3] { 23, 23, 23 };
}
```

Ref.típusok érték szerinti paraméterátadása

```
static void Main(string[] args)
{
    int[] t = { 3, 4, 5 };
    tombkez1(t);
    tombkez2(t);
}
```

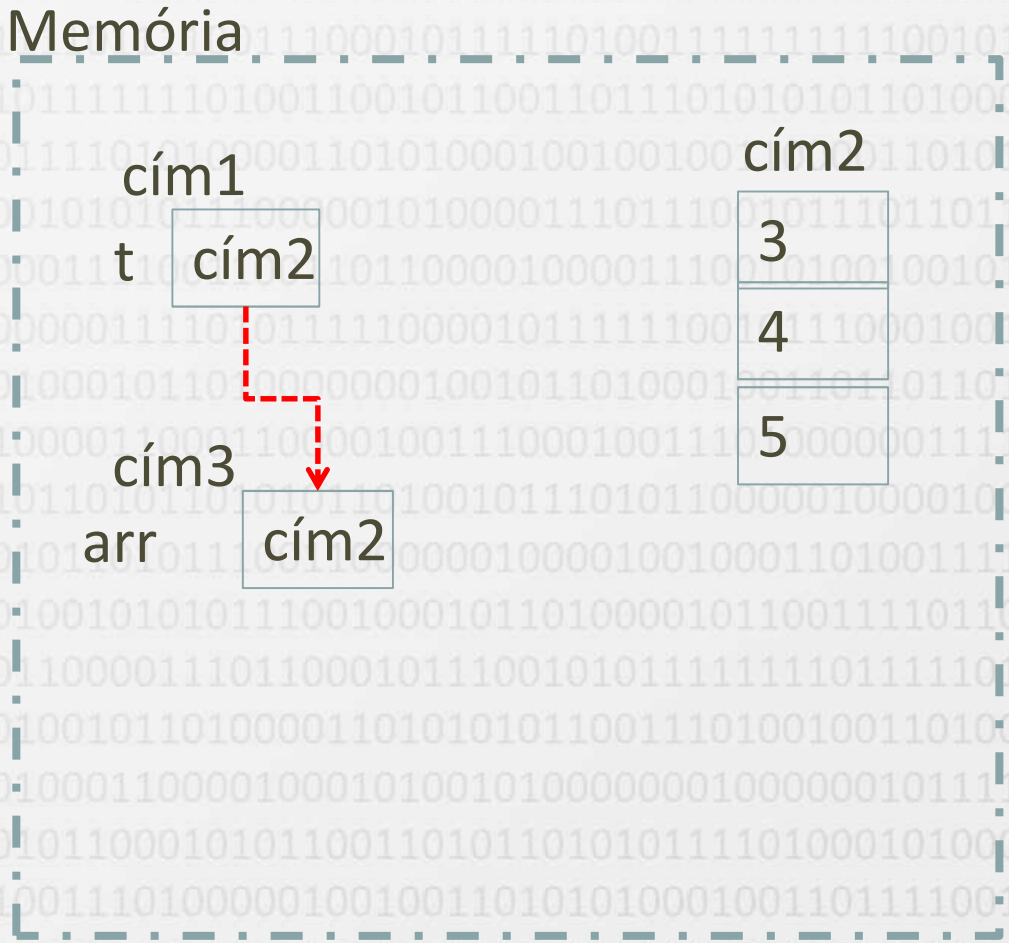
- **Az elv ugyanaz: híváskor a változó értékéről másolat képződik, és az érték kerül át paraméterként a hívott eljárásba ➔ de most az érték egy cím!**
