Osnovi programiranja – C

Dinamičke strukture podataka

Teme

- Dinamička alokacija memorije
- Nizovi
- Liste
- Stekovi
- Redovi za čekanje

Dinamička alokacija memorije

Situacija

- Program tokom izvršavanja traži od OS da odvoji deo neiskorišćene memorije za smeštaj promenljive određenog tipa.
- OS vraća početnu adresu u memoriji gde će promenljiva biti smeštena
- Program može pristupiti varijabli samo preko adrese, a za to je potreban pokazivač.

Dinamička alokacija memorije (2)

- Dinamički alocirana memorija se uzima iz dela memorije koji se zove hrpa (engl. heap)
- Hrpa se razlikuje od drugih područja memorije koje koristi C program.
- Ako se promenljiva definiše kao statička ili globalna, prostor za promenljivu se alocira iz područja globalnih podataka pre nego što počne izvršavanje programa.

Dinamička alokacija memorije (3)

Dinamička alokacija memorije (4)

- Kada funkcija main () završi, OS dealocira prostor za statičke i globalne varijable.
- Lokalna promenljiva (promenljiva deklarisana unutar bloka) se kreira u oblasti steka kada program ulazi u blok, a uništava kada program izlazi iz bloka.
- Dinamička alokacija memorije zahteva funkcije malloc i calloc deklarisane u <stdlib.h>.
- Funkcija free oslobađa memoriju koja je predhodno bila alocirana.

Dinamička alokacija memorije (5)

malloc(velicina) – alocira prostor tražene veličine i vraća pokazivač na njega

calloc(broj, velicina) – kao malloc ali istovremeno briše tu memoriju

realloc(pokaz, velicina) – menja veličinu alociranog prostora; funkcija vraća pokazivač na novoalocirani prostor u koji su prebačeni svi elementi starog, koji je nakon toga oslobođen

free (pokaz) – oslobađa memorijski prostor na koji pokazuje pokazivač

Dinamička alokacija memorije (6)

Opšta sintaksa:

```
void *malloc(size_t n);
void *calloc(size_t n, size_t size);
void *realloc(void *ptr, size_t n);
void free (void *ptr);
```

size_t je cjelobrojni tip bez preznaka definisan u <stddef.h> dovoljno velik da primi vrednost koju vraća sizeof operator.

Dinamička alokacija memorije (7)

- Ako nema dovoljno prostora za alokaciju memorije:

```
double *p
.....
p=(double *) malloc(128*sizeof(double));
if (!p) //ili if (p==NULL)
   printf("greška u alociranju memorije!");
```

Osnovne strukture podataka

- Kolekcija podataka organizovana na specijalni način
- Struktura podataka: skup elemenata i skup operacija nad njima
- Operacije:
  - Konstruisanje nove (prazne) strukture podataka
  - Dodavanje jednog elementa u strukturu podataka
  - Uklanjanje jednog elementa iz strukture podataka

Osnovne strukture podataka

Termin struktura podataka označava način organizacije određenih podataka u programu

Najosnovnije (fundamentalne) strukture podataka u programiranju predstavljaju sekvence elemenata

- Nizovi
- Liste

Specijalni slučajevi listi:

- Stekovi – elementi se dodaju i uklanjaju samo sa jednog kraja
- Redovi za čekanje – elementi se dodaju na jednom kraju a uklanjaju sa drugog kraja

Operacije nad strukturama podataka

- Upiti – kao rezultat daju neku informaciju o strukturi podataka ne menjajući je
- Modifikujuće operacije – menjaju sadržaj strukture podataka

Operacije nad strukturama podataka

Upiti:

- Određivanje broja elemenata strukture podataka tj. njene veličine
- Određivanje da li se dati element nalazi u nekoj strukturi podataka

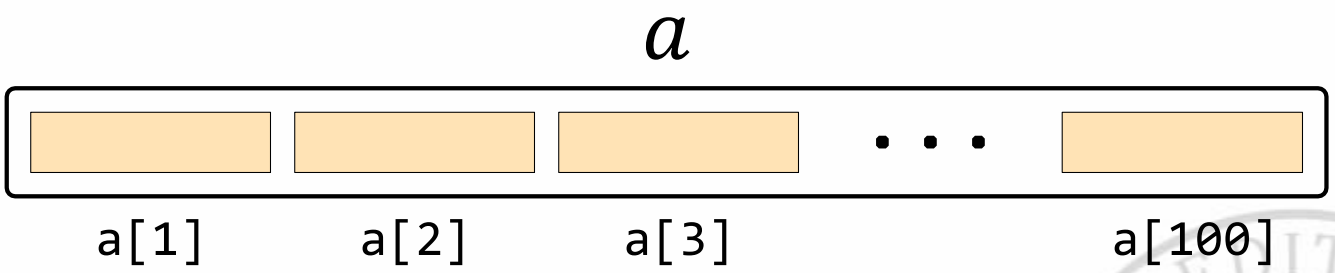
Modifikujuće operacije:

- Konstruisanje nove, prazne strukture podataka
- Dodavanje novih elemenata u postojeću strukturu podataka
- Uklanjanje starih elemenata iz kolekcije elemenata postojeće strukture podataka
- Preuređivanje elemenata strukture podataka u nekom redosledu

Osnovne strukture podataka

- Nizovi
- Matrice
- Liste
- Stekovi
- Redovi

Nizovi

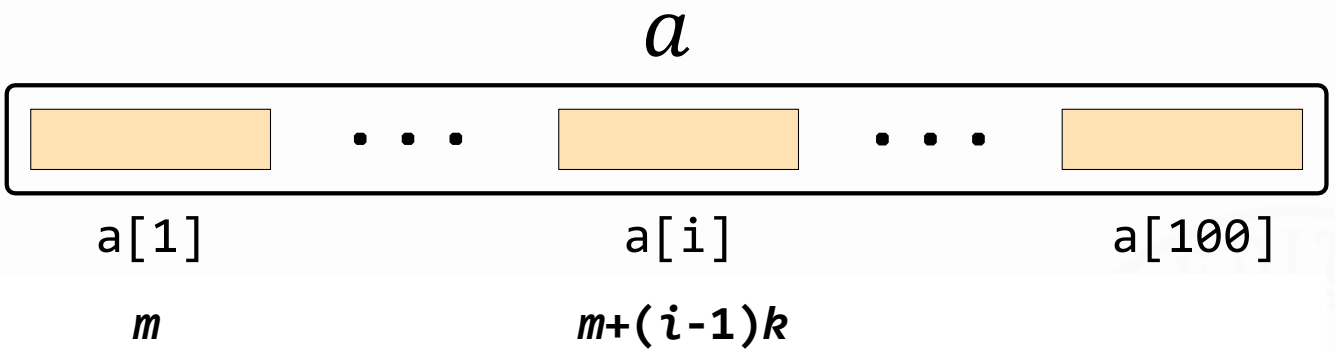


Osnovne osobine niza:

- Svi elementi moraju biti istog tipa
- Dužina niza mora biti unapred zadata i ne može se menjati

Smeštaj niza u memoriji

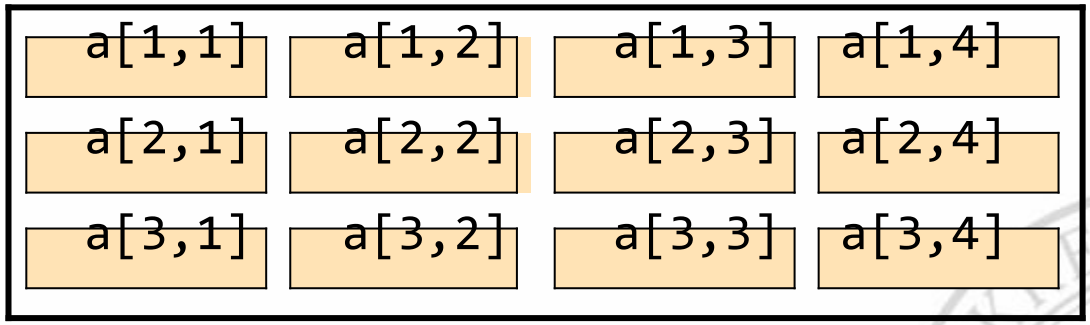
- Elementi niza zauzimaju neprekidni niz susednih memorijskih lokacija
- Lokacija elemenata niza (veličina jednog elementa je *k* bajtova):



