Gyűjtemények felsorolása, algoritmus minták

Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

1. rész
Gyűjtemények és Felsorolók

Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Gyűjtemény és feldolgozása

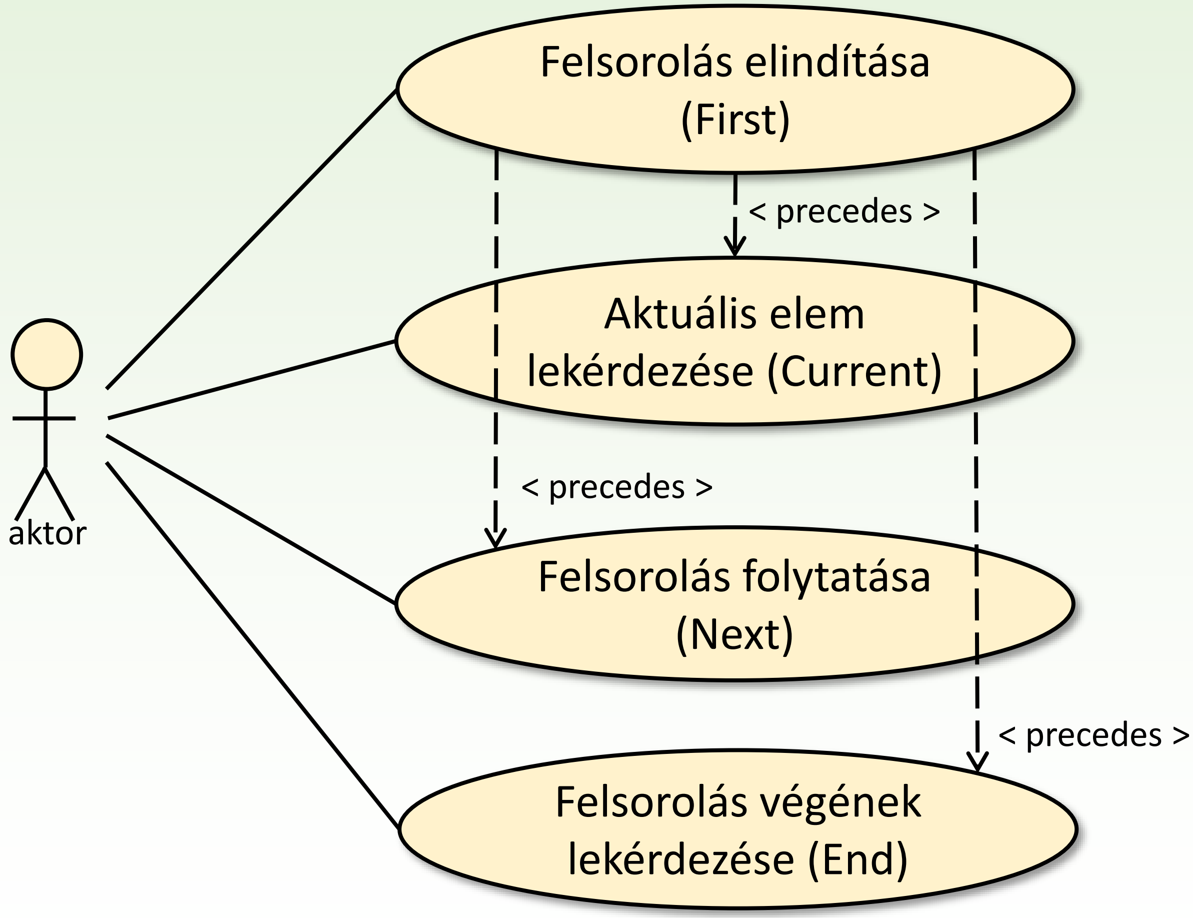
- A gyűjtemény (tároló, kollekció) egy olyan objektum, amely elemek tárolására alkalmas: elemek elhelyezésére, visszakeresésére, eltávolítására biztosít műveleteket.
  Például: halmaz, zsák, sorozat (amely speciálisan lehet verem vagy sor, ha korlátozzuk a műveleteit), tömb, fa, gráf, vagy akár rekord.
- Egy gyűjteményt virtuálisnak mondunk, ha nem kell eltárolni explicit módon az elemeit. Például egész számok intervalluma esetén elég az intervallum végpontjait megadni, vagy egy természetes szám prím-osztóinak visszakereséséhez elég az adott természetes számot ismerni.
- Egy gyűjtemény feldolgozásán a benne levő elemek feldolgozását értjük.
  - Keressük egy halmaz valamilyen szempont szerinti legnagyobb elemét!
  - Hány negatív szám van egy számsorozatban?
  - Keressük meg egy egészeket tartalmazó tömb azon pozitív elemét, amelyet a tömb visszafelé bejárásával elsőként kapunk meg úgy, hogy csak minden második elemet vizsgáljuk meg!
  - Soroljuk fel az *n* természetes szám pozitív prím-osztóit!

Felsorolás

- Egy gyűjtemény elemeinek feldolgozásához fel kell tudnunk sorolni a gyűjtemény elemeit.
- A felsorolásra (bejárásra) úgy tekinthetünk, mint a felsorolandó gyűjtemény elemeiből (jelölje ezek halmazát az E) képzett véges sorozatra, amelyre az alábbi négy művelet érvényes:
  - First() : rááll a sorozat első elemére – elkezdi a felsorolást
  - Next() : rááll a sorozat következő elemére – folytatja a felsorolást
  - e := Current() (e:E): visszaadja a sorozat (felsorolás) aktuális elemét
  - l := End() (l:$\mathbb{L}$) : jelzi, hogy a sorozat (felsorolás) végére értünk-e

Felsorolás használati eset diagramja

- A felsorolás használati eseteinek meghatározott sorrendje van.



Felsorolás algoritmusa

- Célszerű a felsorolást olyan algoritmusba ágyazni, amely garantálja, hogy a felsorolás műveletei csak akkor kerüljenek végrehajtásra, amikor az értelmes.

