ELTE | IK

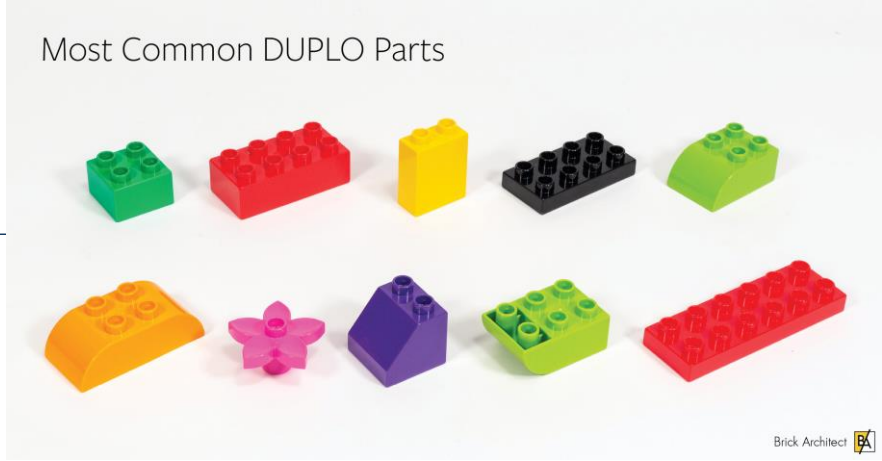
PROGRAMOZÁS

5. előadás

Horváth Győző, Horváth Gyula, Szlávi Péter

Ismétlés

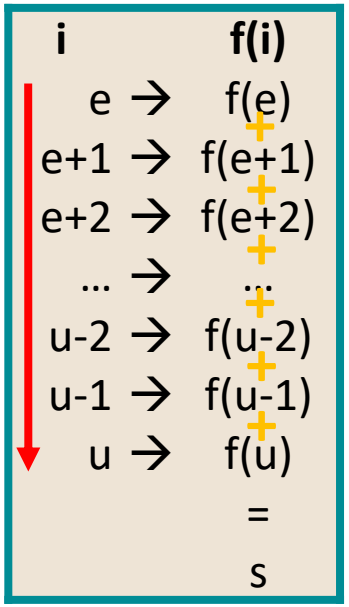
Programozási minták



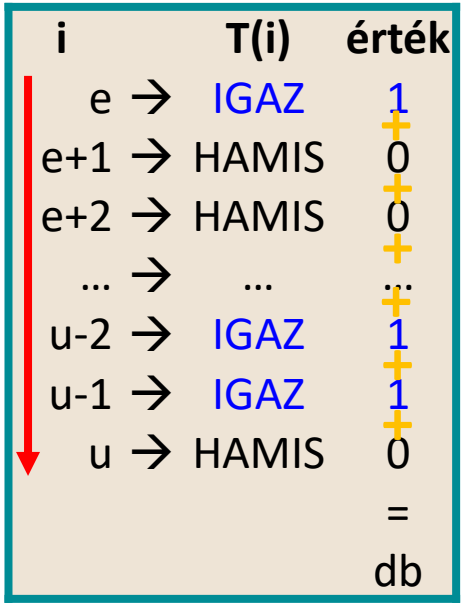
1. Összegzés
2. Megszámolás
3. Maximumkiválasztás
   a. Minimumkiválasztás
4. Feltételes maximumkeresés
5. Keresés
6. Eldöntés
   a. Mind eldöntés
7. Kiválasztás
8. Másolás
9. Kiválogatás

