



ELTE | IK

# PROGRAMOZÁS

## Függvények

Horváth Győző, Pluhár Zsuzsa



# Ismétlés



# Programozási minták

1. Összegzés
2. Megszámolás
3. Maximumkiválasztás
  - a. Minimumkiválasztás
4. Feltételes maximumkeresés
5. Keresés
6. Eldöntés
  - a. Mind eldöntés
7. Kiválasztás
8. Másolás
9. Kiválogatás

Most Common DUPLO Parts



Brick Architect 

# Programozási minták

## Összegzés

i	f(i)
e	→ f(e)
e+1	→ f(e+1)
e+2	→ f(e+2)
...	→ ...
u-2	→ f(u-2)
u-1	→ f(u-1)
u	→ f(u)
=	
s	

## Megszámolás

i	T(i)	érték
e	→ IGAZ	1
e+1	→ HAMIS	0
e+2	→ HAMIS	0
...	→ ...	...
u-2	→ IGAZ	1
u-1	→ IGAZ	1
u	→ HAMIS	0
=		
db		

## Maximum kiválasztás

i	f(i)
e	→ f(e)
e+1	→ f(e+1)
e+2	→ f(e+2)
...	→ ...
u-2	→ f(u-2)
u-1	→ f(u-1)
u	→ f(u)
maxind, maxért	

## Feltételes maximumkeresés

i	T(i)	f(i)
e	→ HAMIS	f(e)
e+1	→ IGAZ	f(e+1)
e+2	→ IGAZ	f(e+2)
...	→ ...	...
u-2	→ HAMIS	f(u-2)
u-1	→ IGAZ	f(u-1)
u	→ HAMIS	f(u)
van, maxind, maxért		

# Programozási minták

## Keresés

i	T(i)
e	→ HAMIS
e+1	→ HAMIS
e+2	→ IGAZ
...	→ ...
u-2	→ IGAZ
u-1	→ IGAZ
u	→ HAMIS
van, ind	

## Eldöntés

i	T(i)
e	→ HAMIS
e+1	→ HAMIS
e+2	→ IGAZ
...	→ ...
u-2	→ IGAZ
u-1	→ IGAZ
u	→ HAMIS
van	

## Kiválasztás

i	T(i)
e	→ HAMIS
e+1	→ HAMIS
e+2	→ IGAZ
...	→ ...
u-2	→ IGAZ
u-1	→ IGAZ
u	→ HAMIS
ind	

# Programozási minták

## Másolás

i			f(i)
e	→	1	f(e)
e+1	→	2	f(e+1)
e+2	→	3	f(e+2)
...	→	...	...
u-2	→	u-e-1	f(u-2)
u-1	→	u-e	f(u-1)
u	→	u-e+1	f(u)
y			

## Kiválogatás

i	T(i)	f(i)		y
e	→ HAMIS	f(e)	1	f(e+1)
e+1	→ IGAZ	f(e+1)	2	f(e+2)
e+2	→ IGAZ	f(e+2)	db= 3	f(u-1)
...	→	...		
u-2	→ HAMIS	f(u-2)		
u-1	→ IGAZ	f(u-1)		
u	→ HAMIS	f(u)		
db, y				

# Függvények



# Négyzet

Példa:

$x=3,3 \rightarrow y=10,89$

## Feladat:

Adjuk meg egy szám négyzetét!

## Specifikáció:

Be:  $x \in \mathbb{R}$

Ki:  $y \in \mathbb{R}$

Ef: -

Uf:  $y = x * x$

Mi van akkor, ha ezt a „bonyolult”  
részfeladatot el szeretném különíteni  
általánosan megfogalmazva?

## Algoritmus:

$y := x * x$



# Négyzet

Példa:

$x=3,3 \rightarrow y=10,89$

## Feladat:

Adjuk meg egy szám négyzetét!

## Specifikáció:

Be:  $x \in \mathbb{R}$

Ki:  $y \in \mathbb{R}$

Ef: -

Uf:  $y = \text{négyzet}(x)$

Egy függvény mögé rejtettük a négyzetre emelést.

