



# **Katedra za računarsku tehniku i informatiku**

## **Algoritmi i strukture podataka**

**Milo V. Tomašević**

**Odsek za softversko inženjerstvo [SI]**

# Sadržaj

---

- I **Linearne strukture podataka**
  - II **Nelinearne strukture podataka**
  - III **Pretraživanje**
  - IV **Sortiranje**
-

# Uvod

---

- ***Algorithms + Data Structures = Programs***
  - Dva osnovna gradivna bloka za implementaciju programskih sistema
  - ***Struktura podataka*** – opis načina organizacije podataka
  - ***Algoritam*** – opis načina obrade podataka
-

# O algoritmima

---

- Osobine
  - ✓ jedan ili više ulaza i izlaza
  - ✓ nedvosmislenost i ostvarljivost naredbi
  - ✓ konačno vreme izvršavanja
  
- Determinističko ponašanje (ali i pseudoalgoritmi)
  
- Mera efikasnosti – vremenska i prostorna složenost
  
- Najčešće u pseudojeziku
  
- Implementacija algoritama (jezik, mašina)

# O strukturama podataka

---

- Modelira objekte i podatke iz problema
- Elementarni, primitivni tipovi
- Složeni, struktuirani tipovi
  - ✓ logička struktura
  - ✓ fizička struktura
- Osnovni načini struktuiranja
  - ✓ homogeni - nizovi
  - ✓ nehomogeni - zapisi

# Zapis

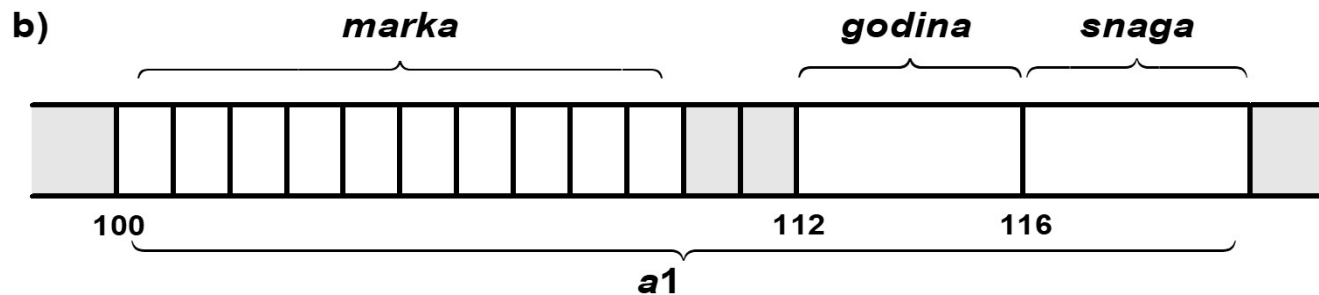
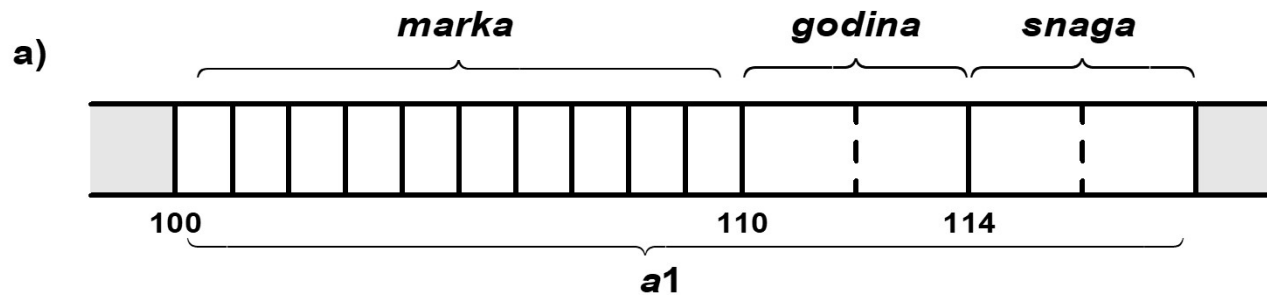
---

- Agregacija elemenata različitog tipa (polja)
- Tretira se kao logička celina
- Koristi se u implementaciji
  - ✓ dinamičkih struktura
  - ✓ datoteka
- Operacije obično nad poljima

# Zapis

Smeštanje u memoriji

- a) kontinualna alokacija polja
- b) vezivanje polja za početak reči



# Podele struktura

---

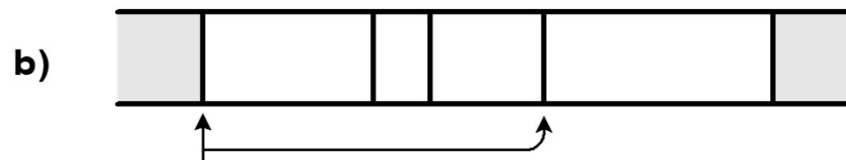
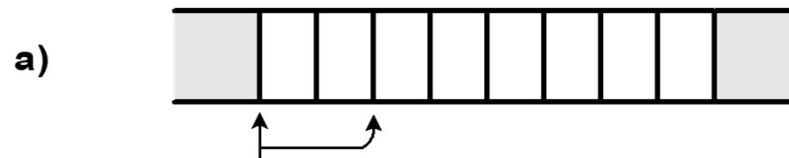
- Po relacijama elemenata
  - ✓ linearne
  - ✓ nelinearne
  
- Po mogućnosti promene veličine
  - ✓ statičke
  - ✓ dinamičke
  
- Po mestu čuvanja
  - ✓ unutrašnje (u operativnoj memoriji)
  - ✓ spoljašnje (datoteke)



# Memorijska reprezentacija

## Sekvencijalna

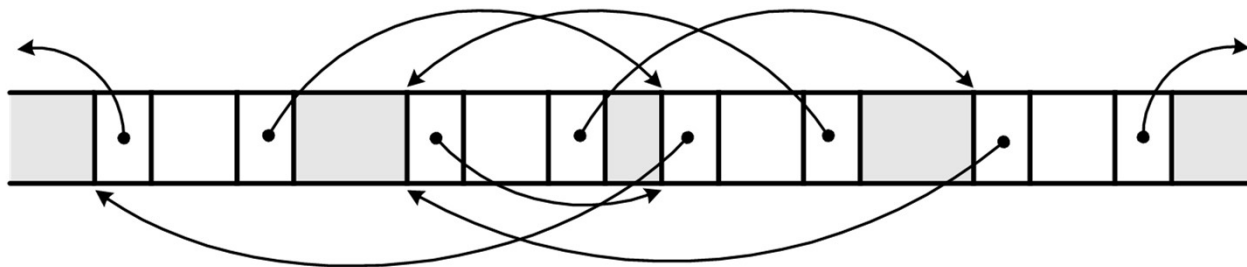
- Fizički i logički poredak isti
- Pristup elementima direktan
- Obično za linearne strukture



# Memorijska reprezentacija

## Ulančana

- Fizički i logički poredak različiti
- Pristup elementima indirektan
- Koristi pokazivački mehanizam
- Obično za nelinearne strukture



# Linearne strukture podataka

---

- **Linearna lista**  $(a_1, a_2, \dots, a_n), n \geq 0$
  - Apstraktni tip nezavisno od implementacije
  - Strukturno svojstvo – jednodimenzionalnost
  - Operacije
    - ✓ obilazak po poretku 1..n
    - ✓ pretraživanje
    - ✓ pristup elementu
    - ✓ umetanje
    - ✓ brisanje, ...
-

# Linearne strukture podataka

---

- Implementacija
    - ✓ sekvencijalna – **niz**
    - ✓ ulančana – **ulančana lista**
  
  - Disciplina pristupa
    - ✓ LIFO - **stek**
    - ✓ FIFO - **red**
-

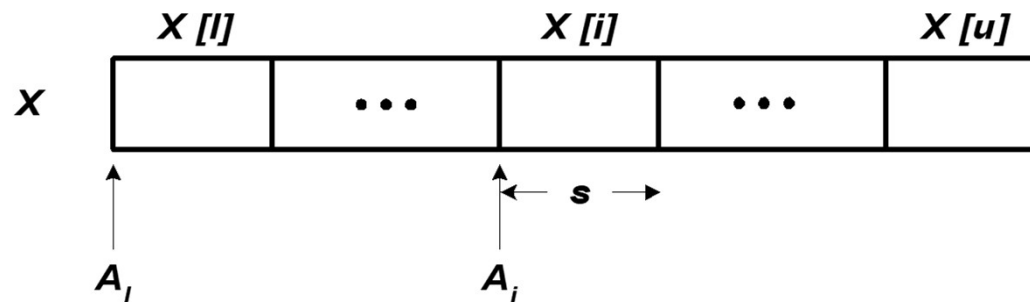
# Nizovi

---

- **Niz** - homogeni strukturirani tip
- Indeks – pozicija elementa u nizu
- $X[l : u] = \{X[i]\}, i = l, l + 1, \dots, u - 1, u$
- Direktan pristup elementu bez ograničenja
- Višedimenzionalni nizovi (nizovi nizova)
- Selekcija elementa
- Operacije

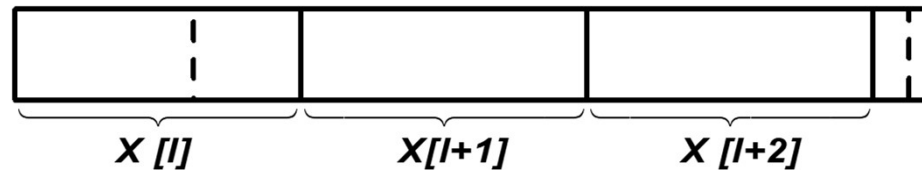
# Smeštanje vektora

- Sekvencijalna reprezentacija, ne razdvaja se logički koncept od implementacije
- $A_i = A_l + (i - l)s$  (jedan element zauzima  $s$  reči)
- Efikasan, direktan pristup, neefikasno umetanje i brisanje

