Gyűjtemények, felsorolók, programozási tételek

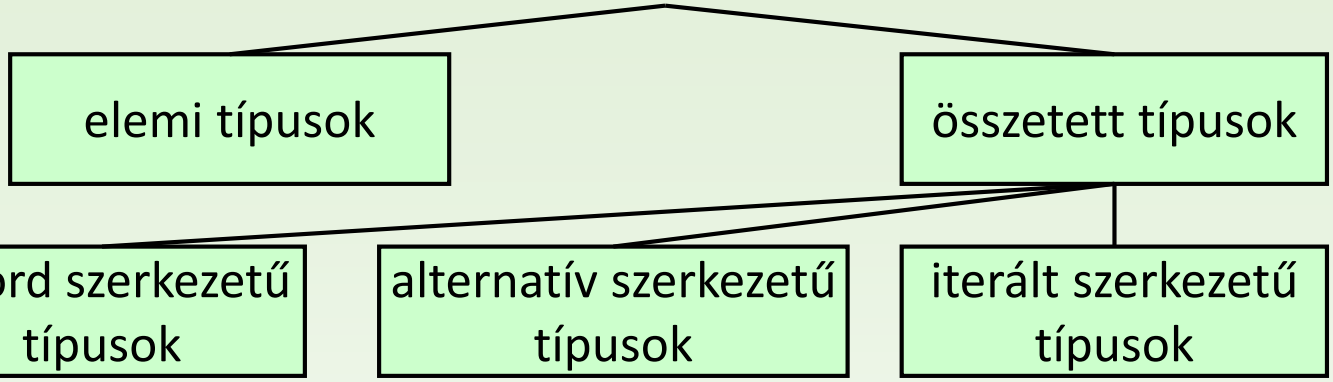
Gregorics Tibor

gt@inf.elte.hu

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

Típus szerkezet

a típusértéket reprezentáló elemek egymáshoz való viszonya



típusértékét több más típus egy-egy értékéből összeállított érték-együttes reprezentálja

T = rec(s_1:T_1, … ,s_n:T_n)
t:T-*nek* i-*edik komponense:* t.s_i

mezőnév

típusértékét egy másik típus véges sok értékéből összeállított gyűjtemény reprezentálja

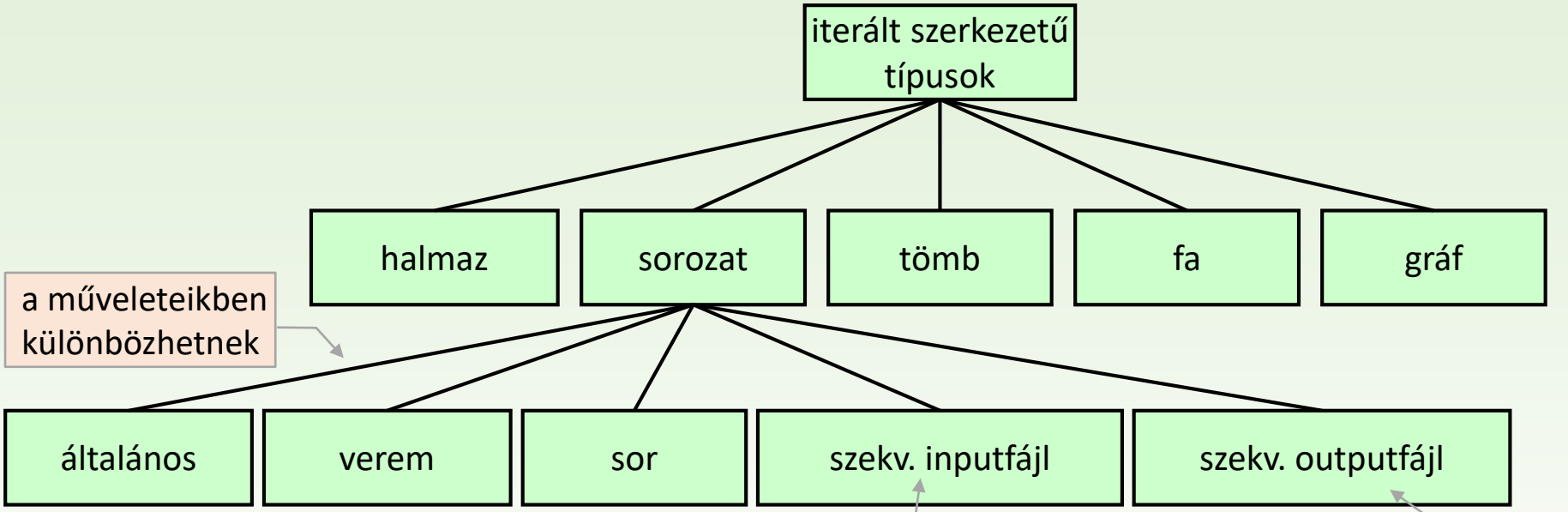
T = it(E)