Programozási minták

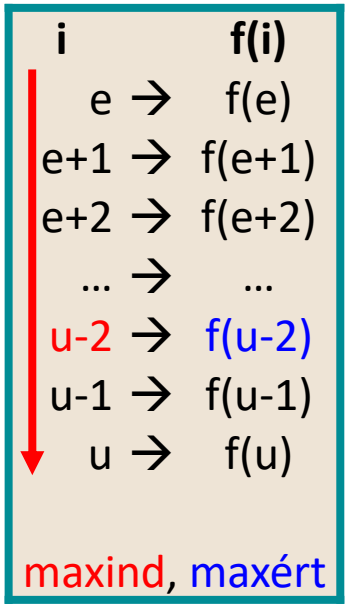
Összegzés



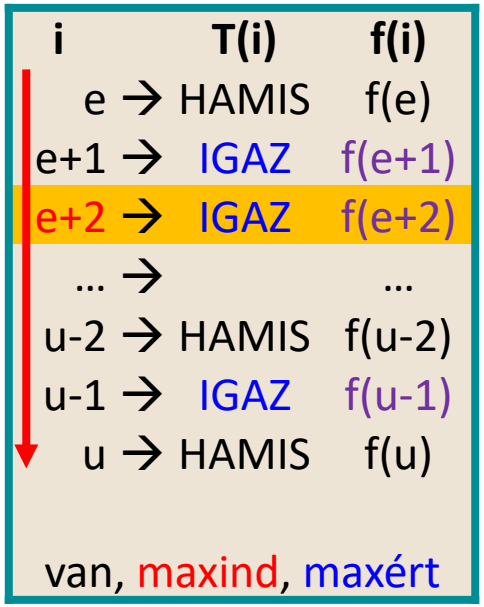
Megszámolás



Maximum kiválasztás

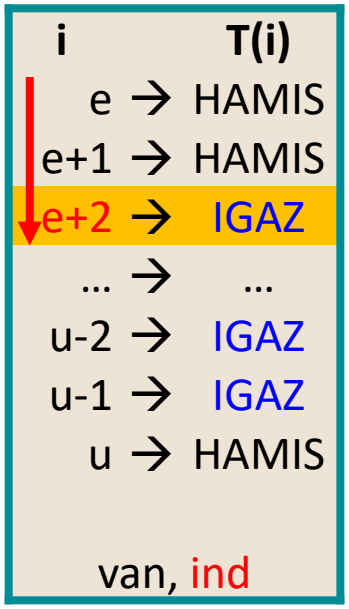


Feltételes maximumkeresés

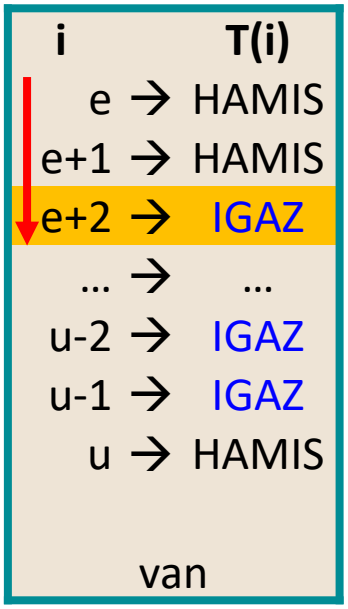


Programozási minták

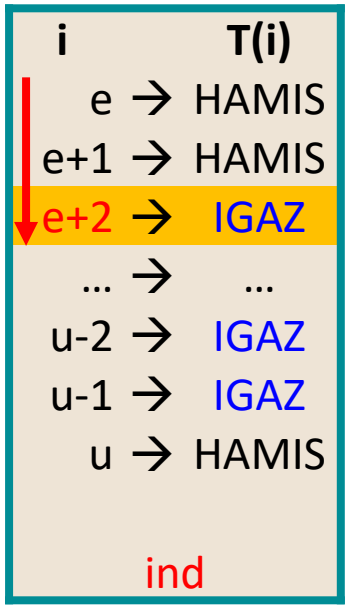
Keresés



Eldöntés

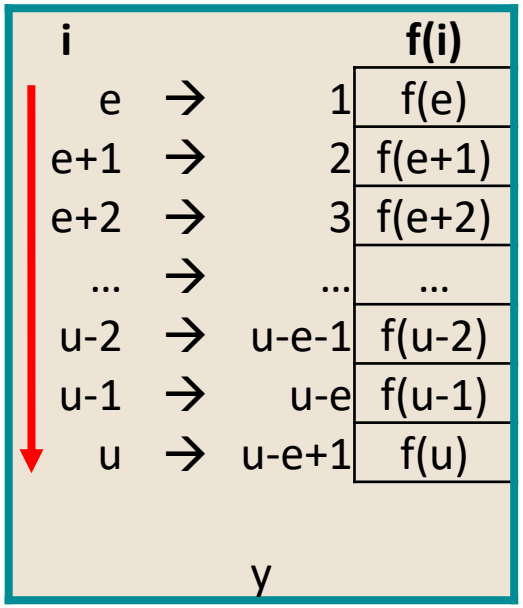


Kiválasztás

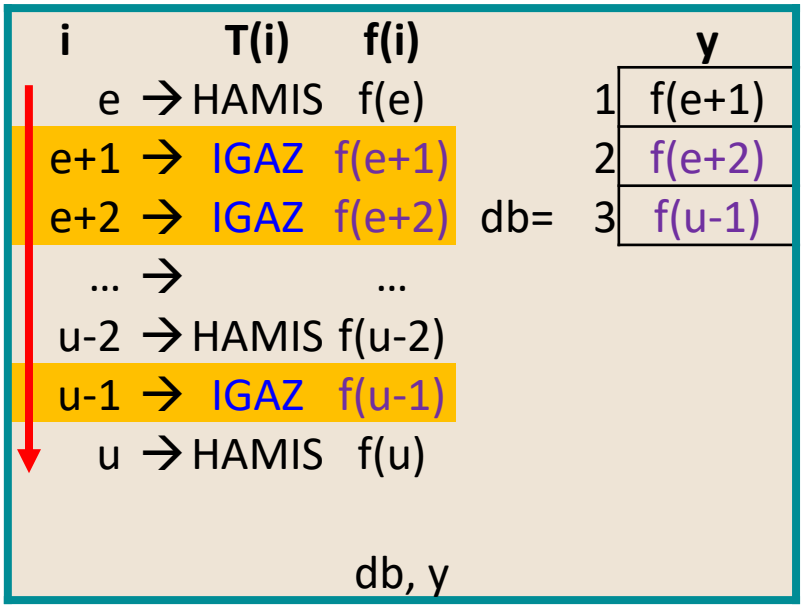


Programozási minták

Másolás



Kiválogatás



Feladatmegoldási minta

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Feladat a Mesterről

Gyorsabb vonat az előzőnél

Ismerjük N vonat menetidejét Budapestről Siófokra.

Írj programot, amely megad egy vonatot, amely gyorsabb, mint az előző!

Bemenet

A *standard bemenet* első sorában a vonatok száma van (1≤N≤100). A következő N sor mindegyike egy-egy egész számot tartalmaz, az egyes vonatok menetidejét (1≤M≤1000).

Kimenet

A *standard kimenet* első sorába egy az előzőnél gyorsabb vonat sorszámát kell írni (ha több ilyen is van, akkor az elsőt)! Ha nincs ilyen vonat, akkor -1-et kell írni!

Példa

Bemenet	Kimenet
6 118 200 199 116 200 122	3

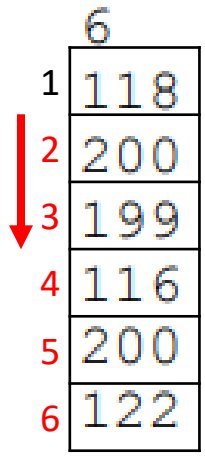
Feladatmegoldási minta
gyorsabb vonat az előzőnél

Biztosan van ilyen vonat? Ha igen, akkor melyik az?

→ keresés: adott tulajdonságú elem létezése és helye

Feladat:

Adj meg egy előzőnél gyorsabb vonatot!



Specifikáció:

```
Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N
Ef: -
Uf: (van,melyik)=KERES(i=2..n, midő[i]<midő[i-1])
```

1. Mik az intervallum határai? (2..6)
2. Milyen tulajdonságot vizsgálunk az intervallum egyes pontján?
3. Milyen néven tároljuk a keresés eredményeit?

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

van, ind

Feladatmegoldási minta
gyorsabb vonat az előzőnél

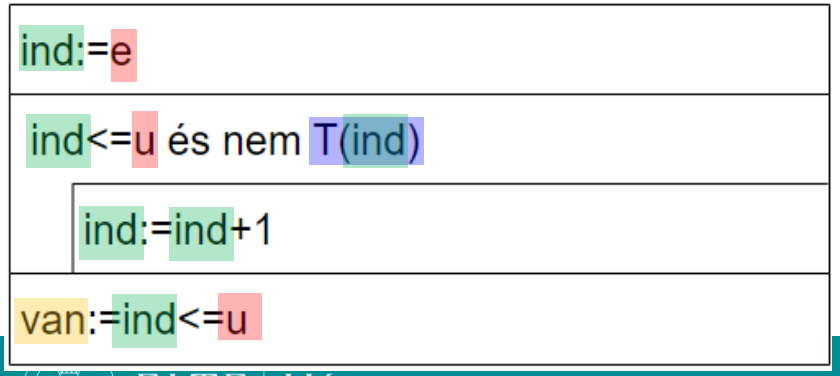
Feladatsablon
(mintafeladat)

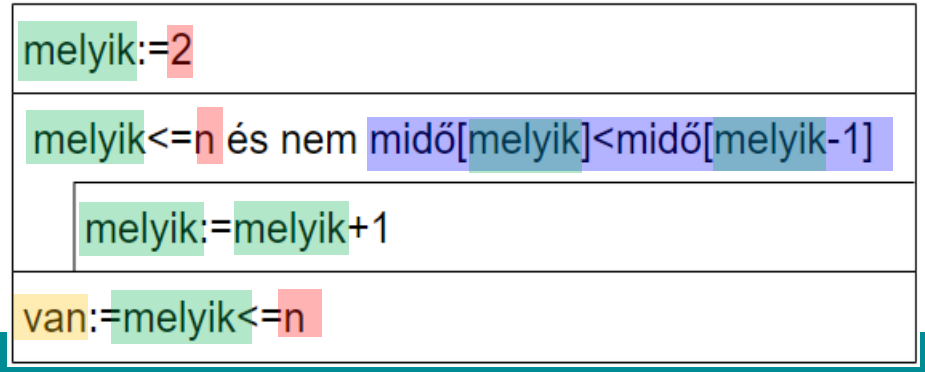
```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Ef: -
Uf: (van,ind)=KERES(i=e..u,
                    T(i))
```

Előzőnél gyorsabb vonat
(konkrét feladat)

```
Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N
Ef: -
Uf: (van,melyik)=KERES(i=2..n,
                       midő[i]<midő[i-1])
```

```
ind   ~ melyik
e..u  ~ 2..n
T(i)  ~ midő[i]<midő[i-1]
```





Feladatmegoldási minta
gyorsabb vonat az előzőnél

Kódolás alapsablonja

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)



  // Beolvasás (specifikáció be)



  // Feldolgozás (algoritmus, stuki)



  // Kiírás (specifikáció ki)



}
```

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Deklarálás

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



  // Feldolgozás (algoritmus, stuki)



  // Kiírás (specifikáció ki)



}
```

Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Beolvasás

```
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 = 1; i <= n; i++) {
    Console.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
  // Feldolgozás (algoritmus, stuki)



  // Kiírás (specifikáció ki)



}
```

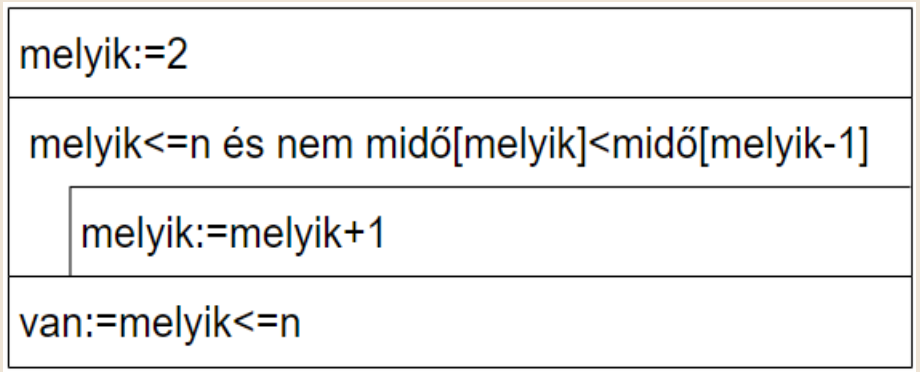
Be: n∈N,

midő∈N[1..n]

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Feldolgozás

```
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 = 1; i <= n; i++) {
    Console.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
  // Kiírás (specifikáció ki)


}
```



Feladatmegoldási minta
gyorsabb vonat az előzőnél

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

Kiírás

```
n = 6
1. menetido = 118
2. menetido = 200
3. menetido = 199
4. menetido = 116
5. menetido = 200
6. menetido = 122
Van, a(z) 3. vonat gyorsabb az előzőnél.
```

Ki: van∈L, melyik∈N

Feladatmegoldási minta
gyorsabb vonat az előzőnél

```
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 = 1; i <= n; i++) {
    Console.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine(melyik);
  }
  else {
    Console.WriteLine(-1);
  }
}
```

