

STRUKTURE PODATAKA

MAGACIN, RED, DEK (STACK, QUEUE, DEQUE)

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

PREGLED PREDAVANJA

○ **Magacin**

- Pregled, definicija
- Memorijska reprezentacija
- Operacije

○ **Red**

- Pregled, definicija
- Memorijska reprezentacija
- Operacije

○ Specijalni tipovi reda

○ **Dek**

- Ciklični red
- Red sa prioritetom



UVOD – MAGACIN, RED

- Jednostavne strukture podataka
- Izuzetno važne, često se koriste, imaju široku primenu
- Često implementirane i u samom hw i CPU
- C++ STL ih sadrži kao osnovne tipove
- Moderni programski jezici imaju ugrađeni podršku za rad sa ovim strukturama podataka

PRIMERI PRIMENE


○ Magacin

- Rekurzija
- Web browseri: čuva adrese posećenih sajtova u magacinu.
- Tekst editori: mogućnost da se promene čuvaju u magacinu – operacije *undo* i *redo*

○ Red

- Printer – red čekanja kod štampe
- Klijentski računari šalju zahteve Web serveru
- Air traffic control
- EFT (electronic funds transfer) transactions requiring handling
- Simulacija bilo koje *real-life* situacije sa redovima

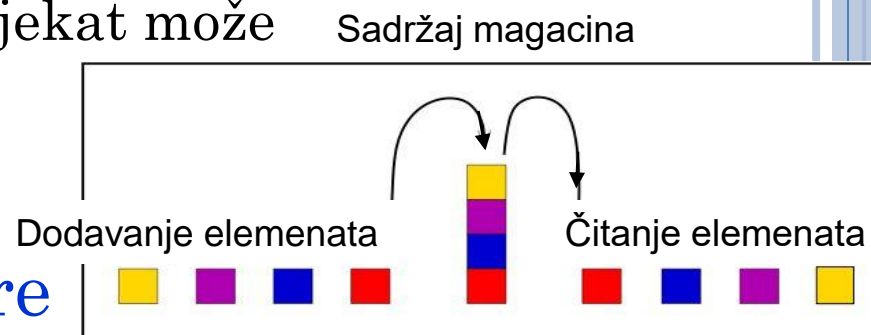
MAGACIN (STACK)

- Naziv “**magacin**”: 
- Linearna struktura podataka kod koje se elementi dodaju ili brišu na jednom kraju, koji se zove **vrh magacina**
- Predstavlja kontejner objekata koji se dodaju i brišu po principu

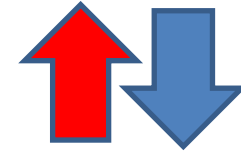


LIFO (last-in-first-out):

- Objekti se mogu dodavati bilo kad (redom), ali se samo poslednji dodati objekat može pročitati
- Dodavanje i brisanje se vrše **na istom kraju strukture**



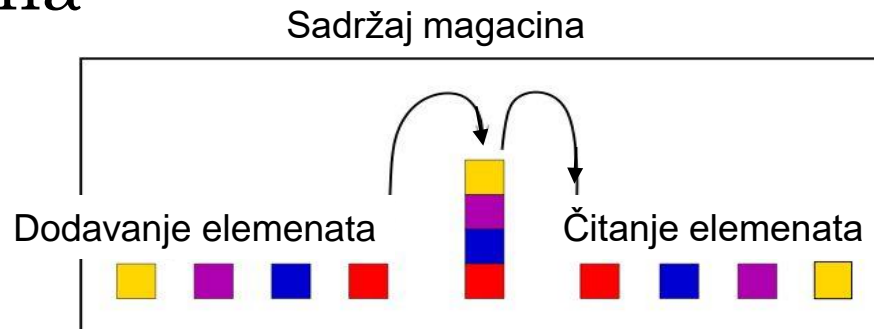
MAGACIN (STACK)



◦ **LIFO** (last-in-first-out):

◦ Dve osnovne operacije

- **Push** – dodavanje elementa na vrh magacina
- **Pop** – **brisanje** elementa sa vrha magacina

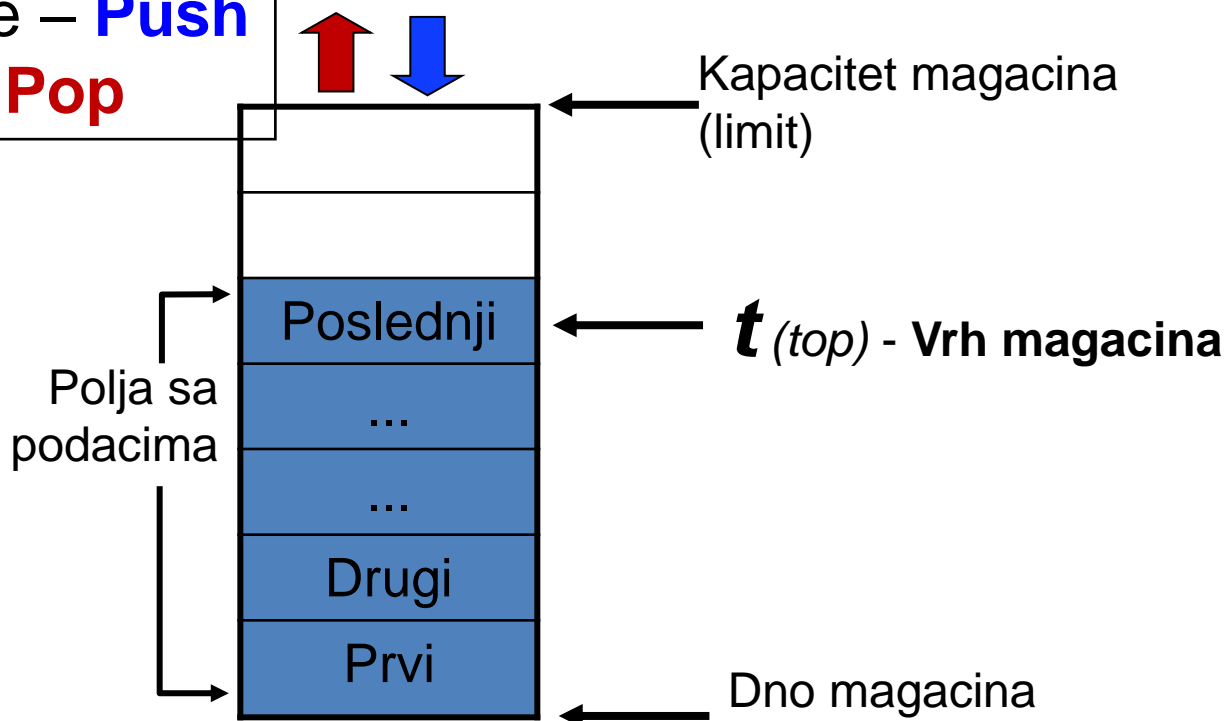


MAGACIN: OSNOVNI POJMOVI

o **Kako korisnik vidi magacin** bez obzira na implementaciju:

Operacije:

