ÖSSZESÍTETT TERVGYAK JEGYZET

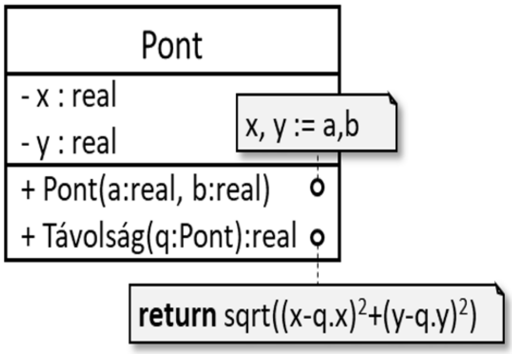
I.

PONT

- Típusdefiníció:

Pont	d := Távolság(p, q) (p, q :Pont, d:$\mathbb{R}$)
x, y : $\mathbb{R}$	d := $\sqrt{(p.x - q.x)^2 + (p.y - q.y)^2}$

- Osztálydiagram:

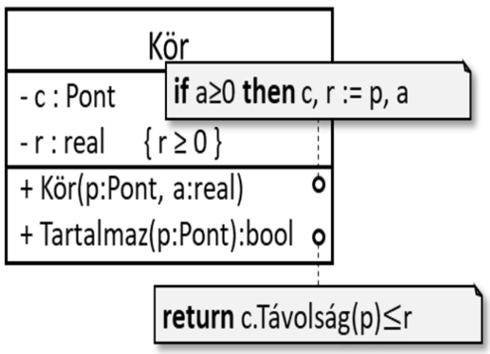


KÖR

- Típusdefiníció:

Kör	l := p∈k (k:Kör, p:Pont, l:$\mathbb{L}$)
c: Pont r: $\mathbb{R}$ Inv: r ≥ 0	l := Távolság(k.c, p) ≤ k.r

- Osztálydiagram:



SÍKBELI PONTOK KÖZÜL HÁNY ESIK A KÖR LEMEZÉRE

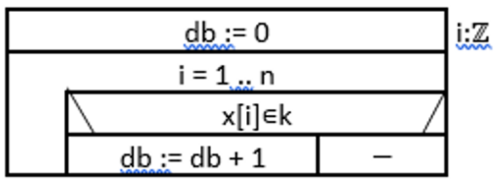
- Specifikáció:

$A = (x:Pont^n, k:Kör, db:\mathbb{N})$

$Ef = (x=x' \wedge k=k')$

$Uf = (Ef \wedge db = \sum_{i=1..n} 1)$

- Algoritmus:

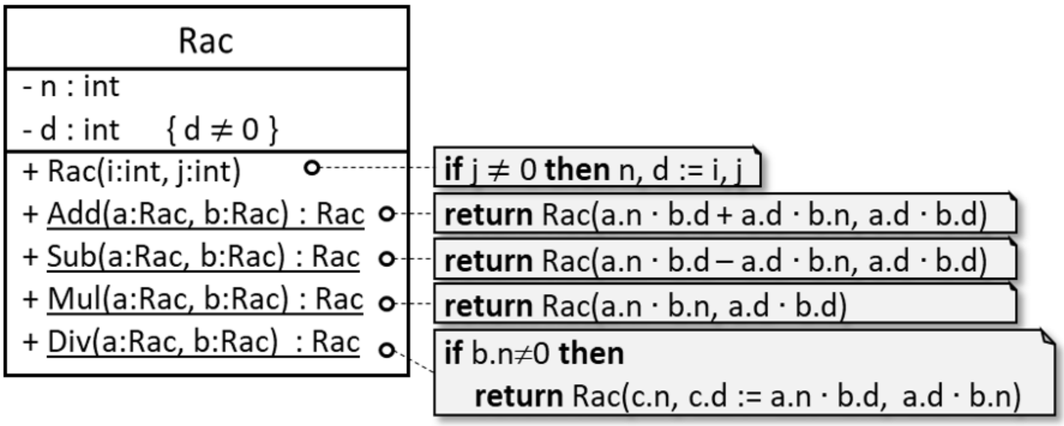


RACIONÁLIS SZÁMOK

- Típusdefiníció:

$\mathbb{Q}$	c := a ± b (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
	c := a · b (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
	c := a / b (b≠0) (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
n, d: $\mathbb{Z}$ // $\frac{n}{d}$ Inv: d≠0	c.n, c.d := a.n · b.d ± a.d · b.n, a.d · b.d
	c.n, c.d := a.n · b.n, a.d · b.d
	c.n, c.d := a.n · b.d, a.d · b.n (b.n≠0)

- Osztálydiagram:



KOMPLEX SZÁMOK

- Típusdefiníció:

$\mathbb{C}$	c := a±b (a: $\mathbb{C}$, b:$\mathbb{C}$, c:$\mathbb{C}$)
	c := a*b (a:$\mathbb{C}$, b:$\mathbb{C}$, c:$\mathbb{C}$)
	c := a/b (b≠0) (a:$\mathbb{C}$, b:$\mathbb{C}$, c:$\mathbb{C}$)
x, y: $\mathbb{R}$ // x+i·y	c.x, c.y := a.x ± b.x, a.y ± b.y
	c.x, c.y := a.x · b.x – a.y · b.y , a.x · b.y + a.y · b.x
	c.x, c.y := (a.x · b.x + a.y · b.y)/(b.x ^{2}+ b.y ^{2}), (a.y · b.x – a.x · b.y) / (b.x ^{2}+ b.y ^{2}) // b.x≠0 ∨ b.y≠0

- Osztálydiagram:

Comp
- x : real - y : real
+ Complex(a:real, b:real) + Add(a:Comp, b: Comp) : Comp + Sub(a:Comp, b: Comp) : Comp + Mul(a:Comp, b: Comp) : Comp + Div(a:Comp, b: Comp) : Comp

x, y := a, b

return Comp(a.x + b.x, a.y + b.y)

return Comp(a.x – b.x, a.y – b.y)

return Comp(a.x · b.x – a.y · b.y , a.x · b.y + a.y · b.x)

II.

