Stog i red

Sadržaj

- Apstrakti tip podataka
- Stog
  - Definicija stoga
  - Implementacija stoga
- Red
  - Definicija reda
  - Implementacija reda

Apstraktni tip podataka (ATP)

- Eng. Abstract data type (ADT)
- Definirana apstraktna struktura podataka i odgovarajuće operacije s podacima
- Definicija na konceptualnoj razini - neovisna o implementaciji
- Struktura podataka (engl. *data structure*)
  - Agregacija jednostavnih i složenih tipova podataka u skup s definiranim relacijama
  - Skup pravila koja ujedinjuju (povezuju) podatke

Apstraktni tip podataka (ATP)

Primjeri struktura podataka (složeni tipovi podataka)	
Niz (array)	Slog (record)
Niz homogenih podataka (bitan je redoslijed podataka)	Kombinacija heterogenih podataka u jedinstvenu strukturu (s istaknutim ključem za identifikaciju)
Primjer. Podaci o prosječnim dnevnim temperaturama za mjesec lipanj: 23, 24,23,...	**Primjer.** Podaci o studentu: Broj indeksa, ime, prezime, ...

Apstraktni tip podataka (ATP)

- Počeci programiranja:
  - Nisu postojali apstraktni tipovi podataka
  - Primjer. kôd za čitanje podataka iz datoteke: piše se kod koji fizički pristupa točno određenoj datoteci i čita podatke iz nje
  - Posljedica. Ponovno pisanje istog koda za svaku novu datoteku
- ADT:
  - Koncept koji je sadržan u svim modernim programskim jezicima
  - Primjer. kôd za čitanje datoteke pohranjen u biblioteci koji se onda može ponovno koristiti za različite datoteke

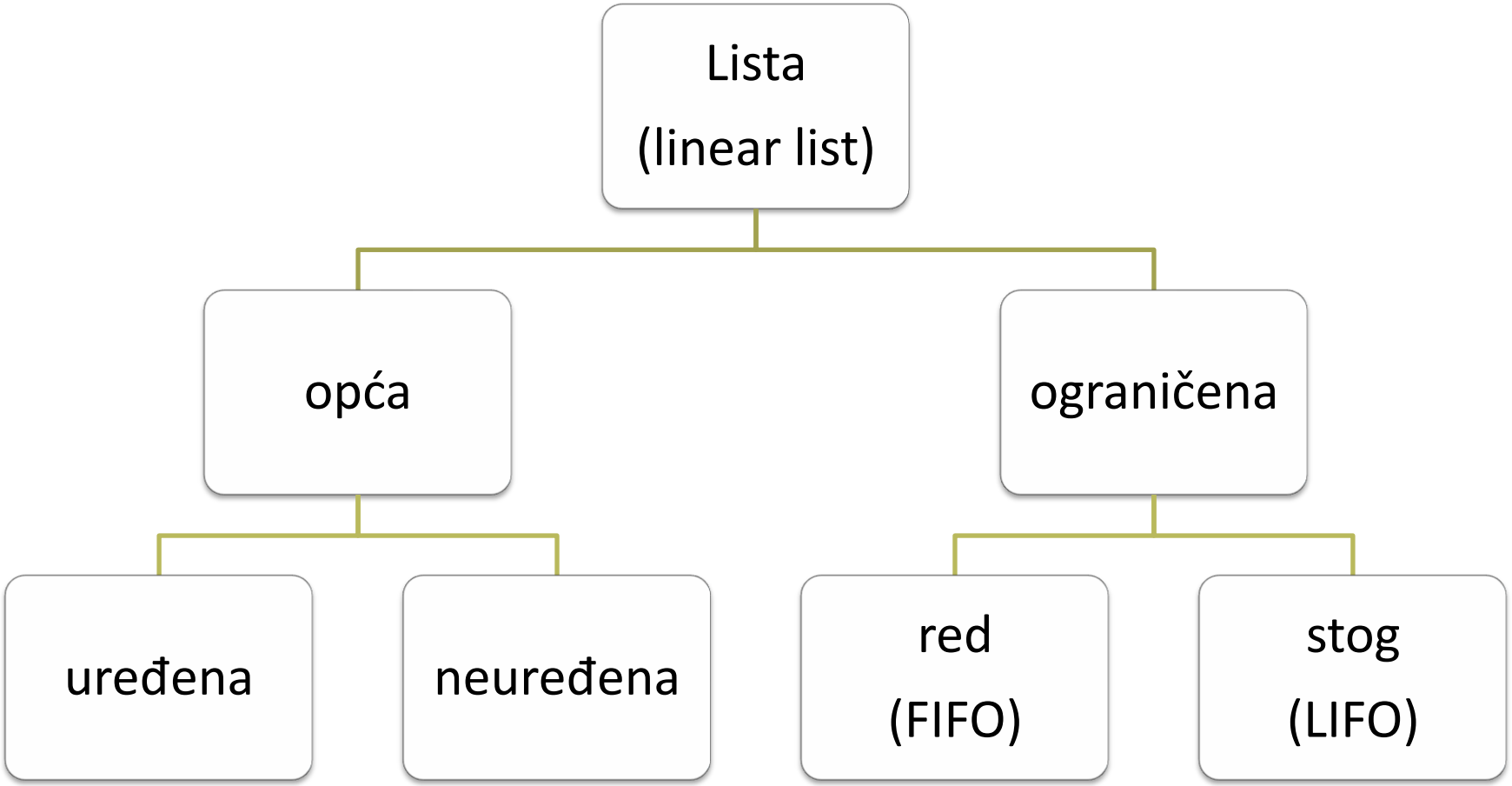
Apstraktni tip podataka (ATP)

- Apstrakcija:
  - generalizacija operacija
  - bez specificiranja implementacije (skrivanje implementacije)
  - zanima nas što se radi s podacima i funkcijama (**what**)
  - ne zanima nas kako se radi (**how**)
  - isti apstraktni tip podataka može se koristiti za različite primjene, na primjer, ADT red se može koristiti:
    - Red čekanja u banci
    - Red čekanja dokumenata za ispis na printeru
    - ...

Apstraktni tip podataka (ATP)

- Primjeri ATP:
  - Lista
  - Stog, red
  - Stablo, binarno stablo
  - ... i druge apstraktne strukture podataka
- **Primjer.** Lista (niz podataka)
  - Više različitih implementacija (polje, povezana lista, datoteka)
  - Lista kao ADT:
    - ne zanima nas kako je implementirana
    - možemo koristiti različite funkcije rad s listama bez da znamo kako su ti podaci pohranjeni

Apstraktni tip podataka (ATP)

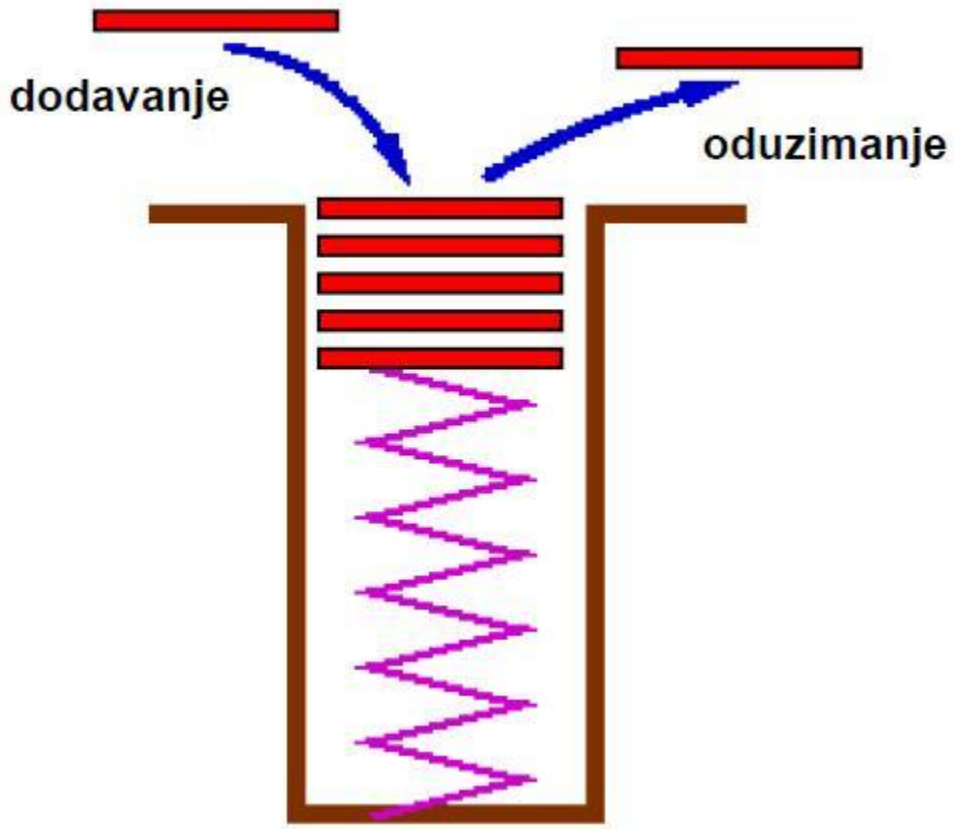


Stog

Stog

- Struktura podataka u kojoj se elementi mogu dodavati i brisati samo na vrhu stoga
- Linearna struktura podataka
- Element koji je zadnji dodan na stog, prvi se ‘uzima’ sa stoga – LIFO struktura (eng. *last in - first out*)
- Primjer:
  - Elementi ulaze u stog zadanim redoslijedom: 7,20,3,5
  - Stog: 5,3,20,7