típusértékét több más típus valamelyikének egyik értéke reprezentálja

T = alt(s_1:T_1, … ,s_n:T_n)
ha t:T *típusa* T_i, *akkor* t.s_i *igaz*

szelektornév

Nevezetes iterált szerkezetű típusok



if f = <> **then** st := abnorm
else st := norm; e := f_1; f := <f_2 , … , $f_{|f|}$>
endif

st, e, f := read(f) **st, e, f : read**
f : *a szekv. inputfájl* (**infile(E)**)
e : *a kiolvasott elem* (**E**)
st: *jelzi, hogy az olvasás sikerült-e* (**Status**)
Status ={norm, abnorm}

f := f ⊕ s

f := write(f, s) **f : write(s)**
f : *a szekv. outputfájl* (**outfile(E)**)
s : hozzáírandó sorozat (**E***)
f := < >

Gyűjtemény

- A gyűjtemény (tároló, kollekció, iterált) egy olyan objektum, amely elemek tárolására alkalmas, és az eltároláshoz, valamint a visszakereséshez biztosít műveleteket.
  - Ilyenek az összetett szerkezetű, de különösen az iterált szerkezetű objektumok: halmaz, sorozat (verem, sor, szekvenciális fájl), tömb, fa, gráf.
  - Vannak úgynevezett virtuális gyűjtemények is, amely elemeit nem kell explicit módon tárolni: pl. egész számok egy intervallumának elemei, vagy egy természetes szám prím-osztói.

Gyűjtemény feldolgozása

- 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 első páros elemét úgy, hogy a tömb elemeit visszafelé járjuk be és csak minden második elemet vizsgálunk meg!
  - Soroljuk fel az *n* természetes szám pozitív prím-osztóit!

Felsorolás

- Egy gyűjtemény felsorolása (bejárása) az elemeinek felsorolását jelenti.
- Erre tekinthetünk úgy, mint a gyűjtemény elemeiből képzett speciális műveletekkel rendelkező véges sorozatra (t : enor(E)).
- A sorozat bejárását az alábbi műveletek végzik:
  - *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

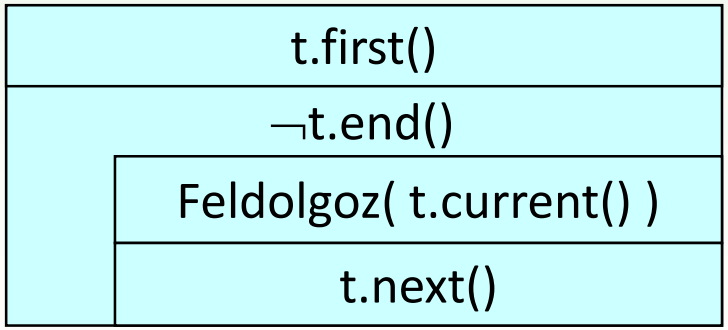
Felsorolás objektummal

- Azt az objektumot nevezzük felsoroló objektumnak, amely rendelkezik egy adott gyűjtemény felsorolásához szükséges first(), next(), end(), current() műveletekkel.
- Ha a felsorolást a felsorolandó gyűjteménytől elkülönülő objektum végzi, akkor ugyanazon gyűjteményen egyszerre több felsoroló is dolgozhat egyszerre.
- A gyűjtemény biztosan értesül arról, hogy őt felsorolják, ha a felsoroló objektumot a felsorolni kívánt gyűjtemény (egyik metódusa) hozza létre.
- A felsoroló megvalósítása a bejárandó gyűjtemény típusától függ. A reprezentáció tartalmazza
  - a bejárt gyűjtemény hivatkozását
  - a műveletek implementációjához szükséges segéd adatokat.

Felsorolás állapotai

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

t : enor(E)




