

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.

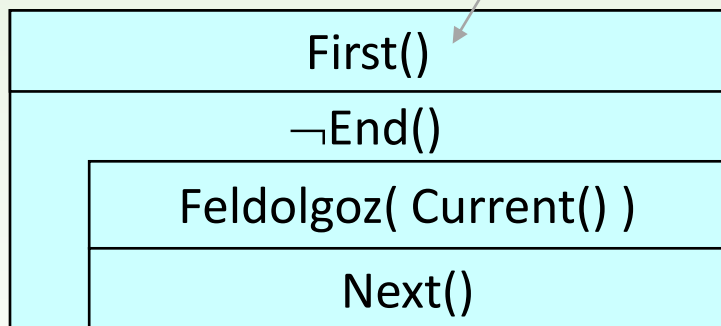
Felsorolás

- ❑ 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
 - $e := \text{Current}()$ ($e:E$): visszaadja a sorozat (felsorolás) E típusú aktuális elemét
 - $l := \text{End}()$ ($l:L$) : 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

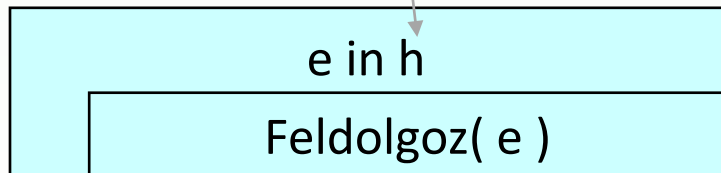
Felsorolás algoritmus

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

gyűjteményt felsoroló műveletek



felsorolandó gyűjtemény



```
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.

```
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

Enumerator	
- ref : Collection	
+ First()	: void
+ Next()	: void
+ Current()	: int {query}
+ End()	: bool {query}

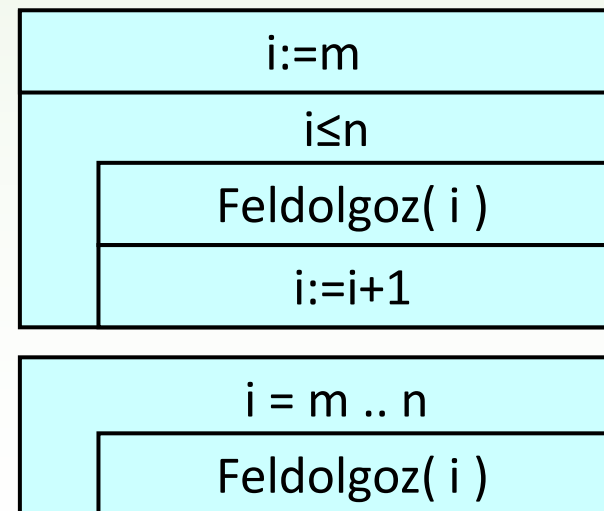
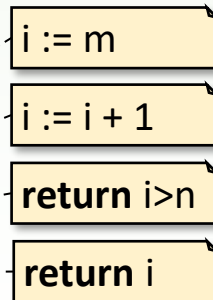
Intervallum klasszikus felsorolója

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

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

az intervallum határai a virtuális gyűjteményt azonosítják

IntervalEnumerator	
- m : int	
- n : int	
- i : int	
+ First()	: void
+ Next()	: void
+ End()	: bool {query}
+ Current(): int	{query}