- **A hívott eljárásokban a nem lehet módosítani ezt a változót ➔ de a hivatkozott memóriaterületet IGEN**

Ref.típusok érték szerinti paraméterátadása



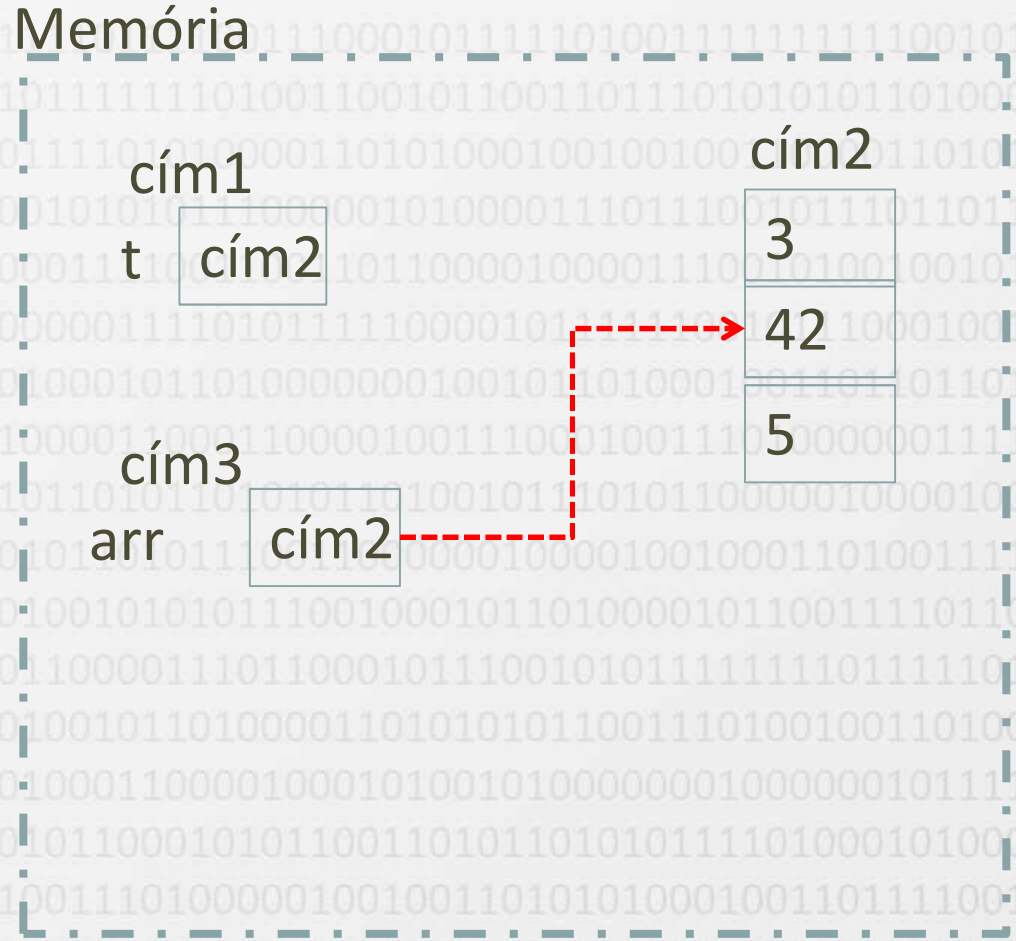
int[] t= {3, 4, 5};

Ref.típusok érték szerinti paraméterátadása



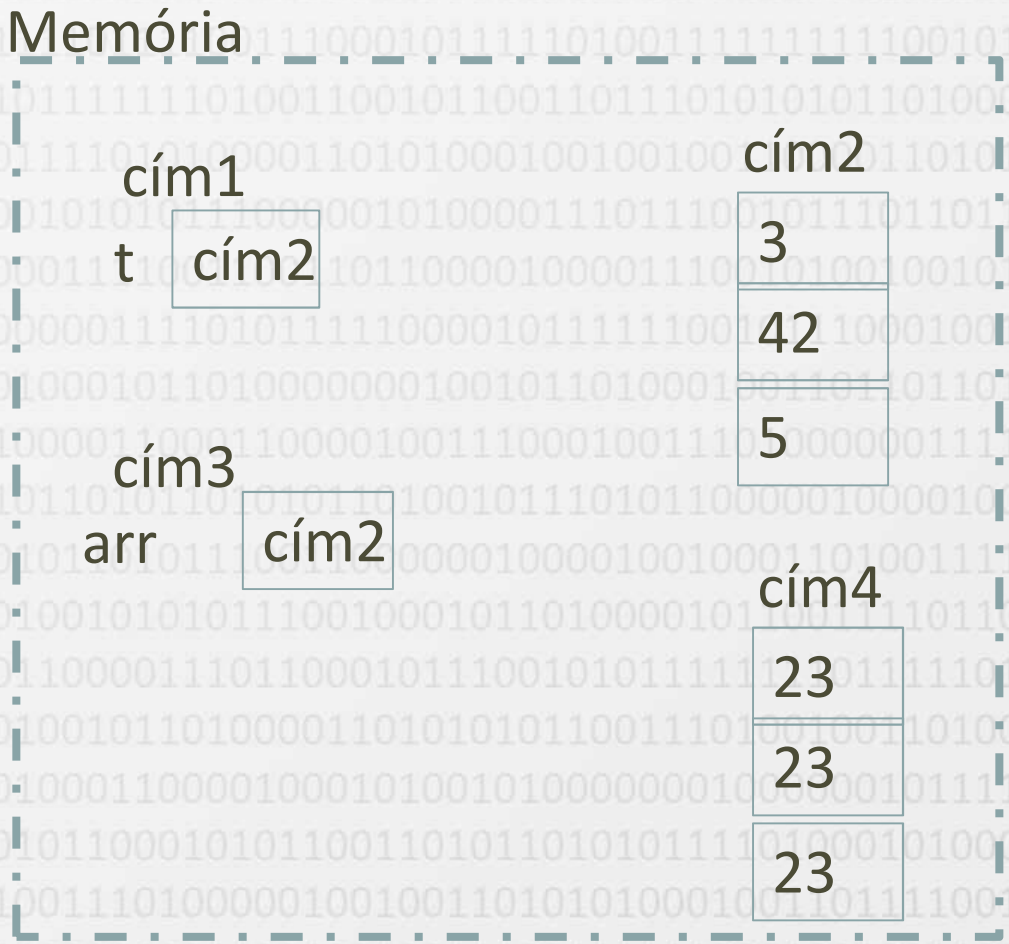
int[] t= {3, 4, 5};
tombkez1(t);

Ref.típusok érték szerinti paraméterátadása



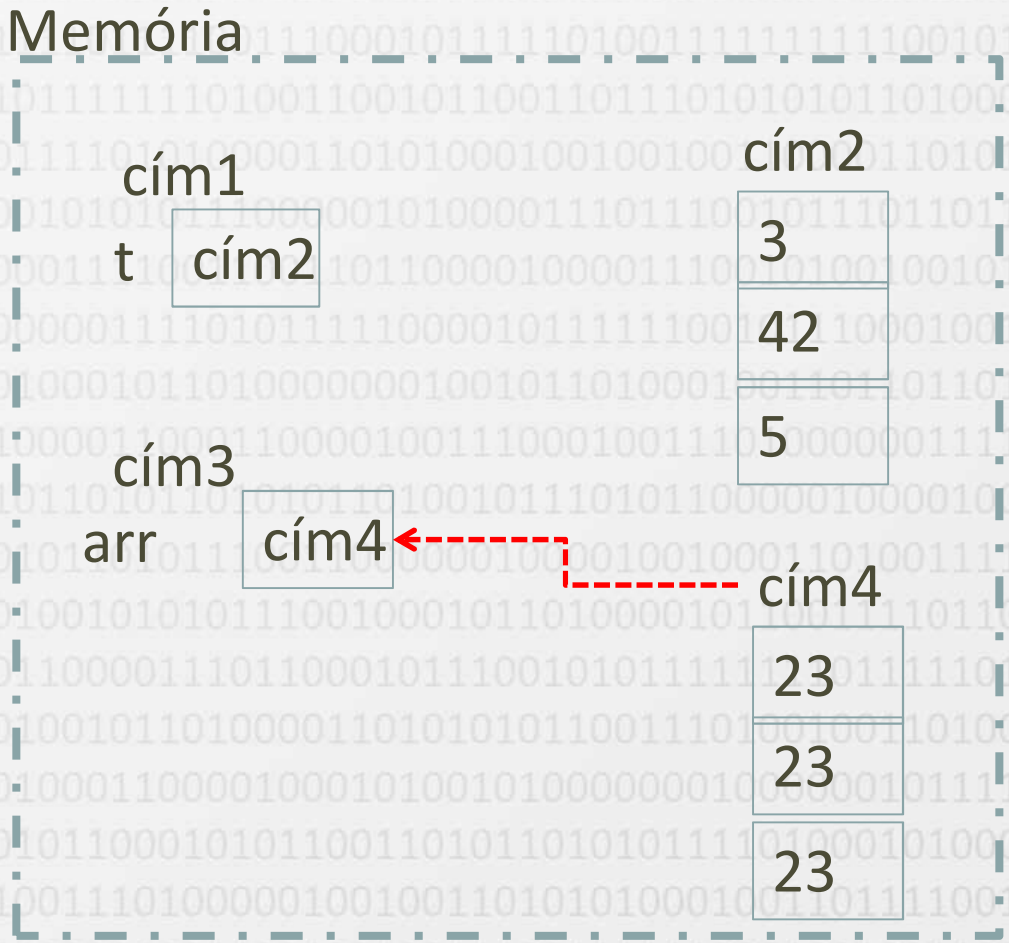