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

a felsorolt elemek típusát helyettesíti

foreach (forall) ciklus: gyűjtemény felsorolása

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

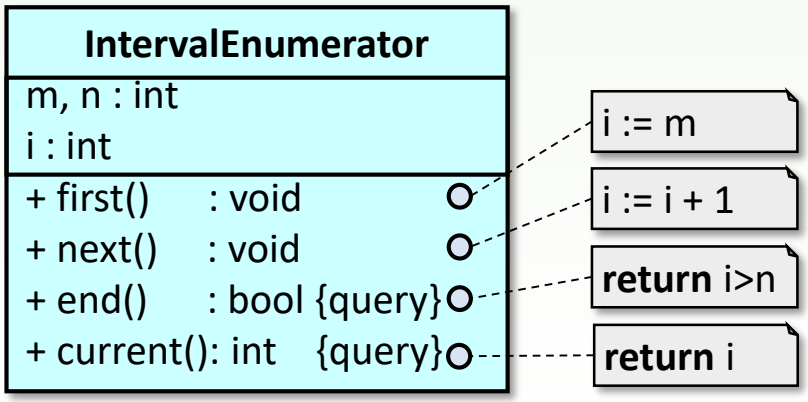
a felsorolható gyűjtemény

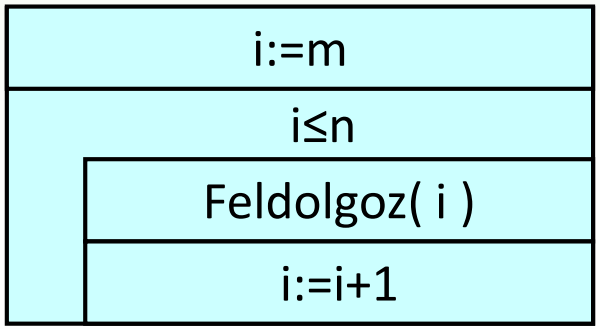
Intervallum klasszikus felsorolója

Egész számok intervallumába eső egész számok felsorolása növekedően.

enor(ℤ)

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



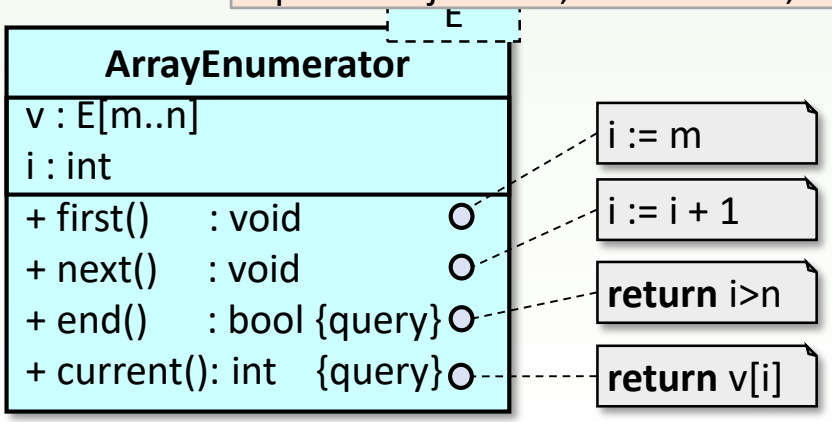


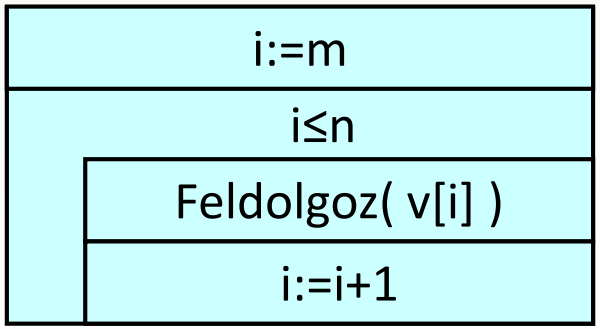
Vektor klasszikus felsorolója

E-beli értékekből álló vektor elemeinek felsorolása elejétől a végéig

enor(E)				
E*	first()	next()	l:= end()	e:= current()
v : $E^{m..n}$ i : $\mathbb{Z}$	i:=m	i:=i+1	l:= i>n l:$\mathbb{L}$	e:=v[i] e:E