- Vreme pristupa svakom elementu niza je konstantno
- U algoritmima se operacija čitanja ili upisivanja nekog elementa niza smatra jediničnom instrukcijom

Matrice

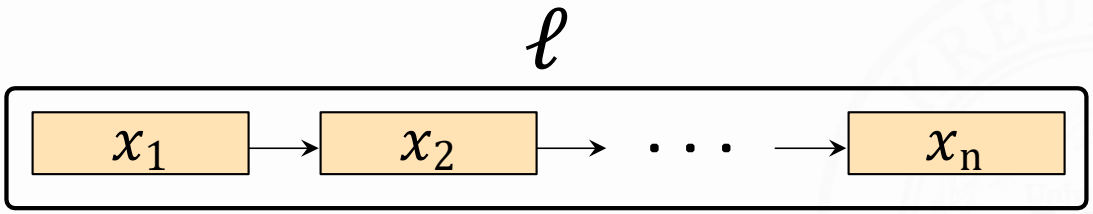
a



- Dvodimenzionalni niz
- Veličina matrice – broj vrsta i broj kolona

Liste

- Niz elemenata: svaki element liste (osim poslednjeg) ima tačno jednog sledbenika u nizu
- Svaki element sadrži pokazivač na sledeći element u listi
- Za razliku od nizova dužina liste nije unapred određena



Liste

- Karakteristični pojmovi
  - Veličina (dužina) liste – broj elemenata liste
  - Sledbenik i prethodnik – dva susedna elementa liste
  - Glava liste – prvi elemenat liste
  - Rep liste – poslednji elemenat liste
  - Prazna lista – lista bez elemenata

Liste

- Operacije
  - Konstruisanje nove (prazne) liste
  - Dodavanje elementa u listu
  - Uklanjanje elementa iz liste
  - Pretraga liste

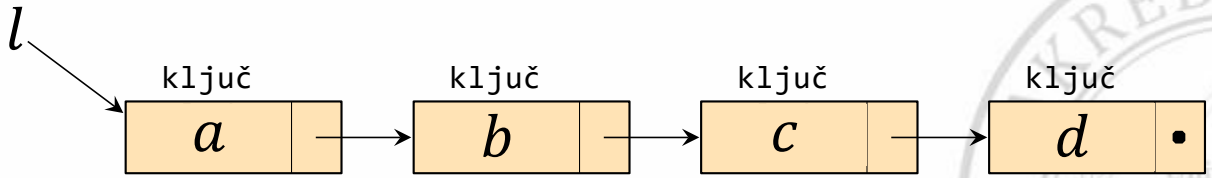
Liste

- Dužina liste se dinamički menja (povećava se i smanjuje)
- Lakoća pristupa pojedinim elementima liste nije ista
- Uvek lak pristup prvom elementu liste, a dodatno i
  - poslednjem elementu
  - „aktuelnom" elementu
  - ...

Implementacija liste

- Primer: lista $\ell = \langle a, b, c, d \rangle$
- Jednostruko povezana lista:

  - Element liste → objekat sa 2 polja → čvor liste

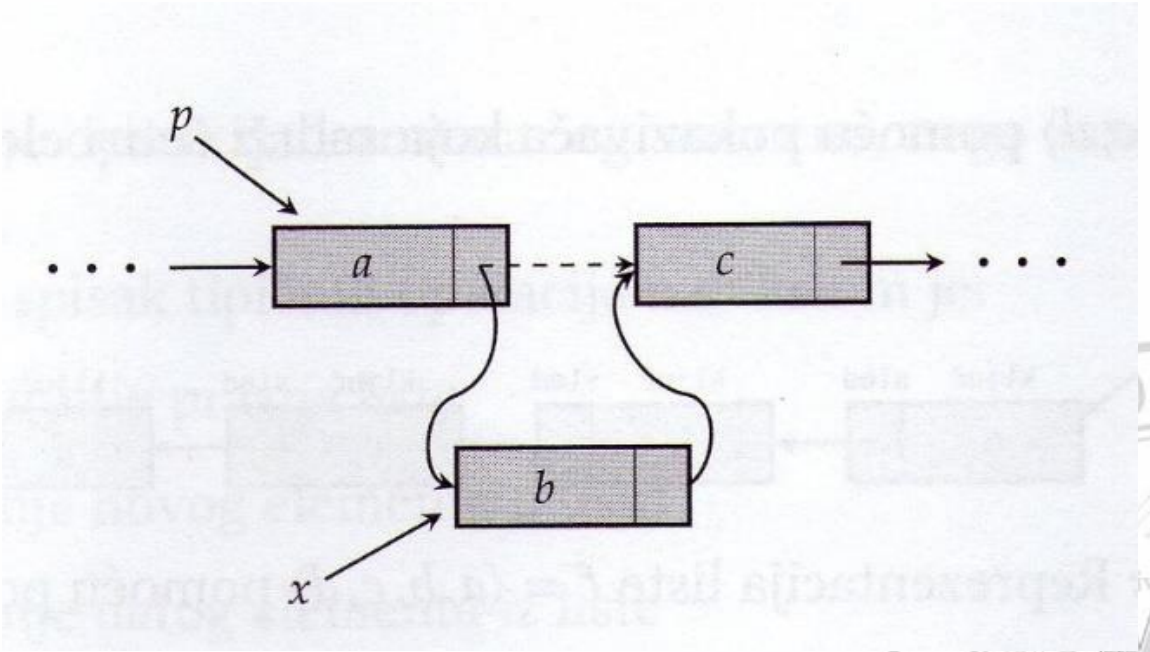


Čvor liste = podatak elementa liste + pokazivač na sledeći čvor
Čvor liste = ključ (sadržaj elementa) + sled (sledbenik elementa)
/– spoljašnji pokazivač (nije deo liste i pokazuje na prvi čvor)

Algoritam 1 - Konstruisanje prazne liste

```
// Ulaz: lista l
// Izlaz: prazna lista l
algorithm list-make (l)

   l = null;

   return l;
```

Dodavanje novog elementa u listu

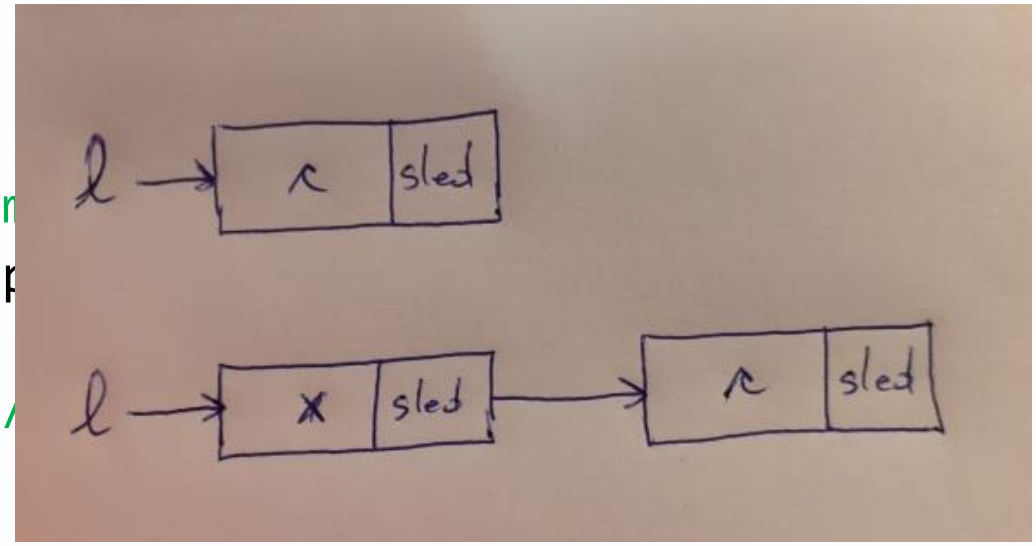


Algoritam 2 - Dodavanje novog elementa u listu

```
// Ulaz: novi čvor x, čvor p u listi l
// Izlaz: lista l sa čvorom x iza čvora p
algorithm list-insert (x, p, l)

   if (p == null) then     // novi čvor dodati na početak
       x.sled = l;
       l = x;
   else                    // novi čvor dodati iza čvora p
       x.sled = p.sled;
       p.sled = x;

   return l;
```