First()

¬End()

Feldolgoz(Current())

Next()

```
for( First(); !End(); Next() )
{
    Process(Current());
}
```

a felsorolt elemek típusát helyettesíti

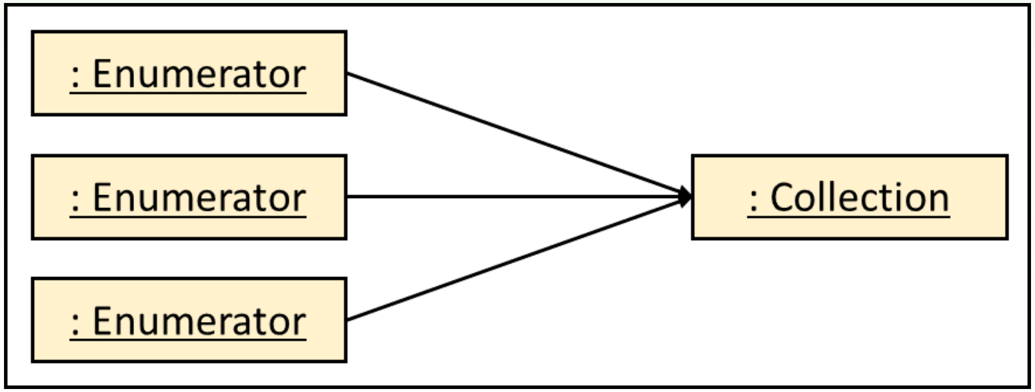
foreach (forall) ciklus: gyűjtemény felsorolása

```
foreach( var e in h )
{
    Process(e);
}
```

a felsorolható objektum `IEnumerable`

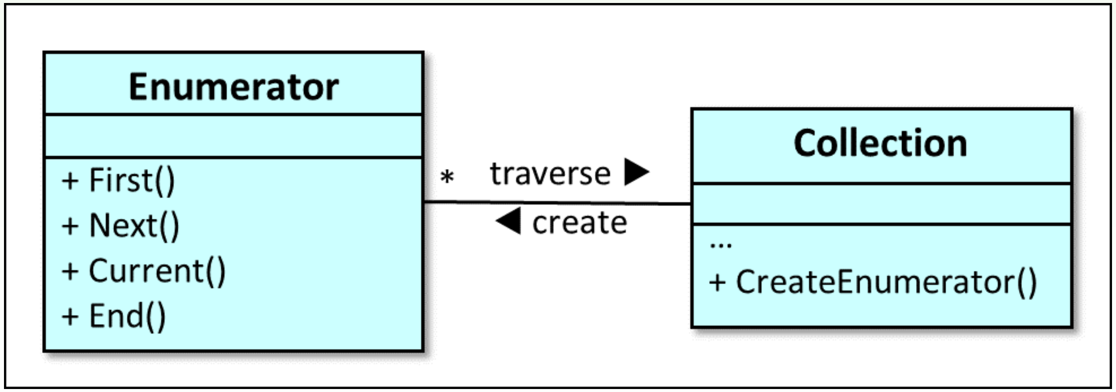
Felsoroló objektum

- Kézenfekvőnek tűnhet, hogy egy gyűjtemény elemeinek felsorolását a gyűjtemény metódusainak felhasználásával végezzük. Ez azonban szembe megy az egyszeres felelősség elvével.
- Ezért jobb, ha a felsorolást a gyűjteménytől elkülönülő ún. felsoroló objektum végzi, amely rendelkezik a First(), Next(), Current(), End() metódusokkal. Annak a felsoroló objektumnak a típusát, amely E-beli elemeket sorol fel, az enor(E) jelöli.
- Ennek előnyös következménye az, hogy ugyanazon a gyűjteményen egyszerre több felsorolás is folyhat egymástól függetlenül.

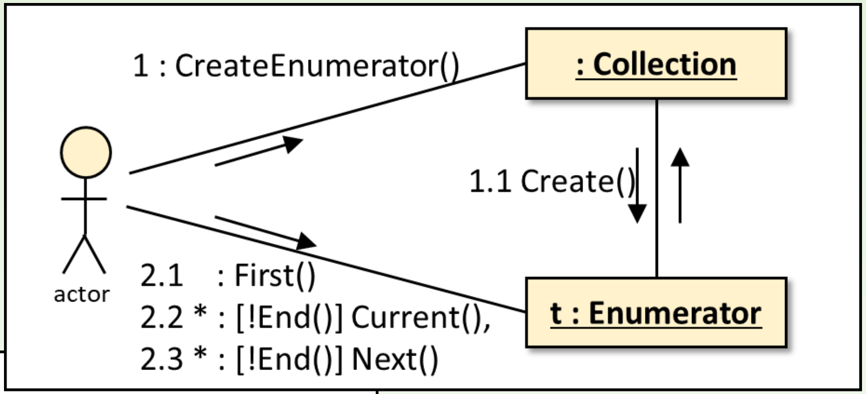


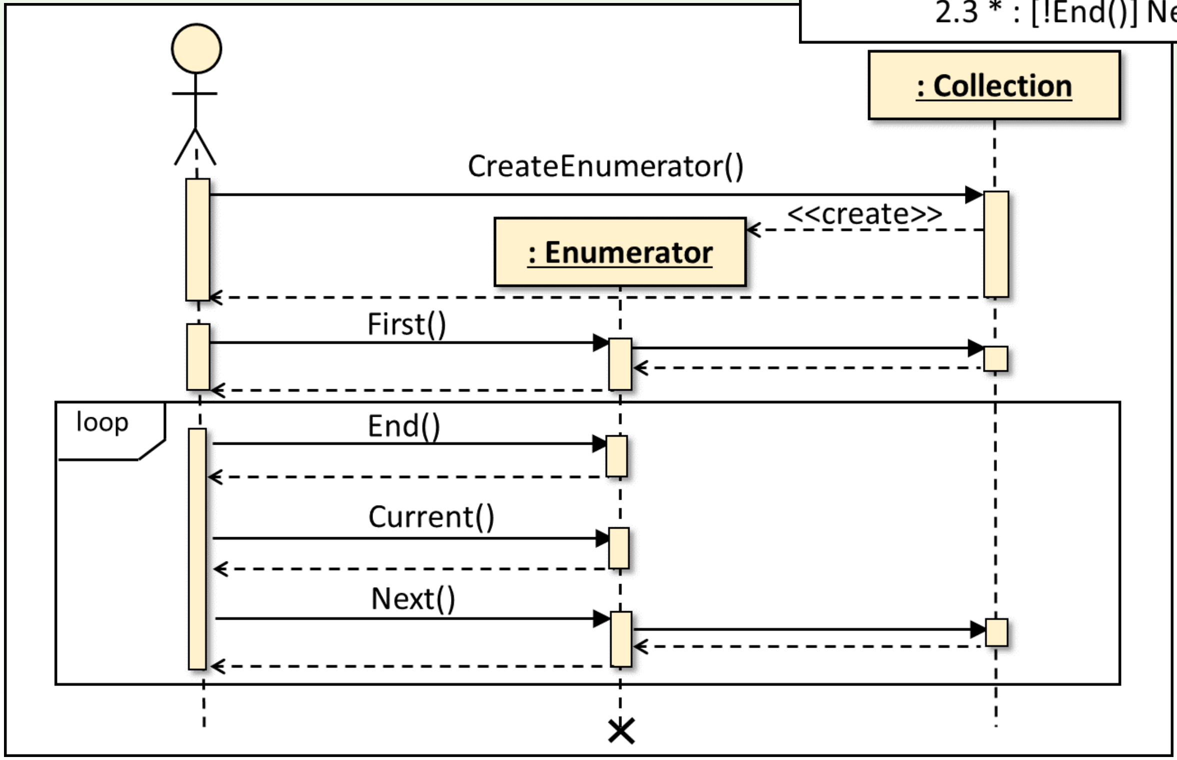
Felsorolók osztálya

- Egy felsoroló objektum reprezentációja mindig tartalmaz egy hivatkozást a felsorolni kívánt gyűjteményre;
- A felsoroló műveleteinek implementációja a gyűjtemény lekérdező (getter) metódusaira támaszkodik.
- A felsorolót a felsorolni kívánt gyűjtemény hozza létre (egy erre a célra szánt metódussal), így a gyűjtemény értesül arról, hogy őt felsorolják.