DIAGONÁLIS MÁTRIX

Diag
 	
//egy a:Diag mátirix mérete:
a.n x a.n ahol a.n ≥ 1
	e := a[i,j] (a : Diag, i,j : [1..a.n], e : ℝ)
	a[i,j] := e (a : Diag, i,j : [1..a.n], e : ℝ) // ha i=j
	c := a + b (a, b, c : Diag) // a.n = b.n = c.n
	c := a · b (a, b, c : Diag) // a.n = b.n = c.n

X: $\mathbb{R}^*$ // |x|= n ≥ 1	**if** i=j **then** e := a.x[i] **else** e := 0.0 **endif** **if** i=j **then** a.x[i] := e **else** error **endif** **if** |a.x|=|b.x|=|c.x| **then** ∀i∈[1..|a.x|]: c.x[i]:=a.x[i] + b.x[i] // |a.x|=|b.x|=|c.x| **if** |a.x|=|b.x|=|c.x| **then** ∀i∈[1..|a.x|]: c.x[i]:=a.x[i]·b.x[i] // |a.x|=|b.x|=|b.x|

Diag
- x : real[]
+ Diag(n:int) + Get(i:int, j:int) : real { query } + Set(i:int, j:int, e:real) + Add(a:Diag, b:Diag) : Diag + Mul(a:Rac, b:Rac) : Diag

if i∉[1..|x|] **or** j ∉[1..|x|] **then error**
if i=j **then return** x[i]
else return 0.0

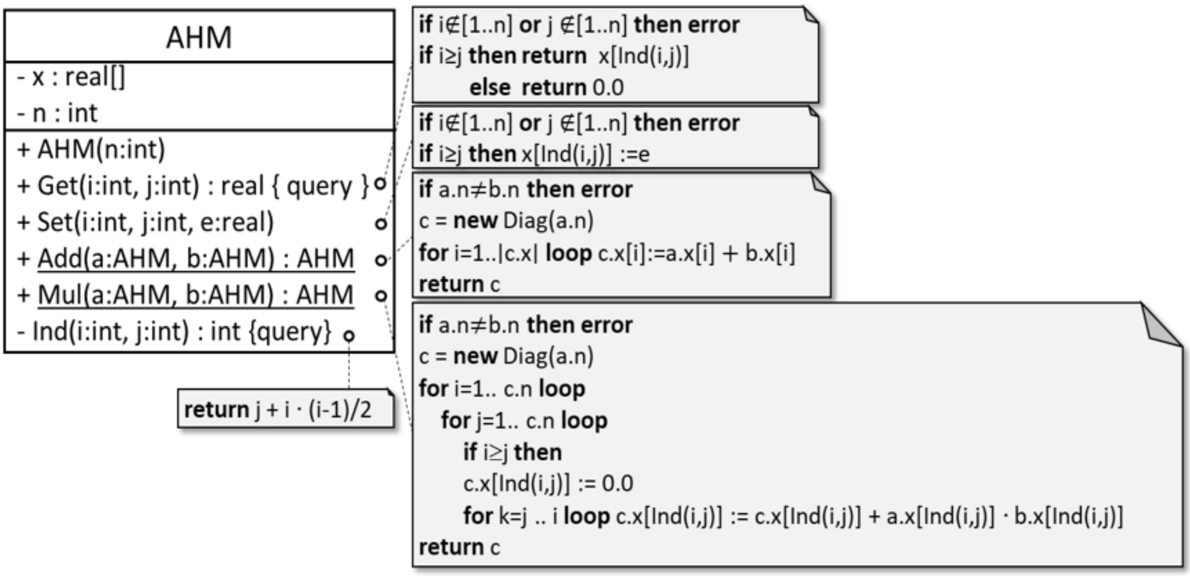
if i∉[1..|x|] **or** j∉[1..|x|] **then error**
if i=j **then** x[i]:=e

if |a.x|≠|b.x| **then error**
c = **new** Diag(|a.x|)
for i=1..|c.x| **loop** c.x[i]:=a.x[i] + b.x[i]
return c

if |a.x|≠|b.x| **then error**
c = **new** Diag(|a.x|)
for i=1..|c.x| **loop** c.x[i]:=a.x[i] · b.x[i]
return c

ALSÓHÁROMSZÖG MÁTRIX

AHM // egy a:AHM mátrix mérete: a.n × a.n ahol a.n≥1	e := a[i,j] (a : AHM, i,j : [1..n], e : ℝ)
	a[i,j] := e (a : AHM, i,j : [1..n], e : ℝ) // ha i ≥ j
	c := a + b (a, b, c : AHM) // a.n = b.n = c.n
	c := a · b (a, b, c : AHM) // a.n = b.n = c.n
X:$\mathbb{R}^*$ n:ℕ // |X|=n·(n+1)/2 // n≥1	**if** i≥j **then** e := a.x[ind(i,j)] **else** e := 0.0
	if i≥j **then** a.x[ind(i,j)] := e **else** error **endif**
	if a.n = b.n = c.n **then** ∀i∈[1..|c.x|]: c.x[i] := a.x[i]+b.x[i]
	if a.n = b.n = c.n **then** { ∀i,j∈[1..c.n]: **if** i≥j **then** c.x[ind(i,j)]:= $\sum_{k=j}^{i}$ a.x[ind(i,k)] · b.x[ind(k,j)] } **endif**



ZSÁK

Bag azon zsákok halmaza, amelyek elemei (E) rendezhetőek	b := SetEmpty(b) b : Bag // *kiüríti a zsákot* l := Empty(b) b : Bag, l : $\mathbb{L}$ // *üres-e zsák* c:= Multipl(b, e) b : Bag, e : E, c : $\mathbb{N}$ // *elem multiplicitása* b := Inser(b, e) b : Bag, e : E // *elemet tesz be* b:= Remove(b, e) b : Bag, e : E // *elemet vesz ki* m:= Max(b) b : Bag, m : E // *leggyakoribb elem*

seq : Element*
maxind : $\mathbb{N}$

ahol
Element =
rec(data: E, count: $\mathbb{N}$)

