



# STRUKTURE PODATAKA

## LETNJI SEMESTAR

# LANČANE LISTE

## (LINKED LISTS)

*Prof. Dr Leonid Stoimenov*  
*Katedra za računarstvo*  
*Elektronski fakultet u Nišu*

# LANČANE LISTE - PREGLED

- Uvod
- Definicija
- Memorijska reprezentacija
- Operacije
- Tipovi listi



# UVOD U LANČANE LISTE

## ○ Definicija liste:

- **Lista** je kolekcija elemenata podataka kojima se pristupa redom polazeći od **glave** liste i završavajući sa **repom** liste
- Glava – prvi element liste
- Rep – poslednji element liste
- **Veličina liste** je broj elemenata liste
- **Linearna lista** ili **lista** je linearni skup elemenata podataka (objekata)

- Lista može biti sortirana ili nesortirana

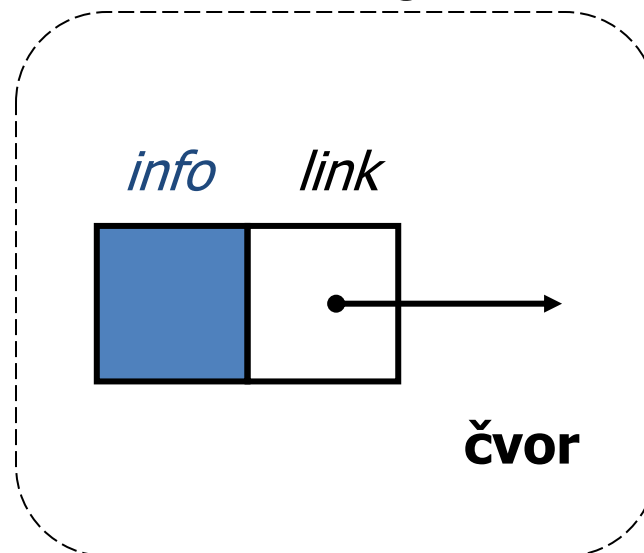
# VRSTE LANČANIH LISTI

- Jednostruko spregnute ili jednosmerne
- Dvostruko spregnute ili dvosmerne
- Necirkularne
- Cirkularne
- Sa zaglavljem
- Bez zaglavlja
- Sortirane
- Nesortirane



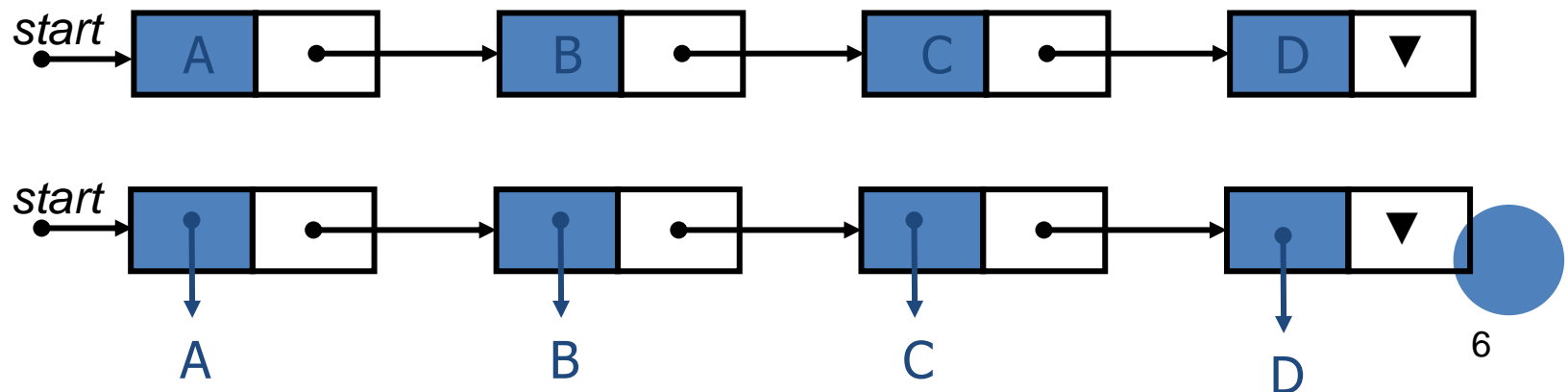
# JEDNOSTRUKO SPREGNUTA LANČANA LISTA

- **Jednostruko spregnuta lančana lista** (*singly linked list*) je struktura podataka koja se sastoji od sekvence čvorova
- Svaki **čvor** sadrži dva polja:
  - **info** – pamti element liste ili adresu elementa liste
  - **link** – pamti adresu sledećeg čvora



# JEDNOSTRUKO SPREGNUTA LANČANA LISTA

- **Jednostruko spregnuta lančana lista** (*singly linked list*) je struktura podataka koja se sastoji od sekvence čvorova
- Listi se pristupa preko eksternog pokazivača **start** koji ukazuje na **prvi element** liste
- **Poslednji element** liste sadrži u link polju specijalnu vrednost – **null** (▼) koja nije validna adresa



# PRAZNA LISTA

- Lista koja nema elemenata se naziva *prazna lista (empty list, null list)*
- Prazna lista ima  
start == null



*start*



# PREDNOSTI I MANE LANČANE LISTE

## ○ Prednosti

- **Dinamička struktura podataka** - veličina liste nije fiksirana, već se dinamički menja
- **Efikasne operacije umetanja i brisanja**,  $O(n)$ , gde je  $n$  veličina liste

## ○ Mane

- **Nemoguć direktan pristup elementima**

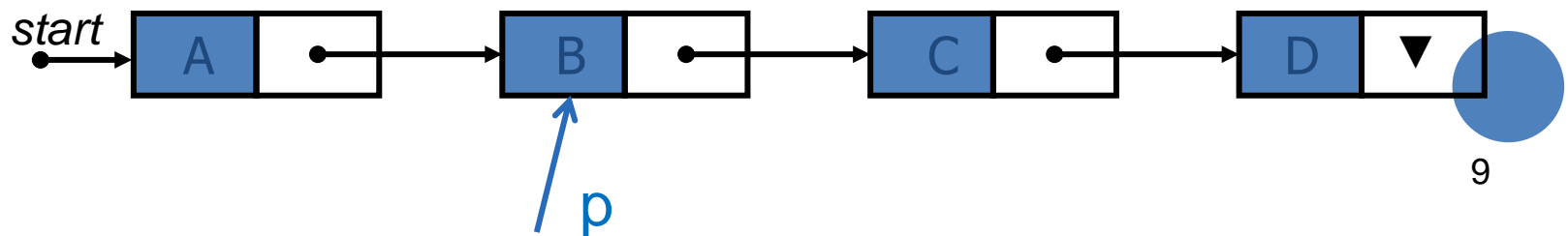
## ○ Upotreba

- Za memorisanje promenljive količine podataka kojima se sekvencijalno pristupa
- Za formiranje složenijih struktura – magacina, redova, deкова, tablica, stabala, grafova

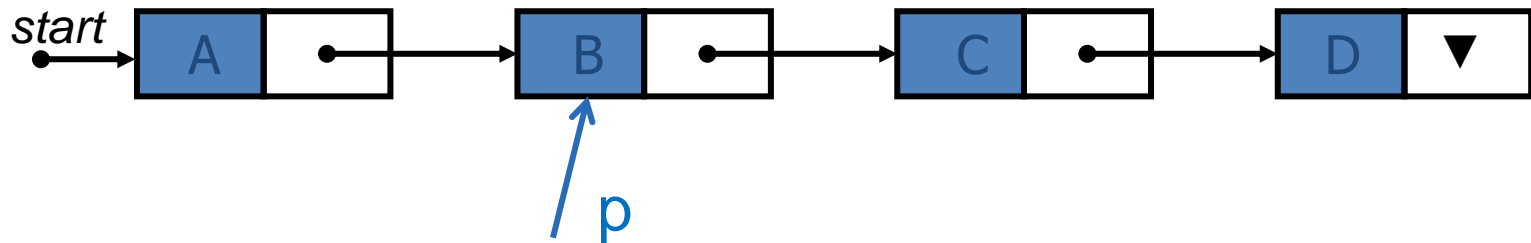


# NOTACIJA KOJA ĆE SE KORISTITI

- $p$  je pokazivač
- $node(p)$  je čvor na koji ukazuje pokazivač  $p$
- $info(p)$  je *info* polje čvora  $node(p)$
- $link(p)$  je *link* polje čvora  $node(p)$
- ako  $link(p) \neq 0$ , tada  $info(link(p))$  označava *info* polje čvora koji sledi čvor  $node(p)$  u listi



# NOTACIJA – PRIMERI



- $E \leftarrow info(p)$ , vrednost  $E$ ?
  - $q \leftarrow link(p)$ , vrednost  $q$ ?
  - $info(q) \leftarrow E$ , vrednost  $info(p)$ ?
  - $r \leftarrow link(link(p))$ , vrednost  $q$ ?
  - $E \leftarrow info(link(r))$ , vrednost  $E$ ?
- 
- Kako sada izgleda LL?