Stog



Stog

- U stogu je omogućen direktan pristup prvom elementu stoga, a to je element koji je zadnji ušao u stog
- Posljednji element stoga (koji se nalazi na dnu stoga) je element koji je prvi ušao u stog.
- Pristup elementu koji se nalazi u stogu – potrebno je prvo izvaditi elemente koji se nalaze iznad njega

Primjene stoga

- Pozivi funkcija
  - Pozvana funkcija mora znati kako vratiti informaciju, pa se adresa povratka sprema na stog
  - Programi u svom izvršnom obliku sadržavaju skokove na potprograme - nakon čijeg izvršavanja se vraća u glavni program.
    - skokovi su realizirani tako da se na tzv. stog za pozive (engl. call stack) neposredno prije izvršavanja skoka stavlja adresa sljedeće instrukcije
    - Nakon izvršenja pozvanog potprograma, sa stoga se uklanja zadnja pohranjena adresa i na nju se skače
  - Pozivi rekurzivne funkcije
    - postupak namotavanja (winding) i odmotavanja (unwinding)

Primjene stoga

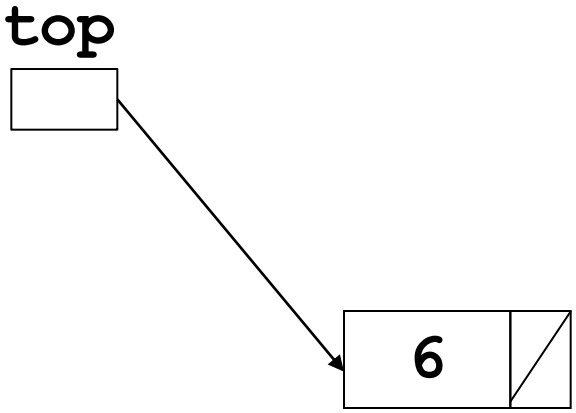
- Za pretvorbe zapisa operatora:
  - infiksni oblik -> postfiksni oblik
  - Stog je vrlo koristan u izradi kalkulatora, gdje služi kod pretvaranja infiks matematičkog zapisa u postfiks matematički zapis, koji se nakon pretvorbe vrlo lako evaluira (izračunava) također koristeći stog
  - U pretvorbi se koristi za privremenu pohranu operatora visokog prioriteta koji se tek nakon dodavanja operatora nižeg prioriteta dodaju u postfiks zapis

Primjene stoga

- Na razini operacijskog sustava (postoji sistemski stog)
- Stog koriste prevoditelji (kompajleri) u procesu raščlanjivanja (parsing)
- Za invertiranje poretka elemenata liste
- …

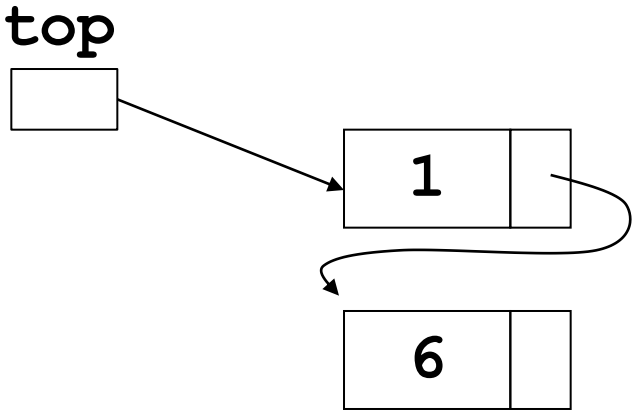
Stog

```
…
push(6);
```

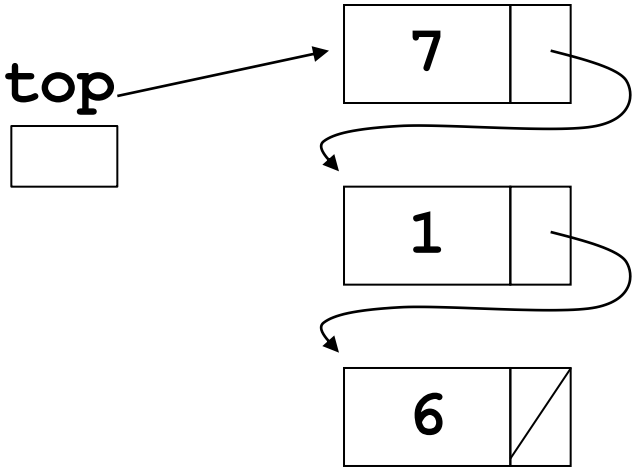


Stog

```
…
push(6);
push(1);
```

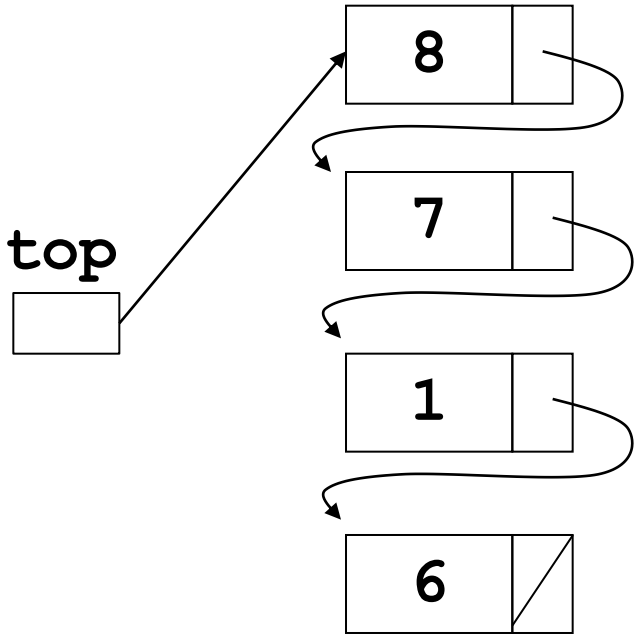


Stog




```
                    …
push(6);
push(1);
push(7);
```

Stog

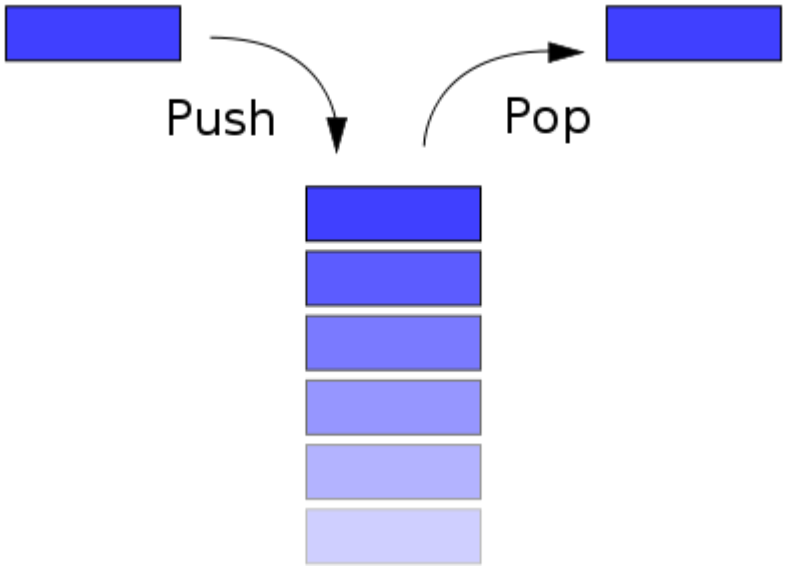



```
push(6);
push(1);
push(7);
push(8);
```

