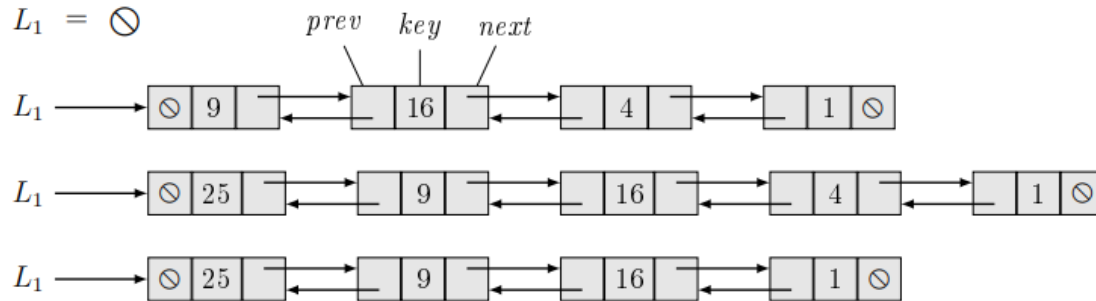


## Egyszerű kétirányú lista (S2L)

Jegyzet példája:



5. ábra. Az  $L_1$  mutató egyszerű kétirányú listákat (S2L) azonosít egy képzetbeli program futásának különböző szakaszaiban. Az első sorban a listát üresre inicializáló értékadás látható.

Hasonlóan az S1L listához a lista elejének, belsejének és végének kezelése különbözik, például befűzés esetén:

- Egy belső elemnek bal és jobb szomszédja is van, a befűzés négy pointert érint.
- Az első elemnek nincs bal szomszédja, az utolsónak nincs jobb szomszédja.
- Üres listába történő befűzés is külön eset: a befűzött elemnek sem bal, sem jobb szomszédja sincs.

Ezt a lista típust majd a láncolt hasító tábláknál használjuk.

## Ciklikus kétirányú listák (C2L)

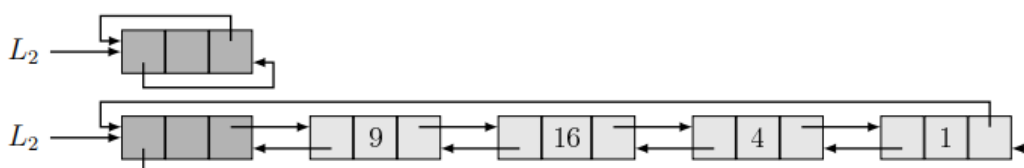
Lehet fejelemes, vagy fejelem nélküli. Legegyszerűbbek a műveletek a fejelemes C2L -ek esetén, így az alábbi feladatokban ezekkel fogunk foglalkozni.

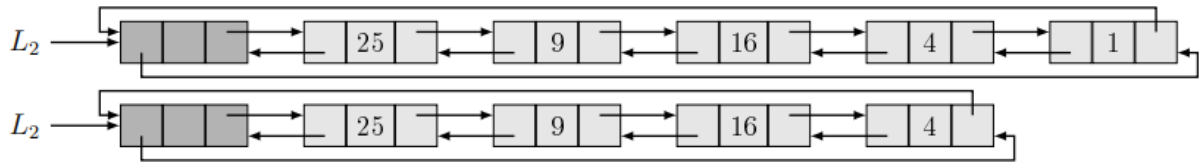
### Elemtípus, alapl műveletek

E2	
$+prev, next : E2^*$	// refer to the previous and next neighbour or be <b>this</b>
$+key : \mathcal{T}$	
$+ E2() \{ prev := next := \mathbf{this} \}$	

Minden adattagja publikus, az üres konstruktor a pointereket „önmagára” állítja!

C2L példák a jegyzetből:





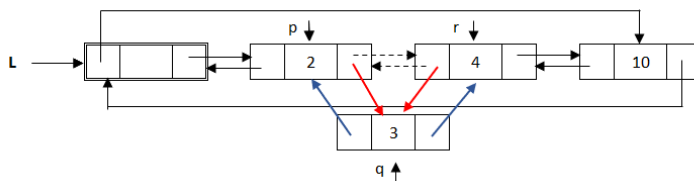
6. ábra. Az  $L_2$  mutató fejelemes kétirányú ciklikus listákat (C2L) azonosít egy képzeletbeli program futásának különböző szakaszaiban. Az első sorban az  $L_2$  lista üres állapotában látható.

Mivel a listaelem konstruktora a pointereket úgy állítja be, hogy azok magára az elemre mutassanak, ezt kihasználva egy új fejelemes C2L lista fejelemének létrehozása:  $L := \text{new } E2$  utasítással történhet!

## Alap műveletek

### Beszúrások:

#### $\text{precede}(q, r: E2^*)$



$\text{precede}(q, r : E2^*)$

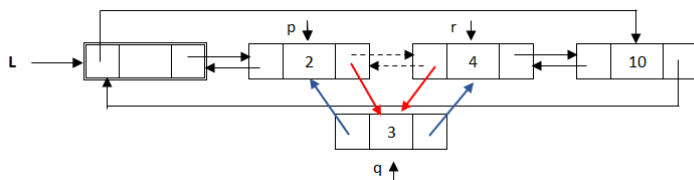
//  $(*q)$  will precede  $(*r)$

$p := r \rightarrow \text{prev}$

$q \rightarrow \text{prev} := p ; q \rightarrow \text{next} := r$

$p \rightarrow \text{next} := r \rightarrow \text{prev} := q$

#### $\text{follow}(p, q: E2^*)$



$\text{follow}(p, q : E2^*)$

//  $(*q)$  will follow  $(*p)$

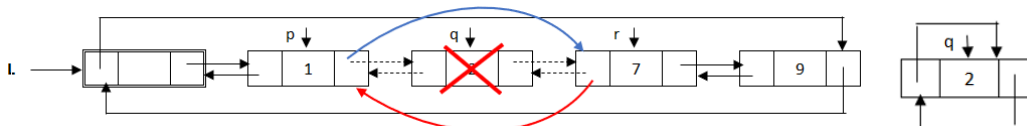
$r := p \rightarrow \text{next}$

$q \rightarrow \text{prev} := p ; q \rightarrow \text{next} := r$

$p \rightarrow \text{next} := r \rightarrow \text{prev} := q$

### Kifűzés:

#### $\text{unlink}(q: E2^*)$



$\text{unlink}(q : E2^*)$

// remove  $(*q)$

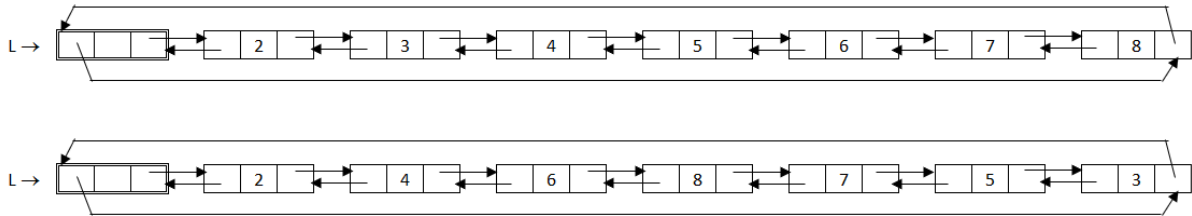
$p := q \rightarrow \text{prev} ; r := q \rightarrow \text{next}$

$p \rightarrow \text{next} := r ; r \rightarrow \text{prev} := p$

$q \rightarrow \text{prev} := q \rightarrow \text{next} := q$

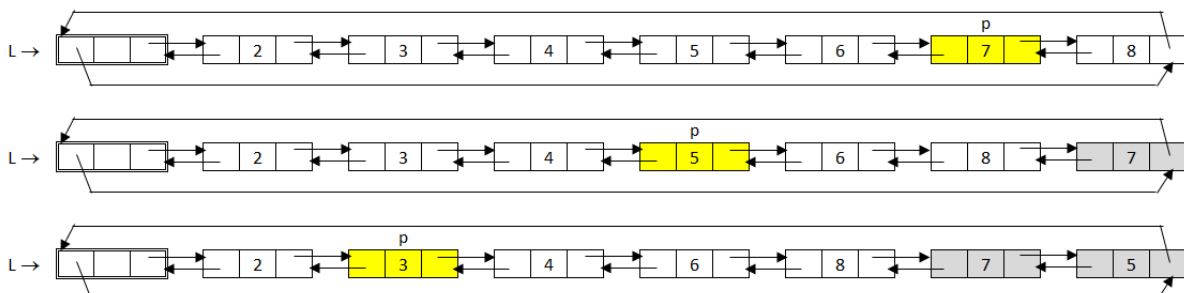
## Gyakorló feladat 1

- Adott egy természetes számokat tartalmazó C2L lista. A lista szigorúan monoton növekvően rendezett.
- A lista egyszeri bejárásával rendezzük át az elemeit úgy, hogy a lista elején legyenek a páros számok növekvően, a végén pedig a páratlanok csökkenően.

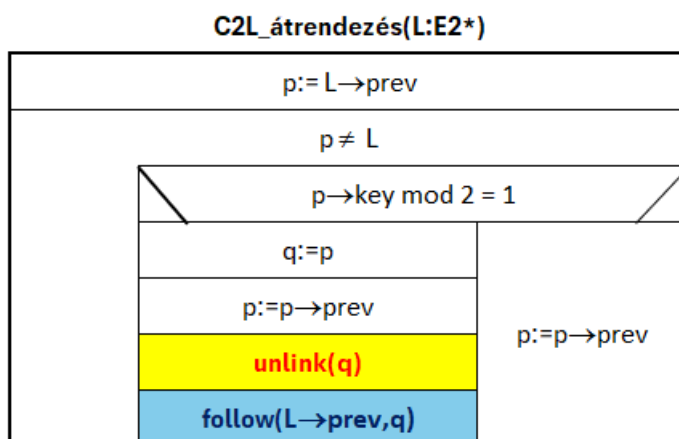


## Megoldási terv:

- Fordított irányban járjuk be a listát. A bejáró pointer  $p$  lesz.
- Ha a  $p$  elem kulcsa páratlan, az elemet kifűzzük, és átláncoljuk a lista végére (a fejelem elé).
- A kiláncoláshoz egy  $q$  segéd pointert fogunk használni,  $p$ -vel pedig a kiláncolás előtt tovább lépünk.



## Megoldás:

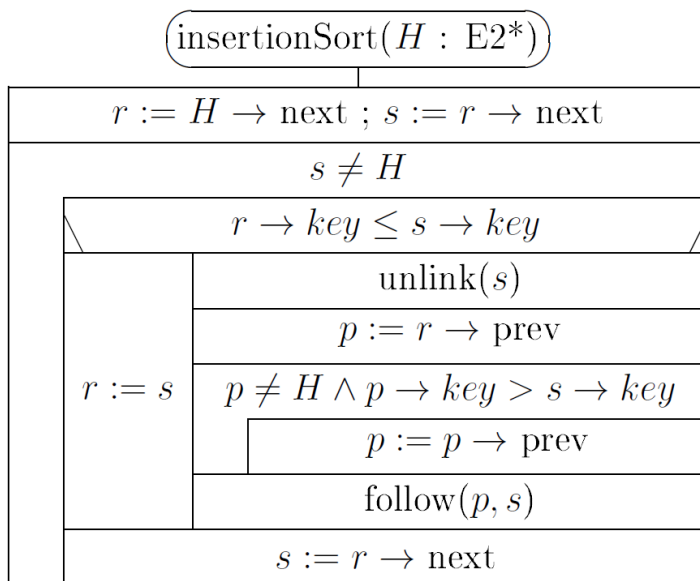


További kérdések:

- unlink és follow nélkül hogyan lehetne megoldani?
- A „follow” helyett hogyan lehetne a „precede”-t használni? precede(q,L)

## Beszűrő rendezés:

Jegyzetben megtalálható a beszűrő rendezés C2L-re (itt is visszafelé halad, a beszúrandó elem helyének megkeresése):



## Rendezett listák összefésülése

Igen gyakoriak azok a feladatok, melyek két rendezett sorozat összefésülésén alapulnak (például egy raktárban tárolt készlet napra-készítése a beszállítás / kiszállítás alapján). Ilyenkor a rendezettség nagyban növeli az algoritmus hatékonyságát. Ha a sorozatok listába vannak fűzve, akkor a pointerok állításával gyorsan és hatékonyan elvégezhető a két lista összefésülése.

Az összefésülési feladatok általában fejelemez listákkal kapcsolatosak, a lista típusa H1L vagy C2L, de például az előadáson láthattuk a merge sort listás változatát, mely egy S1L listát rendez összefésüléssel. (7.1.6. Az összefésülő rendezés S1L-ekre)

## Gyakorló feladat 3

L1 és L2 egy-egy szigorúan monoton növekvően rendezett C2L lista fejelemére mutat. Mivel a kulcsok egyediek, a listát halmaznak tekinthetjük. Állítsuk elő L1-ben a két halmaz unióját, úgy, hogy a szükséges elemeket L2-ből átfűzzük L1-be, a többit felszabadítsuk. Így az L2 lista az algoritmus végére kiürül.

### Megoldási terv:

Egy-egy bejáró pointerrel lépegetünk a listákon. p halad az L1 listában, q az L2 listában.

A pointerek által kijelölt elemek kulcsát hasonlítjuk össze, három eset lehetséges:

- (1)  $p \rightarrow \text{key} < q \rightarrow \text{key}$
- (2)  $p \rightarrow \text{key} = q \rightarrow \text{key}$
- (3)  $p \rightarrow \text{key} > q \rightarrow \text{key}$

A feldolgozó ciklust leállítjuk, ha valamelyik listán körbe érünk, ezért a ciklus feltétele, hogy  $p \neq L1 \wedge q \neq L2$

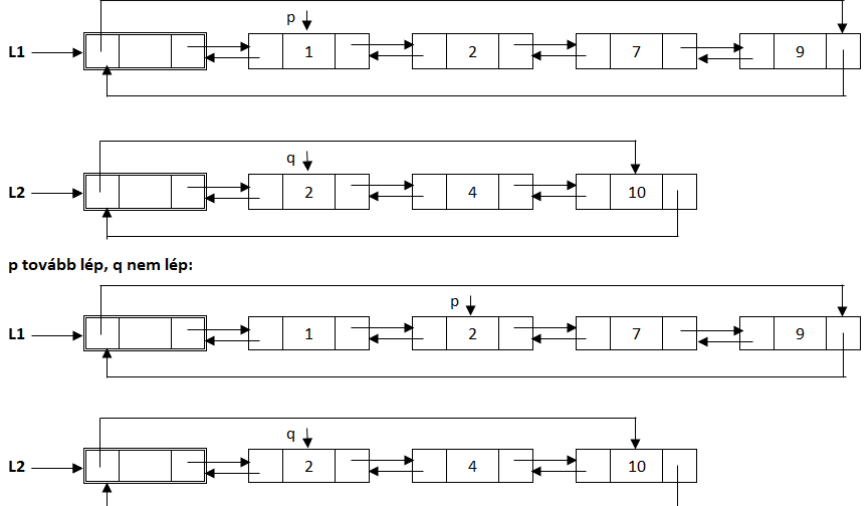


Ha valamelyik listán körbe érünk:

- Ha  $p=L1$  akkor L1, ha  $q=L2$ , akkor L2 listán körbe értünk. (Mivel egyenlő kulcsok esetén mindkét bejáróval lépünk, az is előfordulhat, hogy egyszerre érünk a listák végére.)
- Ha L2 listán értünk körbe, akkor kész vagyunk, ha viszont L2 listában még vannak elemek, azokat be kell fűzni L1 végére. Ezt megtehetjük egyesével, de hatékonyabb, ha a maradék lánc-részt egyszerre, konstans lépésben fűzzük át.

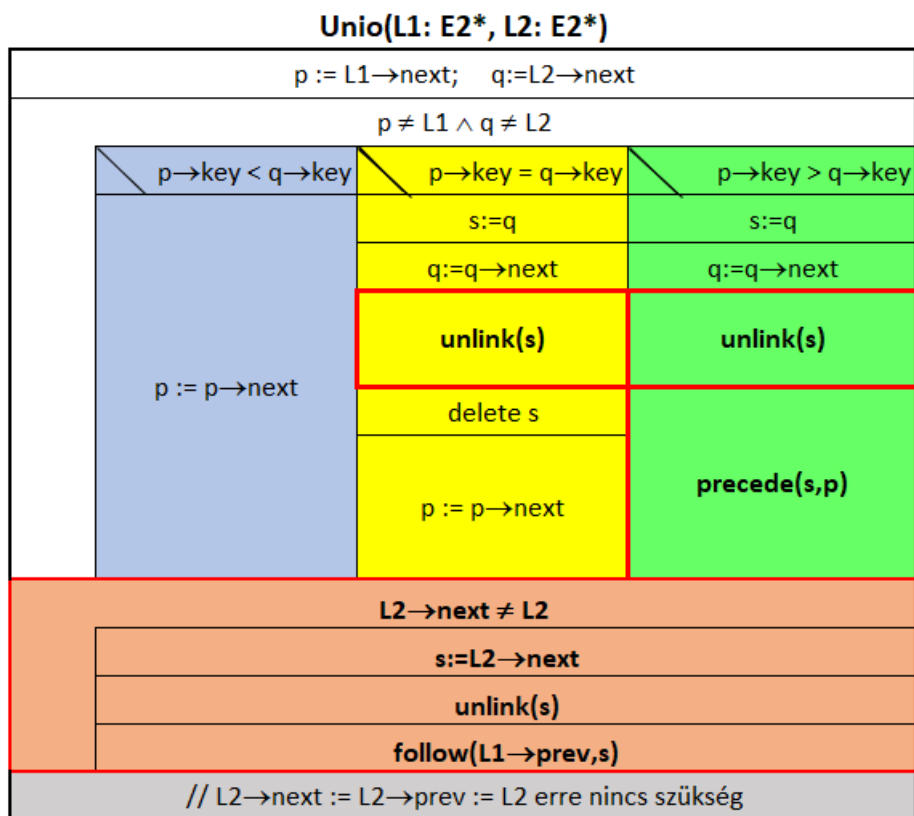
Megjegyzés:

Észrevehető, hogy valójában q bejáró pointerre nem is lenne szükség, mert az algoritmus során, a ciklus indulásakor  $q=L2 \rightarrow \text{next}$  mindig teljesül. De a rövidebb írásmód, és érthetőbb megfogalmazás miatt q-t használunk L2 aktuális elemének címezéséhez.

Az esetek szemléltetése:

<p>(1) L1 lista aktuális elemében lévő kulcs kisebb, mint az L2 lista aktuális elemének kulcsa, ekkor L1 lista bejáró pointerre tovább lép, L2 lista bejárója nem lép.</p>	 <p>p tovább lép, q nem lép:</p>
<p>(2) L1 lista aktuális elemében lévő kulcs egyenlő az L2 lista aktuális elemében lévő kulccsal. Ekkor az L2 listában lévő elemet ki kell láncolni, fel kell szabadítani, a bejáró pointerok mindkét listában tovább lépnek.</p>	 <p>q elemet kiláncoljuk, felszabadítjuk, a bejáró pointerok mindkét listában tovább lépnek.</p>
<p>(3) L1 lista aktuális elemében lévő kulcs nagyobb, mint az L2 lista aktuális elemében lévő kulcs. Ekkor az L2 listában lévő elemet ki kell láncolni, át kell fűzni L1 lista aktuális eleme elé, L2 listában tovább lépünk, L1 listában nem.</p>	 <p>q elemet kiláncoljuk, befűzzük p elé, q-val a következő elemre lépünk, p-val nem lépünk (jöhetne L2-ből az 5 és 6 kulcs)</p>

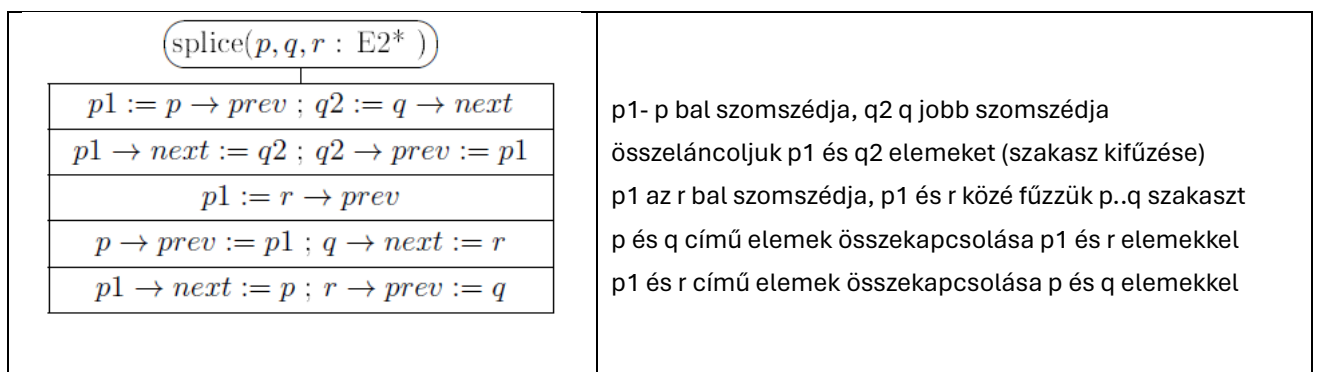
Megoldás:



Hatékonyabbá tehetjük a megoldásunkat, ha amikor főciklusból kilépve, az L2 listában még vannak elemek, azokat nem ciklussal fűzzük át L1 listába, hanem az L2-beli lánc-darabot a két végénél fogva konstans lépésben fűzzük L1 lista végére. Ez már nem oldható meg az alapl műveletekkel, de a jegyzetben definiálva van egy splice nevű elemi algoritmus, mely egy listadarabot tud átfűzni egy adott elem elé.

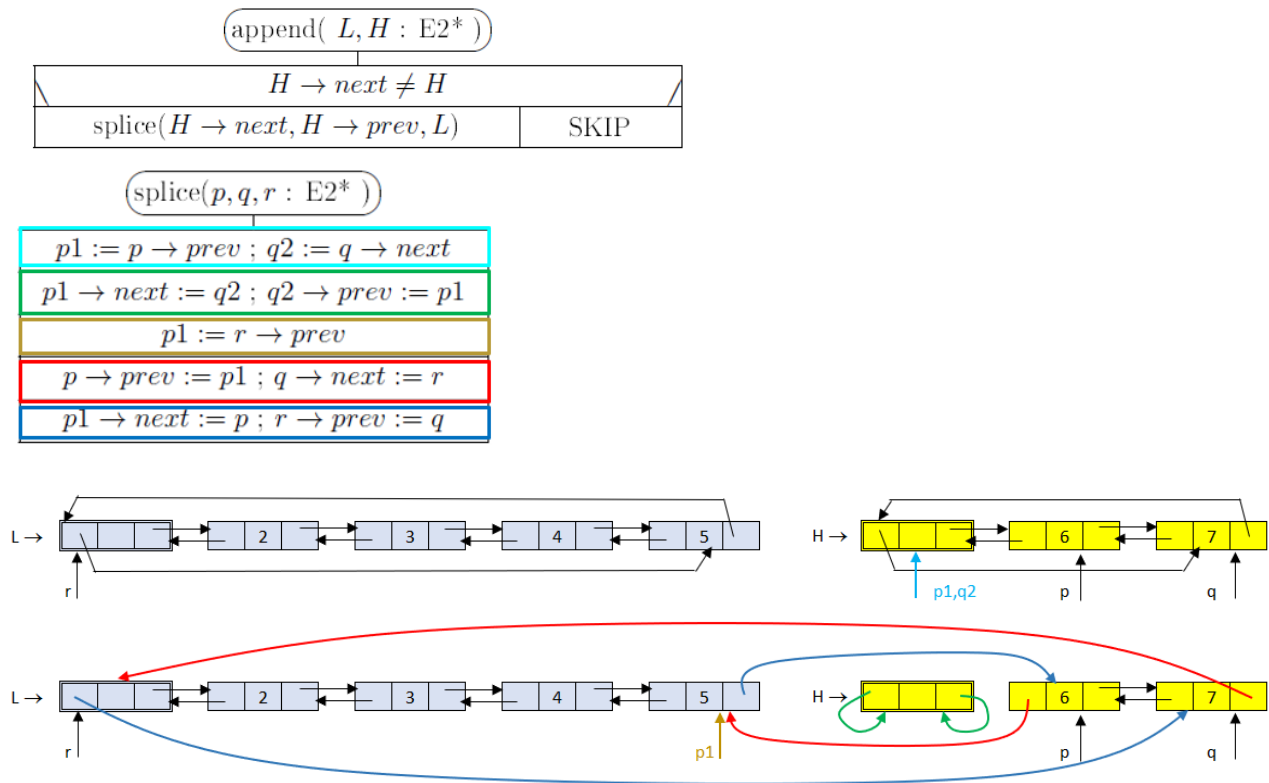
Megoldás: append – splice algoritmusok a jegyzetből (7.15 példa):

A splice(p, q, r : E2\*) egy olyan elemi listaművelet, amely egy C2L egy adott [p, . . . , q] szakaszát eltávolítja, majd az eltávolított szakaszt egy C2L adott \*r eleme elé befűzi. (Előfeltétel, hogy p..q szakasz ne tartalmazza a fejelemet, p után jöjjön q (p=q lehet), valamint \*r ne legyen a p..q szakasz egy eleme, de r lehet egy másik C2L listának az eleme.)

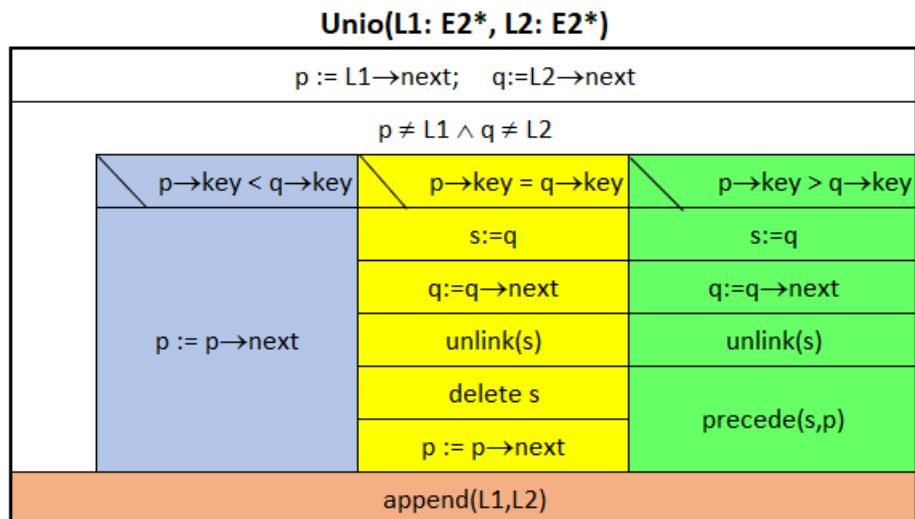


Ennek segítségével definálhatjuk az append műveletet, amely egy L C2L lista végére fűzi egy H nem üres C2L lista összes elemét.

Szemléltetés: L végére fűzzük H elemeit:



Az append használatával a megoldás:





## Műveletigény

- Vizsgáljuk meg az összefésülő algoritmusunk műveletigényét! Legyen L1 lista hossza:  $n$ , L2 lista hossza  $m$ .
- Mit mondhatunk  $mT(n,m)$  illetve  $MT(n,m)$  műveletigényről?
- Az algoritmus az egyik listán mindenképp végig iterál, tehát:  
 $mT(n,m) = \Theta(n)$  abban az esetben, ha L1 minden eleme kisebb, mint L2 első eleme.  
 Miért? Indokoljuk az állítást!  
 $mT(n,m) = \Theta(m)$  abban az esetben, ha L2 minden eleme kisebb, mint L1 első eleme.  
 Miért? Indokoljuk az állítást!
- $MT(n,m) = \Theta(n+m)$  Például pontosan  $n+m$  iterációt hajt végre abban az esetben, ha nincsenek azonos elemek, és L1 utolsó előtti eleme kisebb, a legutolsó pedig nagyobb, mint L2 legutolsó eleme. Indokoljuk meg, miért?

## Gyakorló feladat 4

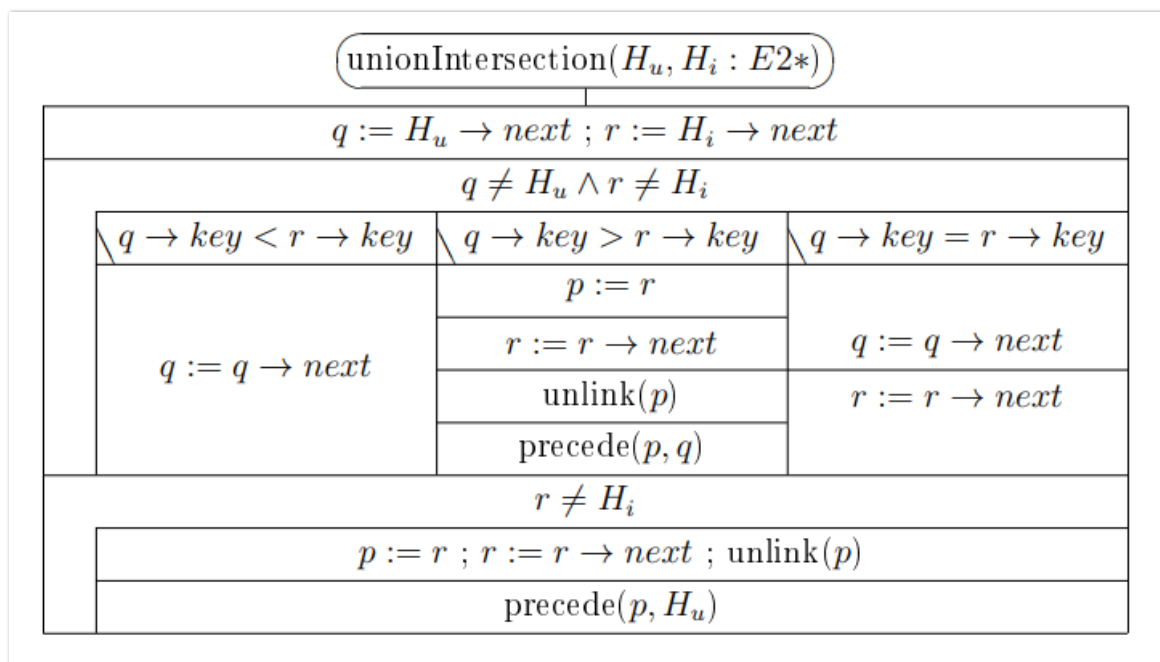
H1L listák összefésülése.

A jegyzetben található a következő algoritmus (7.22 példa):

- Legyenek  $H_u$  és  $H_i$  szigorúan monoton növekvően rendezett C2L-ek!
- Írjuk meg a  $\text{unionIntersection}(H_u, H_i : E2^*)$  eljárást, ami a  $H_u$  listába  $H_i$  megfelelő elemeit átfűzve, a  $H_u$  listában az eredeti listák, mint halmazok unióját állítja elő, míg a  $H_i$  listában a metszetük marad!
- Ne alokáljunk és ne is deallokáljunk listaelemeket, csak az listaelemek átfűzésével oldjuk meg a feladatot!  $MT(n_u, n_i) \in \Theta(n_u + n_i)$ , ahol a  $H_u$  C2L hossza  $n_u$ , a  $H_i$  C2L hossza pedig  $n_i$ . Mindkét lista maradjon szigorúan monoton növekvően rendezett C2L!

Készítsük el az algoritmust ugyanezen feltételekkel H1L listára!

A jegyzet algoritmus:



Ezt írjuk át H1L listákra:

unionIntersection( Hu, Hi : E1*)			
qe:=Hu; q:=Hu->next; re:=Hi; r:=Hi->next			
q ≠ 0 ∧ r ≠ 0			
<div>q-&gt;key &lt; r-&gt;key</div>	<div>q-&gt;key &gt; r-&gt;key</div>	<div>q-&gt;key = r-&gt;key</div>	
<div>qe:=q q:=q-&gt;next</div>	<div>re-&gt;next:=r-&gt;next</div>	<div>qe:= q q:=q-&gt;next  re:=r r:=r-&gt;next</div>	
	<div>qe-&gt;next:=r</div>		
	<div>r-&gt;next:=q</div>		
	<div>qe:=r</div>		
	<div>r:=re-&gt;next</div>		
r ≠ 0			
<div>qe-&gt;next := r</div>		<div>skip</div>	
<div>re-&gt; next:= 0</div>			

## Szorgalmi házi feladatok:

1. Legyen L egy C2L lista fejelemére mutató pointer. A lista egész számokat tartalmaz, rendezzük minimum kiválasztó algoritmussal. Az algoritmusban a tanult listakezelő műveleteket (precede, follow, unlink) kell használni! Ötlet:

- Keressük meg a minimális kulcsú elemet, és fűzzük ki a listából.
- Első esetben a fejelem után kell befűzni, később mindig az utoljára befűzött elem mögé.
- Így a lista elején kezd kialakulni a rendezett rész. Jegyezzük meg, hogy meddig tart ez a rendezett rész, mi az utolsó elemének a címe (legyen ez az r pointer feladata).
- A minimum kiválasztást az r utáni elemekre kell végrehajtani, a kapott elemet r után kell befűzni, majd r-et tovább kell léptetni.
- Ha r után már csak egy elem van, készen vagyunk.

2. A hallgatók egy adott tárgyból akkor szerezhettek jegyet, ha a zárthelyiket legalább 2-esre teljesítik, és elkészítik a beadandót. Adott két C2L lista: L1 és L2. L1-ben vannak azok a hallgatók, akik a zárthelyiket teljesítették, L2-ben azok, akiknek a beadandóját elfogadták. Mindkét lista Neptun kód szerint szigorúan monoton növekvően rendezett. (Lehet olyan hallgató, aki csak az egyik vagy csak a másik listában szerepel.) Készítsen algoritmust, mely L1 listából kiválogatja és átfűzi egy L3 listába azokat, akiknek nincs elfogadott beadandójuk, de a zárthelyiket teljesítették. L1 listában maradjanak azok, akik tárgyat sikeresen teljesítették (Zh+beadandó), L2 listából pedig töröljük azokat, akik nem szerepelnek L1 listában, azaz nem teljesítették a ZH-kat. Tegyük fel, hogy az E2 típus „key” adattagja a Neptun kódot tartalmazza.

- Ha a hallgató szerepel az L1 és L2 listában is, akkor rekordját (listaelemét) hagyjuk mind L1 mind L2 listában.
- Ha csak L1 listában szerepel, akkor rekordját fűzzük ki, és láncoljuk át L3 listába.
- Ha csak L2 listában szerepel, akkor rekordját töröljük L2 listából.
- L3 legyen egy új C2L lista, az algoritmusban hozzuk létre, és fejelemének címét adjuk vissza. L3 is legyen Neptun kód szerint növekvően rendezett.