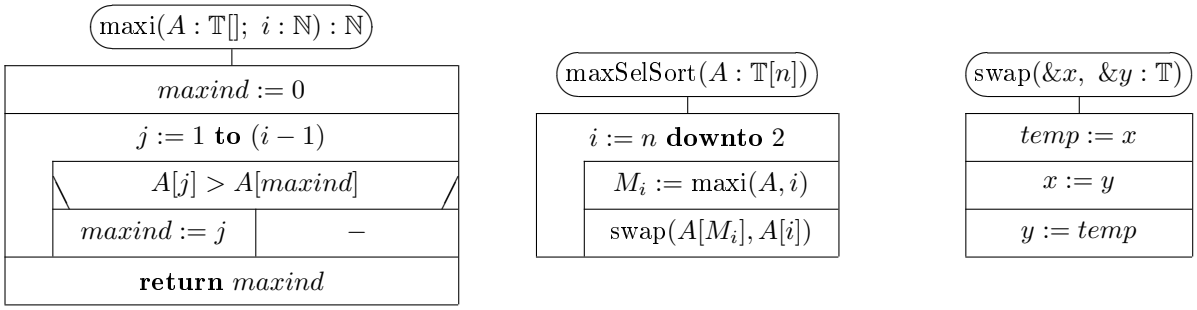
Algoritmusok és adatszerkezetek I.
Gyakorlatjegyzetek

Gyakorlatvezető: Dr. Ásványi Tibor
Csoport: 16 (2022/23/2. szemeszter)

2023. április 5.

1. gyakorlat

1.1. Maximumkiválasztásos rendzés



Példa a használatára: $M_i := maxi(B, 5)$

- a B valójában egy pointer
- felteszsük, hogy elérhetjük a B elemszámát (mintha amolyan objektum lenne): $*B = B[0];\ *(B+i) = B[i]$

Sorbarendezés illusztrációja

$< 3, 1, 2, 4, 2' >$ → monoton növekvő → legvégére

$< 3, 1, 2, 2'|4 >$ → rendezetlen | rendezett

$< 2', 1, 2|3, 4 >$

$< |1, 2', 2, 3, 4 > \rightarrow < 1, 2', 2, 3, 4 >$

Hatékonyság. Ilyenkor **felülről** ($MT(n)$) és **alulról** ($mT(n)$) kell becsülni a program *műveletidejét* / *hatékonyságát* / *futásidejét* ahol csupán a nagyságrendet kell figyelni: $c \cdot n,\ n!,\ n^2,\ \log n$, stb.

$$T(n) =? = \sum_{i=2}^{n}(i-1) = \sum_{k}^{n-1} k = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2} \tag{1}$$

$$T(n) = \frac{1}{2}n^2 - \frac{1}{2} \in \Theta(n^2) \tag{2}$$

ahol $\Theta(\square)$ olyan fgv.-ekből álló fgv.osztály, mely hasonló nagyságrendű függvényekből áll, mint az argumentuma (amit itt $\square$ jelöl)

Az n^2-es algoritmusok eléggé lassúak, több évtizedig is tarthat a futása elég nagy mennyiségű bemeneten. Ez **absztrakt idő**, nem fizikai idő – ingadozni fog 10%-ot egy határon belül.

Rendezés stabilitás. Ha két azonos kulcsú elem egy adott sorrendben van, mindig ugyanabban lesznek, akárhányszor futtajuk le a programot