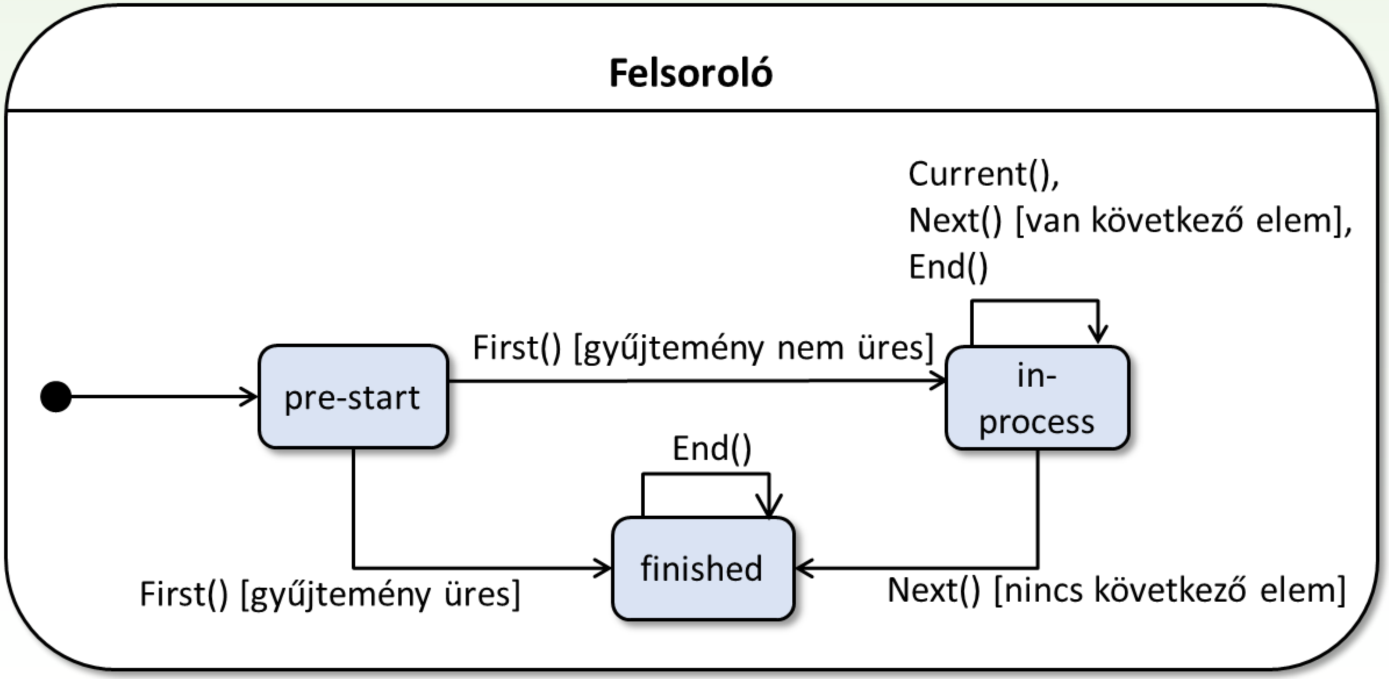
A felsoroló és a gyűjtemény közti kommunikáció





Felsorolás állapotai

- Ahhoz, hogy a felsorolás tevékenységeinek rákövetkezési feltételeit betartsuk, érdemes megkülönböztetni a felsorolás alábbi logikai állapotait: *indulásra kész, folyamatban van, befejeződött.*
- A felsorolás műveletei csak bizonyos állapotokban értelmesek, máskor a hatásuk nem definiált.



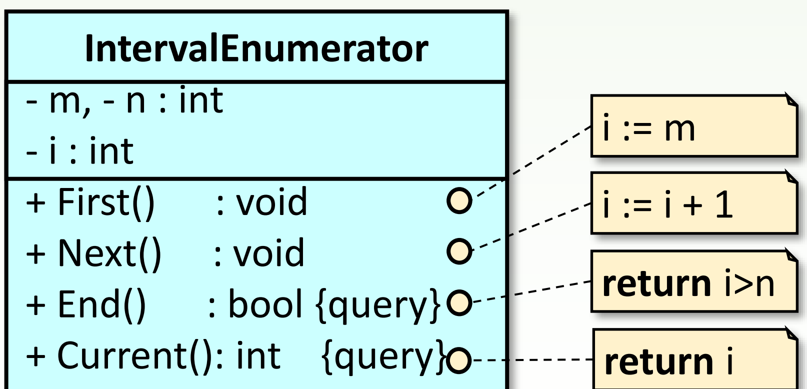
Intervallum klasszikus felsorolója

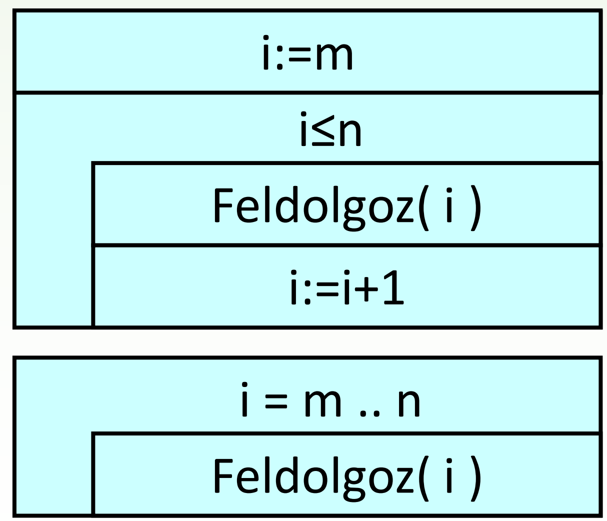
Egész számok intervallumába eső egész számok felsorolása elejétől a végéig.

enor($\mathbb{Z}$)