```
int[] t= {3, 4, 5};
tombkez1(t);
        arr[1]=42;
```

Ref.típusok érték szerinti paraméterátadása




```
int[] t= {3, 4, 5};
tombkez1(t);
        arr[1]=42;
tombkez2(t);
        arr=new int[3]
        { 23, 23, 23};
```

Ref.típusok érték szerinti paraméterátadása




```
int[] t= {3, 4, 5};
tombkez1(t);
        arr[1]=42;
tombkez2(t);
        arr=new int[3]
        { 23, 23, 23};
```

Ref.típusok cím szerinti paraméterátadása

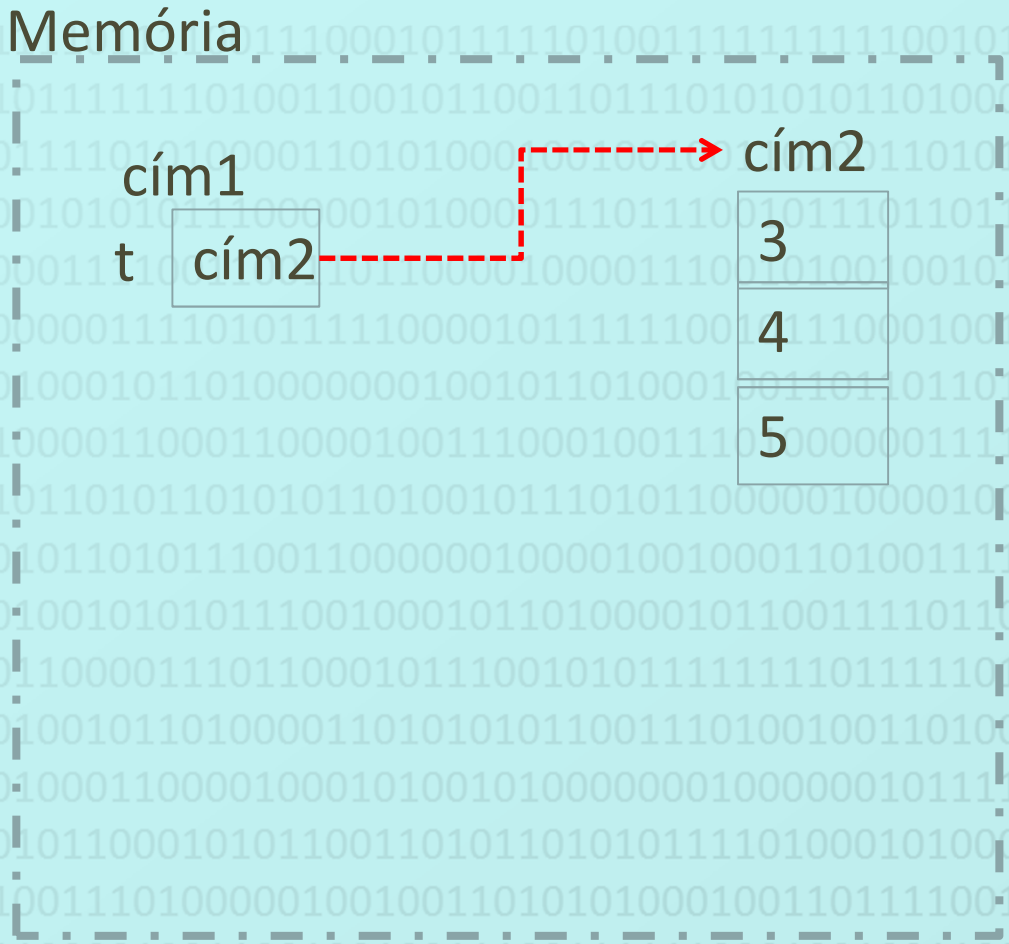
```
static void tombkez1(ref int[] arr)
{
    arr[1] = 42;
}
static void tombkez2(ref int[] arr)
{
    arr = new int[3] { 23, 23, 23 };
}
```

Ref.típusok cím szerinti paraméterátadása