Tekintsük a **négyzet** függvény kiszámítását **önálló feladatnak!**

## Algoritmus:

$y := \text{négyzet}(x)$

# Négyzet

Matematika:

$$x \mapsto x^2$$

Másként:

$$f: \mathbb{R} \rightarrow \mathbb{R}, f(x) = x^2$$

Esetünkben:

$$\text{négyzet}: \mathbb{R} \rightarrow \mathbb{R}, \text{négyzet}(n) = n^2$$

## Feladat:

Add meg a **négyzet függvény** (részfeladat) működését!

## Specifikáció:

Be:  $n \in \mathbb{R}$

Ki:  $\text{négyzet}(n) \in \mathbb{R}$

Ef: -

Uf:  $\text{négyzet}(n) = n * n$

A bemenetben a függvény **paraméterei** ( $\in$  értelmezési tartomány), a kimenetben a függvény paraméteres **értéke** szerepel ( $\in$  értékkészlet), az utófeltételben az **összefüggés**.

## Algoritmus:

$\text{négyzet}(n: \text{Valós}): \text{Valós}$

$\text{négyzet} := n * n$

Visszatérési érték meghatározása a függvénynévhez való értékadással

# Négyzet

Példa:

$x=3,3 \rightarrow y=10,89$

## Feladat:

Adjuk meg egy szám négyzetét!

## Specifikáció:

Be:  $x \in \mathbb{R}$

Ki:  $y \in \mathbb{R}$

Fv:  $\text{négyzet}: \mathbb{R} \rightarrow \mathbb{R}, \text{négyzet}(n) = n * n$

Ef: -

Uf:  $y = \text{négyzet}(x)$

## Algoritmus:

$y := \text{négyzet}(x)$

Matematika:

$$x \mapsto x^2$$

Másként:

$$f: \mathbb{R} \rightarrow \mathbb{R}, f(x) = x^2$$

Esetünkben:

$$\text{négyzet}: \mathbb{R} \rightarrow \mathbb{R}, \text{négyzet}(n) = n^2$$

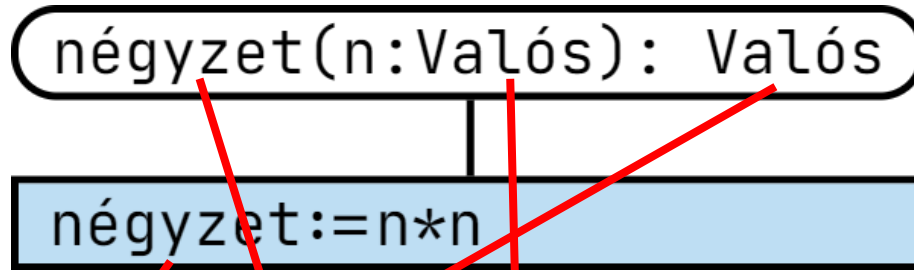
Formális paraméter

Aktuális paraméter

$\text{négyzet}(n: \text{Valós}): \text{Valós}$

$\text{négyzet} := n * n$

# Négyzet kód



```
// négyzet:R->R, négyzet(n)=n*n  
static double negyzet(double n) {  
    return n * n;  
}
```

# Négyzet kód

```
static void Main(string[] args) {  
    Console.WriteLine(3.3 * 3.3);  
  
    // Általánosítsuk a feladatot!  
    Console.WriteLine(negyzet(3.3));  
  
    // Függvényhívás helyettesíthető a visszatérési értékkel:  
    Console.WriteLine(10.89);  
  
    // Aktuális paraméter: konstans  
    double a = negyzet(3.3);  
  
    // Aktuális paraméter: változó értéke  
    double b = negyzet(a);  
}  
  
// négyzet:R->R, négyzet(n)=n*n  
static double negyzet(double n) {  
    return n * n;  
}
```

Aktuális paraméter

Formális paraméter

négyzet(n:Valós): Valós

négyzet:=n\*n

# Maximum

Példa:

$a=10, b=8 \rightarrow m=10$

## Feladat:

Adjuk meg két szám közül a nagyobbbat!

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $m \in \mathbb{Z}$

Ef: -

Uf:  $m = \max(a, b)$

Mi derül ki a használatból?