$\mathbb{Z}$*	First()	Next()	l:= End() l:$\mathbb{L}$	e:= Current() e:$\mathbb{Z}$
m , n : $\mathbb{Z}$ i : $\mathbb{Z}$	i:=m	i:=i+1	l:= i>n	e:=i

az intervallum határai, amelyek a gyűjteményt azonosítják





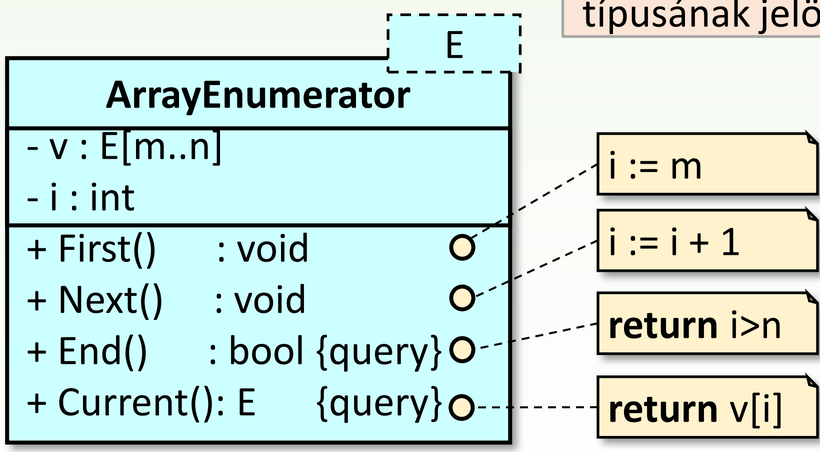
Vektor klasszikus felsorolója

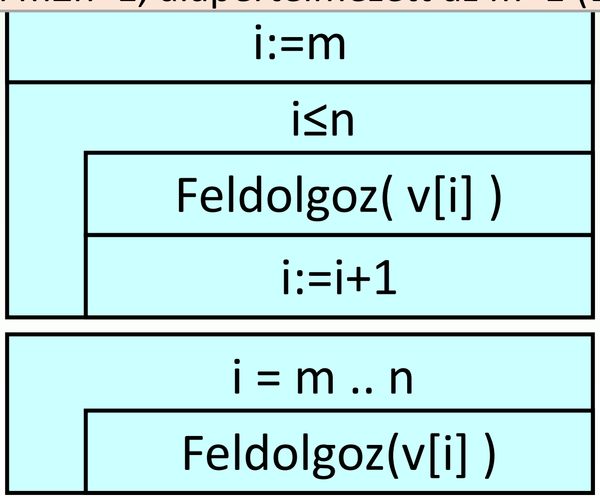
E-beli értékekből álló egy-dimenziós tömb elemeinek felsorolása elejétől a végéig

enor(E)

E*	First()	Next()	l:= End() $l:\mathbb{L}$	e:= Current() e:E
v : $E^{m..n}$ i : $\mathbb{Z}$	i:=m	i:=i+1	l:= i>n	e:=v[i]

egy-dimenziós, E típusú elemeket tartalmazó tömb (vektor) típusának jelölése, ahol m≤n+1, alapértelmezett az m=1 (E^n)





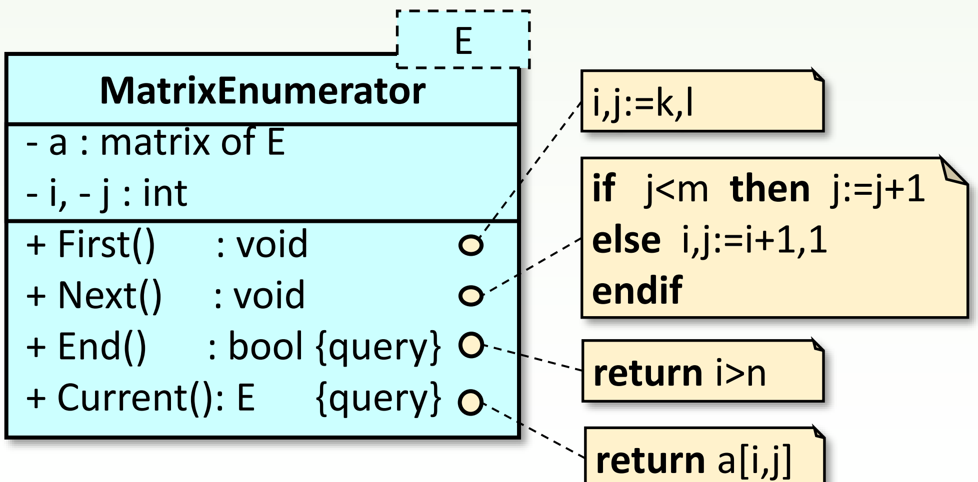
Mátrix sorfolytonos felsorolója

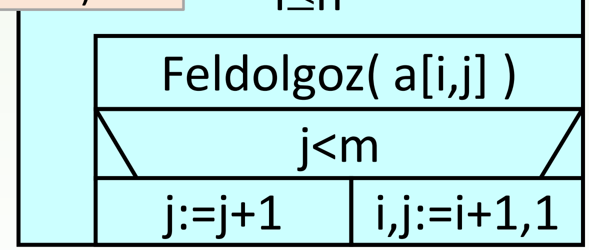
E-beli értékekből álló mátrix elemeinek felsorolása sorfolytonos sorrendben

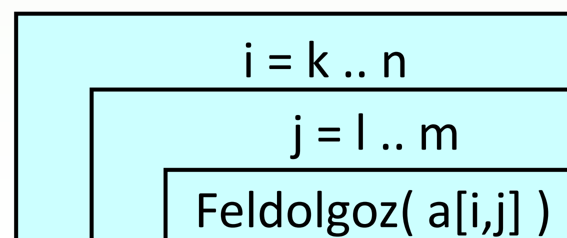
enor(E)

E*	First()	Next()	l:= End() l:$\mathbb{L}$	e:= Current() e:E
a : $E^{k..n \times l..m}$ i,j : $\mathbb{Z}$	i,j:=k,l	**if** j<m **then** j:=j+1 **else** i,j:=i+1,l	l:= i>n	e:= a[i,j]

E típusú elemeket tartalmazó mátrix típusának jelölése, ahol a sorokat k-tól n-ig, az oszlopokat l-től m-ig számozzuk, és k≤n+1, l≤m+1, alapértelmezett a k=1 és az l=1 ($E^{n \times m}$).







2. rész
Algoritmus minták felsorolóval

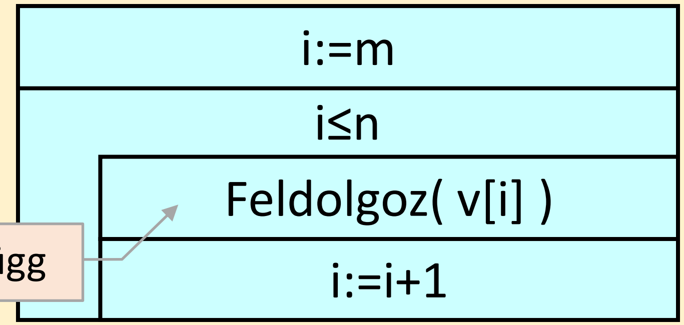
Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

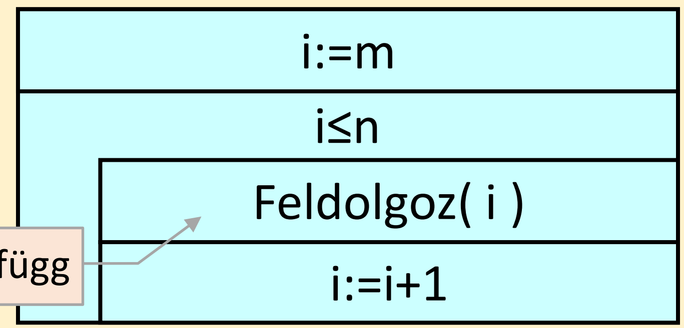
Algoritmus minták általánosítása

- Algoritmus-minták vektorra:
  - i:ℕ az $v:E^{m..n}$ tömb felsorolója
  - felt:E → 𝕃



