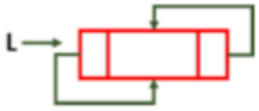
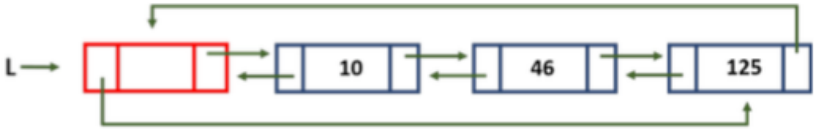
5. Gyakorlat (2019.03.12.)

Fejelemes kétirányú ciklikus listák (C2L)

Alapvetően **C2L** alatt a fejelemes kétirányú listát értjük. A C2L elemei két pointert tartalmaznak, az egyik (**prev**) az aktuális elemet megelőző, a másik (**next**) a következő elemre mutat. Használjuk a fejelemet is, a műveletek átláthatóbb és egyszerűbb megvalósítása miatt. A C2L ciklikus, azaz az utolsó elemének *next* pointere a fejelemre, a fejelem *prev* pointere pedig az utolsó elemre mutat. Az üres C2L lista egy fejelemből áll, melynek mindkét pointere magára a fejelemre mutat. Az ilyen listák elemeinek ábrázolásához az E2 nevű osztályt használjuk.



1. ábra: Üres C2L lista



2. ábra: C2L lista

Az E2 osztály[1]

E2
+ prev, next: E2* + key: T
+ E2() {prev = next = this} //*konstruktor*

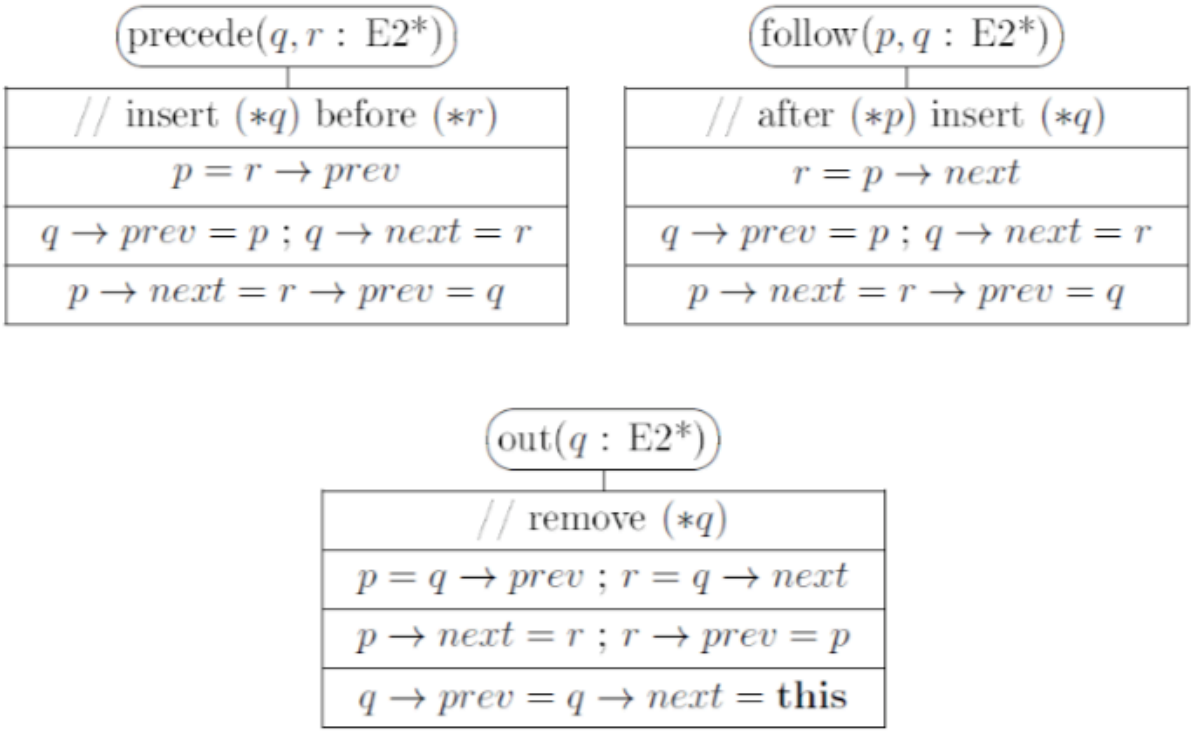
Vegyük észre, hogy az elem konstruktora a pointereket úgy állítja be, hogy azok magára az elemre mutassanak! Ezt kihasználva egy új fejelemes C2L lista fejelemének létrehozása L := new E2 utasítással történhet!

A C2L listákhoz definiált műveletek[1]

- **precede metódus:** listaelem beszúrása egy másik lista elem elé. A metódus első paramétere a beszúrandó listaelemre mutató pointer, a második paramétere arra a listaelemre mutató pointer, ami elé az első paramétert akarjuk beszúrni.
- **follow metódus:** a precede metódushoz hasonló, itt most az első paraméterben lévő listaelem után szúrjuk be a második paraméterben lévő listaelemet.

1- Dr. Ásványi Tibor jegyzete alapján

- **out metódus:** a megadott listaelem kifűzése a listából. A metódus a kifűzött listaelem pointereit önmagára állítja.

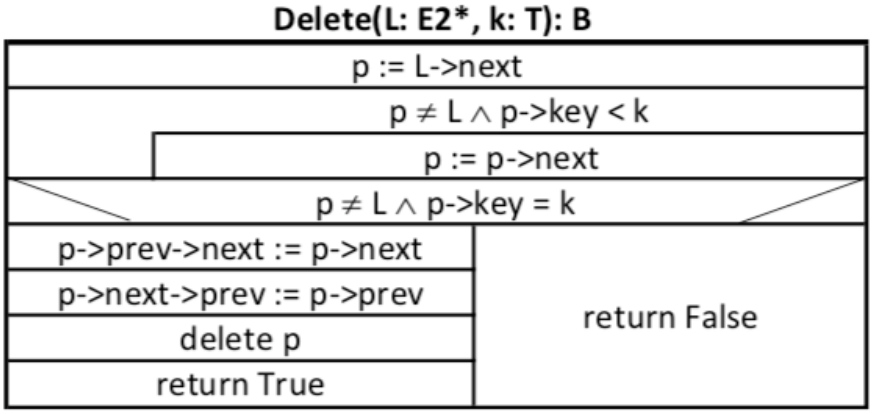


3. ábra: C2L listák alapműveletei

Az algoritmusokban használt pointerek nevének logikája: ábécé sorrendben a három használt pointer: *p q r* ez mindig három egymás utáni listaelemet jelöl a listából, ebben a sorrendben. A két befűző műveleteknél *q* - t fűzzük *p* és *r* közé, ehhez vagy *p* - t, vagy *r* – et adjuk meg paraméterként. Törlésnél elég *q* – t megadni, de itt is *p* – t és *r* – et használ a két szomszéd címének tárolásához.

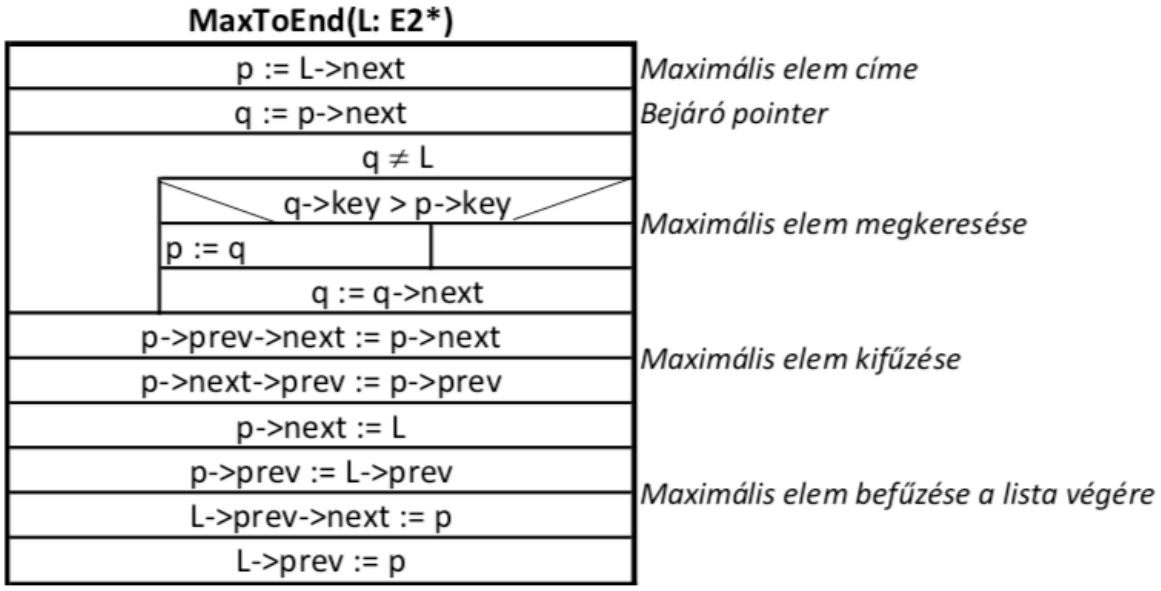
Törlés C2L listából

Készítsünk egy függvényt, ami paraméterül kapott **szigorúan monoton növekvően rendezett** C2L listából törli a paraméterül kapott kulcsú elemet, amennyiben ilyen van. A függvény logikai értékkel tér vissza, ami a törlés sikerességét jelzi. Használjuk a lista rendezettségét.

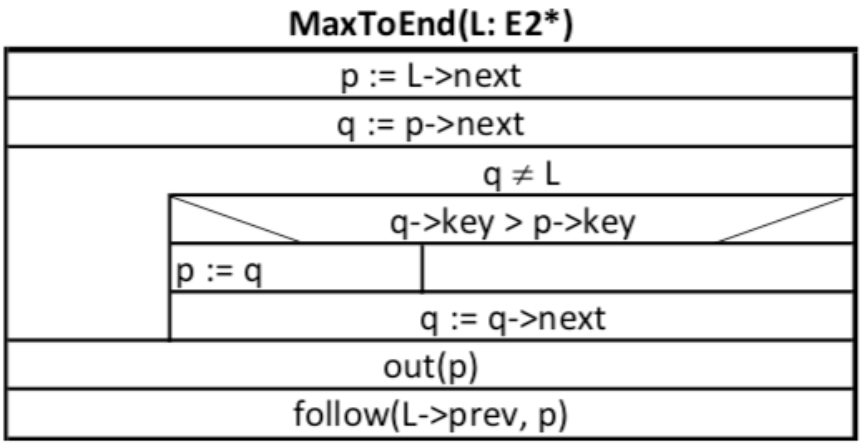


Maximális elem a lista végére

Készítsünk egy listát, ami egy C2L lista maximális kulcsú elemét a lista végére fűzi. Tegyük fel, hogy a listában minden kulcs csak egyszer szerepel.



Használjuk a korábban ismertetett **out** és **follow** műveleteket. Alakítsuk át az algoritmust, hogy ezek meghívásával hajtsa végre a feladatot.

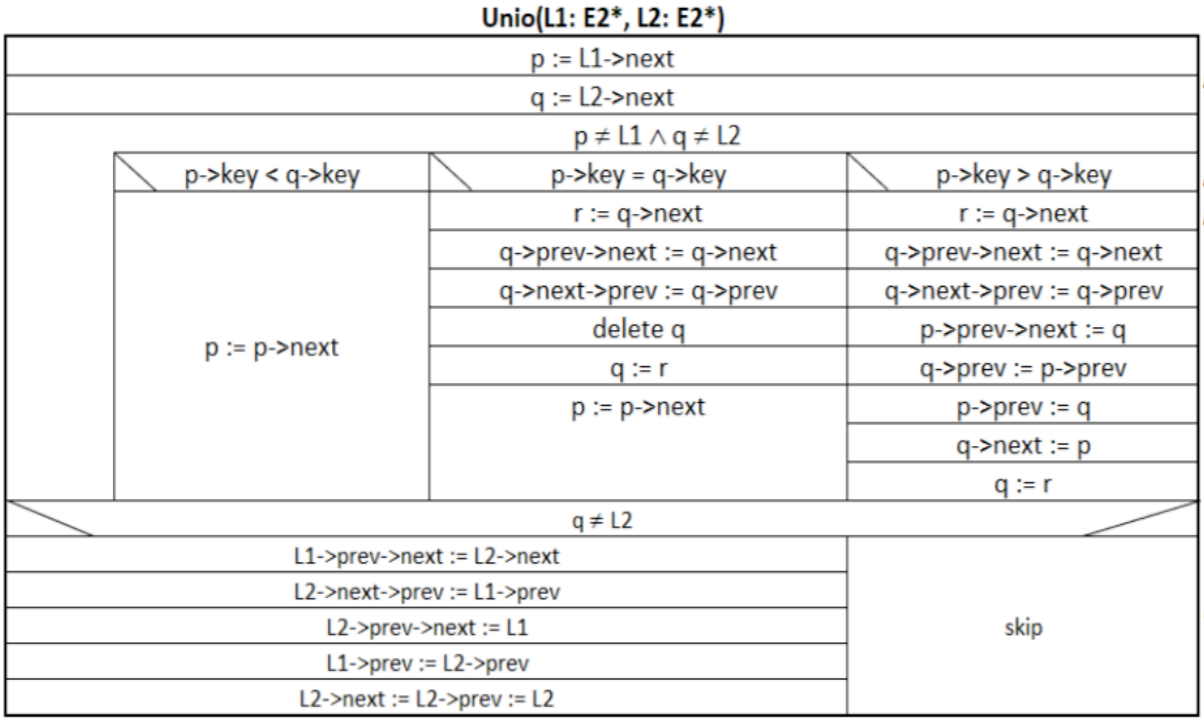


Halmazok uniója

Adott két szigorúan monoton növekvően rendezett C2L lista (halmazt ábrázolnak): **L1, L2.** L1-ben állítsuk elő a két halmaz unióját. L2 elemeit vagy átfűzzük, vagy felszabadítjuk. Ha végig értünk valamelyik listán, akkor az összefésülő ciklusból kilépve, konstans lépésben fejezzük be az algoritmust.

A megoldás során **összefuttatjuk** a két listát, kihasználva azok rendezettségét. Az azonos kulcsú elemeket töröljük L2 listából. Ha L1 listán végig értünk, de L2 listában még maradnak elemek, akkor L2 elemeit (a fejelem kivételével) L1 végére fűzzük, konstans lépésben.

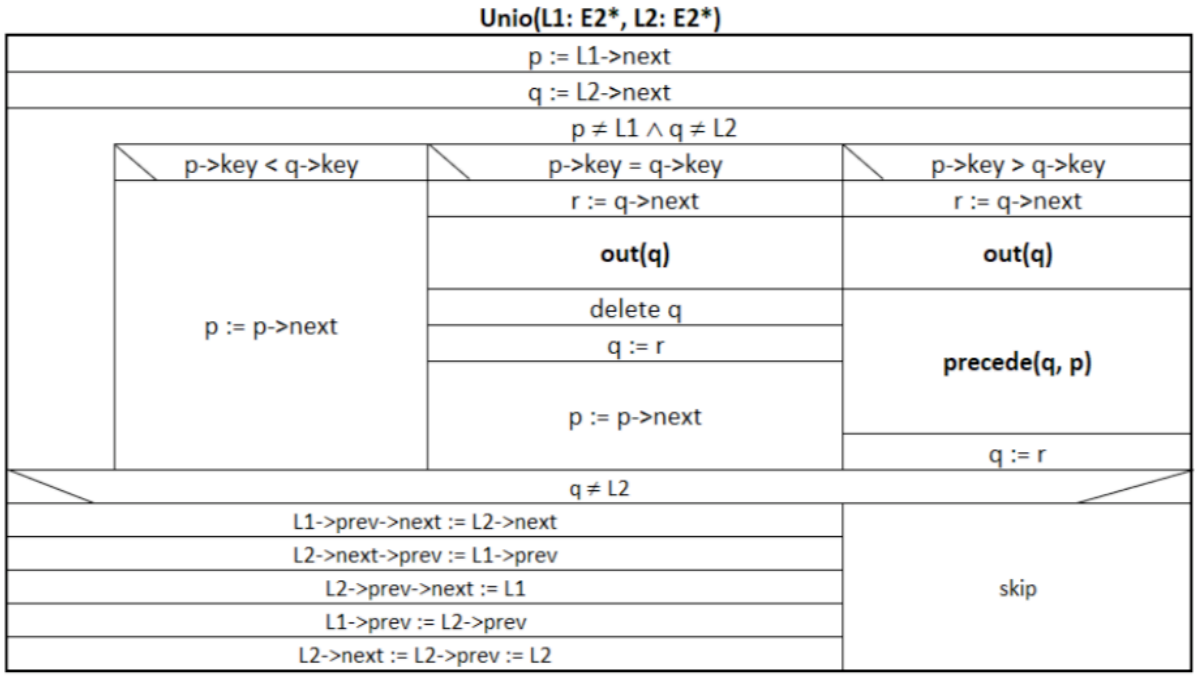
Mivel L2 lista valamennyi elemét kiszedjük a listából, végül L2 fejelemének pointereit az üres listának megfelelően önmagára állítjuk.



Megjegyzések az algoritmushoz:

- **p** pointerrel L1, **q** pointerrel L2 listán iterálunk,
- **p->key = q->key** esetén, q elemet kifűzzük L2 listából, majd felszabadítjuk, hiszen az unióban csak egyszer szerepelhet egy adott kulcsú elem,
- **p->key > q->key** esetén a **q** című elemet átfűzzük L1 listába **p** elem elé,
- az algoritmus végén L2 listát üresre állítjuk.

Az algoritmuson egyszerűsíthetünk, ha használjuk a C2L listához készült metódusokat:



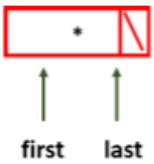
Sor adattípus

A sor egy **FIFO (First In First Out)** adatszerkezet, azaz ellentétben a veremmel a legelőször behelyezett elemet vehetjük ki elsőként. Számos implementáció pl. statikus vagy dinamikus tömbbel (ld. előadás).

Sor megvalósítása egyirányú listával

Két pointert alkalmazunk. A lista első eleme a sornak is az első eleme, erre a first pointer mutasson. A lista végére (ez a sornak is a vége) mutasson a last pointer. Így a műveletek zöme konstans lépésszámú lesz. Kivéve persze a destruktort, és a setEmpty() műveletet, melyeknek le kell bontani a sort ábrázoló listát.

Ötlet: a lista mindig tartalmaz egy fix (joker) lista elemet, ami a sor végén fog elhelyezkedni. Új elem beszúrásakor a beszúrandó kulcsot a joker elemben tároljuk el, majd készítünk egy új üres joker elemet, amit a lista végére fűzünk. A joker elem címét tárolja a last pointer. A lista segítségével elméletileg korlátlan hosszúságú sort hozhatunk létre (amíg a new művelet sikeresen le tud futni).



4. ábra: Üres sor

Queue²
-first, last: E1* *//a sor első és utolsó elemére mutató pointerek* -size: N
+ Queue() + add(x: T) *// új elem hozzáadása a sor végére* + rem(): T *// a sor elején lévő elem eltávolítása* + first() : T *// a sor elején lévő elem lekérdezése* + length(): N + isEmpty(): B + ~Queue() + setEmpty()

Osztály metódusai

Queue::Queue()

first := last := new E1
first->next = null
size := 0

A konstruktor létrehozza a joker elemet.

Queue::add(x: T)

last->next := new E1
last->key := x
last := last->next
last->next := null
size := size + 1

Queue::rem(): T

	size = 0
Error	x := first->key
	s := first
	first := first->next
	delete s
	size := size - 1
	return x

Queue::first(): T

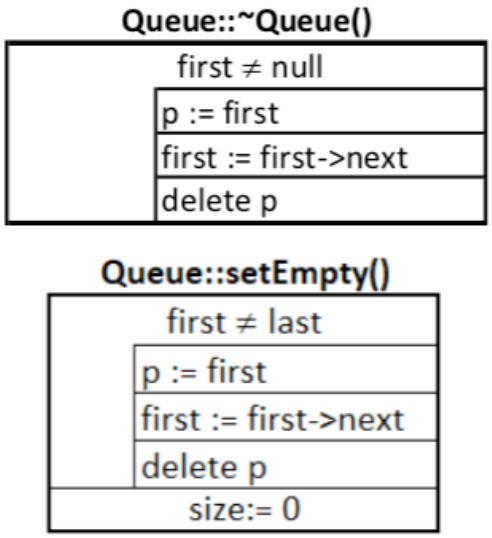
	size = 0
Error	return first->key

Queue::length(): N

return size

Queue::isEmpty(): B

return size = 0



Példa a sor alkalmazására

Oldjuk meg két sor segítségével a következő feladatot:
Olvassunk be karakterenként egy szöveget (hossza nem ismert), és döntsük el, hogy „dadogós”-e. Pl.: *abcabc* dadogós, *abccbb* nem dadogós.

Az első sorba beolvassuk a teljes szöveget karakterenként. Ha a sor mérete páratlan, akkor a beolvasott szöveg biztosan nem „dadogós” ezért nem folytatjuk tovább a vizsgálatot. Ha az első sor mérete páros, akkor az első sor első felét átmozgatjuk a második sorba. Ezt követően egy közös ciklussal végig megyünk mindkét soron a ciklus minden lépésében kivesszük mindkét sorból az első elemet és összehasonlítjuk őket, ha nem egyeznek meg, akkor a szöveg nem „dadogós”.