1. Függvény neve = max
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész, Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

# Maximum

Példa:

$a=10, b=8 \rightarrow m=10$

## Feladat:

Adjuk meg két szám közül a nagyobbbat!

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $m \in \mathbb{Z}$

Fv:  $\text{max} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}, \text{max}(a, b) = \{a, \text{ ha } a \geq b; \\ b \text{ egyébként}\}$

Ef: -

Uf:  $m = \text{max}(a, b)$

Mi derül ki a használatból?

1. Függvény neve = max
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész, Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

# Maximum

Példa:

$a=10, b=8 \rightarrow m=10$

## Feladat:

Adjuk meg két szám közül a nagyobbbat!

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $m \in \mathbb{Z}$

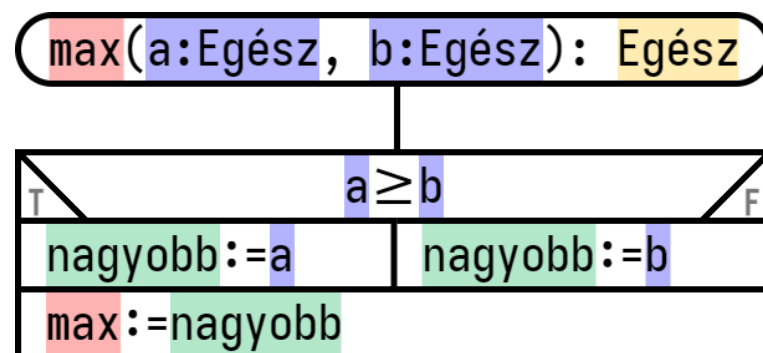
Fv:  $\text{max} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z},$   
 $\text{max}(a, b) = \{a, \text{ ha } a \geq b;$   
 $\text{ b egyébként} \}$