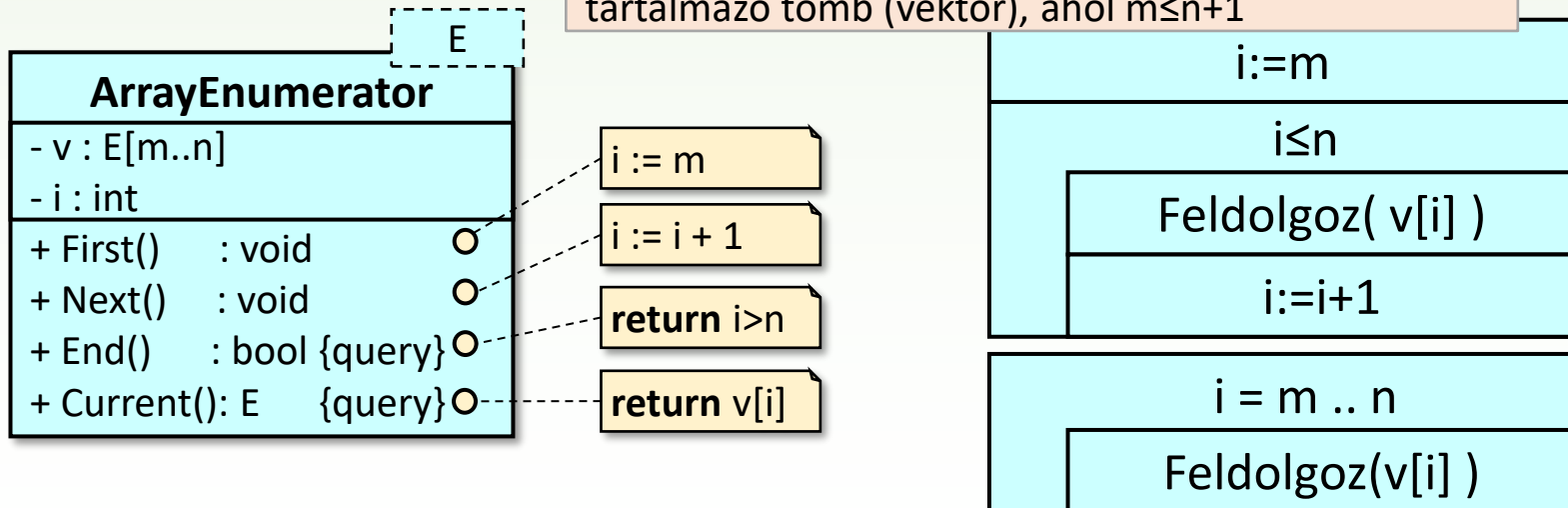


Vektor klasszikus felsorolója

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

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

egy-dimenziós, m-től n-ig indexelt, E típusú elemeket tartalmazó tömb (vektor), ahol $m \leq n+1$



Mátrix sorfolytonos felsorolója

E-beli értékekből álló mátrix elemeinek felsorolása sorfolytonos sorrendben

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

E-beli elemeket tartalmazó mátrix típusa, ahol a sorokat k-tól n-ig, az oszlopokat l-től m-ig számozzuk, és $k \leq n+1$, $l \leq m+1$

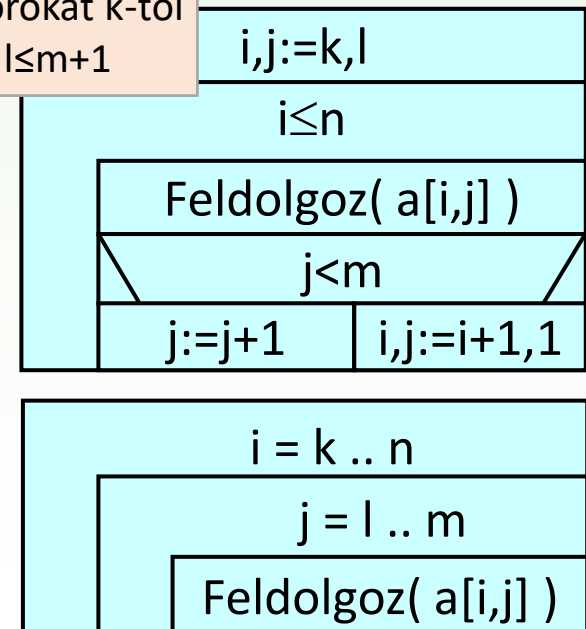
MatrixEnumerator	
- a : matrix of E	
- i, - j : int	
+ First() : void	○
+ Next() : void	○
+ End() : bool {query}	○
+ Current(): E {query}	○

$i, j := k, l$

if $j < m$ **then** $j := j + 1$
else $i, j := i + 1, l$
endif

return $i > n$

return $a[i, j]$



Szekvenciális inputfájl felsorolója

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

enor(E)				
E*	First()	Next()	l:= End() l:ℒ	o:= Current() o:E
x : infile(E) e : E st : Status	st,e,x:=read(x)	st,e,x:=read(x)	l:= st=abnorm	o:= e