Operacije sa stogom

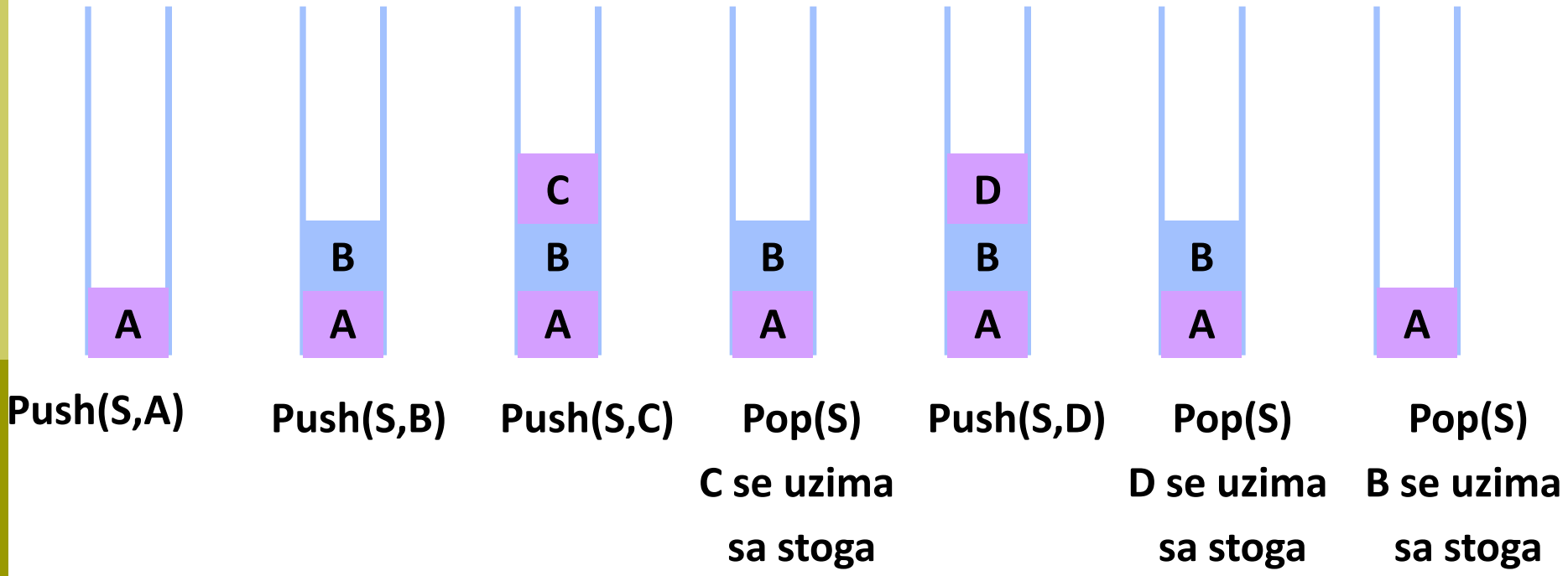
- Osnovne operacije sa stogom:
  - Push
    - Dodaje podatak na stog
    - Podatak se uvijek dodaje na vrh stoga
  - Pop
    - Uzima podatak za obradu - brisanje podatka sa stoga
    - Podatak se uvijek uzima s vrha stoga