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





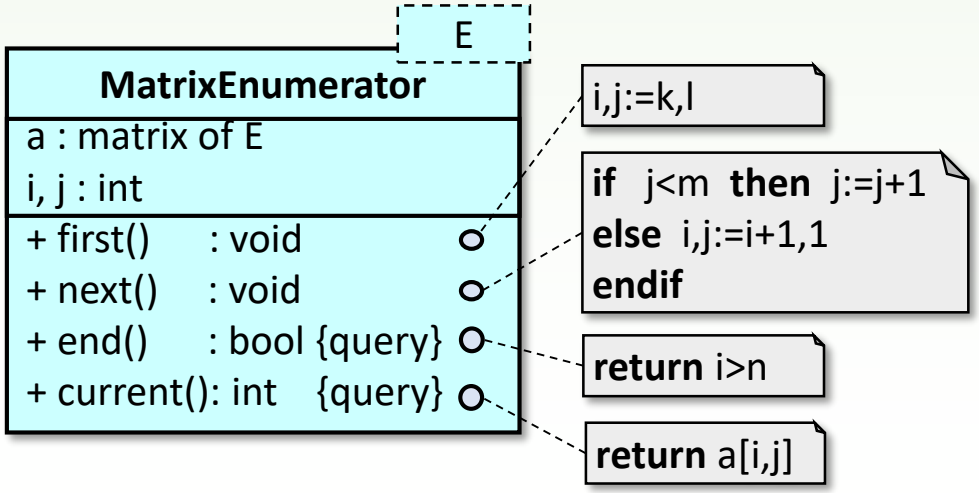
Mátrix sorfolytonos felsorolója

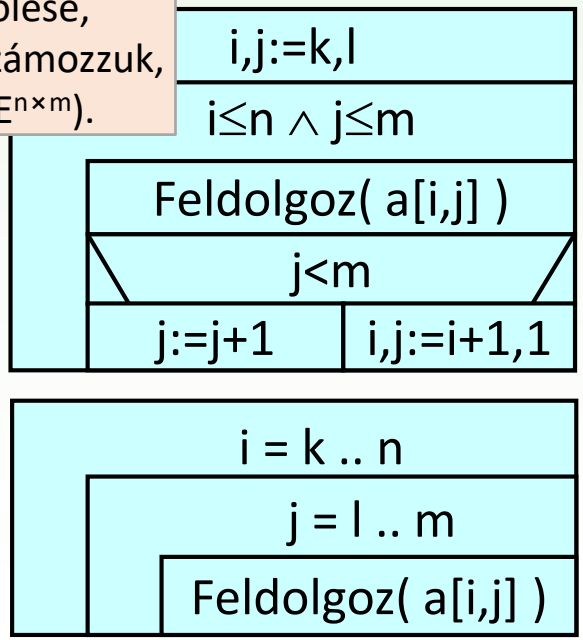
E-beli értékekből álló mátrix elemeinek felsorolása sorfolytonos sorrendben

enor(E)

E*	first()	next()	l:= end()	e:= current()
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+1,1	l:= i>n ∨ j>m l:$\mathbb{L}$	e:= a[i,j] e:E

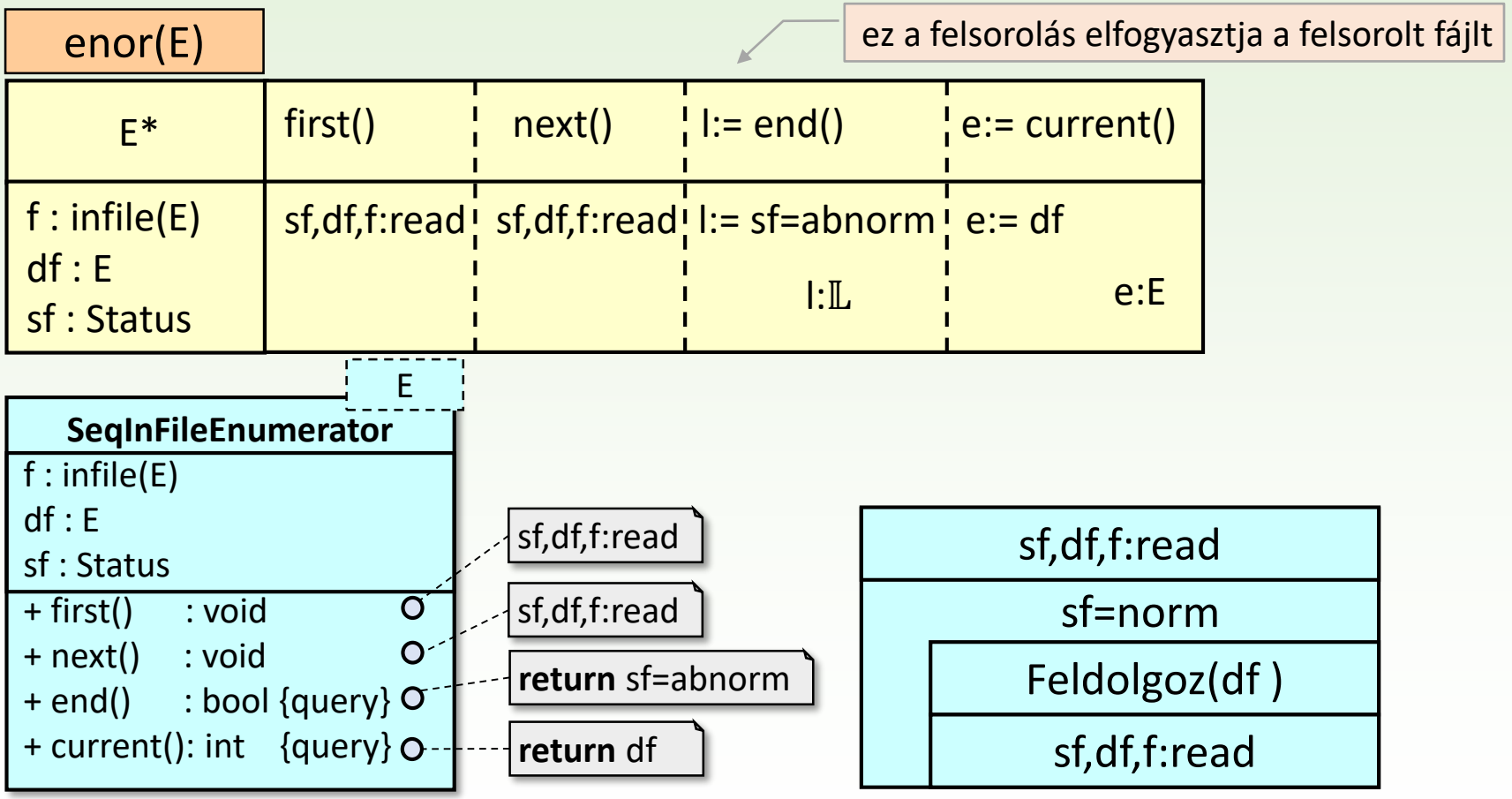
E típusú elemeket tartalmazó mátrix típusának jelölése, ahol a sorokat k-tól n-ig, az oszlopokat l-től m-ig számozzuk, és k≤n+1, l≤m+1, alapértelmezett a k=1 és az l=1 ($E^{n \times m}$).