```
static void Main(string[] args)
{
    int[] t = { 3, 4, 5 };
    tombkez1(ref t);
    tombkez2(ref t);
}
```

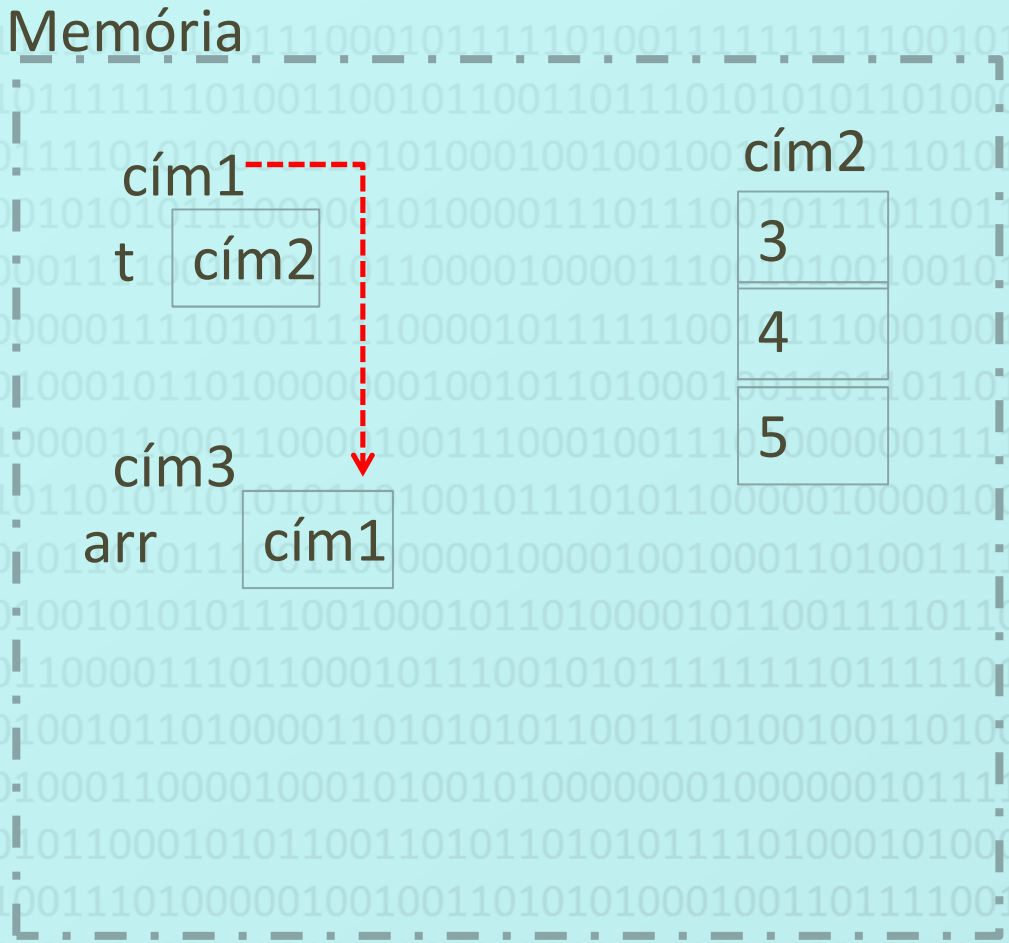
- Az elv ugyanaz: híváskor a változó CÍMÉRŐL másolat képződik, és a CÍM kerül át paraméterként a hívott eljárásokba ➔ Referenciára mutató referencia
