3. Gyűjtemények I.

1. Zsák típus

Valósítsuk meg egy adott halmaz (E) elemeit tartalmazó zsák típusát úgy, hogy nincs felső korlát a zsákba bekerülő elemek számára. A szokásos (üres-e, betesz, kivesz, hányszor van benn egy szám) műveletek mellett szükségünk lesz a leggyakoribb elemet lekérdező műveletre is.

Típusspecifikáció: **Bag**

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

A típus reprezentálására több lehetőség is van:

- Tárolhatjuk az elemeket rendezetlenül egy sorozatban (ahol ugyanaz az elem többször is szerepelhet), esetleg rendezetten.
- Tárolhatjuk az elemeket az előfordulási számukkal együtt egy sorozatban, esetleg az elemek szerint rendezetten.

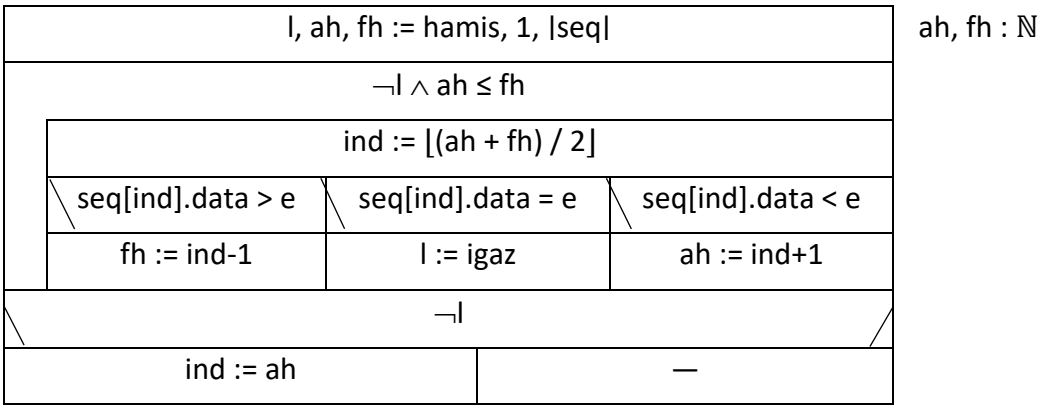
Elem szerinti logaritmikus keresés [elég csak megemlíteni, a leírását a hallgatókkal megosztani]

A = (seq : Pair*, e : E, l : $\mathbb{L}$, ind : $\mathbb{N}$) Pair = rec(data: E, count: $\mathbb{N}$)

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

Uf = (Ef $\wedge$ l = $\exists i \in [1 .. |seq|]$: seq[i].data = e $\wedge$
(l $\rightarrow$ ind$\in$[1 .. |seq|] $\wedge$ seq[ind].data = e) $\wedge$
($\neg$l $\rightarrow$ $\forall i \in$ [1..ind-1]: seq[i].data < e $\wedge$ $\forall i \in$[ind.. |seq|] : seq[i].data > e))

l, ind := logSearch (seq, e)



Típusmegvalósítás:

seq : Pair*

maxind : ℕ

ahol

Pair =
rec(data: E, count: ℕ)

Invariáns:

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

l := Empty(b) b : Bag, l : 𝕃

l := |seq|=0

c := Multipl(b, e) b : Bag, e : E, c : ℕ

l, ind := logSearch(seq, e) **if** l **then** c :=seq[ind].count **endif**

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

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

b := SetEmpty(b) b : Bag

seq := <>

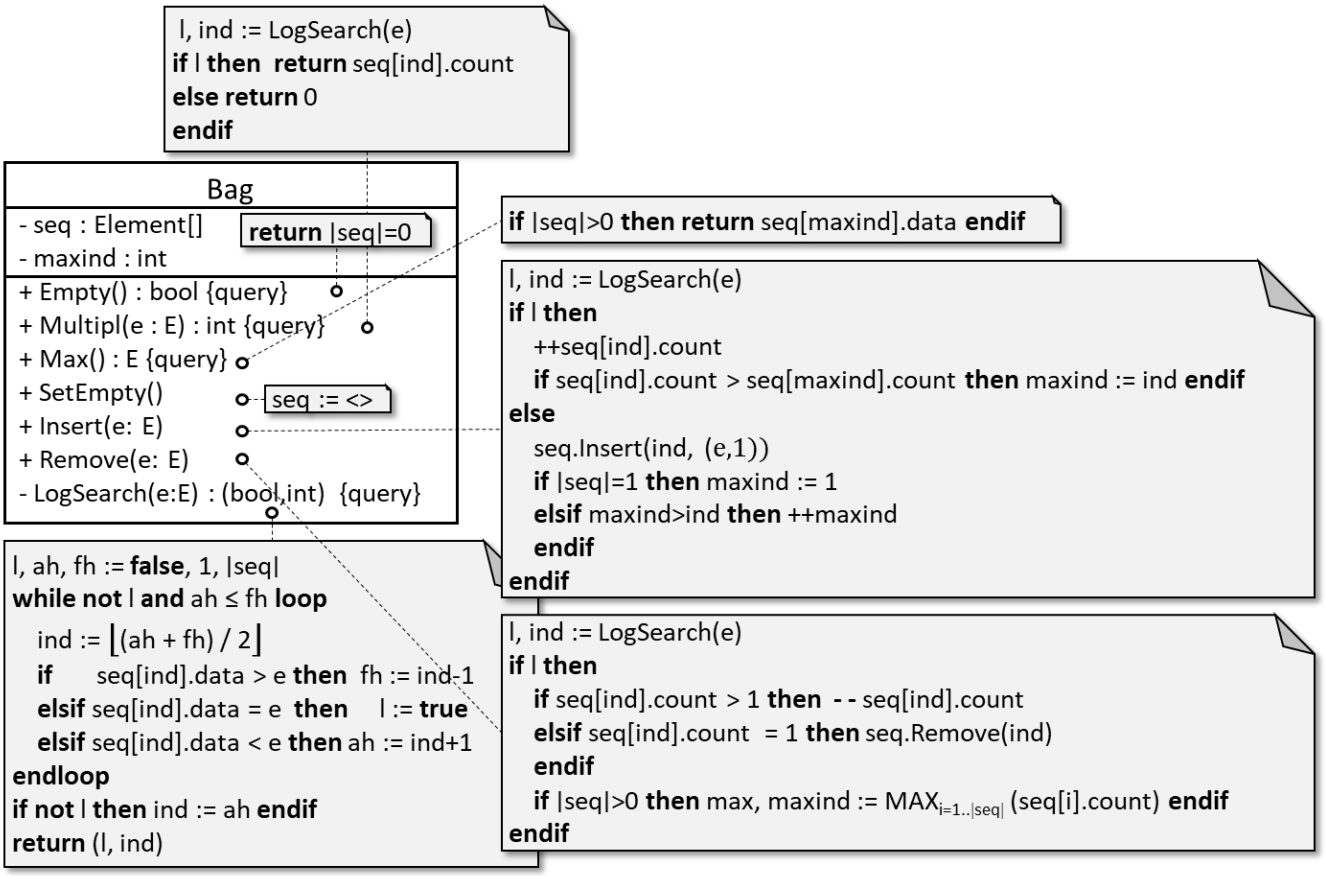
b := Insert(b,e) b : Bag, e : E

l, ind := logSearch(seq, e)				
l				
++seq[ind].count		seq := seq.Insert(ind, (e,1))		
seq[ind].count > seq[maxind].count		|seq|=1	|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.Remove(ind)	–
|seq|>0			
max, maxind := $MAX_{i=1..\|seq\|}$ (seq[i].count)	–		

Osztály:


l, ind := LogSearch(e)
if l then return seq[ind].count
else return 0
endif
Bag
- seq : Element[]
- maxind : int
return |seq|=0
+ Empty() : bool {query}
+ Multipl(e : E) : int {query}
+ Max() : E {query}
+ SetEmpty()
+ Insert(e: E)
+ Remove(e: E)
- LogSearch(e:E) : (bool,int) {query}
seq := <>
if |seq|>0 then return seq[maxind].data endif
l, ind := LogSearch(e)
if l then
++seq[ind].count
if seq[ind].count > seq[maxind].count then maxind := ind endif
else
seq.Insert(ind, (e,1))
if |seq|=1 then maxind := 1
elsif maxind>ind then ++maxind
endif
endif
l, ah, fh := false, 1, |seq|
while not l and ah ≤ fh loop
ind := ⌊(ah + fh) / 2⌋
if seq[ind].data > e then fh := ind-1
elsif seq[ind].data = e then l := true
elsif seq[ind].data < e then ah := ind+1
endloop
if not l then ind := ah endif
return (l, ind)
l, ind := LogSearch(e)
if l then
if seq[ind].count > 1 then - - seq[ind].count
elsif seq[ind].count = 1 then seq.Remove(ind)
endif
if |seq|>0 then max, maxind := MAX i=1..|seq| (seq[i].count) endif
endif