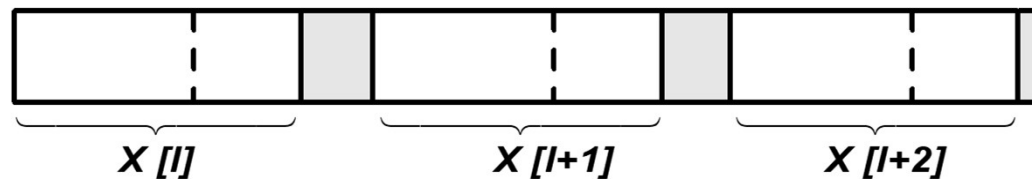


# Smeštanje vektora

- Kontinualno
  - ✓ prostorno efikasno
  - ✓ neefikasan pristup



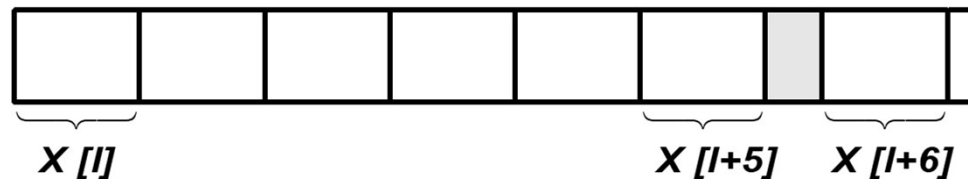
- Sa dopunjavanjem (*padding*)
  - ✓ iskorišćenje prostora  $S_u = s / \lceil s \rceil$
  - ✓ efikasniji pristup



# Smeštanje vektora

## ➤ Pakovanje

- ✓  $k = \lfloor 1/s \rfloor$  elemenata u jednoj reči
- ✓ prostorno iskorišćenje  $S_u = ks$
- ✓  $A_i = A_l + \lfloor (i - l)/k \rfloor$   
pozicija u reči  $(i - l) \bmod k$





# Smeštanje matrica

$X$

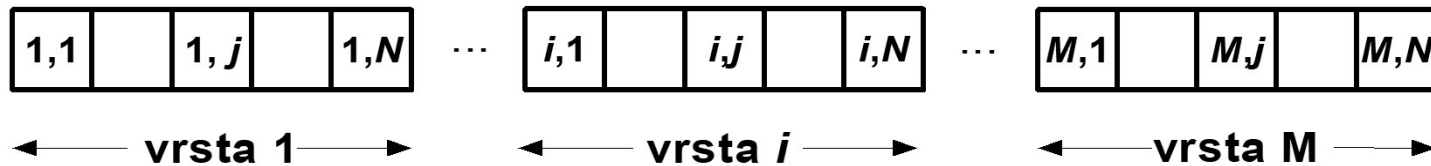
	1	$j$	$N$
1	1,1	1, $j$	1, $N$
$i$	$i,1$	$i, j$	$i, N$
$M$	$M,1$	$M, j$	$M, N$

- Cilj - efikasna adresna funkcija
- Linearizacija
  - ✓ po vrstama (*row-major*)
  - ✓ po kolonama (*column-major*)

# Smeštanje po vrstama

➤  $X[l_1:u_1, l_2:u_2]$

➤  $A_{i,j} = A_{l_1,l_2} + ((i - l_1)(u_2 - l_2 + 1) + j - l_2)s$



➤  $X[1:M, 1:N]$

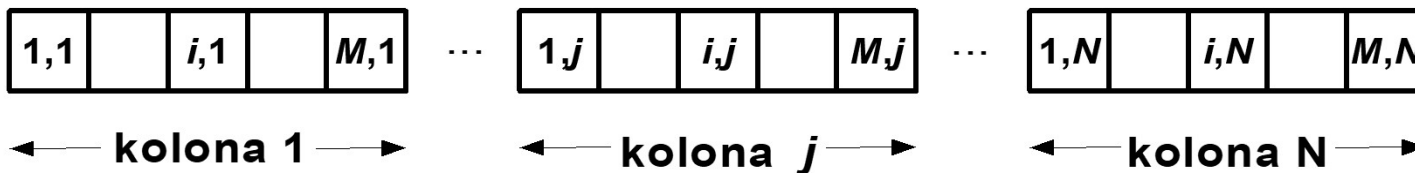
➤  $A_{i,j} = A_{1,1} + ((i - 1)N + j - 1)s$

➤ ne zavisi od broja vrsta

# Smeštanje po kolonama

➤  $X[l_1:u_1, l_2:u_2]$

➤  $A_{i,j} = A_{l_1,l_2} + ((j - l_2)(u_1 - l_1 + 1) + i - l_1)s$



➤  $X[1:M, 1:N]$

➤  $A_{i,j} = A_{1,1} + ((j - 1)M + i - 1)s$

➤ ne zavisi od broja kolona

# Višedimenzionalni nizovi

➤  $X[1:u_1, 1:u_2, \dots, 1:u_n]$

➤ Generalizovano smeštanje po vrstama

$X[1, \dots, 1, 1]$   $X[1, \dots, 1, 2]$  ...  $X[1, \dots, 1, u_n]$

$X[1, \dots, 2, 1]$   $X[1, \dots, 2, 2]$  ...  $X[1, \dots, 2, u_n]$

...

$X[u_1, \dots, u_{n-1}, 1]$   $X[u_1, \dots, u_{n-1}, 2]$  ...  $X[u_1, \dots, u_{n-1}, u_n]$

➤ Leksikografski poredak

# Višedimenzionalni nizovi

$$A_{i_1, \dots, i_n} = A_{1, \dots, 1} + ((i_1 - 1)u_2 u_3 \dots u_n + (i_2 - 1)u_3 u_4 \dots u_n + \dots + (i_{n-1} - 1)u_n + i_n - 1)s$$

- Efikasno izračunavanje

```
offset = 0
for j = 1 to n do
    offset = U[j] offset + I[j] - 1
end_for
A = A1, ..., 1 + s * offset
```

- Implikacije načina smeštanja

# Optimizacije pri smeštanju

- Cilj – ušteda prostora, ali ipak efikasan pristup
- Primer - trougaone matrice

$$\begin{array}{cc} \text{a)} \begin{bmatrix} \underline{x} & & \\ & \underline{x} & \mathbf{0} \\ & \mathbf{X} & \\ & & \underline{x} \end{bmatrix} & \text{b)} \begin{bmatrix} \underline{x} & & \\ & \underline{x} & \mathbf{X} \\ & \mathbf{0} & \\ & & \underline{x} \end{bmatrix} \\ \text{c)} \begin{bmatrix} \mathbf{0} & & \\ & \mathbf{0} & \mathbf{0} \\ & \mathbf{X} & \\ & & \mathbf{0} \end{bmatrix} & \text{d)} \begin{bmatrix} \mathbf{0} & & \\ & \mathbf{0} & \mathbf{X} \\ & \mathbf{0} & \\ & & \mathbf{0} \end{bmatrix} \end{array}$$

# Trougaona matrica

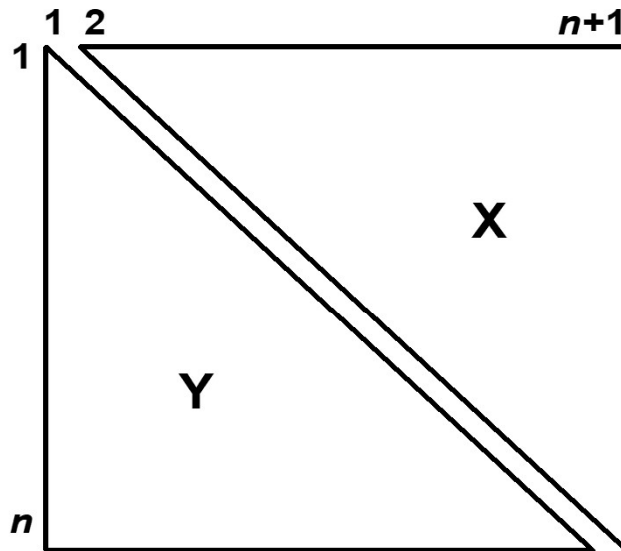
- Npr. donja trougaona matrica  
 $X[i,j] = 0, i < j$

- Primena smeštanja po vrstama

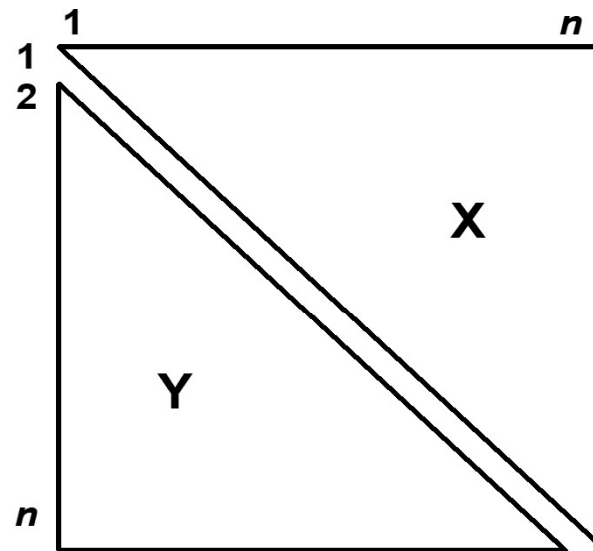
$X[1,1]$   $X[2,1]$   $X[2,2]$   $X[3,1]$   $X[3,2]$   $X[3,3]$  ...