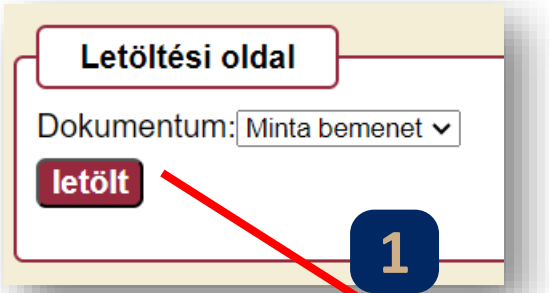
Kiírás módosítás

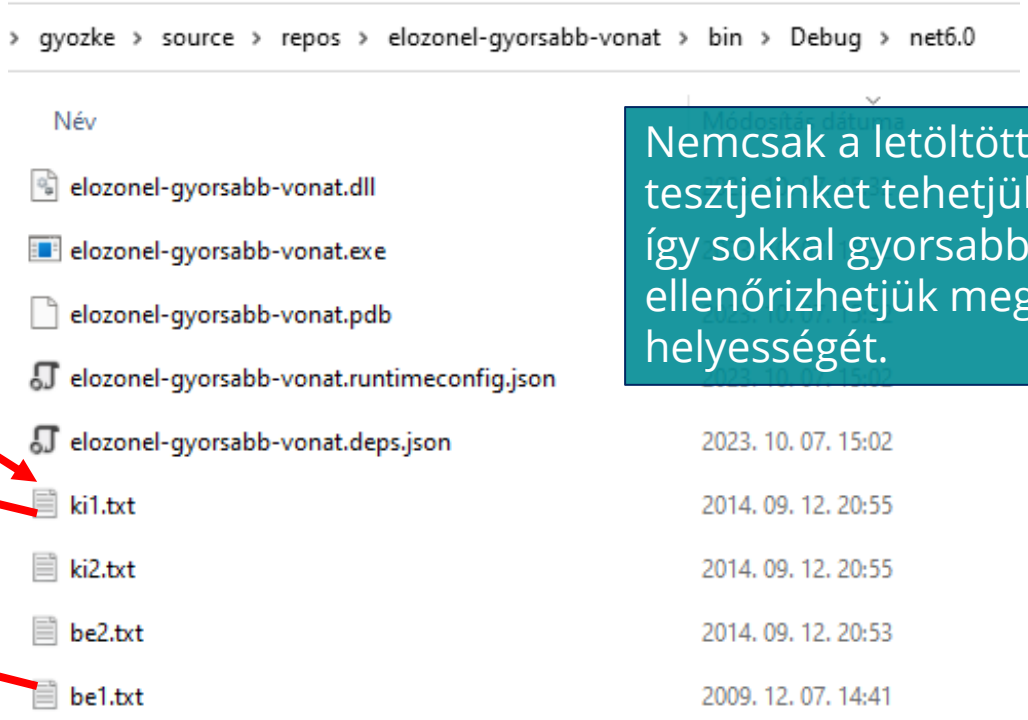
```
n = 6
1. menetido = 118
2. menetido = 200
3. menetido = 199
4. menetido = 116
5. menetido = 200
6. menetido = 122
3
```

Kimenet

A *standard kimenet* első sorába egy az előzőnél gyorsabb vonat sorszámát kell írni (ha több ilyen is van, akkor az elsőt)! Ha nincs ilyen vonat, akkor -1-et kell írni!

Feladatmegoldási minta
gyorsabb vonat az előzőnél





Nemcsak a letöltött, de saját tesztjeinket tehetjük fájlokba, így sokkal gyorsabban ellenőrizhetjük megoldásunk helyességét.

1

be1.txt

```
4
90
80
70
60
```

ki1.txt

```
2
```

2

4

3

```
c:\Users\gyozke\source\repos\elozonel-gyorsabb-
vonat\bin\Debug\net6.0>elozonel-gyorsabb-vonat.exe <be1.txt >ki1.txt
```

ki1.txt (saját)

```
n = 1. menetido = 2. menetido = 3. menetido =
4. menetido = 2
```

Feladatmegoldási minta
gyorsabb vonat az előzőnél

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

```
n = 6
1. menetido = 118
2. menetido = 200
3. menetido = 199
4. menetido = 116
5. menetido = 200
6. menetido = 122
3
```

```
c:\Users\gyozke\source\repos\elozonel-gyorsabb-vonat\bin\Debug\net6.0>elozonel-gyorsabb-vonat.exe <be1.txt >ki1.txt
```

ki1.txt
2

ki1.txt (saját)
2

Feladatmegoldási minta
gyorsabb vonat az előzőnél

```
using System;

static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;
  // Bekérés (specifikáció be)
  n = int.Parse(Console.ReadLine());
  mido = new int[n];
  for (int i = 0; i < n; i = i + 1) {
    mido[i] = int.Parse(Console.ReadLine());
  }
  // Feldolgozás
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik -
    melyik = melyik + 1;
  }
  van = melyik <= n;
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine(melyik);
  }
  else {
    Console.WriteLine(-1);
  }
```

Utolsó beadás eredménye

Összpont: 100/100

Teszt#	Pont	...Üzenet...	Futási idő
1.1	3/3	Helyes	0.032 sec
2.1	3/3	Helyes	0.032 sec
3.1	3/3	Helyes	0.035 sec
4.1	3/3	Helyes	0.031 sec
5.1	3/3	Helyes	0.032 sec
6.1	3/3	Helyes	0.033 sec
7.1	3/3	Helyes	0.033 sec
8.1	3/3	Helyes	0.032 sec
9.1	4/4	Helyes	0.037 sec
10.1	4/4	Helyes	0.033 sec
11.1	4/4	Helyes	0.032 sec
12.1	4/4	Helyes	0.036 sec
13.1	4/4	Helyes	0.037 sec

Visszavezetési esetek

Problémafelvetés

Eddig az utófeltételt hasonlítottuk össze, noha a programozási minta sablonja adott adatokra és ezeken értelmezett függvényekre mond valamit.

Mi van akkor, ha a mintafeladat és a konkrét feladat adatai eltérnek, pl. hiányzik vagy más jellegű adat van? Abban az esetben miért vezet helyes működésre a dolog?

Rövidítés = helyettesítés

- A rövidített utófeltétel az eredeti hosszabb (és logikailag helyes) változatot helyettesíti
- A konkrét feladatot felírva csak akkor írhatjuk fel a rövidített formátumot, ha az minden elemében megfeleltethető a hosszabb formátumnak.
- Ekkor a visszavezetés során előállt algoritmus is biztosan helyes megoldása lesz a feladatnak.

```
Uf: db=SZUMMA(i=e..u, 1, T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

Természetes visszavezetés
amikor minden stimmel

Hány páros szám van a és b intervallumban?

Feladatsablon
(mintafeladat)

```
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u,1,T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

Páros számok száma
(konkrét feladat)

```
Be: a∈Z, b∈Z
Ki: pdb∈N
Ef: -
Uf: pdb=SZUMMA(i=a..b,1,i mod 2=0)
```

Természetes visszavezetés
amikor minden stimmel

Hány páros szám van a és b intervallumban?

Feladatsablon
(mintafeladat)

```
Be: e∈Z, u∈Z
Ki:  db∈N
Ef: -
Uf:  db=SZUMMA(i=e..u,1,T(i)
```

```
Uf:  db=DARAB(i=e..u, T(i))
```

Páros számok száma
(konkrét feladat)

```
Be: a∈Z, b∈Z
Ki: pdb∈N
Ef: -
) Uf: pdb=SZUMMA(i=a..b,1,i mod 2=0)
```

Minden stimmel = az átnevezéseken kívül teljesen megegyezik
- adatok, futóindexek neve
- általános (jelentés nélküli) kifejezések, mint f(i), T(i)
- állandó értékű kifejezések, mint intervallum határ

Természetes visszavezetés
amikor minden stimmel

Hány páros szám van a és b intervallumban?