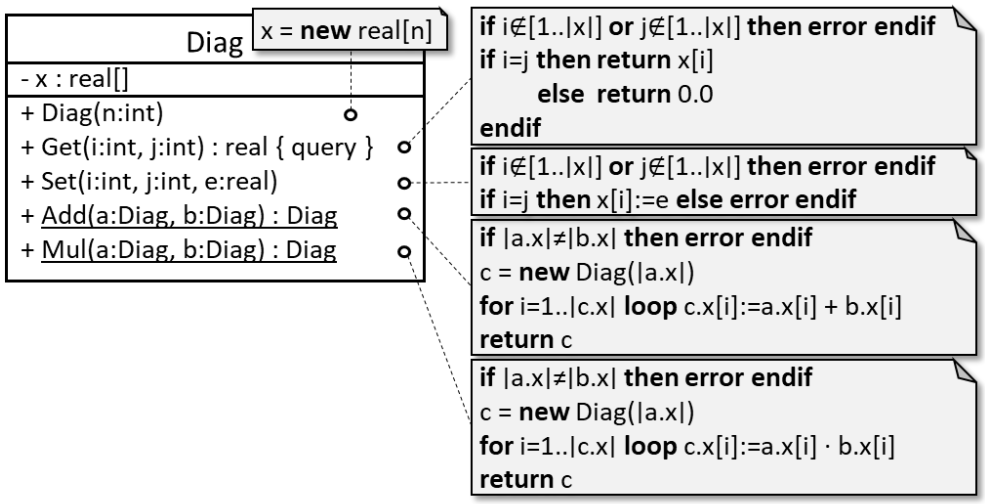
2. Diagonális mátrix

Valósítsuk meg a diagonális mátrix típust (az ilyen mátrixoknak csak a főátlójukban lehetnek nullától eltérő elemek)! Ilyenkor elegendő csak a főátlóbeli elemeket tárolni egy sorozatban. Implementáljuk a mátrix i-edik sorának j-edik elemét lekérdező, illetve megváltoztató műveleteket, valamint két mátrix összegét és szorzatát kiszámoló műveleteket!

Típusdefiníció: **Diag** // dim(a) ~ egy 'a' mátrix sorainak száma, dim(a)≥ 1

olyan négyzetes mátrixok, amelyek a főátlójukon kívül csak nullákat tárolnak	e := a[i,j] (a : Diag, i,j : [1.. dim(a)], e : $\mathbb{R}$)
	a[i,j] := e (a : Diag, i,j : [1.. dim(a)], e : $\mathbb{R}$) // i=j
	c := a + b (a, b, c : Diag) // dim(a)=dim(b)=dim(c)
	c := a · b (a, b, c : Diag) // dim(a)=dim(b)=dim(c)
x : $\mathbb{R}^n$ Invariáns: - 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 not (a.n = b.n = c.n) **then error endif** **for** i=1 .. a.n **loop** c.x[i]:=a.x[i]+b.x[i] **endloop**
	if not (a.n = b.n = c.n) **then error endif** **for** i=1 .. a.n **loop** c.x[i]:=a.x[i]·b.x[i] **endloop**

Osztály diagram:



A kódolásnál több konstruktort is be lehet vezetni, amelyekben a típusinvariánsról mindig gondoskodni kell.

A bevezetett metódusnevekre nincs szükség, ha a Get() és Set() helyett a C# „indexer"-ét használjuk; az Add() és Mul() helyett pedig operátor felüldefiniálást.

3. Alsóháromszög mátrix

Valósítsuk meg az alsó háromszög mátrix típust (a mátrixok a főátlójuk felett csak nullát tartalmaznak)! Ilyenkor elegendő csak a főátló és az alatti elemeket reprezentálni egy sorozatban. Implementáljuk a mátrix *i*-edik sorának *j*-edik elemét *lekérdező, illetve megváltoztató* műveletet, valamint két mátrix összegét és szorzatát!