E-beli elemek olyan sorozata, amelyre csak a read() művelet használható:

st, e, x:=read(x)

if x = <> **then** st := sikertelen

else st, e, x := sikerült, x₁, <x₂, ... , x_{|x|}>

SeqInFileEnumerator	
x : infile(E) e : E st : Status	
+ First()	: void
+ Next()	: void
+ End()	: bool {query}
+ Current(): E	{query}

st,e,x:read

st,e,x:read

return st=abnorm

return e

st,e,x:=read(x)
st=sikerült
Feldolgoz(e)
st,e,x:=read(x)

előre olvasási stratégia

Felsoroló

2. rész

Algoritmus minták felsorolóval

Gregorics Tibor

gt@inf.elte.hu

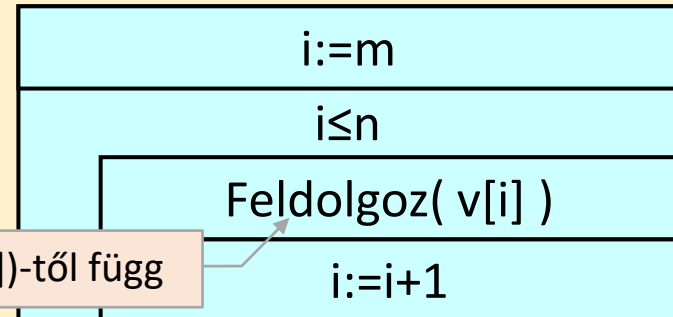
<http://people.inf.elte.hu/gt/oep>

Algoritmus minták általánosítása

Algoritmus-minták vektorra:

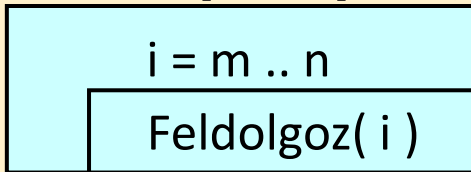
- $i:\mathbb{N}$ az $v:E^{m..n}$ tömb felsorolója
- $felt:E \rightarrow \mathbb{L}$

$v[i]$ -től és/vagy $felt(v[i])$ -től függ

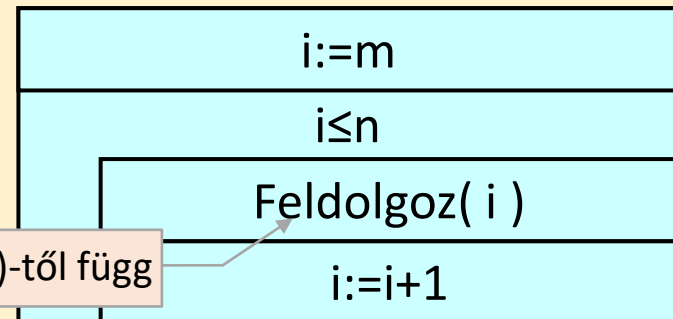


Algoritmus-minták intervallumon értelmezett függvényre:

- $i:\mathbb{N}$ az $[m .. n]$ felsorolója
- $f:[m .. n] \rightarrow H$, $felt:[m .. n] \rightarrow \mathbb{L}$

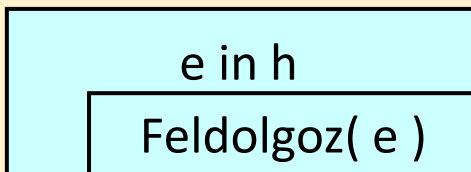


$f(i)$ -től és/vagy $felt(i)$ -től függ

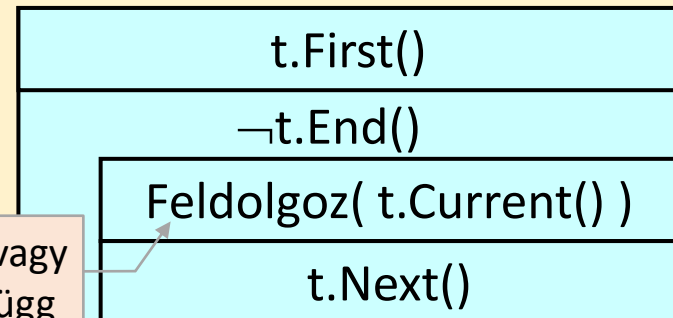


Algoritmus-minták felsorolóra:

- $t: \text{enor}(E)$ felsoroló
- $f:E \rightarrow H$, $felt:E \rightarrow \mathbb{L}$



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



Összegzés

lehet számok szorzása,
logikai értékek összeesélése,
sorozatok összefűzése

Összegezzük egy $\text{enor}(E)$ típusú felsorolás elemeihez rendelt értékeket!

$f : E \rightarrow H$

H elemein értelmezett összegzés:

$+ : H \times H \rightarrow H$ $0 \in H$ *balneutrális elemmel*

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

$Ef = (t = t')$

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

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

speciális eset: feltételes összegzés

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

$\sum_{\text{felt}(e)} g(e)$

azaz $f(e) = \begin{cases} g(e) & \text{ha } \text{felt}(e) \\ 0 & \text{különben} \end{cases}$
(feltéve, hogy 0 jobb neutrális elem is)

$s := 0$

$t.\text{First}()$

$\neg t.\text{End}()$

$s := s + f(t.\text{Current}())$

$t.\text{Next}()$

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

$s = t.\text{Sum}(e \Rightarrow f(e));$

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

$s = t.\text{Where}(e \Rightarrow \text{felt}(e)).\text{Sum}(e \Rightarrow g(e));$?

Számlálás

Számoljuk meg egy $\text{enor}(E)$ típusú felsorolás adott tulajdonságú elemeit!

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

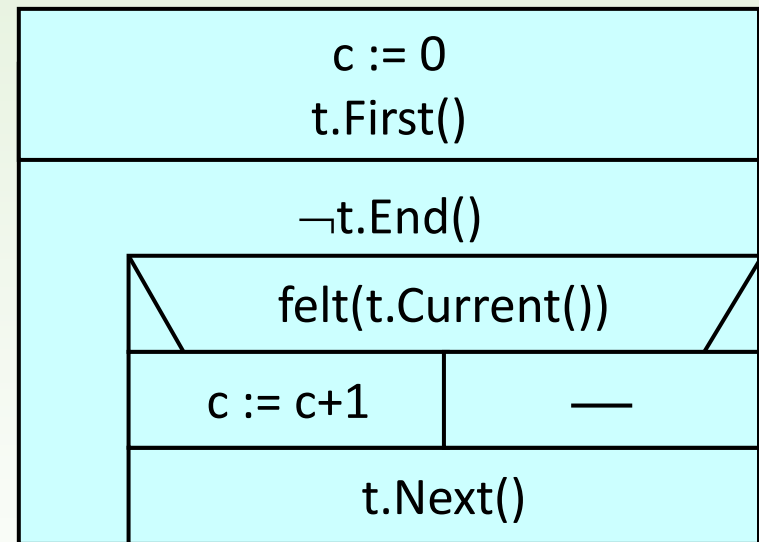
$A = (t : \text{enor}(E), c : \mathbb{N})$

$Ef = (t = t')$

$Uf = (c = \sum_{\text{felt}(e)} 1)$

A számlálás egy speciális összegzés:

$\sum_{e \in t} f(e)$ azaz $f(e) = \begin{cases} 1 & \text{ha } \text{felt}(e) \\ 0 & \text{különben} \end{cases}$



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

$c = t.\text{Count}(e \Rightarrow \text{felt}(e));$

Kiválasztás (biztosan talál)

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

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

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

$Ef = (t = t' \wedge \exists e \in t : \text{felt}(e))$

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

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

$t.\text{First}()$

$\neg \text{felt}(t.\text{Current}())$

$t.\text{Next}()$

$\text{elem} := t.\text{Current}()$

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

$E \text{ elem} = t.\text{Find}(e \Rightarrow \text{felt}(e))!;$

az eredmény biztosan nem lesz null

Lineáris keresés (találat nem biztos)

Keressük meg egy $\text{enor}(E)$ típusú felsorolás adott tulajdonságú első elemét!

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

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

$Ef = (t = t')$

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

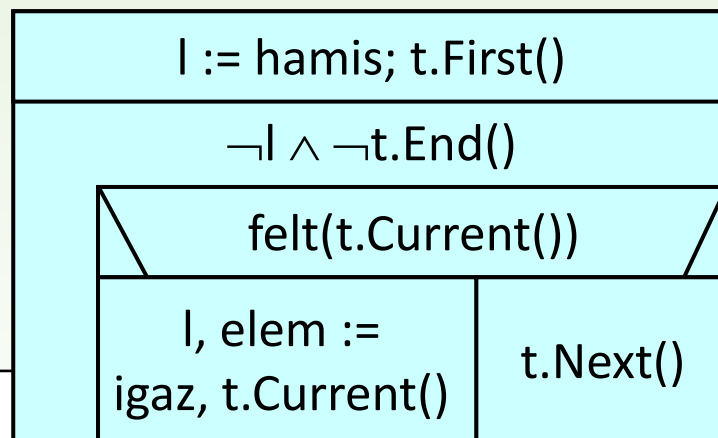
$(l = \exists i \in [1.. |t'|] : \text{felt}(t'_i)) \wedge$
 $(l \rightarrow i \in [1.. |t'|] \wedge \text{felt}(t'_i) \wedge \forall k \in [1..i-1] : \neg \text{felt}(t'_k)$
 $\wedge \text{elem} = t'_i \wedge t = \langle t'_{i+1}, \dots, t'_{|t'|} \rangle)$

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

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

lehet null érték is

$E? \text{elem} = t.\text{First}(e \Rightarrow \text{felt}(e));$



speciálisan eldöntéshez is használhatjuk:

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

$l = t.\text{Any}(e \Rightarrow \text{felt}(e));$

Optimista lineáris keresés

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

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

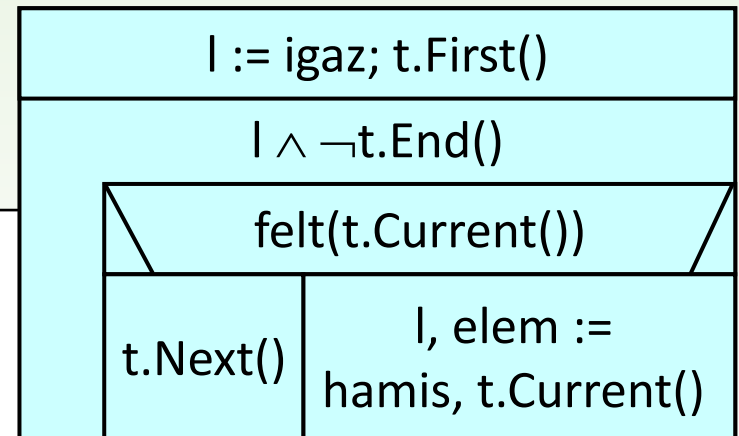
$A = (t : \text{enor}(E), l : \mathbb{L}, \text{elem} : E)$

$Ef = (t = t')$

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

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

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



főleg eldöntésre használjuk:

$l = \forall \text{SEARCH}_{e \text{ in } t'} \text{felt}(e)$ vagy $l = \forall_{e \in t'} \text{felt}(e)$

$l = t.\text{All}(e \Rightarrow \text{felt}(e));$

Maximum kiválasztás

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

$f : E \rightarrow H$

H *elemei rendezhetőek*

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

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

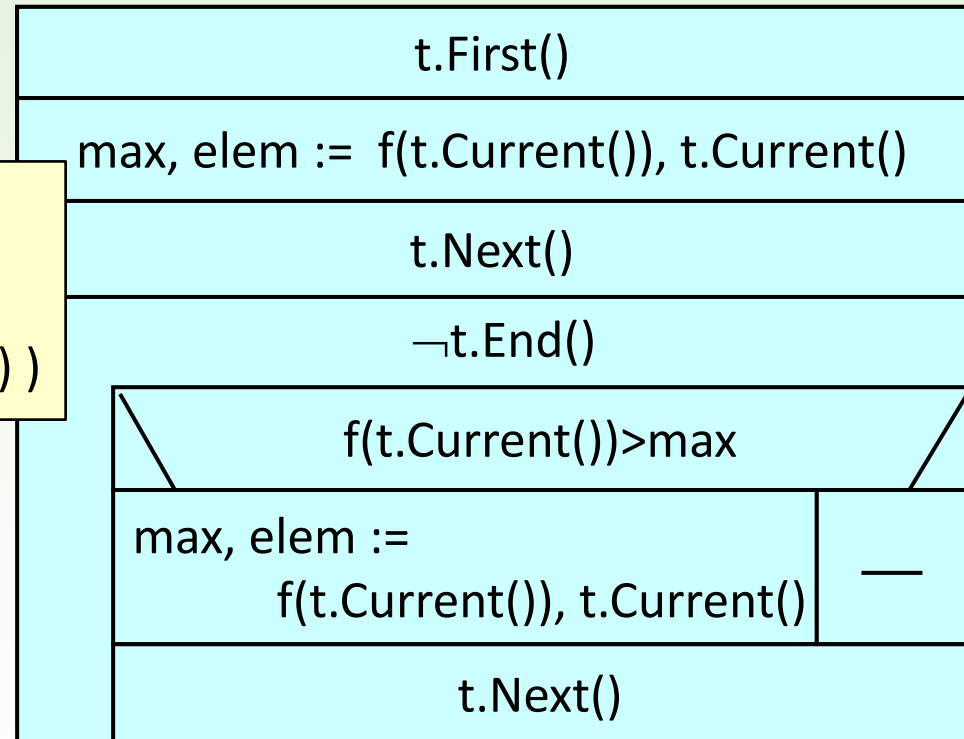
$Uf = ((max, elem) = \mathbf{MAX}_{e \in t'} f(e))$

$\exists i \in [1..|t'|]: \forall k \in [1..|t'|]:$

$f(t'_k) \leq f(t'_i)$

$\wedge elem = t'_i \wedge max = f(elem)$

- 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 $\text{enor}(E)$ típusú felsorolás adott tulajdonságú elemei között egy adott szempont szerinti egyik legnagyobbat és annak értékét!

$f : E \rightarrow H$

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

H *halmaz elemei rendezhetőek*

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

$Ef = (t = t')$

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

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

```
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; }
}
```

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

$l := \text{hamis}$

$t.\text{First}()$

$\neg t.\text{End}()$

$\text{felt}(t.\text{Current}())$

$\neg l \vee f(t.\text{Current}()) > \text{max}$

$l, \text{max}, \text{elem} :=$

$\text{igaz}, f(t.\text{Current}()), t.\text{Current}()$

$t.\text{Next}()$

$s = t.\text{Where}(e \Rightarrow \text{felt}(e)).\text{Max}(e \Rightarrow f(e));$?

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** ($f : E \rightarrow H$, $\text{felt} : E \rightarrow \mathbb{L}$) 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 tesztesei

❑ 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 $f(e)$ 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?

$$A = (x : \mathbb{R}^*, l : \mathbb{L}, \text{max} : \mathbb{R}, \text{hol} : \mathbb{N})$$
$$Ef = (x = x')$$
$$Uf = (Ef \wedge (l, \text{max}, \text{hol}) = \mathbf{MAX}_{i=2..|x|-1} x[i] \wedge x[i-1] > x[i] < x[i+1])$$

Feltételes maximumkeresés:

$$t:\text{enor}(E) \sim i=2..|x|-1$$
$$f(e) \sim x[i]$$
$$\text{felt}(e) \sim x[i-1] > x[i] < x[i+1]$$
$$H, > \sim \mathbb{R}, >$$
$$A = (f : \text{infile}(\text{Hármas}), l : \mathbb{L}, \text{max} : \mathbb{R}, \text{hol} : \mathbb{N})$$
$$\text{Hármas} = \text{rec}(\text{elő}:\mathbb{R}, \text{közép}:\mathbb{R}, \text{utó}:\mathbb{R}, \text{sorsz}:\mathbb{N})$$
$$Ef = (f = f')$$
$$Uf = ((l, \text{max}, \text{elem}) = \mathbf{MAX}_{e \text{ in } f'} e.\text{közép}) \wedge e.\text{elő} > e.\text{közép} < e.\text{utó}$$
$$\wedge l \rightarrow \text{hol} = \text{elem.sorsz}$$

Feltételes maximumkeresés:

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

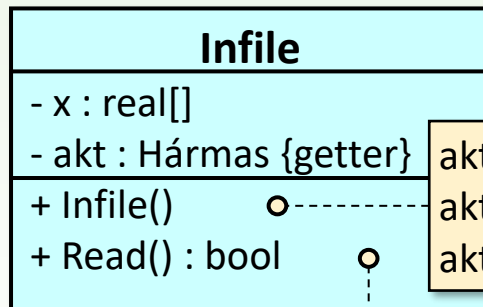
Infile

$\text{read}() : \text{bool} \times \text{Hármas}$

Modell

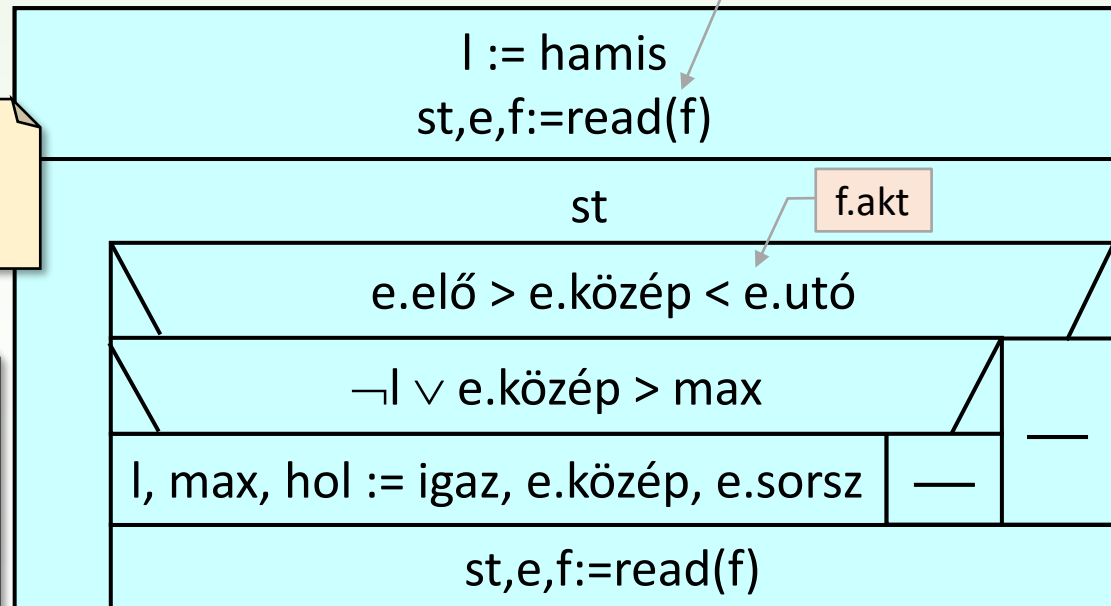
$A = (f : \text{infile}(\text{Hármas}), l : \mathbb{L}, \text{max} : \mathbb{R}, \text{hol} : \mathbb{N})$
 $\text{Hármas} = \text{rec}(\text{elő} : \mathbb{R}, \text{közép} : \mathbb{R}, \text{utó} : \mathbb{R}, \text{sorsz} : \mathbb{N})$
 $Ef = (f = f')$
 $Uf = ((l, \text{max}, \text{elem}) = \mathbf{MAX}_{e \text{ in } f'} e.\text{közép}) \wedge$
 $\quad e.\text{elő} > e.\text{közép} < e.\text{utó}$
 $\quad \wedge l \rightarrow \text{hol} = \text{elem.sorsz})$

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



akt.közép := x[1]
 akt.utó := x[2]
 akt.sorsz := 1

if akt.sorsz+2 > |x| **return false endif**
 akt.elő := akt.közép
 akt.közép := akt.utó
 akt.utó := x[sorsz+2]
 akt.sorsz := akt.sorsz + 1
return true



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 tesztesetei			
felsoroló szerint	hossza: 0	$x = \langle \rangle, \langle 2.0, 1.0 \rangle$	$\rightarrow l = \text{hamis}$
	hossza: 1	$x = \langle 2.1, 1.0, 2.4 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 1.0, \text{hol} = 2$
	hossza: több	$x = \langle 1.0, 2.0, 1.5, 4.0, 2.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 1.5, \text{hol} = 3$
	eleje	$x = \langle 3.0, 2.5, 3.0, 2.0, 3.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 2.5, \text{hol} = 2$
	vége	$x = \langle 3.0, 2.0, 3.0, 2.5, 3.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 2.5, \text{hol} = 4$
tétel szerint	nincs	$x = \langle 1.0, 2.0, 3.0, 4.0, 5.0 \rangle$	$\rightarrow l = \text{hamis}$
	van	$x = \langle 1.0, 2.0, 1.0, 4.0, 2.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 1.0, \text{hol} = 3$
	egy maximum	$x = \langle 3.0, 1.5, 3.0, 2.5, 3.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 2.5, \text{hol} = 4$
	több maximum	$x = \langle 3.0, 2.0, 3.0, 2.0, 3.0 \rangle$	$\rightarrow l = \text{igaz}, \text{max} = 2.0, \text{hol} = 2,4$

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  
{
```

Jelzi a Test Explorer-nek, hogy ennek az osztálynak egy objektumát létre kell hozni a teszteléshez.

```
    [TestClass]
```

```
    public class MaxSearchTests
```

```
    {
```

Jelzi a Test Explorer-nek, hogy le kell futtatni ezt a metódust a tesztelés során.

```
        [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-okat

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(1, 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(1, 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