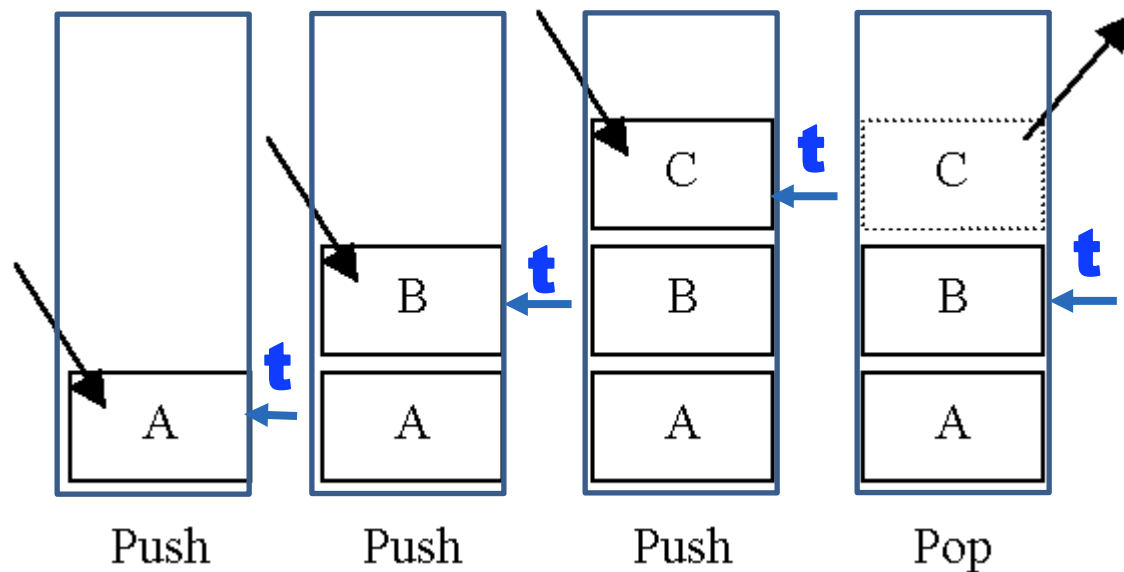
- Dodavanje – **Push**
- Brisanje – **Pop**



PRIMER RADA SA MAGACINOM

- Dodavanje redom elemenata A, B i C
- Čitanje iz magacina

LIFO



MAGACIN: PRIMER

- Prikazati stanja magacina ako su date sledeće informacije:
 - Smestiti u magacin sledeće brojeve u navedenom redosledu: 5, 8, 10, 2, 4, 12
 - Pročitati/obrisati dva elementa iz magacina
 - Izbaciti broj 10 iz magacina.
 - Smestiti 3, 7, 1 u magacin.
 - Na kraju, obrisati 5 sa dna magacina i smestiti ga na vrh magacina.



OPERACIJE ZA RAD SA MAGACINOM

- Osnovne operacije: “push” i “pop”.

- ***Push(s,t,N,e)***: dodaje novi element *e* u magacin
 - *Ulaz: magacin, vrh, max.br.el.*, Objekat koji se dodaje *e*;
- ***Pop(s,t,e)***: uklanja element iz magacina
 - *Ulaz: magacin, vrh Izlaz: Objekat sa vrha magacina*

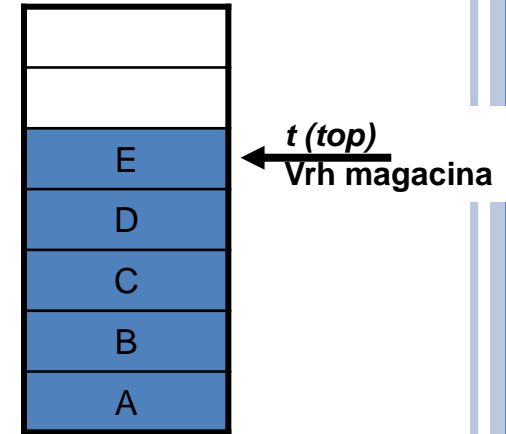
- Ostale operacije

- ***GetSize(s,t)***: vraća broj elemenata u magacinu
 - *Ulaz: magacin, vrh Izlaz: integer*
- ***isEmpty(s,t)***: proverava da li je magacin prazan
 - *Ulaz: magacin, vrh Izlaz: boolean*
- ***IsFull(s,t,N)***: proverava da li je magacin pun
 - *Ulaz: magacin, vrh, max.br.el. Izlaz: boolean*
- ***Top(s,t,e)*** – vraća vršni element magacina, bez brisanja
 - *Ulaz: magacin, vrh. Izlaz: Objekat sa vrha magacina*

IMPLEMENTACIJA MAGACINA

- Rezervisan **memorijski prostor**:

- Polja (statička implementacija)
- Lančane liste (dinamička impl.)



- Promenljiva koja definiše **vrh** (*top*) magacina
- Implementacija **operacija** za rad sa magacinom

- Magacin čine:

- izabrana struktura podataka (polje ili lančana lista),
- vrh magacina i
- implementirane operacije

MEMORIJSKA REPREZENTACIJA: MAGACIN KAO POLJE

- Prirodan način implementacije magacina je preko polja objekata (bilo kog tipa)
- Za magacin se rezerviše **polje S** maksimalne **veličine N** , napr., $N = 1000$.
- Potrebna je celobrojna (integer) **promenljiva t** koja definiše **vrh magacina**
- Vrh magacina t predstavlja **indeks poslednjeg elementa** u polju S , odnosno vršnog elementa u magacinu.
- Pretpostavka: indeks prvog elementa u polju je 0 . Zbog toga vršimo inicijalizaciju $t = -1$.
- **Magacin zajedno čine polje S veličine N i promenljiva t kao vrh magacina**



MOGUĆE VREDNOSTI ZA VRH MAGACINA T

- $t = 0$, ili $t = -1$, magacin je **prazan**
- $t = N$, ili $t = N-1$, magacin je **pun**
- Postoje elementi u magacinu:
 $1 < t \leq N$ (odnosno $0 \leq t \leq N-1$)

Promena vrednosti:

Upis: $t + 1$

Brisanje: $t - 1$

Lokacija na koju ukazuje t je **lokacija vršnog elementa**



PSEUDO KOD OPERACIJA (1)

Algoritam M.1:

Određivanje veličine magacina

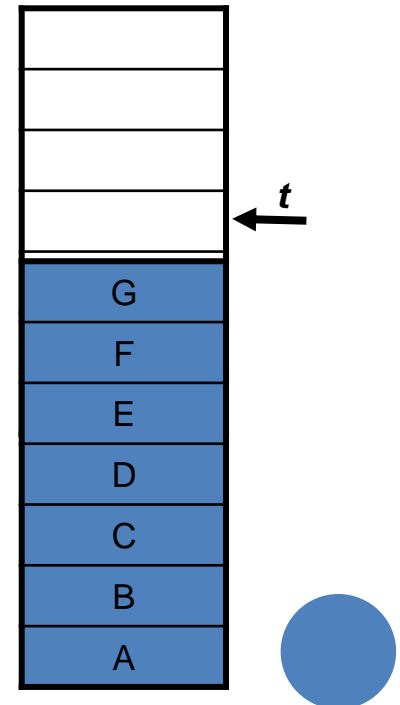
Size(s,t)