Operacije sa stogom

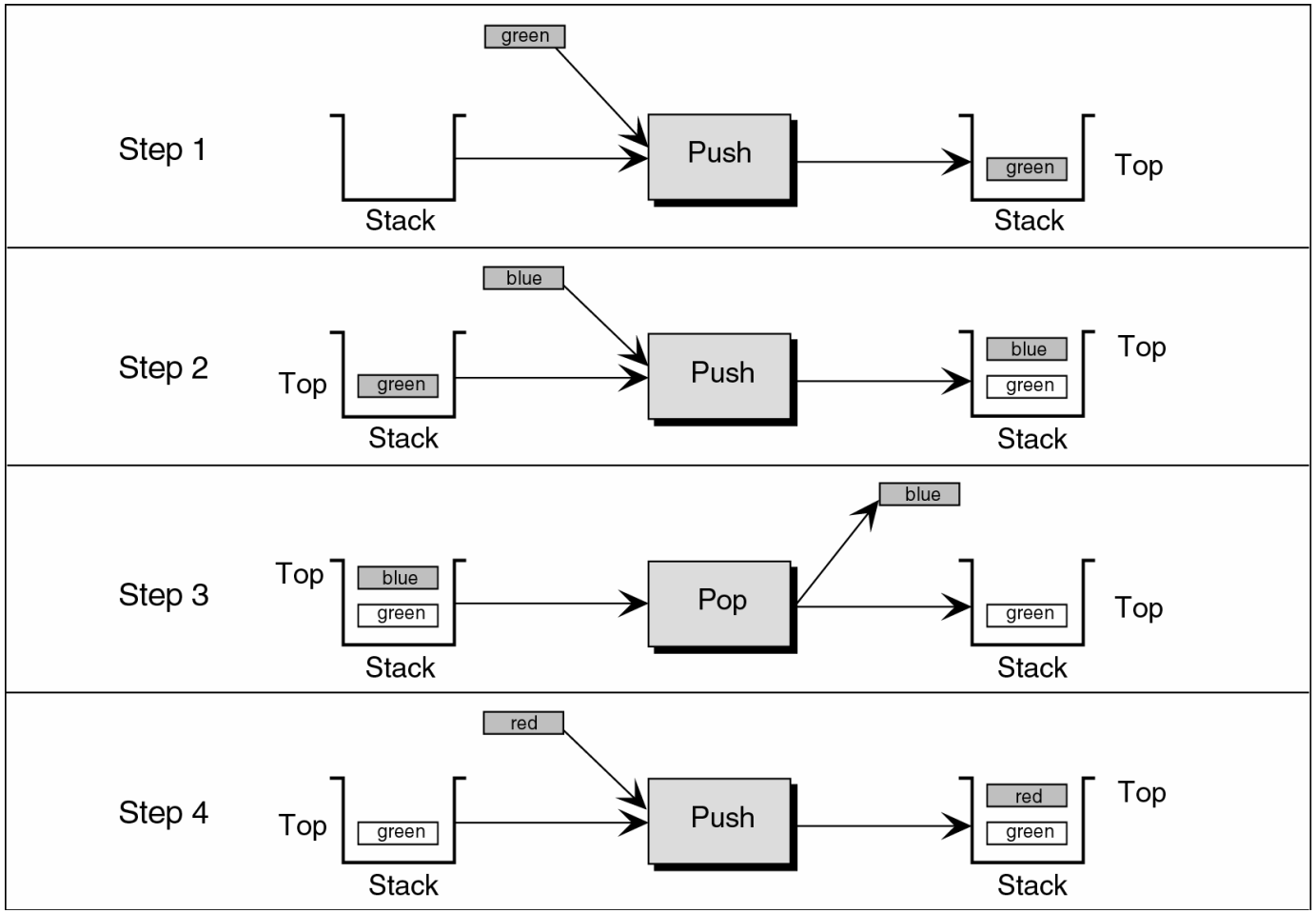


Operacije sa stogom

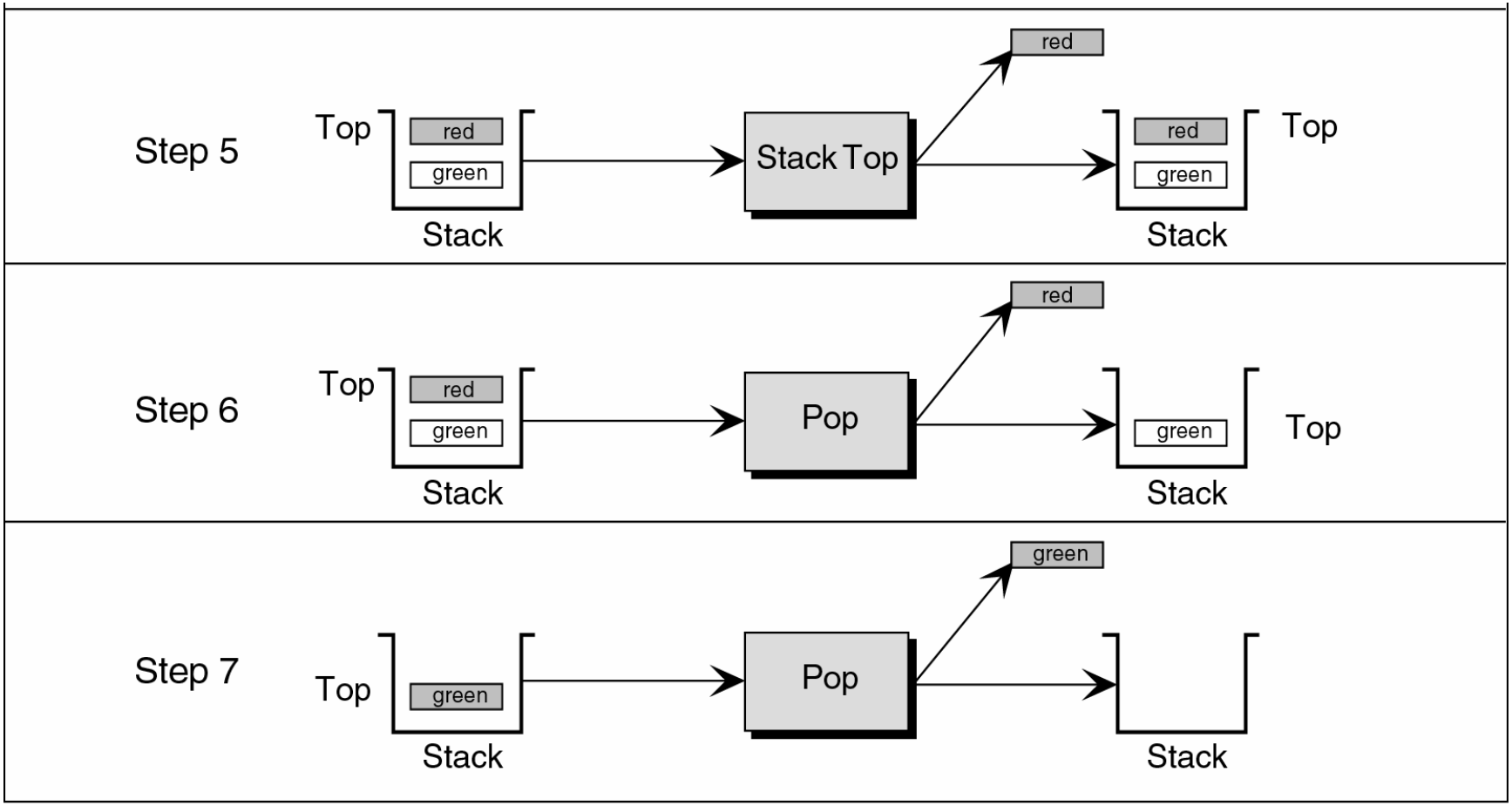
- LIFO – Last In First Out – svojstvo



Operacije sa stogom



Operacije sa stogom



Operacije sa stogom

- Druge bitne funkcije koje se koriste u radu sa stogom
  - **stackTop** (Pristup vrhu stoga) - vraća podatak na vrhu ali ga ne briše sa stoga
  - **Prazan** (isEmpty) – provjera je li stog prazan, vraća true ako je stog prazan, inače false.
  - Obilazak - funkcija za obilazak stoga (i eventualno ispis podataka)
  - **Kreiraj/Uništi stog** (init/delete) **–** funkcije za kreiranje stoga i za brisanje svih elemenata stoga

Pitanje

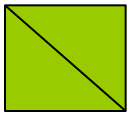
- Na koji način biste implementirali stog u jeziku C++?

Implementacija stoga

- Implementacija stoga
  - Polje
  - Povezana lista
- Implementacija primjenom povezane liste
  - Stog je predstavljen pokazivačem na vrh stoga (**`vrh`**).
  - Čvor liste sadrži podatak i pokazivač na slijedeći element stoga
    - implementiran kao struktura cvor koja je prikazana u prezentaciji za povezane liste

Kreiranje stoga

- Kreira se pokazivač na vrh stoga
- Na početku se vrh stoga inicijalizira na NULL
- Kod stoga je uobičajeno da se pokazivač na početak zove **vrh** (a ne glava kao kod listi)



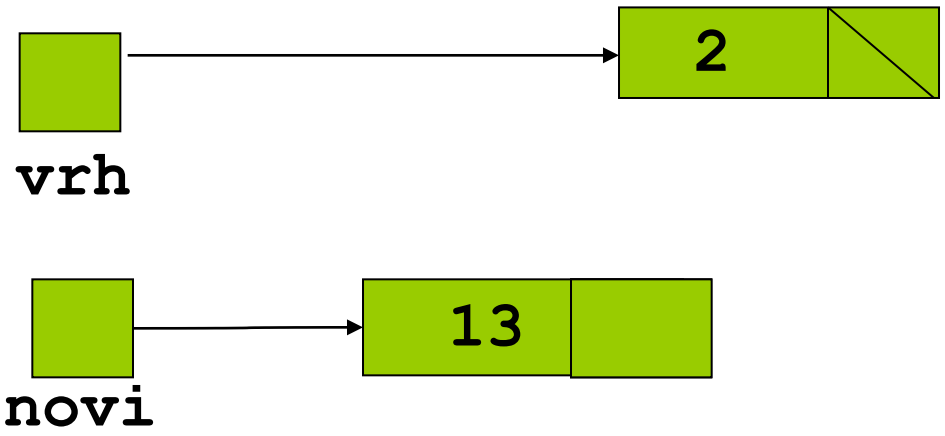
vrh

```
cvor *vrh = NULL;
```

Dodavanje elementa na stog

- Funkcija **`push`**
- Operacija umeće element u stog
  - uvijek na vrh stoga
- Izvodi se kao i umetanje čvora na poziciji glave povezane liste.
- Potrebno je:
  - Alocirati memoriju za novi čvor.
  - Preusmjeriti vezu novog čvora na trenutni čvor koji se nalazi na vrhu stoga.
  - Preusmjeriti pokazivač vrh na novi čvor.

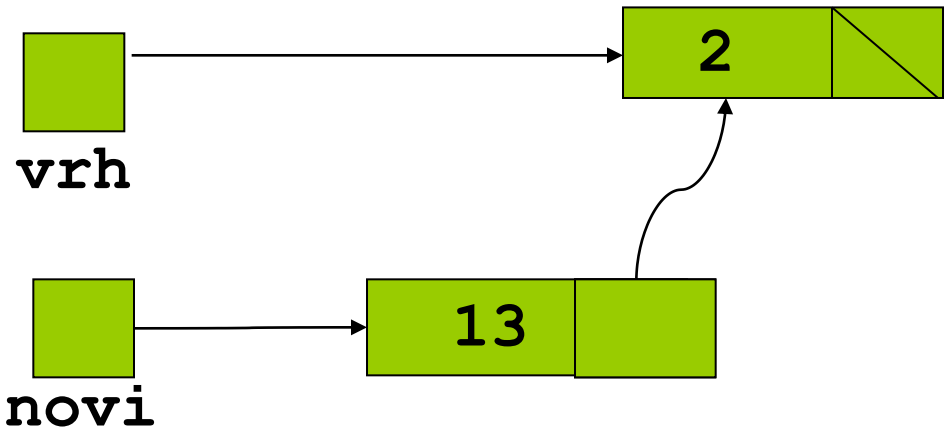
Dodavanje elementa na stog



Kod:

```
novi = new cvor;
novi -> podatak = 13;
```

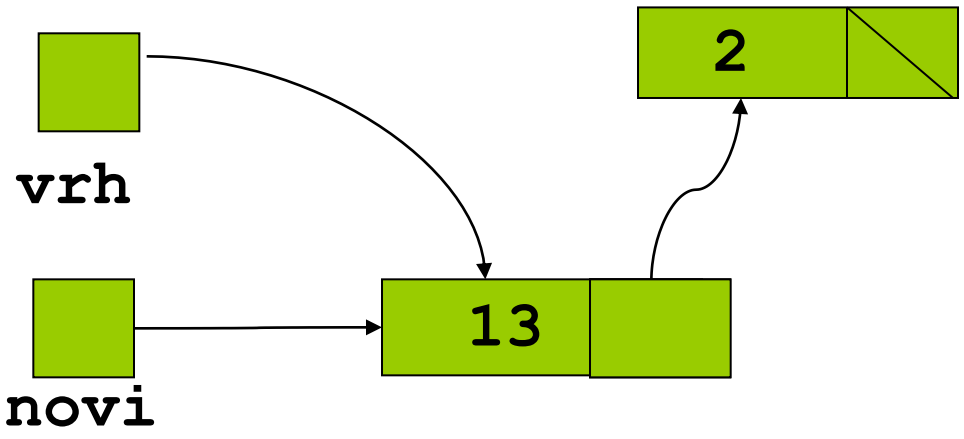
Dodavanje elementa na stog



Kod:

```
novi = new cvor;
novi -> podatak = 13;
novi -> veza = vrh;
```

Dodavanje elementa na stog



Kod:

```
novi = new cvor;
novi -> podatak = 13;
novi -> veza = vrh;
vrh = novi;
```