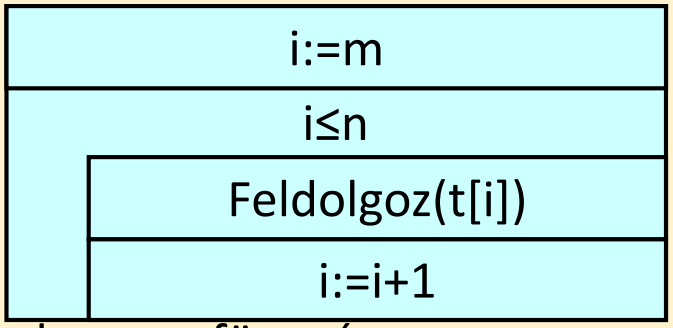
Szekvenciális inputfájl felsorolója

E-beli értékeket tartalmazó szekvenciális inputfájl elemeinek felsorolása

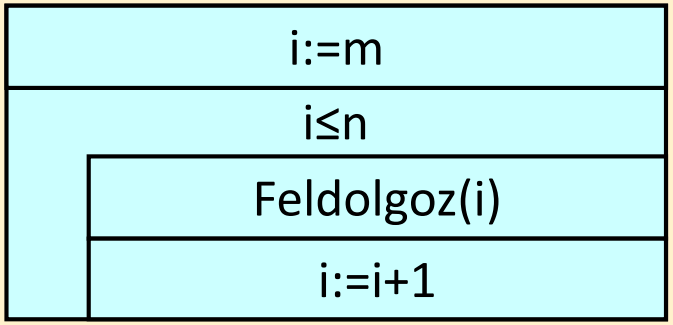


Programozási tételek általánosítása

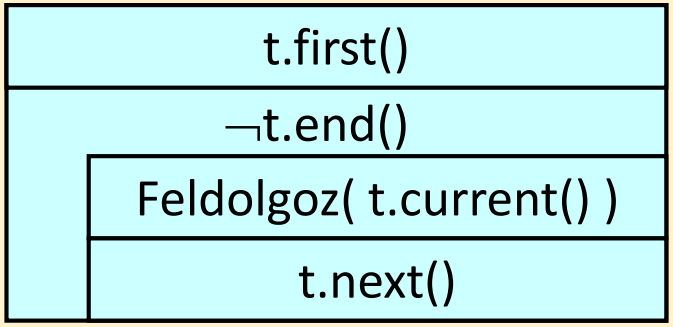
- Programozási tételek tömbre:
  - $t : E^{m..n}$ $(E^{1..n} = E^n)$
  - $f : E \to H$, felt:$E \to \mathbb{L}$



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



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



Összegzés

Összegezzük egy felsorolás elemeihez rendelt értékeket!

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

Ef : t = t'

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

$f : E \to H$

$+ : H \times H \to H$

$0 \in H$ *baloldali neutrális elem*

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

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

speciális eset: feltételes összegzés:

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

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

Számlálás

Számoljuk meg egy felsorolás adott tulajdonságú elemeit!