Maximumkiválasztásos rendezés: **nem stabil**

Lehet javítai a maximukiválasztásos rendezésen

1.2. (Naív) Buborékrendezés

$< 3, 1, 2, 4, 2' >$

$< 1, 3, 2, 4, 2' >$

$< 1, 2, 3, 4, 2' >$

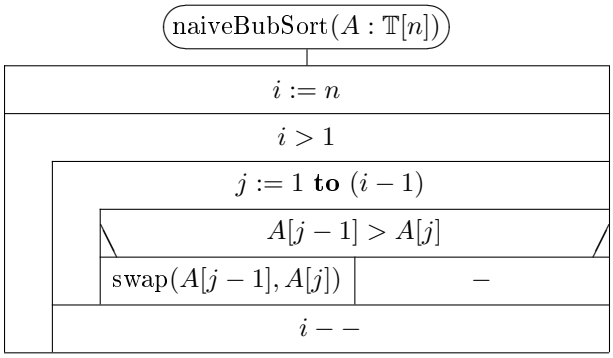
$< 1, 2, 3, 2'|4 >$

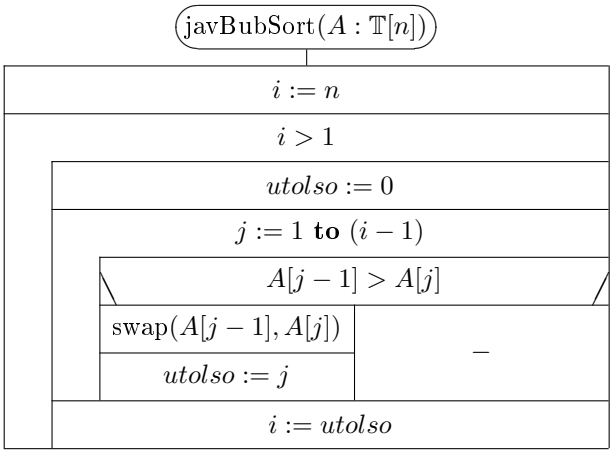
$< 1, 2|2', 3, 4 > \rightarrow < 1, 2, 2', 3, 4 >$

Stabilitás: stabil
Iterációk: pontosan ugyanannyit
Hatékonyság: elég rossz (még rosszabb is), mert picit fölöslegesen dolgozik ($\Theta(n^2)$)
Ha egy adott szakaszon / ponttól nem cserél, ott biztosan rendezett → megoldás: *jegyezzük meg, h hol cserélt utoljóra*

1.3. (Javított) Buborékrendezés

Megjegyzi, hol cserélt utoljára





min. műveletigény: $mT(n) = n - 1 \in \Theta(n)$

max. műveletigény: $MT(n) = \frac{1}{2}n^2 - \frac{1}{2} \in \Theta(n^2)$ → ha csökkenő sorrendben van az eredeti

2. gyakorlat

2.1. Beszúró rendezés

- a buborékoshoz képest javított
- ketté vágjuk
  - hagyományosan az *eleje* rendezett
  - nincs a kettő résztömb között kapcsolat
- stabil rendezés, helyben $\rightarrow \Theta(1)$ **munkatár**ral tudja rendezni
- ösztönösen használt, intuitíve jövő algoritmus

  $< 5, 3, 7, 2, 4, 3' >$

  $< 3, 5, 7 | 2, 4, 3' > \rightarrow$ beillesztjük a megfelelő helyre a rendezetlen első elemét

  $< 2, 3, 5, 7 | 4, 3' >$

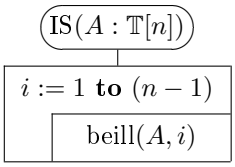
  $< 2, 3, 4, 5, 7 | 3' > (*)$

  $< 2, 3, 4, 5, \square, 7 >$ ahol $x := 3'$

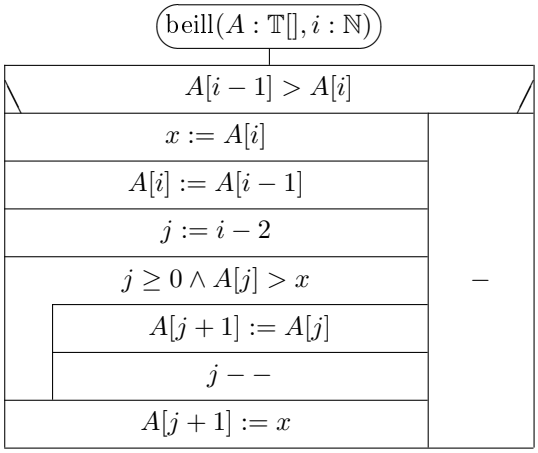
  $< 2, 3, 4, \square, 5, 7 >$

  $< 2, 3, \square, 4, 5, 7 > \leftarrow x$-et beillesztjük az üres helyre

  $< 2, 3, 3', 4, 5, 7 >$ beillesztés után



IS = insertion sort
Megjegyzés: itt a tömböt 0-tól indexeljük

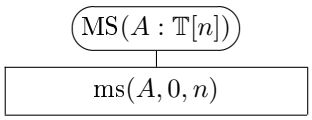


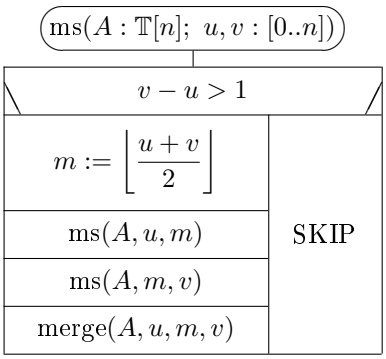
- műveletigény:
  - $mT(n) = n - 1 \in \Theta(n)$
  - $MT(n) = (n-1) + \sum_{i=1}^{n-1}(n-1) = (*) = \frac{1}{2}n^2 - \frac{1}{2} \in \Theta(n^2)$
  - átlagos műveletigény $\rightarrow$ ea. jegyzetben található
- mitől jobb a buborékrendezésnél?
  - csak akkor cserél, ha nincs a helyén
  - $(k+2)$ db a mozgatások száma, míg a bubisé $3k$ (?) (igazából gőzöm sincs)

2.2. Összefésülő rendezés (merge sort)

- „oszd meg és uralkodj”-elv, azaz:
  - szimmetrikusan elfelezzük → külön-külön sorbarendezzük a résztömböket
  - akkor is szétválasztja, ha sorba van rendezve
  - az egyelemű részsorozatok eleve rendezve vannak, így ezeket összefuttatja
  - rekurziós szintekkel feljebb megy és összefésüli a résztömböket
- rekurziós mélység: $\simeq \log_2 n$;
  - minden szinten szétvágást és összefésülést hajt végre
  - rekurziós memóriaigény / futásidő minden szintn $\Theta(n)$, kivéve a 0-1. szinteken
- rendezés: stabil
- hatékonyság: akár $mT(n) \in \Theta(\log n)$, akár $MT(n) \in \Theta(n \log n)$, nem lehet lineáris (de ettől függetlenül még jónak számít)

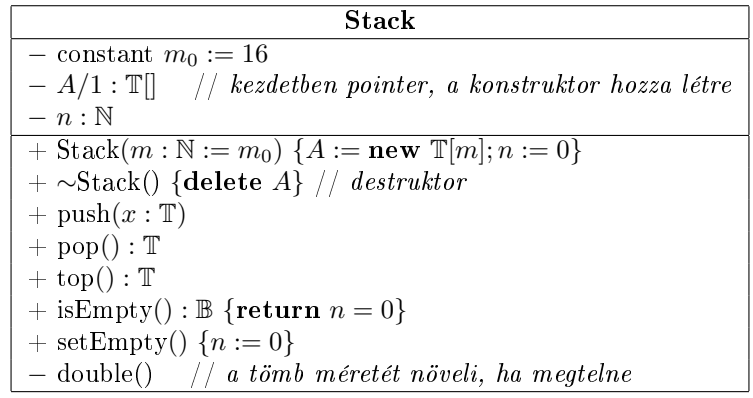
Az ábrát hadd ne gépeljem be, megtalálható amúgyis az ea. jegyzetben.

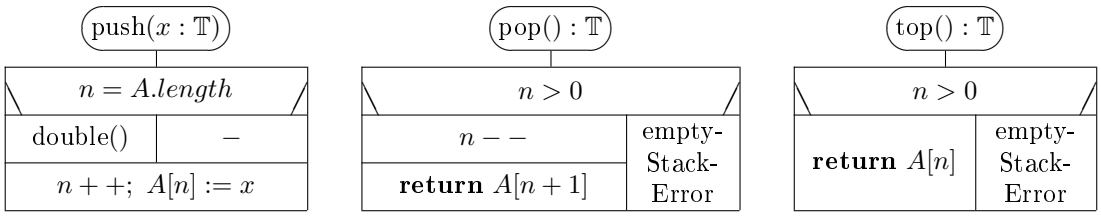


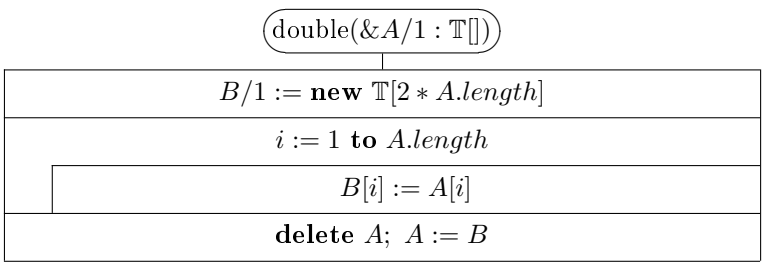


2.3. Verem (stack) adattípus

Megjegyzés: 1-től indexeljük az elemeket (ahol az $n := 0$ a pillanatnyi fizikai mérete a tömbnek)







3. gyakorlat

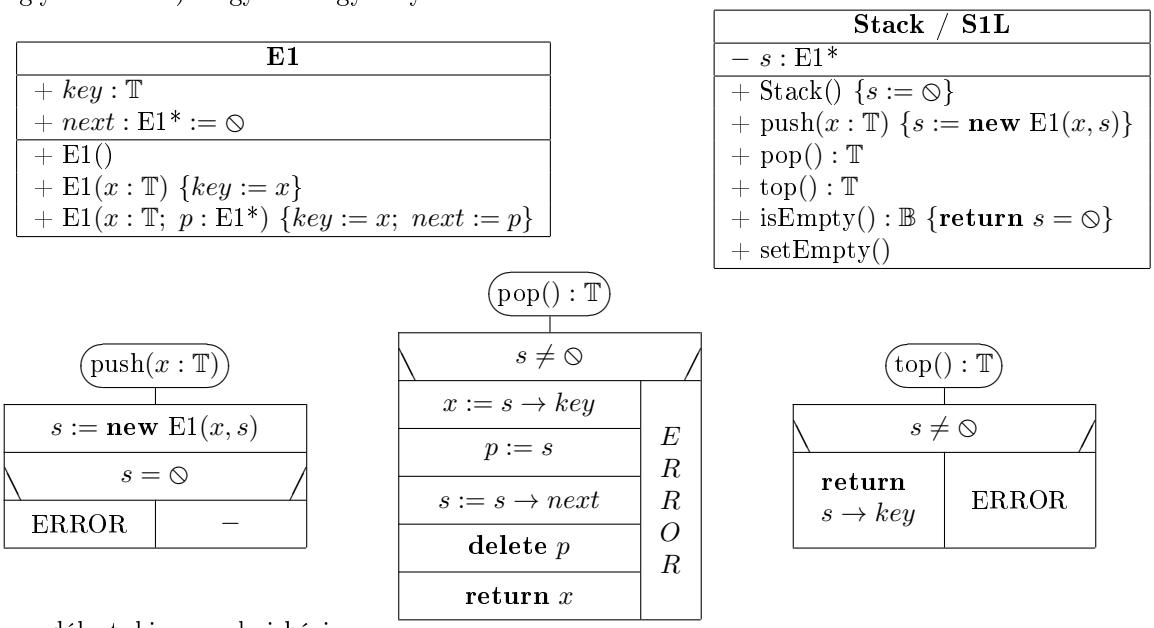
3.1. Verem felhasználása: lengyel jelölés kiértékelése

- matematikai kifejezések jelölése:
  - infix jelölés (hagyományos): $5 + 3 \cdot 4/(2 + 3 - 1)$
  - postfix jelölés (lengyel forma): $5\ 3\ 4 * 2\ 3 + 1 - / + \#$
    - $\#$: a string végét jelöli ($\sim$ C-ben `'\0'`)
    - btw, a Wikipédia szerint a lengyel jelölés valójában a *prefix* jelölést jelenti, míg a *postfix* jelölést *reverse Polish notation*nek nevezi ¯_(._.)_/¯
- miért jó? $\rightarrow$ egyes fordítóprogramok átalakítják lengyel formára, mert nagyon könnyű kiértékelni
- kiértékelésük veremmel:
  - *lengyel forma*: számok a verembe, az operátorok „kiveszik” (pop()-olják) a szükséges számú argumentumokat
  - *infix forma*: műveleti jelek a verembe ($+, -$, zárójelek, $\#$), a számokkal meg lesz valami – sajnos nem értem, hogy működik :')

Az ábrákat egyelőre nem rajzolnám fel. Gyakorlásként meg lehet írni a struktogramját a postfix-kiértékeléses és az infix-kiértékeléses algoritmusnak.

3.2. Verem adattípus másképp

S1L (singly linked list) – egyszerű egyirányú lista



A maradék stuki szorgalmi-házi.

4. gyakorlat

4.1. Sor / Lánc adatszerkezet

First In First Out (FIFO) adatszerkezet

Queue	
$-\ f, t : \mathbf{E1}^*$	
$-\ n : \mathbb{N} := 0$	
+ Queue() $\{f := t := \mathbf{new}$ E1$\}$	// *konstruktor*
+ add($x : \mathbb{T}$)	// *sor végére*
+ rem() : $\mathbb{T}$	// pop()*-nak felel meg*
+ first() : $\mathbb{T}$	// *ritkán használják*
+ length() : $\mathbb{T}$ $\{\mathbf{return}\ n\}$	// isEmpty() : $\mathbb{B}$ helyett
+ setEmpty()	
+ $\sim$Queue() {setEmpty(); **delete** f}	// *destruktor*

Ha képes az $[i]$-edik elemét lekérdezni, akkor már nem queue-ról / sorról van szó.
3 klasszikus megoldása létezik (ábrákat pls hadd ne):

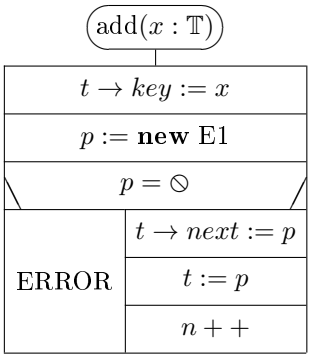
A) f közvetlenül az első elemre, t közvetlenül az utolsóra mutat

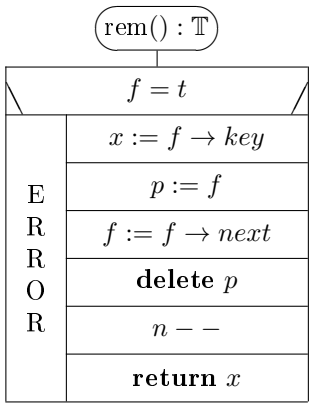
B) f közvetlenül az első elemre, t közvetlenül egy önálló végelemre mutat

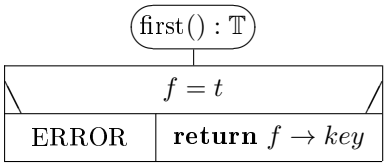
- *végelem*: inicializálatlan kulcsú, $\oslash$-ra mutat a *next* adattagja
- amikor az add(x) metódusát meghívjuk, inicializálja eme végelemet (beállítja x-re a kulcsot) és generál a végére egy új végelemet

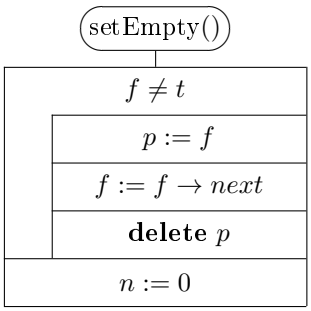
C) *ciklikus*: a végelem rámutat a fejelemre (azaz az első elemre)

Itt a B) modell szerint írjuk meg. A megjegyzések az osztálydiagramban ezt a modellt szemléltetik.
Szorg-hf: megírni a maradék stratégia egyikével.









Alternatív megoldás: $\mathbb{B}$ típussal tér vissza, nem használ kivételt, így FALSE-szal tér vissza, ha nem sikerül hozzáadni. Ellentétben könnyebben elrejtődnek a hibák.

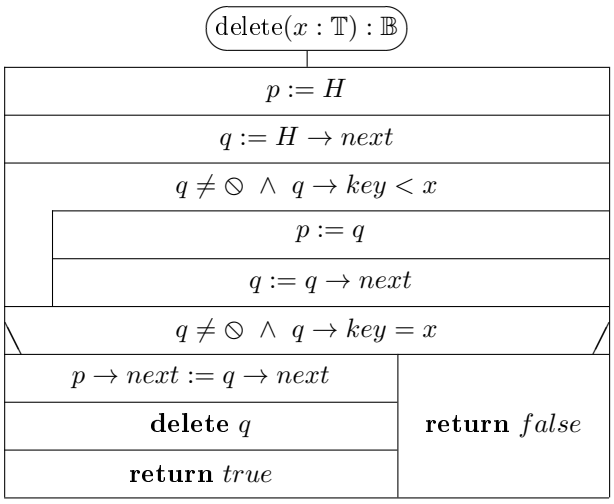
A p segédváltozóval eltároljuk az f memóriacímét a memóriaszivárgás elkerülése végett.

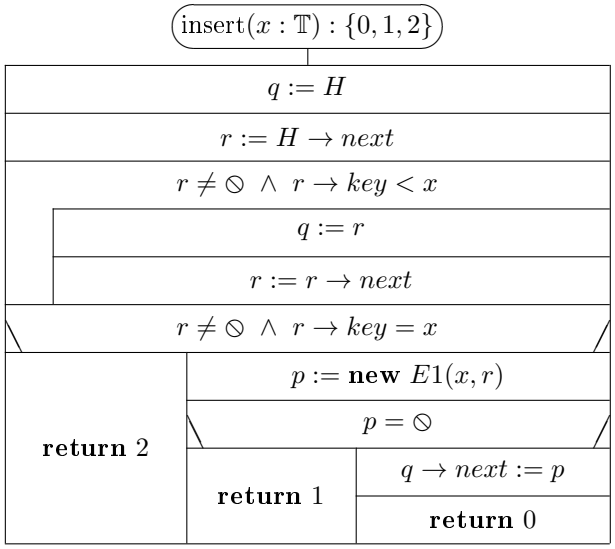
4.2. Halmaz adatszerkezet fejelemes listával (H1L)

- Növekvő sorrendben tárolja az elemeket (szig. mon. növ. ↑).
- Minden elem csak egyszer szerepelhet.
- A halmaz fejelemes egyirányú listát alkalmaz (H1L).

SET
$-\ H : \mathbf{E1}^*$
$+$ SET() $\{H := \mathbf{new}\ E1()\}$ $+$ insert$(x : \mathbb{T}) : \{0,1,2\}$ $+$ delete$(x : \mathbb{T}) : \mathbb{B}$ $+$ search$(x : \mathbb{T}) : \mathbb{B}$ $+$ setEmpty() $+$ isEmpty$() : \mathbb{B}$ $\{\mathbf{return}\ H \to next = \oslash\}$ $+ \sim$SET() $\{setEmpty();\ \mathbf{delete}\ H\}$

Az insert()-hez némi magyarázat: 0 - siker; 1 - memória túlcsordulás; 2 - már benne van.
A search() TRUE-val tér vissza, ha megtalálja a szóban forgó elemet.





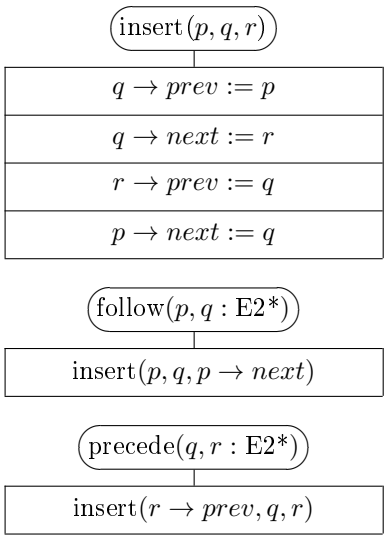
Érdemes megfigyelni a ciklusfeltételekben az állítások sorrendjét: $q \neq \oslash \ \wedge\ q \to key < x$. Ha a $q \neq \oslash$ FALSE-szal tér vissza, a hátralévő részét ki sem értékeli → hangyányit hatákonyabb, mint a fordított sorrendben.
A maradék metódus szorg-hf.-nek lett feladva