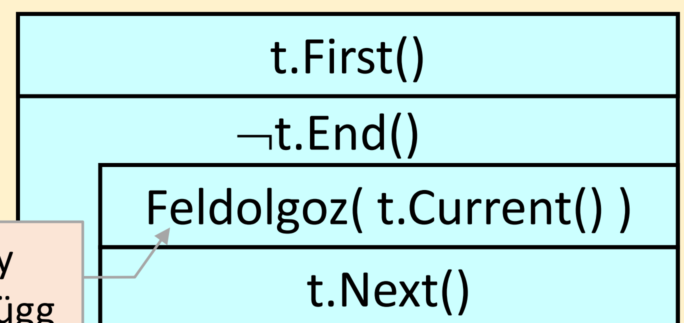
v[i]-től, esetleg felt(v[i])-től függ

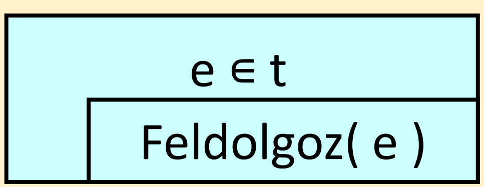
- Algoritmus-minták intervallumon értelmezett függvényre:
  - i:ℕ az [m .. n] felsorolója
  - f : [m .. n] → H, felt : [m .. n] → 𝕃



f(i)-től vagy felt(i)-től függ

- Algoritmus-minták felsorolóra:
  - t : enor(E) felsoroló
  - f : E → H, felt : E → 𝕃





f(t.Current())-től vagy felt(t.Current())-től függ

Összegzés

Összegezzük egy enor(E) típusú felsorolás elemeihez rendelt értékeket!

$f : E \to H$

H elemein értelmezett: $+ : H \times H \to H$

a $0 \in H$ *balneutrális elemmel*

$A = (\ t{:}enor(E),\ s{:}H\ )$

$Ef = (\ t = t'\ )$

$Uf = (\ s = \Sigma_{e \in t'} f(e)\ )$

A felsorolás végén t = <> , azaz nem marad t = t'

$s = \Sigma_{\ i=1..|t'|}\ f(t'[i])$

speciális eset: feltételes összegzés

$g : E \to H$, $felt : E \to \mathbb{L}$

$$\Sigma_{\substack{e \in t' \\ felt(e)}} g(e)$$

$$\text{azaz } f(e) = \begin{cases} g(e) & \text{ha } felt(e) \\ 0 & \text{különben} \end{cases}$$

(feltéve, hogy 0 jobb neutrális elem is)

s := 0

t.First()

¬t.End()

s := s + f(t.Current())

t.Next()

Számlálás

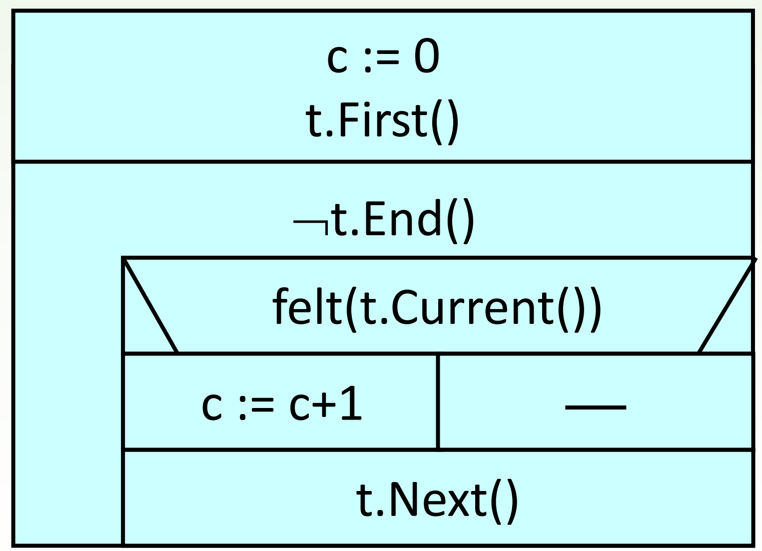
Számoljuk meg egy enor(E) típusú felsorolás adott tulajdonságú elemeit!

felt : $E \rightarrow \mathbb{L}$

$A \;= (\; t:\text{enor}(E),\; c:\mathbb{N} \;)$

$Ef = (\; t = t' \;)$

$Uf = (\; c = \sum_{\substack{e \in t' \\ \text{felt}(e)}} 1 \;)$



A számlálás egy speciális összegzés:

$$\sum_{e \in t'} f(e) \quad \text{azaz } f(e) = \begin{cases} 1 & \text{ha felt}(e) \\ 0 & \text{különben} \end{cases}$$

Maximum kiválasztás

Adjuk meg egy enor(E) típusú felsorolás adott szempont szerinti egyik legnagyobb elemét és annak értékét!

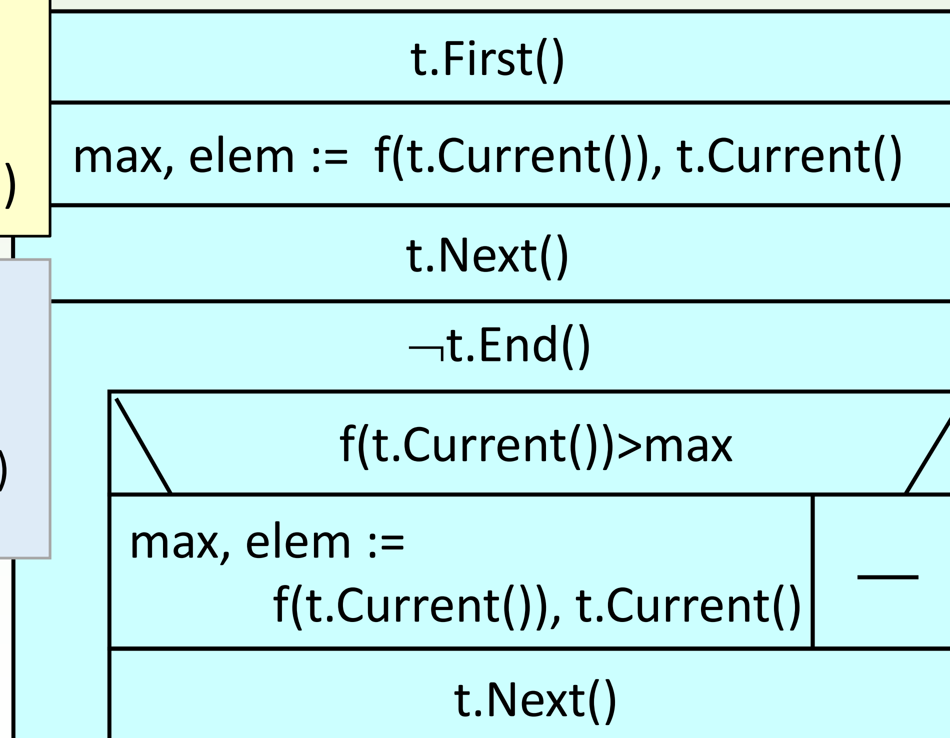
$f : E \to H$
H *elemei rendezhetőek*

$A = (\ t{:}enor(E),\ elem{:}E,\ max{:}H\ )$

$Ef = (\ t = t' \wedge |t|>0\ )$

$Uf = (\ (max,\ elem) = \mathbf{MAX}_{e \in t'}\ f(e)\ )$

$max = \mathbf{MAX}_{i=1..|t'|}\ f(t'_i)$
$\wedge\ elem \in t' \wedge max = f(elem)$
vagy
$ind \in [1..|t'|] \wedge \forall i \in [1..|t'|] : f(t'_i) \leq f(t'_{ind})$
$\wedge\ elem = t'_{ind} \wedge\ max = f(elem\ )$



- MAX (>) *helyett lehet* MIN (<)
- elem *elhagyható,* max *nem*

Kiválasztás (biztosan talál)

Keressük meg egy enor(E) típusú felsorolás adott tulajdonságú első elemét, ha tudjuk, hogy van ilyen!

felt : E → $\mathbb{L}$

A = (t:enor(E), elem:E)

Ef = (t = t' ∧ ∃e∈t : felt(e))

Uf = ((elem, t) = **SELECT**$_{e\in t'}$ felt(e))

Megadja a *t'* felsorolás első olyan elemét, amelyre a feltétel teljesül (ez lesz az *elem* értéke). A keresés végén a *t* felsoroló *„folyamatban van"* státuszú lesz.

$i\geq 1 \wedge felt(t'_i) \wedge \forall k\in[1..i-1] : \neg felt(t'_k) \wedge elem = t'_i \wedge t = <t'_{i+1}, \dots , t'_{|t'|}>$

```
foreach( var e in t )
{
   if ( felt(e) ){ elem = e; break; }
}
```

t.First()
¬felt(t.Current())
t.Next()
elem := t.Current()

Lineáris keresés (találat nem biztos)

Keressük meg egy enor(E) típusú felsorolás adott tulajdonságú első elemét!

$felt : E \to \mathbb{L}$

$A = (t:enor(E),\ l:\mathbb{L},\ elem:E)$

$Ef = (t = t')$

$Uf = ((l, elem, t) = \mathbf{SEARCH}_{e \in t'}\ felt(e))$

$(l = \forall i \in [1..|t'|] : felt(t'_i)) \wedge$

$(l \longrightarrow i \in [1..|t'|] \wedge felt(t'_i) \wedge \forall k \in [1..i-1] : \neg felt(t'_k)$

$\wedge\ elem = t'_i \wedge t = <t'_{i+1}, \dots, t'_{|t'|}>)$

Ha van a *t'* felsorolásnak olyan eleme, amelyre teljesül-e a feltétel, akkor *l* igaz, az *elem* a legelső feltételt kielégítő elem lesz, és lehet, hogy *t „folyamatban van”* státuszban marad. Egyébként *l* hamis, *elem* nem definiált, és a *t „befejeződött”* státuszú lesz.

speciálisan eldöntéshez is használhatjuk:

$l = \mathbf{SEARCH}_{e \in t'}\ felt(e)$ vagy $l = \exists_{e \in t'}\ felt(e)$

l := hamis; t.First()

$\neg l \wedge \neg t.End()$

felt(t.Current())

l, elem := igaz, t.Current()	t.Next()

```
l = false;
foreach( var e in t )
{
   if ( l = felt(e) ) { elem = e; break; }
}
```

```
E Search()
{
   foreach( var e in t ) if ( felt(e) ) return e;
   return null;
}
```

Optimista lineáris keresés

Vizsgáljuk meg, hogy egy enor(E) típusú felsorolás minden elemére igaz-e egy adott tulajdonság. Ha nem, adjuk meg az első olyan elemet, amelyikre nem!

felt : E → $\mathbb{L}$

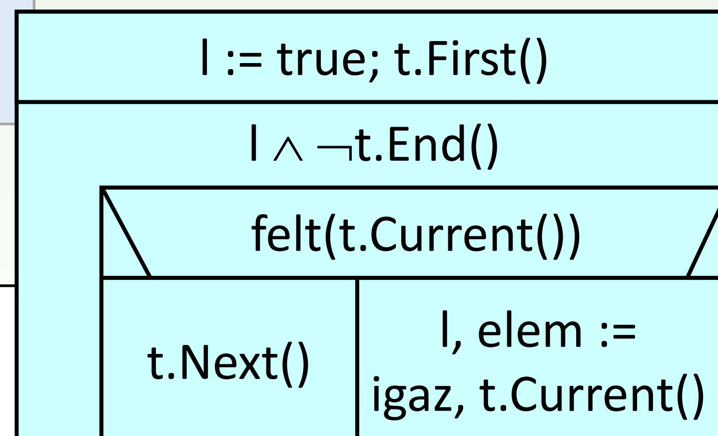
A = (t:enor(E), l:$\mathbb{L}$, elem:E)

Ef = (t = t')

Uf = ((l, elem, t) = ∀**SEARCH**$_{e\in t'}$ felt(e))

Ha a *t'* felsorolás minden elemére teljesül a feltétel, akkor *l* igaz, *elem* nem definiált, és *t „befejeződött”* státuszú lesz. Egyébként *l* hamis, az *elem* a legelső feltételt ki nem elégítő elem lesz, és lehet, hogy a *t* státusza *„folyamatban van”* marad.

$(l = \exists i\in[1..|t'|] : felt(t'_i)) \wedge$
$(\neg l \longrightarrow i\in[1..|t'|] \wedge \neg felt(t'_i) \wedge \forall k\in[1..i-1] : felt(t'_k)$
$\wedge\ elem = t'_i$ és $t = <t'_{i+1}, \dots, t'_{|t'|}>)$



főleg eldöntésre használjuk:
l = ∀**SEARCH**$_{e\in t'}$ felt(e) vagy l = $\forall_{e\in t'}$ felt(e)

```
l = true;
foreach( var e in h )
{
   if ( l = felt(e) ); else { elem = e; break; }
}
```

```
bool Search(ref E elem)
{
   foreach( var e in t ) if ( felt(e) ); else { elem = e; return false; }
   return true;
}
```

Feltételes maximum keresés

Keressük egy enor(E) típusú felsorolás adott tulajdonságú elemei között egy adott szempont szerinti egyik legnagyobbat és annak értékét!

$f : E \to H$
$felt : E \to \mathbb{L}$
H *halmaz elemei rendezhetőek*

$A = (\ t{:}enor(E),\ l{:}\mathbb{L},\ elem{:}E,\ max{:}H\ )$

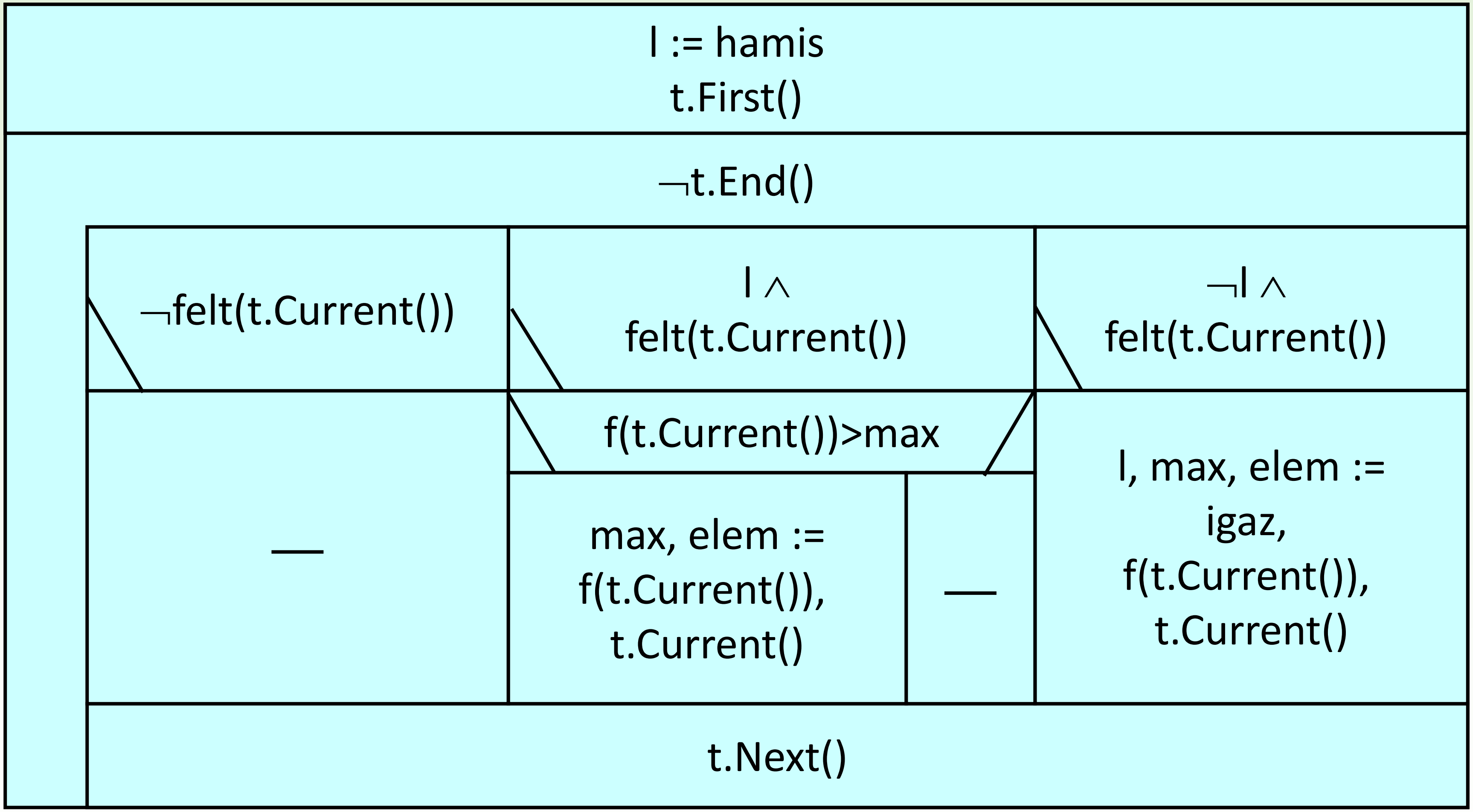
$Ef = (\ t = t'\ )$

$Uf = (\ (l, max, elem) = \mathbf{MAX}_{\substack{e \in t' \\ felt(e)}}\ f(e)\ )$

$(\ l = \exists i \in [1..|t'|] : felt(t'_i)\ ) \wedge$
$(\ l \longrightarrow i \in [1..|t'|] \wedge felt(t'_i) \wedge$
$\wedge\ \forall k \in [1..t'|] : (\ felt(t'_k) \longrightarrow f(t'_k) \leq f(t'_i)\ )$
$\wedge\ elem = t'_i \wedge max = f(elem)\ )$

Ha van a *t'* felsorolásnak olyan eleme, amelyre teljesül-e a feltétel, akkor *l* igaz, az *elem* a *t'* olyan feltételt kielégítő eleme, amelynek *f* értéke *max*, ami nagyobb vagy egyenlő a *t'* bármelyik olyan elemének *f* értékénél, amely kielégíti a feltételt. Egyébként az *l* hamis, az *elem* és *max* nem definiált.

Feltételes maximum keresés



- MAX (>) *helyett lehet* MIN (<)
- elem *elhagyható,* max *nem*

3. rész
Visszavezetés módszere
és a tesztelés

Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Visszavezetés

1. Megsejtjük a feladatot megoldó algoritmus-mintát (feladat-algoritmus párt).
2. Specifikáljuk a feladatot az algoritmus-mintáéhoz hasonló utófeltétellel.
3. Rögzítjük a feladat és az algoritmus-minta közötti eltéréseket:
   - a *felsoroló* és a *felsorolt elemek* konkrét típusát
   - a *függvények* (f : E→H, felt : E→$\mathbb{L}$) aktuális megfelelőit
   - a H halmaz *műveletét*, ha van ilyen
     - (H, >) helyett például ($\mathbb{Z}$, >) vagy ($\mathbb{Z}$, <)
     - (H, +) helyett például ($\mathbb{Z}$, +) vagy ($\mathbb{R}$, *) vagy ($\mathbb{L}$, $\wedge$) stb.
   - a *változók átnevezéseit*
4. Beleírjuk az algoritmus-minta algoritmusába a fenti különbségeket, és így megkapjuk a feladatot megoldó algoritmust.

Programozási tétel:
Ha egy feladat és egy algoritmus-minta feladata megfeleltethetők egymásnak, akkor az algoritmus-minta algoritmusának ezen megfeleltetés szerint átalakított változata megoldja a kitűzött feladatot.

Tesztelési stratégiák

- Fekete doboz: a feladat (specifikációja) alapján felírt tesztesetek.
  - az előfeltételt kielégítő (érvényes), illetve azt megszegő (érvénytelen) tesztadatokkal felírt tesztesetek.
  - az utófeltétel alapján (?) generált tesztesetek vizsgálata
  - …

- Fehér doboz: a kód alapján felírt tesztesetek.
  - algoritmus minden utasításának kipróbálása
  - algoritmus minden vezérlési csomópontjának (elágazás, ciklus) kipróbálása
  - …

- Szürke doboz: végrehajtható specifikáció által előrevetített algoritmus működését ellenőrző tesztesetek.
  - Ha a végrehajtható specifikáció ráadásul egy algoritmus-mintából származik, akkor az algoritmus-minta szokásos teszteseteit kell megvizsgálni.

Algoritmus-minták tesztesetei

- Felsoroló szerint (mindegyik algoritmus-minta esetén)
  - eltérő *hosszúságú* felsorolások: nulla, egy illetve hosszabb felsorolásokra is kipróbájuk az algoritmust
  - Feldolgozza-e az algoritmus a felsorolás *első* ill. *utolsó* elemét

Számlálás: eleje és a vége adott tulajdonságú
Keresés: eleje vagy csak a vége adott tulajdonságú
Maximum kiválasztás: eleje vagy a vége legnagyobb

- Funkció szerint
  - *összegzés*: neutrális elem vizsgálata, felsorolás hosszának skálázása
  - *keresés*, *számlálás*: van vagy nincs keresett tulajdonságú elem
  - *max. kiv.*: egyetlen, illetve több azonos maximális érték
  - *felt. max. ker.*: - van vagy nincs keresett tulajdonságú elem
    - feltételt kielégítő egyetlen, illetve több azonos maximális érték
    - a legnagyobb értékű elem nem elégíti ki a feltételt
- A *felt*(*e*) és *f*(*e*) kifejezéseiben használt műveletek sajátosságai, értelmezési tartományuk.

Feladat

A Föld felszín egy vonalán adott pontokon megmértük a felszín tengerszint feletti magasságát, és az adatokat egy tömbben tároltuk el. Hol található és milyen magas a felszín legmagasabb horpadása?

$A\ = (\ x : \mathbb{R}^n\ ,\ l : \mathbb{L},\ max : \mathbb{R},\ \ ind : \mathbb{N})$

$Ef\ = (\ x = x_0\ )$

$$Uf = (\ Ef \wedge (\ l,\ max,\ ind\ ) = \mathbf{MAX}_{\substack{i=2..n-1 \\ x[i-1]>x[i]<x[i+1]}} x[i]\ )$$

<u>Feltételes maximumkeresés</u>:

$e \in t$:enor(E) ~	$i \in [2\ ..\ n-1]$
f(e) ~	x[i]
felt(e) ~	x[i-1]>x[i]<x[i+1]
H, > ~	$\mathbb{R}$, >

l := hamis

i = 2 .. n-1

$\neg$(x[i-1]>x[i]<x[i+1]) | l $\wedge$ x[i-1]>x[i]<x[i+1] | $\neg$l $\wedge$ x[i-1]>x[i]<x[i+1]

—

x[i]>max

max, ind := x[i], i | —

l, max, ind := igaz, x[i], i

Főprogram

```
static void Main()
{
   List<double> x = FillInFromFile("input.txt");

   if (MaxSearch(in x, out double max, out int ind))
   {
      Console.WriteLine($"Height of the highest depression: "
                            { max } meter on the place { ind } ");
   }
   else
   {
      Console.WriteLine("There is no depression.");
   }
}
```

Feltételes maximumkeresés kódja

```
static bool MaxSearch(in List<double> x, out double max, out int ind)
{
   bool l = false; max = 0.0; ind = 0;
   for ( int i = 1; i < x.Count; ++i ) {
      if (!(x[i-1] > x[i] && x[i] <x [i+1])) continue;
      if (l)
      {
         if ( max < x[i] )
         {
            max = x[i]; ind = i;
         }
      }
      else
      {
         l = true; max = x[i]; ind = i;
      }
   }
   return l;
}
```

ciklusmag végére ugrik a vezérlés

Tesztelés

Feltételes maximum keresés tesztesetei			
felsoroló szerint	hossza: 0	x = < > , < 1.0, 2.0>	→ l = hamis
	hossza: 1	x = < 2.1, 1.0, 2.4 >	→ l = igaz, max = 1.0, ind = 2
	hossza: több	x = < 1.0, 2.0, 1.5, 4.0, 2.0 >	→ l = igaz, max = 1.5, ind = 3
	eleje	x = < 3.0, 2.5, 3.0, 2.0, 3.0 >	→ l = igaz, max = 2.5, ind = 2
	vége	x = < 3.0, 2.0, 3.0, 2.5, 3.0 >	→ l = igaz, max = 2.5, ind = 4
tétel szerint	nincs	x = < 1.0, 2.0, 3.0, 4.0, 5.0 >	→ l = hamis
	van	x = < 1.0, 2.0, 1.0, 4.0, 2.0 >	→ l = igaz, max = 1.0, ind = 3
	egy maximum	x = < 3.0, 1.5, 3.0, 2.5, 3.0 >	→ l = igaz, max = 2.5, ind = 4
	több maximum	x = < 3.0, 2.0, 3.0, 2.0, 3.0 >	→ l = igaz, max = 2.0, ind = 2,4

Tesztkörnyezet

- Olyan tesztelő projektet (TestDepression) készítünk C#-ban, amelyik a feladatot megoldó projekt MaxSearch eljárását ellenőrzi (unit test).

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Depression;

namespace TestDepression
{
  [TestClass]
  public class CondMaxSearchTests
  {
    [TestMethod]
    public void Test_NullLengthEnumerator() { ... }

    [TestMethod]
    public void Test_OneElement() { ... }

    [TestMethod]
    public void Test_NoDepression() { ... }

    [TestMethod]
    public void Test_SeveralMaxs () { ... }
  }
}
```

CondMaxSearchTestsTest.cs

Egység tesztek

```
[TestMethod]
public void Test_NullLengthEnumerator()
{
  List<double> x = new ();
  bool l = Program.MaxSearch(in x, out double max, out int ind);
  Assert.AreEqual(l, false);

  x.Add(1.0); x.Add(2.0);
  l = Program.MaxSearch(in x, out max, out ind);
  Assert.AreEqual(l, false);
}
[TestMethod]
public void Test_OneElement()
{
  List<double> x = new () { 2.1, 1.0, 2.4 };
  bool l = Program.MaxSearch(in x, out double max, out int ind);
  Assert.AreEqual(l, true);
  Assert.AreEqual(max, 2.0);
  Assert.IsTrue(ind == 2);
}
...
```

CondMaxSearchTestsTest.cs

Egység tesztek

```
...
[TestMethod]
public void Test_NoDepression()
{
  List<double> x = new () { 1.0, 2.0, 3.0, 4.0, 2.0 };
  bool l = Program.MaxSearch(in x, out double max, out int ind);
  Assert.AreEqual(l, false);
}
[TestMethod]
public void Test_SeveralMaxs()
{
  List<double> x = new () { 3.0, 2.0, 3.0, 2.0, 3.0 };
  bool l = Program.MaxSearch(in x, out double max, out int ind);
  Assert.AreEqual(l, true);
  Assert.AreEqual(max, 2.0);
  Assert.IsTrue(ind == 1 || ind==3);
}
...
```

CondMaxSearchTestsTest.cs