Feladatsablon
(mintafeladat)

```
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u,1,T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

Páros számok száma
(konkrét feladat)

```
Be: a∈Z, b∈Z
Ki: pdb∈N
Ef: -
Uf: pdb=SZUMMA(i=a..b,1,i mod 2=0)
```

```
Uf: pdb=DARAB(i=a..b, i mod 2=0)
```

Minden stimmel = az átnevezéseken kívül teljesen megegyezik
- adatok, futóindexek neve
- általános (jelentés nélküli) kifejezések, mint f(i), T(i)
- állandó értékű kifejezések, mint intervallum határ

Általános visszavezetés
szigorúbb előfeltétel

Hány prímszám van a és b intervallumban?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u, 1, T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

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

A konkrét feladat előfeltétele lehet szigorúbb a mintáénál. Ha sok adatra helyes, akkor nyilván a szűkítettre is helyes lesz.

Prímszámok száma

```
Be: a∈Z, b∈Z
Ki: pdb∈N
Ef: a>0
Uf: pdb=SZUMMA(i=a..b, 1, prím(i))
```

```
Uf: pdb=DARAB(i=a..b, prím(i))
```

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

Általános visszavezetés
gyengébb utófeltétel

Adj meg egy prímszámot az a és b intervallumban!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Ef: -
Uf: van=∃i∈[e..u]:(T(i)) és
    van->(ind∈[e..u] és T(ind) és
       ∀i∈[e..ind-1]:(nem T(i)))
```

```
Uf: (van,ind)=KERES(i=e..u,T(i))
```

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

A konkrét feladat utófeltétele lehet gyengébb a mintáénál. Ha a feladat több megoldást is elfogad, akkor nyilván a minta szűkített megoldása is helyes lesz. → Szűkítsük le a feladatot!

Prímszámok száma

```
Be: a∈Z, b∈Z
Ki: van∈L, p∈N
Ef: a>0
Uf: van=∃i∈[a..b]:(prím(i)) és
    van->(p∈[a..b] és prím(p))
```

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

Általános visszavezetés
gyengébb utófeltétel

Adj meg egy prímszámot az a és b intervallumban!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Ef: -
Uf: van=∃i∈[e..u]:(T(i)) és
    van->(ind∈[e..u] és T(ind) és
       ∀i∈[e..ind-1]:(nem T(i)))
```

```
Uf: (van,ind)=KERES(i=e..u,T(i))
```

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

A konkrét feladat utófeltétele lehet gyengébb a mintáénál. Ha a feladat több megoldást is elfogad, akkor nyilván a minta szűkített megoldása is helyes lesz. → Szűkítsük le a feladatot!

Prímszámok száma

```
Be: a∈Z, b∈Z
Ki: van∈L, p∈N
Ef: a>0
Uf: van=∃i∈[a..b]:(prím(i)) és
    van->(p∈[a..b] és prím(p) és
       ∀i∈[e..p-1]:(nem prím(i)))
```

```
Uf: (van,p)=KERES(i=a..b,prím(i))
```

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

Alteres visszavezetés
kevesebb kimeneti adat

Az a és b intervallumbeli egész értékeken hol veszi fel a sin(x) függvény a legnagyobb értékét?

Feladatsablon

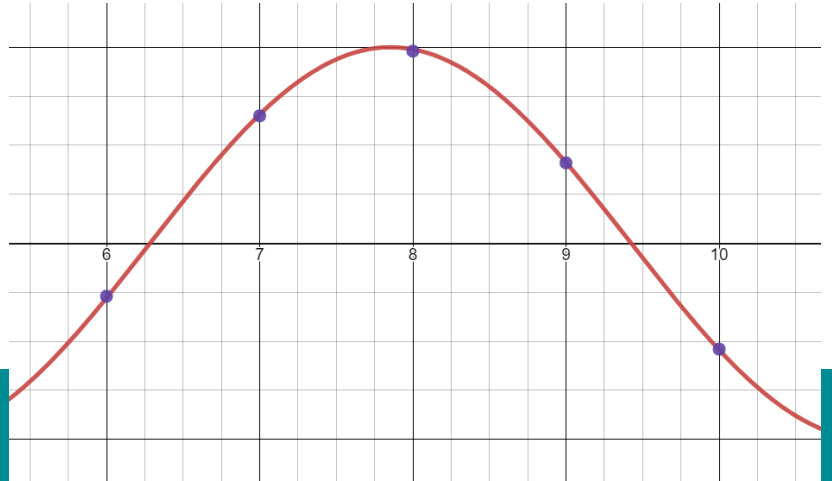
```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H

Ef: -
Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

⬇

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

sin(x) maximuma

```
Be: a∈Z, b∈Z
Ki: hol∈Z

Ef: -
Uf: hol∈[a..b] és
    ∀i∈[a..b]:(sin(hol)>=sin(i))
```



Alteres visszavezetés
kevesebb kimeneti adat

Az a és b intervallumbeli egész értékeken hol veszi fel a sin(x) függvény a legnagyobb értékét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z
Sa: maxért∈H
Ef: -
Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

sin(x) maximuma

```
Be: a∈Z, b∈Z
Ki: hol∈Z
Sa: maxért∈R
Ef: -
Uf: hol∈[a..b] és
    ∀i∈[a..b]:(sin(hol)>=sin(i)) és
    maxért=sin(hol)
```

```
Uf: (hol,maxért)=
                MAX(i=a..b,sin(i))
```

A mintafeladat kimeneti adatait **segédadat**tá lehet minősíteni.

→ szigorúbb lesz az utófeltétel, hiszen a megmaradt kimeneti adaton túl másra is tesz megszorítást, de ez nem baj (ld. előzőleg)

→ segédváltozó az algoritmusban

Ezzel párhuzamosan a konkrét feladatba is segédadatot kell bevezetni és szigorítani kell az utófeltételét.

Alteres visszavezetés
kevesebb kimeneti adat

Az a és b intervallumbeli egész értékeken hol veszi fel a sin(x) függvény a legnagyobb értékét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z
Sa: maxért∈H
Ef: -
Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

sin(x) maximuma

```
Be: a∈Z, b∈Z
Ki: hol∈Z

Ef: -
```

maxind, maxért	~	hol, maxért
e..u	~	a..b
f(i)	~	sin(i)

```
Uf: (hol,)=
                MAX(i=a..b,sin(i))
```

A konkrét feladatnak nincs szüksége a segédadatra, csak azért vezettük be, hogy a rövidítés és visszavezetés alkalmazható legyen. Végső soron teljesen ki is hagyható a specifikációból.
A visszavezetési táblázatban ugyanolyan nevűnek tekintjük.

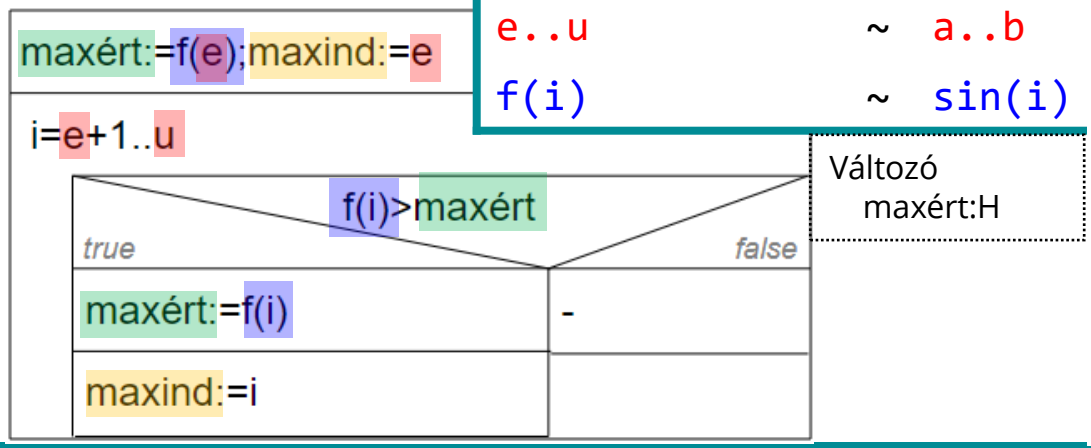
Alteres visszavezetés
kevesebb kimeneti adat

Az a és b intervallumbeli egész értékeken hol veszi fel a sin(x) függvény a legnagyobb értékét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z
Sa: maxért∈H
Ef: -
Uf: (maxind,maxért)=
```

MAX(i=e..u,f(i))



maxind, maxért ~ hol, maxért
e..u ~ a..b
f(i) ~ sin(i)

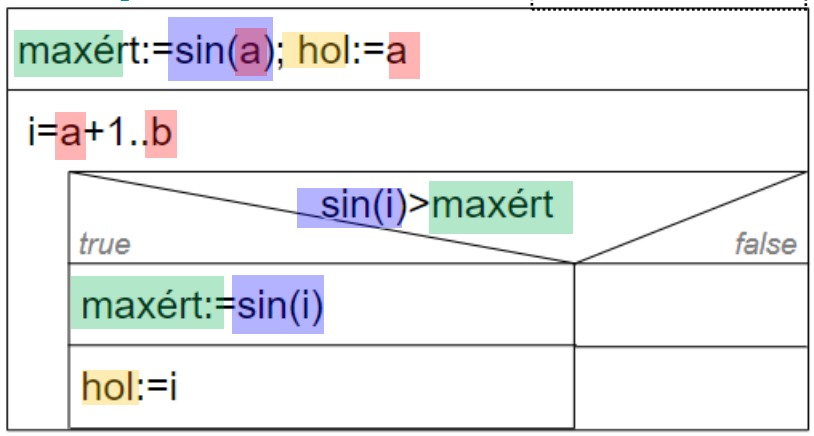
Változó
maxért:H

sin(x) maximuma