# OPERACIJE

- **Obilazak liste**
- **Traženje elementa u listi**
  
- **Umetanje čvora u listu**
- **Brisanje čvora iz liste**
  
- **Brisanje liste**
- **Kopiranje liste**
- **Cepanje liste**
- **Spajanje (konkatenacija) listi**



# OBILAZAK JEDNOSTRUKO SPREGNUTE LANČANE LISTE

**Algoritam SLL.1.** Obilazak lančane liste

**Traversal(START)**

*// Ovaj algoritam obilazi listu i nad svakim elementom polja*

*// primenjuje operaciju OBRADA*

*// Pokazivač POK stalno ukazuje na čvor koji će se sledeći obrađivati*

1 POK ← START      */\* POK se postavlja na prvi element liste \*/*

2 **while**(POK ≠ NULL)

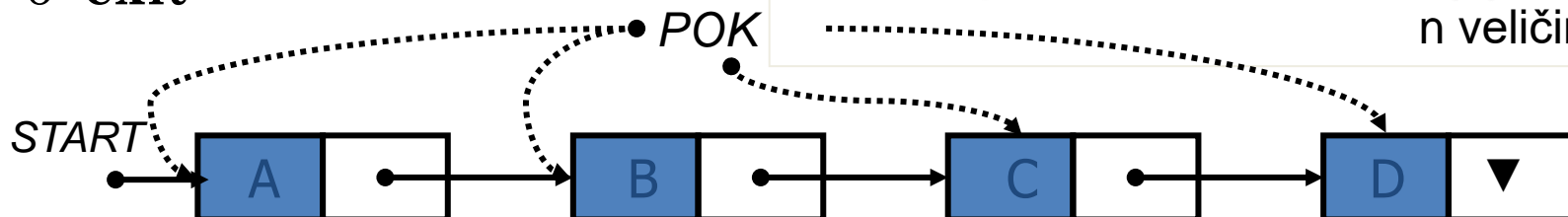
3    OBRADA(info(POK))

4    POK ← link(POK)   */\* POK se postavlja na sledeći čvor liste \*/*

5 **endwhile**

6 **exit**

**Asimptotska složenost:**  $O(n)$ , linearna  
n veličina liste



# TRAŽENJE U JEDNOSTRUKO SPR. L. LISTI

**Algoritam SLL.2.** Linearno traženje (u nesortiranoj listi)

**searchNonSorted(START,E,LOC)**

// Ovaj algoritam traži element  $E$  u nesortiranoj lančanoj listi  $START$

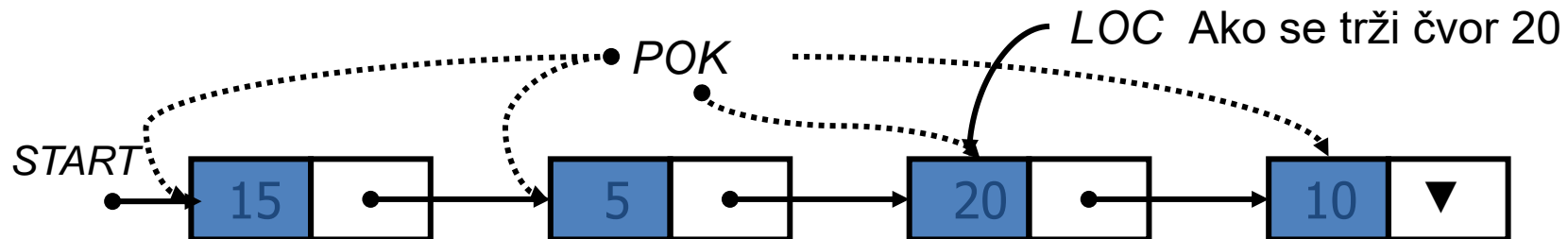
// i vraća njegovu lokaciju  $LOC$ , ako je traženje uspešno,

// ili  $LOC=NULL$ , ako je traženje neuspešno

```
1  POK ← START
2  while (POK ≠ NULL AND info(POK) ≠ E)
3      POK ← link(POK)
4  endwhile
5  if info(POK) = E
6  then
7      LOC ← POK          /*traženje uspešno*/
8  else
9      LOC ← NULL         /*traženje neuspešno*/
10 exit
```

**Asimptotska  
složenost:**

**$O(n)$ ,**  
linearna



# PRIMER: TRAŽENJE U SORTIRANOJ LISTI

**Algoritam SLL.3.** Linearno traženje u sortiranoj listi

**searchSorted(START,E,LOC)**

*// Ovaj algoritam traži element E u sortiranoj lančanoj listi START.*

*// Vraća njegovu lokaciju LOC, ako je traženje uspešno, ili je*

*// LOC=NULL, ako je traženje neuspešno*

```
1  POK ← START
2  while (POK≠NULL)
3      if(info(POK)=E) then
4          LOC ← POK
5          exit      /*traženje uspešno*/
6      else if(E<info(POK) then
7          LOC ← NULL
8          exit      /*traženje neuspešno*/
9      else
10         POK ← link(POK)
11     endif      /*kraj if strukture */
12 endwhile
13 LOC ← NULL /*traženje neuspešno*/
14 exit
```

**Asimptotska  
složenost:**

**$O(n)$ ,**  
linearna

**Da li se dobija nešto ako  
je lista sortirana?**

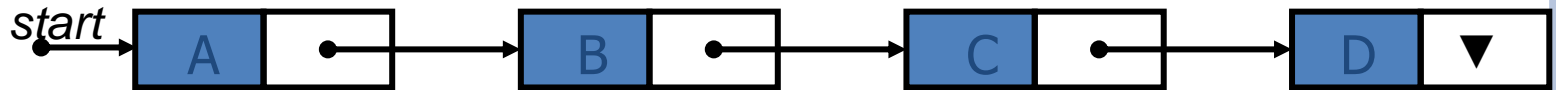


# UMETANJE U LANČANU LISTU

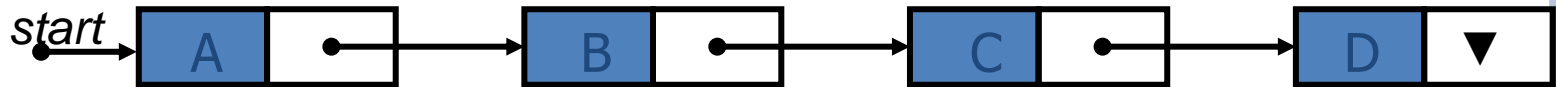
- Ovom operacijom se **umeće novi čvor** u lančanu listu
- Novi čvor se umeće:
  - Na početak liste
  - Na kraj liste
  - Ispred ili iza zadatog čvora
- Čvor se zadaje
  - Pozicijom u listi (prvi, drugi,...)
  - **Pokazivačem (adresom)**
  - **Vrednošću info polja** (npr. čvor koji sadrži D)



# UMETANJE NA POČETAK LISTE - PRIMER

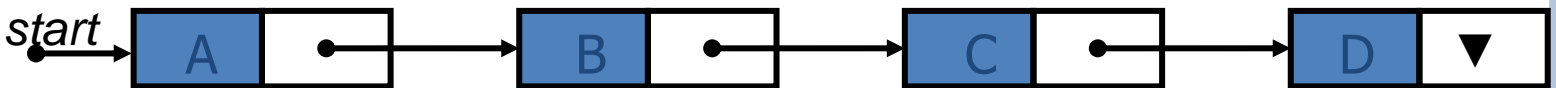


①



1  $\text{novi} \leftarrow \text{getnode}()$  //uzimanje praznog čvora iz LRMP

②

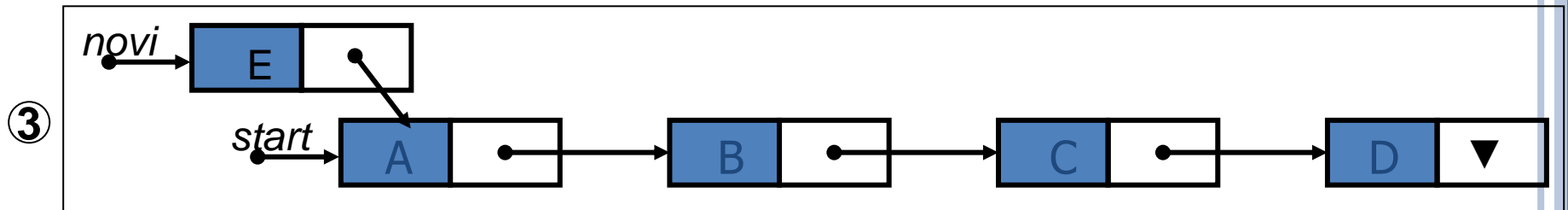


2  $\text{info}(\text{novi}) \leftarrow E$  //upis elementa E u novi čvor

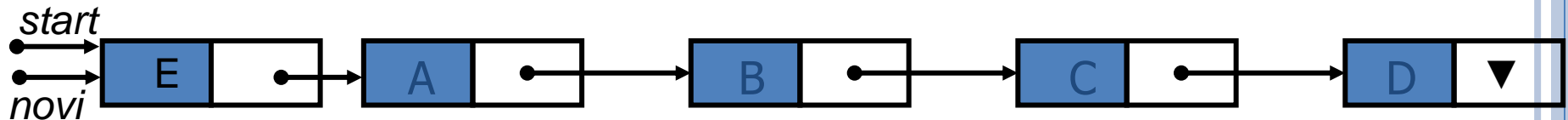




# UMETANJE NA POČETAK LISTE - PRIMER



3     $\text{link}(\text{novi}) \leftarrow \text{START}$     //upis ukazivača na čvor