Típusspecifikáció: **AHM**

olyan valós számokat tartalmazó legalább 1×1-es négyzetes mátrixok, amelyek a főátlójuk felett csak nullákat tartalmaznak	e := a[i,j] (a : AHM, i,j : [1.. dim(a)], e : $\mathbb{R}$)
	a[i,j] := e (a : AHM, i,j : [1.. dim(a)], e : $\mathbb{R}$) // i ≥ j
	c := a + b (a, b, c : AHM) // dim(a)=dim(b)=dim(c)
	c := a · b (a, b, c : AHM) // dim(a)=dim(b)=dim(c)

7×7-es alsóháromszög mátrix:

34						
-3	42					
6	3	8				
9	11	0	4			
5	23	-5	7	15		
53	22	72	36	0	34	
84	60	-7	0	57	48	89

A mátrix alsóháromszög részének elemeit sorfolytonosan felsorolva:

1.	34	(1,1)
2.	-3	(2,1)
3.	42	(2,2)
4.	6	(3,1)
5.	3	(3,2)
…	…	…
27	48	(7,6)
28	89	(7,7)

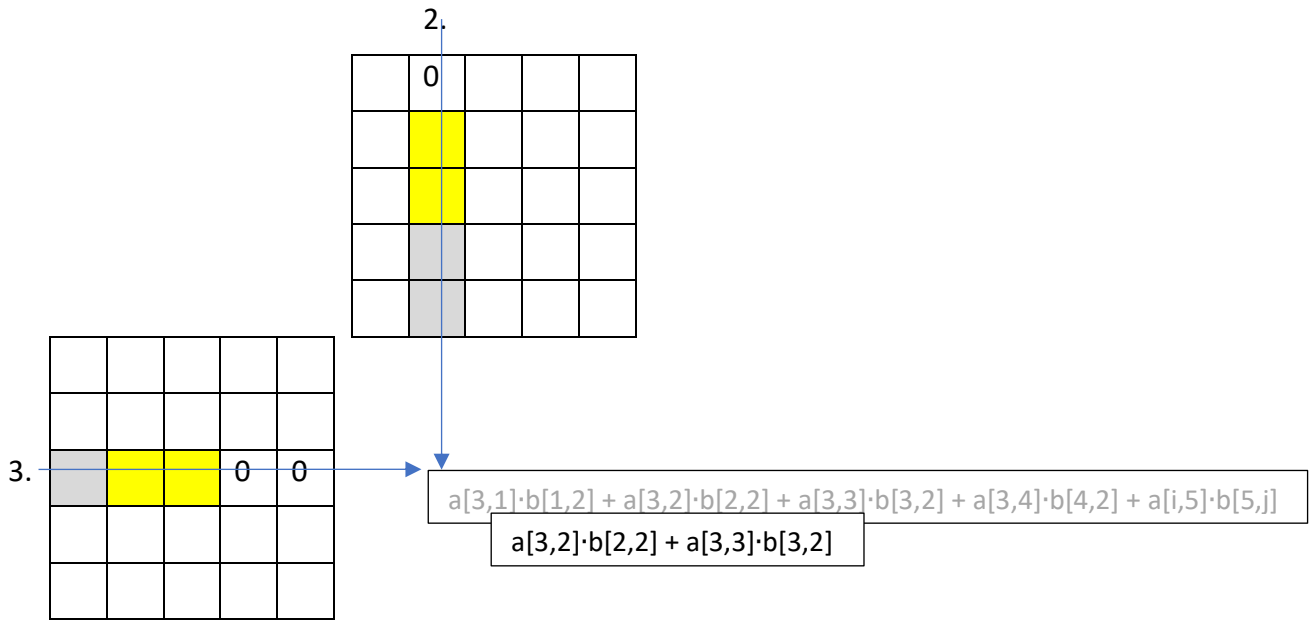
Kell egy index függvény, amely egy mátrixbeli elem indexeihez hozzárendeli az elem tárolási helyének indexét az egydimenziós tömbben 1-től, illetve 0-tól indexelt tömbök esetében:

$ind(i,j) = j + \sum_{k=1}^{i-1} k = j + \frac{i\cdot(i-1)}{2}$, illetve $ind(i,j) = j + \sum_{k=1}^{i} k = j + \frac{i\cdot(i+1)}{2}$