- A hívott eljárásokban ezt a változót és a hivatkozott memóriaterületet is lehet módosítani

Ref.típusok cím szerinti paraméterátadása



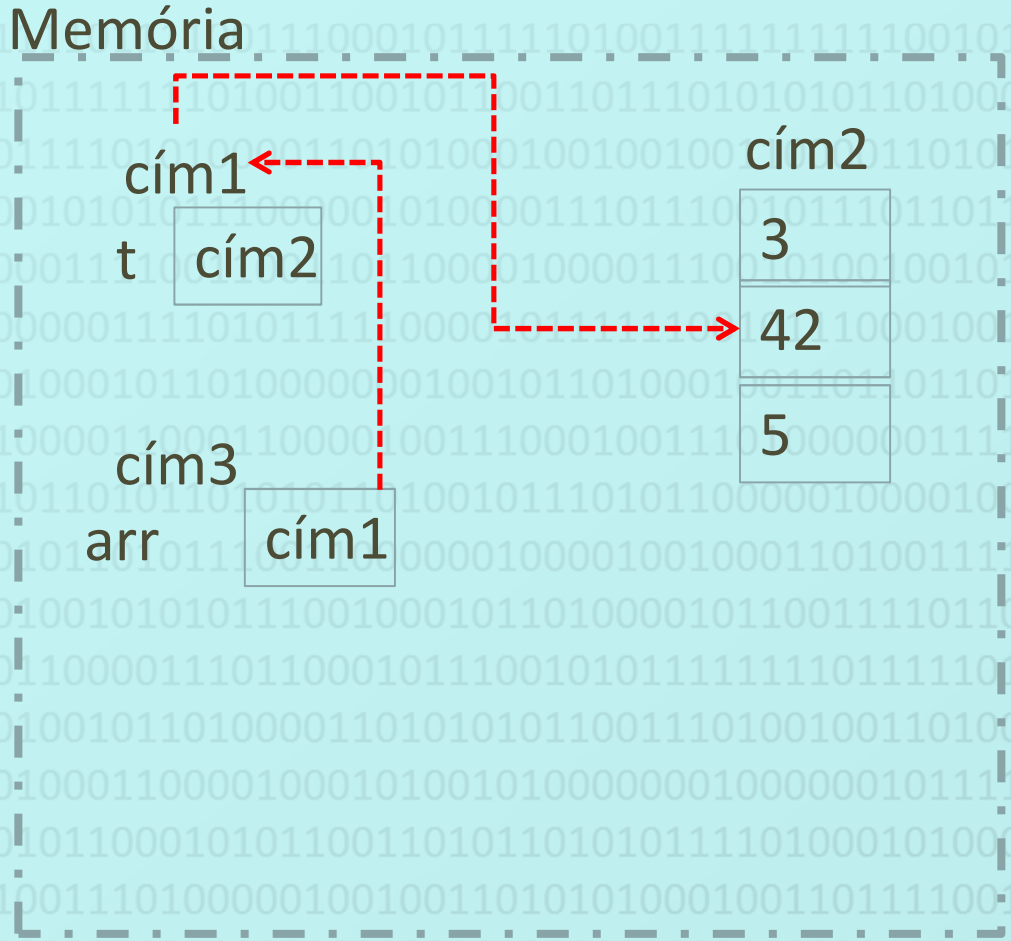
int[] t= {3, 4, 5};

Ref.típusok cím szerinti paraméterátadása



int[] t= {3, 4, 5};
tombkez1(t);

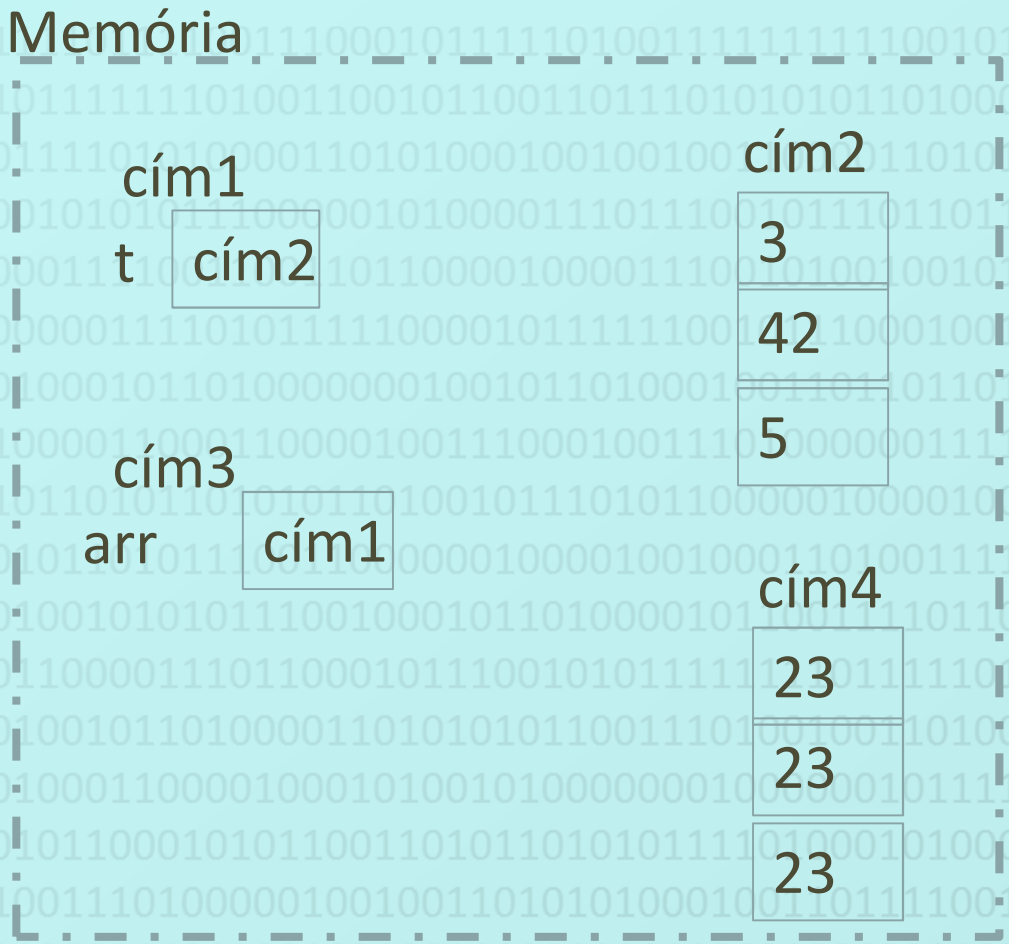
Ref.típusok cím szerinti paraméterátadása



int[] t= {3, 4, 5};
tombkez1(t);
arr[1]=42;

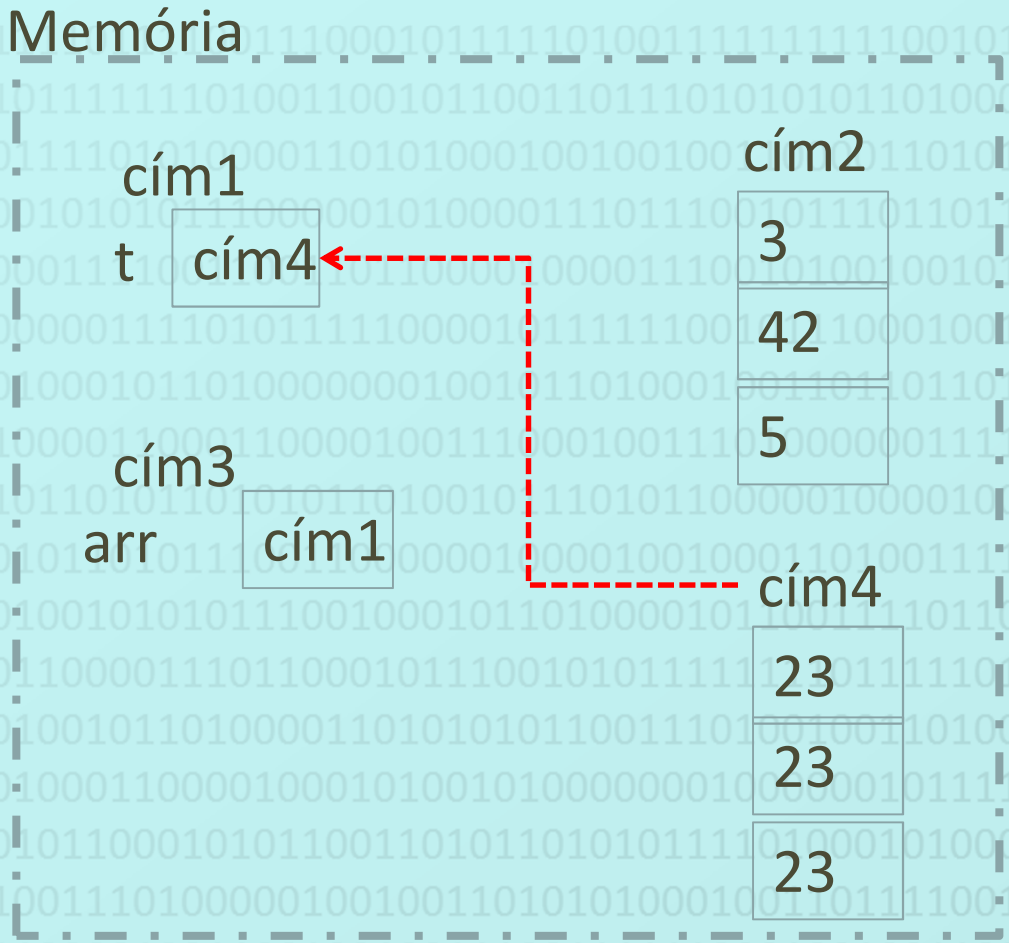
Teljesen mindegy, hogy hány referencián keresztül van az értékadás, mindig a megfelelő értékre történik a hivatkozás!

Ref.típusok érték szerinti paraméterátadása




