Gyűjtemények, felsorolók, programozási tételek

Gregorics Tibor

gt@inf.elte.hu

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

Gyűjtemény és feldolgozása

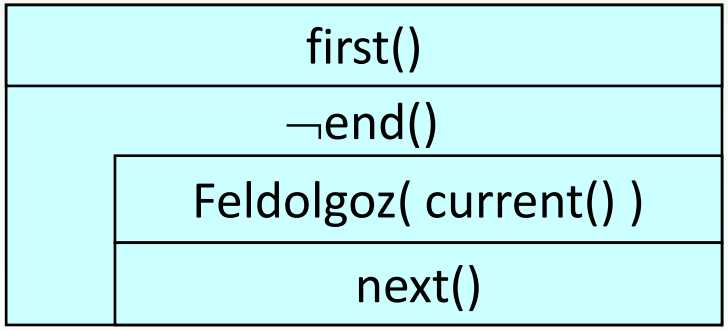
- A gyűjtemény (tároló, kollekció) egy olyan objektum, amely elemek tárolására alkalmas, és az eltároláshoz, valamint a visszakereséshez biztosít műveleteket. Ilyenek például az
  - iterált szerkezetű (azonos típusú elemeket tartalmazó) objektumok
    - Pl.: halmaz, zsák, sorozat (verem, sor), tömb, fa, gráf
- Speciálisan egy gyűjtemény lehet virtuális is, amelyiknek nem tároljuk explicit módon az elemeit
    - Pl.: egész számok egy intervalluma, természetes szám prím-osztói
- 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 feldolgozásához szükségünk van a gyűjtemény felsorolására (bejárására).
- Erre a felsorolásra úgy tekinthetünk, mint a gyűjtemény elemeiből képzett véges sorozatra, amely az alábbi műveletekkel rendelkezik:
  - *first()* : rááll a sorozat első elemére, azaz elkezdi a felsorolást
  - *next()* : rááll a sorozat soron következő elemére, azaz folytatja a felsorolást
  - *l* := *end()* (*l*:𝕃) : megmutatja, hogy a sorozat, azaz a felsorolás végére értünk-e
  - *e*:= *current()* (*e*:*E*): visszaadja a sorozat, a felsorolás aktuális elemét

Felsorolás állapotai

- Egy felsorolásnak különböző állapotai vannak (*indulásra kész*, *folyamatban van*, *befejeződött*): műveletei csak bizonyos állapotokban értelmesek (máskor nem definiált a hatásuk).
- Célszerű a felsorolást olyan algoritmusba beágyazni, amely garantálja, hogy a felsoroló egy műveletét csak akkor (csak olyan állapotban) hajtja végre, amikor az értelmes.




```
for( first(); !end(); next() )
{
    process(current());
}
```

a felsorolt elemek típusát helyettesíti

foreach (forall) ciklus: gyűjtemény felsorolása

```
for( auto e : h )
{
    process(e);
}
```