```
Be: a∈Z, b∈Z
Ki: hol∈Z

Ef: -
Uf: (hol,)=
```

MAX(i=a..b,sin(i))

Változó
maxért:Valós



A kihagyott adat segédváltozó lesz az algoritmusban.

Alteres visszavezetés
példa: eldöntés

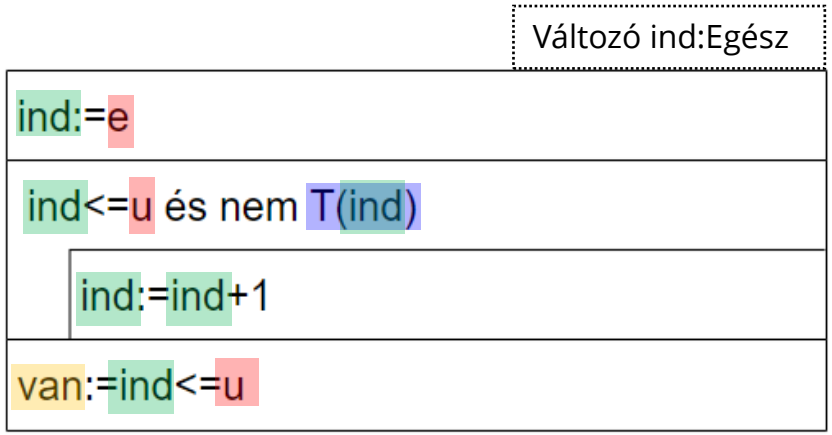
Eldöntés

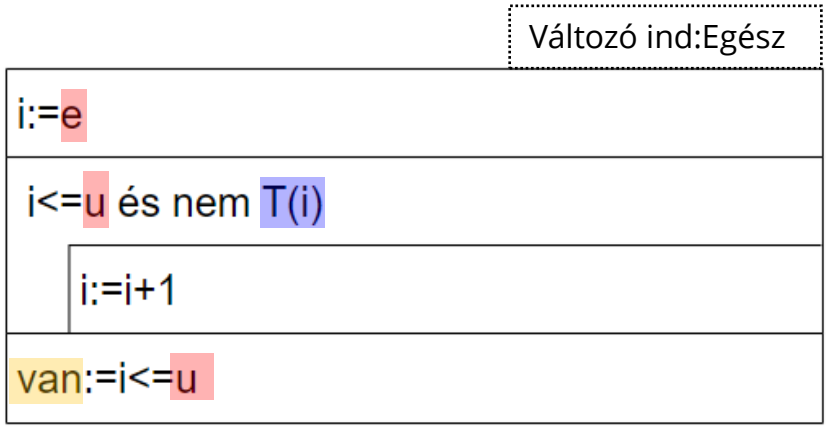
Keresés

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Sa: ind∈Z
Ef: -
Uf: (van,ind)=KERES(i=e..u,T(i))
```

Eldöntés

```
Be: e∈Z, u∈Z
Ki: van∈L
Sa: ind∈Z
Ef: -
Uf: (van,ind)=KERES(i=e..u,T(i))
Uf: (van,)=KERES(i=e..u,T(i))
Uf: van=VAN(i=e..u,T(i))
```





A kihagyott adat segédváltozó lesz az algoritmusban.

Alteres visszavezetés
kevesebb bemeneti adat

Adjuk meg egy természetes szám valódi osztóinak számát!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u,1,T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

Valódi osztók száma

```
Be:      n∈N
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=2..n div 2,1,i|n)
```

Ha a mintafeladat egy olyan bemeneti adata hiányzik a konkrét feladatból, amelyik nem változtatja értékét, akkor ezt helyettesíthetjük egy olyan adattal, amelyik értéke állandó (konstans). Ettől a feladat nem változik meg. Az értéket az előfeltételben határozzuk meg, azaz a konkrét feladat szigorítja azt. A bevezetett konstans értékű adatnak megfelelő változókat segédváltozóknak tekinthetjük.

Alteres visszavezetés
kevesebb bemeneti adat

Adjuk meg egy természetes szám valódi osztóinak számát!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u,1,T(i))
```

```
Uf: db=DARAB(i=e..u, T(i))
```

Valódi osztók száma

```
Be: e∈N, n∈N
Ki: db∈N
Ef: e=2
Uf: db=SZUMMA(i=e..n div 2,1,i|n)
```

```
Uf: db=DARAB(i=e..n div 2, i|n)
```

Ha a mintafeladat egy olyan bemeneti adata hiányzik a konkrét feladatból, amelyik nem változtatja értékét, akkor ezt helyettesíthetjük egy olyan adattal, amelyik értéke állandó (konstans). Ettől a feladat nem változik meg. Az értéket az előfeltételben határozzuk meg, azaz a konkrét feladat szigorítja azt. A bevezetett konstans értékű adatnak megfelelő változókat segédváltozóknak tekinthetjük.

Paraméteres visszavezetés
több bemeneti adat

Osztja-e k az a és b közötti számok valamelyikét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L
Ef: -
Uf: van=∃i∈[e..u]:(T(i))
```

```
Uf: van=VAN(i=e..u, T(i))
```

Valódi osztók száma

```
Be: a∈Z, b∈Z, k∈N
Ki: van∈L
Ef: k>0
Uf: van=∃i∈[a..b]:(k|i)
```

Ha a konkrét feladat olyan plusz bemeneti adatot is tartalmaz, amelynek értéke állandó, akkor

1. rögzítsük az értékét (konstans, nem kell feltüntetni az adatok között)
2. a visszavezetés így adott érték mellett megtehető, hiszen az adatok egyeznek
3. ez k minden lehetséges értéke mellett megtehető, ezért ezt olyan adatként jelezzük, amely a program elején felvesz egy állandó értéket

Paraméteres visszavezetés
több bemeneti adat

Osztja-e k az a és b közötti számok valamelyikét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L
Ef: -
Uf: van=∃i∈[e..u]:(T(i))
```

```
Uf: van=VAN(i=e..u, T(i))
```

Valódi osztók száma

```
Be: a∈Z, b∈Z
Ki: van∈L
Ef: k>0
Uf: van=∃i∈[a..b]:(13|i)
```

k=13

```
Uf: van=VAN(i=a..b, 13|i)
```

Ha a konkrét feladat olyan plusz bemeneti adatot is tartalmaz, amelynek értéke állandó, akkor

1. rögzítsük az értékét (konstans, nem kell feltüntetni az adatok között)
2. a visszavezetés így adott érték mellett megtehető, hiszen az adatok egyeznek
3. ez k minden lehetséges értéke mellett megtehető, ezért ezt olyan adatként jelezzük, amely a program elején felvesz egy állandó értéket

Paraméteres visszavezetés
több bemeneti adat

Osztja-e k az a és b közötti számok valamelyikét?

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L
Ef: -
Uf: van=∃i∈[e..u]:(T(i))
```

```
Uf: van=VAN(i=e..u, T(i))
```

Valódi osztók száma

```
Be: a∈Z, b∈Z, k∈N
Ki: van∈L
Ef: k>0
Uf: van=∃i∈[a..b]:(k|i)
```

minden k-ra

```
Uf: van=VAN(i=a..b, k|i)
```

Ha a konkrét feladat olyan plusz bemeneti adatot is tartalmaz, amelynek értéke állandó, akkor

1. rögzítsük az értékét (konstans, nem kell feltüntetni az adatok között)
2. a visszavezetés így adott érték mellett megtehető, hiszen az adatok egyeznek
3. ez k minden lehetséges értéke mellett megtehető, ezért ezt olyan adatként jelezzük, amely a program elején felvesz egy állandó értéket

Példa

Ez miért is tehető meg, amikor nem is hasonlítanak?