// magacin **s**, vrh magacina **t**

// **t = -1** kada je magacin prazan

1. **return** $t + 1$

Složenost $O(1)$

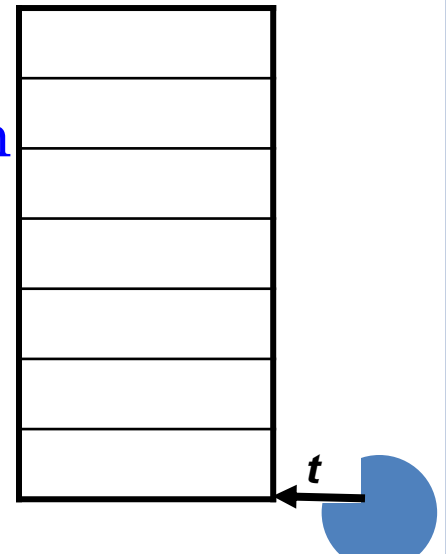


PSEUDO KOD OPERACIJA (2)

Algoritam M.2: *Uslov potkoračenja*
(Proverava da li je magacin prazan)

isEmpty(s,t)

1. **if** ($t < 0$) **then**
2. **return** true // magacin je prazan
3. **else**
4. **return** false // magacin nije prazan



Složenost **$O(1)$**

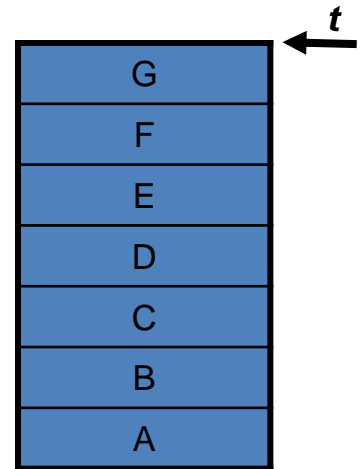
PSEUDO KOD OPERACIJA (3)

Algoritam M.3: *Uslov prekoračenja*
Provera da li je magacin pun

IsFull(s,t,N)

// magacin s , vrh magacina t ,
// veličina magacina N

1. **if** ($t = N - 1$) **then**
 // uslov moze biti ($\text{size}(s,t) = N$)
2. **return** true // magacin je pun
3. **else**
4. **return** false // magacin nije pun



Složenost $O(1)$

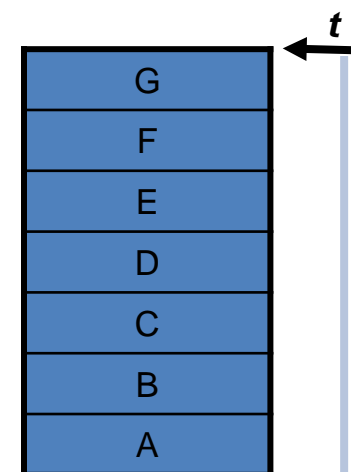


PSEUDO KOD OPERACIJA (4)

Algoritam M.4: Vršni element magacina,
bez brisanja

top(s,t,e)

1. **if** isEmpty(s,t) **then**
2. *Potkoračenje*
3. **else**
4. $e = s[t]$
 // element na koji ukazuje vrh magacina
5. **return**



Složenost $O(1)$

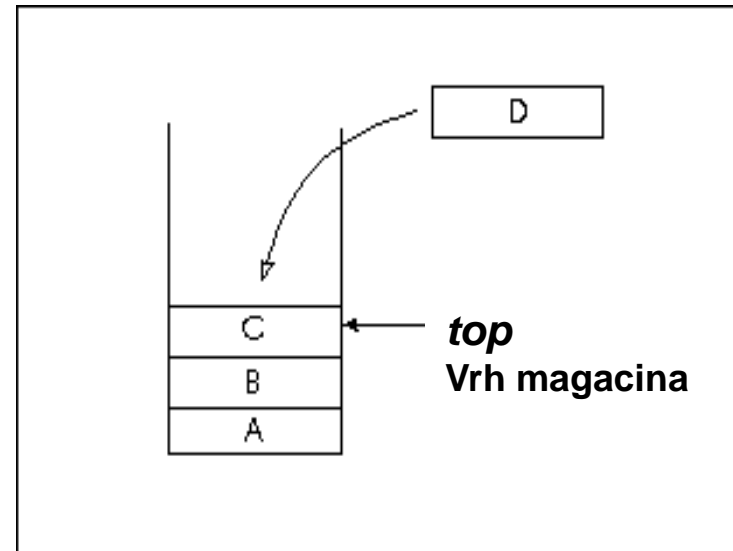
PSEUDO KOD OPERACIJA (5)

Algoritam M.5: Upis elementa u magacin

push(s,t,N,e)

// magacin s , vrh mag. t , vel.mag. N , element koji se
// upisuje e

1. **if** IsFull(s,t) **then**
2. **return** -1 // prekoračenje
3. **else**
4. $t \leftarrow t + 1$
5. $S[t] \leftarrow e$
6. **endif**

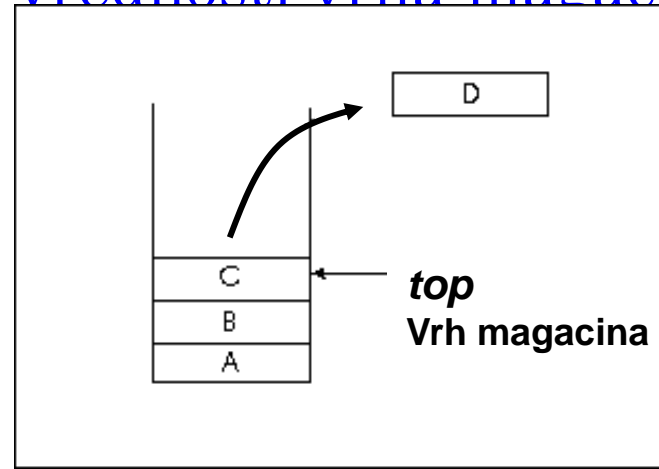


PSEUDO KOD OPERACIJA (6)

Algoritam M.6: Brisanje/Čitanje elementa iz magacina

pop(s,t,e)

1. **if** isEmpty(s,t) **then** // uslov potkoračenja
2. *Potkoračenje*
3. **else**
4. $e \leftarrow S[t]$
5. $S[t] \leftarrow \text{null}$ // brisanje elementa
6. $t \leftarrow t - 1$ // promena vrednosti vrha magacina
7. **return**



KARAKTERISTIKE MEMORIJSKE REPREZENTACIJE PREKO POLJA

- Implementacija preko polja je vrlo **jednostavna**
- Vrlo **efikasna**
- Problem:
 - Nije adaptabilna - gornja granica broja elemenata je određena veličinom polja N
 - taj broj može biti isuviše mali za neku aplikaciju, odnosno
 - Može biti isuviše veliki i nepotrebno zauzimati memoriju.