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

Ef : t = t'

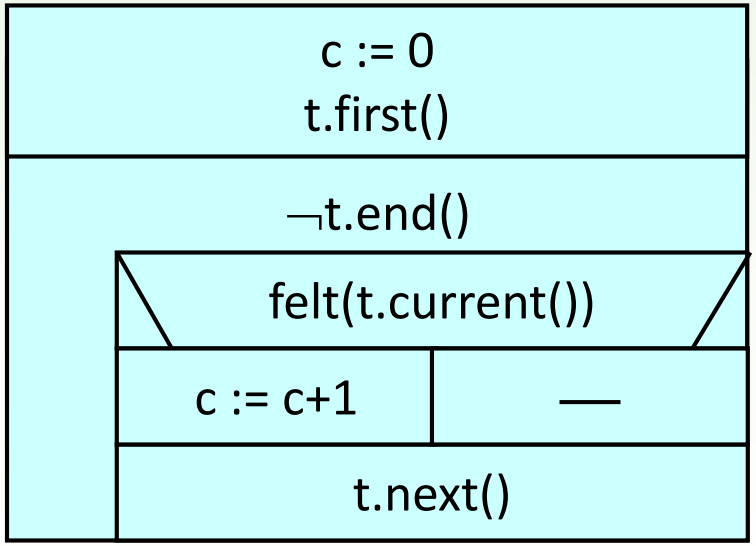
Uf: c = $\Sigma_{e \in t'}$ 1
felt(e)

felt : E → $\mathbb{L}$

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!

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

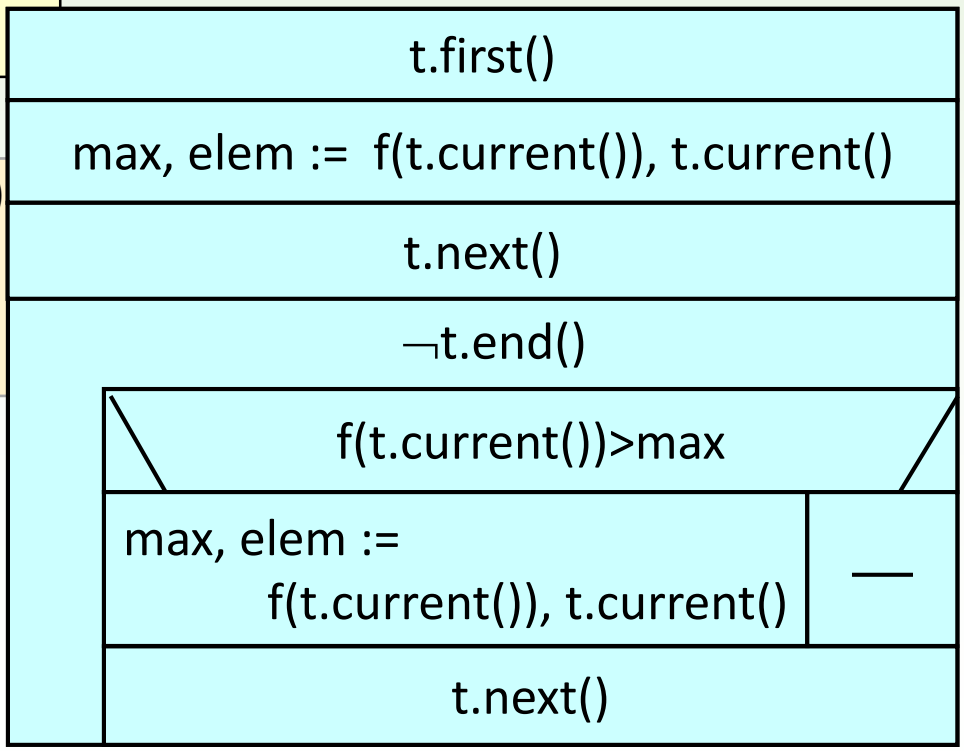
Ef : $t = t' \wedge |t|>0$

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

f : E → H
H *halmaz elemei rendezhetőek*

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

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



Kiválasztás (biztosan talál)

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

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

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

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

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

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 biztosan *„befejeződött”*; egyébként maradhatnak feldolgozatlan elemei.

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

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)

Ha a *t'* felsorolás minden elemére teljesül a feltétel, akkor *l* igaz lesz, különben hamis. Az utóbbi esetben az *elem* a felsorolás első olyan eleme, amelyre nem teljesül a feltétel.

A *t* felsorolása akkor lesz csak biztosan *„befejeződött”*, ha minden elemére teljesül a feltétel; egyébként a felsorolás még *„folyamatban van”* állapotban lehet.

(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!

A : t:enor(E), l:𝕃, elem:E, max:H

Ef : t = t'

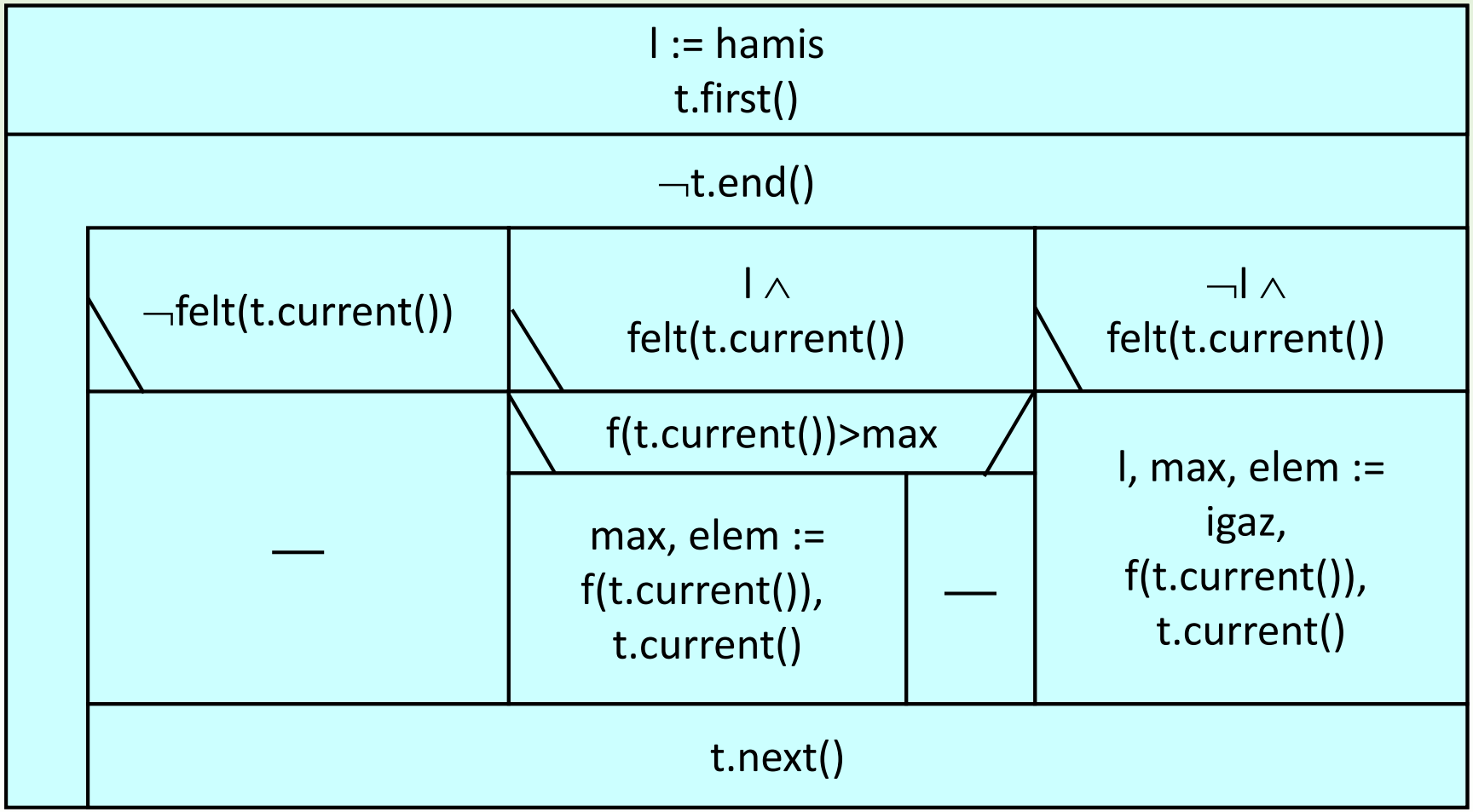
Uf : (l, max, elem) = $\mathbf{MAX}_{e \in t'}^{felt(e)} f(e)$

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

Ha van a *t'* felsorolásban olyan elem, amelyre teljesül a feltétel, akkor 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 lépései

1. Megsejtjük a feladatot (részfeladatot) megoldó programozási tételt.
2. Specifikáljuk a feladatot a programozási tételre utaló végrehajtható utófeltétellel.
3. Megadjuk a feladat és a programozási tétel közötti eltéréseket:
   - *felsoroló* típusát a *felsorolt elemek* típusával együtt
   - *függvények* (f : E→H, felt : E→$\mathbb{L}$) konkrét megfelelőit
   - a H halmaz *műveletét*, ha kell
     - (H, >) helyett például ($\mathbb{Z}$, >) vagy ($\mathbb{Z}$, <)
     - (H, +) helyett például ($\mathbb{Z}$, +) vagy ($\mathbb{R}$, *) vagy ($\mathbb{L}$, $\wedge$)
   - *változók átnevezéseit*
4. Felülírjuk a programozási tétel algoritmusát a fenti különbségek alapján, hogy megkapjuk a feladatot megoldó algoritmust.

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 a specifikáció programozási tételekre utal, akkor a programozási tételekre jellemző teszteseteket kell vizsgálni.