Dodavanje elementa na stog

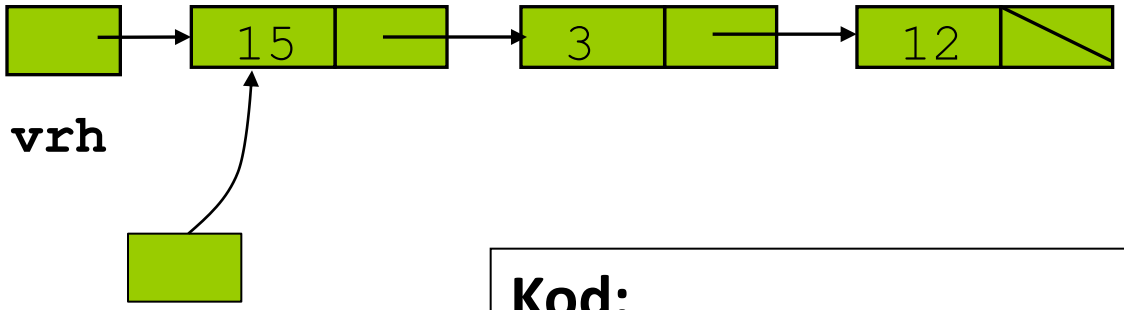
- Funkcija **push** za dodavanje elementa na vrh stoga (elementi tipa int).
- Kao argumenti funkcije navode se pokazivač na vrh stoga (obavezno s referencom jer se mijenja u funkciji!) i podatak koji želimo dodati na vrh

```
void push(cvor* &vrh, int podatak){
  cvor* novi = new cvor;
  novi->podatak = podatak;
  novi->veza = vrh;
  vrh = novi;
}
```

Izlazak elementa iz stoga

- Funkcija **`pop`**
- Operacija ‘uzima’ element s vrha stoga
  - Element izlazi iz stoga, oslobađa se memorija
- Izvodi se kao i brisanje čvora na poziciji glave kod povezane liste
- Potrebno je:
  - Vratiti (izvaditi) podatak sa vrha stoga
  - Preusmjeriti pokazivač vrha na novi vrh stoga
  - Osloboditi memoriju čvora koji je bio vrh stoga

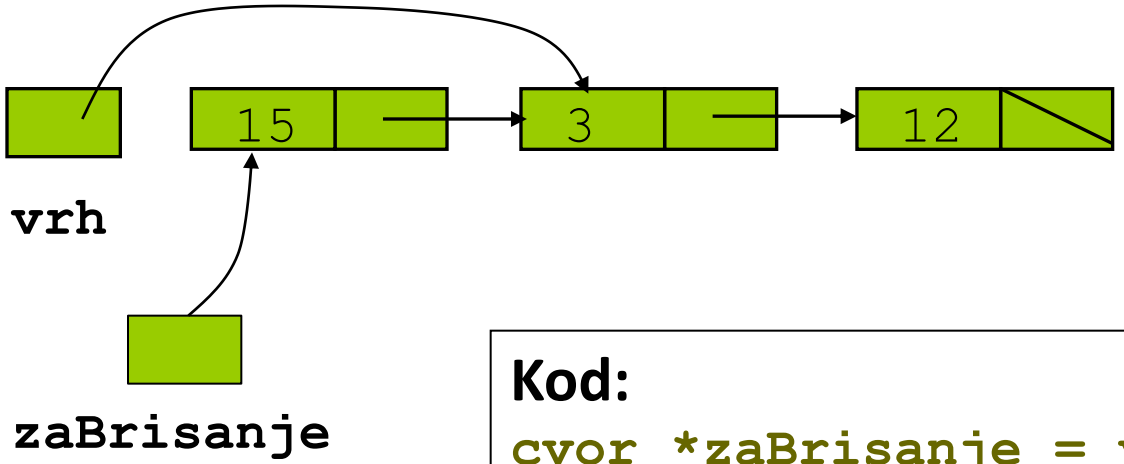
Izlazak elementa iz stoga



Kod:

```
cvor *zaBrisanje = vrh;
podatak = vrh -> podatak;
```

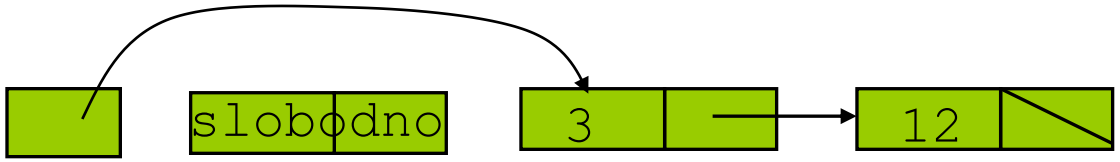
Izlazak elementa iz stoga



Kod:

```
cvor *zaBrisanje = vrh;
podatak = vrh -> podatak;
vrh = vrh -> veza;
```

Izlazak elementa iz stoga



vrh

zaBrisanje

Kod:

```
cvor *zaBrisanje = vrh;
podatak = vrh -> podatak;
vrh = vrh -> veza;
delete zaBrisanje;
```

Izlazak elementa iz stoga

```
cvor* pop(cvor *&vrh){
   cvor* zaBrisanje= NULL;
   if(!stogJePrazan(vrh)){
       zaBrisanje = vrh;
       vrh = vrh -> veza;
   }
   return zaBrisanje;
} //vraćen je pokazivač na 1. element u stogu
//nakon toga je potrebno realizirati oslobađanje
   memorije
```

Operacija Prazan stog

- Ova operacija provjerava da li je stog prazan.

- Kod:

```
bool stogJePrazan(Cvor* vrh){
 return (vrh == NULL);
}
```

Uništavanje stoga

- Funkcija briše sve elemente stoga.

```
void brisiStog(Cvor* & vrh) {
Cvor *zaBrisanje;
while( vrh != NULL ) {
 zaBrisanje = vrh;
 vrh = vrh->veza;
 delete zaBrisanje;
}
}
```