Invariáns:
- a seq-ben az elemeket tartalmuk (data) szerint rendezetten tároljuk
- a maxind a seq sorozat legnagyobb count értékű elemének indexe

b := SetEmpty(b) b : Bag

seq := <>

l := Empty(b) b : Bag, l : $\mathbb{L}$

l := |seq|=0

c := Multipl(b, e) b : Bag, e : E, c : $\mathbb{N}$

l, ind := logSearch(seq, e)
c :=seq[ind].count

m := Max(b) b : Bag, m : E

|seq| > 0	
m := seq[maxind].data	hiba

b := Insert(b,e) b : Bag, e : E

l, ind := logSearch(seq, e)

l	
++seq[ind].count	seq := seq[1..ind-1] ⊕ <(e,1)> ⊕ seq[ind..|seq|]

seq[ind].count > seq[maxind].count		|seq|=1	maxind≥ind	else
maxind := ind	–	maxind := 1	++maxind	–

b := Remove(b,e) b : Bag, e : E

l, ind := logSearch(seq, e) // data szerint

l		
seq[ind].count > 1	seq[ind].count = 1	
seq[ind].count := seq[ind].count–1	seq := seq[1..ind-1] ⊕ seq[ind+1..|seq|]	–

|seq|>0	
max, maxind := $\mathrm{MAX}_{i=1..\lvert seq\rvert}$ (seq[i].count)	–

$A = (seq : Element^*, e : E, l : \mathbb{L}, ind : \mathbb{N})$

$Ef = (seq=seq_0 \land e=e_0 \land \forall i\in[1 .. |seq|-1] : seq[i].data < seq[i+1].data)$

$Uf = (Ef \land l = \exists i\in[1 .. |seq|] : seq[i].data = e \land$

$(l \to ind\in[1 .. |seq|] \land seq[ind].data = e) \land$

$(\neg l \to \forall i\in [1..ind-1]: seq[i].data < e \land \forall i\in[ind.. |seq|] : seq[i].data > e))$

l, ind := logSearch (seq, e)

ah, fh : $\mathbb{N}$

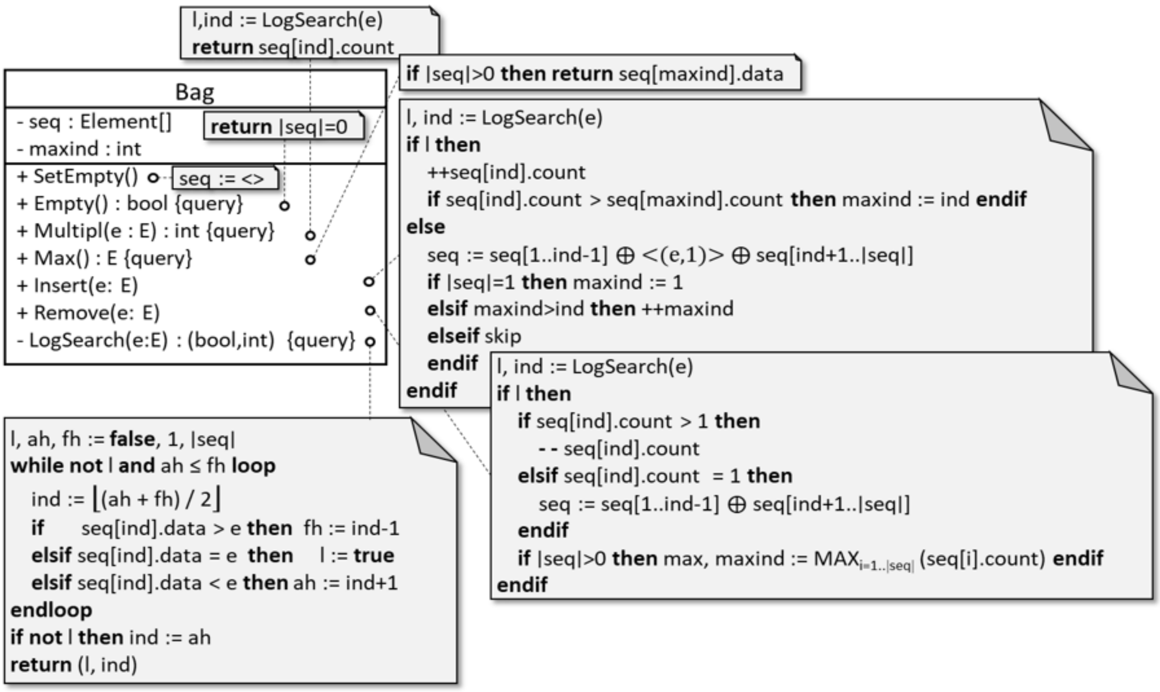
l, ah, fh := hamis, 1, |seq|

$\neg l \land ah \le fh$

ind := ⌊(ah + fh) / 2⌋

seq[ind].key > key	seq[ind].key = key	seq[ind].key < key
fh := ind-1	l := igaz	ah := ind+1

¬l	
ind := ah	—



III.

ASSZOCIATÍV TÖMB

Típus értékek:	**Típus műveletek:**
Map asszociatív tömbök (speciális tárolók) halmaza. Egy tároló elemei $\mathbb{Z} \times \mathbb{S}$ (kulcs-adat) típusú párok, és egy ilyen elem a kulcsa alapján egyértelműen beazonosítható	**map := SetEmpty(map)** map : Map *//kiüríti az asszociatív tömböt* **c := Count(map)** map : Map, c : $\mathbb{N}$ *//megadja az elemek számát* **map := Insert(map,e)** map : Map, e : $\mathbb{Z} \times \mathbb{S}$ *//új elemet tesz be, ha a kulcsa még nem létezik* **map := Erase(map, key)** map : Map, key : $\mathbb{Z}$ *// törli az adott kulcsú elemet, ha a kulcs létezi, különben hiba* **l := In(map, key)** map : Map, key : $\mathbb{Z}$, l : $\mathbb{L}$ *//lekérdezi, van-e adott kulcsú elem* **data := map[key]** map : Map, key : $\mathbb{Z}$, data : $\mathbb{S}$ *// lekérdezi az adott kulcsú elem adatát, ha a kulcs létezik, különben hiba*

Típus reprezentáció:	**Típusműveletek implementációja:**
seq: Element* – az elemeket kulcsuk szerint rendezetten tároló sorozat, ahol Element = rec(key: $\mathbb{Z}$, data: $\mathbb{S}$)	see below

SetEmpty(map)

seq := <>

c := Count(map)

c := |seq|

map := Insert(map,e) l : $\mathbb{L}$, ind : $\mathbb{N}$

l, ind := logSearch(seq, e.key)	
¬l	
seq := seq[1 .. ind-1] ⊕ <e> ⊕ seq[ind .. |seq|]	—

map := Erase(map, key) l : $\mathbb{L}$, ind : $\mathbb{N}$

l, ind := logSearch(seq, key)	
l	
seq := seq[1 .. ind-1] ⊕ seq[ind+1 .. |seq|]	**Hiba**: nem létező kulcs

l := In(map, key)

l, ind := logSearch(seq, key)

ind : $\mathbb{N}$

data := map[key]

l, ind := logSearch(seq, key)	
l	
data := seq[ind].data	**Hiba**: nem létező kulcs