Ef: -

Uf:  $m = \text{max}(a, b)$

## Algoritmus:

$m := \text{max}(a, b)$

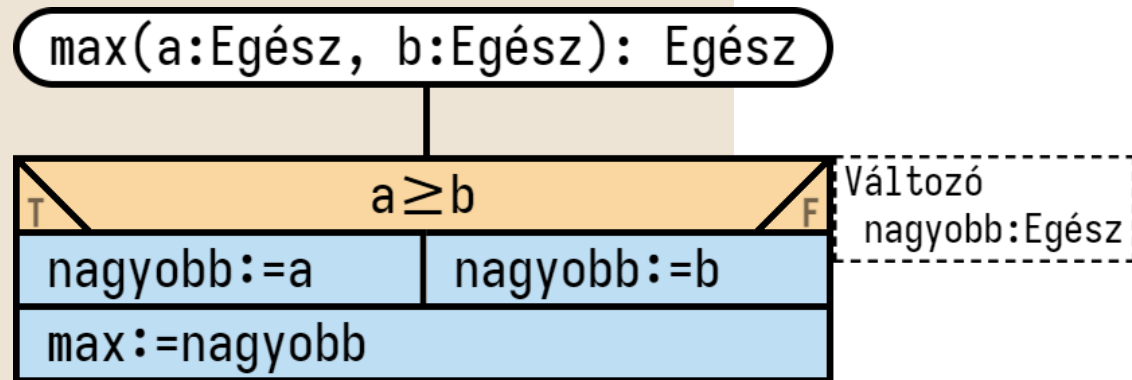


Változó  
 $\text{nagyobb} : \text{Egész}$



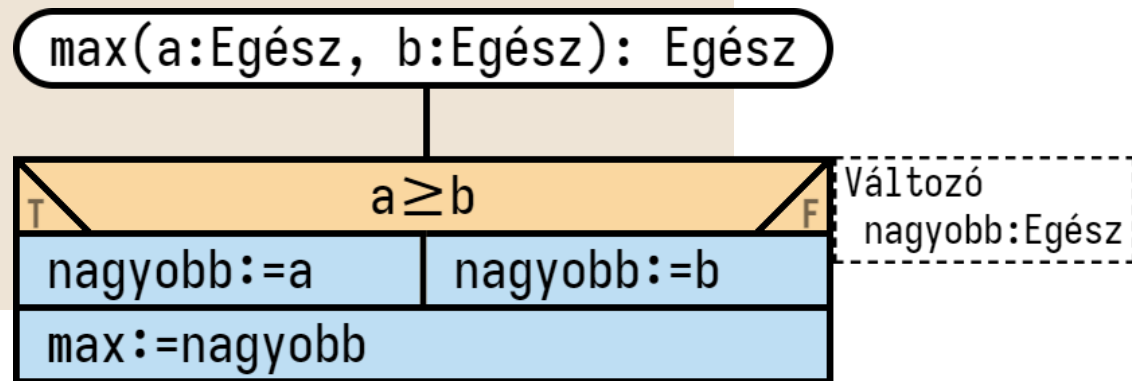
# Maximum kód

```
static void Main(string[] args) {  
    // A használatból kiderül a függvény szignatúrája  
    // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
    Console.WriteLine(max(3, 7));  
}  
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
static int max(int a, int b) {  
    int nagyobb;  
    if (a >= b) {  
        nagyobb = a;  
    }  
    else {  
        nagyobb = b;  
    }  
    return nagyobb;  
}
```



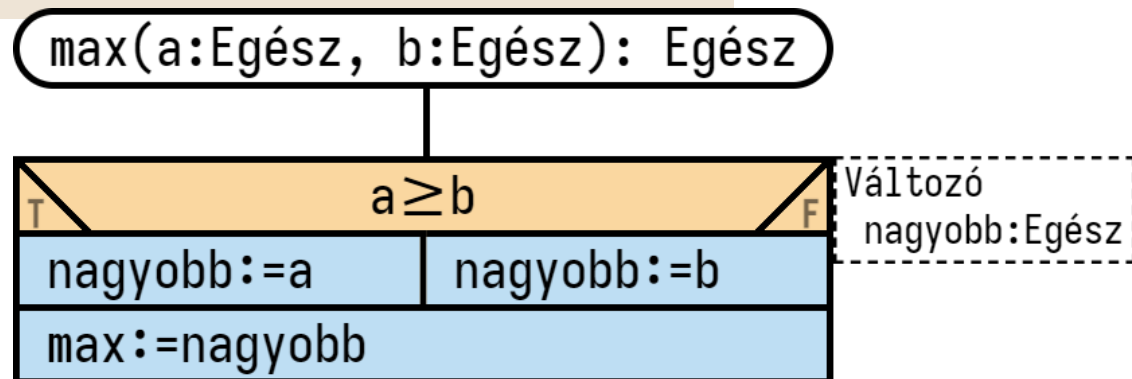
# Maximum kód

```
static void Main(string[] args) {  
    // A használatból kiderül a függvény szignatúrája  
    // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
    Console.WriteLine(max(3, 7));  
}  
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
static int max(int a, int b) {  
    if (a >= b) {  
        return a;  
    }  
    else {  
        return b;  
    }  
}
```



# Maximum kód

```
static void Main(string[] args) {  
    // A használatból kiderül a függvény szignatúrája  
    // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
    Console.WriteLine(max(3, 7));  
}  
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}  
static int max(int a, int b) {  
    return a >= b ? a : b;  
}
```



# Növelés

## Feladat:

Példa:  
 $n=10, d=8 \rightarrow n'=18$

```
// Példa  
int a = 10;  
// így szeretném növelni:  
novel(a, 8);  
// és nem így:  
a = novel(a, 8);
```

Növeljük meg egy változót egy előre megadott  $d$  értékkel!

## Specifikáció:

Be:  $n \in \mathbb{Z}, d \in \mathbb{Z}$

Ki:  $n' \in \mathbb{Z}$

Ef: -

Uf:  $n' = \text{novel}(n, d)$

Mi derül ki a használatból?

1. Függvény neve = novel
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész, Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

# Növelés

## Feladat:

Példa:  
 $n=10, d=8 \rightarrow n'=18$

```
// Példa  
int a = 10;  
// így szeretném növelni:  
novel(a, 8);  
// és nem így:  
a = novel(a, 8);
```

Növeljük meg egy változót egy előre megadott  $d$  értékkel!