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

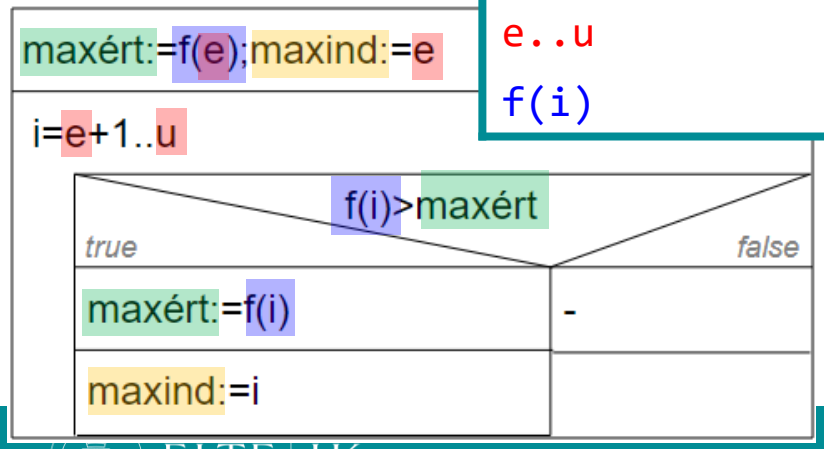
```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u

Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

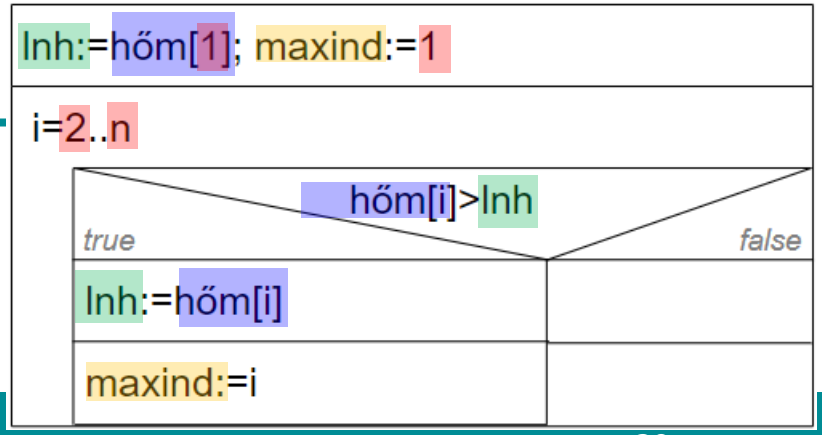
Legnagyobb hőmérséklet

```
Be: n∈N, hőm∈R[1..n]
Ki: lnh∈R
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: (,lnh)=
                MAX(i=1..n,hőm[i])
```

maxind, maxért	~	maxind, lnh
e..u	~	1..n
f(i)	~	hőm[i]

Változó
maxind:Egész





Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Legnagyobb hőmérséklet

```
Be:             n∈N, hőm∈R[1..n]
Ki:             lnh∈R
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: ∃i∈[1..n]:(lnh=hőm[i]) és
    ∀i∈[1..n]:(lnh>=hőm[i])
```

Ez maximumkiválasztás!

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

Be:		n∈N, hőm∈R[1..n]
Ki:		lnh∈R
Ef:	n>0 és ∀i∈[1..n]:(-100<=hőm[i]<=100)	
Uf:	∃i∈[1..n]:(lnh=hőm[i]) és	
	∀i∈[1..n]:(lnh>=hőm[i])	

De ha összehasonlítjuk, akkor maximumkiválasztást csak nyomokban tartalmaz!

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

```
Be:                 n∈N, hőm∈R[1..n]
Ki:                 lnh∈R
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: ∃i∈[1..n]:(lnh=hőm[i]) és
    ∀i∈[1..n]:(lnh>=hőm[i])
```

1. Paraméteres visszavezetés

A tömb olyan értékét nem változtató bemenő adat, amelyet – értékét állandó paraméternek véve – elhagyhatónak tekintünk.

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:           lnh∈R
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
    és e=1 és u=n és e<=u
Uf: ∃i∈[1..n]:(lnh=hőm[i]) és
    ∀i∈[1..n]:(lnh>=hőm[i])
```

2. Alteres visszavezetés (kevesebb bemeneti adat)

A feladat implicit módon tartalmazza az intervallum határait. Ezek állandó értékek → beemelhetők az adatok közé → előfeltétel

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Sa: maxind∈Z
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)

Uf: (maxind,maxért)=
                MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:                lnh∈R
Sa: maxind∈Z
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
    és e=1 és u=n és e<=u
Uf: ∃i∈[1..n]:(lnh=hőm[i]) és
    ∀i∈[1..n]:(lnh>=hőm[i])
```

3. Alteres visszavezetés (kevesebb kimeneti adat)

A mintafeladat kimeneti adata segédadattá avanzsálható, a konkrét feladatban ugyanilyen néven megjelenik segédadat

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki:              maxért∈H
Sa: maxind∈Z
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
    maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                  MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:           lnh∈R
Sa: maxind∈Z
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
    és e=1 és u=n és e<=u
Uf: ∃i∈[1..n]:(lnh=hőm[i]) és
    ∀i∈[1..n]:(lnh>=hőm[i])
```

4. Általános visszavezetés (szigorúbb előfeltétel)

A konkrét feladat előfeltétele szigorúbb, így szűkíti a megoldás értelmezési tartományát, ami megtehető

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki:              maxért∈H
Sa: maxind∈Z
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind)>=f(i)) és
     maxért=f(maxind)
```

```
Uf: (maxind,maxért)=
                  MAX(i=e..u,f(i))
```

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:           lnh∈R
Sa: maxind∈Z
Ef: n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
    és e=1 és u=n és e<=u
Uf: maxind∈[1..n] és
    ∀i∈[1..n]:(hőm[maxind]>=hőm[i])
 és lnh=hőm[maxind]
```

5. Általános visszavezetés (gyengébb utófeltétel)

Utófeltétel átalakítása: a bevezetett segédadattal átfogalmazható, mit tudunk lnh és e,u,hőm kapcsolatáról → szigorítjuk az utófeltételt

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki:              maxért∈H
Sa: maxind∈Z
Ef: e<=u


Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind  )>=f  (i))
 és maxért=f  (maxind)
```

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:              lnh   ∈R
Sa: maxind∈Z
Ef: e<=u és e=1 és u=n és
    n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: maxind∈[1..n] és
    ∀i∈[1..n]:(hőm[maxind]>=hőm[i])
 és lnh   =hőm[maxind]
```

Kis átrendezéssel a konkrét feladat megfeleltethető a mintafeladatnak.

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki:              maxért∈H
Sa: maxind∈Z
Ef: e<=u

Uf: maxind∈[e..u] és
    ∀i∈[e..u]:(f(maxind  )>=f  (i))
 és maxért=f  (maxind)
```

```
Uf: (maxind,maxért)=
                  MAX(i=e..u,f(i))
```

maxind, maxért	~	maxind, lnh
e..u	~	1..n
f(i)	~	hőm[i]

Legnagyobb hőmérséklet

```
Be: e∈N, u∈N, n∈N, hőm∈R[1..n]
Ki:              lnh   ∈R
Sa: maxind∈Z
Ef: e<=u és e=1 és u=n és
    n>0 és
    ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: maxind∈[1..n] és
    ∀i∈[1..n]:(hőm[maxind]>=hőm[i])
 és lnh   =hőm[maxind]
```