A = (seq : Element*, key : $\mathbb{Z}$, l : $\mathbb{L}$, ind : $\mathbb{N}$)

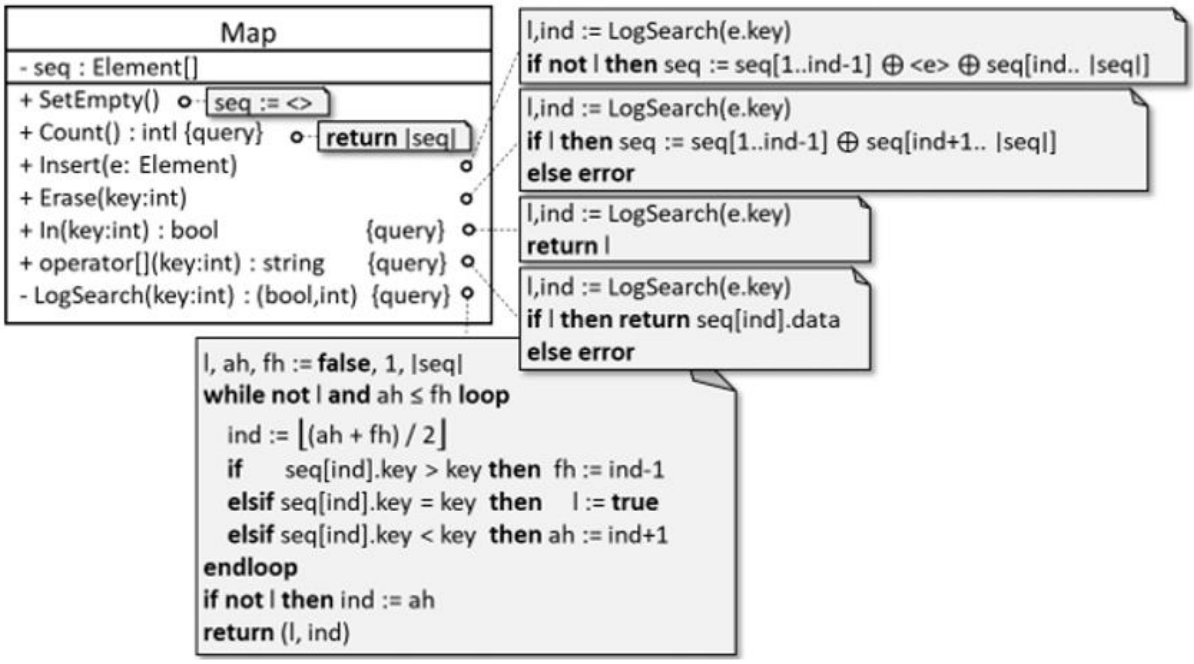
Ef = (seq=seq_0 ∧ key=key_0 ∧ ∀i∈[1 .. |seq|-1] : seq[i].key < seq[i+1].key)

Uf=(Ef ∧ l = ∃i∈[1 .. |seq|] : seq[i].key = key ∧
(l → ind∈[1 .. |seq|] ∧ seq[ind].key = key) ∧
(¬l → ∀i∈ [1..ind-1]: seq[i].key < key ∧ ∀i∈[ind.. |seq|] : seq[i].key > key))

l, ind := logSearch (seq, key)

ah, fh : $\mathbb{N}$

l, ah, fh := hamis, 1, |seq|		
¬l ∧ ah ≤ fh		
ind := ⌊(ah + fh) / 2⌋		
seq[ind].key > key	seq[ind].key = key	seq[ind].key < key
fh := ind-1	l := igaz	ah := ind+1
¬l		
ind := ah	—	



PRIORITÁSI SOR

Típus értékek:	Típus műveletek:
PrQueue a maximum prioritásos sorok halmaza, amely soroknak az elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok.	**SetEmpty(pq)** pq : PrQueue *// Kiüríti a pr sort* **l := isEmpty(pq)** pq : PrQueue, l : $\mathbb{L}$ *//Igazat ad, ha üres a pr sor, hamisat ha nem.* **pq := Add(pq, e)** pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$ *//Betesz egy új elemet a pr sorba.* **e := GetMax(pq)** pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$ *//Visszadja az egyik legnagyobb prioritású elemet, nem veszi ki. Fontos, hogy a sor nem lehet üres.* **pq, e := RemMax(pq)** pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$ *//Kiveszi az egyik legnagyobb prioritású elemet. Fontos, hogy a sor nem lehet üres.*

Rendezetlen sorozatot használunk. Az elemeket folyamatosan helyezzük el a sorozatban. Nem tudjuk, melyik közülük a legnagyobb prioritású.

- **SetEmpty** : üres sorozatot készít. Θ(1) (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)
- **isEmpty:** hosszból azonnal eldönthető. Θ(1)
- **Add:** az új elemet a sorozat végéhez fűzzük. Θ(1)
- **GetMax:** ha nem üres s sor, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és visszaadjuk. A sorozat nem változik. Θ(n)
- **RemMax:** ha nem üres a sorozat, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és kivesszük. Θ(n)

Rendezett sorozatot használunk. A sorozatban az elemek prioritás szerint növekvő a sorrendben vannak.

- **SetEmpty** : üres sorozatot készít. Θ(1) (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)
- **isEmpty:** hosszból azonnal eldönthető Θ(1)
- **Add:** az új elemet betesszük a rendezettség szerinti helyére, amelyet lineáris kereséssel vagy kiválasztással kell megkeresnünk O(n)
- **GetMax:** ha nem üres a sorozat, akkor a rendezettség miatt a sorozat utolsó eleme az egyik legnagyobb prioritású elem. A sorozat nem változik. Θ(1)
- **RemMax:** ha nem üres a sorozat, akkor a rendezettség miatt a sorozat utolsó eleme az egyik legnagyobb prioritású elem. Azt ki is kell vennünk a sorozatból. Θ(1)

Típus reprezentáció:

seq: Element*

- a prioritásos sor elemeit rendezetlenül tároló sorozat, ahol Element = rec(pr : ℤ, data : 𝕊)

Típusműveletek implementációja:

SetEmpty(pq)

seq := <>   (C++ vector: clear())

l := isEmpty(pq)

l := |seq| =0   (C++ vector: size())

pq := Add(pq, e)

seq := seq ⊕ <e>   (C++ vector: push_back())

e := GetMax(pq)

|seq|>0	
max, ind := $MAX_{i=1..\|seq\|}$(seq[i].pr)	**Hiba:** üres prioritásos sor
e := seq[ind]	

pq, e := RemMax(pq)

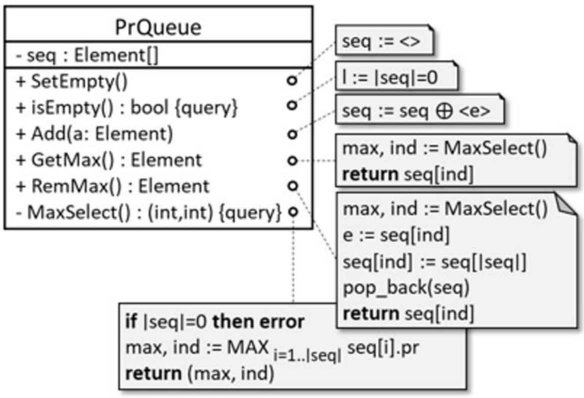
|seq|>0	
max, ind := $MAX_{i=1..\|seq\|}$(seq[i].pr)	**Hiba** üres prioritásos sor
e := seq[ind]	
seq[ind] := seq[|seq|]	
pop_back(seq)	

A max, ind := $MAX_{i=1..|seq|}$(seq[i].pr) segédművelet specifikációja és algoritmusa:

A = (seq:Element*, e:Element)
Ef = (seq=seq' ∧ |seq|>0)
Uf = (*Ef* ∧ e = seq[ind] ∧
(max, ind) = $MAX_{i=1..|seq|}$ seq[i].pr)
ahol max:ℤ és ind:ℕ

max, ind := $MAX_{i=1..|seq|}$(seq[i].pr)

max, ind := seq[1].pr, 1	
i = 2 .. |seq|	
max < seq[ind].pr	
max, ind := seq[ind].pr, i	—



Prioritásos sor megvalósításának tesztelése:

Vegyük sorra, milyen tesztelést képzelnének el az egyes metódusokhoz (példákat a forráskódban lehet látni):

- SetEmpty() (végrehajtása után az isEmpty() igazat ad)
- isEmpty() (üres / nem üres állapotra kipróbáljuk)
- Add(Item e) (egymás után berakunk elemeket, majd ellenőrizzük az elhelyezésüket)
- MaxSelect() (max és az ind vizsgálandó)
- GetMax() (a maxsearch()-höz képest még a hibás esetet kell tesztelni)
- RemMax() (a max()-hoz képest még a tömb átrendeződését is ellenőrizzük)

pq, e := RemMax(pq)

|seq|>0	
max, ind := $MAX_{i=1..\|seq\|}$(seq[i].pr)	**Hiba** üres prioritásos sor
e := seq[ind]	
seq[ind] := seq[|seq|]	
pop_back(seq)	

remMax() tesztelése

teszteset	teszt tömb	eredmény	tömb új tartalma
üres intervallum	<>	hiba (kivétel dobás)	<>
egy elemű	<3>	3	<>
több elemű esetek:			
első a legnagyobb	<5,2,3>	5	<3,2>
utolsó a legnagyobb	<1,2,3>	3	<1,2>
belső a legnagyobb	<1,3,2>	3	<1,2>
nem egyértelmű, első és utolsó a legnagyobb	<5,2,5'>	5	<5',2> (az adat rész segítségével ellenőrizhető, hogy az elsőt vettük ki)
nem egyértelmű, belső és utolsó a legnagyobb	<1,3,3'>	3	<1,3'>
mind egyforma	<3,3',3''>	3	<3'',3'>
több egymás utáni remMax(), majd add hatása	<2,3,1>	3 2 1 3 2 1	<2,1> remMax() <1> remMax() <> add(3) add(2) add(1) remMax() <1,2> remMax() <1> remMax() <>

IV.

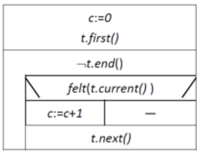
KAKTUSZ: NÉV, ŐSHAZA, SZÍN, MÉRET

a) Számoljuk meg a piros virágú kaktuszokat!

P.t.: Számlálás

Specifikáció:

$A = (t{:}enor(E), c{:}\mathbb{N})$
$Ef = (t=t')$
$Uf = (c = \sum_{e \in t'} 1)$
$\qquad\qquad felt(e)$

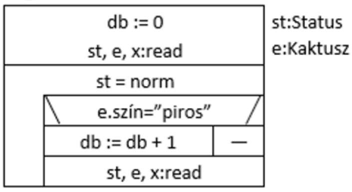


t:enor(E)	~	x:infile(Kaktusz) (st,e,x:read)
felt(e)	~	e.szín="piros"
c	~	db

Specifikáció:

$A = ($ x:infile(Kaktusz), db:$\mathbb{N}$ $)$
Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)
$Ef = (x=x_0)$
$Uf = (db = \sum_{e \in x_0} 1)$
$\qquad$ e.szín="piros"

Algoritmus:

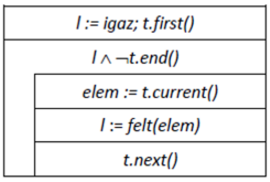


b) Igaz-e, hogy minden kaktusz virága piros?

P.t.: Optimista lineáris keresés

Specifikáció:

$A = (t{:}enor(E), l{:}\mathbb{L}, elem{:}E)$
$Ef = (t=t')$
$Uf = (l, elem, t = \forall \mathbf{SEARCH}_{e \in t'} felt(e))$

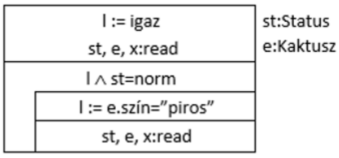


t:enor(E)	~	x: infile(Kaktusz) (st,e,x:read)
felt(e)	~	e.szín="piros"

Specifikáció:

$A = ($ x:infile(Kaktusz), l:$\mathbb{L}$ $)$
Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)
$Ef = (x=x_0)$
$Uf = (l = \forall\mathbf{SEARCH}_{e \in x_0}$ e.szín="piros" $)$

Algoritmus:

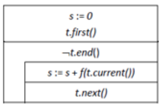


c) Válogassuk ki egy szekvenciális outputfájlba a piros virágú kaktuszok neveit!