## Specifikáció:

Be:  $n \in \mathbb{Z}, d \in \mathbb{Z}$

Ki:  $n' \in \mathbb{Z}$

Fv:  $\text{novel} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z},$   
 $\text{novel}(\text{szám}, \text{mivel}) = \text{szám} + \text{mivel}$

Ef: -

Uf:  $n' = \text{novel}(n, d)$

Mi derül ki a használatból?

1. Függvény neve = novel
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész, Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

# Növelés

## Feladat:

Példa:  
 $n=10, d=8 \rightarrow n'=18$

```
// Példa  
int a = 10;  
// így szeretném növelni:  
novel(a, 8);  
// és nem így:  
a = novel(a, 8);
```

Növeljük meg egy változót egy előre megadott  $d$  értékkel!

## Specifikáció:

Be:  $n \in \mathbb{Z}, d \in \mathbb{Z}$

Ki:  $n' \in \mathbb{Z}$

Fv:  $\text{novel} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z},$   
 $\text{novel}(\text{szám}, \text{mivel}) = \text{szám} + \text{mivel}$

Ef: -

Uf:  $n' = \text{novel}(n, d)$

## Algoritmus:

$\text{novel}(a, 8)$

$\text{novel}(\text{Vált}, \text{szám:Egész}, \text{mivel:Egész})$

$\text{szám} := \text{szám} + \text{mivel}$

Változó paraméterként határozzuk meg!

# Növelés kód

növel(Vált szám:Egész,  
mivel:Egész)

szám:=szám+mivel

```
static void Main(string[] args) {  
    int a = 10;  
    Console.WriteLine("Előtte: {0}", a);  
    novel(a, 8);  
    Console.WriteLine("Utána: {0}", a);  
}
```

```
static void novel(int szám, int mivel) {  
    szám = szám + mivel;  
}
```

Előtte: 10  
Utána: 10

Rossz!

**Érték szerinti paraméterátadás** van: a változó értékéről (10) másolat készül, ez kerül a formális paraméternek átadásra (szám), a másolat módosul, és szűnik meg a függvény végén.



# Növelés kód

növel(Vált szám:Egész,  
mivel:Egész)

szám:=szám+mivel

```
static void Main(string[] args) {  
    int a = 10;  
    Console.WriteLine("Előtte: {0}", a);  
    novel(ref a, 8);  
    Console.WriteLine("Utána: {0}", a);  
}
```

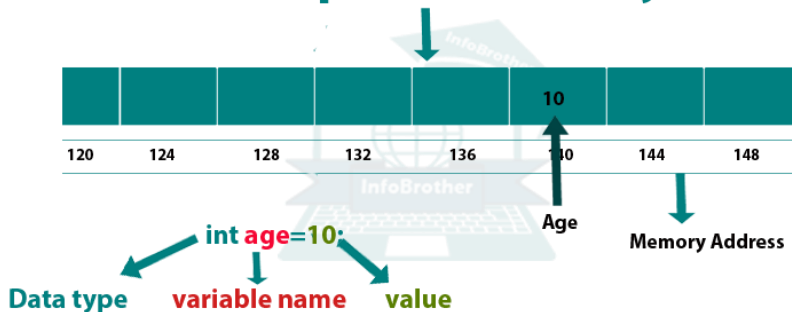
```
static void novel(ref int szam, int mivel) {  
    szam = szam + mivel;  
}
```

Előtte: 10  
Utána: 18

Helyes!

**Referencia szerinti paraméterátadás:**  
a változó referenciája (memóriacíme)  
kerül átadásra, így használatkor a  
függvény a külső változó értékével  
dolgozik.

## Computer Memory





# Csere

```
// Példa
int a = 10, b = 20;
// Csere:
Console.WriteLine("Előtte: {0}, {1}", a, b);
csere(a, b);
Console.WriteLine("Utána: {0}, {1}", a, b);
```

## Feladat:

Cseréljük fel két változó értékét!

