2. *Sorozattal reprezentált típusok*

Olyan típusok definiálása, majd osztály diagrammal történő leírása, ahol a reprezentációhoz elemek sorozatára van szükségünk.

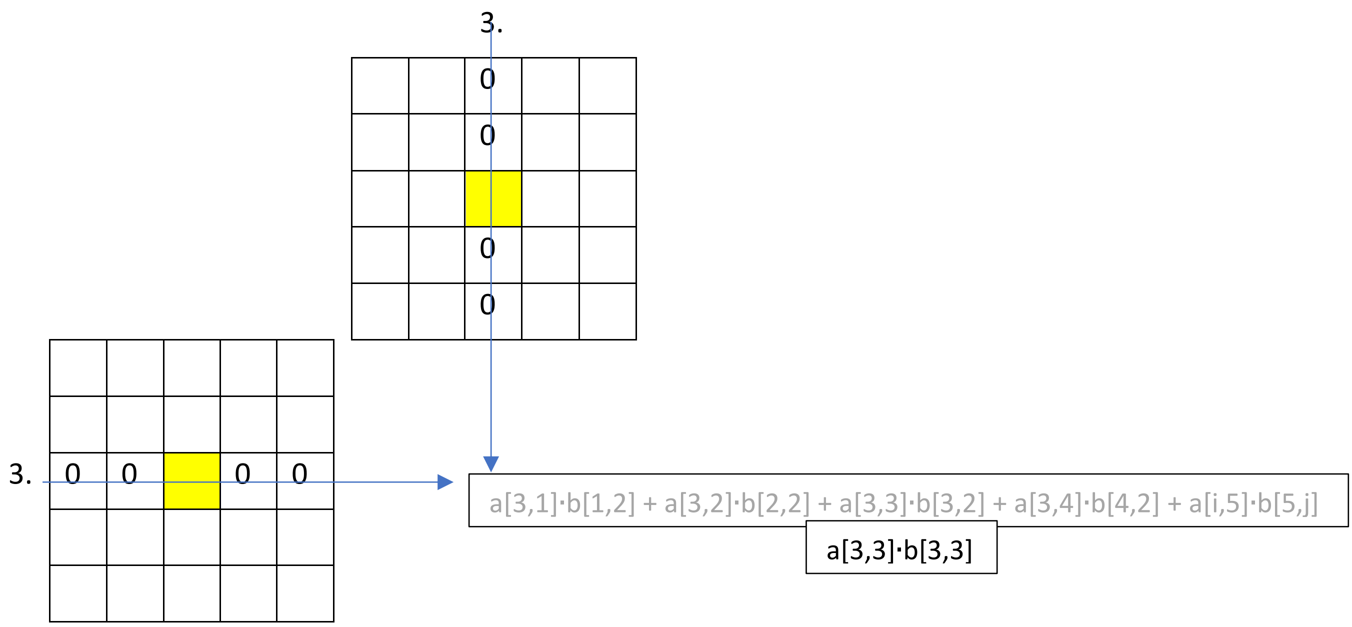
1. Diagonális mátrix
   Valósítsuk meg a diagonális mátrix típust (az ilyen mátrixoknak csak a főátlójukban lehetnek nullától eltérő elemek)! Ilyenkor elegendő csak a főátlóbeli elemeket tárolni egy sorozatban. Implementáljuk a mátrix i-edik sorának j-edik elemét lekérdező, illetve megváltoztató műveleteket,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

   Típusdefiníció:

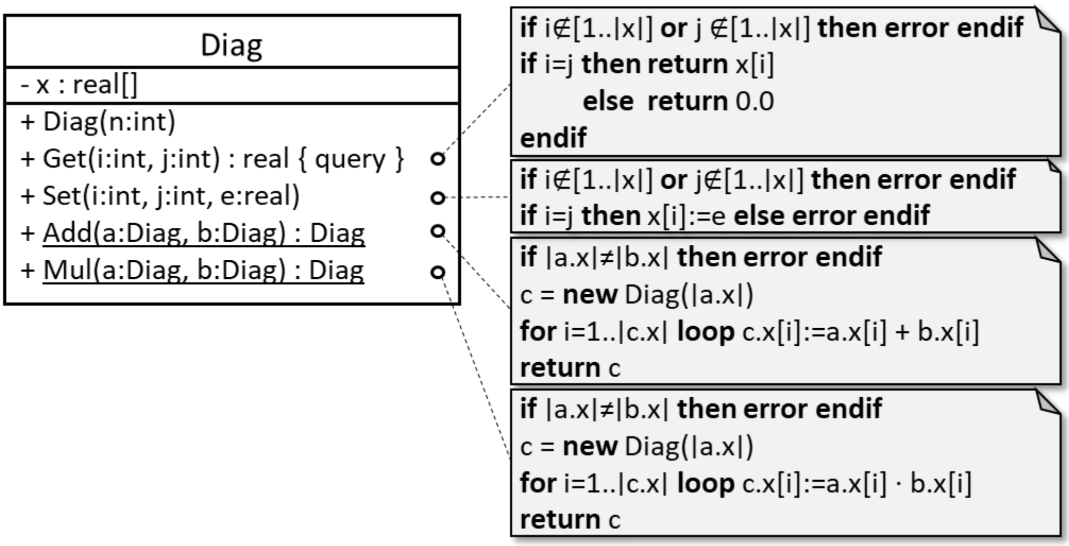
<table>
<tr><td rowspan="4">Diag(ℝ)

// ha a:Diag(ℝ) mátrix mérete: dim(a)×dim(a) akkor dim(a)≥1</td><td>e := a[i,j] (a : Diag(ℝ), i,j : [1..a.n], e : ℝ)</td></tr>
<tr><td>a[i,j] := e (a : Diag(ℝ), i,j : [1..a.n], e : ℝ) // i=j</td></tr>
<tr><td>c := a + b (a, b, c : Diag(ℝ)) // dim(a)=dim(b)=dim(c)</td></tr>
<tr><td>c := a · b (a, b, c : Diag(ℝ)) // dim(a)=dim(b)=dim(c)</td></tr>
<tr><td rowspan="4">x:ℝ*

// |x| ≥ 1</td><td>if i=j then e := a.x[i] else e := 0.0 endif</td></tr>
<tr><td>if i=j then a.x[i] := e else error endif</td></tr>
<tr><td>if |a.x|=|b.x|=|c.x| then
∀i∈[1..|a.x|]: c.x[i]:=a.x[i] + b.x[i]</td></tr>
<tr><td>if |a.x|=|b.x|=|c.x| then
∀i∈[1..|a.x|]: c.x[i]:=a.x[i]·b.x[i] [1.kvíz]</td></tr>
</table>



Osztály diagram:



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

2. 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!

7×7-es alsóháromszög mátrix (a biztosan nulla elemeket nem jelöljük):

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 (a biztosan nulla elemek nélkül) sorfolytonosan helyezzük el egy egydimenziós tömbben:

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.