- Ušteda u prostoru oko 50%
- $A_{i,j} = A_{1,1} + (i(i-1)/2 + j-1)s$  ako je  $i \geq j$

# Trougaone matrice



a)



b)

- $X[i,j] = Z[i,j + 1]$  ako je  $i \leq j$        $X[i,j] = 0$  za  $i > j$
- $Y[i,j] = Z[i,j]$  ako je  $i \geq j$        $Y[i,j] = 0$  za  $i < j$



# Retki nizovi

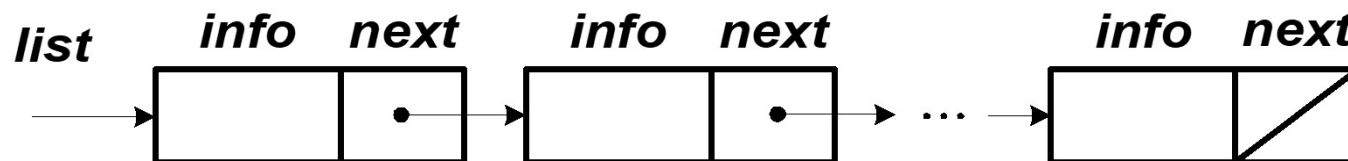
- Relativno veliki broj nultih elemenata
- Pamte se samo nenulti elementi – ušteda prostora
- Vektorska reprezentacija
- Neefikasne operacije!

$$X = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 \end{bmatrix}$$

R	C	V
1	3	4
2	4	5
2	6	11
4	1	9
4	4	8
5	7	15

# Ulančane liste

- Nedostaci sekvencijalne implementacije
- **Ulančana lista** - ulančana implementacija linearne liste
- Element liste - čvor
- Dinamička alokacija, pokazivački mehanizam



# Vrste lista

---

- Po načinu povezanosti
  - ✓ jednostruko ulančane
  - ✓ dvostruko ulančane
  
  - ✓ kružne
  - ✓ nekružne
  
- Po organizaciji
  - ✓ uređene
  - ✓ neuređene

# Jednostruko ulančane liste

- Umetanje iza zadatog čvora

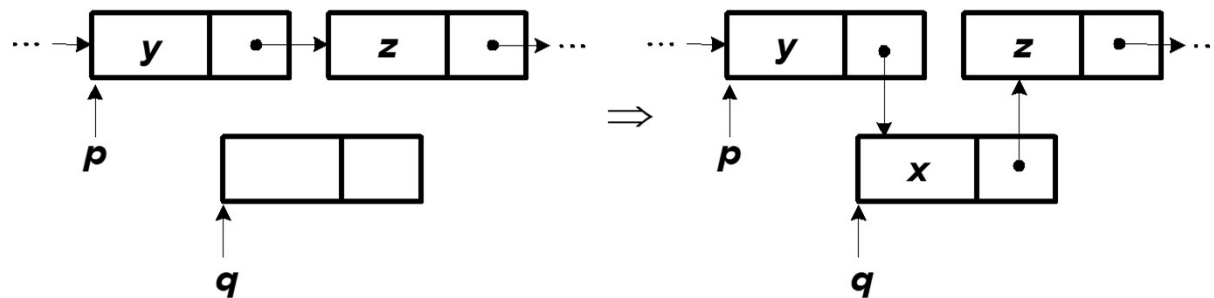
INSERT-AFTER( $p, x$ )

$q = \text{GETNODE}$

$\text{info}(q) = x$

$\text{next}(q) = \text{next}(p)$

$\text{next}(p) = q$



# Jednostruko ulančane liste

- Umetanje ispred zadatog čvora

INSERT-BEFORE( $p, x$ )

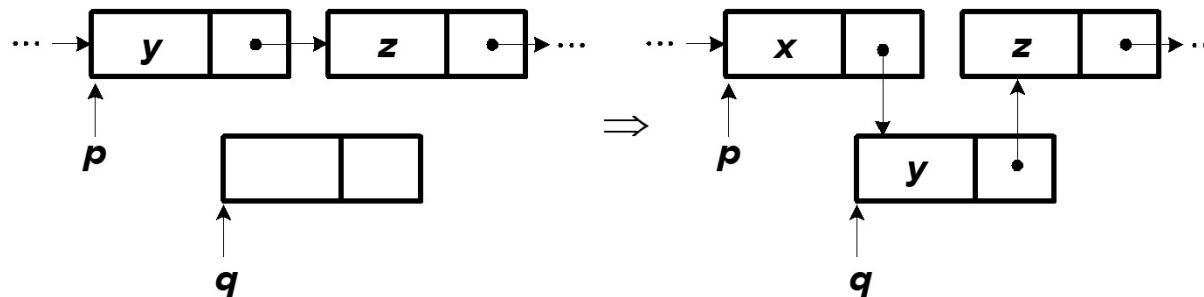
$q = \text{GETNODE}$

$\text{next}(q) = \text{next}(p)$

$\text{info}(q) = \text{info}(p)$

$\text{info}(p) = x$

$\text{next}(p) = q$



# Jednostruko ulančane liste

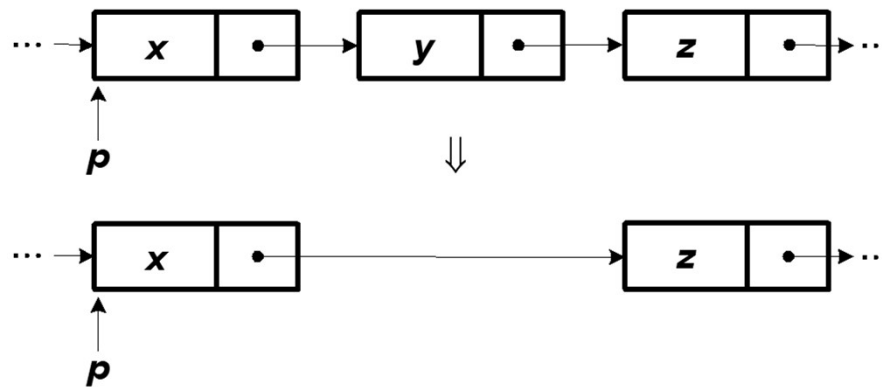
- Brisanje iza zadatog čvora

DELETE-AFTER( $p$ )

$q = next(p)$

$next(p) = next(q)$

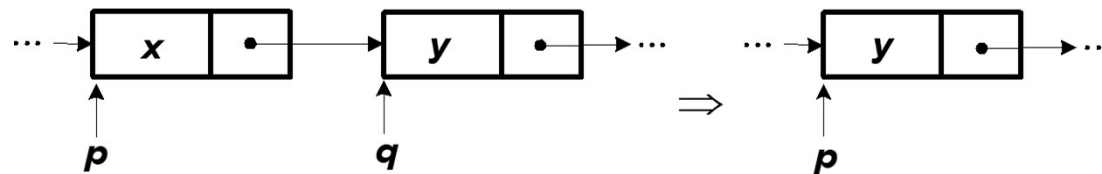
FREENODE( $q$ )



# Jednostruko ulančane liste

- Brisanje zadatog čvora

```
DELETE(p)  
q = next(p)  
next(p) = next(q)  
info(p) = info(q)  
FREENODE(q)
```



# Jednostruko ulančane liste

---

## Pretraživanje neuređene liste

```
SEARCH(list, x)
```

```
p = list
```

```
while (p ≠ nil) and (info(p) ≠ x) do
```

```
    p = next (p)
```

```
end_while
```

```
return p
```



# Jednostruko ulančane liste

## Pretraživanje uređene liste

SEARCH-ORD(*list*, *x*)

*p* = *list*

**while** (*p* ≠ nil) and (*info*(*p*) < *x*) **do**

*p* = *next*(*p*)

**end\_while**

**if** (*p* ≠ nil) and (*info*(*p*) > *x*) **then**

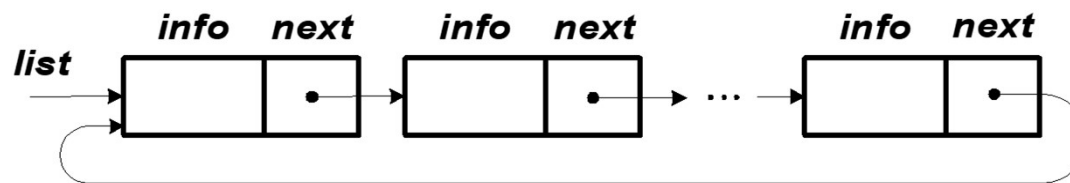
*p* = nil

**end\_if**

**return** *p*

# Kružne liste

- Poslednji čvor ukazuje na prvi



- Omogućeno kružno kretanje po listi
- Spoljašnji pokazivač na listu ponekad ukazuje na poslednji čvor

# Kružne liste

INSERT-AFTER-C( $p, x$ )

$q = \text{GETNODE}$

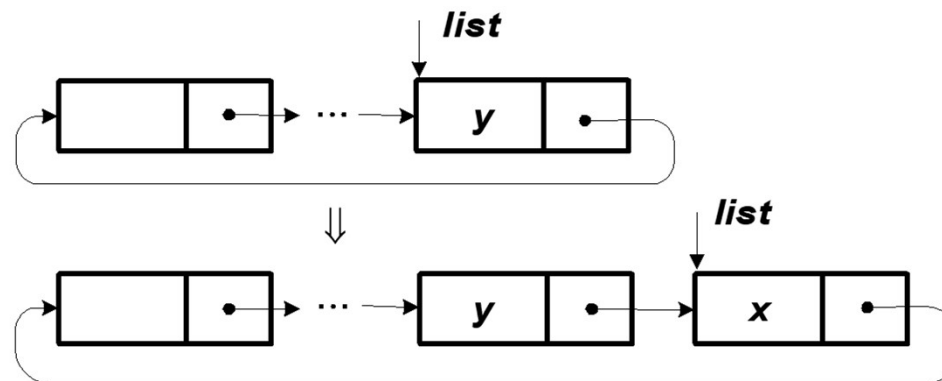
$\text{info}(q) = x$

$\text{next}(q) = \text{next}(p)$

$\text{next}(p) = q$

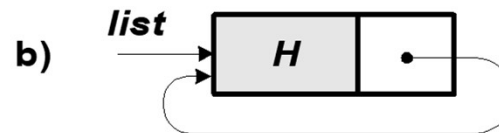
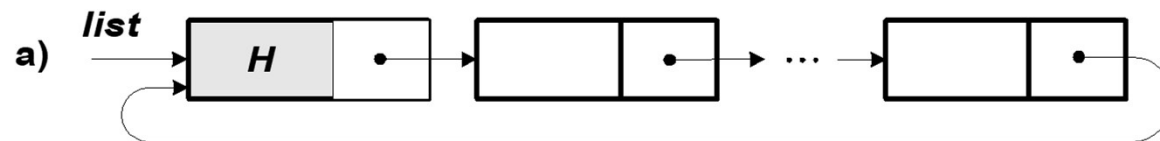
**if** ( $\text{list} = p$ ) **then**  
      $\text{list} = q$