4     $\text{START} \leftarrow \text{novi}$     //izmena početka liste



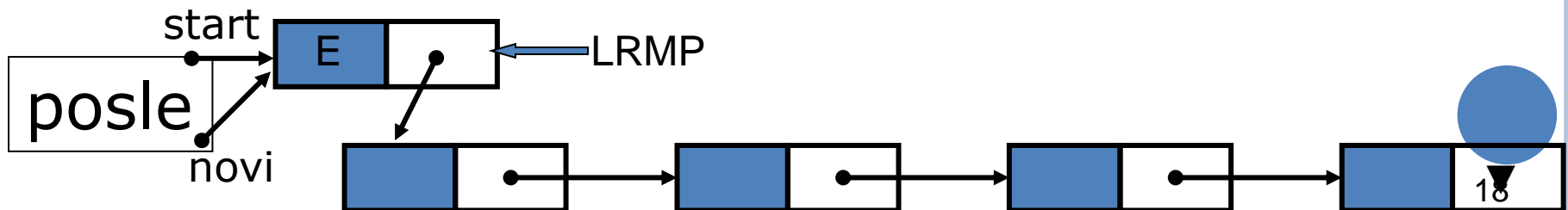
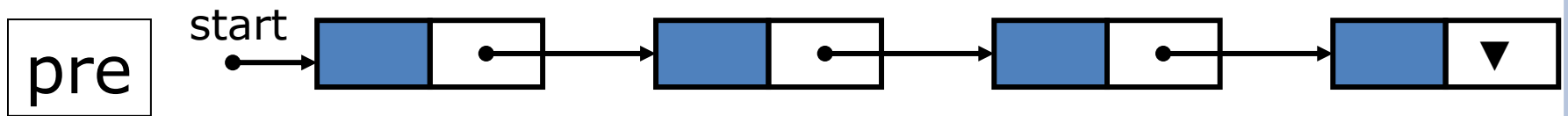
# UMETANJE NA POČETAK LISTE

**Algoritam SLL.4.** Umetanje na početak lančane liste

**insertAtStart(START,E)**

*// Ovaj algoritam umeće element E na početku lančane liste START*

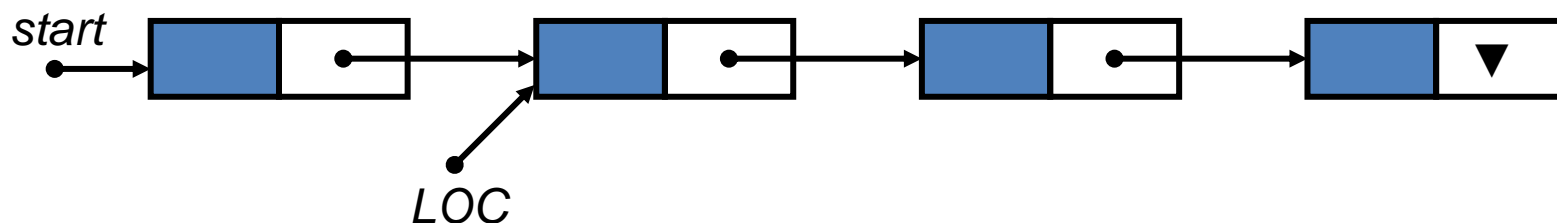
```
1  novi ← getnode()  /*uzimanje praznog čvora iz LRMP*/  
2  info(novi) ← E      /*upis elementa E u novi čvor*/  
3  link(novi) ← START /*povezivanje novog čvora*/  
4  START ← novi      /*izmena početka liste*/  
5  exit              /*kraj algoritma*/
```



# UMETANJE POSLE ZADATOG ČVORA

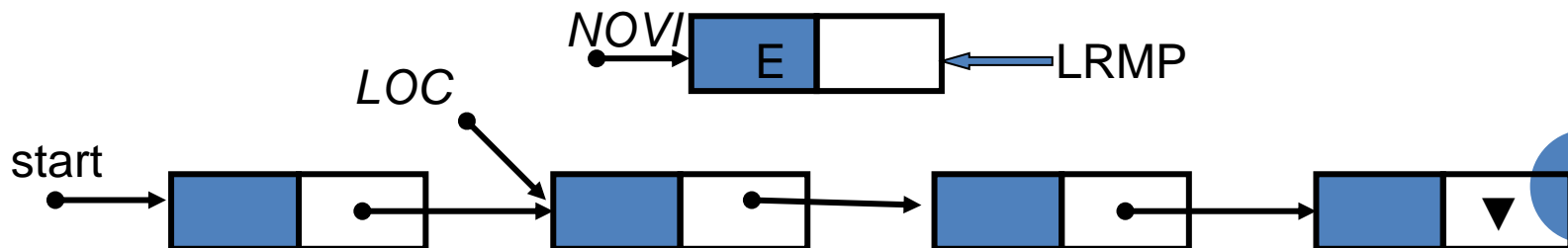
Dodaje se novi čvor koji ima vrednost E iza čvora čija je lokacija LOC zadata

Pre dodavanja: Čvor iza koga se dodaje novi čvor je zadat lokacijom LOC



NOVI se odnosi na novi čvor u koji upisujemo vrednost E

novi  $\leftarrow$  getnode()  
info(novi)  $\leftarrow$  E

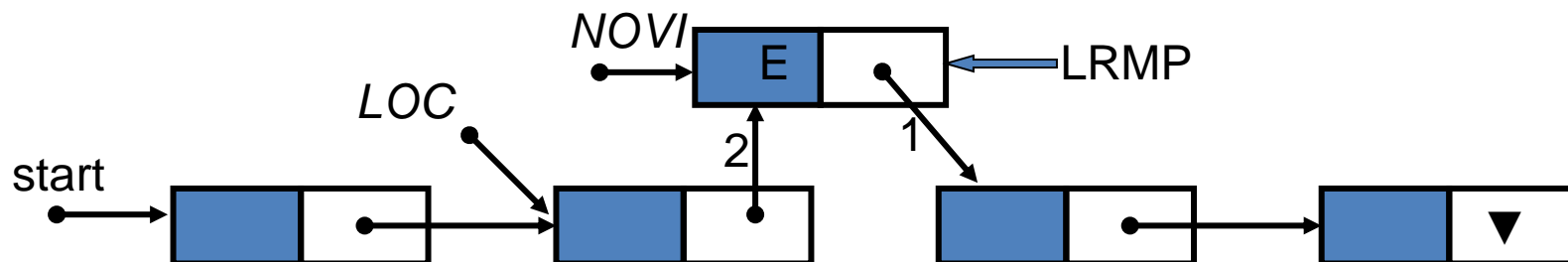


# UMETANJE POSLE ZADATOG ČVORA

Dodaje se novi čvor koji ima vrednost E iza čvora čija je lokacija LOC zadata

**Posle** dodavanja

NOVI se odnosi na novi čvor u koji upisujemo vrednost E



1.  $\text{link}(\text{novi}) \leftarrow \text{link}(\text{LOC})$
2.  $\text{link}(\text{LOC}) \leftarrow \text{novi}$



# UMETANJE POSLE ZADATOG ČVORA

**Algoritam SLL.5.** Umetanje na zadatu lokaciju lančane liste

**insertAfterLoc(START,LOC,E)**

*// Ovaj algoritam umeće element E iza čvora LOC*

*// Ako je LOC=null, E se umeće kao prvi čvor*

1 novi ← getnode()   */\*uzimanje praznog čvora iz LRMP\*/*

2 info(novi) ← E   */\*upis elementa E u novi čvor\*/*

3 **if** (LOC=null) **then**

4     link(novi) ← START

5     START ← novi                   */\*E se umeće kao prvi čvor\*/*

6 **else**

7     link(novi) ← link(LOC)

8     link(LOC) ← novi           */\*E se umeće posle čvora loc\*/*

9 **exit**           */\*kraj algoritma\*/*

# PRIMER: UMETANJE U SORTIRANU LL

**Algoritam SLL.6.** Umetanje u sortiranu lančanu listu

**insertSorted(START,E)**

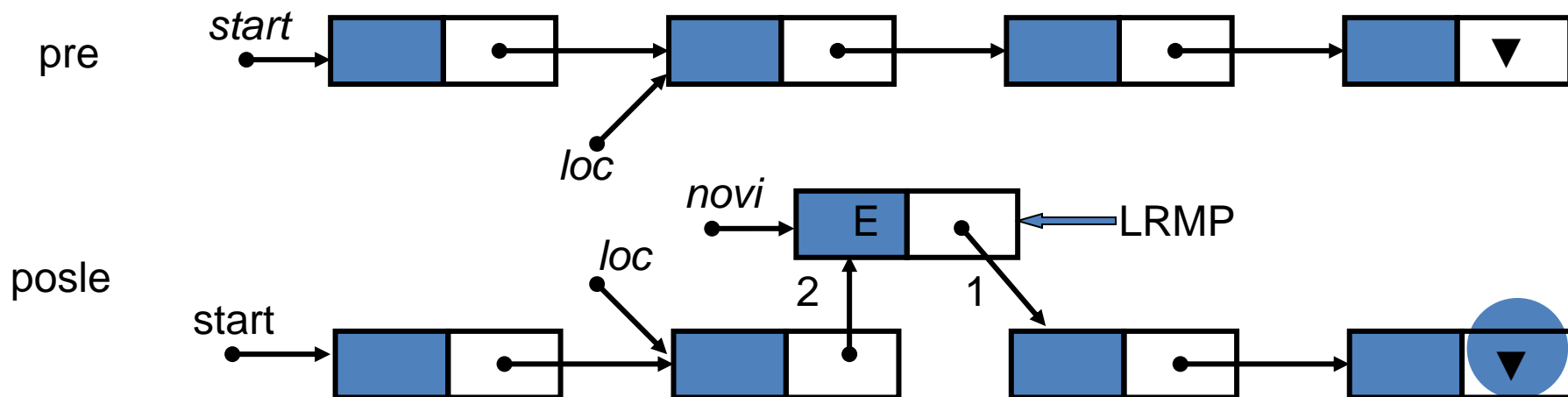
*// Ovaj algoritam umeće element E u sortiranu lančanu listu*