a felsorolható gyűjtemény

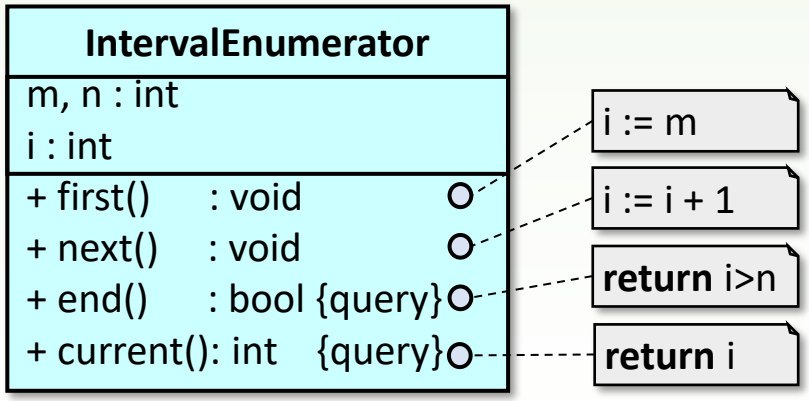
Felsorolás objektummal

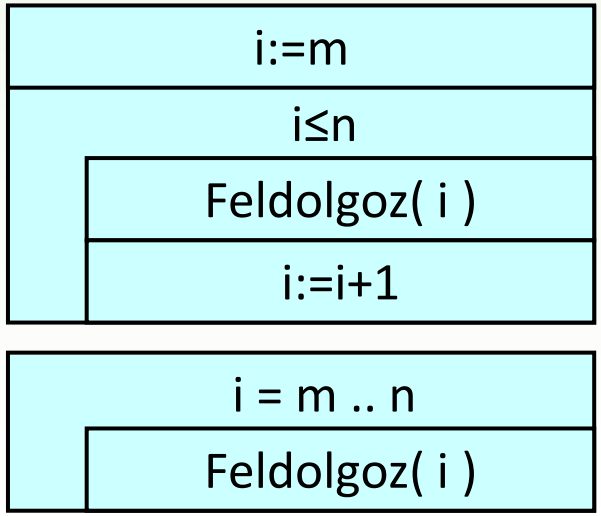
- Azt az objektumot nevezzük felsoroló objektumnak (t : enor(E)) , amely rendelkezik egy adott gyűjtemény felsorolásához szükséges first(), next(), end(), current() műveletekkel.
- A felsoroló típusának megvalósításakor a reprezentáció tartalmazza
  - a felsorolni kívánt gyűjtemény hivatkozását
  - a felsoroló műveletek implementációjához szükséges segéd adatokat.
- Célszerű, ha a felsoroló objektum elkülönül felsorolandó gyűjteménytől, mert így egy gyűjteményen egyszerre több felsoroló is dolgozhat.
- Ha a felsoroló objektumot a felsorolni kívánt gyűjtemény (egyik metódusa) hozza létre, akkor a gyűjtemény biztosan értesül arról, hogy őt felsorolják.

Intervallum klasszikus felsorolója

Egész számok intervallumába eső egész számok felsorolása növekedően.