```
Uf: (maxind,lnh)=
                  MAX(i=1..n,hőm[i])
```

A visszavezetés megtehető

Példa

Egy hőmérsékletstatisztika alapján add meg a legnagyobb hőmérsékletet!

Feladatsablon

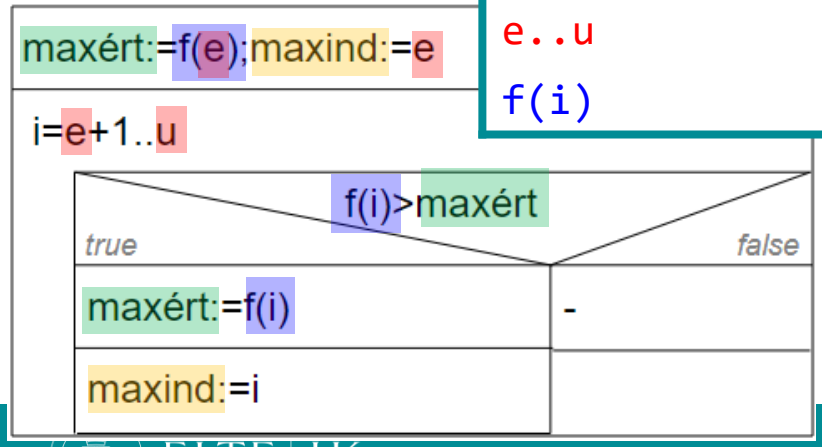
Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u

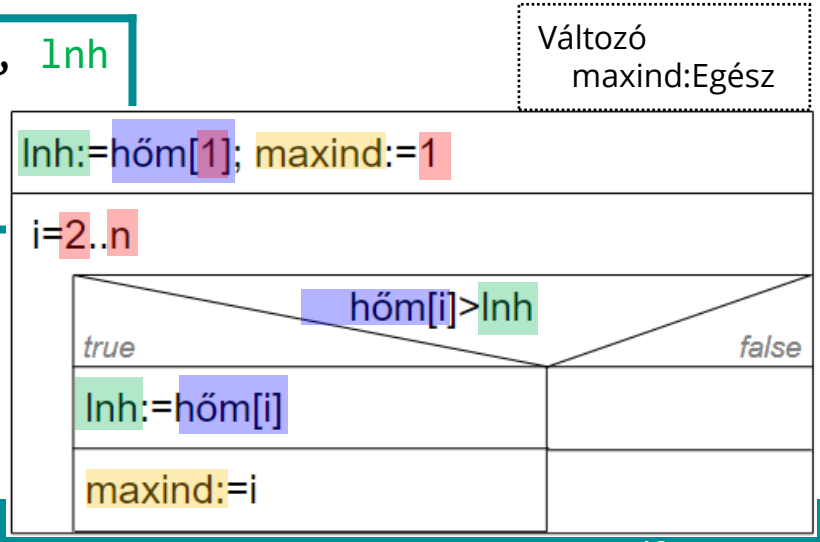
Uf: (maxind,maxért)=
MAX(i=e..u,f(i))

Legnagyobb hőmérséklet

Be: n∈N, hőm∈R[1..n]
Ki: lnh∈R
Ef: n>0 és
∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: (,lnh)=
MAX(i=1..n,hőm[i])

maxind, maxért	~	maxind, lnh
e..u	~	1..n
f(i)	~	hőm[i]





Függvények

Négyzet

Példa:
x=3,3 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈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 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈R
Ef: -
Uf: y=négyzet(x)
```

Egy függvény mögé rejtettük a négyzetre emelést.

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

Algoritmus:

```
y:=négyzet(x)
```

Négyzet

Matematika:
$x \mapsto x^2$
Másként:
$f: \mathbb{R} \to \mathbb{R}, f(x) = x^2$
Esetünkben:
$négyzet: \mathbb{R} \to \mathbb{R}, négyzet(n) = n^2$

Feladat:

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

Specifikáció:

```
Be: n∈R
Ki: négyzet(n)∈R
Ef: -
Uf: négyzet(n)=n*n
```

A bemenetben a függvény **paraméter**ei (∈értelmezési tartomány), a kimenetben a függvény paraméteres **érték**e szerepel (∈értékkészlet), az utófeltételben az **összefüggés**.

Algoritmus:

négyzet(n:Valós): Valós
négyzet:=n*n

Függvényszignatúra

négyzet(n:Valós): Valós

négyzet:=n*n

Visszatérési érték meghatározása a függvénynévhez való értékadással

Négyzet

Példa:
x=3,3 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈R
Fv: négyzet:R->R, négyzet(n)=n*n
Ef: -
Uf: y=négyzet(x)
```

Matematika:
$x \mapsto x^2$
Másként:
$f: \mathbb{R} \to \mathbb{R}, f(x) = x^2$
Esetünkben:
$négyzet: \mathbb{R} \to \mathbb{R}, négyzet(n) = n^2$

Szignatúra

Formális paraméter

Aktuális paraméter

Algoritmus:

```
y:=négyzet(x)
```

```
négyzet(n:Valós): Valós
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

Feladat:

Példa:
a=10, b=8 → m=10

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: m∈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 → m=10

Feladat:

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: m∈Z
Fv: max:Z x Z->Z, max(a,b)={a, ha a>=b;
                            b egyébként}
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

Feladat:

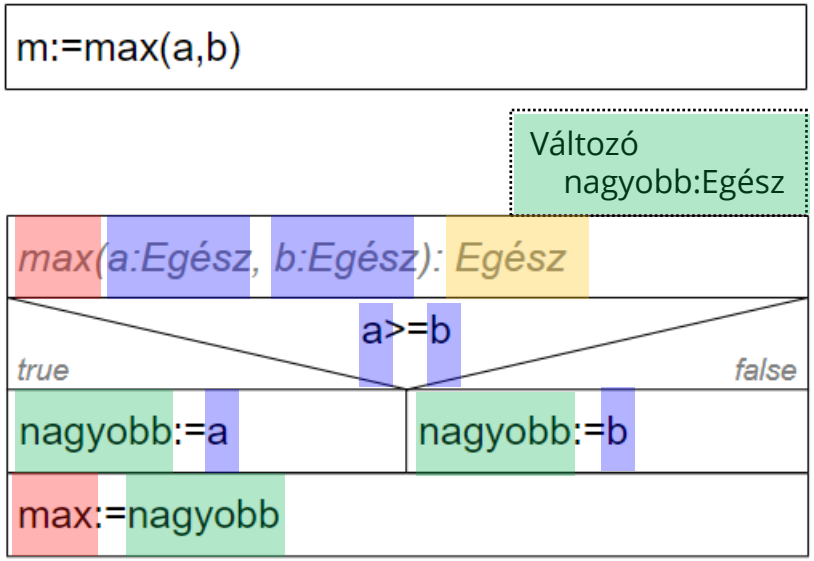
Példa:
a=10, b=8 → m=10

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

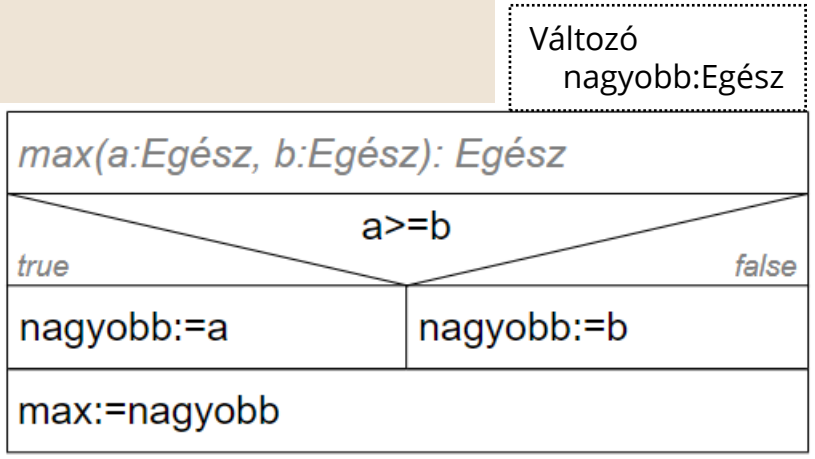
```
Be: a∈Z, b∈Z
Ki: m∈Z
Fv: max:Z x Z->Z,
    max(a,b)={a, ha a>=b;
              b egyébként}
Ef: -
Uf: m=max(a,b)
```

Algoritmus:



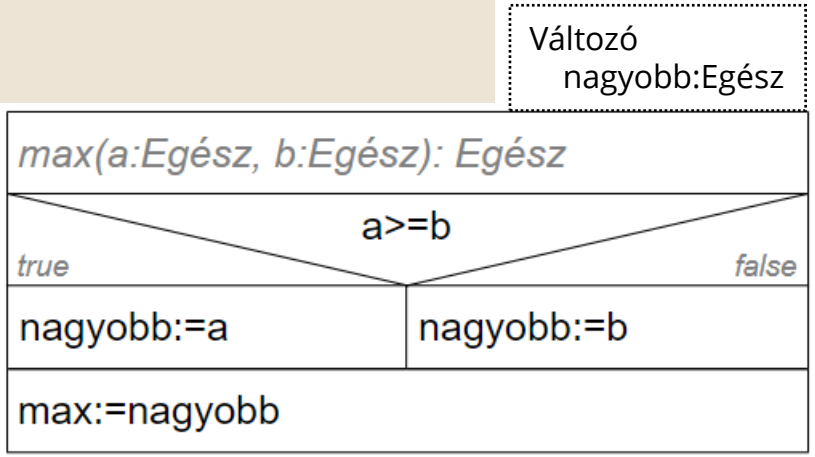
Maximum
kód

```
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;
}
```