1 **call** findA(start, E, loc)

*/\*nalazi lokaciju čvora koji prethodi čvoru E\*/*

2 **call** insertAfterLoc(start, E, loc) */\*umeće E posle čvora loc\*/*

3 **exit** */\*kraj algoritma\*/*



# TRAŽENJE LOKACIJE ELEMENTA U LL

**Algoritam SLL.7.** Traženje elementa E

**findA(START,E,LOC,LOCP)**

*// Nalazi lokaciju LOC prvog čvora koji sadrži E, kao i lokaciju LOCP čvora koji mu prethodi*

*// Ako E nije u listi, tada je LOC=null, a ako je E u prvom čvoru liste, tada je LOCP=null.*

1 **if** (START=null) **then**

2     LOC ← null

3     LOCP ← null

4     **return**

5 **if**(info(START)=E) **then**

6     LOC ← START

7     LOCP ← null

8     **return**

9   spok ← START

10 npok ← link(START)     */\*inicijalizacija pokazivača\*/*

11 **repeat while** (npok≠NULL)

12   **if** (info(npok)=E) **then**

13     LOC ← npok

14     LOCP ← spok

15     **return**}     */\*traž.uspešno\*/*

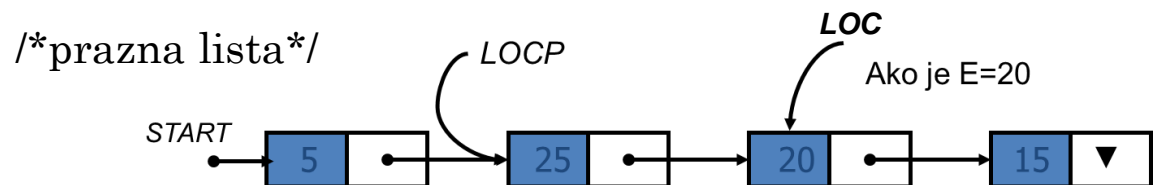
16   spok ← npok

17   npok ← link(npok)     */\*ažuriranje indeksa\*/*

18 **endrepeat**

19 LOC ← null     */\*traženje neuspešno\*/*

20 **return**



*/\*E je u prvom čvoru\*/*

**Asimptotska  
složenost:  
O(n), linearna**



# PRIMER: TRAŽENJE LOKACIJE ELEMENTA U SORTIRANOJ LL

**Algoritam SLL.7.1.** Linearno traženje u sortiranoj lančanoj listi

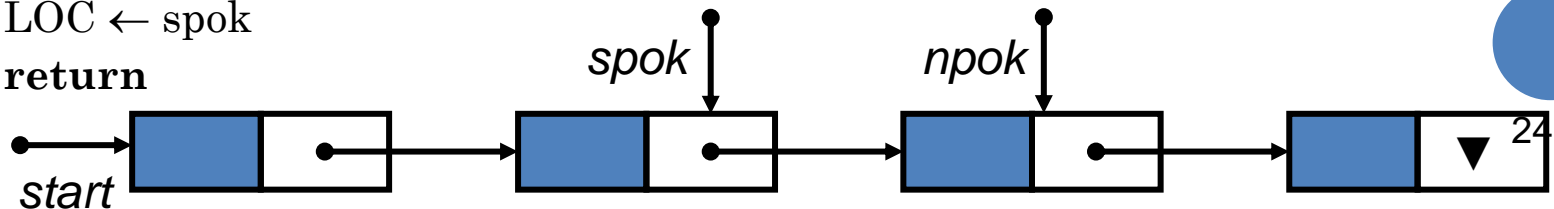
**findB(START,E,LOC)**

*// Nalazi lokaciju LOC poslednjeg čvora u sortiranoj listi čiji je info(LOC)<E*

*// ili vraća LOC=null, ako je traženje neuspešno*

```
1  if (START=null) then
2      LOC ← null
3      return                /*prazna lista*/
3  if (E<info(START)) then
4      LOC ← null
5      return                /*granični slučaj*/
6  spok ← START
7  npok ← link(START)        /*inicijalizacija pokazivača*/
8  while (npok≠NULL)
9      if(E<info(npok))then
10         LOC ← spok
11         return
12     spok ← npok
13     npok ← link(npok)      /*ažuriranje indeksa*/
14 endwhile
15 LOC ← spok
16 return
```

**Asimptotska  
složenost:  
 $O(n)$ , linearna**



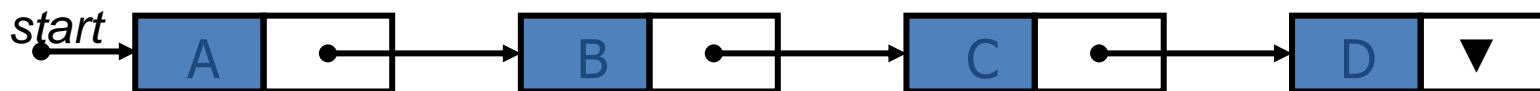


# BRISANJE ČVORA IZ LANČANE LISTE

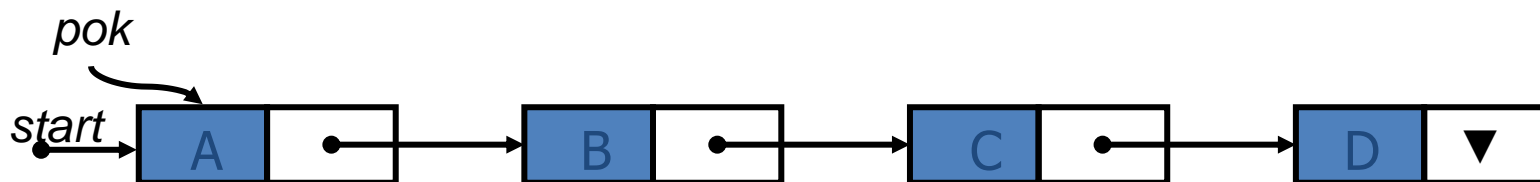
- Ovom operacijom se briše iz lančane liste zadati čvor
- Čvor se briše:
  - Sa početka liste
  - Sa kraja liste
  - Ispred ili Iza zadanog čvora
- Čvor se zadaje
  - Pozicijom u listi (prvi, drugi,...)
  - **Pokazivačem** (adresom)
  - **Vrednošću *info* polja** (čvor koji sadrži E)



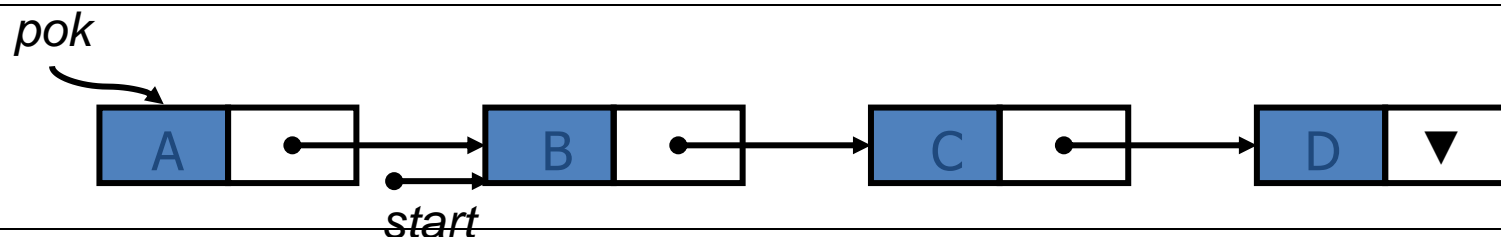
# BRISANJE ČVORA SA POČETKA LISTE - PRIMER