Ispis stoga

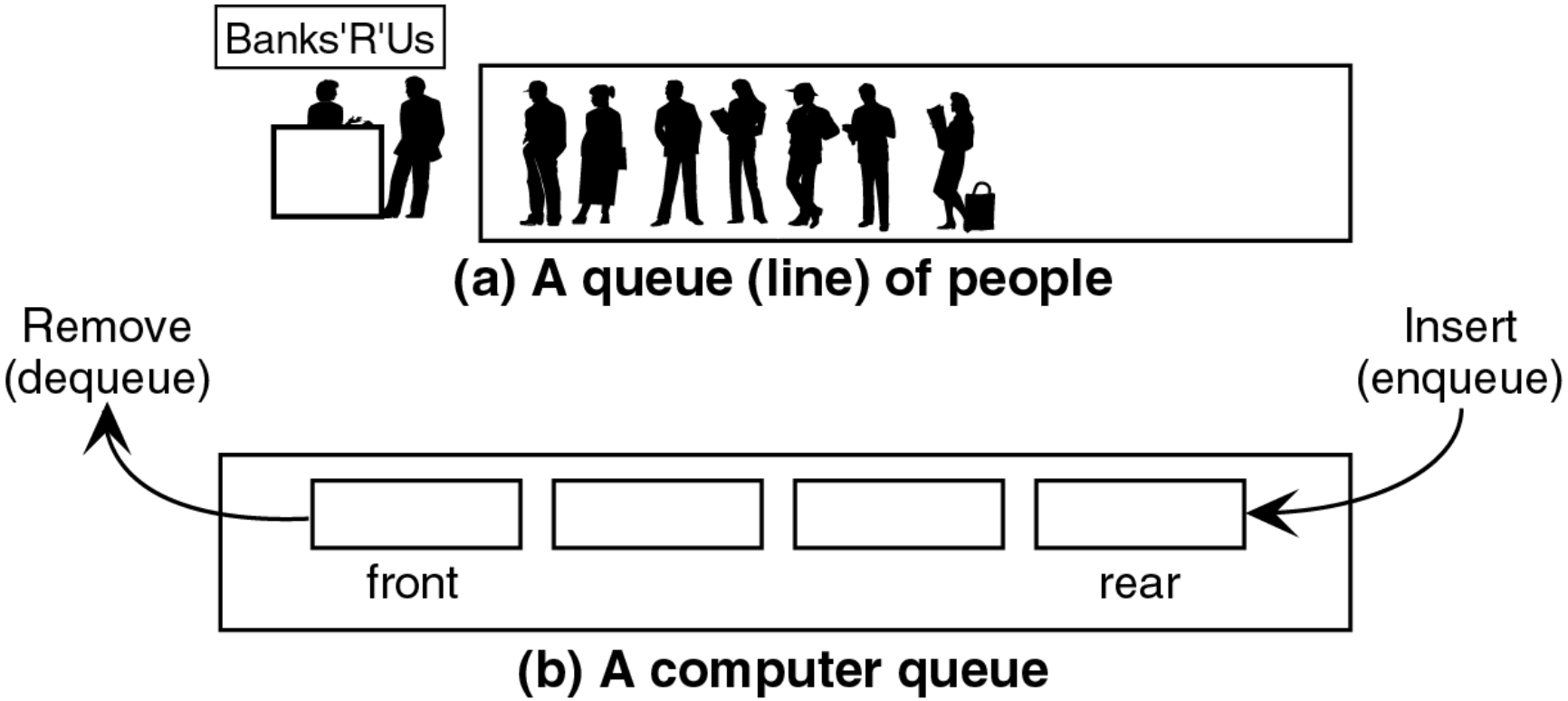
```
void ispisStoga(Cvor *vrh){
  Cvor* tekuci = vrh;
  if(vrh==0)cout<<"Stog je prazan!";
  else
  while(tekuci != NULL){
   cout << tekuci->podatak << " ";
   tekuci = tekuci->veza;
  }
  cout << endl;
}
```

Red

Red

- Struktura podataka u kojoj se elementi mogu dodavati samo na kraj, a uzimati samo s početka reda
- Linearna struktura podataka
- Element koji je prvi dodan u red, prvi se ‘uzima’ iz reda – FIFO struktura (eng. *first in - first out*)
- Primjer:
  - Elementi ulaze u red zadanim redoslijedom: 7,20,3,5
  - Stog: 7,20,3,5

Red


Banks'R'Us
(a) A queue (line) of people
Remove
(dequeue)
Insert
(enqueue)
front
rear
(b) A computer queue

Red

- U redu je omogućen direktan pristup prvom elementu reda, a to je element koji je prvi ušao u red
- Posljednji element reda (koji se nalazi na kraju reda) je element koji je zadnji ušao u red.
- Pristup elementu koji se nalazi u redu – proći kroz cijeli red da bi se došlo do traženog elementa reda

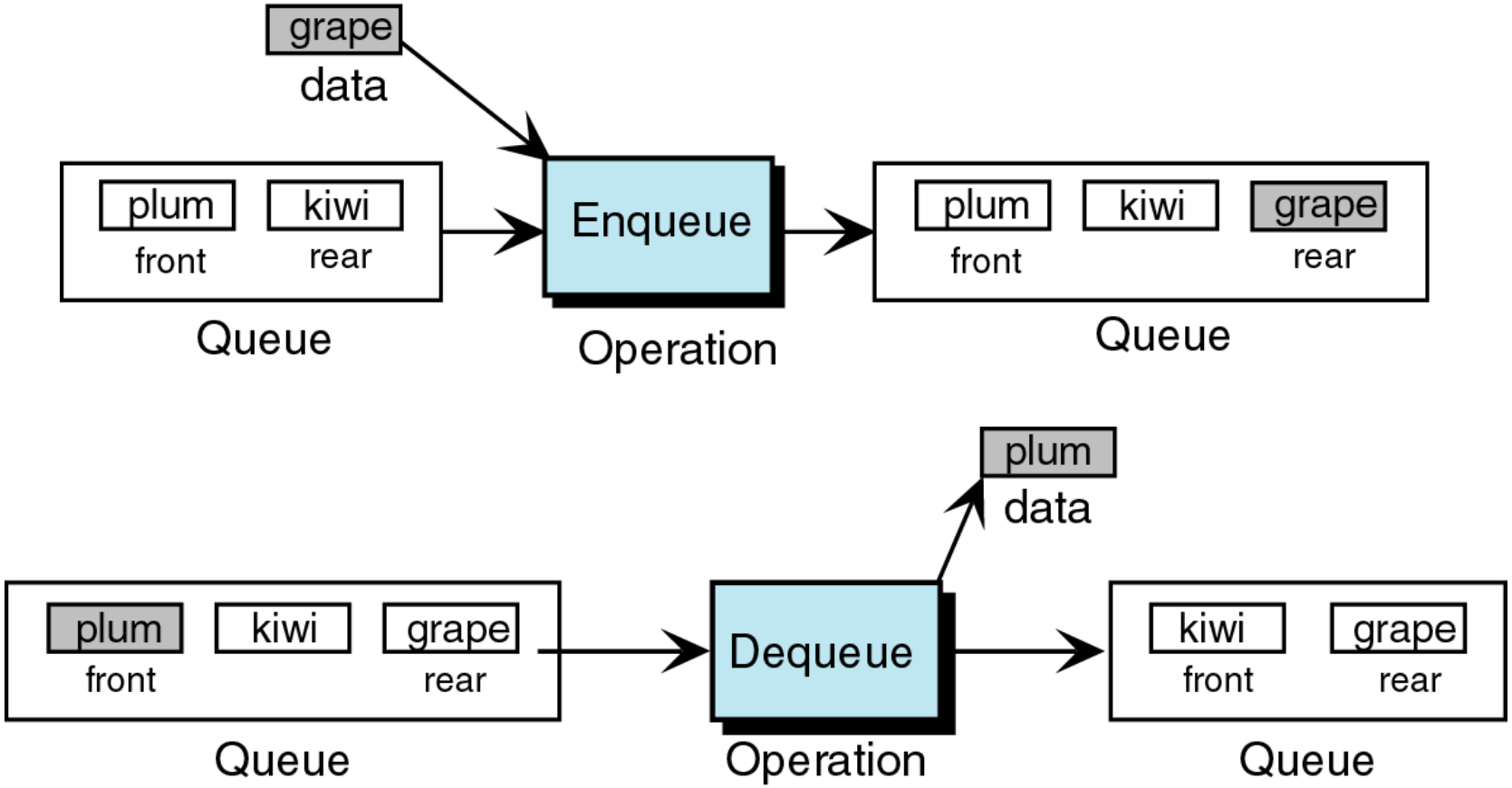
Primjene reda

- Implementacija različitih redova čekanja
  - šalteri za čekanje, procjene i planiranja redova čekanja
- Računala koriste red u različitim aplikacijama:
  - Za rješavanje zahtjeva za pisač (prvi zahtjev za ispis je prvi u redu i prvi je obavljen)
  - Paketi koji putuju Internetom stižu u rutere i obrađuju se redoslijedom kojim stižu.

Operacije s redom

- Osnovne funkcije
  - Enqueue
    - Dodavanje elementa u red (na kraj reda)
  - Dequeue
    - ‘uzimanje’ elementa iz reda (s početka reda)

Operacije s redom



Operacije s redom

- Ostale operacije sa redom:
  - queueFront - pristupanje elementu u glavi
  - queueRear - pristupanje elementu u repu
  - Prazan (isEmpty) – provjera je li red prazan, vraća true ako je red prazan, inače false.
  - Kreiraj/Uništi red (init/delete) **–** funkcije za kreiranje reda i za brisanje svih elemenata reda
  - Obilazak – funkcija za obilazak reda (i eventualno ispis podataka)

Pitanje

- Na koji način biste implementirali red u jeziku C++?