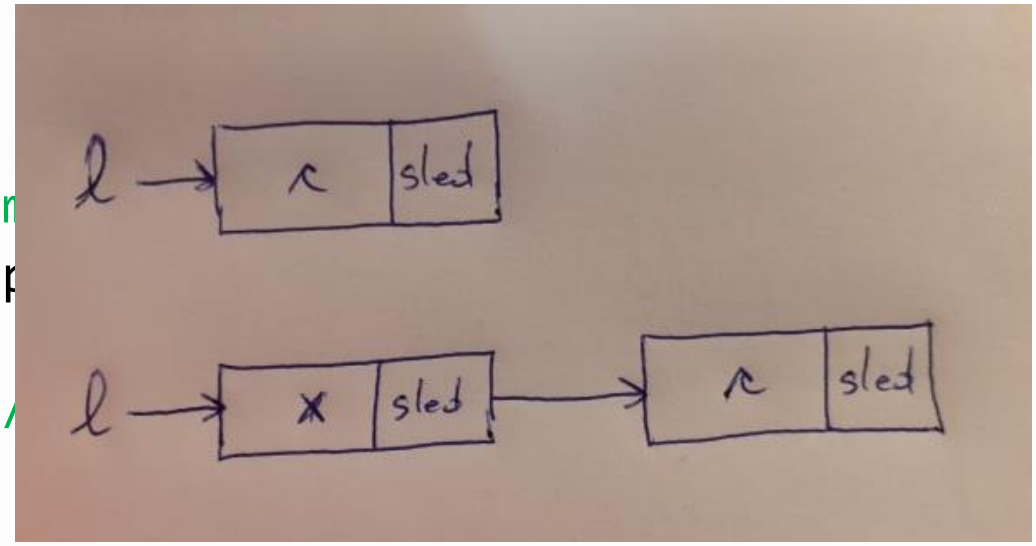
Algoritam 2 - Dodavanje novog elementa u listu

```
// Ulaz: novi čvor x, čvor
// Izlaz: lista l sa čvoron
algorithm list-insert (x, p

   if (p == null) then     /
       x.sled = l;
       l = x;
   else                        // novi čvor dodati iza čvora p
       x.sled = p.sled;
       p.sled = x;

   return l;
```

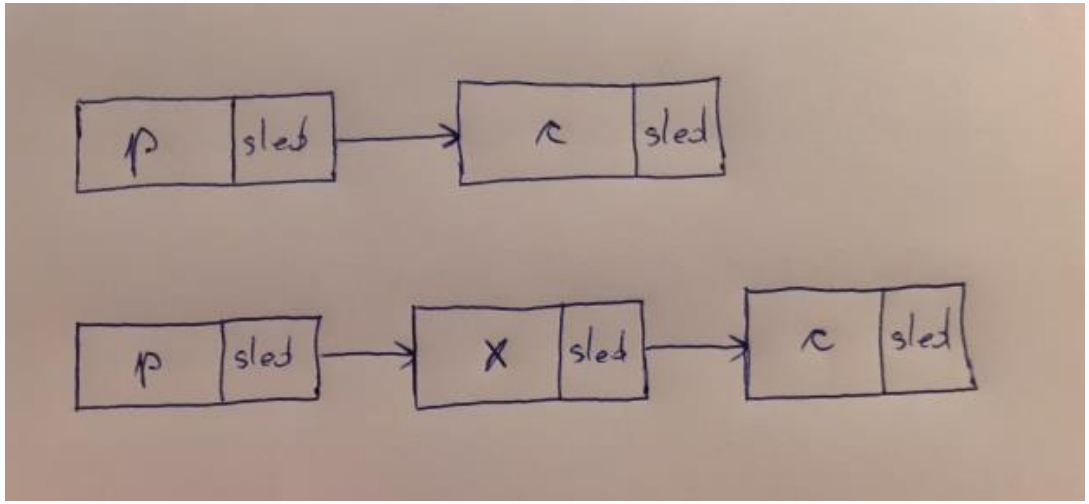


Algoritam 2 - Dodavanje novog elementa u listu

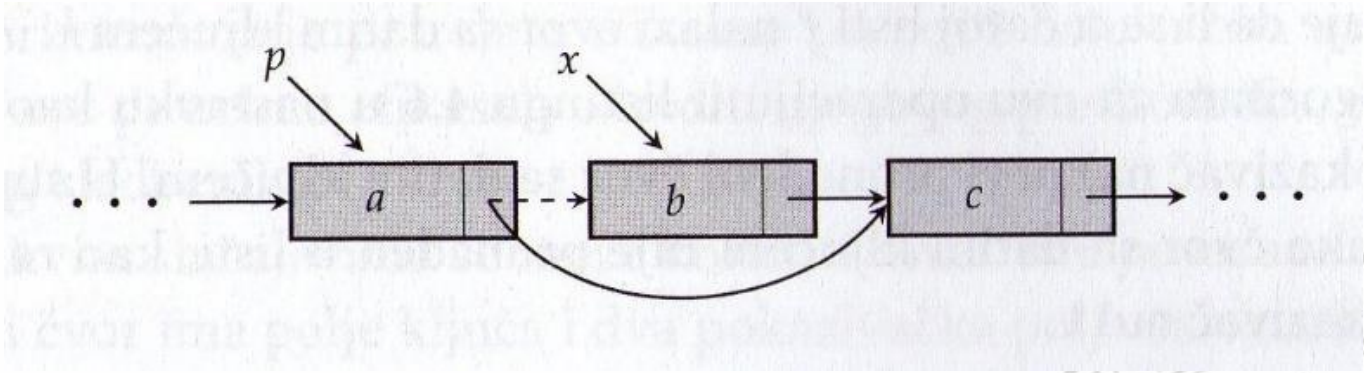
```
// Ulaz: novi čvor x, čvor
// Izlaz: lista l sa čvoro
algorithm list-insert (x, p

   if (p == null) then
        x.sled = l;
        l = x;
   else
        x.sled = p.sled;
        p.sled = x;

   return l;
```




```
                                    // novi čvor dodati iza čvora p
```



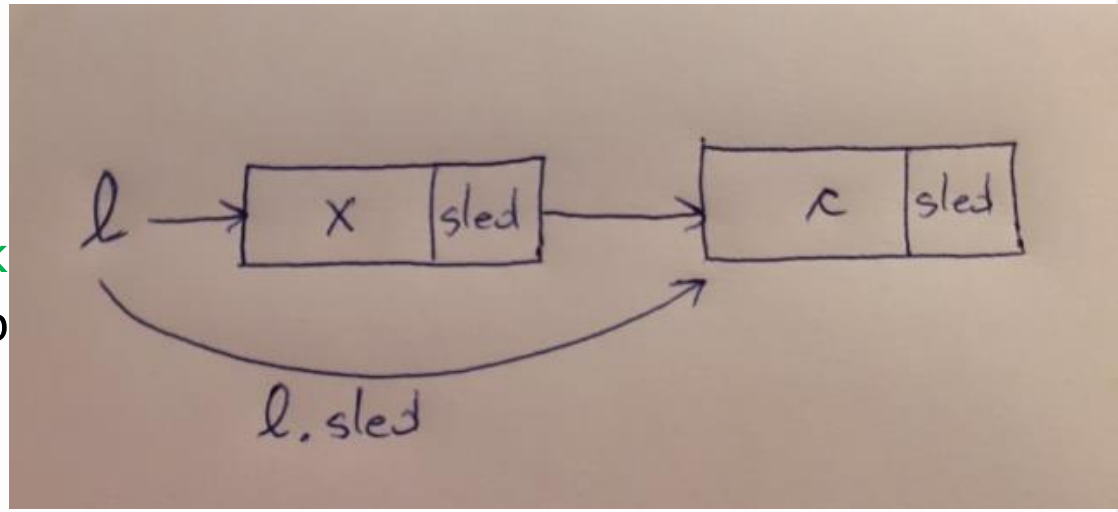
Uklanjanje elementa iz liste



Algoritam 3 - Uklanjanje elementa iz liste

```
// Ulaz: čvor p u listi l iza koga treba ukloniti čvor
// Izlaz: lista l bez uklonjenog čvora
algorithm list-delete (p, l)

   if (p == null) then    // ukloniti čvor na početku
       l = l.sled;
   else                   // ukloniti čvor iza čvora p
       p.sled = p.sled.sled;

   return l;
```