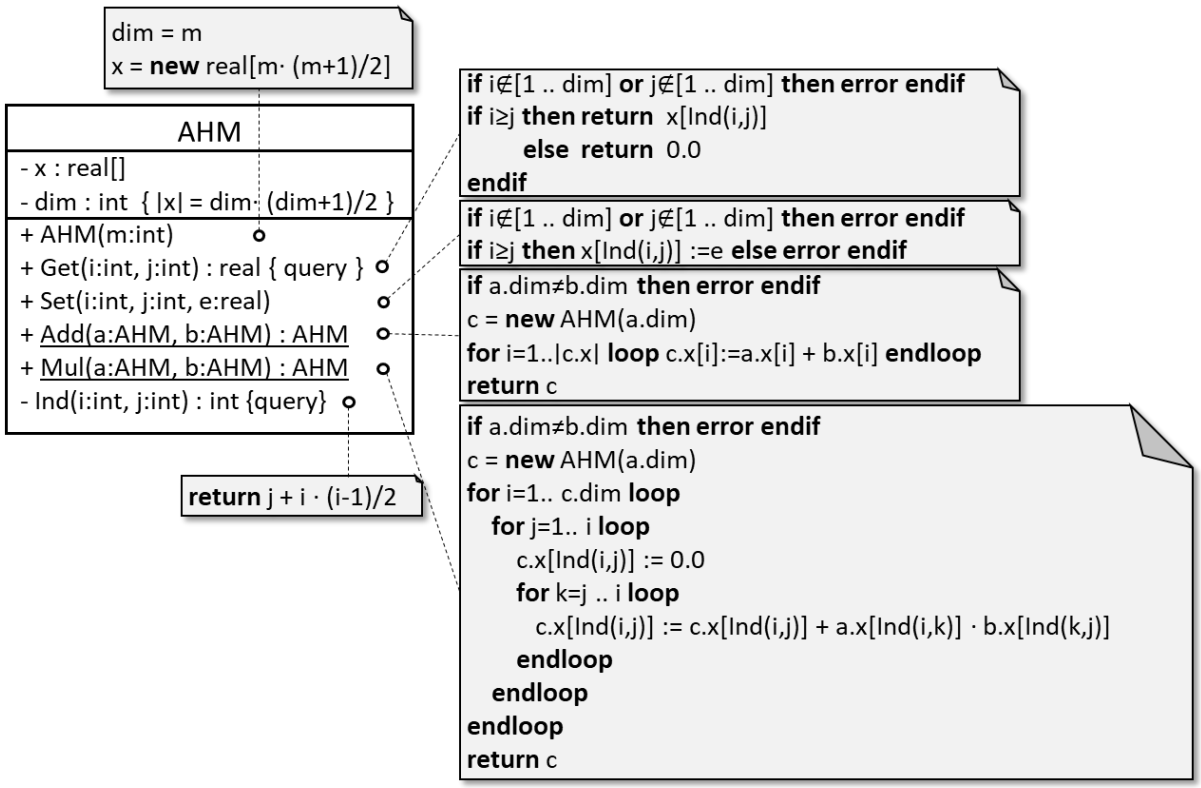
Típusmegvalósítás:

x : $\mathbb{R}^n$ **dim : $\mathbb{N}$** // Invariáns: - dim ≥ 1 - n = dim· (dim+1)/2	**if** i≥j **then** e := a.x[ind(i,j)] **else** e := 0.0 **endif**
	if i≥j **then** a.x[ind(i,j)] := e **else error endif**
	if not (a.dim = b.dim = c.dim) **then error endif** **for** i=1 .. c.n **loop** c.x[i] := a.x[i]+b.x[i] **endloop**

	if not (a.dim = b.dim = c.dim) **then error endif**
	for i=1 .. c.dim **loop**
	  for j=1 .. i **loop**
	  c.x[ind(i,j)]:= $\sum_{k=j}^{i}$(a.x[ind(i,k)] · b.x[ind(k,j)])
	  endloop
	endloop



Osztály diagram:



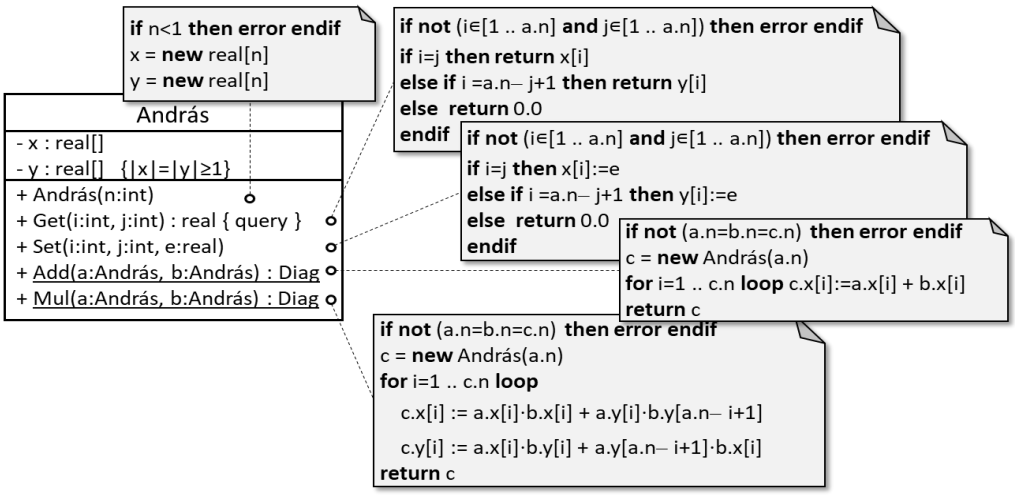
4. Andráskereszt mátrix

   Valósítsuk meg az andráskereszt mátrix típust. Ezek olyan négyzetes mátrixok, amelyeknek csak a fő- és mellékátlójukban lehetnek nullától eltérő elemek. Ilyenkor elegendő csak a fő- és mellékátlóbeli elemeket eltárolni. Implementáljuk a mátrix i-edik sorának j-edik elemét lekérdező, illetve megváltoztató műveleteket, valamint két mátrix összegét és szorzatát kiszámoló műveleteket!

   Típusdefiníció: **András**

olyan valós számokat tartalmazó legalább 1×1-es négyzetes mátrixok, amelyek a két átlójukon kívül csak nullákat tartalmaznak	e := a[i,j] (a : András, i,j : [1.. dim(a)], e : $\mathbb{R}$)
	a[i,j] := e (a : András, i,j : [1.. dim(a)], e : $\mathbb{R}$) // i=j
	c := a + b (a, b, c : András) // dim(a)=dim(b)=dim(c)
	c := a · b (a, b, c : András) // dim(a)=dim(b)=dim(c)
x, y:$\mathbb{R}^n$ // Invariáns: n ≥ 1	**if not** (i∈[1 .. a.n] **and** j∈[1 .. a.n]) **then error endif** **if** i = j **then** e := a.x[i] **else if** i =a.n– j+1 **then** e := a.y[i] **else** e := 0.0 **endif**
	if not (i∈[1 .. a.n] **and** j∈[1 .. a.n]) **then error endif** **if** i=j **then** a.x[i] := e **else if** i=a.n– j+1 **then** a.y[i] := e **else error endif**
	if not (a.n=b.n=c.n) **then error endif** **for** i=1 .. c.n **loop** c.x[i], c.y[i] := a.x[i] + b.x[i], a.y[i] + b.y[i] **endloop**
	if not (a.n=b.n=c.n) **then error endif** **for** i=1..c.n **loop** c.x[i] := a.x[i]·b.x[i] + a.y[i]·b.y[|a.x|– i+1] c.y[i] := a.x[i]·b.y[i] + a.y[|a.x|– i+1]·b.x[i] **endloop**

Osztály diagram:

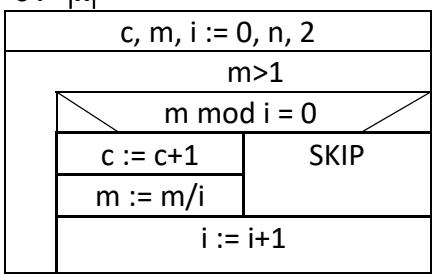


5. Prímek halmazai. Ábrázoljuk az ilyen halmazokat a bennük levő számok szorzatával; az üres halmazt az 1-gyel.

Típusdefiníció: PrimSet

prímszámok véges halmazai	l := p ∈ h (h : PrimSet, p : ℙ, l : 𝕃)
	h := h ∪ p (h : PrimSet, p : ℙ) // ha p ∈ h, akkor hatástalan
	h := h \ p (h : PrimSet, p : ℙ) // ha p ∉ h, akkor hatástalan
	l := h = ∅ (h : PrimSet, l : 𝕃)
	h := ∅ (h : PrimSet)
	c := |h| (h : PrimSet, c : ℕ)
n : ℕ ahol az n számnak a prímtényezős felbontásában a prímek egyszeresen fordulnak elő	l := (n mod p=0)
	if n mod p ≠ 0 **then** n := n·p **endif**
	if n mod p = 0 **then** n := n/p **endif**
	l := (n=1)
	n := 1
	lásd külön

c := |h|



Osztály: