

## 4. Gyakorlat (2019.03.05.)

**Téma:** láncolt ábrázolás, egyszerű lista, fejelemes lista. Keresés, beszúrás, törlés rendezett egyszerű listával, beszúrás fejelemes lista esetén, egyszerű lista megfordítása, egyszerű lista szétfűzése, prímszita listával.

### Láncolt listák

Egy vagy két irányúak lehetnek.

Összehasonlítás a tömbbel:

- Előny: a rendezett beszúrás / törlés nem igényel elemmozgatást. Persze a beszúrás / törlés helyének megtalálása rendezett esetben  $O(n)$ .
- Hátrány: nem indexelhető konstans műveletigénnyel, csak  $O(n)$ -nel!

### Egyirányú lista

Listaelem típusa (jegyzetből):

| E1                                                                                       |
|------------------------------------------------------------------------------------------|
| <i>+key</i> : $\mathcal{T}$<br>... // satellite data may come here<br><i>+next</i> : E1* |
| <i>+E1()</i> { <i>next</i> = $\emptyset$ }                                               |

Bejáráshoz pointereket használunk: p, q: E1\*

Elem adattagjainak elérése:  $p \rightarrow \text{key}$ ,  $p \rightarrow \text{next}$  (helyes még a  $*(p).\text{key}$ ,  $*(p).\text{next}$ )

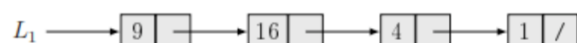
Ha új listaelemet szeretnénk létrehozni:  $p = \text{new E1}$ ,

feleslegessé vált listaelem felszabadítása: `delete p` (utána már nem hivatkozhatunk rá)

### Egyirányú listák fajtái (jegyzetből)

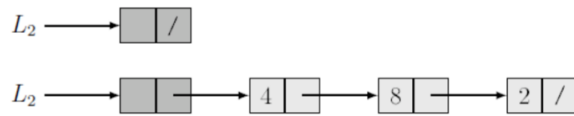
1. **Egyszerű egyirányú láncolt lista (S1L):** a legegyszerűbb forma, az első elemre egy pointer mutat. Ha még nincs eleme a listának, ez a pointer  $0(\text{Null}, \text{Nil})$  értékű.

$L_1 == \emptyset$



2. **Fejelemes egyirányú láncolt lista (H1L):** gyakori trükk, hogy egy valódi adatot nem tároló elemet helyezünk el a lista elejére. Célja: a lista elején (vagy az üres listával) végzett műveletek megkönnyítése, mert így a lista

elejére mutató pointerünk soha nem 0 értékű, továbbá az első elem előtt is van egy listaelem



## Feladatok

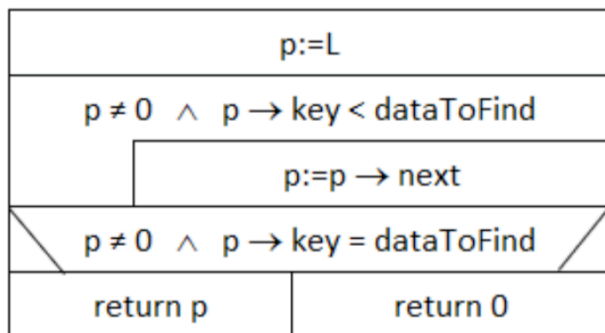
S1L kezelésének bemutatása: egy halmazt ábrázolunk egyszerű listával (egy adott kulcs csak egyszer fordulhat elő). Típus műveletek:

- egy elem benne van-e a halmazban: **keresés** kulcs alapján,
- egy elem betevése a halmazba (ha még nincs benne): **beszúrás**,
- egy elem kivétele a halmazból (ha benne van): **törlés**.

Legyen a listánk növekedően rendezett. Készítsük el a keresés, beszúrás és törlés algoritmusát:

### Keresés:

**findInS1L(L:E1\*, dataToFind :T) : E1\***



Sikertelen keresés eredménye: Null pointert ad vissza, így nem kell a logikai változó ami lineáris keresés tételénél szerepelt.

### Beszúrás:

A kereséshez hasonló ciklussal indul, meg kell keresni a kulcs helyét a rendezett sorozatban. A műveletekhez világos, hogy két elem címe kell: az az elem, ami elé fogunk befűzni (az első olyan, melynek a kulcsa nagyobb a beszúrandó elem kulcsánál), valamint az előtte lévő elem címe. Ehhez két bejáró pointert használjunk. Egyirányú listák esetén, ha a lista felépítését megváltoztatja az algoritmus, mindig két bejáró pointert használunk.

**insertIntoS1L(&L: E1\*, dataToInsert: T)**

|                                |                         |               |
|--------------------------------|-------------------------|---------------|
| pe:=0; p:=L                    |                         |               |
| p ≠ 0 ∧ p → key < dataToInsert |                         |               |
| pe:=p                          |                         |               |
| p:=p → next                    |                         |               |
| p ≠ 0 ∧ p → key = dataToInsert |                         |               |
| skip                           | q := new E1             |               |
|                                | q → key := dataToInsert |               |
|                                | q → next:= p            |               |
|                                | pe = 0                  |               |
|                                | L:=q                    | pe → next:= q |

A két bejáró pointer: pe, és p.

pe pointert 0-ról indítsuk, ez jelzi majd, hogy nincs még „előző” elem!

Ha már van ilyen kulcsú elemünk, nem történik semmi.

A beszúrásnál csak az a lépés kerül elágazásba, amikor beszúrt elem előtti elemnek a next pointerét módosítjuk, ugyanis, ha nincs előző elem, akkor L módosul!

Mivel a műveletnél L pointer módosulhat, így fontos, hogy az cím szerint átvett paraméter legyen!

## Törlés:

**deleteFromS1L(&L:E1\*, dataToDelete: T)**

|                                |                      |      |
|--------------------------------|----------------------|------|
| pe:=0; p:=L                    |                      |      |
| p ≠ 0 ∧ p → key < dataToDelete |                      |      |
| pe:=p                          |                      |      |
| p:=p → next                    |                      |      |
| p ≠ 0 ∧ p → key = dataToDelete |                      |      |
| pe = 0                         |                      |      |
| L := p → next                  | pe → next:= p → next | skip |
| delete p                       |                      |      |

Hasonlóan a beszúráshoz, itt is pe és p a két bejáró pointer.

Ha az adott kulcsú elem nem található meg, akkor nem történik semmi.

Itt is fontos, hogy L cím szerinti paraméter.

Hívjuk fel a figyelmet, miért fontos a delete művelet!

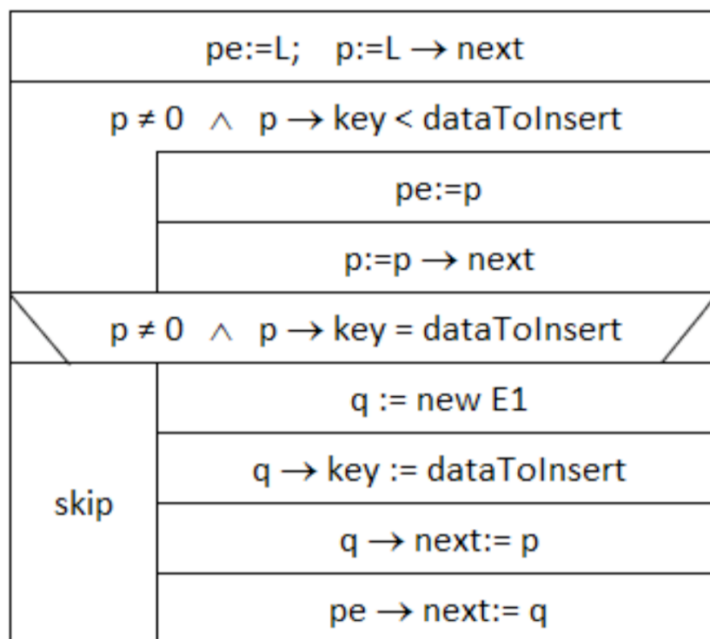
Nézzük meg fejelemes listákra is (H1L) a műveleteket!

**Keresés:** csak az indulás különbözik, a fejelemben nincs kulcs, így  $p := L \rightarrow \text{next}$  az induló lépés.

**Beszúrás:** indulásnál  $pe$  a fejelemre mutathat, hiszen az első elem előtt van mindig a fejelem,  $p$  pedig hasonlóan az előbbi algoritmushoz a lista első elemére mutat. Mivel  $pe$  nem lehet Null, nincs szükség az elágazásra a befűzésnél.

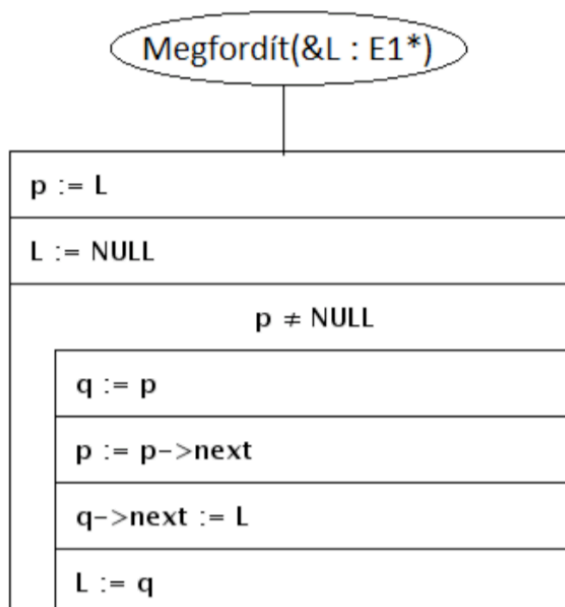
Fejelemes lista esetén a fejelem címét megadó paraméter nem cím szerint adódik át, hiszen a fejelem nem változtatja meg a címét, így az arra mutató pointer biztosan nem fog változni!

**insertIntoH1L(L: E1\*, dataToInsert: T)**



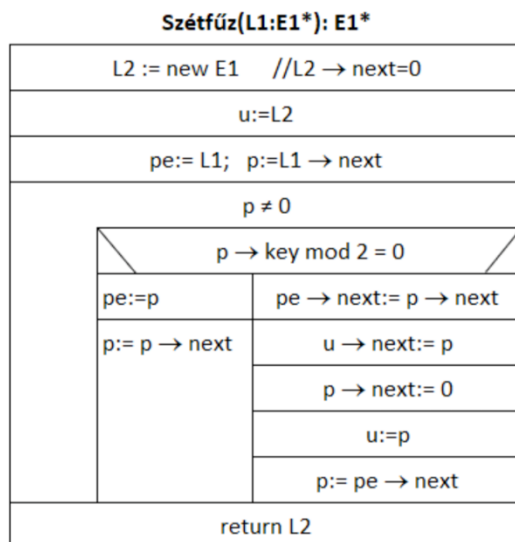
**Törlés:** indulás hasonlóan, itt sem kell elágazás a kifűzésnél, mert *pe* nem lehet Null értékű.

**Egyelemű lista megfordítása:** mintha egy verembe raknánk be a lista elemeit, az eredmény listának mindig az elejére fűzzük be az eredeti lista aktuális elemét.



## H1L szétfűzése két listába

Adott egy H1L lista, egész számokat tartalmaz, a fejelemre L1 mutat. Fűzzük ketté az elemeket: L1-ben maradjanak a páros elemek, egy új L2 H1L listába fűzzük át a páratlan elemeket. Az eredeti sorrendet tartsuk meg, azaz átfűzésnél mindig a lista végére kell majd fűzni. L1 listát egyszer lehet bejárni, L2-be a befűzés konstans műveletigényű legyen!

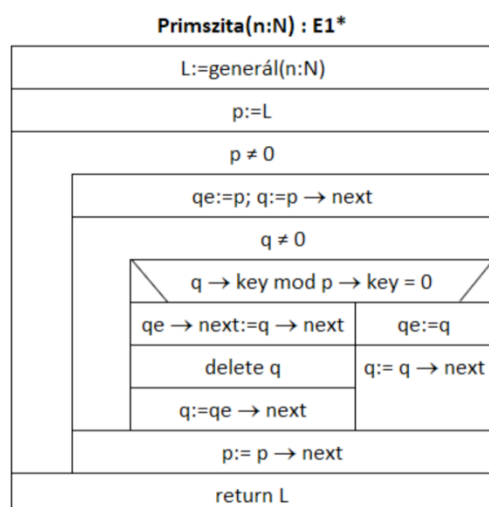


Első lépésként egy új H1L létrehozását mutatjuk be. Kihaszználjuk, hogy E1 konstruktora az elem next pointerét 0-ra állítja! u mindig L2 utolsó elemére fog mutatni, ez kezdetben a fejelem. pointererek L1 bejárásához

Ha páratlan kulcsú elemmel találkozunk, a lépések:  
 p kifűzése  
 p befűzése L2 végére  
 mivel p az L2 utolsó eleme, 0-ra állítjuk a next pointerét  
 módosítjuk u-t  
 p-vel L1 következő elemére állunk

## Prímszita egyszerű listán

Készítsünk egy listát, mely 2-től  $n$ -ig tartalmazza a prímekeket, az Erasztóthenészi szita algoritmus ötletét felhasználva. Töltsünk fel egy egyszerű listát  $2..n$  természetes számokkal. Az első elem prím, azokat, melyek oszthatók ezzel az elemmel, töröljük a listából. A következő megmaradt szám megint prím. Többszöröseit ismét töröljük. A végén a prímekek maradnak a listánkban.



Generálunk egy egyszerű listát  $2..n$  természetes számokból.  
 p mindig a következő, még a szitában lévő prímszámmra mutat (elsőként a 2-re). Amíg ez nem 0 ...  
 q-val a p utáni elemeket fogjuk bejárni, és kifűzzük p → key többszöröseit  
 Ha p→key osztója q→key-nek, a q című elemet töröljük a láncból. Ehhez szükségünk van q elem előzőjére, ez lesz a qe pointerben.  
 ha p című elem többszöröseit töröltük, a következő elem megint prím lesz.

| generál(n:N) : E1* |                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| L := 0             | Egyszerű lista feltöltése 2..n természetes számokkal.<br>Trükk: a ciklust csökkenőleg futtatjuk, így mindig a lista elejére kell befűzni. |
| i = n downto 2     |                                                                                                                                           |
| p := new E1        |                                                                                                                                           |
| p → key := i       |                                                                                                                                           |
| p → next := L      |                                                                                                                                           |
| L := p             |                                                                                                                                           |
| return L           |                                                                                                                                           |

### Házi feladatok:

1. Egy rendezetlen egyirányú fejelemes listában (H1L) keressük meg a 2. legnagyobb elemet, egyszeri bejárással. Feltehető, hogy a listánknak legalább két eleme van (a fejelemen kívül).
2. Rendezzünk egy H1L listát maximum kiválasztó rendezéssel. Ügyeljünk, hogy a megoldó algoritmus üres listára is működjön.
  - a. Ötlet: megkeressük és kifűzzük a lista legnagyobb elemét. A kifűzött elemekből egy egyszerű listát kezdünk építeni úgy, hogy mindig a lista elejére fűzzük be a rendezendő listából kifűzött elemet, nevezzük az így kapott listát segéd listának. Ez a segéd lista nyilván valóan növekvően rendezett lesz. Amikor a rendezni kívánt listának már csak egy eleme maradt, hozzáláncoljuk a segéd listánkat az utolsónak megmaradt elemhez, és készen vagyunk.