Példa:  
 $a=10, b=8 \rightarrow a'=8, b'=10$

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $a' \in \mathbb{Z}, b' \in \mathbb{Z}$

Ef: -

Uf:  $a' = b$  és  $b' = a$

# Csere

```
// Példa  
int a = 10, b = 20;  
// Csere:  
Console.WriteLine("Előtte: {0}, {1}", a, b);  
csere(a, b);  
Console.WriteLine("Utána: {0}, {1}", a, b);
```

## Feladat:

Cseréljük fel két változó értékét!

Példa:

$a=10, b=8 \rightarrow a'=8, b'=10$

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $a' \in \mathbb{Z}, b' \in \mathbb{Z}$

Fv:  $csere: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \times \mathbb{Z}$ ,  
 $csere(x, y) = (y, x)$

Ef: -

Uf:  $(a', b') = csere(a, b)$

Mi derül ki a használatból?

1. Függvény neve = csere
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész, Egész
4. Visszatérési érték(ek) száma = 2
5. Visszatérési érték(ek) típusa = Egész, Egész

# Csere

```
// Példa
int a = 10, b = 20;
// Csere:
Console.WriteLine("Előtte: {0}, {1}", a, b);
csere(a, b);
Console.WriteLine("Utána: {0}, {1}", a, b);
```

## Feladat:

Cseréljük fel két változó értékét!

Példa:

$a=10, b=8 \rightarrow a'=8, b'=10$

## Specifikáció:

Be:  $a \in \mathbb{Z}, b \in \mathbb{Z}$

Ki:  $a' \in \mathbb{Z}, b' \in \mathbb{Z}$

Fv:  $csere: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \times \mathbb{Z}$ ,  
 $csere(x, y) = (y, x)$

Ef: -

Uf:  $(a', b') = csere(a, b)$

## Algoritmus:

$csere(a, b)$

$csere(\text{Vált } x: \text{Egész},$   
 $\text{Vált } y: \text{Egész})$

$z := x$

$x := y$

$y := z$

Változó  
 $z: \text{Egész}$

# Csere kód

```
static void Main(string[] args) {  
    int a = 10, b = 8;  
    Console.WriteLine("Előtte: {0}, {1}", a, b);  
    csere(ref a, ref b);  
    Console.WriteLine("Utána: {0}, {1}", a, b);  
}
```

```
static void csere(ref int x, ref int y) {  
    int z = x;  
    x = y;  
    y = z;  
}
```

Előtte: 10, 8  
Utána: 8, 10

csere(Vált x:Egész,  
Vált y:Egész)

z:=x

x:=y

y:=z

Változó  
z:Egész

# Kódszervezés függvényekkel előzőnél gyorsabb vonat

Kiírás

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
    // Beolvasás (specifikáció be)  
    Console.Write("n = ");  
    int.TryParse(Console.ReadLine(), out n);  
    mido = new int[n];  
    for (int i = 0; i < n; i++) {  
        Console.Write("{0}. menetido = ", i);  
        int.TryParse(Console.ReadLine(), out mido[i]);  
    }  
    // Feldolgozás (algoritmus, stuki)  
    melyik = 2;  
    while (melyik < n && !(mido[melyik] < mido[melyik - 1])) {  
        melyik = melyik + 1;  
    }  
    van = melyik < n;  
    // Kiírás (specifikáció ki)  
    if (van) {  
        Console.WriteLine("Van, a(z) {0}. vonat gyorsabb az előzőnél.", melyik+1);  
    }  
    else {  
        Console.WriteLine("Nincs gyorsabb vonat az előzőnél.");  
    }  
}
```

# Kódszervezés függvényekkel előzőnél gyorsabb vonat

Elvárások (pszeudo-kód)

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
    // Beolvasás (specifikáció be)  
    beolvas(n, mido);  
  
    // Feldolgozás (algoritmus, stuki)  
    keres_vonat(n, mido, // bemeneti adatok  
                van, melyik); // kimeneti adatok  
    // vagy:  
    (van, melyik) = keres_vonat(n, mido); // kimeneti adatok ← bemeneti adatok  
    // Kiírás (specifikáció ki)  
    kiir(van, melyik);  
}
```

# Kódszervezés függvényekkel előzőnél gyorsabb vonat

## Beolvasás függvény

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
  
    beolvas(out n, out mido);  
  