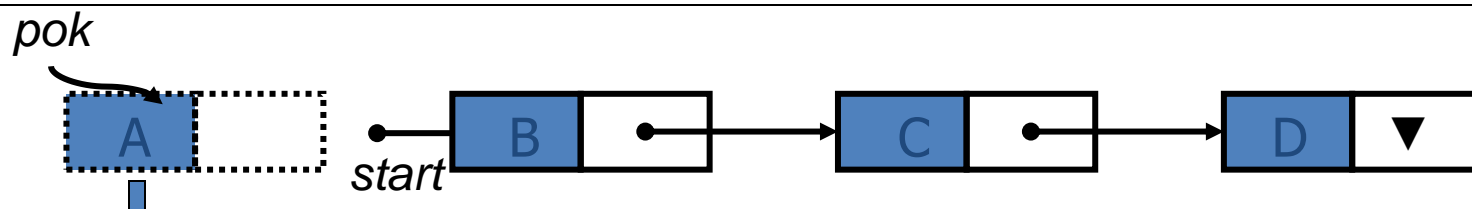
①



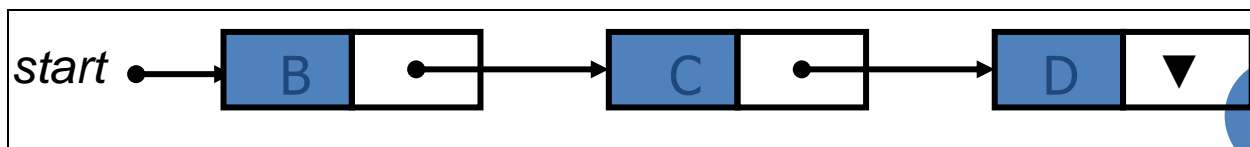
②



③



LRMP



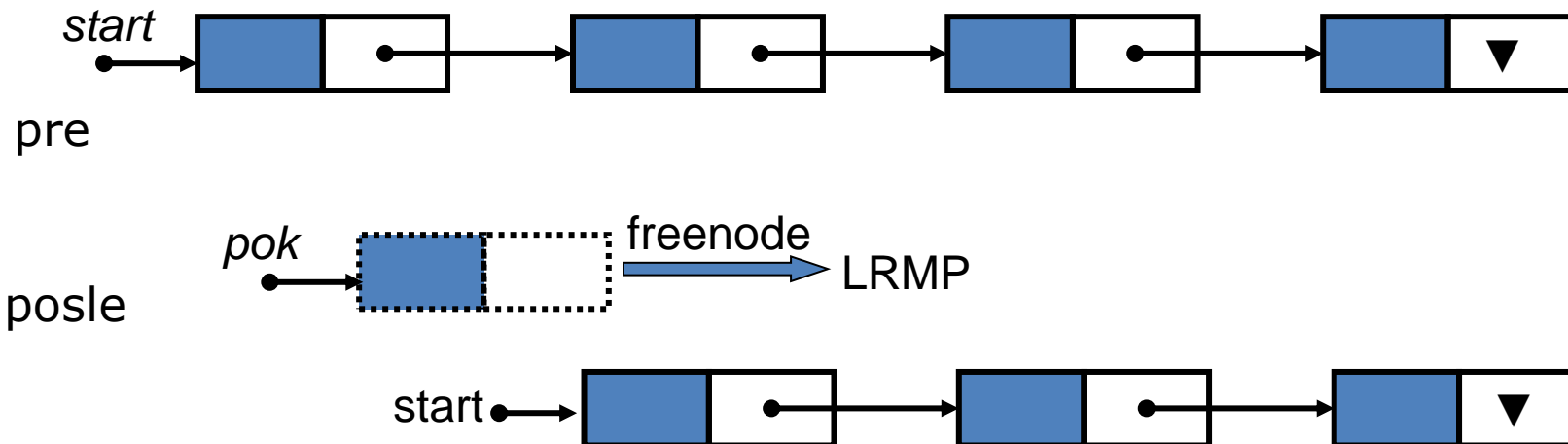
# BRISANJE ČVORA SA POČETKA LISTE - ALGORITAM

**Algoritam SLL.8.** Brisanje čvora sa početka lančane liste

**deleteFromStart(START,E)**

*// Ovaj algoritam briše element E sa početka lančane liste START*

```
1  pok ← START      /*postavljanje pokazivača na početak liste*/
2  START ← link(pok) /*izmena početka liste*/
3  E ← info(pok)    /*čitanje elementa E*/
4  freenode(pok)    /*oslobađanje čvora, tj. vraćanje čvora u LRMP*/
5  exit             /*kraj algoritma*/
```



# BRISANJE ČVORA SA ZADATE LOKACIJE - ALGORITAM

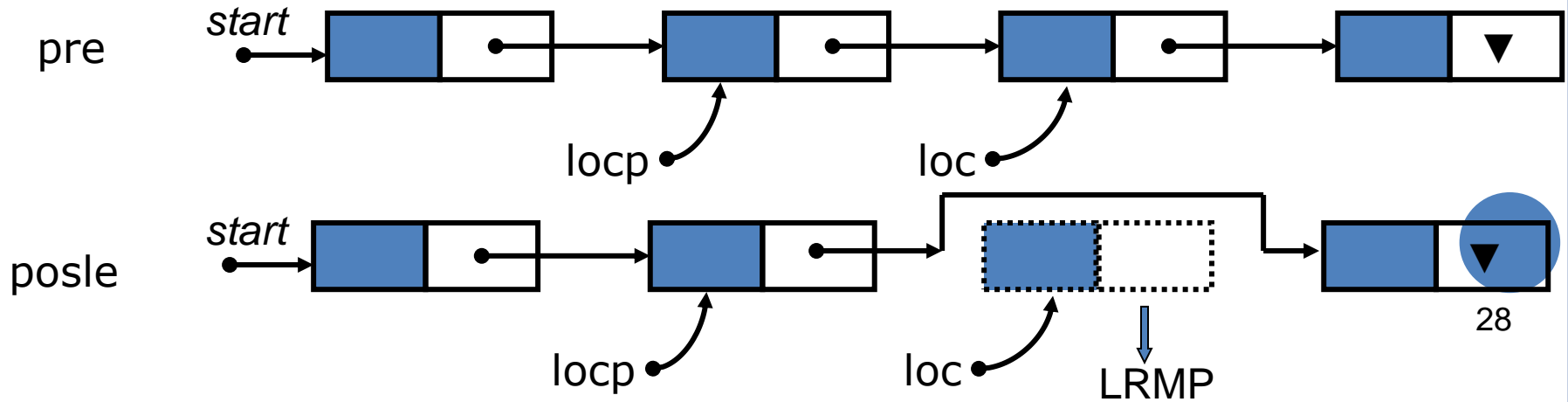
**Algoritam SLL.9.** Brisanje čvora sa lokacije LOC

**deleteFromLOC(START,LOC,LOCP)**

*// Ovaj algoritam briše čvor sa lokacije LOC lančane liste START*

*// LOCP je lokacija čvora koji prethodi čvoru LOC koji se briše*

```
1  if (LOCP=null)
2  then START ← link(START)           /*brisanje prvog čvora*/
3  else link(LOCP) ← link(LOC)        /*brisanje čvora X*/
4  freenode (LOC)                     /*oslobađanje čvora, tj. vraćanje čvora u LRMP*/
5  return                             /*kraj algoritma*/
```



# BRISANJE ČVORA ZADATE VREDNOSTI

**Algoritam SLL.10.** Brisanje čvora koji sadrži E

**deleteE(START,E, STATUS)**

*// Ovaj algoritam briše iz lančane liste prvi čvor koji sadrži E*

*// STATUS vraća informaciju o tome da li je brisanje uspešno izvedeno*

1 **call** findA(START,E,LOC,LOCP) */\* nalaženje lokacije LOC čvora koji sadrži E i lokacije njegovog prethodnika LOCP\*/*

2 **if** (LOC=null) **then**

3     status ← 1

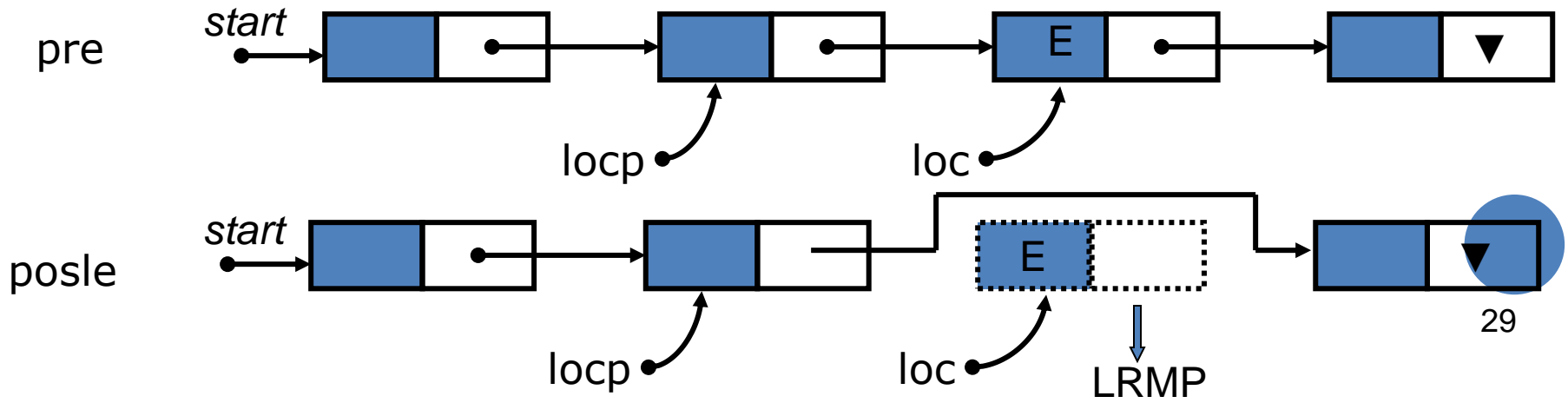
4     **exit**

*/\*brisanje neuspešno, E nije u listi\*/*

5 **call** deleteFromLOC(START,LOC,LOCP) */\*brisanje čvora\*/*

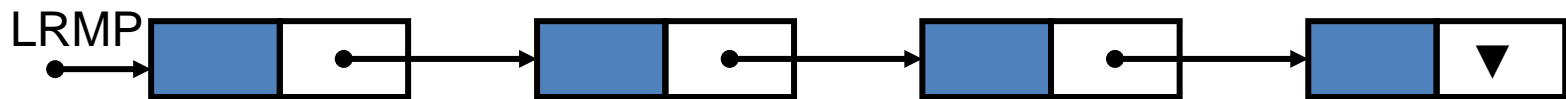
6 status ← 0 */\*brisanje uspešno, E je u listi\*/*