Implementacija reda

- Implementacija reda
  - Polje
  - Povezana lista
- Implementacija reda primjenom povezane liste
  - Čvor povezane liste sadrži podatak i pokazivač na slijedeći element reda.
  - Moramo voditi računa o pokazivačima na prvi i zadnji čvor povezane liste (**glava** i **rep**).

Kreiranje reda

- Kreiraju se pokazivači:
  - **`glava`** - pokazuje na prvi element u redu
  - **`rep`** – pokazuje na zadnji element u redu
- Na početku su oba pokazivača inicijalizirana na NULL

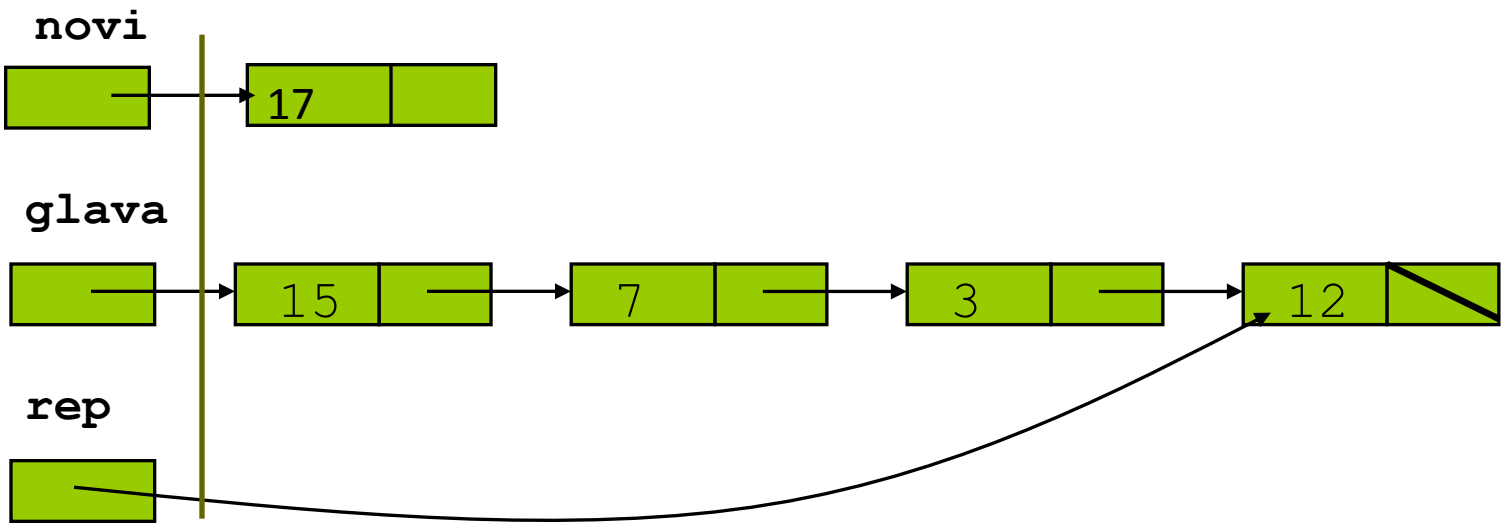
`glava` **`rep`**

```
cvor *glava = NULL;
cvor *rep = NULL;
```

Dodavanje elementa u red

- Funkcija **`enqueue`**
- Operacija umeće element u red
  - uvijek na kraj reda
- Izvodi se kao i umetanje čvora na poziciji repa povezane liste.
- Potrebno je:
  - Alocirati memoriju za novi čvor
  - Preusmjeriti vezu zadnjeg čvora da pokazuje na novi čvor
  - Preusmjeriti pokazivač repa na novi element reda na kraju reda

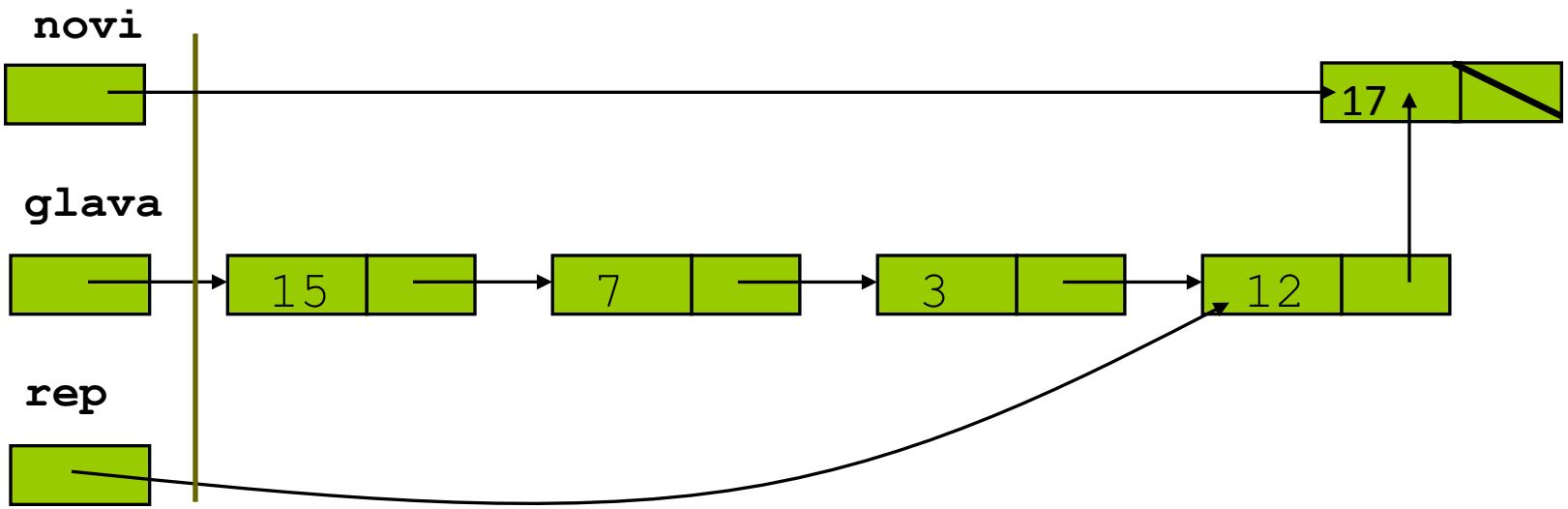
Dodavanje elementa u red



Kod:

```
novi = new cvor;
novi -> podatak = 17;
```

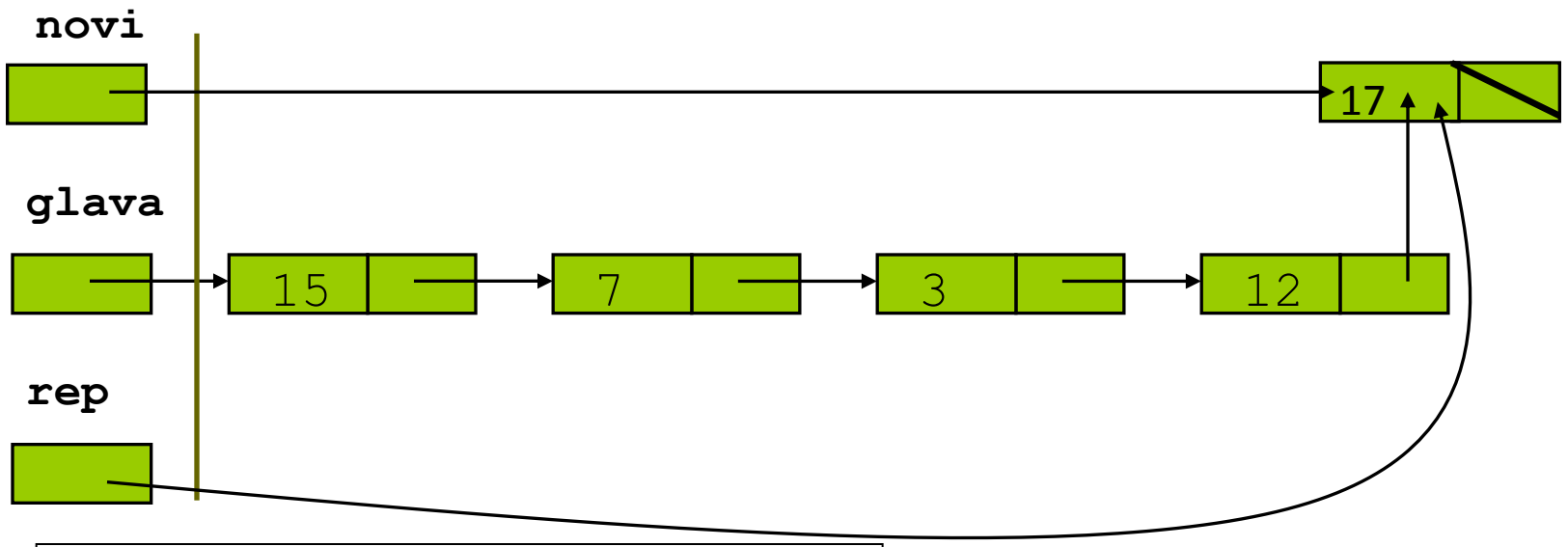
Dodavanje elementa u red



Kod:

```
novi = new cvor;
novi -> podatak = 17;
novi -> veza = NULL;
rep -> veza = novi;
```