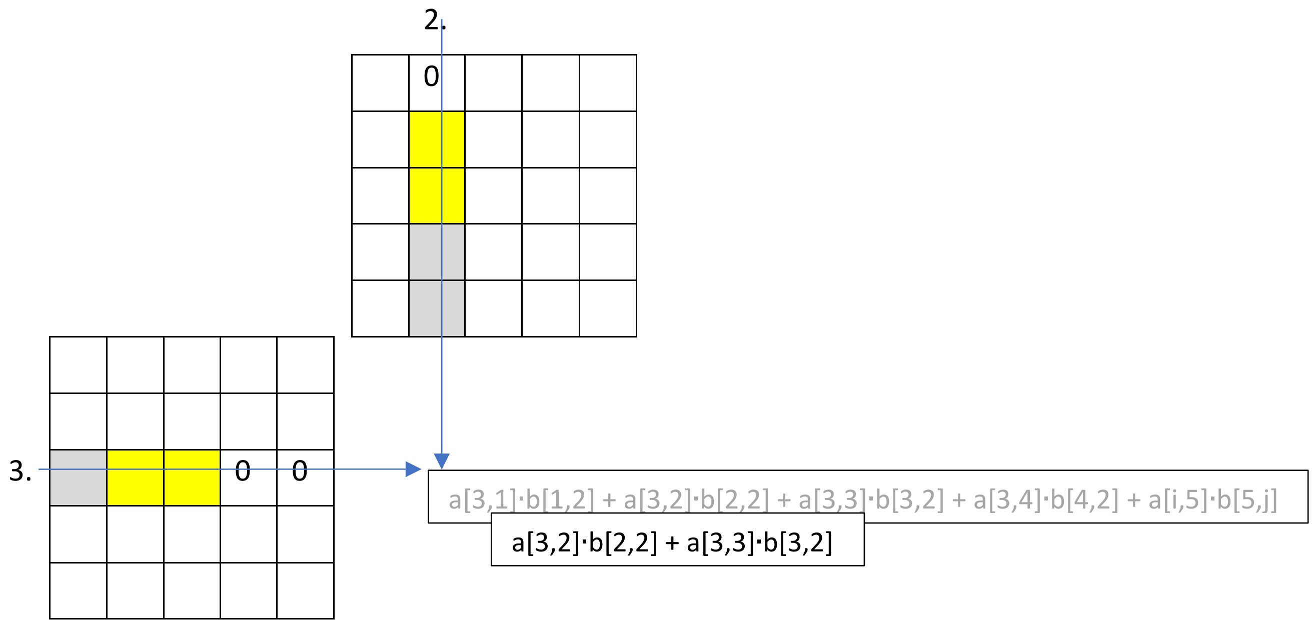
$ind(i,j) = j + \sum_{k=1}^{i-1} k = j + \frac{i \cdot (i-1)}{2}$, ha $1 \le j \le i \le n$ [2.kvíz]

Típusdefiníció:

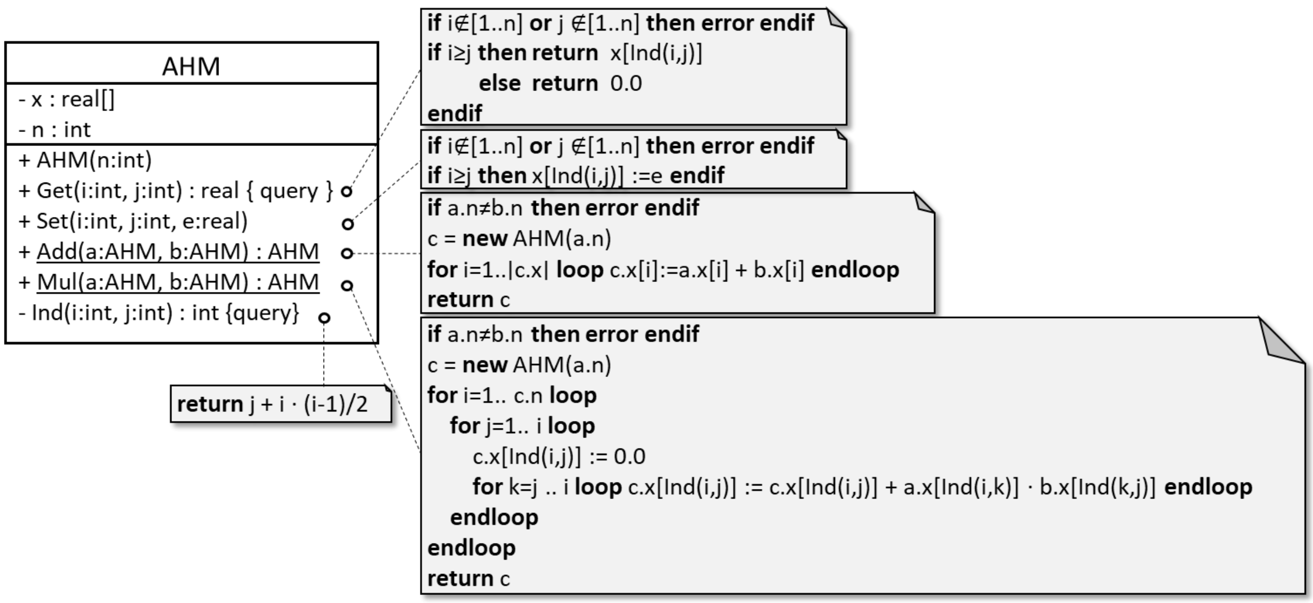
<table>
<tr><td rowspan="4">AHM(ℝ)

