Felsoroló

Gregorics Tibor

gt@inf.elte.hu

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

Felsoroló
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: elemeket lehet benne elhelyezni, visszakeresni, eltávolítani; és ehhez a gyűjtemény biztosít metódusokat.
  Például: halmaz, zsák, sorozat (amely speciálisan lehet verem vagy sor, ha korlátozzuk a műveleteit), tömb, fa, gráf, vagy akár rekord.
- Egy gyűjtemény lehet virtuális is, amikor nem kell explicit módon eltárolni az elemeit. Például egész számok egy intervallumát elég az intervallum végpontjaival megadni, vagy egy természetes szám prím-osztóinak visszakereséséhez elég az adott természetes számot ismerni.
- Egy gyűjtemény feldolgozásán a benne levő elemek feldolgozását értjük.
  - Adjuk meg egy halmaz valamilyen szempont szerinti legnagyobb elemét.
  - Számoljuk meg a negatív számokat 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.
- Az E típusú elemek felsorolására, bejárására (jele: enor(E)) úgy tekintünk, mint a felsorolandó gyűjtemény elemeiből képzett olyan 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) E típusú aktuális elemét
  - l := End() (l:𝕃) : jelzi, hogy a sorozat (felsorolás) végére értünk-e
- A felsorolás műveletei csak bizonyos helyzetekben (állapotokban) értelmesek, máskor a hatásuk nem definiált.
  - A felsorolásnak három állapotát különböztethetjük meg: indulásra kész, folyamatban van, befejeződött
  - First() csak az „indulásra kész” állapotban értelmezett
  - Next() és Current() csak a „folyamatban van” állapotban értelmezett
  - End() a „folyamatban van” és a „befejeződött” állapotban értelmezett

Felsorolás algoritmusa

- Célszerű a gyűjtemények felsorolását olyan algoritmusba ágyazni, amely garantálja, hogy a felsorolás műveletei mindig csak a számukra „értelmes” állapotban kerüljenek végrehajtásra.

gyűjteményt felsoroló műveletek

```
First()
¬End()
    Feldolgoz( Current() )
    Next()
```

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

Itt h egy felsorolható (`IEnumerable`) objektum, amelynek tartalmát a foreach ciklus nem változtathatja meg. Ha azonban h-ban objektumok, pontosabban azok hivatkozásai vannak, akkor azok adattagjai módosíthatók.

felsorolandó gyűjtemény

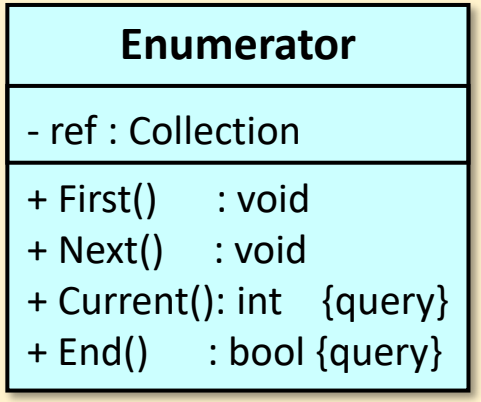
```
e in h
    Feldolgoz( e )
```

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

a felsorolt elemek típusa automatikusan kideríthető

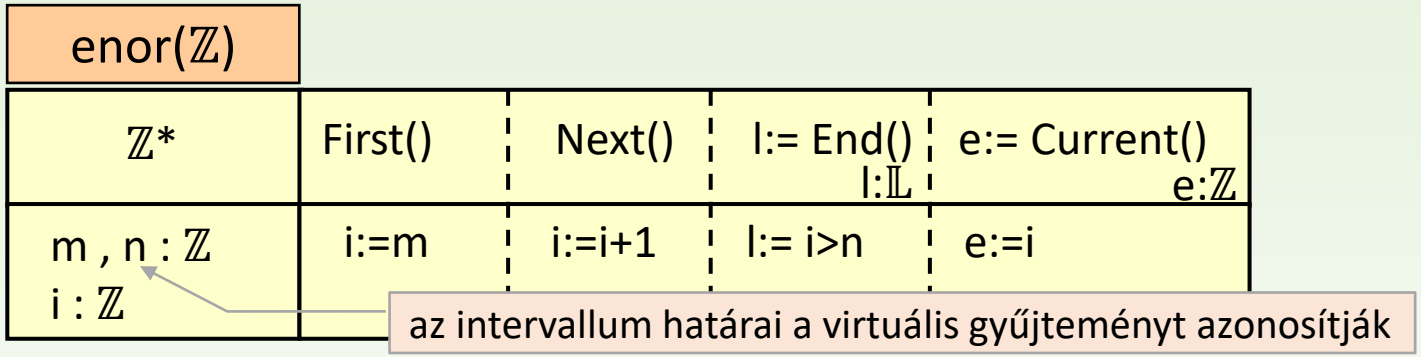
Felsoroló objektum

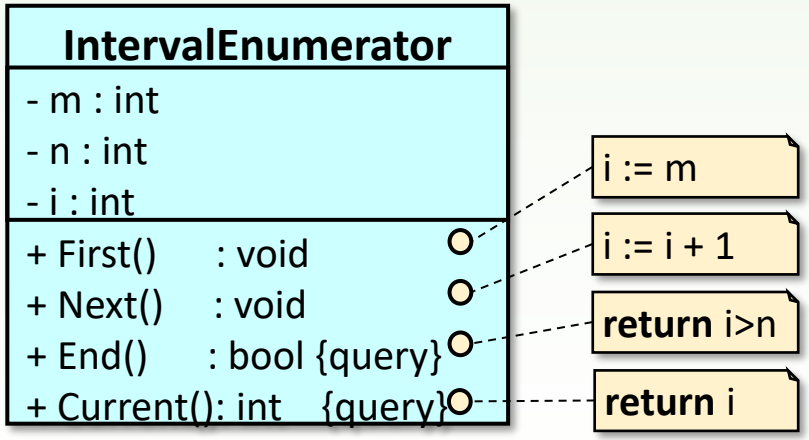
- Ha egy gyűjtemény elemeinek felsorolását közvetlenül a gyűjtemény metódusaival oldjuk meg, akkor ugyanarra a gyűjteményre egyidejűleg több felsorolást nem tudunk indítani.
- Célszerű, ha a felsorolást a gyűjteménytől elkülönülő ún. felsoroló objektum végzi (ennek típusát is az enor(E) jelöli majd), mert ilyen objektumból egyszerre többet is létrehozhatunk.
- Egy felsoroló objektummal szemben az alábbiakat követeljük meg:
  - rendelkezzen felsorolást végző First(), Next(), Current(), End() műveletekkel
  - érje el a felsorolandó gyűjteményt, és támaszkodjon annak metódusaira
  - példányosítását a gyűjtemény egy metódusa végezze

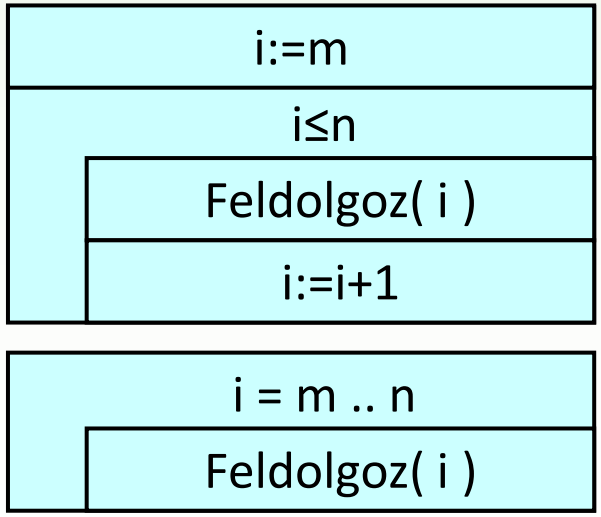


Intervallum klasszikus felsorolója

Egész számok intervallumába eső egész számok felsorolása az intervallum alsó határától a felső határáig.