```
int[] t= {3, 4, 5};
tombkez1(t);
        arr[1]=42;
tombkez2(t);
        arr=new int[3]
        { 23, 23, 23};
```

Ref.típusok érték szerinti paraméterátadása

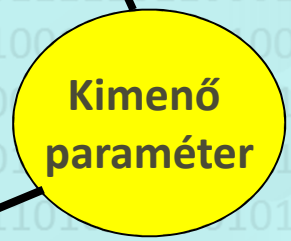



```
int[] t= {3, 4, 5};
tombkez1(t);
        arr[1]=42;
tombkez2(t);
        arr=new int[3]
        { 23, 23, 23};
```

Példa out-ra

```
static void bekér(out int szám)
{
        System.Console.Write("Kérek egy számot: ");
        szám=int.Parse(System.Console.ReadLine());
}

static void Main(string[] args)
{
        int x;
        bekér(out x);
        System.Console.WriteLine("A megadott szám: "+x);
        System.Console.ReadLine();
}
```



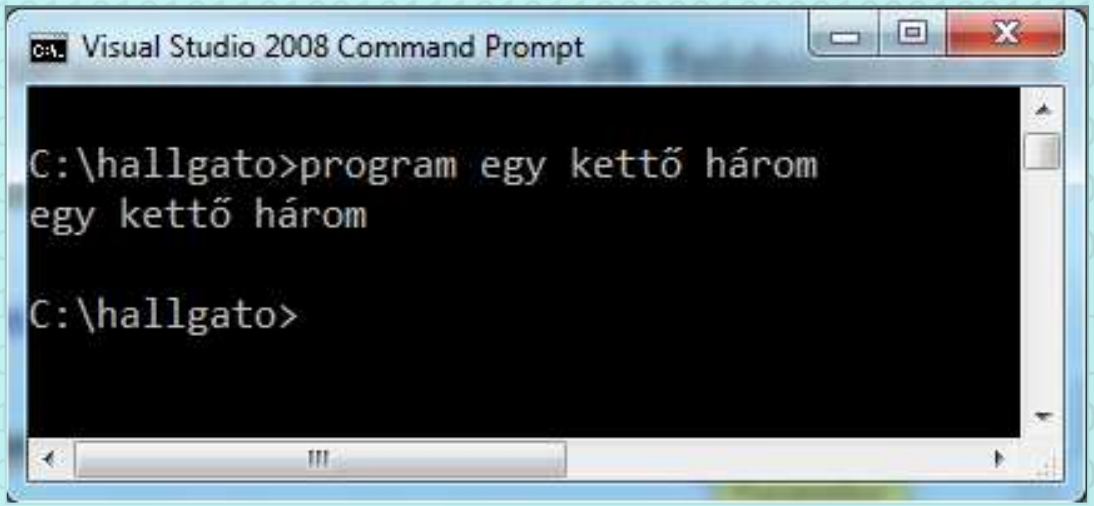
Példa a parancssori paraméterek feldolgozására

```
static void Main(string[] args)
{
        for (int i = 0; i < args.Length; i++)
                System.Console.Write(args[i] + " ");
        System.Console.ReadLine();
}
```

Paraméter

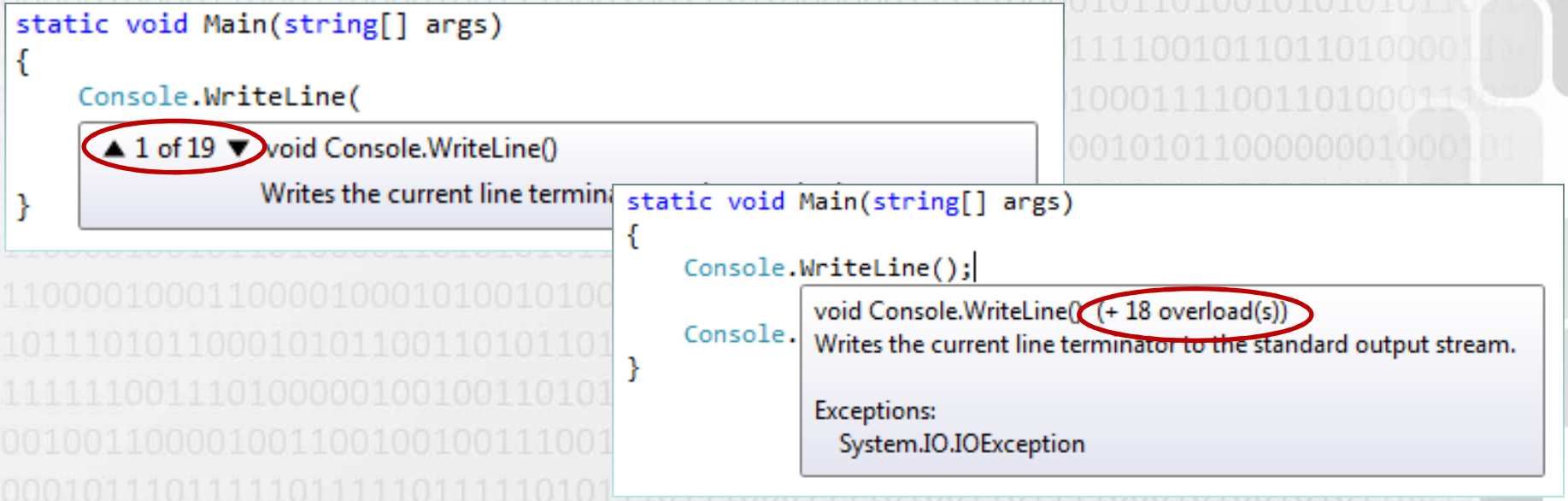
Stringek száma

Paraméter stringek



Overloading

- **Metódus "felüldefiniálás" – több azonos nevű metódus**
- **A metódusok szignatúrája egyedi az osztályon belül.**
  - Szignatúra = metódusnév + paraméterek száma, típusa és sorrendje
- **Azaz: írhatunk több azonos *nevű* metódust, ám ezek *paraméterlistája* különböző kell legyen!**



Objektumorientált Programozás VI.

Metódusok
Paraméterek átadása
Programozási tételek
Feladatok

Programozási tételek

- **Egy adott feladatosztályba tartozó összes feladatra megoldást adó algoritmus.**
  - pl.: a keresés tétel bármilyen elemsorozatban bármilyen feltételnek megfelelő elemek keresésére használható

Sorozatszámítás

- **Bemenet:**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
- **Kimenet:**
  R: Művelet eredménye

Eljárás Sorozatszámítás(A, N, R)

R := R_0

Ciklus i := 1-től N-ig

R := R művelet A[i]

Ciklus vége

Eljárás vége

- **Figyelem! Indexelés 1-től N-ig!**

Sorozatszámítás: példa

- **Számítsuk ki egy int tömb elemeinek összegét. (Indexelés 0-tól (N-1)-ig!)**

```
static int szumma(int[] tömb)
{
    int összeg = 0;
    for (int i = 0; i < tömb.Length; i++)
        összeg += tömb[i];
    return összeg;
}
```

Eldöntés

- **Bemenet:**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
  T: Tulajdonság függvény
- **Kimenet:**
  VAN: Logikai változó

```
Eljárás Eldöntés(A, N, T, VAN)
    i := 1
    Ciklus amíg (i ≤ N) és ¬(A[i] teljesíti T-t)
        i := i + 1
    Ciklus vége
    VAN := (i ≤ N)
Eljárás vége
```

Eldöntés: példa

- **Döntsük el, hogy egy int tömb elemei között található-e öttel osztható szám!**

```
static bool öttel_osztható(int[] tömb)
{
    int i = 0;
    while (i < tömb.Length && tömb[i] % 5 != 0)
        i++;
    return i < tömb.Length;
}
```

Kiválasztás

- **Bemenet**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
  T: Tulajdonság függvény
- **Kimenet:**
  SORSZ: Első T tulajdonságú elem indexe

```
Eljárás Kiválasztás(A, N, T, SORSZ)
    i := 1
    Ciklus amíg ¬ (A[i] teljesíti T-t)
        i := i + 1
    Ciklus vége
    SORSZ := i
Eljárás vége
```

Kiválasztás: példa

- **Adjuk meg a megadott e-mail címben a @ jel pozícióját (IndexOf() használata nélkül)!**

```
static int holakukac(string email)
{
    int poz = 0;
    while (email[poz] != '@')
        poz++;
    return poz;
}
```

Lineáris keresés

- **Bemenet:**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
  T: Tulajdonság függvény
- **Kimenet:**
  VAN: Logikai változó
  SORSZ: Első T tulajdonságú elem indexe

```
Eljárás Keresés(A, N, T, VAN, SORSZ)
    i := 1
    Ciklus amíg (i ≤ N) és ¬(A[i ] teljesíti T-t)
        i := i + 1
    Ciklus vége
    VAN := (i ≤ N)
    Ha VAN akkor
        SORSZ := i
Eljárás vége
```

Keresés: példa

- **Keressük meg egy int tömbben a 42 számot, ha nem szerepel benne, adjunk vissza -1-et!**

```
static int negyvenkettő(int[] tömb)
{
    int i = 0;
    while (i < tömb.Length && tömb[i] != 42)
        i++;
    if (i == tömb.Length)
        return -1;
    else return i;
}
```

Megszámlálás

- **Bemenet:**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
  T: Tulajdonság függvény
- **Kimenet:**
  DB: T tulajdonságú elemek száma

```
Eljárás Megszámlálás(A, N, T, DB)
    DB := 0
    Ciklus i := 1-től N-ig
        Ha (A[i] teljesíti T-t) akkor
            DB := DB + 1
        Elágazás vége
    Ciklus vége
Eljárás vége
```

Megszámlálás: példa

- **Állapítsuk meg egy adott mondatban a nagybetűk számát!**

```
static int nagybetű(string mondat)
{
    int db = 0;
    for (int i = 0; i < mondat.Length; i++)
    {
        if (Char.IsUpper(mondat[i])) db++;
    }
    return db;
}
```

Maximumkiválasztás

- **Bemenet:**
  A: Feldolgozandó tömb
  N: Tömb elemeinek száma
- **Kimenet:**
  MAX: Maximális elem indexe

```
Eljárás Maximumkiválasztás(A, N, MAX)
    MAX := 1
    Ciklus i := 2-től N-ig
        Ha A[i] > A[MAX] akkor
            MAX := i
        Elágazás vége
    Ciklus vége
Eljárás vége
```

Maximumkiválasztás: példa

- **Tudjuk egy cég havi kiadásait. Az adott félév hanyadik hónapjában költöttek a legtöbbet?**

```
static int legköltségesebb(double[] költségek)
{
    int max = 0;
    for (int i = 1; i < költségek.Length; i++)
    {
        if (költségek[i] > költségek[max])  max = i;
    }
    return max;
}
```

Objektumorientált Programozás VI.

Metódusok
Paraméterek átadása
Programozási tételek
Feladatok

Gyakorló feladatok

Adottak egy tankör zárthelyi pontszámai, 100 pont a maximum. (Tároljuk el ezeket egy tömbben.) Oldja meg az alábbi feladatokat!

Határozza meg a csoport átlagpontszámát!

Állapítsa meg, volt-e bukás! (Teljesített-e valaki 51 pont alatt?)

Hány pontos volt a leggyengébb teljesítmény?

Hány hallgató kapott ötöst a tankörben? (90% vagy afölötti pontszám)

Gyakorló feladatok

Egy meteorológiai állomáson mérik a napi közép-hőmérséklet-értékeket egy hónapon keresztül. (Tároljuk el ezeket egy tömbben.) Oldja meg az alábbi feladatokat külön-külön metódusok segítségével!

Határozza meg a havi átlaghőmérsékletet!

Állapítsa meg, fagyott-e ebben a hónapban!

Adjon meg egy olyan napot, amikor 5 °C-nál hidegebb volt!

Hány olyan nap volt a hónapban, amikor a hőmérséklet a havi átlag alatt volt?

Hány °C volt a hónap leghidegebb napján?

Objektumorientált Programozás VI.

- ✓Metódusok
- ✓Paraméterek átadása
- ✓Programozási tételek
- ✓Feladatok

Irodalom, feladatok

- **Kotsis-Légrádi-Nagy-Szénási: Többnyelvű programozástechnika, PANEM, Budapest, 2007**
- **Zsakó-Szlávi: Mikrológia 19.**
- **Faraz Rasheed: C# School, Synchron Data, 2006 http://www.programmersheaven.com/2/CSharpBook**
- **Reiter István: C# jegyzet, DevPortal, 2010, http://devportal.hu/content/CSharpjegyzet.aspx**