// ha a:AHM(ℝ) mátrix mérete: dim(a)×dim(a) akkor dim(a)≥1</td><td>e := a[i,j] (a : AHM(ℝ), i,j : [1..n], e : ℝ)</td></tr>
<tr><td>a[i,j] := e (a : AHM(ℝ), i,j : [1..n], e : ℝ) // i ≥ j</td></tr>
<tr><td>c := a + b (a, b, c : AHM(ℝ)) // dim(a)=dim(b)=dim(c)</td></tr>
<tr><td>c := a · b (a, b, c : AHM(ℝ)) // dim(a)=dim(b)=dim(c)</td></tr>
<tr><td rowspan="4">x:$\mathbb{R}^*$
n:ℕ

// |x|=n·(n+1)/2
// n≥1</td><td>if i≥j then e := a.x[ind(i,j)] else e := 0.0</td></tr>
<tr><td>if i≥j then a.x[ind(i,j)] := e else error</td></tr>
<tr><td>if a.n = b.n = c.n then
∀i∈[1..|c.x|]: c.x[i] := a.x[i]+b.x[i]</td></tr>
<tr><td>if a.n = b.n = c.n then
∀i,j∈[1..c.n]: [3.kvíz]
if i≥j then c.x[ind(i,j)]:= $\sum_{k=j}^{i}$ a.x[ind(i,k)] · b.x[ind(k,j)]</td></tr>
</table>



Osztály diagram:



3. 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.

Érdemes külön figyelmet fordítani arra, amikor az E elemei sorba rendezhetők (például E a természetes számok halmaza).

<table>
<tr>
<td>Bag

azon zsákok halmaza, amelyek elemei (E) rendezhetőek</td>
<td>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 := Insert(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</td>
</tr>
<tr>
<td>[4.kvíz]

seq : Pair*

maxind : $\mathbb{N}$

ahol

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

Invariáns:

- a seq-ben az elemeket tartalmuk (data) szerint rendezetten tároljuk

- a maxind a nem üres seq sorozat legnagyobb count értékű elemének indexe</td>
<td>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)
if l then 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 | |seq|>1 ∧ maxind≥ind | else
maxind := ind | – | maxind := 1 | ++maxind | –</td>
</tr>
</table>

Elem szerinti keresés [átkerülhet a következő gyakorlat elejére]

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

Ef = (seq=seq$_0$ $\wedge$ e=e$_0$ $\wedge$ $\forall$i$\in$[1 .. |seq|-1] : seq[i].data < 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)

l, ah, fh := hamis, 1, |seq| ah, fh : $\mathbb{N}$

$\neg$l $\wedge$ 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

$\neg$l

ind := ah	—

[5.kvíz]

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 := MAX$_{i=1..\|seq\|}$ (seq[i].count)	–

Osztály:

Bag
- seq : Element[] - maxind : int
+ SetEmpty() + Empty() : bool {query} + Multipl(e : E) : int {query} + Max() : E {query} + Insert(e: E) + Remove(e: E) - LogSearch(e:E) : (bool,int) {query}

SetEmpty():
```
seq := <>
```

Empty():
```
return |seq|=0
```

Multipl(e):
```
l, ind := LogSearch(e)
if l then return seq[ind].count
else return 0
endif
```

Max():
```
if |seq|>0 then return seq[maxind].data endif
```

Insert(e):
```
l, ind := LogSearch(e)
if l then
   ++seq[ind].count
   if seq[ind].count > seq[maxind].count then maxind := ind endif
else
   seq := seq[1..ind-1] ⊕ <(e,1)> ⊕ seq[ind+1..|seq|]
   if |seq|=1 then maxind := 1
   elsif maxind>ind then ++maxind
   elseif skip
   endif
endif
```

Remove(e):
```
l, ind := LogSearch(e)
if l then
   if seq[ind].count > 1 then
      - - seq[ind].count
   elsif seq[ind].count = 1 then
      seq := seq[1..ind-1] ⊕ seq[ind+1..|seq|]
   endif
   if |seq|>0 then max, maxind := MAX_{i=1..|seq|} (seq[i].count) endif
endif
```

LogSearch(e):
```
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)
```