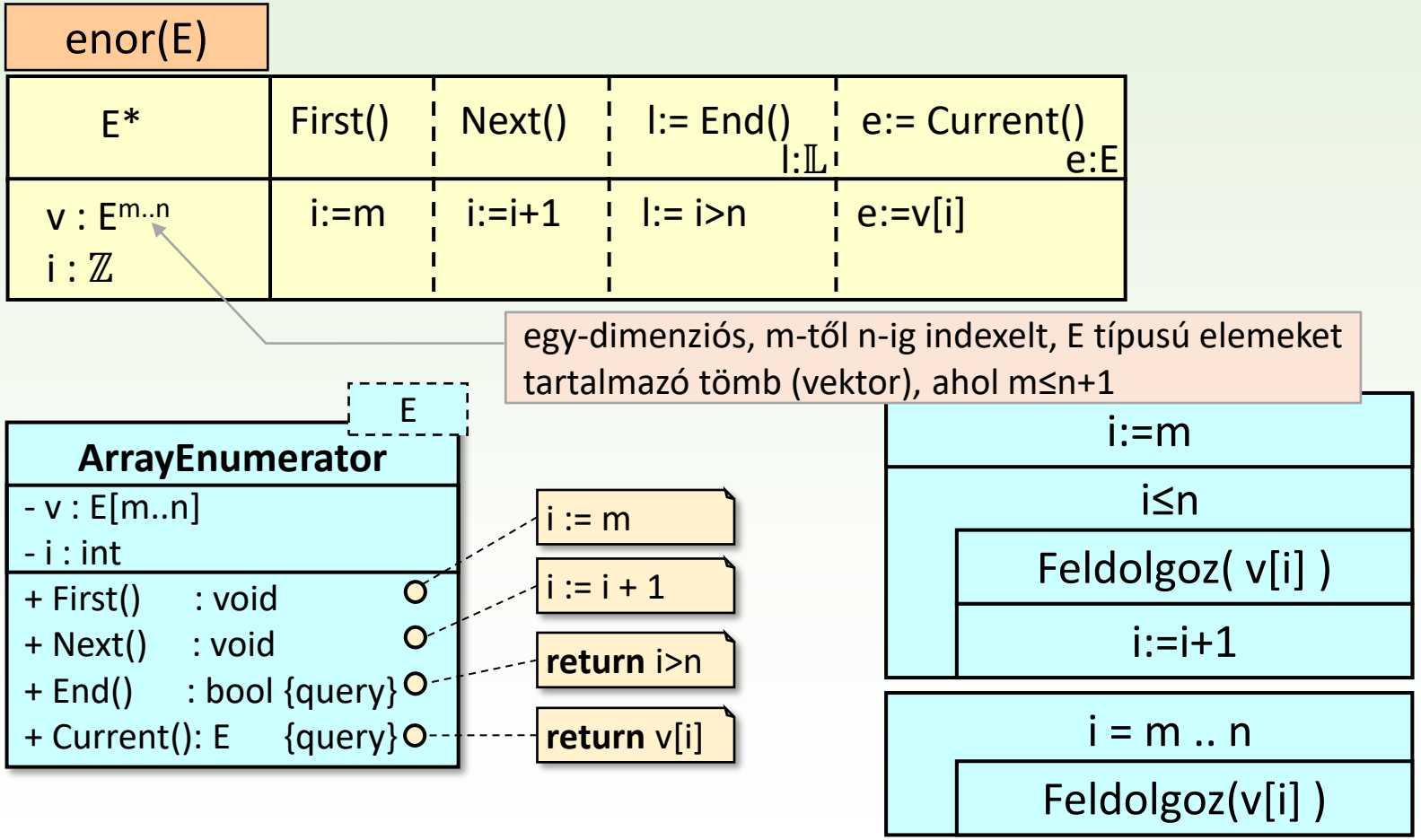






Vektor klasszikus felsorolója

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



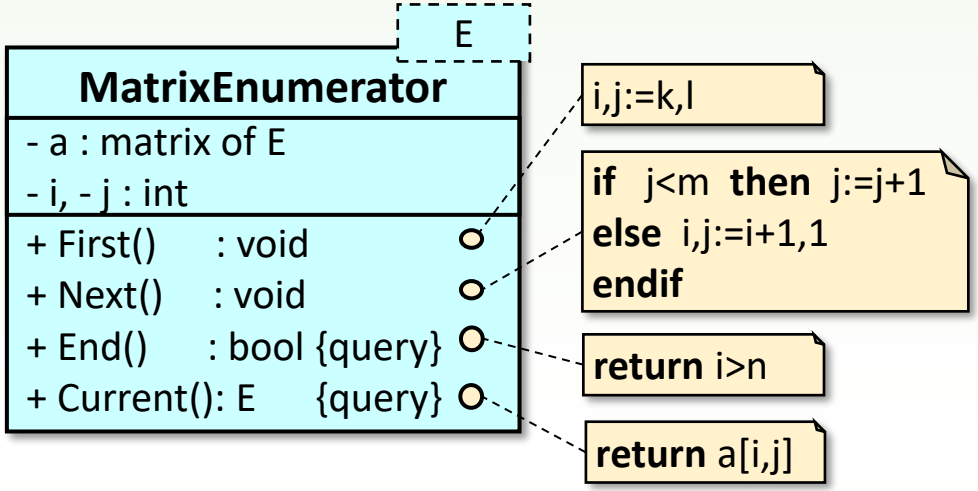
Mátrix sorfolytonos felsorolója

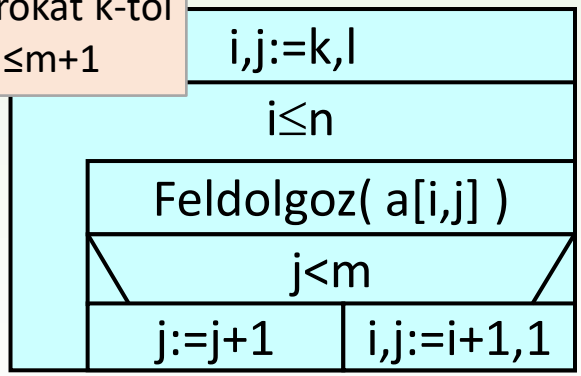
E-beli értékekből álló mátrix elemeinek felsorolása sorfolytonos sorrendben

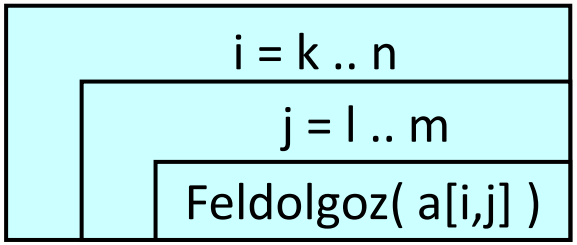
enor(E)

E*	First()	Next()	l:= End() l:ℕ	e:= Current() e:E
a : $E^{k..n \times l..m}$ i,j : 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-beli elemeket tartalmazó mátrix típusa, ahol a sorokat k-tól n-ig, az oszlopokat l-től m-ig számozzuk, és k≤n+1, l≤m+1

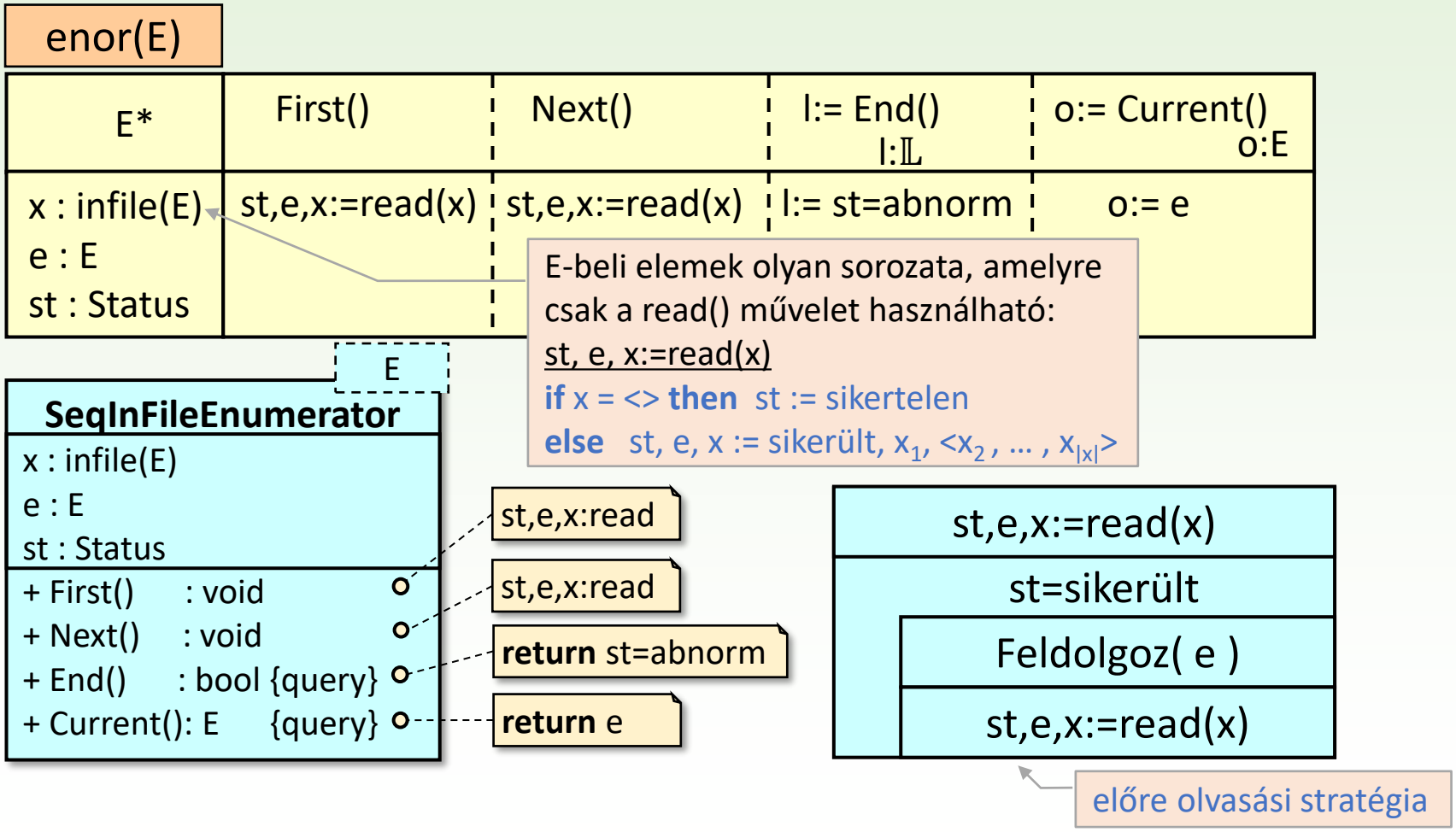






Szekvenciális inputfájl felsorolója

E-beli értékek sorozatának sorban egymás után történő felsorolása úgy, hogy a sorozat végig olvasása során a sorozat logikai értelemben elfogy



Felsoroló
2. rész
Algoritmus minták felsorolóval

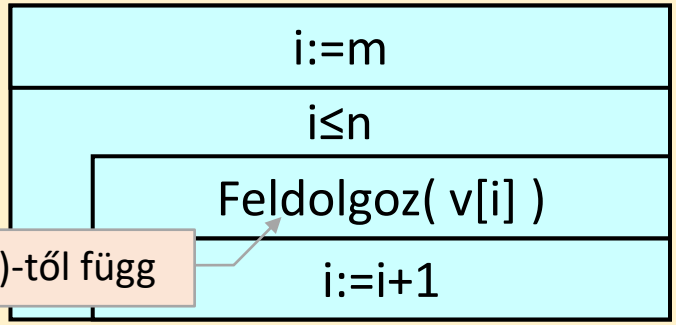
Gregorics Tibor

gt@inf.elte.hu

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

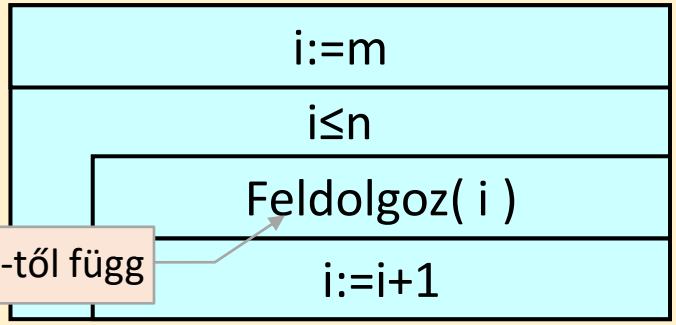
Algoritmus minták általánosítása

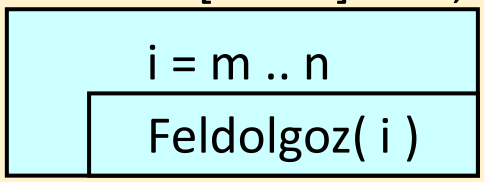
- Algoritmus-minták vektorra:
  - i:ℕ az v:$E^{m..n}$ tömb felsorolója
  - felt:E → 𝕃