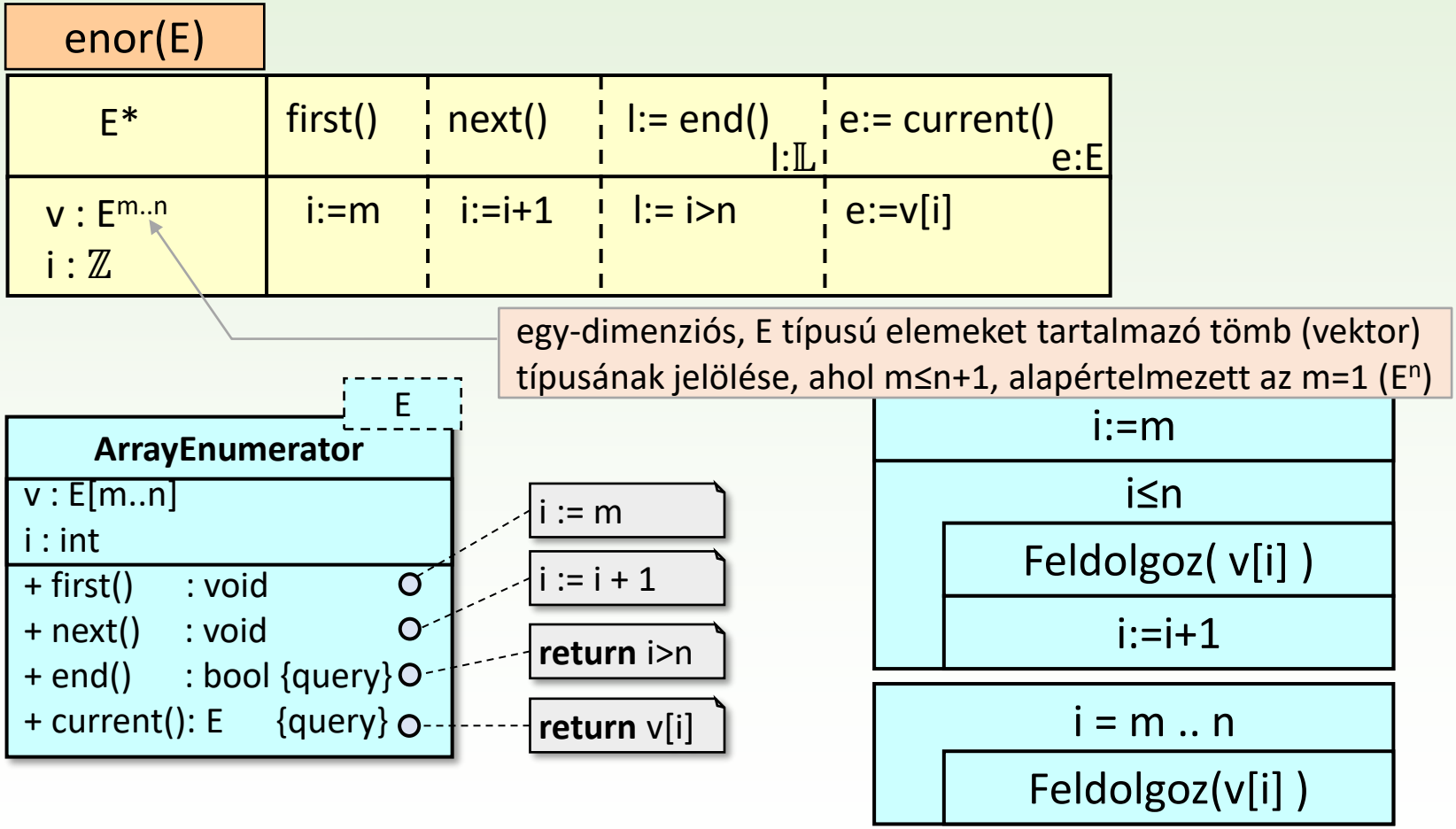
enor(ℤ)				
ℤ*	first()	next()	l:= end() l:𝕃	e:= current() e:ℤ
m , n : ℤ i : ℤ	i:=m	i:=i+1	l:= i>n	e:=i





Vektor klasszikus felsorolója

E-beli értékekből álló vektor elemeinek felsorolása elejétől a végéig



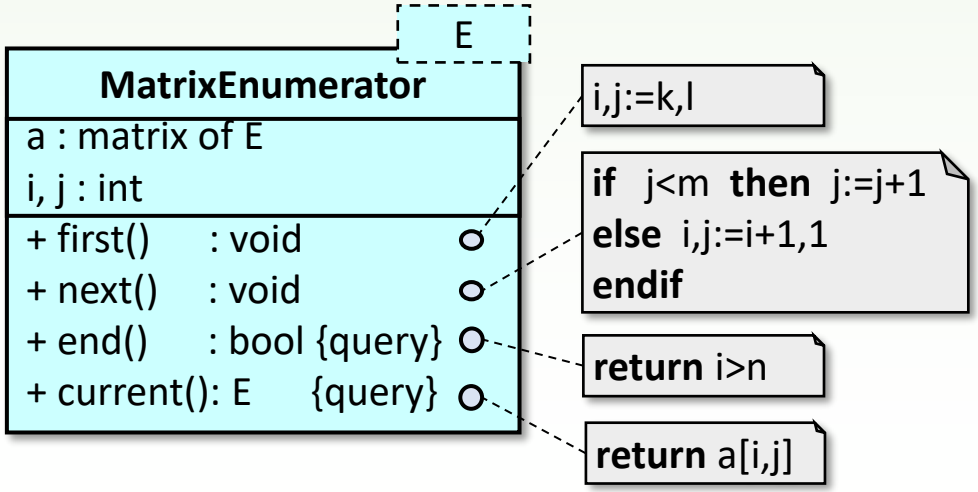
Mátrix sorfolytonos felsorolója

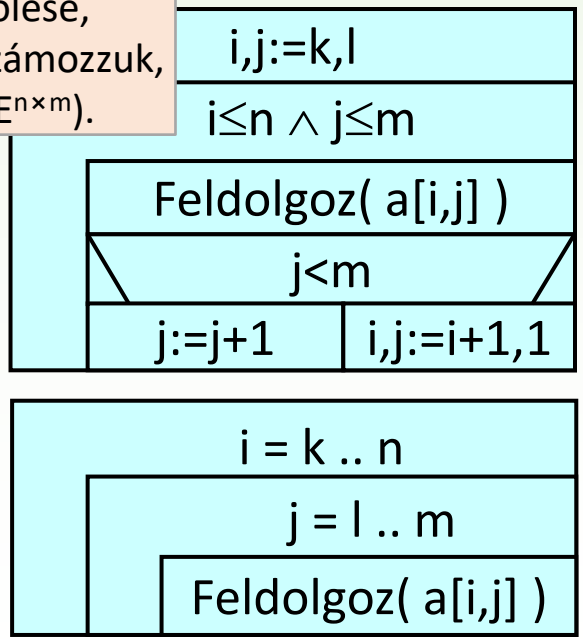
E-beli értékekből álló mátrix elemeinek felsorolása sorfolytonos sorrendben

enor(E)

E*	first()	next()	l:= end() l:𝕃	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+1,1	l:= i>n ∨ j>m	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 ($E^{n \times m}$).





Algoritmus minták általánosítása

- Algoritmus minták vektorra:
  - $t : E^{m..n}$ $(E^{1..n} = E^n)$
  - $f : E \to H$, felt:$E \to \mathbb{L}$

```
i:=m
i≤n
    Feldolgoz(t[i])
    i:=i+1
```

- Algoritmus minták intervallumon értelmezett függvényre:
  - $t : [m \,..\, n]$
  - $f : [m \,..\, n] \to H$, felt : $[m \,..\, n] \to \mathbb{L}$

```
i:=m
i≤n
    Feldolgoz(i)
    i:=i+1
```

- Algoritmus minták felsorolóra:
  - t : enor(E)
  - $f : E \to H$, felt : $E \to \mathbb{L}$

```
e ∈ t
    Feldolgoz( e )
```

```
t.first()
¬t.end()
    Feldolgoz( t.current() )
    t.next()
```

Összegzés

Összegezzük egy felsorolás elemeihez rendelt értékeket!

$f : E \to H$

$+ : H \times H \to H$

$0 \in H$ *baloldali neutrális elem*

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

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

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

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

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

speciális eset: feltételes összegzés

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

```
s := 0
t.first()
¬t.end()
    s := s + f(t.current())
    t.next()
```

Számlálás

Számoljuk meg egy felsorolás adott tulajdonságú elemeit!

felt : E → 𝕃

A = (t:enor(E), c:ℕ)

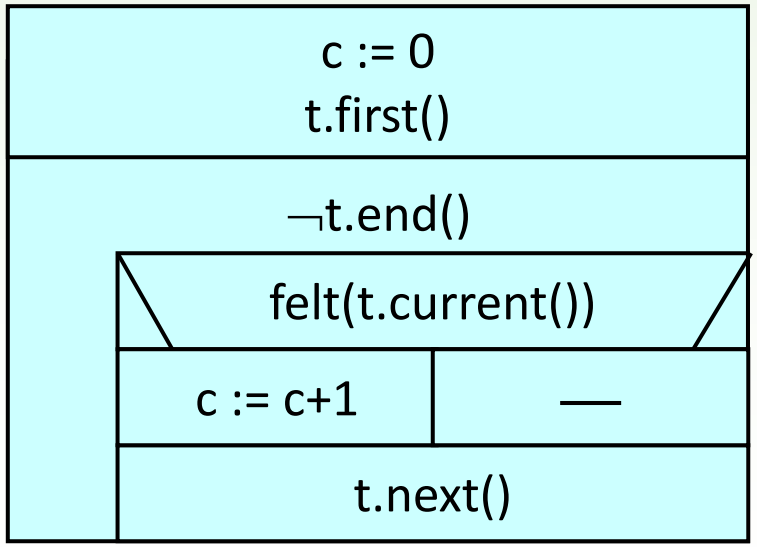
Ef = (t = t')

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

a természetes számokon értelmezett feltételes összegzés

A számlálás egy speciális összegzés:

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



Maximum kiválasztás

Adjuk meg egy felsorolás adott szempont szerinti egyik legnagyobb elemét és annak értékét!

$f : E \rightarrow H$

H *halmaz elemei rendezhetőek*

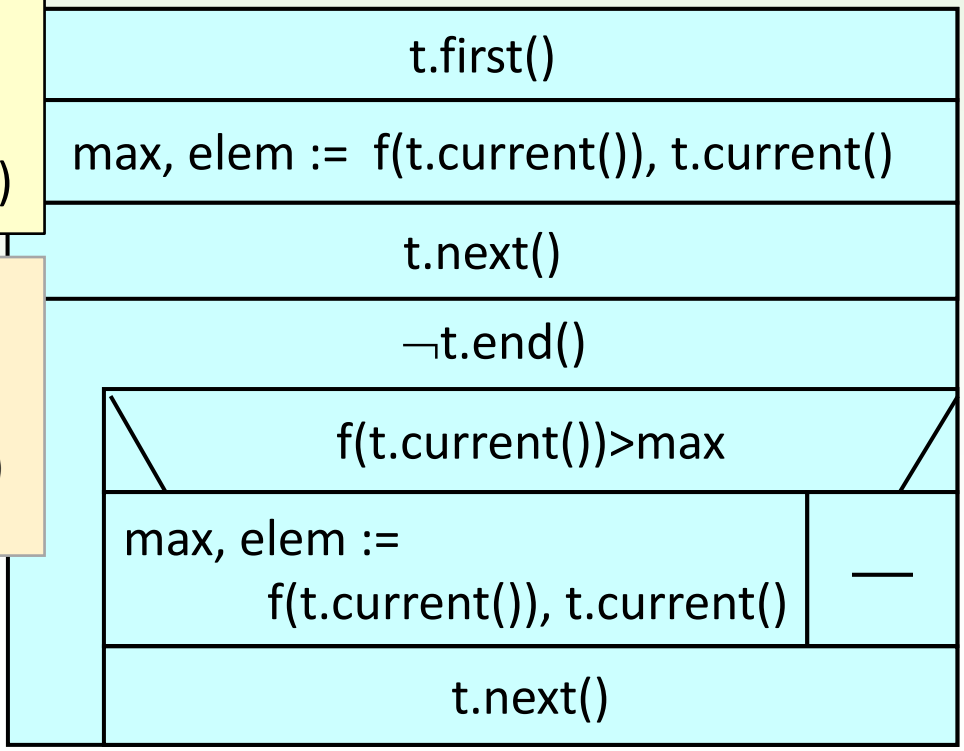
A = (t:enor(E), elem:E, max:H)

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 felsorolás adott tulajdonságú első elemét, ha tudjuk, hogy van ilyen!

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

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

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

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

Megkeresi a *t'* felsorolás első olyan elemét (ez lesz az *elem*), amelyre a feltétel teljesül. A *t* felsorolása a kiválasztás végén még *„folyamatban van”*, nem értünk a végére.

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

t.first()

¬felt(t.current())

t.next()

elem := t.current()

Lineáris keresés (találat nem biztos)

Keressük meg egy felsorolás adott tulajdonságú első elemét!

felt : E → 𝕃

A = (t:enor(E), l:𝕃, elem:E)

Ef = (t = t')

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

Megkeresi a *t'* felsorolás első olyan elemét (ez lesz az *elem*), amelyre a feltétel teljesül. Ha talál ilyet, *l* igaz, különben hamis lesz. A *t* felsoroló csak sikertelen keresés esetén lesz *„befejeződött”*; egyébként maradhatnak *„folyamatban van”* feldolgozatlan elemei.

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

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

```
l := hamis; t.first()
¬l ∧ ¬t.end()
    elem := t.current()
    l := felt(elem)
    t.next()
```

Optimista lineáris keresés

Ellenőrizzük, hogy egy felsorolás minden elemére igaz egy adott tulajdonság, de ha nem, megadjuk az első olyan elemet, amelyikre nem teljesül!

felt : E → 𝕃

A = (t:enor(E), l:𝕃, elem:E)

Ef = (t = t')

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

Megkeresi a *t'* felsorolás első olyan elemét (ez lesz az *elem*), amelyre nem teljesül a feltétel.
Ha nem talál ilyet, az *l* igaz, különben hamis lesz. A *t* felsoroló csak sikertelen keresés esetén lesz *„befejeződött”*; egyébként *„folyamatban van”*, azaz maradhatnak feldolgozatlan elemei.

(l = ∃i∈[1.. |t'|] : felt(t'$_i$)) ∧
(¬l ⟶ i∈[1.. |t'|] ∧ ¬felt(t'$_i$) ∧ ∀k∈[1..i-1] : felt(t'$_k$)
∧ elem = t'$_i$ és t = <t'$_{i+1}$, … , t'$_{|t'|}$>)

speciálisan eldöntésre is használhatjuk:
l = ∀**SEARCH**$_{e∈t'}$ felt(e) vagy l = ∀$_{e∈t'}$ felt(e)

l := igaz; t.first()

l ∧ ¬t.end()

elem := t.current()

l := felt(elem)

t.next()

Feltételes maximum keresés

Keressük egy 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*

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

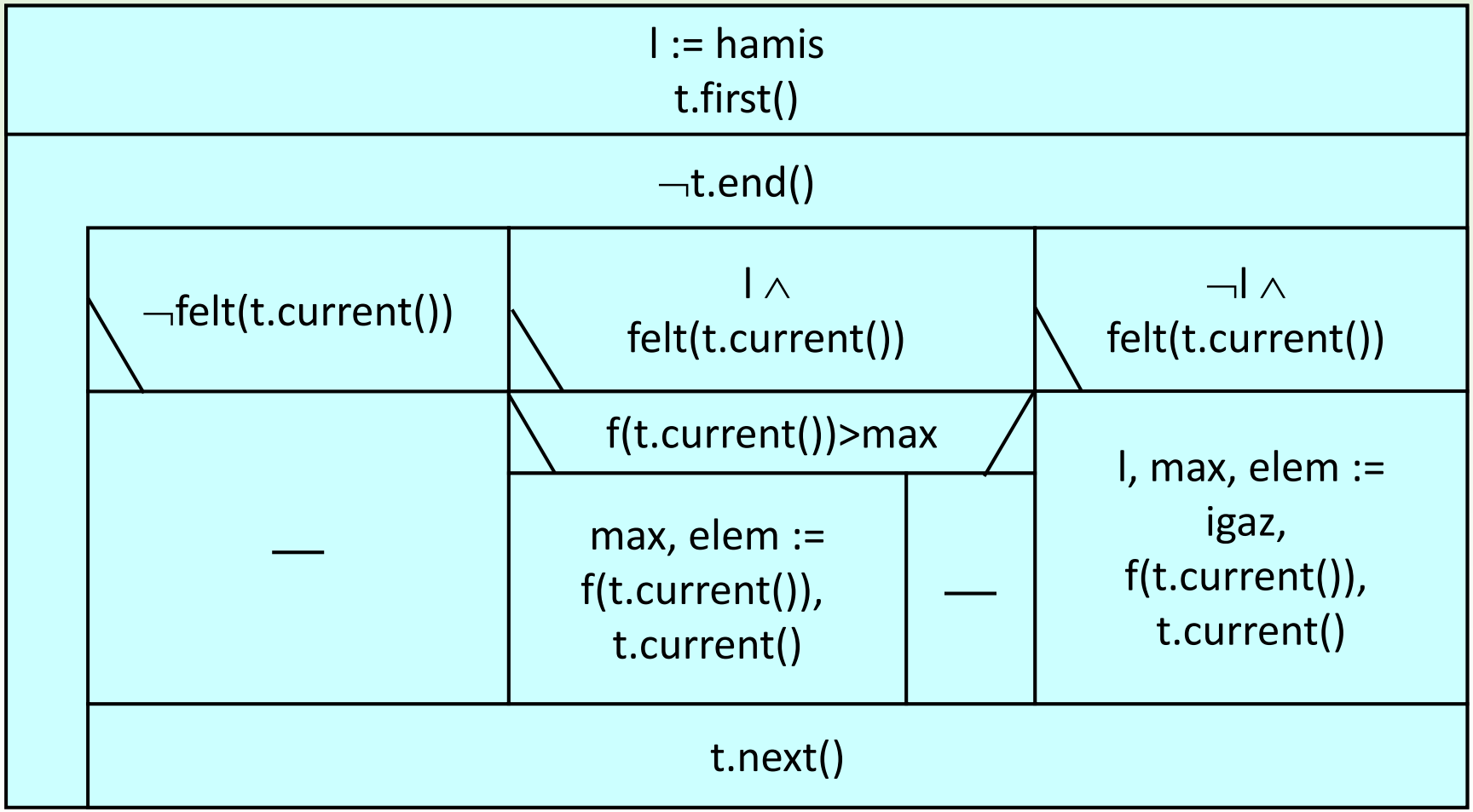
$Ef = (t = t')$

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

Ha van a *t'* felsorolásban olyan elem, amelyre teljesül a feltétel, akkor *l* igaz lesz. Ekkor az *elem* a felsorolás olyan eleme, amelynek *f* szerinti értéke a *max*, ami nagyobb vagy egyenlő a felsorolás bármelyik olyan elemének *f* szerinti értékénél, amely kielégíti a feltételt.

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

Feltételes maximum keresés



- MAX *helyett lehet* MIN
- elem *elhagyható,* max *nem*

Visszavezetés

1. Megsejtjük a feladatot (részfeladatot) megoldó algoritmus mintát.
2. Specifikáljuk a feladatot az algoritmus mintára utaló utófeltétellel.
3. Rögzítjük a feladat és az algoritmus minta közötti eltéréseket:
   - a *felsoroló* és a *felsorolt elemek* típusát
   - a *függvények* (f : E→H, felt : E→𝕃) konkrét megfelelőit
   - a H halmaz *műveletét*, ha kell
     - (H, >) helyett például (ℤ, >) vagy (ℤ, <)
     - (H, +) helyett például (ℤ, +) vagy (ℝ, *) vagy (𝕃, ∧)
   - 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 alapfeladata 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 megszegő ún. érvénytelen tesztesetek
  - az utófeltétel eseteinek 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 ez a végrehajtható specifikáció algoritmus mintákra utal, akkor az algoritmus minták szokásos teszteseteit kell vizsgálni.

Algoritmus minták tesztesetei

- Felsoroló szerint (mindegyik algoritmus minta esetén)
  - eltérő *hosszúságú* felsorolások: nulla, egy illetve hosszabb felsorolásokra is kipróbájuk az algoritmust
  - Felsorolás kezdete és vége: feldolgozza-e az algoritmus a felsorolás *első* ill. *utolsó* elemét
- Funkció szerint (algoritmus mintánként eltérő)
  - *összegzés*: 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*) kifejezések kiszámolásánál használt műveletek sajátosságai.

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?

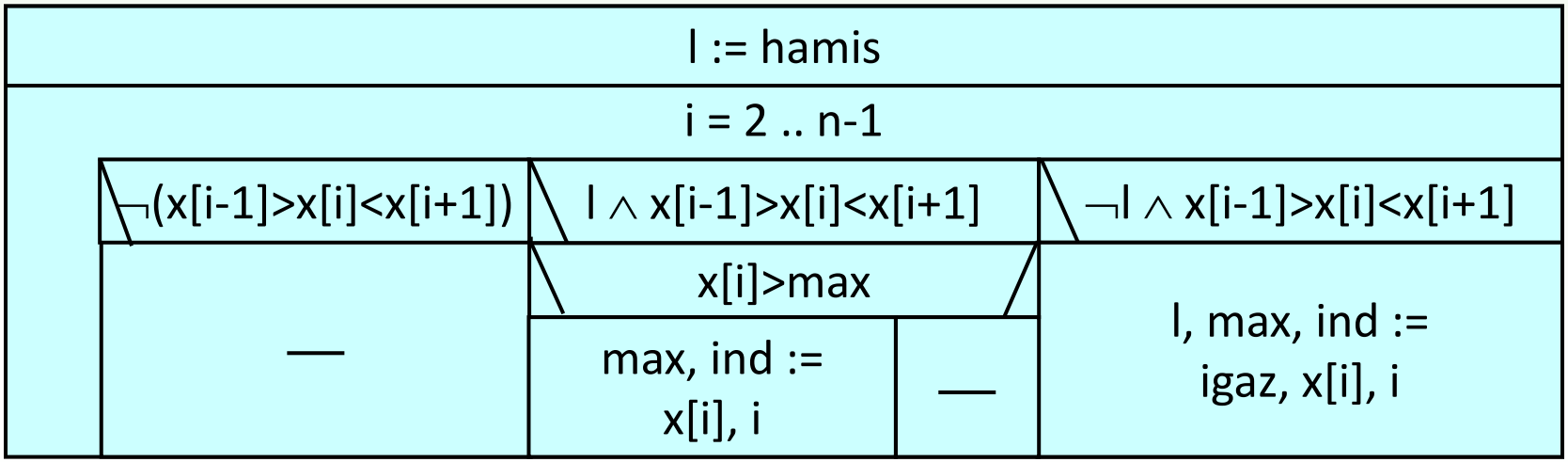
$A = (x : \mathbb{R}^n , l : \mathbb{L}, max : \mathbb{R}, ind : \mathbb{N})$

$Ef = (x = x_0)$

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

Feltételes maximumkeresés:

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



Feltételes maximumkeresés kódja

```
bool maxsearch(const vector<double> &x, double& max, unsigned int& ind);
{
	bool l = false;
	for(unsigned int i=1; t<x.size(); ++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;
}
```

Tesztelés

Feltételes maximum keresés tesztesetei			
felsoroló szerint	hossza: 0	x = < > , < 1.0, 2.0>	→ l = hamis
	hossza: 1	x = < 2.0, 1.0, 2.0 >	→ l = igaz, max = 1.0, ind = 2
	hossza: több	x = < 1.0, 2.0, 1.0, 4.0, 2.0 >	→ l = igaz, max = 1.0, ind = 3
	eleje	x = < 3.0, 2.0, 3.0, 1.0, 3.0 >	→ l = igaz, max = 2.0, ind = 2
	vége	x = < 3.0, 1.0, 3.0, 2.0, 3.0 >	→ l = igaz, max = 2.0, 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, 1.0, 3.0, 2.0, 3.0 >	→ l = igaz, max = 2.0, ind = 4
	több maximum	x = < 3.0, 2.0, 3.0, 2.0, 3.0 >	→ l = igaz, max = 2.0, ind = 2

https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md

Automatikus tesztelés

```
...
using namespace std;

vector<double> FillInFromFile(const string &str);
bool maxsearch(const vector<double> &x, double& max, unsigned int& ind);
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
TEST_CASE("empty interval", "[input1.txt]")
      ifstream f( "input1.txt" ); REQUIRE(!f.fail());
      vector<double> x = FillInFromFile("input1.txt");
      double max=0;
      unsigned int ind=0;
      CHECK(false == maxsearch(x, max, ind));
}
TEST_CASE("one element in interval", "[inpu
      ifstream f( "input2.txt" ); REQUIRE(!f.fa
      vector<double> x = FillInFromFile("input2.txt");
      double max=0;
      unsigned int ind=0;
      CHECK(true == maxsearch(x, max, ind)); CHECK(1 == max); CHECK(2 == ind);
}
...
```

felmaxker
intervallum hossza: 0
x = < > → l = hamis

feltmaxker
intervallum hossza: 1
x = < 2.0, 1.0, 2.0 > → l = igaz, max = 1.0, ind = 2