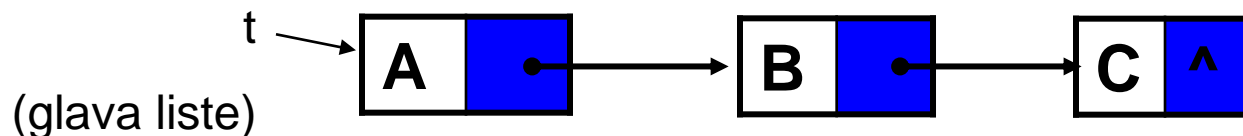
MEMORIJSKA REPREZENTACIJA: MAGACIN KAO LANČANA LISTA

Osnovna ideja

- Implementacija: korišćenje osnovnih operacija za rad sa **jednostruko spregnutom lančanom listom**:
- Dodavanje elemenata na **početak** liste funkcioniše kao i dodavanje elemenata u magacin.
- **Brisanje elemenata** iz magacina odgovara **brisanju prvog elementa** lančane liste (elementa na koga ukazuje *start* odnosno *head/glava* liste)
- Magacin je **prazan** ako je **lista prazna**

Prednosti:

- Nema ograničenja u broju elemenata odnosno kapacitetu/veličini magacina



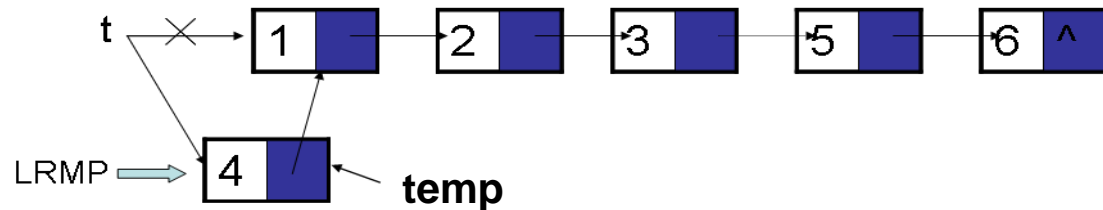
MAGACIN KAO L.LISTA – PUSH

**Algoritam M.7: Dodavanje elementa
magacinu (lančana lista) – odgovara
algoritmu SLL.4**

Push(s,t,e)

// magacin s , vrh magacina t , element koji se
// dodaje e

1. $\text{temp} \leftarrow \text{getnode}()$ // novi čvor iz LRMP
2. $\text{info}(\text{temp}) \leftarrow e$
3. $\text{link}(\text{temp}) \leftarrow t$
4. $t \leftarrow \text{temp}$

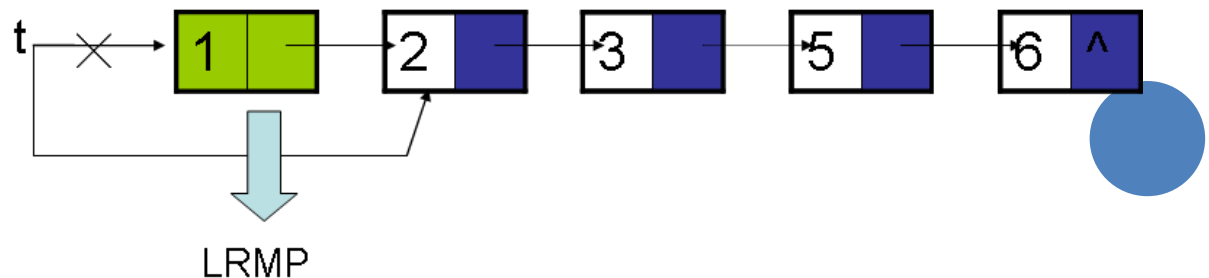


MAGACIN KAO LISTA – POP

Algoritam M.8: Brisanje elementa iz magacina (lančana lista) odgovara algoritmu SLL.8

Pop(s,t,e)

1. if ($t = \text{NULL}$) then *Potkoračenje*
2. else
3. $\text{temp} \leftarrow t$
4. $t \leftarrow \text{link}(t)$
5. $e \leftarrow \text{info}(t)$
6. $\text{freenode}(\text{temp})$ // vraćanje u LRMP
7. return



MAGACIN KAO LISTA: PSEUDOKOD ZA OSTALE OPERACIJE

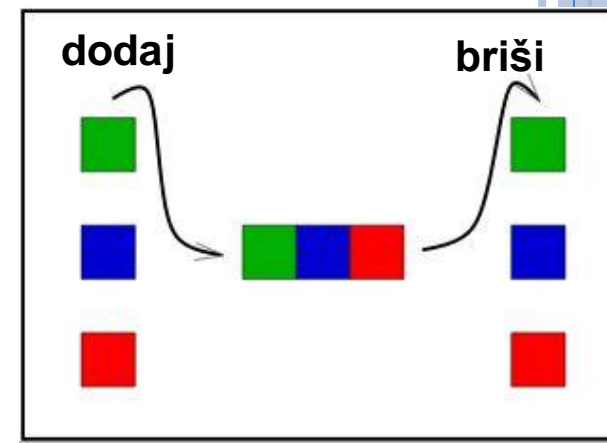
- *Size(s,t)* – svodi se na brojanje elemenata liste (obilazak liste) – [Algoritam SLL.1](#)
- *IsFull(s,t,N)* - nema ograničenja u veličini magacina (zavisi od raspoloživog mem. prostora)
- *IsEmpty(s,t)* – ispitivanje da li je lista prazna, odnosno da li je $t = \text{NULL}$



RED (QUEUE)