5 **exit** */\*kraj algoritma\*/*



# OPERACIJE GETNODE I FREENODE (1)

- Za memorisanje lančane liste se rezerviše u memoriji određeni broj čvorova koji čine tzv. *listu raspoloživog memorijskog prostora* čiji je eksterni pokazivač **LRMP**
- Operacija **getnode** uzima jedan čvor sa početka ove liste
- Operacija **freenode** vraća jedan čvor u ovu listu
- Pošto je ova lista konačna može se desiti da je prazna i tada operacija **getnode** treba da registruje da je došlo do *prekoračenja* memorijskog prostora



Prazna lista LRMP=null



# LANČANA LISTA KAO POLJE

- LL se može implementirati kao **polje čvorova**
- Čvorovi liste nisu uređeni kao **Lista2** → elementi polja, tako da svakom elementu polja, gde je memorisan čvor liste, treba **dodati pokazivač** na sledeći čvor
- Inicijalno se svi rezervisani elementi polja stavljaju u LRMP
- Može se koristiti jedno 2D polje LISTA, gde je LISTA[i,1] *info* polje, a LISTA[i,2] *link* polje ili se mogu koristiti dva 1D polja INFO i LINK

	info	link
1	25	0
2		7
3	20	1
4	10	5
5	15	13
6		2
7		8
8		9
9		10
10		12
11	5	4
12		16
13	20	14
14	20	15
15	20	19
16		17
17		18
18		20
19	25	0
20		0
	20	-1

**link=0**  
kraj  
liste,  
ako su  
indeksi  
1,2,...

**link=-1**  
kraj  
liste,  
ako su  
indeksi  
polja  
0,1,...



# OGRANIČENJA IMPLEMENTACIJE LISTE KAO POLJA

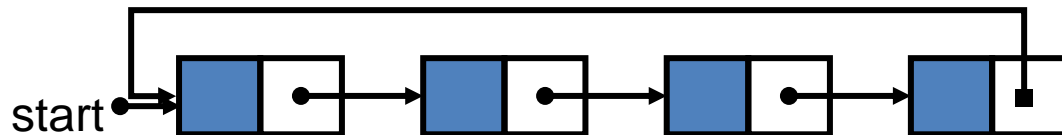
- Na početku se uspostavlja fiksni skup čvorova
- Pokazivač na čvor je predstavljen relativnom pozicijom čvora unutar polja
- Otuda proističu sledeća dva nedostatka:
  - Broj elemenata polja se obično teško može predvideti kada se program piše. Obično je broj čvorova određen podacima koje program obrađuje dok se izvršava.
  - Ma koliko čvorova mi deklarirali, određeni broj će ostati prazan u toku izvršenja programa
    - Na primer, ako smo rezervisali 100 čvorova, a trenutno obrađujemo samo 10, 90 će biti rezervisano za taj program i ta se memorija ne može dodeliti drugom programu
- Rešenje ovog problema je umesto ovih *statičkih* čvorova koristiti *dinamičke* čvorove, što znači da se memorija rezerviše kada je potreban novi čvor, a kada čvor više nije potreban, memorija se oslobađa. Takođe, nema predefinisano ograničenja u broju čvorova.





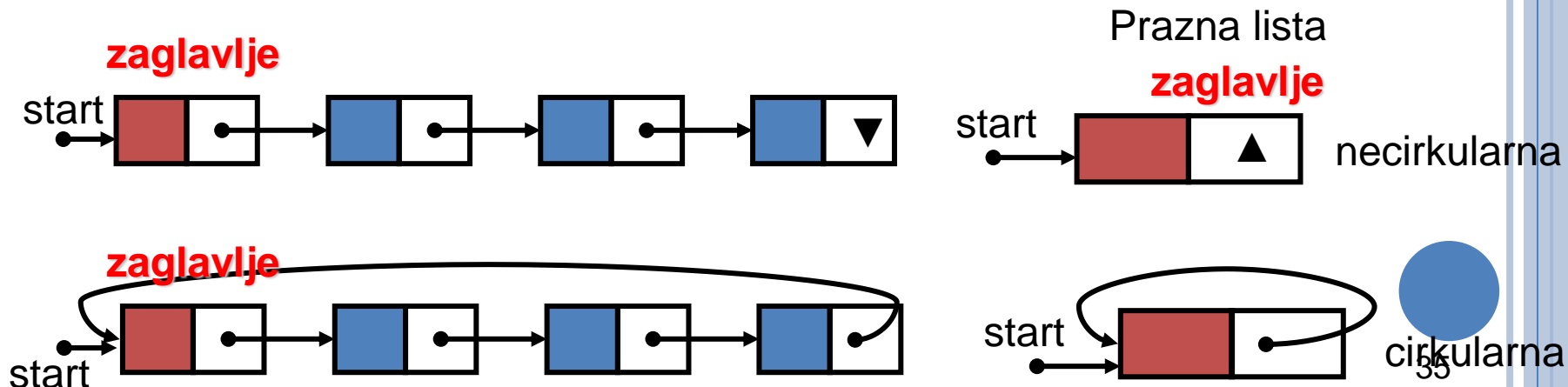
# CIRKULARNE LANČANE LISTE

- Poslednji čvor liste ukazuje na početak liste
- Iz bilo kog čvora se može doći do bilo kog čvora
- Nema potrebe da cirkularna lista ima prvi i poslednji čvor
- Magacin i red se često implementiraju kao cirkularna lista
- Operacije:
  - Obilazak
  - Traženje lokaciono ili asocijativno
  - Umetanje čvora pre ili posle zadatog čvora
  - Brisanje čvora pre ili posle zadatog čvora
  - ...



# LANČANE LISTE SA ZAGLAVLJEM

- LL sadrži specijalan čvor nazvan *zaglavlje* i to kao prvi čvor
- Mogu biti: necirkularne i cirkularne
- *Zaglavlje* je specijalni čvor koji se drži na početku liste i nije element liste
- *info* polje zaglavlja može biti prazno, ali češće sadrži globalne informacije o listi (kao što su broj čvorova liste, datum kreiranja liste, itd.) i/ili pokazivač na kraj liste i/ili pokazivač na tekući čvor pri obilasku liste, itd.



# CIRKULARNE LANČANE LISTE

## LANČANE LISTE SA ZAGLAVLJEM

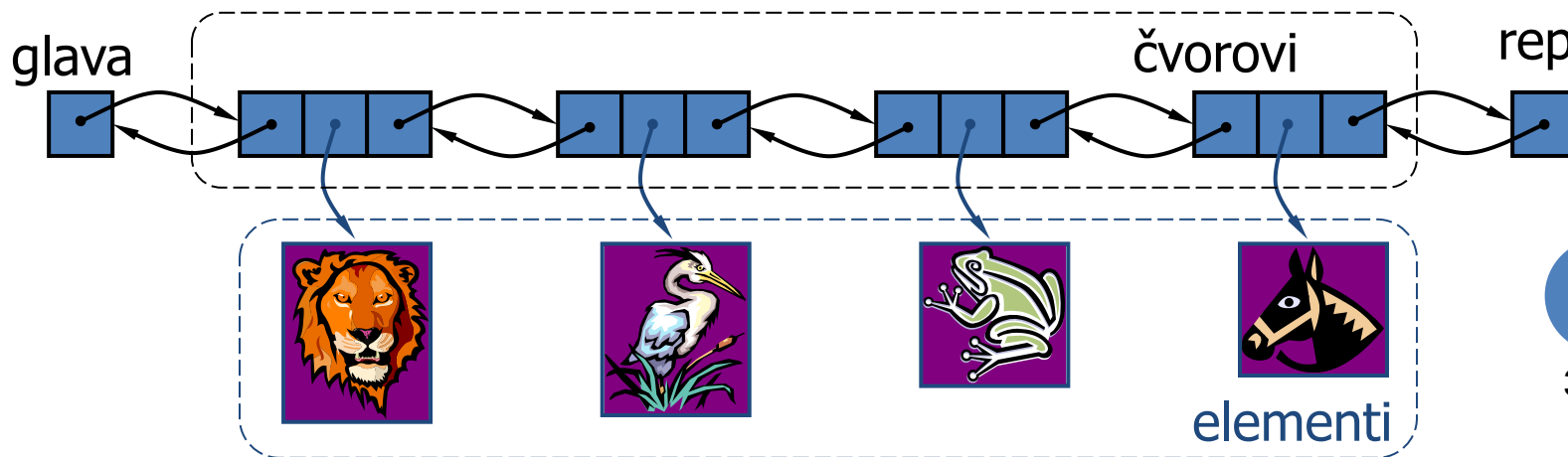
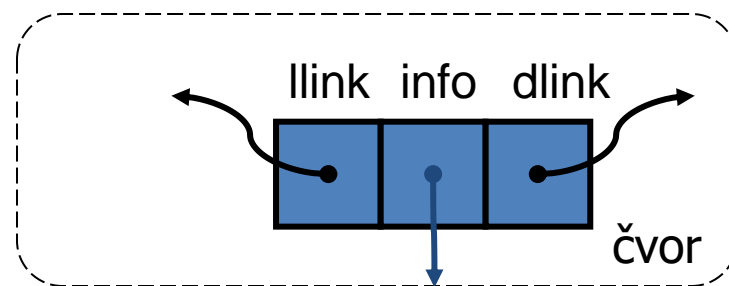
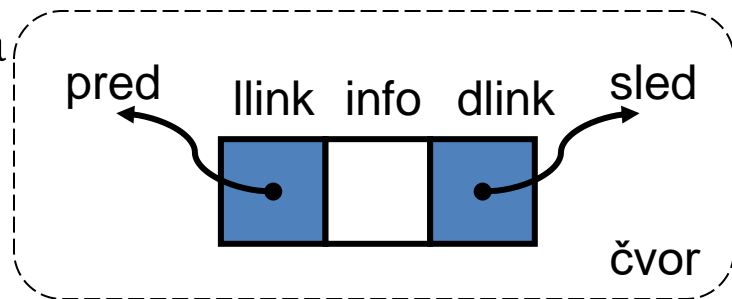
- Operacije za samostalni rad