    // ...  
}  
static void beolvas(out int n, out int[] mido) {  
    // Beolvasás (specifikáció be)  
    Console.Error.Write("n = ");  
    int.TryParse(Console.ReadLine(), out n);  
    mido = new int[n];  
    for (int i = 0; i < n; i++) {  
        Console.Error.Write("{0}. menetido = ", i+1);  
        int.TryParse(Console.ReadLine(), out mido[i]);  
    }  
}
```

# Kódszervezés függvényekkel előzőnél gyorsabb vonat

## Feldolgozás függvény

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
  
    beolvas(out n, out mido);  
    keres_vonat1(n, mido,  
                out van, out melyik);  
    // ...  
}  
static void keres_vonat1(int n, int[] mido, out bool van, out int melyik) {  
    // Feldolgozás (algoritmus, stuki)  
    melyik = 1;  
    while (melyik < n && !(mido[melyik] < mido[melyik-1])) {  
        melyik = melyik + 1;  
    }  
    van = melyik < n;  
}
```



# Kódszervezés függvén előzőnél gyorsabb vonat

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
  
    beolvas(out n, out mido);  
    (van, melyik) = keres_vonat2(n, mido);  
    // ...  
}  
  
static (bool van, int melyik) keres_vonat2(int n, int[] mido) {  
    // Feldolgozás (algoritmus, stuki)  
    int melyik;  
    bool van;  
  
    melyik = 1;  
    while (melyik < n && !(mido[melyik] < mido[melyik - 1])) {  
        melyik = melyik + 1;  
    }  
    van = melyik < n;  
  
    return (van, melyik);  
}
```

## Specifikáció:

Be:  $n \in \mathbb{N}$ ,  $\text{midő} \in \mathbb{N}[1..n]$

Ki:  $\text{van} \in \mathbb{L}$ ,  $\text{melyik} \in \mathbb{N}$

Ef: -

Uf:  $(\text{van}, \text{melyik}) = \text{KERES}(i=2..n, \text{midő}[i] < \text{midő}[i-1])$

A megoldás tekinthető olyan függvénynek, ami  
 $(n, \text{mido}) \rightarrow (\text{van}, \text{melyik})$

Másképpen:

$\text{keres\_vonat}: \mathbb{N} \times \mathbb{N}[] \rightarrow \mathbb{L} \times \mathbb{N}$

$(\text{van}, \text{melyik}) := \text{keres\_vonat}(n, \text{mido})$

# Kódszervezés függvényekkel előzőnél gyorsabb vonat

## Kiírás függvény

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be,ki)  
    int n;  
    int[] mido;  
    bool van;  
    int melyik;  
  
    beolvas(out n, out mido);  
    (van, melyik) = keres_vonat2(n, mido);  
    kiir(van, melyik);  
}  
static void kiir(bool van, int melyik) {  
    // Kiírás (specifikáció ki)  
    if (van) {  
        Console.WriteLine(melyik+1);  
    }  
    else {  
        Console.WriteLine(-1);  
    }  
}
```

# Kódszervezés függvényekkel

Teljes megoldás

```
static void Main(string[] args) {
    // Deklarálás (változók, specifikáció be,ki)
    int n;
    int[] mido;
    bool van;
    int melyik;

    beolvas(out n, out mido);
    keres_vonat1(n, mido, out van, out melyik);
    kiir(van, melyik);
}

static void beolvas(out int n, out int[] mido) {
    // Beolvasás (specifikáció be)
    Console.Error.Write("n = ");
    int.TryParse(Console.ReadLine(), out n);
    mido = new int[n];
    for (int i = 0; i < n; i++) {
        Console.Error.Write("{0}. menetido = ", i+1);
        int.TryParse(Console.ReadLine(), out mido[i]);
    }
}

static void keres_vonat1(int n, int[] mido, out bool van, out int melyik) {
    // Feldolgozás (algoritmus, stuki)
    melyik = 1;
    while (melyik < n && !(mido[melyik] < mido[melyik - 1])) {
        melyik = melyik + 1;
    }
    van = melyik < n;
}

static void kiir(bool van, int melyik) {
    // Kiírás (specifikáció ki)
    if (van) {
        Console.WriteLine(melyik+1);
    }
    else {
        Console.WriteLine(-1);
    }
}
```