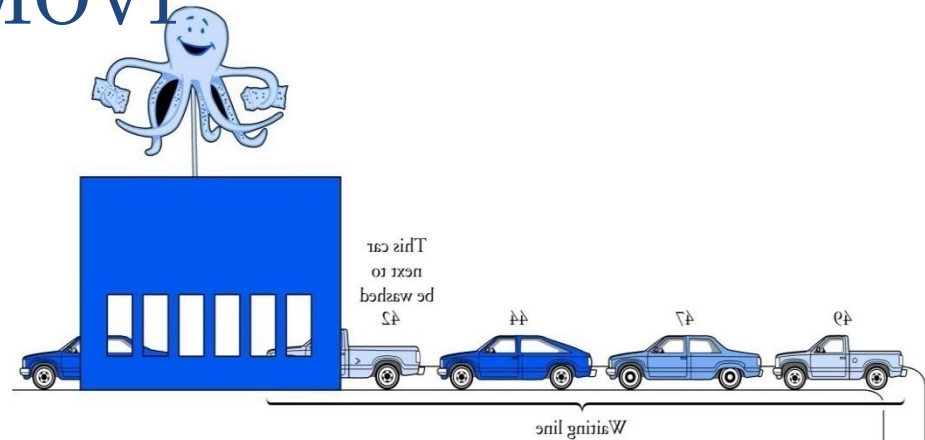
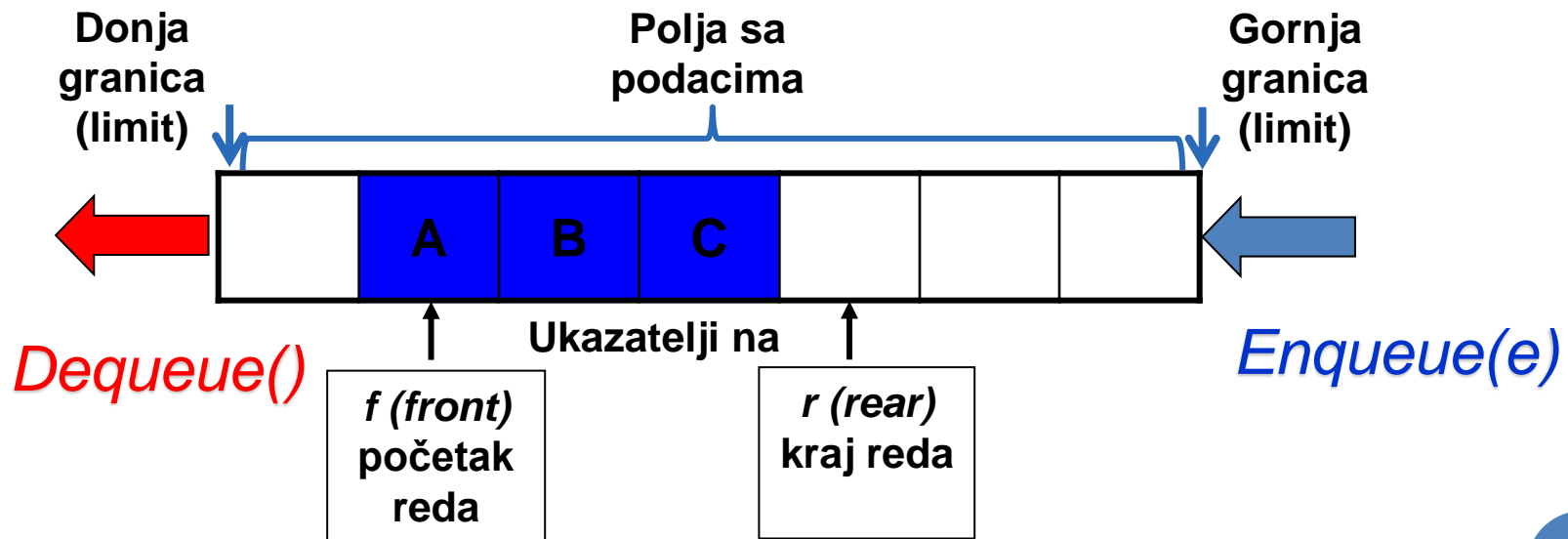


- Struktura podataka slična magacinu, razlika je u redosledu upisa i čitanja elemenata.
- Linearna lista kod koje se elementi **dodaju samo na jednom kraju** a **brišu na drugom kraju**
- Sadrži objekte koji se dodaju po principu **FIFO (First-In-First-Out)**, odnosno *prvi-ušao-prvi-izašao*.
 - Podrazumeva dodavanja elemenata na jednom kraju (**kraj** reda)
 - Čitanje elemenata na drugom kraju reda (**početak** reda)
- Elementi se mogu dodavati bilo kad, ali se samo najstariji čita iz reda

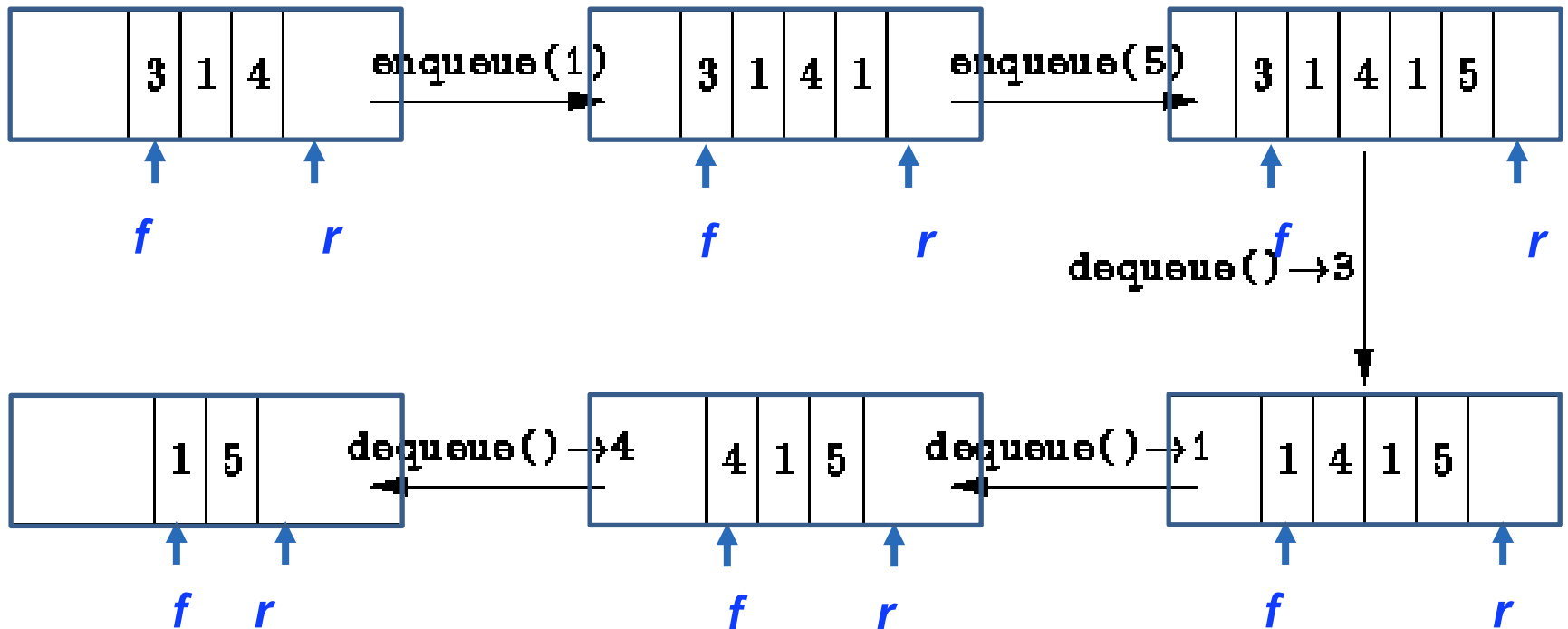


RED: OSNOVNI POJMOVI

Kako korisnik vidi
red bez obzira na
implementaciju:



PRIMER RADA SA REDOM



OPERACIJE ZA RAD SA REDOM

○ Osnovne operacije:

- *Enqueue*(Q, f, r, N, e): dodaje objekat na kraj reda
- *Dequeue* (Q, f, r, N, e): briše objekat sa početka reda i vraća taj objekat

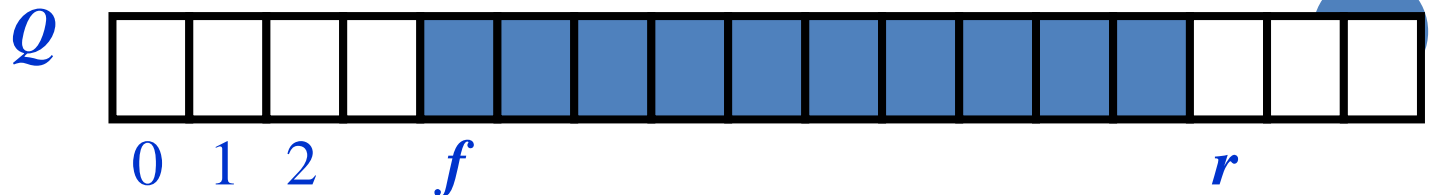
○ Ostale operacije:

- *isEmpty*(Q, f, r): proverava da li je red prazan
- *isFull*(Q, f, r, N): proverava da li je red pun
- *size*(Q, f, r, N): vraća broj objekata u redu
- *front*(Q, f, r, N, e): vraća prvi element iz reda (sa početka reda) ali ga ne briše iz reda.