# DVOSTRUKO SPREGNUTE LANČANE LISTE

## (DOUBLY LINKED LIST)

- Liste kojima se možemo kretati u dva smera – od prvog ka poslednjem čvoru ili od poslednjeg ka prvom čvoru
- Svaki čvor liste sadrži tri polja:
  - **info** – pamti element liste ili adresu elementa liste
  - **dlink** – pamti adresu sledećeg čvora
  - **llink** – pamti adresu prethodnog čvora
- Postoje dva specijalna čvora
  - **glava** i **rep** liste



# DVOSTRUKO SPREGNUTE LANČANE LISTE

- Mogu biti:
  - Necirkularne
    - Bez zaglavlja
    - Sa zaglavljem
  - Cirkularne
    - Bez zaglavlja
    - Sa zaglavljem
- Mogu se implementirati:
  - Statički
  - Dinamički



# OPERACIJE

## ○ Obilazak

- Koristi se **isti algoritam** kao kod jednosmernih listi
- Lista se može obilaziti od **glave** ili od **repa**
- **Dvostruko ulančavanje nema prednosti**

## ○ Traženje

- Koristi se **isti algoritam** kao kod jednosmernih listi
- Lista se može pretraživati od **glave** ili od **repa**, zavisno od toga gde se očekuje traženi podatak
- **Dvostruko ulančavanje nema prednosti**

## ○ Generičke:

- `size()`, `isEmpty()`

## ○ Upiti:

- `isFirst(p)`, `isLast(p)`

## ○ Pristup:

- `first()`, `last()`
- `before(p)`, `after(p)`

## ○ Ažuriranje:

- `replaceElement(p,e)`
- `swapElements(p,q)`
- `insertBefore(p,e)`
- `insertAfter(p,e)`
- `insertFirst(e)`
- `insertLast(e)`
- `remove(p)`



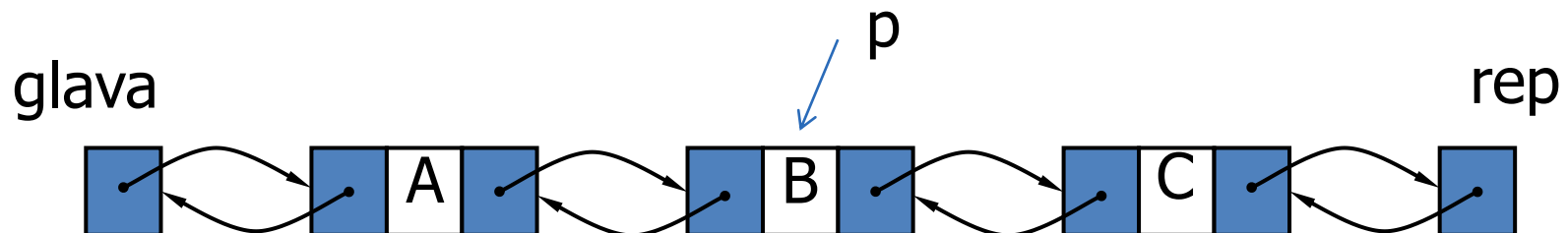
# OPERACIJE

## ○ Umetanje

- Dvostruko ulančavanje **može da** ima prednosti,
- nije potrebna pozicija prethodnog čvora

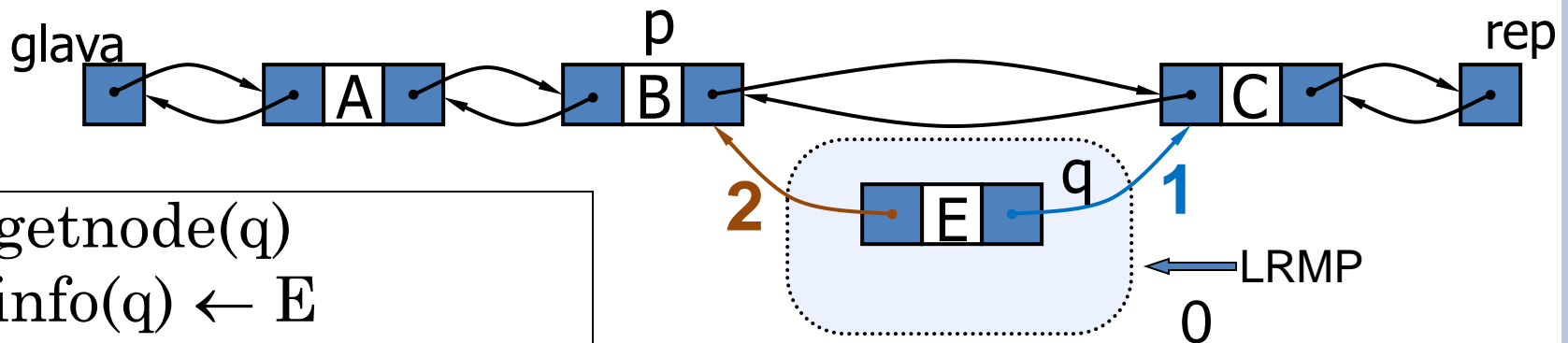
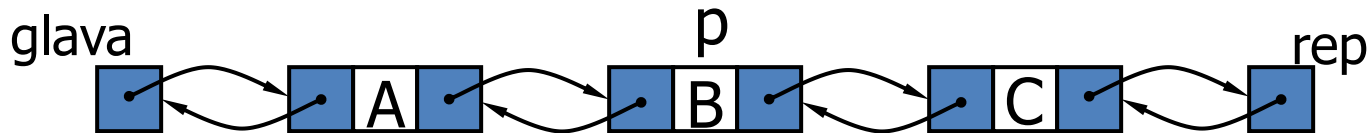
## ○ Brisanje

- Dvostruko ulančavanje ima prednosti,
- nije potrebna pozicija prethodnog čvora

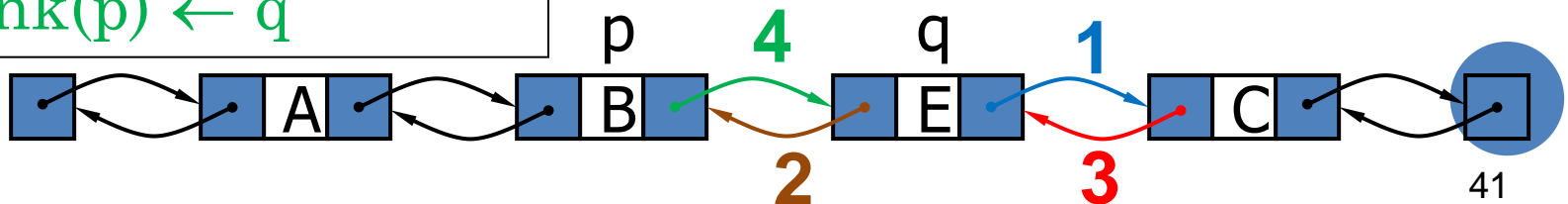


# UMETANJE (INSERTION)

Operacija *insertAfter*( $p, E$ ), koja vraća poziciju  $q$



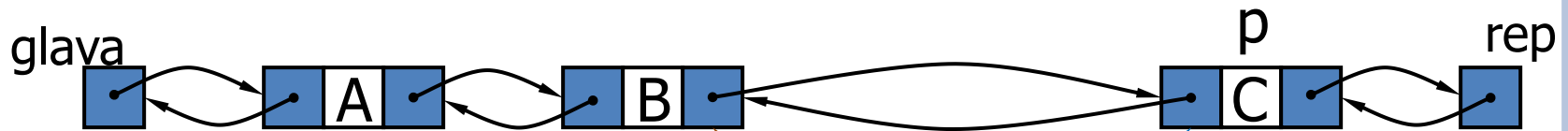
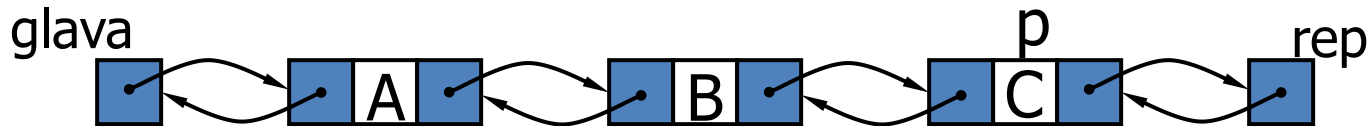
0.  $\text{getnode}(q)$   
 $\text{info}(q) \leftarrow E$
1.  $\text{dlink}(q) \leftarrow \text{dlink}(p)$
2.  $\text{llink}(q) \leftarrow p$
3.  $\text{llink}(\text{dlink}(p)) \leftarrow q$
4.  $\text{dlink}(p) \leftarrow q$



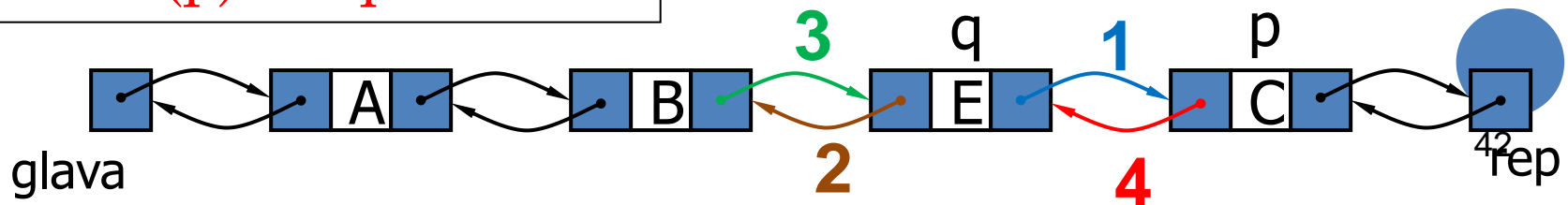


## UMETANJE (INSERTION) (2)

Operacija *insertBefore*( $p, E$ ), koja vraća poziciju  $q$

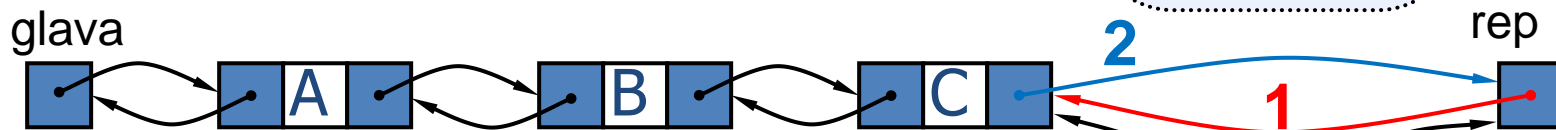
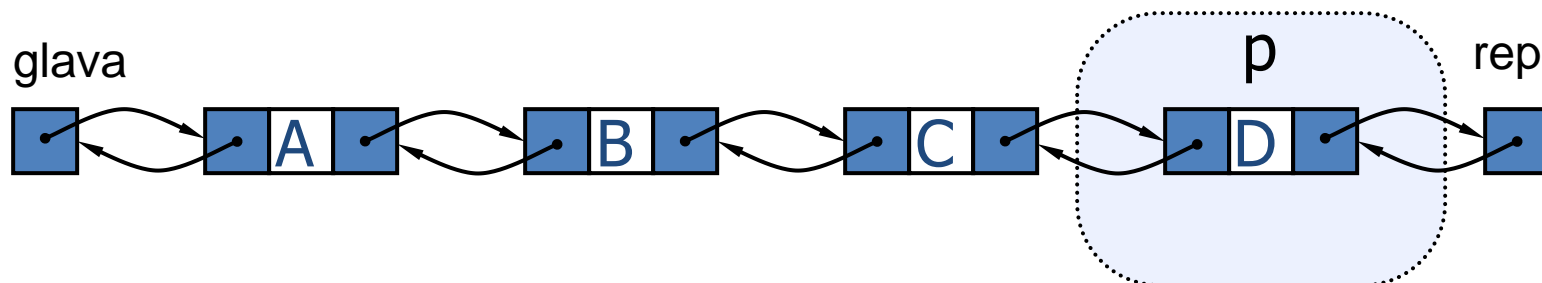


0. `getnode(q)`  
    `info(q) ← E`
1. `dlink(q) ← p`
2. `llink(q) ← llink(p)`
3. `dlink(llink(p)) ← q`
4. `llink(p) ← q`



# BRISANJE (DELETION)

Operacija *remove(p)*, gde je  $p = \text{last}()$

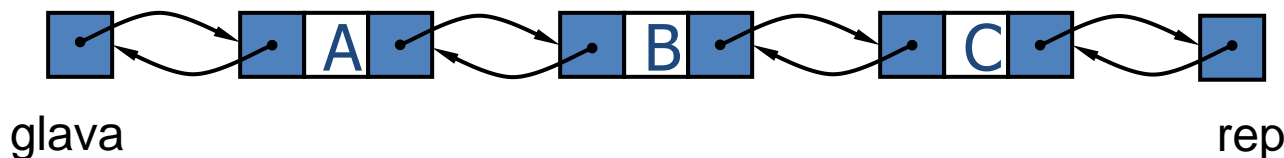
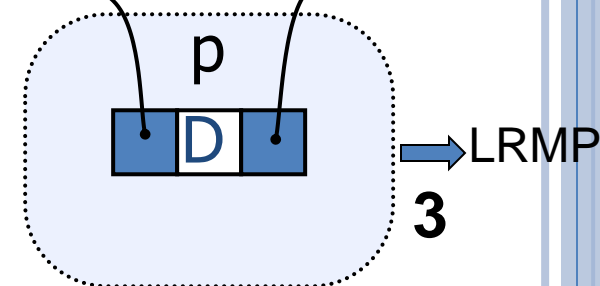


0.  $E \leftarrow \text{info}(p)$

1.  $\text{rep} \leftarrow \text{llink}(p)$

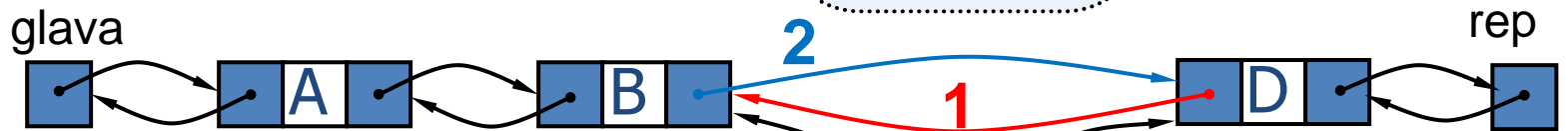
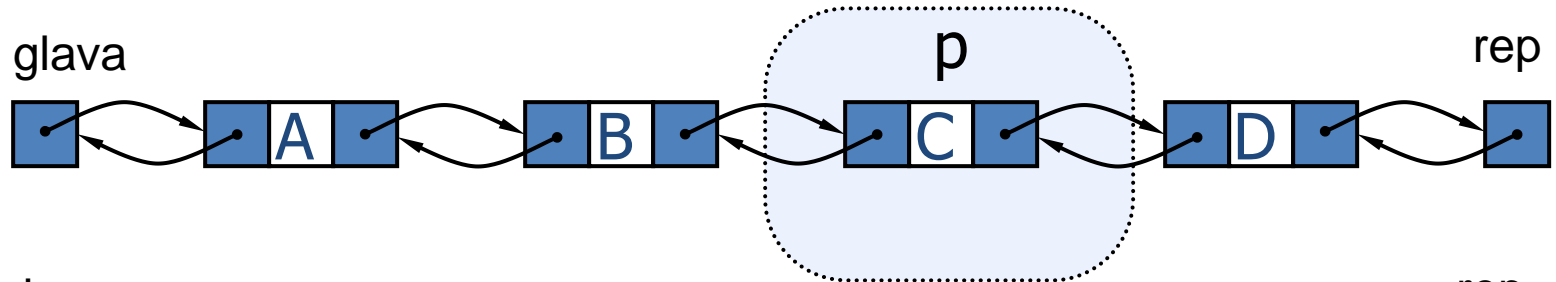
2.  $\text{dlink}(\text{llink}(p)) \leftarrow \text{dlink}(p)$

3.  $\text{freenode}(p)$



# BRISANJE (DELETION) (2)

Operacija *remove(p)*

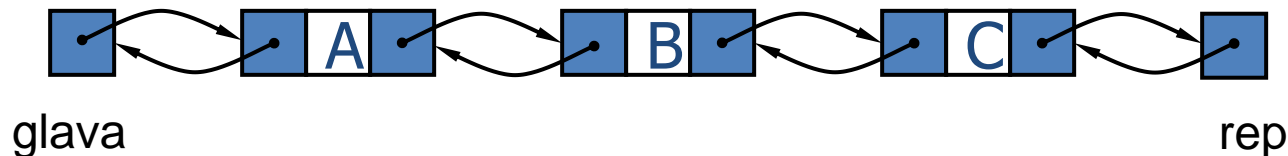
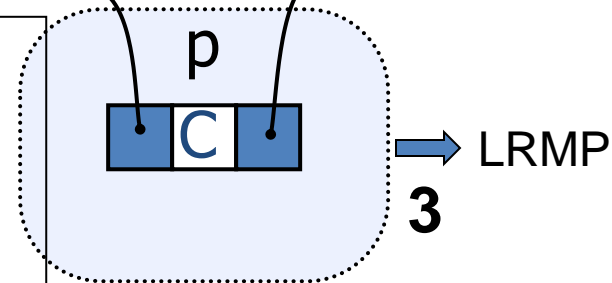


0.  $E \leftarrow \text{info}(p)$

1.  $\text{llink}(\text{dlink}(p)) \leftarrow \text{llink}(p)$

2.  $\text{dlink}(\text{llink}(p)) \leftarrow \text{dlink}(p)$

3.  $\text{freenode}(p)$



# DVOSTRUKO SPREGNUTE LANČANE LISTE

- Napisati pseudokod na osnovu zadatog redosleda koraka za operacije:
  - Umetanje
  - Brisanje
- Pseudokod za ostale operacije:
  - za samostalni rad!
- Vodite računa o graničnim slučajevima!!
  - Prazna lista,
  - Početak liste
  - Poslednji element (kraj liste)



# PITANJA, IDEJE, KOMENTARI