# Kódszervezés függvényekkel

```
static void Main(string[] args) {  
    // Deklarálás (változók, specifikáció be, ki)  
    int[] mido;  
    bool van; int melyik;  
  
    beolvas(out mido);  
    (van, melyik) = keres_vonat!(mido);  
    // ...  
}  
static void beolvas(out int[] mido) {  
    // Beolvasás (specifikáció be)  
    int n;  
    Console.Error.Write("n = ");  
    int.TryParse(Console.ReadLine(), out n);  
    mido = new int[n];  
    for (int i = 0; i < n; i++) {  
        Console.Error.Write("{0}. menetido = ", i+1);  
        int.TryParse(Console.ReadLine(), out mido[i]);  
    }  
}  
static (bool van, int melyik) keres_vonat!(int[] mido) {  
    // Feldolgozás (algoritmus, stuki)  
    int melyik;  
    bool van;  
  
    int n = mido.Length;  
    melyik = 2;  
    while (melyik < n && !(mido[melyik] < mido[melyik - 1])) {  
        melyik = melyik + 1;  
    }  
    van = melyik < n;  
    return (van, melyik);  
}
```

Tömb tartalmazza a hosszát,  
n elhagyható, lekérdezhető

# Függvények

## fogalmak (c#)

---

**Blokk.** A { és a hozzá tartozó } közötti programszöveg.

**Hatáskör.** Egy X azonosító hatásköre az a programszöveg (nem feltétlenül összefüggő), ahol az azonosítóra hivatkozni lehet.

A hatáskört a blokkstruktúra határozza meg. Ha két blokknak van közös része, akkor az egyik teljes egészében tartalmazza a másikat.

Egy azonosító hatásköre a deklarációját követő karaktertől a blokkot lezáró } végzárójelig tart, kivéve azt a beágyazott blokkot, és ennek beágyazottjait, amelyben újra lett deklarálnva.

# Függvények

## fogalmak (c#)

---

**Hatáskör.** Egy adott helyen hivatkozott azonosító **lokális**, ha a hivatkozás helyét tartalmazó legszűkebb blokkban lett deklarálva. Egy azonosító **globális** (az adott blokkra nézve), ha **nem lokális**.

**Élettartam.** Minden B blokkban deklarált változó élettartalma a blokkba való belépéstől a blokk utolsó utasításának befejeződéséig tart.

## C# tudnivalók – összefoglalás:

- **Formális** skalár paraméter:

- o bemeneti → **nincs** speciális kulcsszó
- o kimeneti → **ref/out** a speciális prefix „kulcsszó”

Pontosabban:  
**nem tömb**

- **Aktuális** skalár paraméter:

ha a megfelelő formális paraméter

- o bemeneti → **akár** konstans, **akár** változó
- o kimeneti → **csak** változó, **ref/out** a speciális prefix-szel lehet.

Pontosabban:  
**nem tömb**

**ref**: a paramétert meg **lehet** változtatni

**out**: a paramétert meg **kell** változtatni

**Tömb** paraméter esetén mindig **hivatkozás szerinti** a paraméterátadás.

vek  
(c#)

## C# tudnivalók – összefoglalás:

- Skalár paraméterátadás:

- o **Értékszerinti** – a formális paraméterből „keletkezett” lokális változóba másolódik a híváskor az aktuális paraméter **értéke**, így ennek a törzsön belüli megváltozása nincs hatással az aktuális paraméterre. Pl.:

```
static int max(int x, int y)
```

Input-paraméterek.

- o **Hivatkozás szerinti** – a formális paraméterbe az aktuális paraméter **címe** (*rá való hivatkozás*) kerül, a lokális néven is elérhetővé válik. Pl.:

Input-paraméterek.

```
static void max(int x, int y, ref int max_xy)
```

In-/Output-paraméter.

```
static void max(int x, int y, out int max_xy)
```

Input-paraméterek.

Output paraméter.



# Összefoglalás



# Függvények

---

- Függvények szerepe
  - Részfeladatok csoportosítása (alprogram)
  - Általánosítás (paraméterekkel)