Tesztelési szempontok

- Felsoroló szerint (mindegyik tétel esetén)
  - eltérő *hosszúságú*: nulla, egy illetve hosszabb felsorolásokra is kipróbájuk az algoritmust
  - feldolgozza-e az algoritmus a felsorolás *első* ill. *utolsó* elemét
- Funkció szerint
  - *összegzés*: felsorolás hosszának skálázása
  - *keresés*, *számlálás*: van vagy nincs keresett tulajdonságú elem
  - *max. kiv.*: egyetlen illetve több azonos maximális érték van
  - *felt. max. ker.*: - van vagy nincs keresett tulajdonságú elem
    - egyetlen illetve több azonos, feltételt kielégítő maximális érték van
    - a legnagyobb értékű elem nem elégíti ki a feltételt
- A *felt*(*i*) és *f*(*i*) 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?

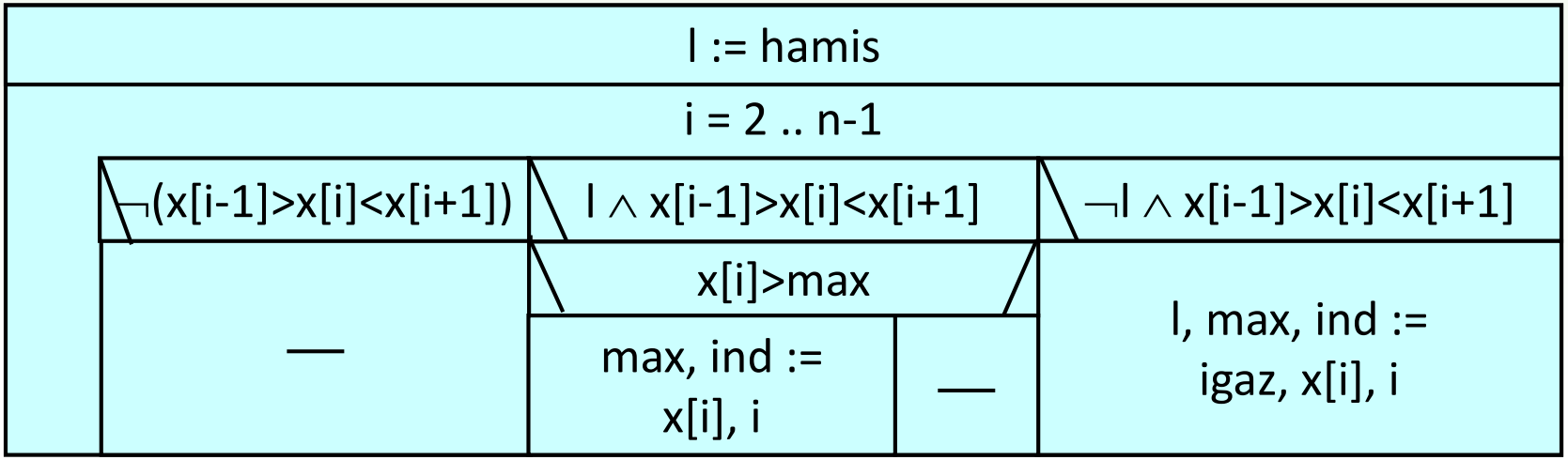
A : $x : \mathbb{R}^n$, $l : \mathbb{L}$, $max : \mathbb{Z}$, $ind : \mathbb{N}$

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, >	~	$\mathbb{R}$, >



Tesztelés

Feltételes maximum keresés tesztesetei			
felsoroló szerint	hossza: 0	x = < > , < 1, 2>	→ l = hamis
	hossza: 1	x = < 2, 1, 2 >	→ l = igaz, max = 1, ind = 2
	hossza: több	x = < 1, 2, 1, 4, 2 >	→ l = igaz, max = 1, ind = 3
	eleje	x = < 3, 2, 3, 1, 3 >	→ l = igaz, max = 2, ind = 2
	vége	x = < 3, 1, 3, 2, 3 >	→ l = igaz, max = 2, ind = 4
tétel szerint	nincs	x = < 1, 2, 3, 4, 5 >	→ l = hamis
	van	x = < 1, 2, 1, 4, 2 >	→ l = igaz, max = 1, ind = 3
	egy maximum	x = < 3, 1, 3, 2, 3 >	→ l = igaz, max = 2, ind = 4
	több maximum	x = < 3, 2, 3, 2, 3 >	→ l = igaz, max = 2, ind = 2

Főprogram

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
using namespace std;
vector<int> FillInFromFile(const string &str);
bool maxsearch(const vector<int> &x, int& maxelem, unsigned int& i);


int main()
{
	vector<int> x = FillInFromFile("input.txt");
	int max=0;
	unsigned int ind=0;
	if(maxsearch(x, max, ind))
		cout << "A legmagasabb horpadas: " << max
			<< " m, a(z) " << ind << "–edik helyen\n";
	} else {
		cout << "Nem volt horpadas!" << endl;
	}
	return 0;
}
```

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

Automatikus tesztelés

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

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

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