**end\_if**



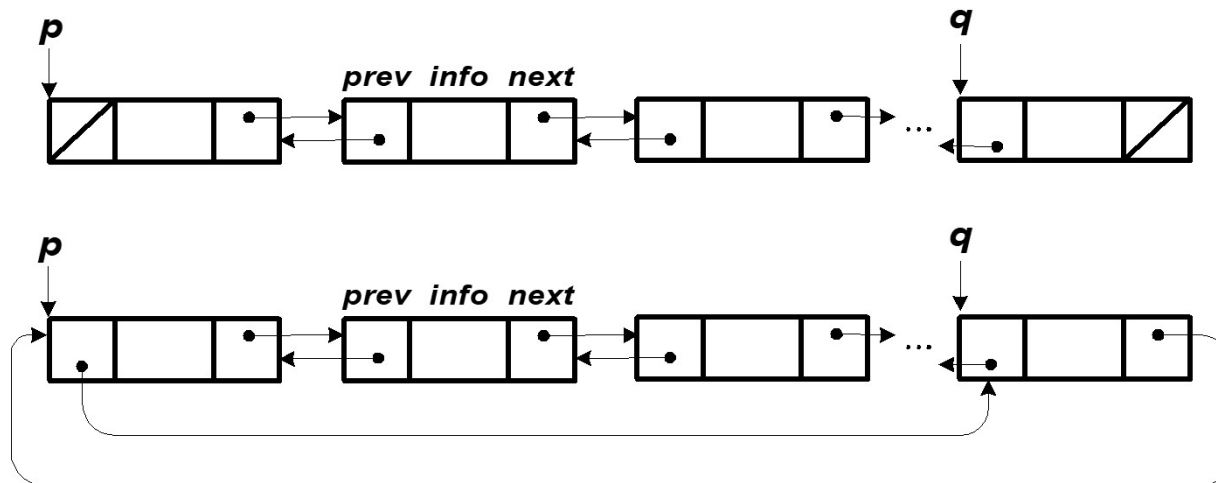
# Kružne liste sa zaglavljem

- Na početku poseban čvor – **zaglavlje**
- Ukazuje na prvi čvor
- Lista nikad nije prazna!



# Dvostruko ulančane liste

- Pokazivači na prethodnika i sledbenika
- Lako kretanje u oba smeru
- $next(prev(p)) = p = prev(next(p))$



# Dvostruko ulančane liste

## Umetanje

INSERT-AFTER-D( $p, x$ )

$q = \text{GETNODE}$

$\text{info}(q) = x$

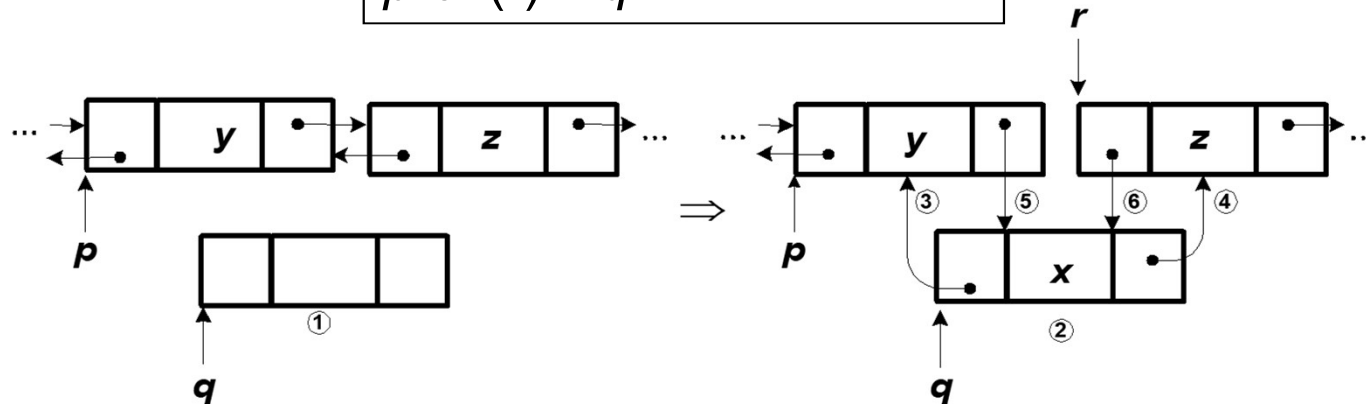
$r = \text{next}(p)$

$\text{prev}(q) = p$

$\text{next}(q) = r$

$\text{next}(p) = q$

$\text{prev}(r) = q$



# Dvostruko ulančane liste

Brisanje

DELETE-D( $p$ )

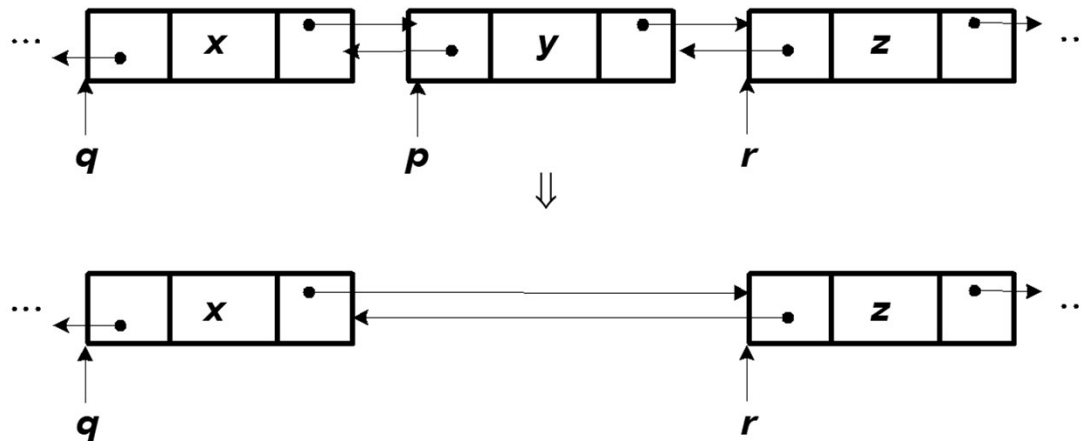
$q = \text{prev}(p)$

$r = \text{next}(p)$

$\text{next}(q) = r$

$\text{prev}(r) = q$

$\text{FREENODE}(p)$



# Poređenje nizova i lista

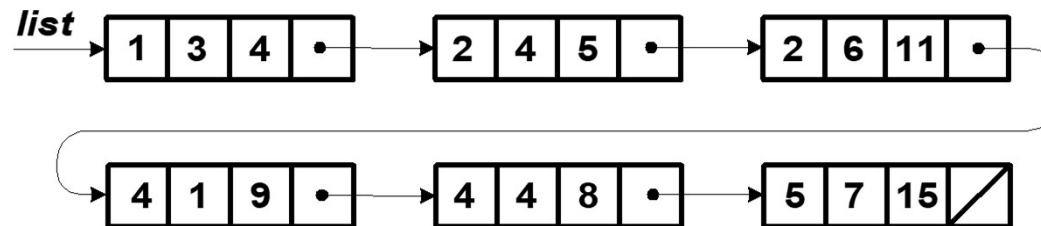
---

- Ulančana reprezentacija
  - ✓ obično bolje koristi prostor
  - ✓ efikasnija za operacije umetanja, brisanja, spajanja, ...
  - ✓ fleksibilnost za dinamičke strukture
  
- Sekvencijalna reprezentacija
  - ✓ efikasniji pristup elementu
  - ✓ pogodna za statičke strukture
  
- Vektorska implementacija liste



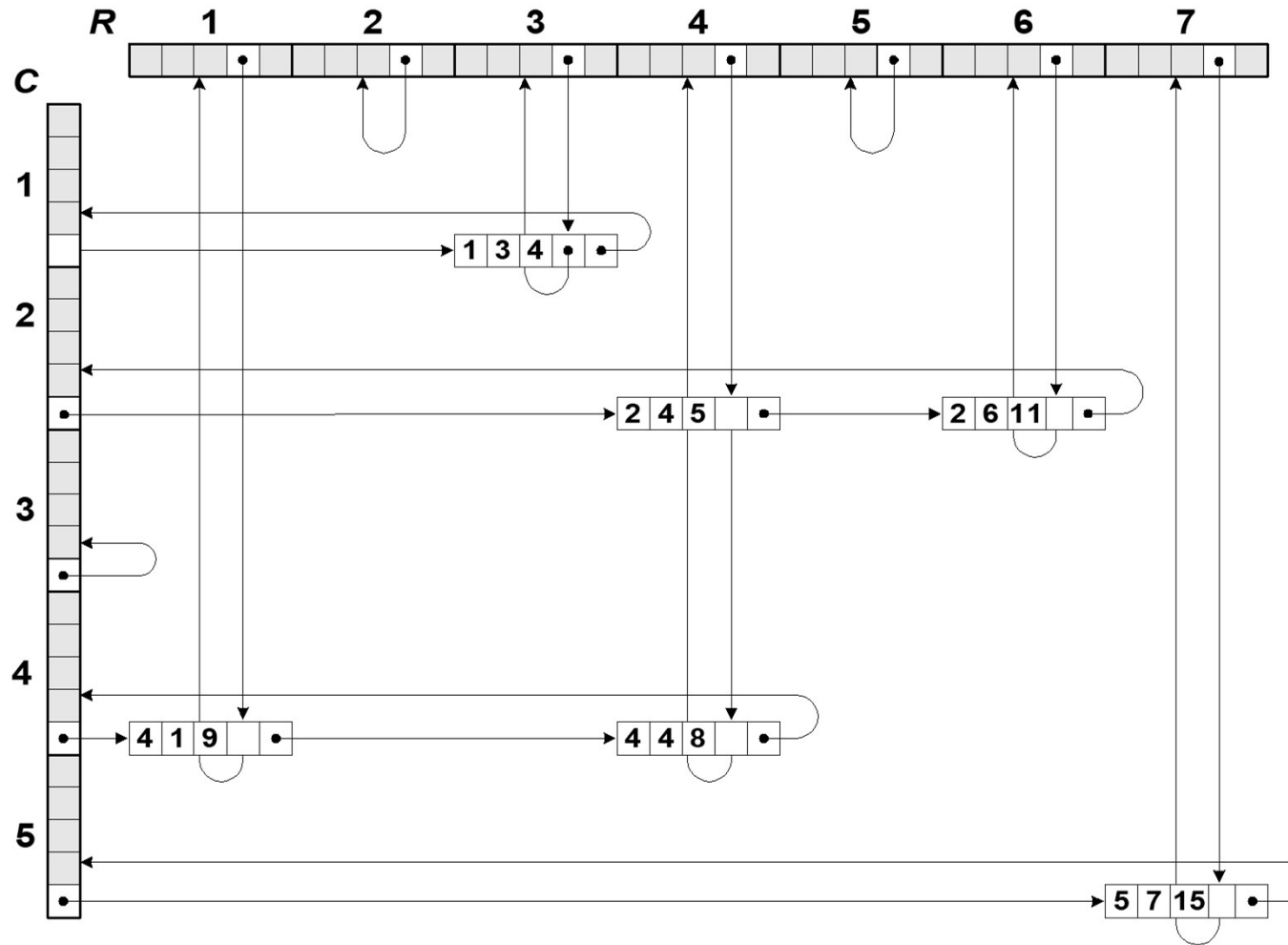
# Predstavljanje retkih nizova

- Jednostruko ulančane liste (čvor - nenulti element)



- Lakše umetanje i brisanje
- Pristup i dalje neefikasan
- Poboljšanje – kružne liste ulančane po vrstama i kolonama

# Predstavljanje retkih matrica



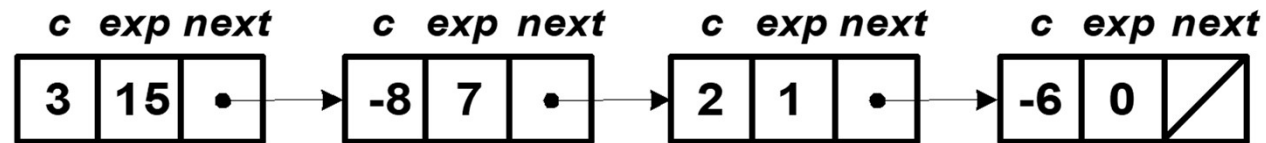
# Predstavljanje skupova

---

- Vektorska realizacija
  - ✓ prostorna neefikasnost
  - ✓ efikasne operacije  
(unija, presek, razlika, pripadnost)
  
- Ulančana implementacija
  - ✓ čvor - element skupa
  - ✓ prostorno efikasnije
  - ✓ manje efikasne operacije  
(za uređene liste –  $O(n)$ ,  
za neuređene liste –  $O(n^2)$ )

# Predstavljanje polinoma

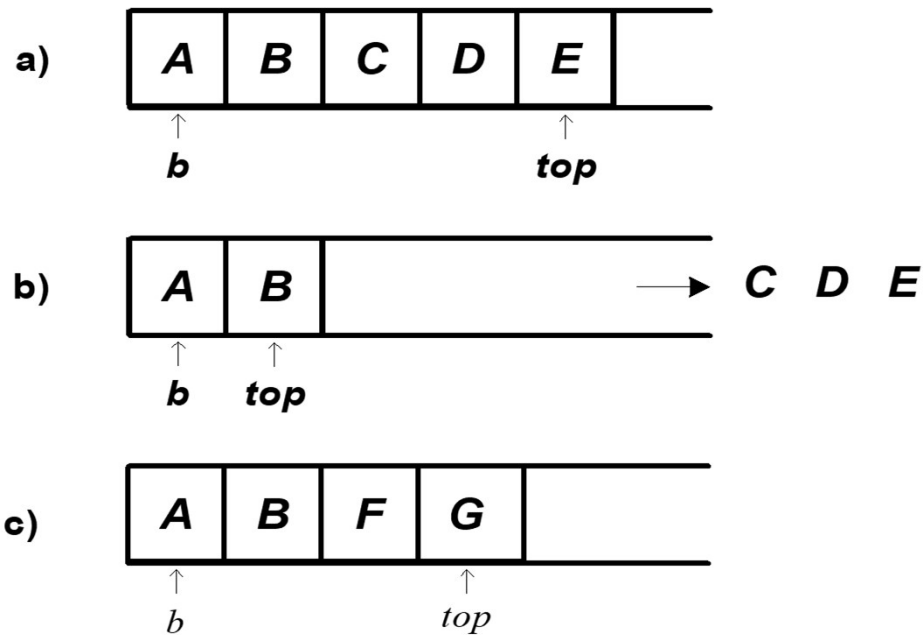
- Jednostruko ulančane kružne liste sa zaglavljem
- Npr.  $3x^{15} - 8x^7 + 2x - 6$



- Ako su liste uređene po vrednosti polja eksponenta, efikasnost operacija veća

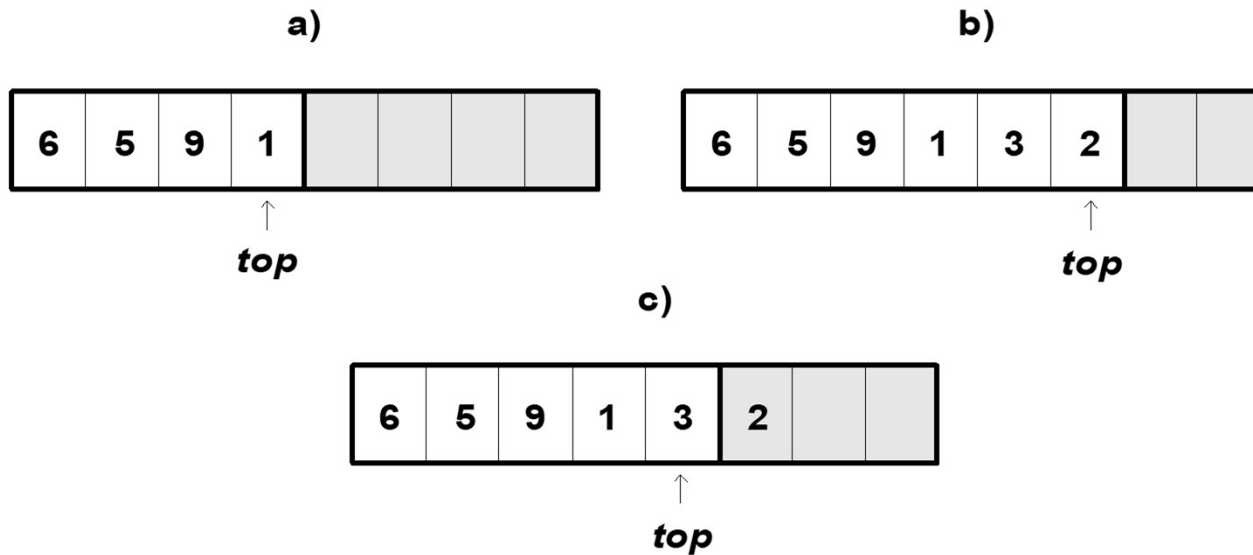
# Stekovi

- Linearna lista sa LIFO disciplinom pristupa
- Jedan pristupni kraj – vrh steka
- Dinamička, homogena struktura



# Sekvencijalna reprezentacija

- Niz  $S[1:n]$
- Pokazivač steka  $top[S]$
- Sadržaj  $S[1], \dots, S[top[S]]$



# Operacije sa stekom

Provera da li je  
prazan

```
STACK-EMPTY(S)  
if (top[S] = 0) then  
    return true  
else  
    return false  
end_if
```

Umetanje

```
PUSH(S, x)  
if (top[S] = n) then  
    ERROR(Overflow)  
else  
    top[S] = top[S] + 1  
    S[top[S]] = x  
end_if
```

# Operacije sa stekom

Uklanjanje

```
POP(S)  
if ( $top[S] = 0$ ) then  
    return underflow  
else  
     $x = S[top[S]]$   
     $top[S] = top[S] - 1$   
    return  $x$   
end_if
```

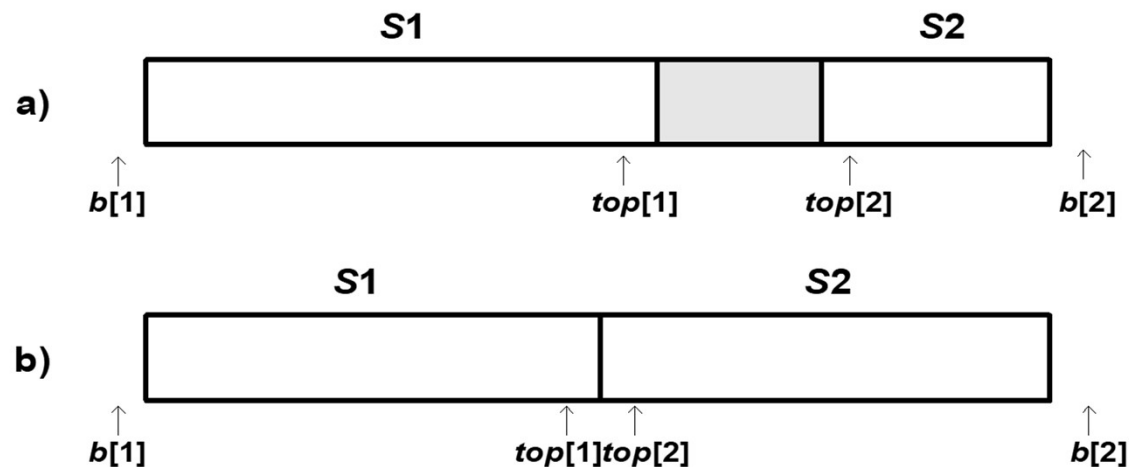
Čitanje bez  
uklanjanja

```
TOP(S)  
if ( $top[S] = 0$ ) then  
    return underflow  
else  
    return  $S[top[S]]$   
end_if
```

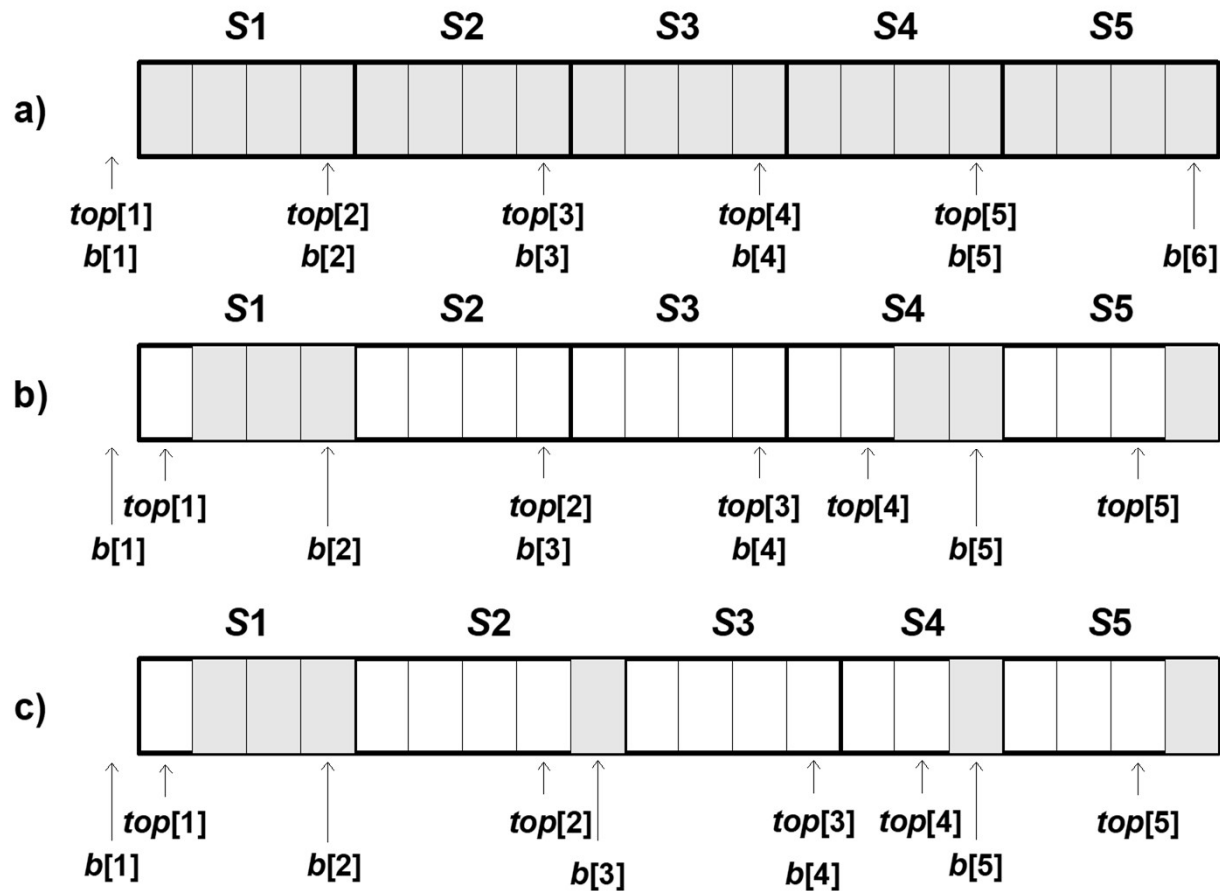


# Implementacija dva steka

- Rastu jedan prema drugom
- Realokacija prostora između stekova
- Uslov prekoračenja  $top[2] = top[1] + 1$



# Implementacija više stekova



# Implementacija više stekova

Brisanje

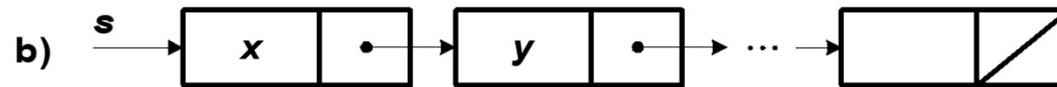
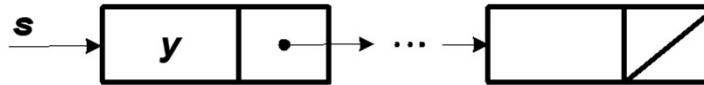
```
POP-M(i)  
if ( $top[i] = b[i]$ ) then  
    return underflow  
else  
     $x = V[top[i]]$   
     $top[i] = top[i] - 1$   
    return  $x$   
end_if
```

Umetanje

```
PUSH-M(i,  $x$ )  
if ( $top[i] = b[i + 1]$ ) then  
    ERROR(Overflow)  
else  
     $top[i] = top[i] + 1$   
     $V[top[i]] = x$   
end_if
```

➤ Poboljšanje - *Garwick*-ov algoritam

# Ulančana reprezentacija steka



```

PUSH-L(s, x)
p = GETNODE
info(p) = x
next(p) = s
s = p
    
```

```

POP-L(s)
if (s = nil) then
    return underflow
else
    p = s
    s = next(p)
    x = info(p)
    FREENODE(p)
    return x
end_if
    
```

# Obrada aritmetičkih izraza

---

- Infiksna notacija (npr.  $A+B$ )
  - ✓ prirodna i uobičajena
  - ✓ grupisanje operanada uz operatore uz pomoć zagrada (npr.  $(A+B)*C$ ) i prioriteta (npr.  $A+B*C$ )
  - ✓ nepogodno za prevodioca (zahteva više skeniranja)
  
- Prefiksna (poljska) notacija (npr.  $+AB$ )
  
- Postfiksna (inverzna poljska) notacija (npr.  $AB+$ )

# Obrada aritmetičkih izraza

- Konverzija iz infiksne u postfiksnu
  1. infiksni izraz sa potpunim zagradama
  2. operator zameni odgovarajuću desnu zagradu
  3. uklone se leve zagrade

Primer:  $A+B*(C-D)+(E-F)*G/H$   
 $((A+(B*(C-D)))+(((E-F)*G)/H))$   
 $ABCD-*+EF-G*H/+$

- Osobine postfiksne notacije
  - ✓ isti poredak operanada kao u infiksnoj
  - ✓ operatori se izvršavaju po redosledu nailaska
  - ✓ nema potrebe za zagradama i prioritetom

# Izračunavanje postfixnog izraza

```
EVAL-EXP(postfix)
INIT_STACK(S, n)
while (not_end_of postfix) do
    x = INPUT(postfix)
    if (x = operand) then
        PUSH(S, x)
    else if (x = un_op) then
        oprnd = POP(S)
        rez = x oprnd
        PUSH(S, rez)
    else if (x = bin_op) then
        oprnd2 = POP(S)
        oprnd1 = POP(S)
        rez = oprnd1 x oprnd2
        PUSH(S, rez)
end_if
end_while
```

```
rez = POP(S)
if (STACK-EMPTY(S)) then
    return rez
else
    ERROR(Nepravilan izraz)
end_if
```

# Konverzija infiks u postfiks

```
IN2POST(infix, postfix)
INIT-STACK(S, n)
rank = 0
next = INPUT(infix)
while (next) do
    if (next = operand) then
        OUTPUT(next, postfix)
        rank = rank + 1
    else
        while (not(STACK-EMPTY(S)) and (IPR(next) ≤ SPR(TOP(S)))) do
            x = POP(S)
            OUTPUT(x, postfix)
            rank = rank + R(x)
            if (rank < 1) then
                ERROR(Nepravilan izraz)
            end_if
        end_while
        if (next ≠ ')') then
            PUSH(S, next)
        else
            x = POP(S)
        end_if
    end_if
    next = INPUT(infix)
end_while
```



# Obrada aritmetičkih izraza

```
while (not(STACK-EMPTY(S))) do  
    x = POP(S)  
    OUTPUT(x, postfix)  
    rank = rank + R(x)  
end_while  
if (rank ≠ 1) then  
    ERROR(Nepravilan izraz)  
end_if
```

operator	ipr	spr	R
+, -	2	2	-1
*, /	3	3	-1
↑	5	4	-1
(	6	0	-
)	1	-	-

# Generisanje koda

1-A mašina

Algoritam

Šablon za  $O1+O2$

1. LOAD O1
2. ADD O2
3. STORE T1

- operand na stek
- operator:
  - ✓ O2, pa O1 sa steka
  - ✓ kod po šablonu za dati operator
  - ✓ međurezultat na stek

# Obrada aritmetičkih izraza

infix:  $(A+B*C)/D$

postfix:  $ABC*+D/$

1-A mašina

```
LOAD B
MUL C
STORE T1
LOAD A
ADD T1
STORE T2
LOAD T2
DIV D
STORE T3
```

1-A mašina  
(optimizovano)

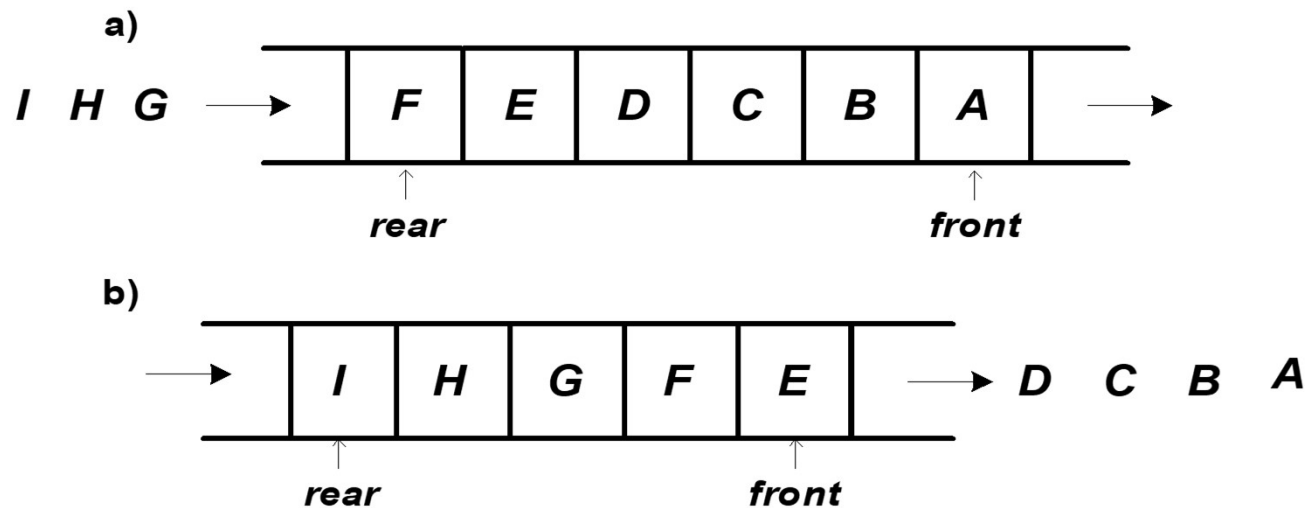
```
LOAD B
MUL C
ADD A
DIV D
STORE T1
```

0-A mašina

```
PUSH A
PUSH B
PUSH C
MUL
ADD
PUSH D
DIV
```

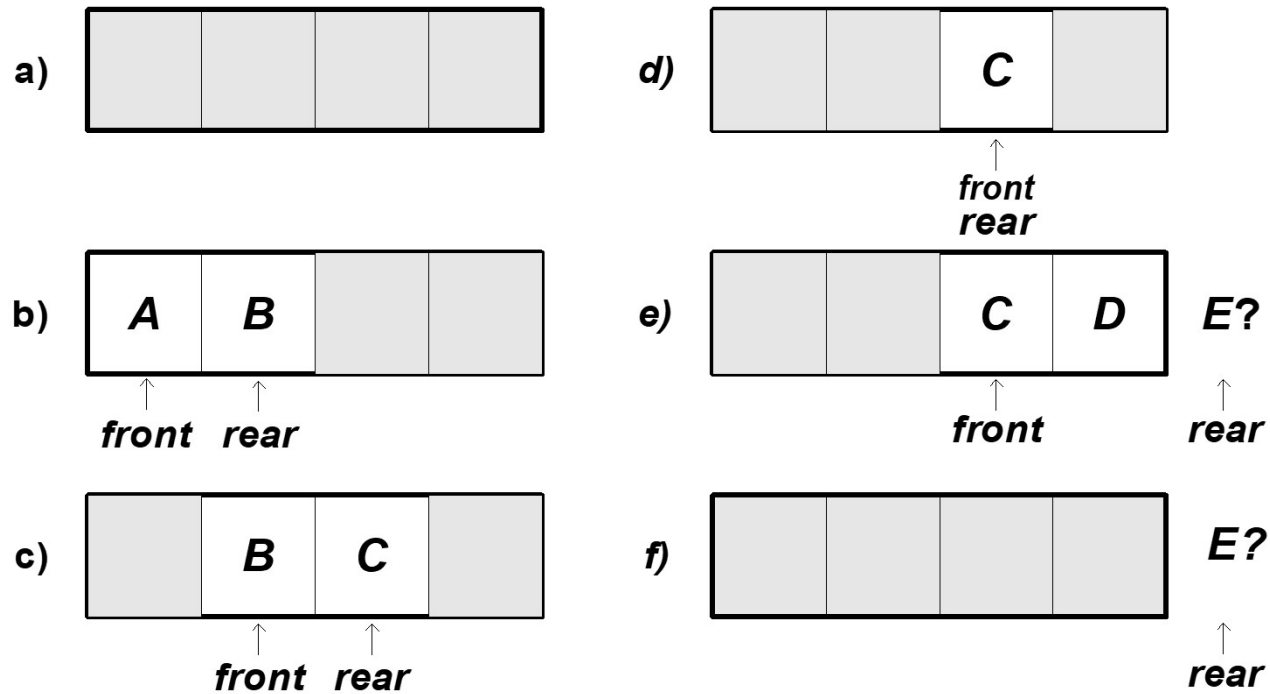
# Redovi

- Linearna lista sa FIFO disciplinom pristupa
- Dva pristupna kraja – čelo i začelje
- Dinamička, homogena struktura



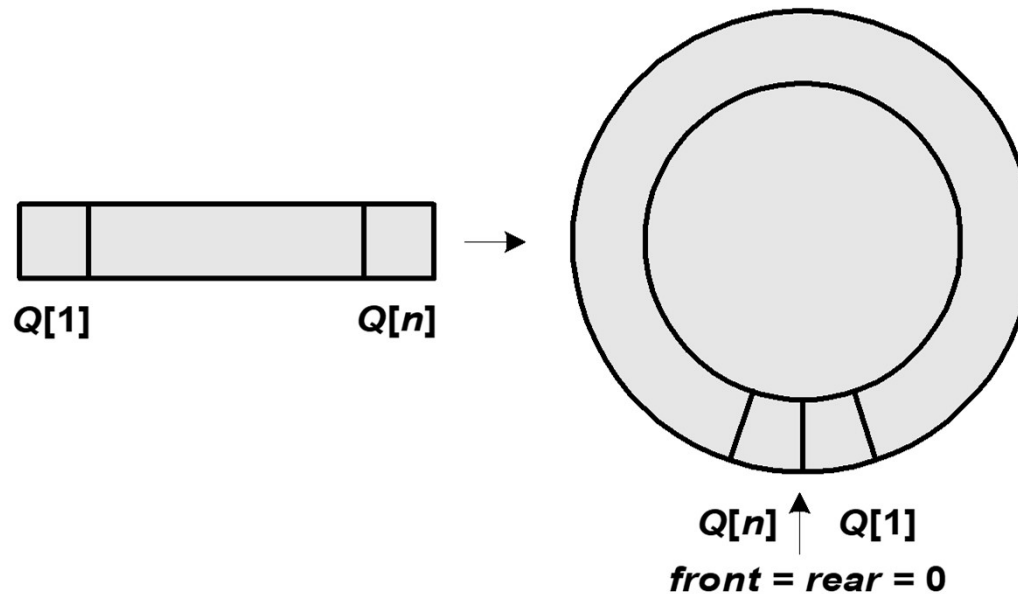
# Sekvencijalna reprezentacija reda

- Niz  $Q[1:n]$
- Pokazivači *front* i *rear*

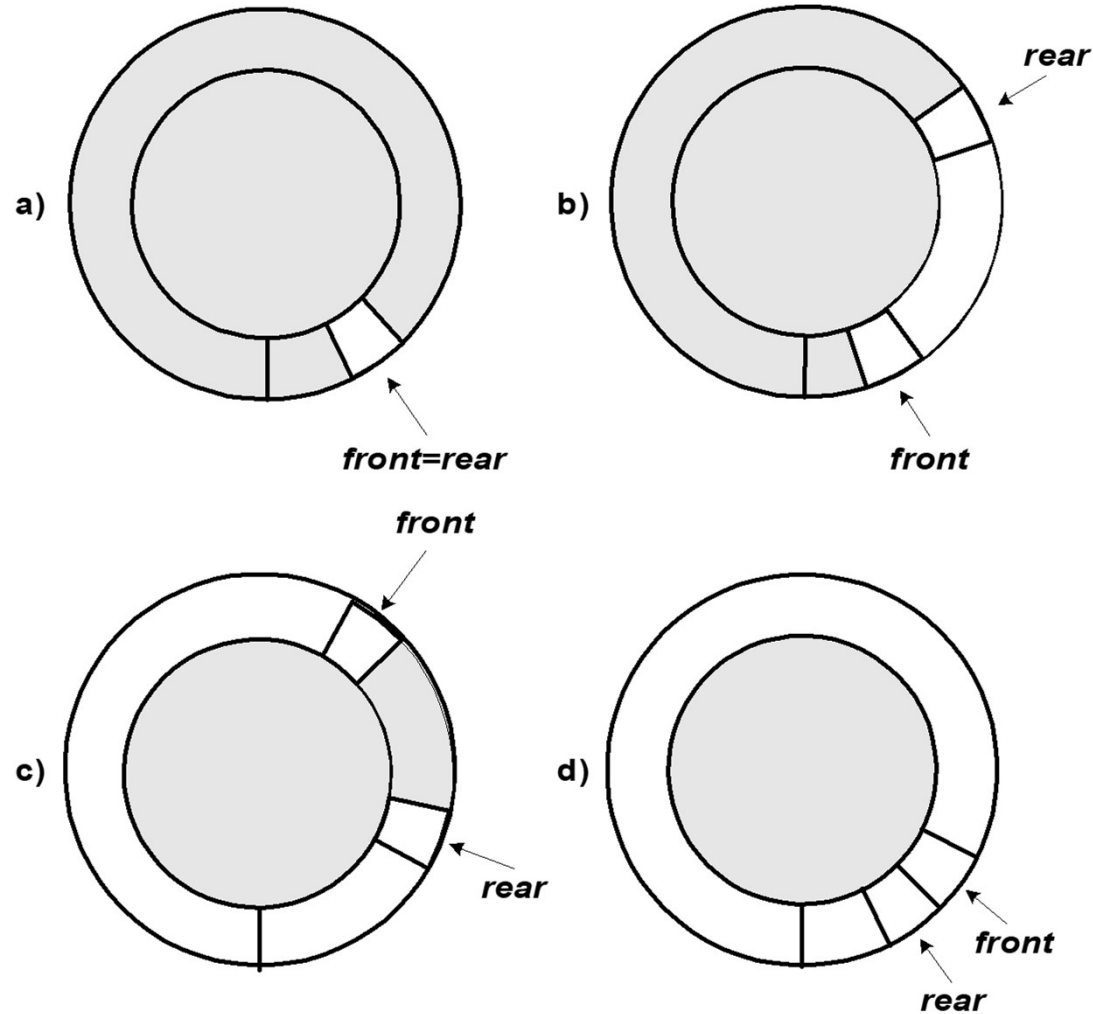


# Sekvencijalna reprezentacija reda

- Kružni bafer
- $Q[1]$  i  $Q[n]$  logički susedne



# Sekvencijalna reprezentacija reda



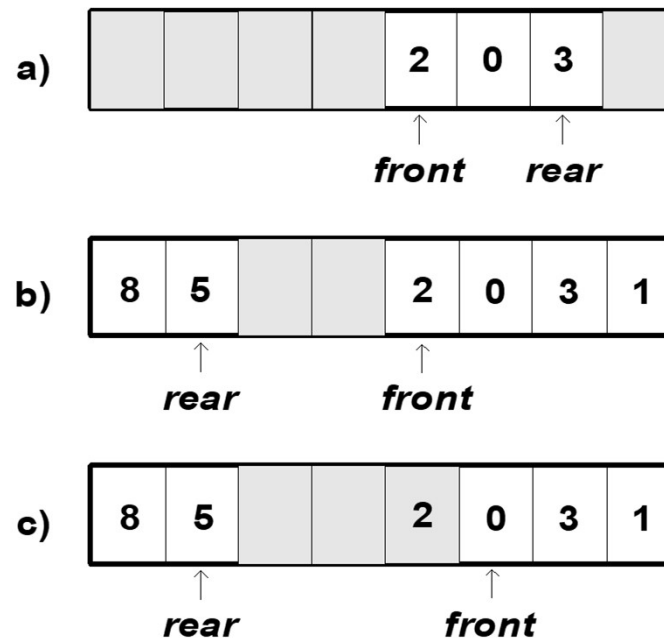
# Operacije sa redom

Umetanje

```
INSERT(Q, x)  
rear[Q] = rear[Q] mod n + 1  
if (front[Q] = rear[Q]) then  
    ERROR(Overflow)  
else  
    Q[rear[Q]] = x  
    if (front[Q] = 0) then  
        front[Q] = 1  
    end_if  
end_if
```



# Operacije sa redom



# Operacije sa redom

DELETE(Q)

```
if (front[Q] = 0) then
    return underflow
else
    x = Q[front[Q]]
    if (front[Q] = rear[Q]) then
        front[Q] = rear[Q] = 0
    else
        front[Q] = front[Q] mod n + 1
    end_if
    return x
end_if
```

QUEUE-EMPTY(Q)

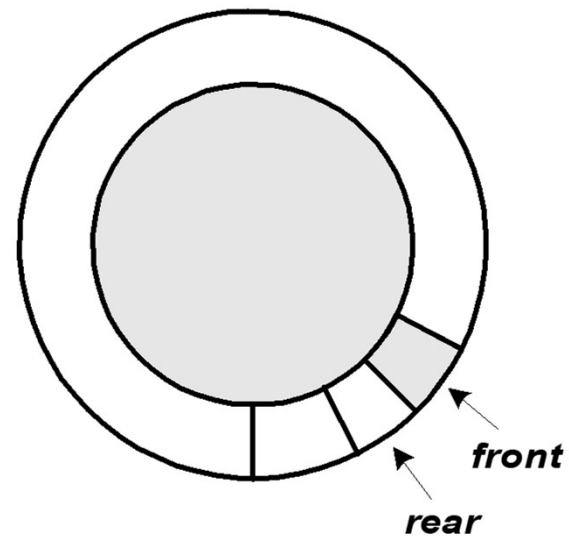
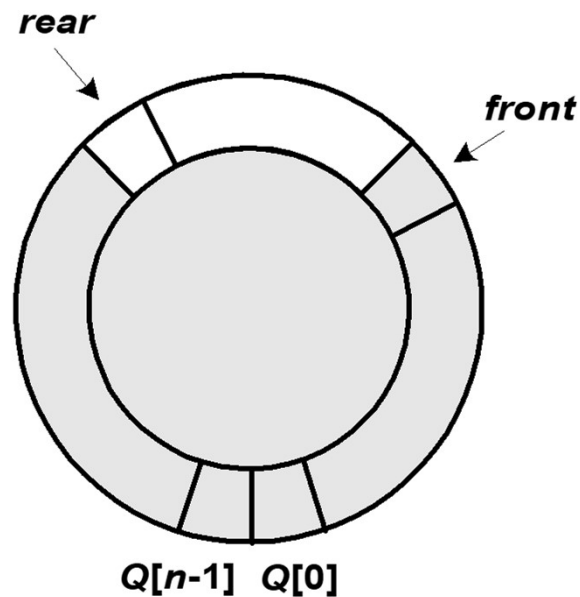
```
if (front[Q] = 0) then
    return true
else
    return false
end_if
```

Provera

Brisanje

# Alternativna reprezentacija

- Niz  $Q[0:n-1]$
- Prazan red  $front = rear$



# Operacije sa redom

Provera

```
QUEUE-EMPTY-0(Q)  
if (front[Q] = rear[Q]) then  
    return true  
else  
    return false  
end_if
```

Brisanje

```
DELETE_0(Q)  
if (front[Q] = rear[Q]) then  
    return underflow  
else  
    front[Q] = (front[Q] + 1) mod n  
    return Q[front[Q]]  
end_if
```

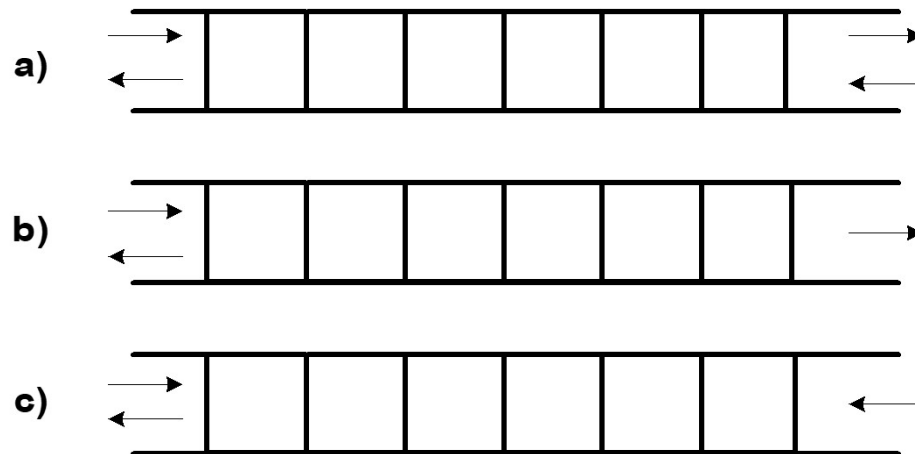
# Operacije sa redom

## Umetanje

```
INSERT_0(Q, x)  
rear[Q] = (rear[Q] + 1) mod n  
if (front[Q] = rear[Q]) then  
    ERROR(overflow)  
else  
    Q[rear[Q]] = x  
end_if
```

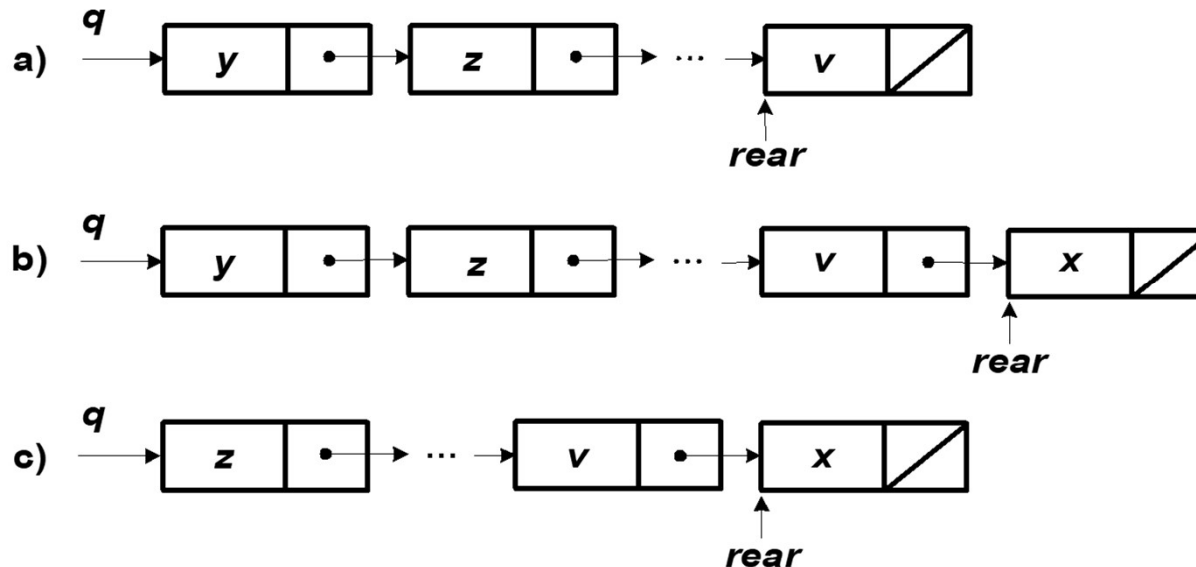
# Dvostrani red

- a) Umetanje i brisanje na oba kraja
- b) Umetanje samo na jednom kraju
- c) Brisanje samo na jednom kraju



# Ulančana reprezentacija reda

- Jednostruko ulančana lista (*rear* na poslednji čvor)
- Može i kružna lista



# Ulančana reprezentacija reda

INSERT-L( $q, x$ )

$p = \text{GETNODE}$

$\text{info}(p) = x$

$\text{next}(p) = \text{nil}$

**if** ( $\text{rear}[q] = \text{nil}$ ) **then**

$q = p$

**else**

$\text{next}(\text{rear}[q]) = p$

**end\_if**

$\text{rear}[q] = p$

DELETE-L( $q$ )

**if** ( $q = \text{nil}$ ) **then**

**return** underflow

**else**

$p = q$

$x = \text{info}(p)$

$q = \text{next}(p)$

**if** ( $q = \text{nil}$ ) **then**

$\text{rear}[q] = \text{nil}$

**end\_if**

$\text{FREENODE}(p)$

**return**  $x$

**end\_if**



# Prioritetni red

---

- Poredak brisanja elemenata zavisi od *prioriteta* – sadržaja elementa
- Rastući i opadajući prioritetni red
- Poredak umetanja može biti od značaja samo kod elemenata sa istim prioritetom
- Više implementacija različite efikasnosti

# Prioritetni red

PQ-INSERT( $pq, x$ )

$q = \text{nil}$

$p = pq$

**while** ( $p \neq \text{nil}$ ) and ( $x \geq \text{info}(p)$ ) **do**

$q = p$

$p = \text{next}(p)$

**end\_while**

**if** ( $q = \text{nil}$ ) **then**

    PUSH-L( $pq, x$ )

**else**

    INSERT-AFTER( $q, x$ )

**end\_if**