5. gyakorlat

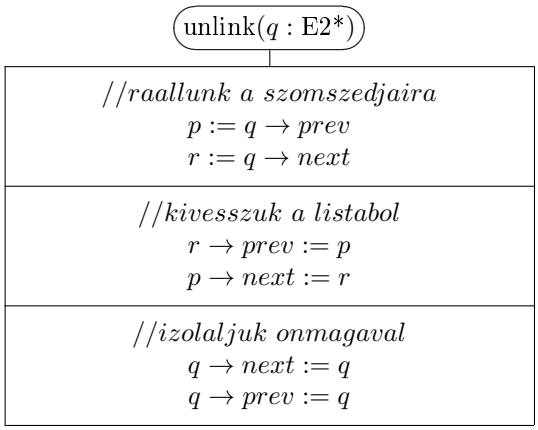
5.1. Ciklikus kétirányú lista (C2L)

E2
$+\ key : \mathbb{T}$ $+\ next, prev$: E2* := **this** // *önmagára inicializálja*
$-$ insert(p, q, r) $+$ follow$(p, q :$ E2*$)$ $+$ precede$(q, r :$ E2*$)$ $+$ unlink$(q :$ E2*$)$

- fejelemes (alapértelmezetten ilyen): H (key adattagja inicializálatlan)
- végén lehetne NULL is, de praktikusabb, ha összekötjük → hence: ciklikus
- rövidítése: C2L
- alapvetően 2 művelet (3, ha szigorúan vesszük; 4, ha annál is szigorúbban)
  1. Beszúrás két elem közé: insert(p, q, r) (segédfüggvény, jellemzően privát)
  2. Beszúrás elé / mögé: follow$(p, q :$ E2*$)$, precede$(q, r :$ E2*$)$
  3. Elem kifűzése: unlink$(q :$ E2*$)$

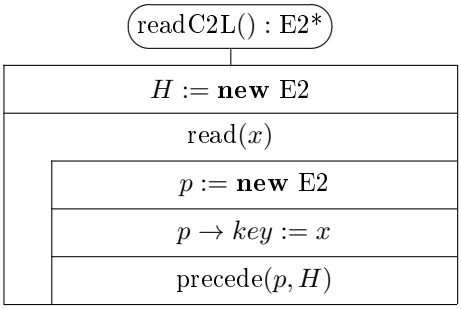


Fontos: nem törli, csak függetleníti a láncból



Miért jó, hogy ciklikus?

- bárhova be lehet szúrni
- legvégére hogyan a leghatékonyabb?
  1. follow($H \to prev, q$)
  2. precede(q, H)
- ekvivalensek, de a második elhanyagolhatóan hatékonyabb



További lehetséges műveletek: readC2L() : E2*, slice() : E2* (utóbbi egy egész listaszakaszt hoz létre)

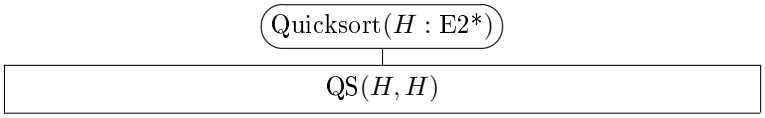
Ennek fényében nézzük meg a gyorsrendezést!

5.2. Quicksort

- működése:
  - kiválasztunk egy tengelyt (pivotot) (az eredetiben véletlenszerűen, itt az első eleme a listának)
  - átrendezzük az elemeit úgy, h a tengely elé kerüljenek a kisebbek, mögé a nagyobbak
  - felbontjuk részlistákra (partícionáljuk) és ugyanígy ismételjük az eljárást, amíg 2-eleműeket nem kapunk
  - legvégül összefésüljük
- *stabil* rendezés (C2L-re, a tömbrendező változatanem az)
- rendelkezik fejelemmel (H)
- kvízkérdésekből szerzett állítások
  - „oszd meg és uralkodj” elvű algoritmus
  - a particionálás egy-elemű résztömbre nem hajtódik végre
  - a tengely kiválasztása és a részekre bontás mindig lineáris időben befejeződik
  - a quicksort particionáló eljárása közben a tengellyel egyenlők a tengely elé és mögé is kerülhetnek

A kapcsolódó ábrák szintúgy a Teams-es órai jegyzetben találhatók.

Jelöljék a köv. változók az alábbi értékeket: $p := H \to next$, $r := H \to prev$



Itt p és r közötti részét rendezzük a listának. Ilyenkor nem kell rendezni: *egyelemű*, *üreslista*.