IMPLEMENTACIJA REDA

- Rezervisan **memorijski prostor**:
 - Polja (statička implementacija)
 - Lančane liste (dinamička impl.)
- Dve promenljive koje definišu **početak** ($f - front$) i **kraj** ($r - rear$)
- Implementacija **operacija** za rad sa redom
- Red čine:
 - izabrana struktura podataka (polje ili lančana lista),
 - Promenljive za početak i kraj reda
 - implementirane operacije

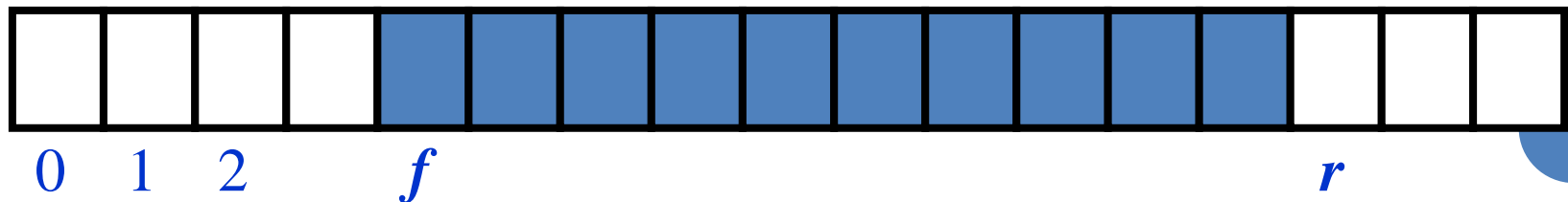


MEMORIJSKA REPREZENTACIJA

RED KAO POLJE

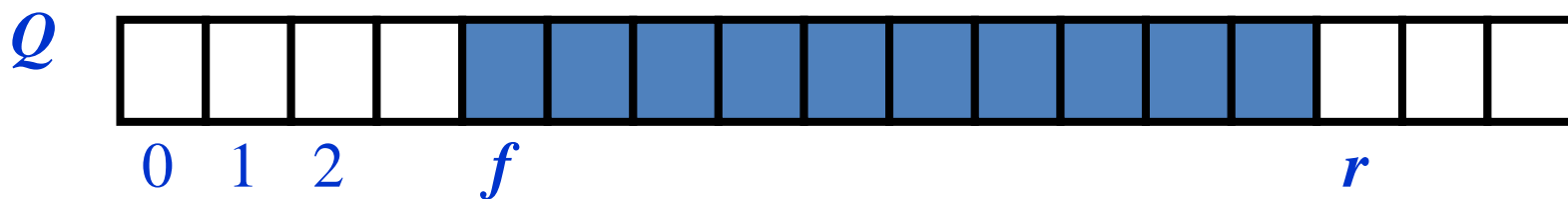
- Koristi se polje veličine N
- Dve promenljive koje se odnose na početak i kraj reda (ukazatelj na početak i ukazatelj na kraj reda)
 - f (front): indeks prvog elementa reda
 - r (rear): indeks elementa koji je odmah iza poslednjeg elementa u redu
- Lokacija na koju ukazuje r je prazna

Q

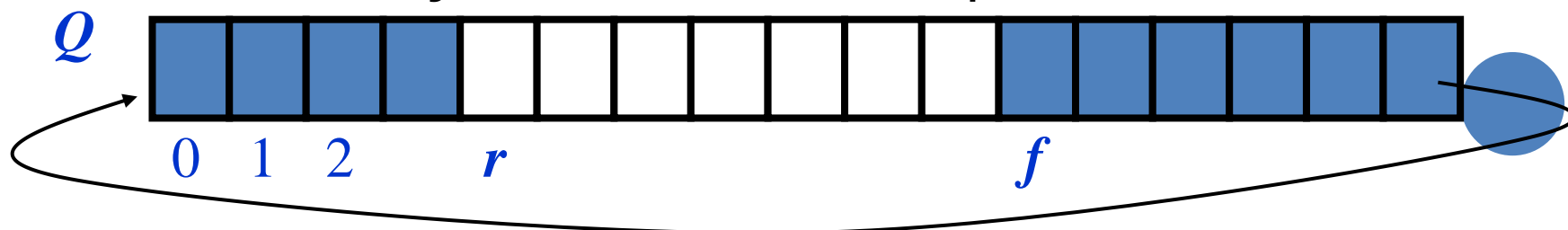


RED KORISTI CIRKULARNO POLJE

- Koristi se polje veličine N ali se tretira kao **cirkularno**
 - kada se kod upisa ili čitanja dođe do kraja polja, ako ima mesta/elementa, nastavlja se dodavanje/čitanje od donje granice
 - Izbegava se složenost operacija reda $O(n)$
 - Složenost svih operacija je $O(1)$



Stanje nakon cirkularne upotrebe



RED KAO CIRKULARNO POLJE (2)

- U ovom slučaju može biti $r < f$.
- $Q[f]$ = element na početku reda
- $Q[r-1]$ = element na kraju reda

Promene ukazatelja: “po modulu” $x \bmod y$

- Promena f kod **čitanja**:

$$f \leftarrow (f + 1) \bmod N$$

- Promena r kod **upisa**:

$$r \leftarrow (r + 1) \bmod N$$

Granični slučajevi

- Prazan red: $f = r$
- Pun red: $f = (r + 1) \bmod N$



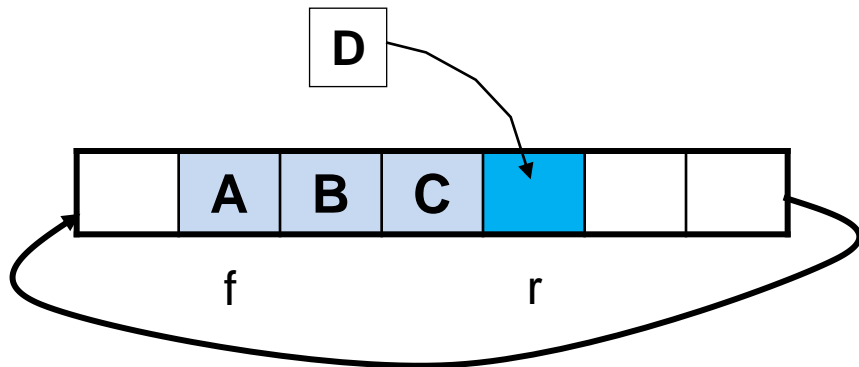
RED, OPERACIJE:

DODAVANJE

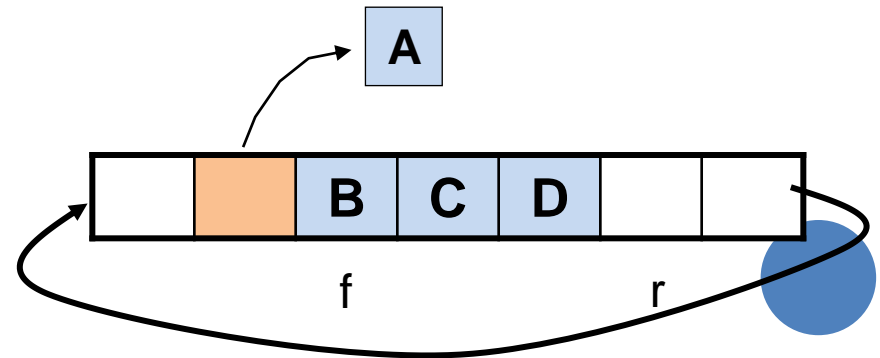
I

BRISANJE

- Provera prekoračenja
- Dodavanje elementa **D** redu
- Promena ukazatelja
 $r = (r + 1) \bmod N$



- Provera potkoračenja
- Čitanje elementa /
Brisanje iz reda
- Promena ukazatelja
 $f = (f + 1) \bmod N$



OPERACIJE ZA RAD SA REDOM (1)

Algoritam R.1: Broj elemenata reda

size(Q, f, r, N)

// Q – red, f – početak reda, r – kraj reda, N – veličina reda

1. **return** $(N - f + r) \bmod N$

Algoritam R.2: Provera potkoračenja, da li je red prazan

isEmpty(Q, f, r)

1. **if** $(f = r)$ **then**

2. **return** true // potkoračenje, red je prazan

3. **else**

4. **return** false



OPERACIJE ZA RAD SA REDOM (2)

Algoritam R.3: Provera prekoračenja, da li je
red pun

isFull(Q, f, r, N)

1. **if** ($f = (r + 1) \bmod N$) **then**
2. **return** true // prekoračenje, red je pun
3. **else**
4. **return** false // red nije pun



OPERACIJE ZA RAD SA REDOM (3)

Algoritam R.4: Upis elemanta u red

enqueue(Q, f, r, N, o)

1. **if** $(f = (r+1) \bmod N)$ **then**
2. *Prekoracenje*
3. **else**
4. $Q[r] \leftarrow o$
5. $r \leftarrow (r + 1) \bmod N$

Ili *isFull*(Q, f, r, N)



OPERACIJE ZA RAD SA REDOM (4)

Algoritam R.5: Prvi element reda, bez brisanja

front(Q, f, r, N, o)

1. **if** **isEmpty**(Q, f, r) **then**
2. *Potkoračenje*
3. **else**
4. $o = Q[f]$
5. **return**

isEmpty(Q, f, r)
if ($f = r$) **then**
 return true
else
 return false



OPERACIJE ZA RAD SA REDOM (5)

Algoritam R.6 Čitanje elementa iz reda

dequeue(Q, f, r, N, o)

1. **if** (**isEmpty**(Q, f, r)) **then**

2. *Potokračenje*

3. **else**

4. $o \leftarrow Q[f]$

5. $f \leftarrow (f + 1) \bmod N$

6. **return**

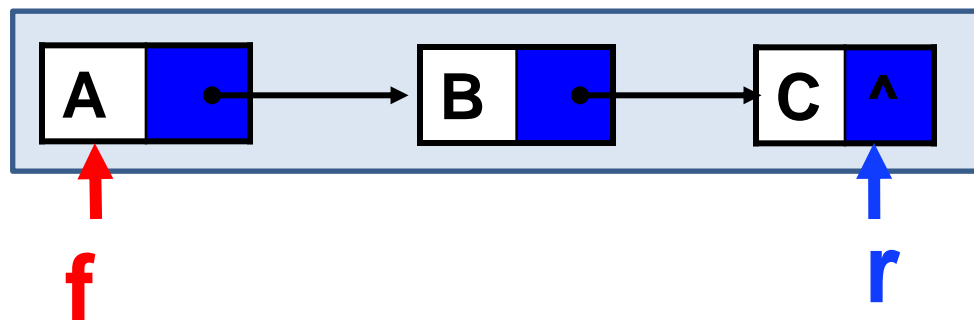
isEmpty(Q, f, r)
if ($f = r$) **then**
 return true
else
 return false



MEMORIJSKA REPREZENTACIJA

RED KAO LANČANA LISTA

- Jednostruko spregnuta lančana lista, uz dodatni ukazatelj
- Ukazatelj na početak reda je glava (početak) liste
- Ukazatelj na kraj reda je rep liste
- Zašto ne obrnuto?



RED KAO LANČANA LISTA

PSEUDOKOD OSNOVNIH OPERACIJA

Algoritam R.7: Brisanje elementa iz reda **(lančana lista)**

-- Odgovara operaciji *Pop* kod magacina, odnosno
Algoritmu SLL.8 za lančane liste

Dequeue(Q,f,r,N,o)

1. **if** (IsEmpty(Q,f,r)) **then**
2. *Potkoracenje*
3. **else**
4. $p \leftarrow f$
5. $o \leftarrow \text{info}(f)$
6. $f \leftarrow \text{link}(f)$
7. freenode(p) // LRMP $\leftarrow p$
8. **return**

Implementacija??

RED KAO LANČANA LISTA

PSEUDOKOD OSNOVNIH OPERACIJA (2)

Algoritam R.8: Dodavanje elementa redu (lančana lista)

Enqueue(Q,f,r,N,o)

1. $p \leftarrow \text{getnode}()$
2. $\text{info}(p) \leftarrow o$
3. $\text{link}(p) \leftarrow \text{NULL}$
4. $\text{link}(r) \leftarrow p$
5. $r \leftarrow p$

{ !!!

ovde treba dodati proveru
da li je red prazan – zašto?



PSEUDOKOD – OSTALE OPERACIJE

- *Size()* – broj elemenata liste, obilazak liste
- *IsEmpty()* – da li je lista prazna?
f=NULL
- *IsFull()* – nema ograničenja!
- Koji algoritmi iz dela “Lančane liste” odgovaraju ovim operacijama?