v[i]-től és/vagy felt(v[i])-től függ

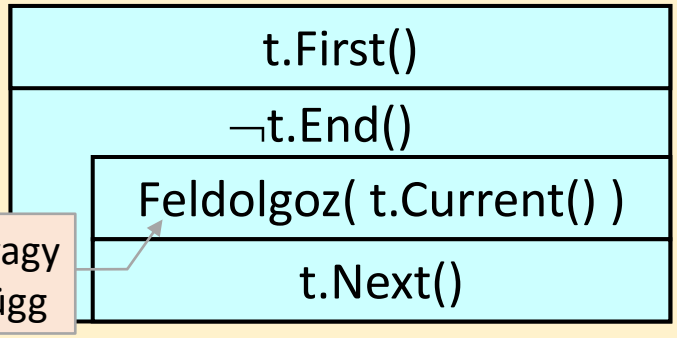
- Algoritmus-minták intervallumon értelmezett függvényre:
  - i:ℕ az [m .. n] felsorolója
  - f : [m .. n] → H, felt : [m .. n] → 𝕃

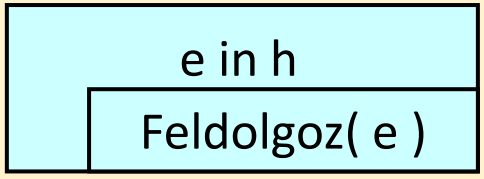




f(i)-től és/vagy felt(i)-től függ

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





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

Összegzés

lehet számok szorzása,
logikai értékek összeéselése,
sorozatok összefűzése

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

$f : E \to H$

H elemein értelmezett összegzés:

$+ : H \times H \to H \; 0 \in H$ *balneutrális elemmel*

A = (t:enor(E), s:H)

Ef = (t = t')

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

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

```
s := 0
t.First()
¬t.End()
    s := s + f(t.Current())
    t.Next()
```

```
s = 0;
foreach( var e in t )
{
    s += f(e);
}
```

```
s = t.Sum(e => f(e));
```

speciális eset: feltételes összegzés

$g : E \to H$, $felt : E \to \mathbb{L}$ $\qquad \Sigma_{\substack{e \in t' \\ felt(e)}} g(e)$

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

(feltéve, hogy 0 jobb neutrális elem is)

```
s = 0;
foreach( var e in t )
{
    if(felt(e)) s += g(e);
}
```

```
s = t.Where(e => felt(e))
     .Sum(e => g(e));
```

?

Számlálás

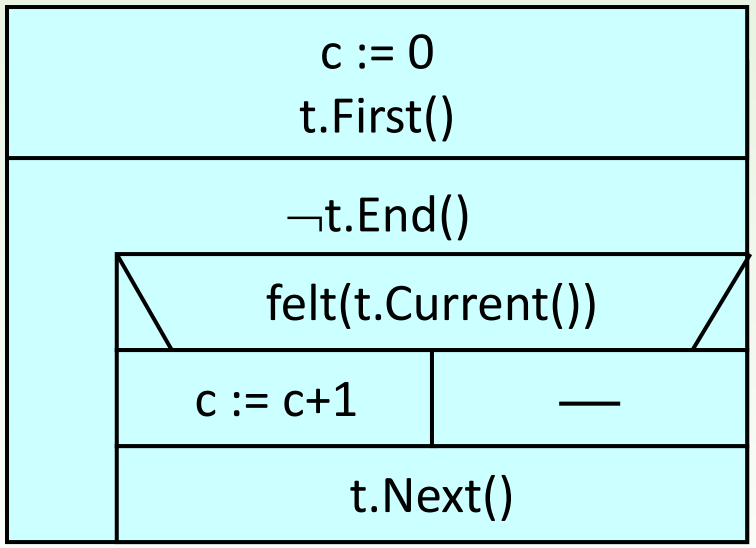
Számoljuk meg egy enor(E) típusú felsorolás adott tulajdonságú elemeit!

felt : E → 𝕃

A = (t:enor(E), c:ℕ)

Ef = (t = t')

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



A számlálás egy speciális összegzés:

$\Sigma_{e \in t'}$ f(e) azaz f(e)= { 1 ha felt(e) ; 0 különben }

```
c = 0;
foreach( var e in t )
{
    if(felt(e)) ++c;
}
```

```
c = t.Count(e => felt(e));
```

Kiválasztás (biztosan talál)

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

felt : E → 𝕃

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

Ef = (t = t' ∧ ∃e∈t : felt(e))

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

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

t.First()

¬felt(t.Current())

t.Next()

elem := t.Current()

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

```
E elem = t.Find(e => felt(e))!;
```

az eredmény biztosan nem lesz null

Lineáris keresés (találat nem biztos)

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

felt : E → 𝕃

A = (t:enor(E), l:𝕃, elem:E)

Ef = (t = t')

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

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

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

l := hamis; t.First()

¬l ∧ ¬t.End()

felt(t.Current())

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

t.Next()

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

lehet null érték is

```
E? elem = t.Find(e => felt(e));
```

speciálisan eldöntéshez is használhatjuk:
l = **SEARCH**$_{e \in t'}$ felt(e) vagy l = $\exists_{e \in t'}$ felt(e)

```
l = t.Any(e => felt(e));
```

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!

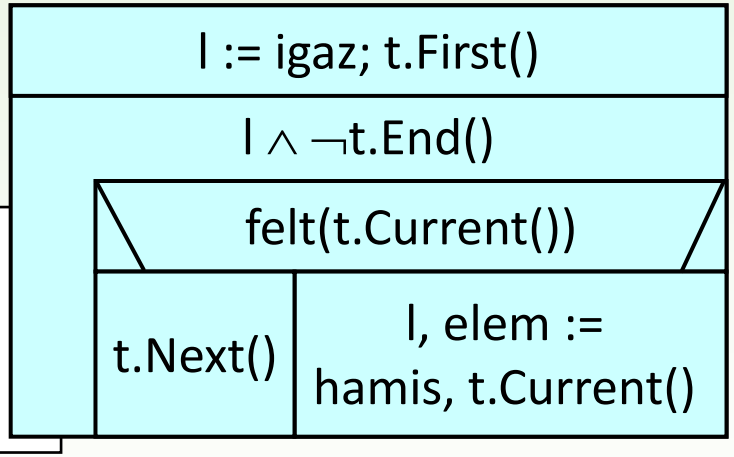
felt : E → 𝕃

A = (t:enor(E), l:𝕃, elem:E)

Ef = (t = t')

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

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




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

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

```
l = t.All(e => felt(e));
```

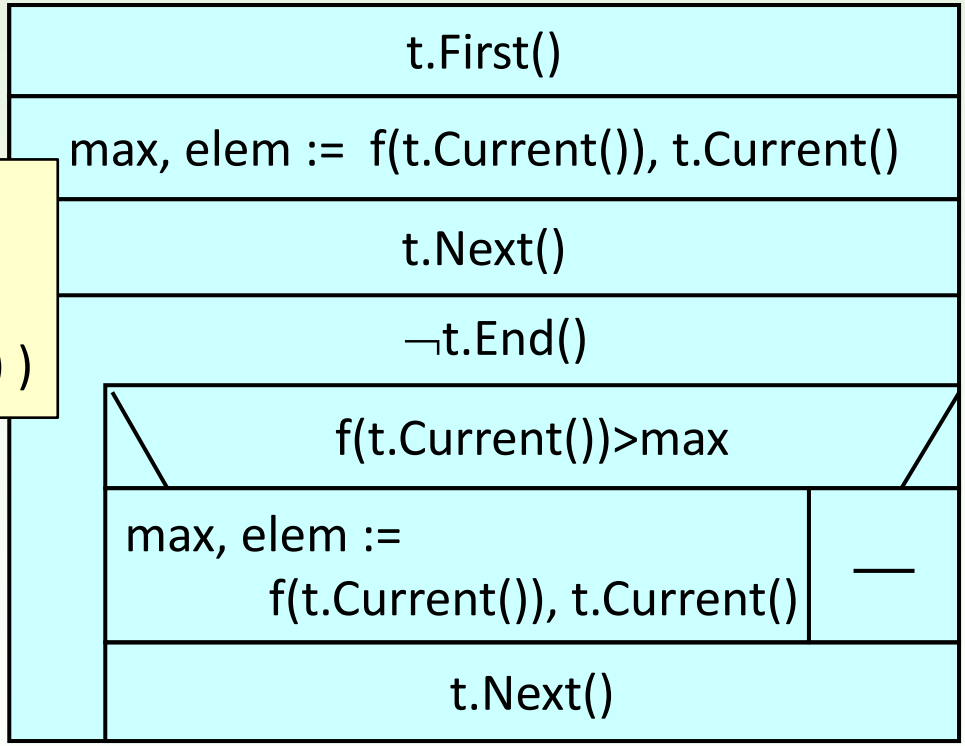
Maximum kiválasztás

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) = $\mathbf{MAX}_{e \text{ in } t'}$ f(e))

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



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

```
max = t.Max(e => f(e));
```

```
elem = t.MaxBy(e => f(e));
```

Feltételes maximum keresés

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

f : E → H
felt : E → 𝕃
H *halmaz elemei rendezhetőek*

(l = ∃i∈[1..|t'|] : felt(t'$_i$)) ∧
(l ⟶ i∈[1..|t'|] ∧ felt(t'$_i$) ∧
∧ ∀k∈[1..t'|] : (felt(t'$_k$) ⟶ f(t'$_k$)≤f(t'$_i$))
∧ elem = t'$_i$ ∧ max = f(elem))

A = (t:enor(E), l:𝕃, elem:E, max:H)

Ef = (t = t')

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

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

l := hamis
t.First()

¬t.End()

felt(t.Current())

¬l ∨ f(t.Current())>max

l, max, elem :=
igaz, f(t.Current()), t.Current()

—

—

t.Next()

```
bool l = false;
foreach ( var e in h )
{
   if (!felt(e)) continue;
   v = f(e);
   if ( !l ) {l = true; max = v; elem = e; }
   else if ( v > max ) {max = v; elem = e; }
}
```

```
s = t.Where(e => felt(e))
     .Max(e => f(e));
```

?

Felsoroló
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 készülnek a tesztesetek
  - az előfeltételt kielégítő (érvényes), illetve azt megszegő (érvénytelen) bemenő adatokkal felírt tesztesetek.
  - az utófeltétel alapján (?) generált tesztesetek vizsgálata
  - …

- Fehér doboz: a kód alapján készülnek 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 algoritmus-mintákra támaszkodik, akkor az adott algoritmus-minta szokásos teszteseteit kell megvizsgálni.

Algoritmus-minták tesztesetei

- Felsoroló szerint (mindegyik algoritmus-minta esetén)
  - eltérő *hosszúságú* felsorolások: nulla, egy illetve hosszabb felsorolásokat is kipróbálunk
  - a felsorolás *első* ill. *utolsó* elemét feldolgozzuk-e

Számlálás: csak az 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*) kifejezéseinek tesztelése (a kifejezésekben használt műveletek tulajdonságai, értelmezési tartományuk)

Feladat

A Föld felszínének egy vonalán adott pontokon megmértük a felszín tengerszint feletti magasságát, és a mért értékeket eltároltuk egy sorozatban. Milyen magas a felszín legmagasabb horpadása, és hol található ez?

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

Ef = (x = x')

Uf = ($Ef \wedge$ (l, max, hol) = **MAX**$_{i=2..|x|-1,\ x[i-1]>x[i]<x[i+1]}$ x[i])

Feltételes maximumkeresés:

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

A = (f : infile(Hármas), l : $\mathbb{L}$, max : $\mathbb{R}$, hol: $\mathbb{N}$)

Hármas = rec(elő:$\mathbb{R}$, közép:$\mathbb{R}$, utó:$\mathbb{R}$, sorsz:$\mathbb{N}$)

Ef = (f = f')

Uf = ((l, max, elem) = **MAX**$_{e \text{ in } f',\ e.elő>e.közép<e.utó}$ e.közép) $\wedge$

$\wedge$ l $\rightarrow$ hol = elem.sorsz)

Feltételes maximumkeresés:

t:enor(E)	~ st,e,f:=read(f)
f(e)	~ e.közép
felt(e)	~ e.elő>e.közép<e.utó
H, >	~ $\mathbb{R}$, >

Infile
read() : bool×Hármas

Modell

A = (f : infile(Hármas), l : $\mathbb{L}$, max : $\mathbb{R}$, hol: $\mathbb{N}$)

Hármas = rec(elő:$\mathbb{R}$, közép:$\mathbb{R}$, utó:$\mathbb{R}$, sorsz:$\mathbb{N}$)

Ef = (f = f')

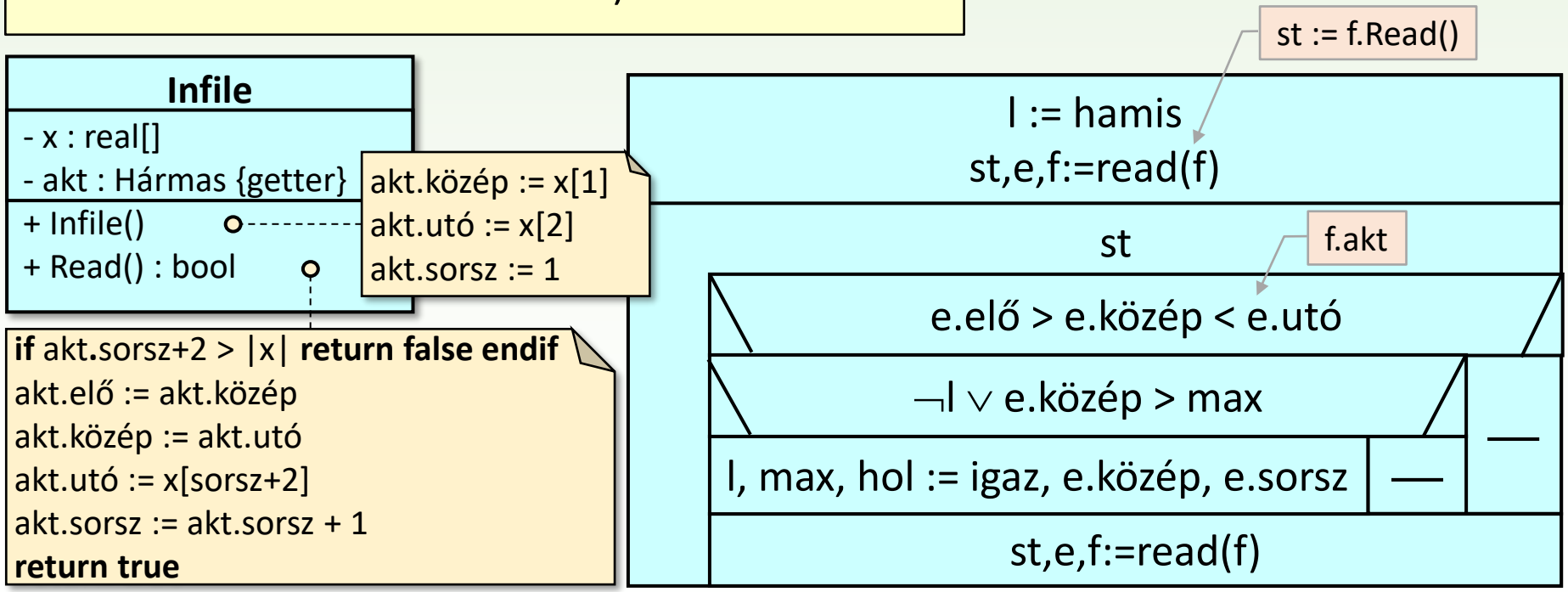
Uf = ((l, max, elem) = $\mathbf{MAX}_{e \text{ in } f'}$ e.közép) $\wedge$

e.elő>e.közép<e.utó

$\wedge$ l $\rightarrow$ hol = elem.sorsz)

Feltételes maximumkeresés:

t:enor(E)	~ st,e,f := read(f)
f(e)	~ e.közép
felt(e)	~ e.elő>e.közép<e.utó
H, >	~ $\mathbb{R}$, >



Szekvenciális inputfájl kódja

```
class Infile
{
   private TextFileReader reader;
   public record Triple
   {
      public double prev, centre, next;
      public int serial;
   }

   public Triple current { get; private set; }

   public Infile(string fname) { … }

   public bool Read() { … }
}
```

```
public Infile(string fname)
{
   reader = new(fname);
   current = new Triple();
   reader.ReadDouble(out current.centre);
   reader.ReadDouble(out current.next);
   current.serial = 1;
}
```

```
public bool Read()
{
   if (reader.ReadDouble(out double hight)) return false;
   current.prev = current.centre;
   current.centre = current.next;
   current.next = hight;
   ++current.serial;
   return true;
}
```

Feltételes maximum keresés kódja

```
static bool MaxSearch(string fname, out double max, out int where)
{
   Infile f = new (fname);

   bool l = false; max = 0.0; where = 0;
   while (f.Read())
   {
      Triple e = f.current;
      if (!(e.prev > e.centre && e.centre < e.next)) continue;
      if (!l) {        l = true; max = e.centre; where = e.serial; }
      else if (e.centre > max) { max = e.centre; where = e.serial; }    }
   return l;
}
```

```
static void Main()
{
   if (MaxSearch("input.txt", out double max, out int where))
        Console.WriteLine($"Height of the highest depression: "
                          { max:f2 } meter on the place { where } ");

   else Console.WriteLine("There is no depression.");
}
```

Tesztelés

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

Tesztkörnyezet

- A projekt metódusainak teszteléséhez egy tesztelő projektet készítünk a VS-ben, amelyben a tesztelendő metódusokhoz (most csak a MaxSearch-höz) teszteseteket definiálunk.

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

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

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

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

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

1. Hozzunk létre egy új projektet az előzővel (Depression) azonos solution-ben: Add new project
2. Az új projekt TestProject legyen
3. Add/Project Reference : Depression
4. Hozzunk létre TestMethod-oket

MaxSearchTests.cs

Egységteszt esetei

```
[TestMethod]
public void Test_EmptyEnumerator()
{
   bool l = Program.MaxSearch("empty.txt", out double max, out int where);
   Assert.AreEqual(l, false);

   // 2.0, 1.0
   l = Program.MaxSearch("two_values.txt", out max, out where);
   Assert.AreEqual(l, false);
}

[TestMethod]
public void Test_NoDepression()
{
   // 1.0, 2.0, 3.0, 4.0, 2.0
   bool l = Program.MaxSearch("no_depressions.txt", out double max, out int where);
   Assert.AreEqual(l, false);
}

...
```

MaxSearchTests.cs

Egységteszt esetei

```
...
[TestMethod]
public void Test_OneLengthEnumerator()
{
   // 2.1, 1.0, 2.4
   bool l = Program.MaxSearch("one_depression.txt", out double max, out int where);
   Assert.AreEqual(l, true);
   Assert.AreEqual(max, 1.0);
   Assert.IsTrue(where == 2);
}
[TestMethod]
public void Test_SeveralMaxs()
{
   // 3.0, 2.0, 3.0, 2.0, 3.0
   bool l = Program.MaxSearch("two_depressions.txt", out double max, out int where);
   Assert.AreEqual(l, true);
   Assert.AreEqual(max, 2.0);
   Assert.IsTrue(where == 2 || where==4);
}
[TestMethod]
public void Test_NoTextFile()
{
   Assert.ThrowsException<FileNotFoundException>(
      ()=>Program.MaxSearch("blabla.txt", out double max, out int where) );
}
```

MaxSearchTests.cs