P.t.: *Összegzés* (kiválogatás)

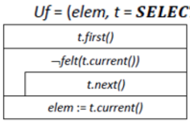
Specifikáció:

$A = (t{:}enor(E), s{:}H)$
$Ef = (t=t')$
$Uf = (s = \sum_{e \in t'} f(e))$



Specifikáció:

$A = (t{:}enor(E), elem{:}E)$
$Ef = (t=t' \land \exists i \in [1..|t|]: felt(t_i))$
$Uf = (elem, t = \mathbf{SELECT}_{e \in t'} felt(e))$

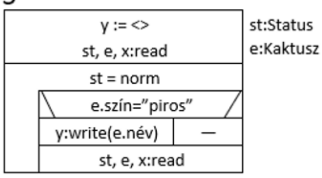


t:enor(E)	~	x:infile(Kaktusz) (st,e,x:read)
f(e)	~	<e.név> ha e.szín="piros"
s	~	y
H, +, 0	~	$\mathbb{S}^*$, $\oplus$, <>

Specifikáció:

$A = ($ x:infile(Kaktusz), y:outfile($\mathbb{S}$) $)$
Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)
$Ef = (x=x_0)$
$Uf = (y = \bigoplus_{e \in x_0} <e.név>)$
$\qquad$ e.szín="piros"

Algoritmus:



d) Válogassuk ki egy szekvenciális outputfájlba a piros virágú kaktuszok, egy másikba a mexikói őshazájú kaktuszok neveit!

P.t.: *2 sima- vagy 1 duplaösszegzés* (kiválogatás)

t:enor(E)	~ x:infile(Kaktusz) (st,e,x:read)
$f_1(e)$	~ <e.név> ha e.szín="piros"
$f_2(e)$	~ <e.név> ha e.ős="Mexikó"
s	~ y, z
H, +, 0	~ $\mathbb{S}^*$, $\oplus$, <>

Specifikáció:

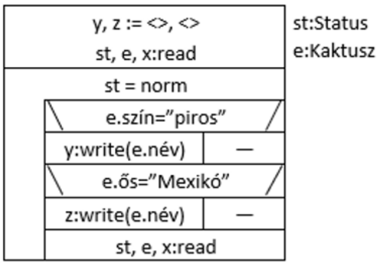
A = (x:infile(Kaktusz), y, z :outfile($\mathbb{S}$))

Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)

Ef = ($x=x_0$)

Uf = ($y=\bigoplus_{\substack{e\in x0 \\ e.szín=\text{"piros"}}}$<e.név> $\wedge$ $z=\bigoplus_{\substack{e\in x0 \\ e.ős=\text{"Mexikó"}}}$<e.név>)

Algoritmus:



LEGNAGYOBB SZÁM ÉS PÁROS KERESÉS

P.t.: *Két összegzés (linker és maxkiv átalakítva)*

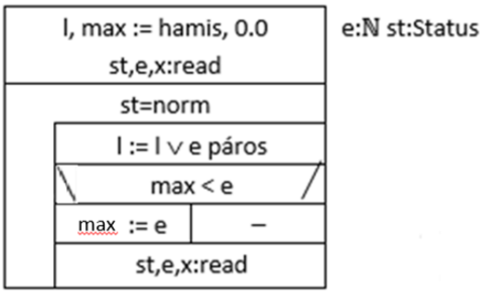
t:enor(E)	~ x:infile($\mathbb{N}$) (st,e,x:read)
f(e)	~ e páros, e
s	~ l, max
H, +, 0	~ ($\mathbb{L}$, $\vee$, hamis), ($\mathbb{N}$, max, 0)

Specifikáció:

A = (x:infile($\mathbb{N}$), l:$\mathbb{L}$, max:$\mathbb{N}$)

Ef = ($x=x' \wedge |x|\geq 1$)

Uf = (l = $\mathbf{SEARCH}_{e\in x'}$ (e páros) $\wedge$ max = $\mathbf{MAX}_{e\in x'}$ e)



BEVÉTEL <- SZÁMLA: VÁSÁRLÓ NEVE, TERMÉK(EK) ÉS ÁR(AK)

P.t.: *Összegzés*

t:enor(E)	~ x:infile(Számla) (st,sz,x:read)
f(e)	~ sz.össz
c	~ bevét

Specifikáció:

A = (x:infile(Számla), bevét:$\mathbb{N}$)

Számla=rec(név:$\mathbb{S}$, lista:Áru*)

Áru = rec(cikkszám:$\mathbb{S}$, ár:$\mathbb{N}$)

vagy

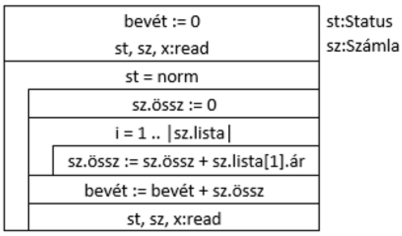
Számla=rec(név:$\mathbb{S}$, lista:Áru*, össz: $\mathbb{N}$) ahol össz = $\sum_{i=1}^{|lista|}$ lista[i].ár

Összegzés

t:enor(E)	~ i = 1 .. |lista|
f(e)	~ lista[i].ár
c	~ össz

Ef = ($x=x_0$)

Uf = (bevét = $\sum_{sz\in x_0}$ sz. össz)



Egy számla összesítését bízzuk a beolvasásra.

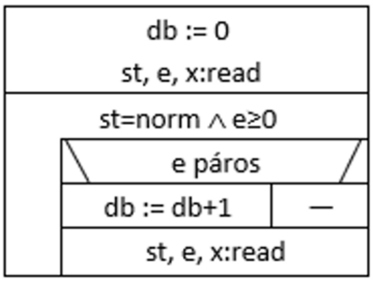
V.

EGÉSZ SZÁMOKAT TARTALMAZÓ SZEK.INPUTFÁJL

a) Hány páros szám előzi meg az első negatívat?

P.t.: *Számlálás, feltétel fennállásáig*

t:enor(E) ~ x:infile(ℤ) (st,e,x:read) amíg: e≥0
felt(e) ~ e páros
c ~ db



e:ℤ
st:Status

Specifikáció:

A = (x:infile(ℤ), db:ℕ)
Ef = ($x=x_0$)
Uf = ($db = \sum_{\substack{e\in x_0 \\ e\ \text{páros}}}^{e\geq 0} 1$)

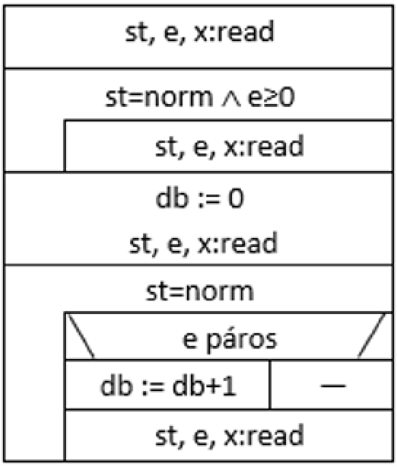
Megjegyzés: A specifikációban a programozási tétel kulcs szava (Σ) feletti extra feltétel jelzi, hogy meddig tartson a felsorolás.

b) Hány páros szám követi az első negatív számot?

P.t.: *Kiválasztás + Számlálás*

t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
felt(e) ~ st=abnorm ∨ e<0
c ~ db

t:enor(E) ~ x:infile(ℤ) „folytatása"
felt(e) ~ e páros
c ~ db



e:ℤ
st:Status

Specifikáció:

A = (x:infile(ℤ), db:ℕ)
Ef = ($x=x_0$)
Uf = ($(l, e', (st',e',x')) = \textbf{SEARCH}_{e\in x0}(e<0) \wedge \ db = \sum_{\substack{e\in x' \\ e\ \text{páros}}} 1) =$

$= ((e', (st',e',x')) = \textbf{SELECT}_{st,e\in x0}(st=abnorm \vee e<0) \wedge \ db = \sum_{\substack{e\in x' \\ e\ \text{páros}}} 1)$

c) Hány páros szám van az első negatív számot megelőzően, és hány azt követően?

P.t.: *Számlálás, feltétel fennállásáig + Számlálás*

t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
amíg: e≥0
felt(e) ~ e páros
c ~ dbe

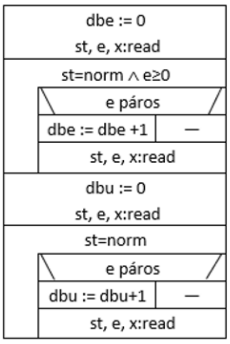
t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
first() helyett next()
felt(e) ~ e páros
c ~ dbu

Specifikáció:

A = (x:infile(ℤ), dbe, dbu:ℕ)

Ef = ($x=x_0$)

Uf = ((dbe, (st',e',x')) = $\sum_{e\in x_0}^{e\geq 0} 1$ ∧ dbu = $\sum_{e\in x'} 1$)
e páros e páros



e:ℤ
st:Status

Megjegyzés:
A második számlálás egy félbehagyott felsorolást (st',e',x') folytat úgy, hogy olvassa a következő elemet. Ezért a ciklus előtti read itt nem a first(), hanem a next() művelet.

d) Hány páros szám van az első negatív számot megelőzően, és hány azt követően azzal együtt?

P.t.: *Számlálás, feltétel fennállásáig + Számlálás*

t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
amíg: e≥0
felt(e) ~ e páros
c ~ dbe

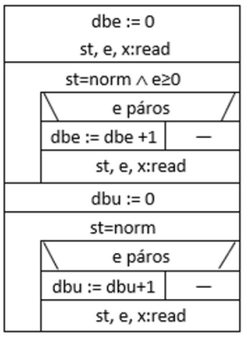
t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
first() nélkül
felt(e) ~ e páros
c ~ dbu

Specifikáció:

A = (x:infile(ℤ), dbe, dbu:ℕ)

Ef = ($x=x_0$)

Uf = ((dbe, (st',e',x')) = $\sum_{e\in x_0}^{e\geq 0} 1$ ∧ dbu = $\sum_{e\in(e',x')} 1$)
e páros e páros



e:ℤ
st:Status

Megjegyzés:
A második számlálás úgy folytat egy félbehagyott felsorolást (st',e',x'), hogy első két az utoljára beolvasott elemet (feltéve, hogy nem értünk a fájl végére) dolgozza fel (erre utal a specifikációban az e∈(e',x') szimbólum), ezért nincs szüksége előre olvasásra.

TENGERSZINT FELETTI MAGASSÁG, LEGMAGASABB HORPADÁS

P.t.: *Felt. max. ker.*

t:enor(E) ~ t:enor(ℝ×ℝ×ℝ),
ahol (e, a, k) := t.current()
f(e) ~ a
felt(e) ~ e>a<k
H, > ~ ℤ, >

Specifikáció:

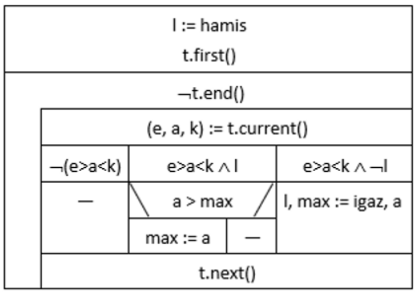
A = (x:infile(ℝ), l:𝕃, max:ℝ)

Ef = ($x=x_0$)

A = (t:enor(ℝ×ℝ×ℝ), l:𝕃, max:ℝ)

Ef = ($t=t_0$)

Uf = ((l, max) = $\mathbf{MAX}_{(e,a,k)\in t0}$ a)
e>a<k



e, a, k: ℝ

Felsoroló

t:enor(ℝ×ℝ×ℝ)

(ℝ×ℝ×ℝ)*	first()	next()	current() : ℝ×ℝ×ℝ	end() : 𝕃
x:infile(ℝ) e, a, k: ℝ st : Status	st, e, x:read st, a, x:read st, k, x:read	e := a a := k st, k, x:read	**return** (e, a, k)	**return** st=abnorm

HORGÁSZVERSENY

Részfeladat: **ok := jó(e.fogás)**

jó : Fogás* → $\mathbb{L}$

A = (x:Fogás*, ok:$\mathbb{L}$)

Ef = ($x=x_0$)

Uf = ($x=x_0 \ \wedge\ i' = \textbf{SELECT}_{i\in[1\,..\,|x|]}\,((x[i].hal=\text{"ponty"} \wedge x[i].súly\geq 1.0) \vee i>|x|) \ \wedge\ db = \sum_{\substack{i\in[i'\,..\,|x|] \\ x[i].hal=\text{"harcsa"} \wedge x[i].hossz\geq 1.0}} \mathbf{1} \ \wedge\ ok = db\geq 4$)

Kiválasztás

t:enor(E) ~ i∈[1 .. |x|]

felt(e) ~ (x[i].hal="ponty" ∧ x[i].súly≥1.0) ∨ i>|x|

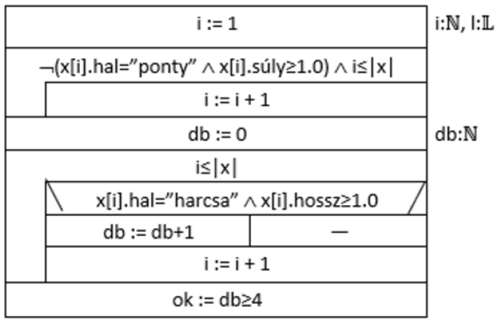
Számlálás

t:enor(E) ~ i∈[i' .. |x|]

felt(e) ~ x[i].hal="harcsa" ∧ x[i].hossz≥1.0

c ~ db

ok := jó(x)



MELYIK SZÁMBÓL HÁNY

P.t.: *Összegzés (másolás)*

f(e) ~ <e>

s ~ y

H, +, 0 ~ Össz*, ⊕, <>

Specifikáció:

A = (x:infile($\mathbb{Z}$), y:outfile(Össz))

Össz = rec(szám:$\mathbb{Z}$, db:$\mathbb{N}$)

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

($x\nearrow$ azt jelzi, hogy az x növekedően rendezett)

Ötlet:

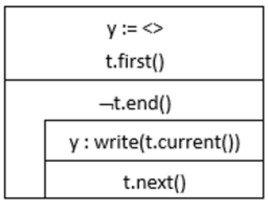
Ha lenne egy olyan felsorolónk, amelyik az eredményt, az összesítéseket tartalmazó rekordokat tudná felsorolni, akkor elég lenne ezeket az output fájlba átmásolni.

Új Specifikáció:

A = (t:enor(Össz), y:outfile(Össz))

Ef = ($t = t_0$)

Uf = ($y = t_0$) = ($y = \bigoplus_{e\in t0} \langle e\rangle$)



Felsoroló:

t:enor(Össz)   Össz = rec(szám:$\mathbb{Z}$, db:$\mathbb{N}$)

Össz*	first()	next()	current() : Össz	end() :𝕃
x : infile(ℤ) dx : ℤ sx : Status akt : Össz vége : 𝕃	sx,dx,x:read next()	lásd külön	**return** akt	**return** vége

Megj:

Az összegzésnek két eredménye van: a darabszám (akt.db); és a felsoroló aktuális állapota, amelyet az sx,dx,x változók értékei írnak le a next() művelet végén.

next() művelet

A = (x:infile($\mathbb{Z}$), dx:$\mathbb{Z}$, sx:Status, akt:Össz, vége:$\mathbb{L}$)

Ef = ($x = x' \wedge x\nearrow \wedge dx = dx' \wedge sx = sx'$)

Uf = (vége = (sx'=abnorm) ∧ (¬vége → akt.szám=dx' ∧ (akt.db, (sx,dx,x))=$\sum_{dx\in(dx',x')}^{dx\,=\,akt.szám} 1$))

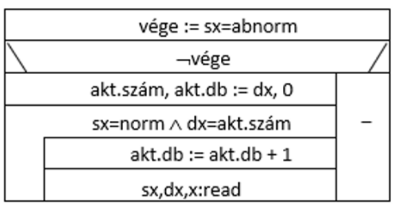
Összegzés (megszámolás)

t:enor(E) ~ x:infile($\mathbb{Z}$) (sx,dx,x:read)
first() nélkül, felt: dx=akt.szám

f(e) ~ 1

s ~ akt.db

H, +, 0 ~ $\mathbb{N}$, +, 0



+Órai feladat:

Egy szekvenciális inputfájlban napi átlaghőmérsékleteket tárolunk. Mennyi az első fagypont alatti értéket megelőző napok (ilyenek biztosan vannak) hőmérsékleteinek átlaga, és mondjuk meg, hogy a többi napon (beleértve az első fagypont alatti napot is, amely biztosan létezett) vajon minden nap fagypont alatt maradt-e a hőmérséklet, és mi volt a legalacsonyabb hőmérséklet?

Két összegzés, feltétel fennállásáig tartanak

t:enor(E) ~ x:infile(ℝ) (st,e,x:read)
amíg: e≥0

f(e) ~ e, 1

S ~ s, db

H, +, 0 ~ (ℝ, +, 0.0), (ℕ +, 0)

Két összegzés (optker és minkiv átalakítva)

t:enor(E) ~ x:infile(ℝ) (st,e,x:read)
first() nélkül

f(e) ~ e<0, e

S ~ l, min

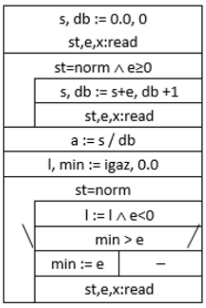
H, +, 0 ~ (𝕃, ∧, igaz), (ℝ, min, 0)

Specifikáció:

A = (x:infile(ℝ), a:ℝ, l:𝕃, min:ℝ)

Ef = ($x=x_0 \wedge |x|\geq 2 \wedge x[1]\geq 0 \wedge \exists i\in[2..|x|]: x[i]<0$)

Uf = ($((s, db), (st',e',x')) = \sum_{e\in x_0}^{e\geq 0}(e, 1) \wedge a=s/db \wedge$
$\wedge\ l = \forall\textbf{SEARCH}_{e\in(e',x')}(e<0) \wedge min = \textbf{MIN}_{e\in(e',x')}\ e$)



e, s:ℝ, db:ℕ
st:Status

Két összegzés, feltétel fennállásáig tartanak

T:enor(E) - x:infile(R) (st,e,x:read)
Amig e >=0

F(e) - e, 1

S - s, db

H,+,0 - (R,+, 0.0), (N , +, 0)

Két összegzés (optker és minkiv átalakitva)

T:enor(E) - x:infile(R) (st,e,x:read)
First() nélkül

F(e) - e <0 , e

S - l , min

H,+,0 - (L,és,igaz), (R,min, 0)

Specifikáció:

A= (x:infile(R), a:R,l:L,min:R)

Ef=(x=x' és |x| >= 2 és x[1] >=0 és életik i€[2..|x|] : X[i] < 0)

Uf = (((s,db), (st',e',x'))= szumma e >=0 e € x' (e,1) és a=s/db és
L = minden SERACH e € (e',x') (e<0) és min =MIN e € (e',x') e)