MOGUĆI TIPOVI REDOVA

- Običan red (FIFO)
- Ciklični red (običan red)
- **Dek**
- Redovi sa prioritetom



SPECIJALNI TIPOVI REDA:

CIKLIČNI RED

- Običan red
- Promena vrednosti ukazatelja na početak i kraj “u krug”
- Često se koristi naziv “ring buffer”
- Složenost operacija $O(1)$



DEK (DEQUEUE)

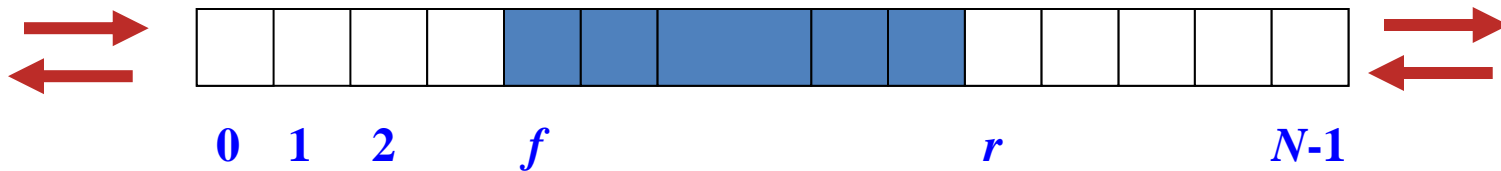
- **Dvostruki red** (double-ended queue), ili *dek* (*deque*),
- Podržava dodavanje i brisanje na oba kraja reda. **Parametri?**
- Dek ADT - operacije
 - **insertFirst(..., *e*)**: dodavanje elementa *e* na početak deka
Ulaz: Objekat; *Izlaz*: nema
 - **insertLast(..., *e*)**: Dodavanje *e* na kraj deka.
Ulaz: Objekat; *Izlaz*: nema
 - **removeFirst(...)**: briše i vraća prvi element deka.
Ulaz: nema; *Izlaz*: Objekat
 - **removeLast(...)**: briše i vraća poslednji element.
Ulaz: nema; *Izlaz*: Objekat
- Dodatne operacije
 - first(...)
 - last(...)
 - size(...)
 - isEmpty(...)
 - isFull(...)



MEMORIJSKA REPREZENTACIJA DEKA

Polje

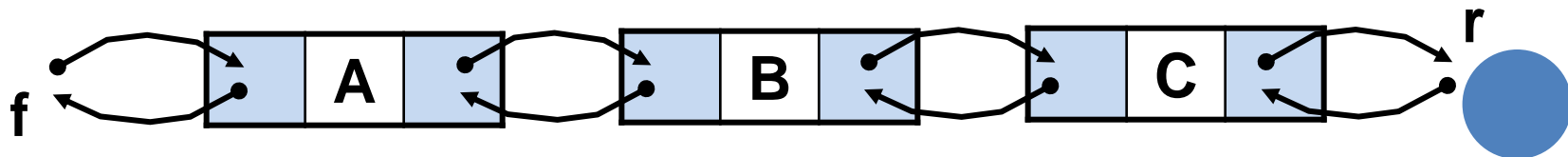
- Kao kod reda, s tim da upis i čitanje idu na oba kraja
- Implementacija, granični slučajevi??
- Promena f i r ??



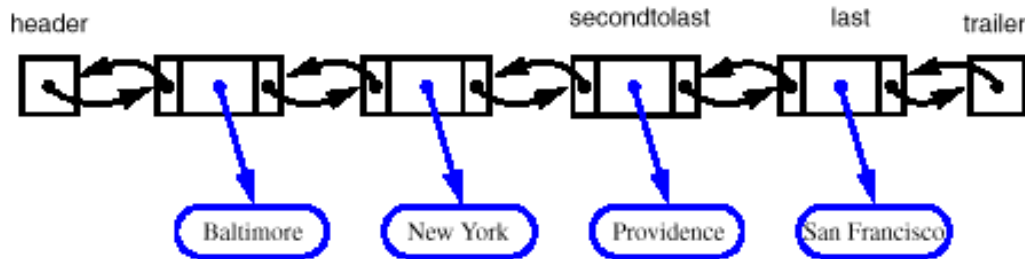
MEMORIJSKA REPREZENTACIJA DEKA

Lančana lista

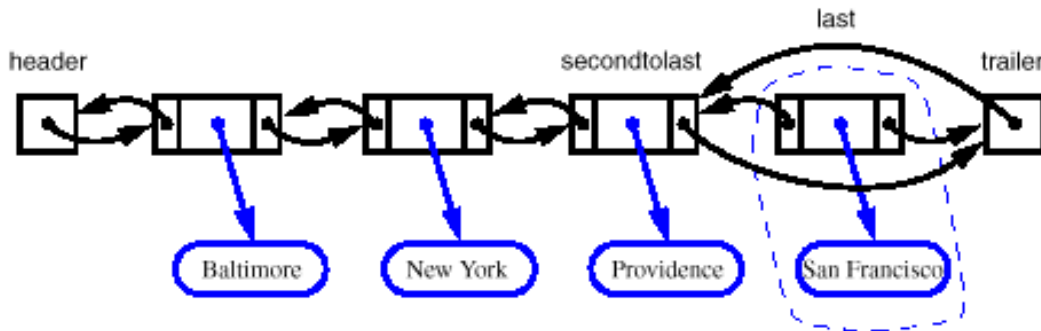
- Ako se primeni **jednostruko spregnuta lančana lista**, operacija **brisanja** na repu deka je isuviše **zahtevna**.
- Zbog toga se koriste **dvostruko spregnete lančane liste** sa specijalnih zaglavljima koja ukazuju na prvi odnosno poslednji element deka.
- Svaki element liste ima ukazatelj na prethodni i sledeći element liste



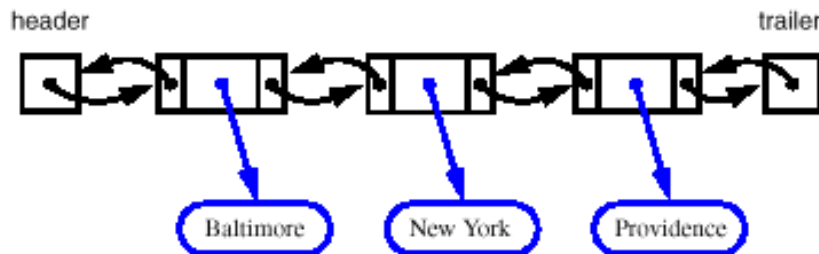
REMOVE LAST OPERACIJA



- Početno stanje



- Brisanje i promena pointera



- Stanje posle primene operacije


ZA SAMOSTALNI RAD

- Napisati pseudokod osnovnih operacija za dodavanje i brisanje elementa u/iz Dek/a:
 - za statičku implementaciju preko polja
 - za dinamičku implementaciju preko lančane liste



SPECIJALNI TIPOVI REDA:

REDOVI SA PRIORITETOM

- Specijalizovane strukture podataka.
 - Slične Redu - imaju ukazatelj na početak i ukazatelj na kraj reda.
 - Elementi se brišu sa početka.
 - Elementi su uređeni na osnovu vrednosti jednog polja (*key* ili prioritet) tako da je element sa najmanjom (ili najvećom) vrednošću prioriteta na početku reda.
 - Elementi sa istim prioritetom se brišu u skladu sa dogovorenim kriterijumom, napr. po FIFO principu kao kod običnog reda.
 - Elementi se dodaju tako da se održi prethodno navedeni uslov.
 - Koriste se u *multitasking* OS
 - Generalno, predstavljaju se korišćenjem “heap” strukture podataka.
 - Složenost operacije dodavanja je $O(n)$
 - Brisanje zahteva $O(1)$
- 

PITANJA, IDEJE, KOMENTARI