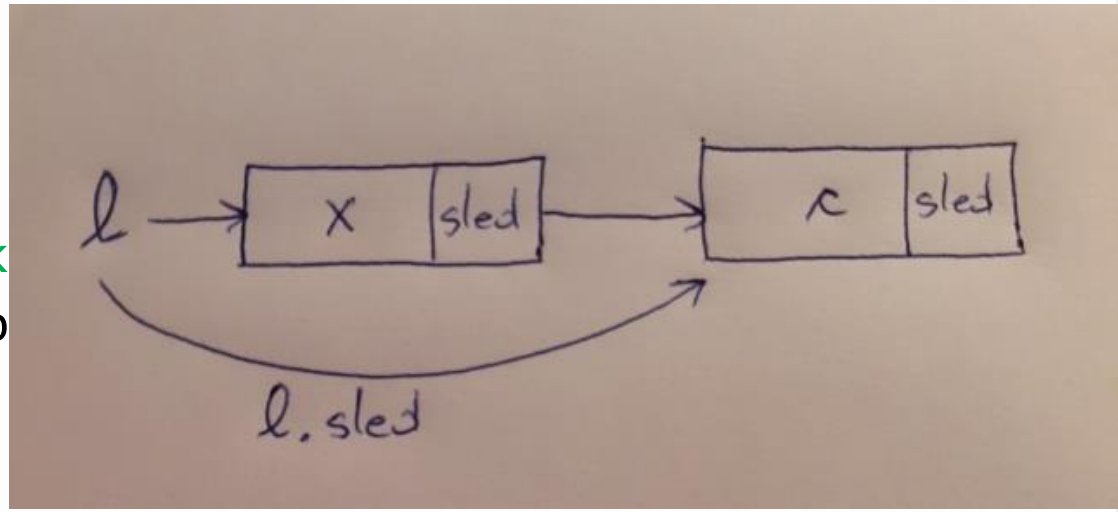
Algoritam 3 - Uklanjanje elementa iz liste

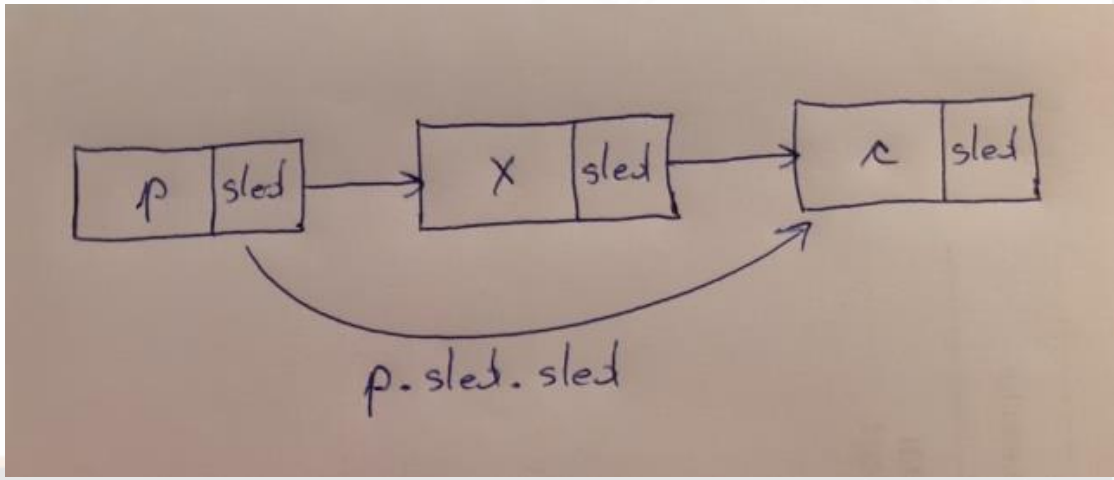



```
// Ulaz: čvor p u listi
// Izlaz: lista l bez uk
algorithm list-delete (p

    if (p == null) then
        l = l.sled;
    else                          // ukloniti čvor iza čvora p
        p.sled = p.sled.sled;

    return l;
```

Algoritam 3 - Uklanjanje elementa iz liste

```
// Ulaz: čvor p u listi
// Izlaz: lista l bez uk
algorithm list-delete (p

    if (p == null) then
        l = l.sled;
    else                          // ukloniti čvor iza čvora p
        p.sled = p.sled.sled;

    return l;
```



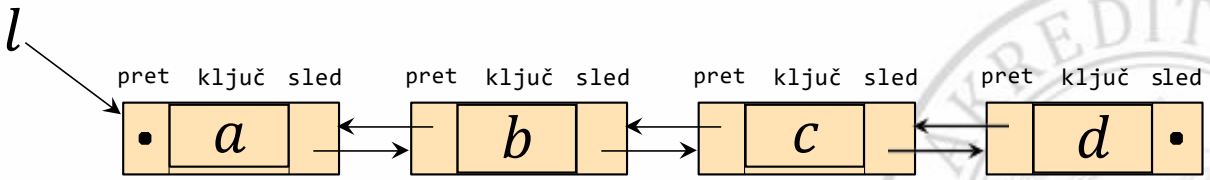


Algoritam 4 - Pretraga liste

```
// Ulaz: lista l, ključ k
// Izlaz: čvor x u listi l sa ključem k ili vrijednost
null
algorithm list-search (l, k)

   x = l;
   while ((x != null) && (x.ključ != k)) do
            x = x.sled;

   return x;
```

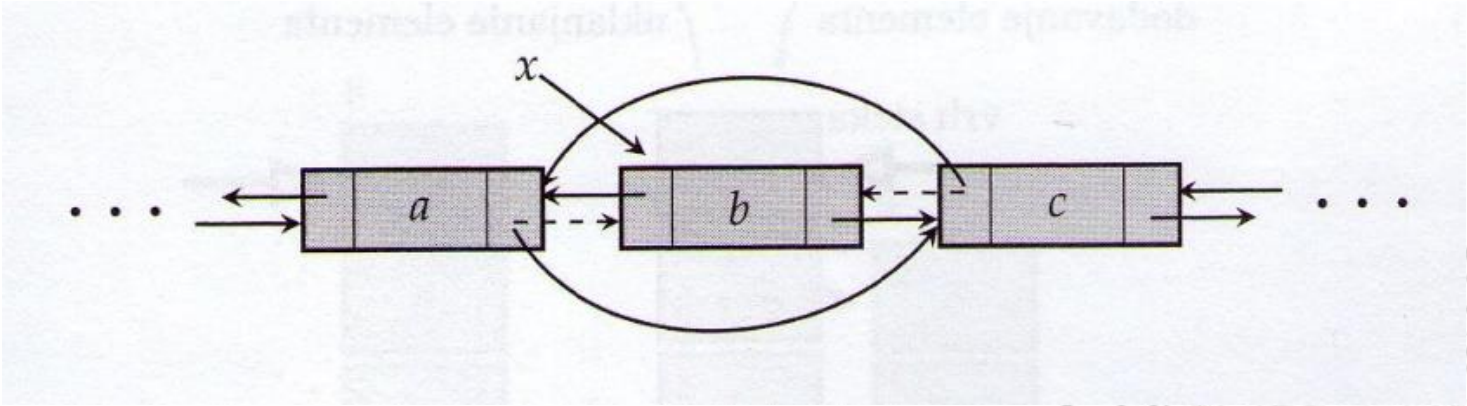
Implementacija dvostruko povezane liste

- Primer: lista $\ell = \langle a, b, c, d \rangle$
- Dvostruko povezana lista:



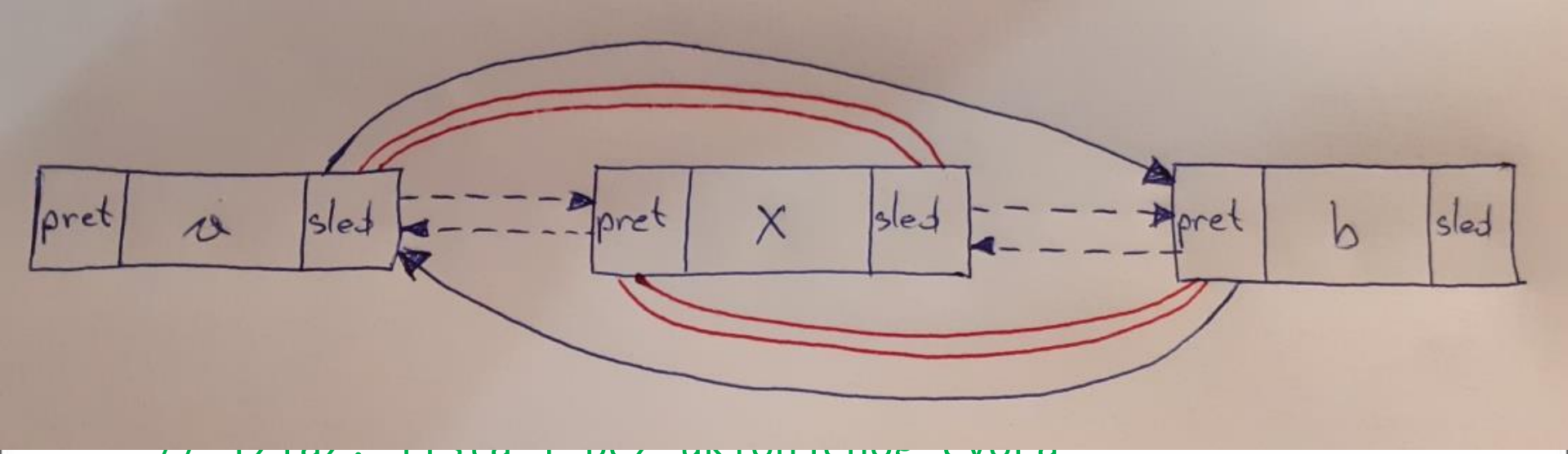
/– spoljašnji pokazivač (nije deo liste i pokazuje na prvi čvor)

Uklanjanje elementa iz dvostruko povezane liste



Algoritam 5 - Uklanjanje elementa iz dvostruko povezane liste

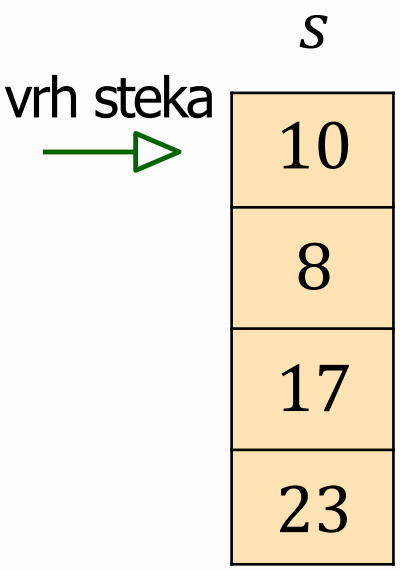
```
// Ulaz: čvor x koji treba ukloniti iz liste l
// Izlaz: lista l bez uklonjenog čvora
algorithm dlist-delete (x, l)

    if (x.pret != null) then    // x nije prvi čvor
        x.pret.sled = x.sled;
    else
        l = x.sled;
    if (x.sled != null) then    // x nije poslednji
    čvor
        x.sled.pret = x.pret;

    return l;
```




```
// izlaz: lista l bez uklonjenog čvora
algorithm dlist-delete (x, l)

    if (x.pret != null) then     // x nije prvi čvor
        x.pret.sled = x.sled;
    else
        l = x.sled;
    if (x.sled != null) then     // x nije poslednji
    čvor
        x.sled.pret = x.pret;

    return l;
```

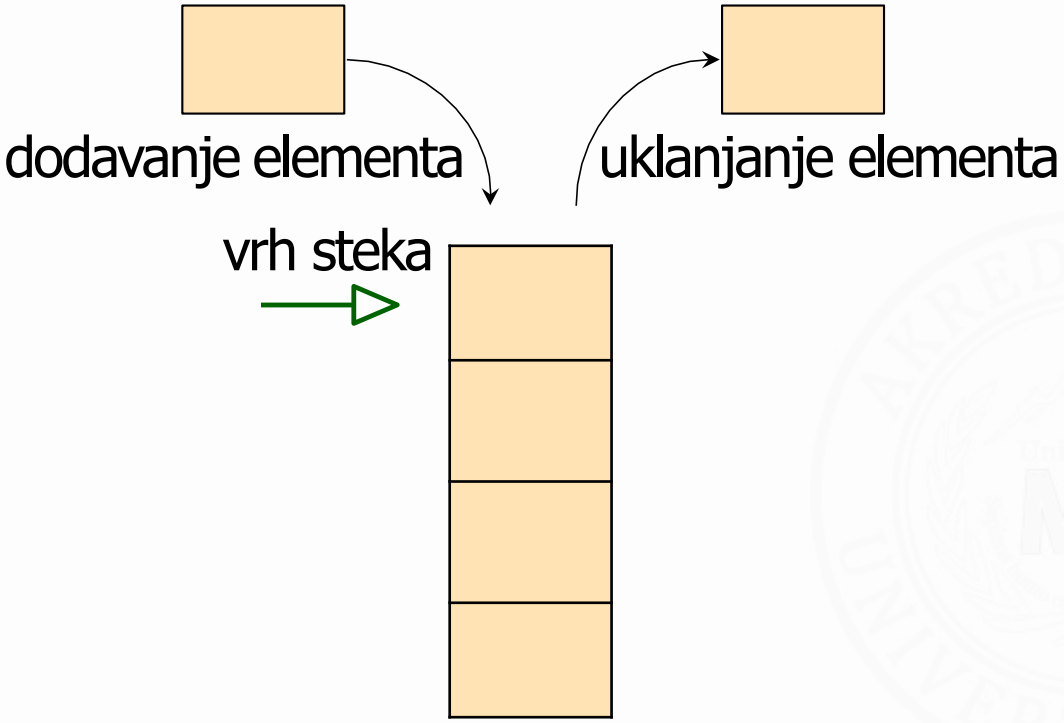
Stekovi

- Linearna struktura – niz elemenata
- Elementi steka su „poredani jedan na drugi”
- Samo je vrh steka pristupačan
- Primer:



Stekovi

- Rad sa stekom je moguć samo preko vrha steka
- Dodavanje i uklanjanje elemenata:



Stekovi

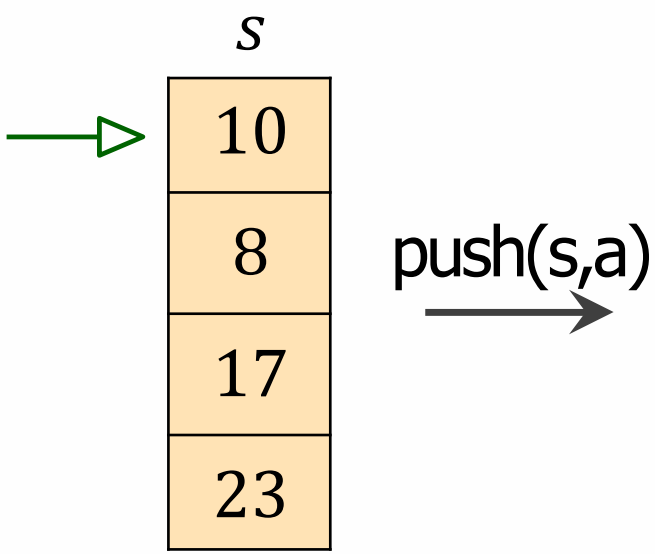
- Veličina steka se dinamički menja (povećava se i smanjuje)
- Analogija: stog sena, poslužavnici za hranu, ražnjići
- LIFO (*last-in, first-out*): poslednji element koji je dodat na stek je prvi koji se uklanja sa steka

Stekovi

- Operacije nad stekom
  - Konstruisanje novog (praznog) steka (make)
  - Dodavanje elementa na stek (push)
  - Uklanjanje elementa iz steka (pop)
  - Vrednost elementa na vrhu steka (peek)

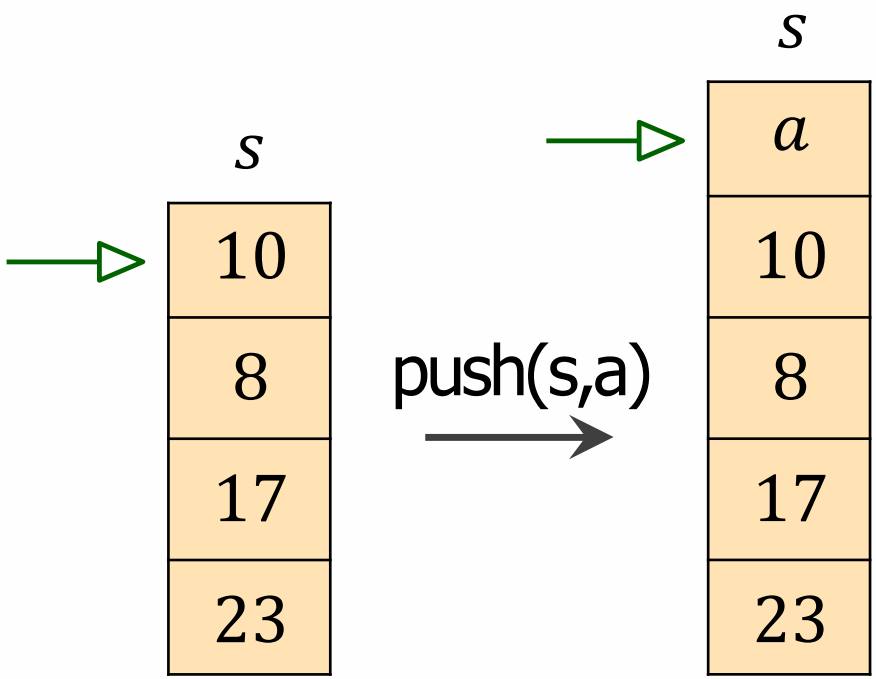
Stekovi

- Dodavanje elementa na stek (push)



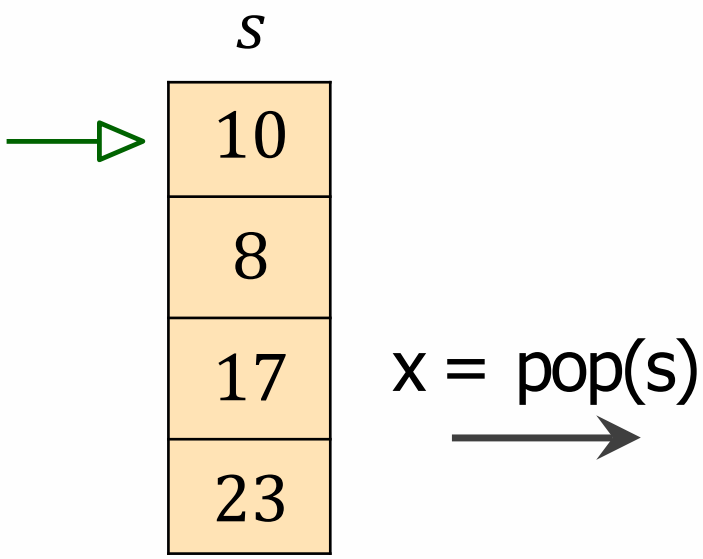
Stekovi

- Dodavanje elementa na stek (push)



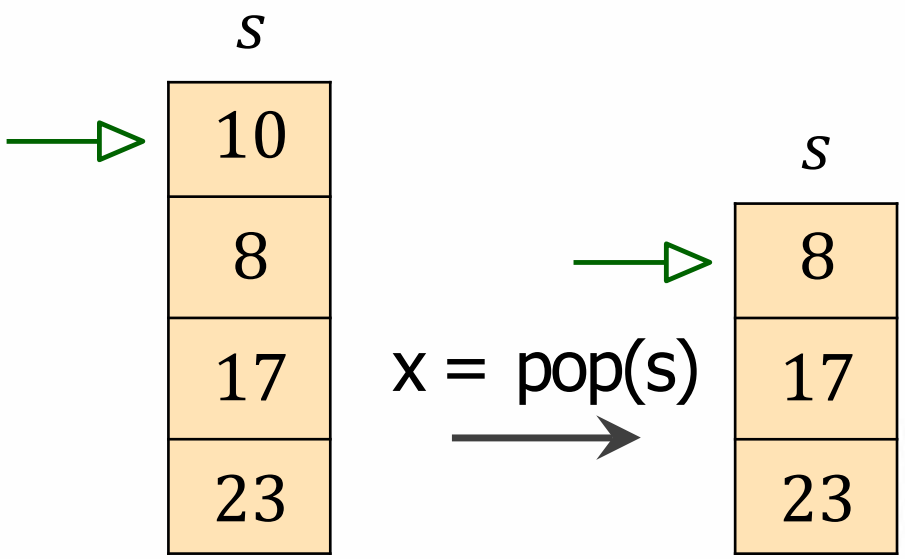
Stekovi

- Uklanjanje elementa iz steka (pop)



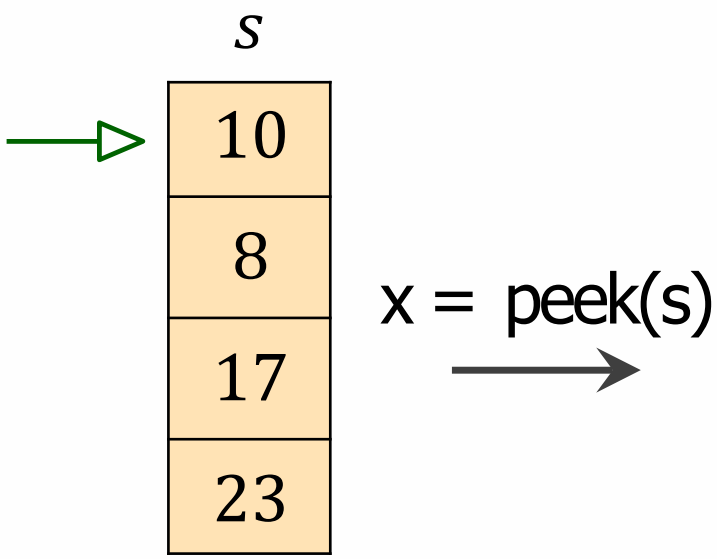
Stekovi

- Uklanjanje elementa iz steka (pop)



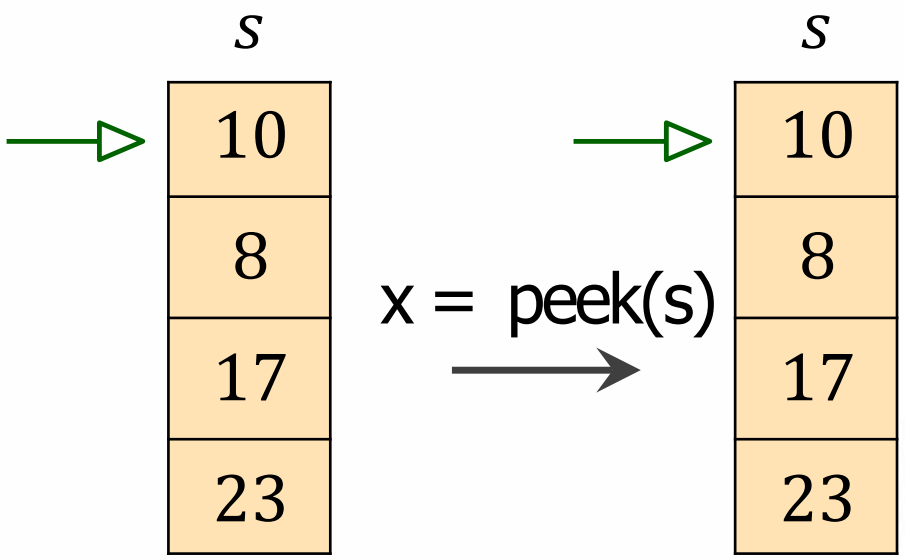
Stekovi

- Vrednost elementa na vrhu steka (peek)



Stekovi

- Vrijednost elementa na vrhu steka (peek)



Stekovi

- Primene stekova
  - Operacija „undo” u programima
  - Nalaženje izlaza iz lavirinta u igrama
  - Izračunavanje izraza
  - …

Implementacija steka

- Slično implementaciji liste
- Operacije push, pop i peek se primenjuju na vrh steka (“početak liste”)

Algoritam 1 – Konstruisanje praznog steka

```
// Ulaz: stek s
// Izlaz: prazan stek s
algorithm stack-make (s)

   s.v = 0;

   return;
```

v – promenljiva čiji sadržaj ukazuje na vrh steka (na poziciju elementa niza koji je poslednji dodat na stek)

Algoritam 2 – Ispitivanje da li je stek prazan

```
// Ulaz: stek s
// Izlaz: tačno ako je s prazan, inače
netačno
algorithm stack-empty (s)

   if (s.v == 0)
       return true;
   else
        return false;
```

Algoritam 3 – Ispitivanje da li je stek pun

```
// Ulaz: stek s
// Izlaz: tačno ako je s pun, inače netačno
algorithm stack-full (s)

   if (s.v == n)
       return true;
   else
        return false;
```

Algoritam 4 – Dodavanje novog elementa na stek

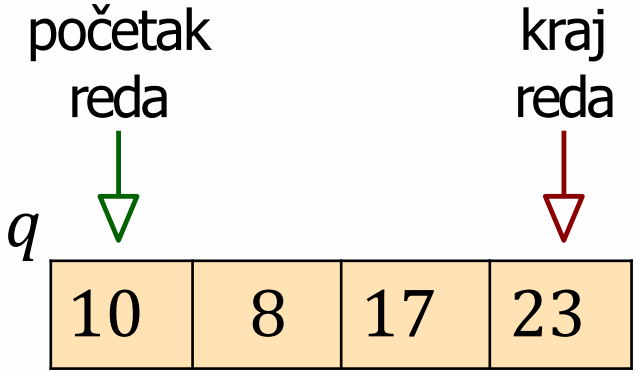
```
// Ulaz: novi element x, stek s
// Izlaz: stek s sa elementom x dodatim na
vrh
algorithm push (x, s)

   // Pretpostavlja se da stek nije pun
   s.v = s.v + 1;
   s.a[s.v] = x;

   return;


   a – niz preko kojeg predstavljamo stek
```

Algoritam 5 – Uklanjanje elementa sa steka

```
// Ulaz: stek s
// Izlaz: element x uklonjen sa vrha steka s
algorithm pop (s)

   // Pretpostavlja se da stek nije prazan
   x = s.a[s.v];
   s.v = s.v - 1;

   return x;
```

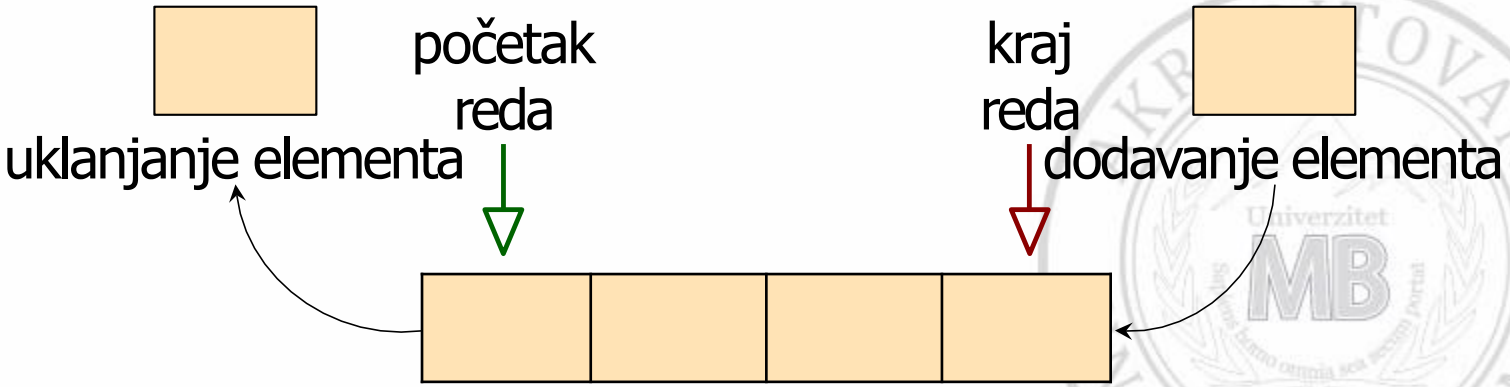
Redovi

- Red (za čekanje): niz elemenata kod koga se razlikuje samo početak i kraj
- Primer:



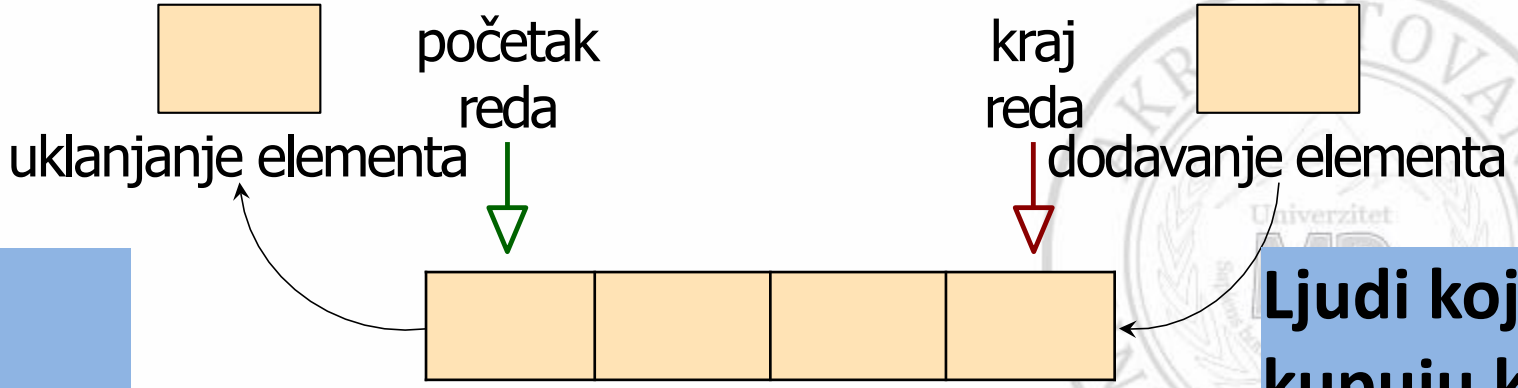
Redovi

- Rad sa redom za čekanje je moguć samo preko početka i kraja reda
- Dodavanje i uklanjanje elemenata:



Redovi

- Rad sa redom za čekanje je moguć samo preko početka i kraja reda
- Dodavanje i uklanjanje elemenata:



Bioskop

Ljudi koji kupuju karte

Redovi

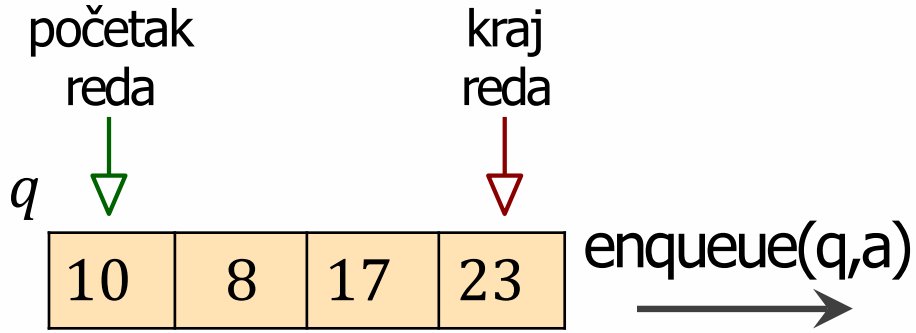
- Veličina reda se dinamički mienja (povećava se i smanjuje)
- Analogija: ljudi koji čekaju u nekom redu
- FIFO (*first-in, first-out*): prvi element koji je dodat u red je prvi koji se uklanja iz reda

Redovi

- Operacije nad redom
  - Konstruisanje novog (praznog) reda (make)
  - Dodavanje elementa na kraj reda (enqueue)
  - Uklanjanje elementa sa početka reda (dequeue)
  - …

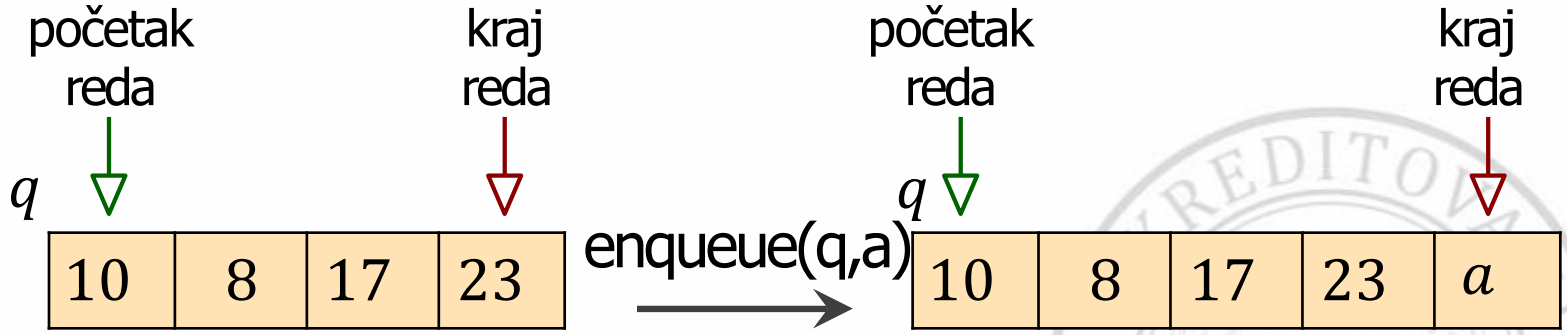
Redovi

- Dodavanje elementa na kraj reda (enqueue)



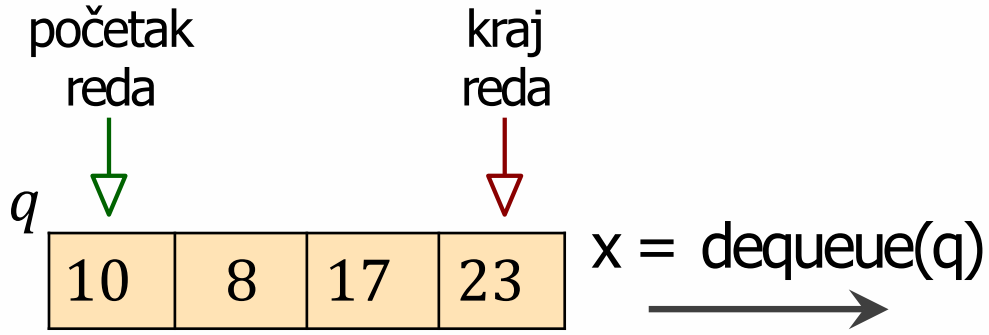
Redovi

- Dodavanje elementa na kraj reda (enqueue)



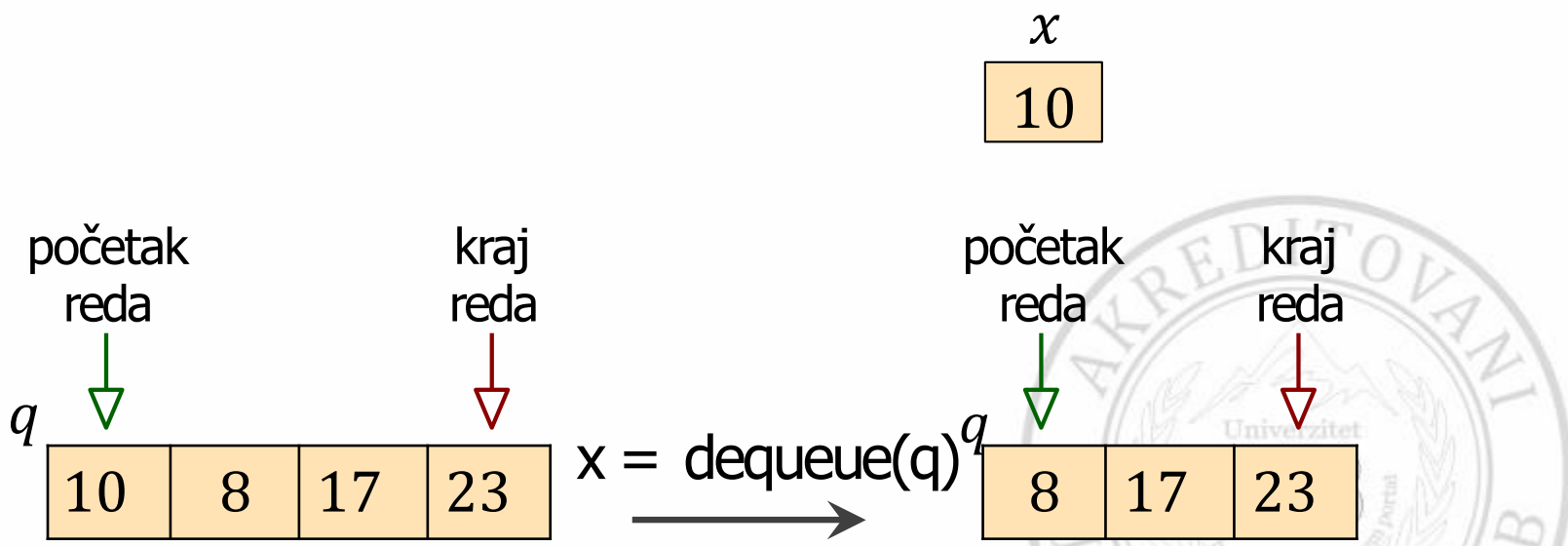
Redovi

- Uklanjanje elementa sa početka reda (dequeue)



Redovi

- Uklanjanje elementa sa početka reda (dequeue)



Redovi

- Primene redova
  - Red programa za izvršavanje u OS
  - Red paketa u ruterima
  - Simulacije
  - …

Redovi

Implementacija reda za čekanje

- Slično implementaciji liste
- Operacija enqueue se primenjuje na kraj reda (“kraj liste”)
- Operacija dequeue se primenjuje na početak reda (“početak liste”)

Zaključak (1)

- Dinamička alokacija memorije znači da program tokom izvršavanja traži od računara da odvoji deo neiskorišćene memorije da bi u njega smestio promenljive određenog tipa
- Osnovne strukture podataka: nizovi, matrice, liste, stekovi, redovi za čekanje
- Struktura podataka: skup elemenata i skup operacija nad njima

Zaključak (2)

- Operacije nad strukturama podataka – upiti, modifikujuće operacije
- Liste - svaki element liste (osim poslednjeg) ima tačno jednog sledbenika u nizu
- Stekovi - linearna struktura, niz elemenata gde su elementi „poredani jedan na drugi”
- Red (za čekanje) - niz elemenata kod koga se razlikuje samo početak i kraj

Kraj prezentacije

HVALA NA PAŽNJI!