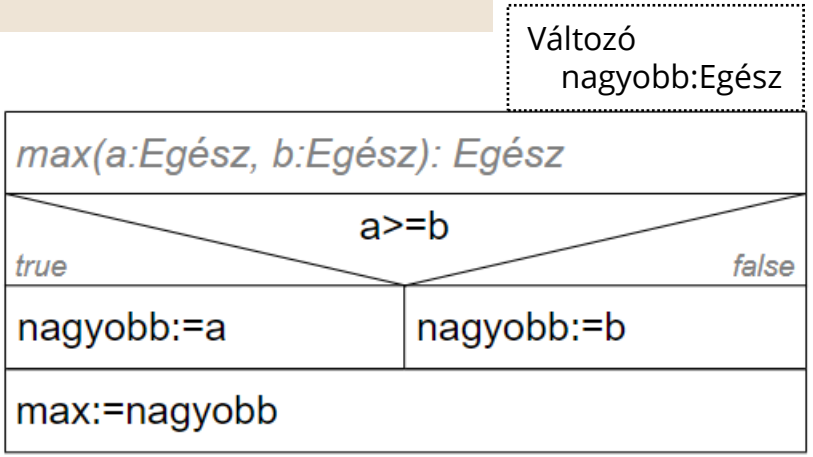
Maximum
kód

```
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

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

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

Be: n∈Z, d∈Z
Ki: n'∈Z

Ef: -
Uf: n'=növel(n,d)

Mi derül ki a használatból?

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

Növelés

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

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

```
Be: n∈Z, d∈Z
Ki: n'∈Z
Fv: növel:Z x Z->Z,
    növel(szám,mivel)=szám+mivel
Ef: -
Uf: n'=növel(n,d)
```

Mi derül ki a használatból?

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

Növelés

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

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

```
Be: n∈Z, d∈Z
Ki: n'∈Z
Fv: növel:Z x Z->Z,
    növel(szám,mivel)=szám+mivel
Ef: -
Uf: n'=növel(n,d)
```

Változó paraméterként határozzuk meg!

Algoritmus:

növel(a, 8)

növel(Vált szám:Egész, mivel:Egész)

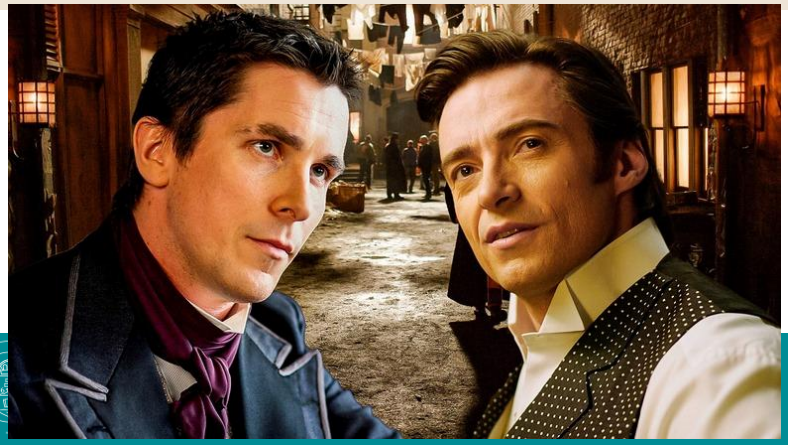
szám:=szám+mivel

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 szam, int mivel) {
  szam = szam + 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

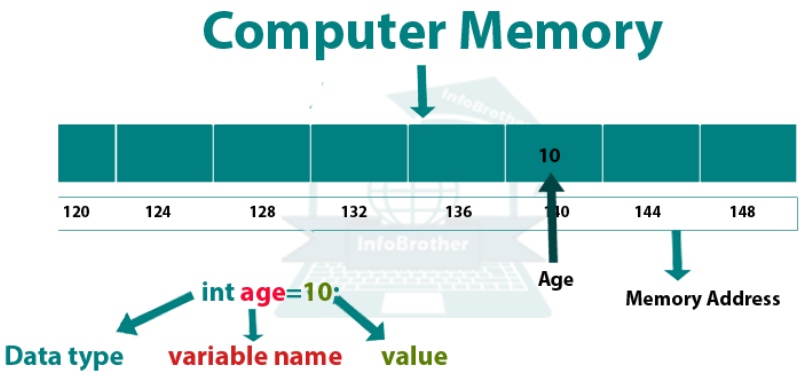
növel(Vált szám:Egész, mivel:Egész)
szám:=szám+mivel

```
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.



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 → a'=8, b'=10

Specifikáció:

Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈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 → a'=8, b'=10

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈Z
Fv: csere:Z x Z->Z x 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 → a'=8, b'=10

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈Z
Fv: csere:Z x Z->Z x Z,
    csere(x,y)=(y, x)
Ef: -
Uf: (a',b')=csere(a,b)
```

Algoritmus:

csere(a, b)

Változó z:Egész

csere(Vált x:Egész, Vált y:Egész)
z:=x
x:=y
y:=z

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
```

Változó z:Egész

csere(Vált x:Egész, Vált y:Egész)
z:=x
x:=y
y:=z

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

Monolitikus kód

```
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 = 1; i <= n; i++) {
    Console.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine("Van, a(z) {0}. vonat gyorsabb az előzőnél.", melyik);
  }
  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, ido); // 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 = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
```

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 = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
}
```

Kódszervezés függvén
előzőnél gyorsabb vonat

Specifikáció:

```
Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N
Ef: -
Uf: (van,melyik)=KERES(i=2..n,
                       midő[i]<midő[i-1])
```

A megoldás tekinthető olyan függvénynek, ami

```
(n, mido) → (van, melyik)
```

Másképpen:

```
keres_vonat:NxN[] -> LxN
(van,melyik):=keres_vonat(n,mido)
```

```
tatic void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,
  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 = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;

  return (van, melyik)
}
```

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);
  }
  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 = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
static void keres_vonat1(int n, int[] mido, out bool van, out int melyik) {
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
}
static void kiir(bool van, int melyik) {
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine(melyik);
  }
  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_vonat2(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 = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
static (bool van, int melyik) keres_vonat2(int[] mido) {
  // Feldolgozás (algoritmus, stuki)
  int melyik;
  bool van;

  int n = mido.Length;
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 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 deklarálva.

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.

Függvények
fogalmak (c#)

C# tudnivalók – összefoglalás:

- **Formális** skalár paraméter:
  - bemeneti → nincs speciális kulcsszó
  - kimeneti → ref/out a speciális prefix „kulcsszó”
- **Aktuális** skalár paraméter:
  ha a megfelelő formális paraméter
  - bemeneti → akár konstans, akár változó
  - kimeneti → **csak** változó, ref/out a speciális prefix-szel

  lehet.

  ref: a paramétert meg lehet változtatni
  out: a paramétert meg kell változtatni

Pontosabban: **nem tömb**

Pontosabban: **nem tömb**

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

vek
c#)

C# tudnivalók – összefoglalás:

- Skalár paraméterátadás:
  - **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.:

Input-paraméterek.

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

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

In-/Output-paraméter.

Input-paraméterek.

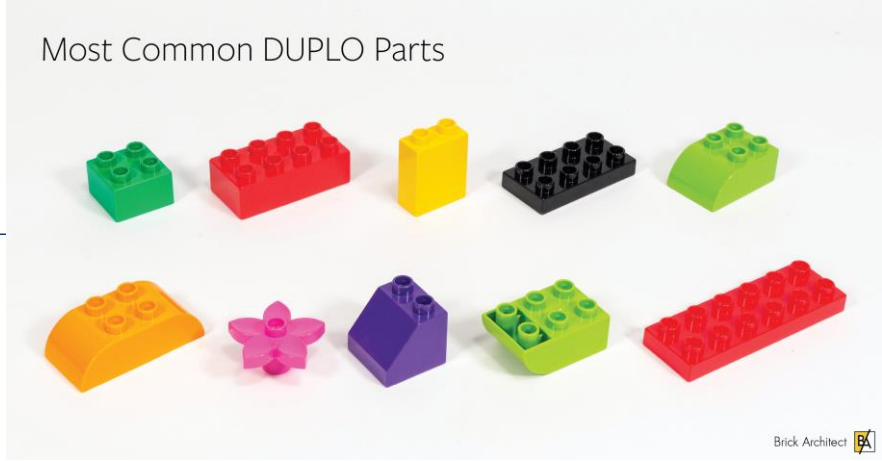
```
static void max(int x, int y, ref int max_xy)
static void max(int x, int y, out int max_xy)
```

Input-paraméterek.

Output paraméter.

Összefoglalás

Programozási minták



1. Összegzés
   a. Megszámolás
   b. (Feltételes összegzés)
   c. Másolás
   d. Kiválogatás
2. Maximumkiválasztás
   a. Minimumkiválasztás
3. Feltételes maximumkeresés
4. Keresés
   a. Eldöntés
   b. Mind eldöntés
5. Kiválasztás

Függvények

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