Dodavanje elementa u red



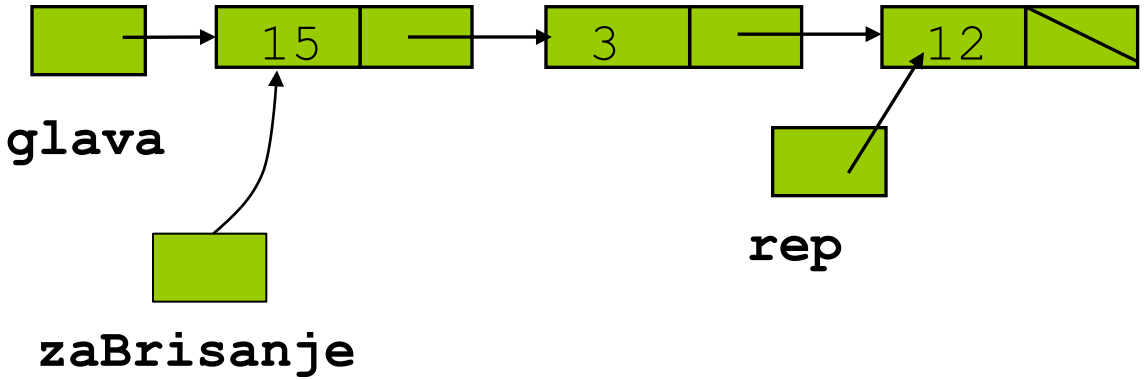
Kod:

```
novi = new cvor;
novi -> podatak = 17;
novi -> veza = NULL;
rep -> veza = novi;
rep = novi;
```

Izlazak elementa iz reda

- Funkcija **`dequeue`**
  - Operacija ‘uzima’ prvi element iz reda
    - Element izlazi iz reda, oslobađa se memorija
  - Izvodi se kao i brisanje čvora na poziciji glave kod povezane liste
- Postupak:
  - Pohraniti (vratiti) podatak s početka reda
  - Preusmjeriti pokazivač koji pokazuje na početak reda (glava) na novi element koji će biti početak reda
  - Osloboditi memoriju čvora koji je bio na početku reda
  - Voditi računa o pokazivaču na zadnji element (rep)

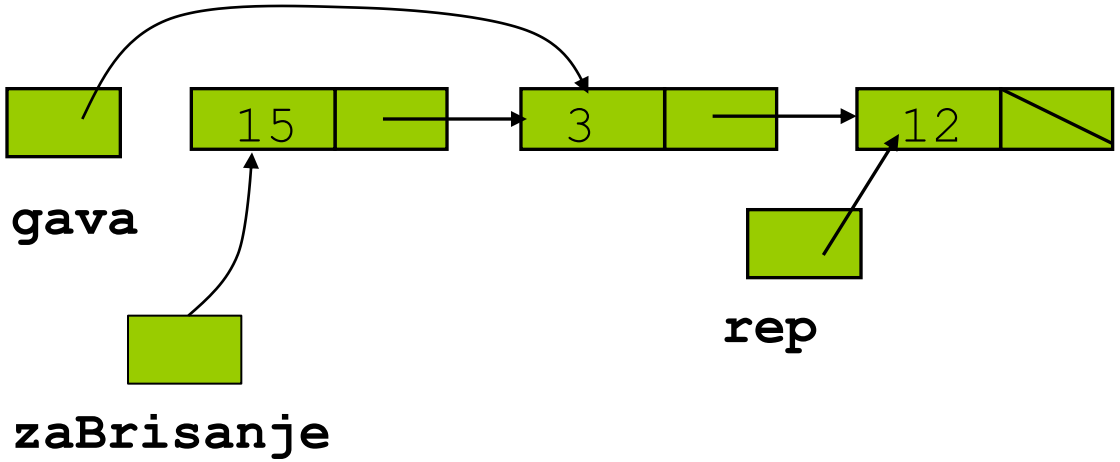
Izlazak elementa iz stoga



Kôd:

```
cvor *zaBrisanje = glava;
podatak = glava -> podatak;
```

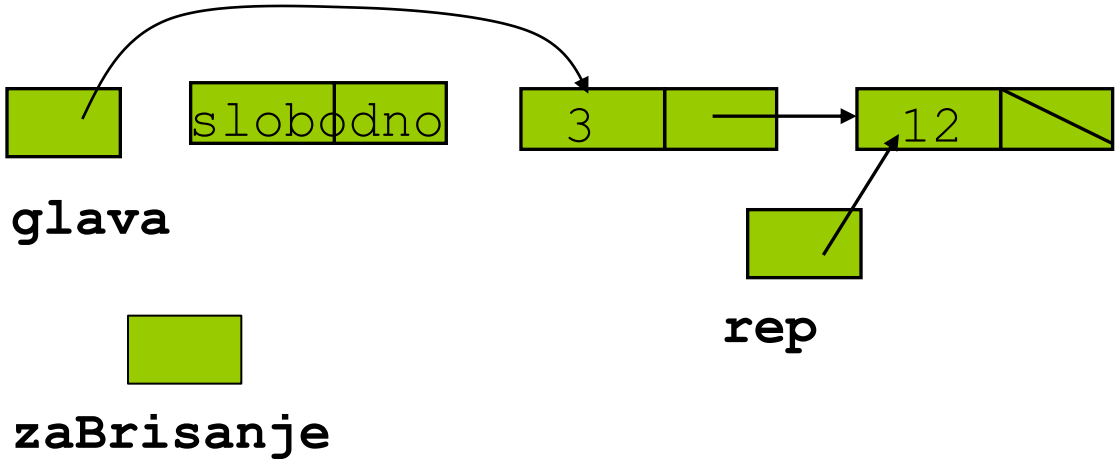
Izlazak elementa iz stoga



Kôd:

```
cvor *zaBrisanje = glava;
podatak = gava -> podatak;
glava = glava -> veza;
```

Izlazak elementa iz stoga



Kôd:

```
cvor *zaBrisanje = glava;
podatak = glava -> podatak;
glava = glava -> veza;
delete zaBrisanje;
```

Provjera je li red prazan

```
bool jePrazan(cvor *glava) {
if (glava == 0)
    return true;
else
    return false;
}
```

Uništavanje reda

```
void brisiRed(cvor *&glava, cvor*&rep)
   {
   cvor *zaBrisanje;
   while( glava != NULL ) {
       zaBrisanje = glava;
       glava = glava->veza;
       delete zaBrisanje;
   }
   rep = NULL;
}
```

Ispis elemenata reda

```
void ispisReda(cvor *glava) {

   cvor *tekuci = glava;
   if (jePrazan(glava))cout<<"Red je prazan"<<endl;
   while( tekuci != NULL ) {
       cout << (tekuci->podatak) << endl;
       tekuci = tekuci->veza;
   }
}
```

Literatura

- Data Structures (A Pseudocode Approach with c), autori: R.h. Gilberg, B.A. Forouzan
- Nick Parlante: Linked List Basics
  - http://cslibrary.stanford.edu/103/LinkedListBasics.pdf
- Nick Parlante: Pointers and Memory
  - http://cslibrary.stanford.edu/102/PointersAndMemory.pdf
- Nick Parlante: Linked List Pronlems
  - http://cslibrary.stanford.edu/105/LinkedListProblems.pdf