Programski jezici i strukture podataka

13

Stek

- Stek je mehanizam za prihvatanje podataka.
- Stek je linearni tip podataka koji se ponaša kao memorijski mehanizam tipa „poslednji-u, prvi-iz“ (na engleskom "last-in, first-out" - LIFO).
- Najčešće korišcena analogija za stek u svakodnevnom životu je gomila tanjira koja se može naci u vecini restorana. Poslednji tanjir stavljen na vrh gomile uzima se prvi.

Stek

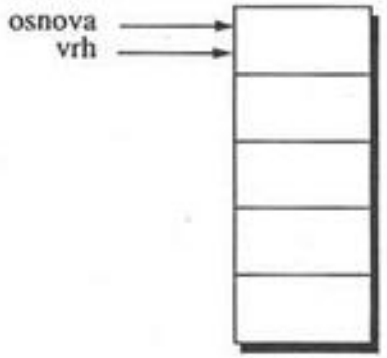
- Savremeni računari su mašine orijentisane na rad sa stekom.
- Stekovi se koriste u razne svrhe, kao što su:
  - čuvanje stanja mašine za vreme dok prekidne rutine opslužuju zahteve višeg prioriteta,
  - predaja parametara funkcijama,
  - vođenje evidencije o preklapajućim prozorima u interfejsimai tako dalje.

Stek

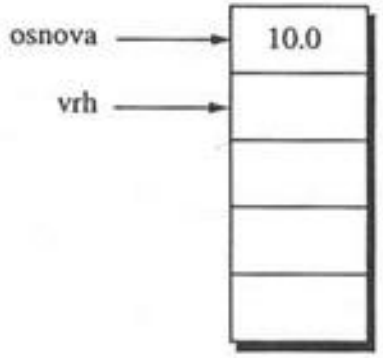
- Struktura steka je linearna.
- Dozvoljen je pristup samo prvom elementu.
- Dodati se može samo ispred prvog elementa.
- Funkcije za rad sa stekom:
  - push
  - pop

STEK (poslednji-u, prvi-iz)

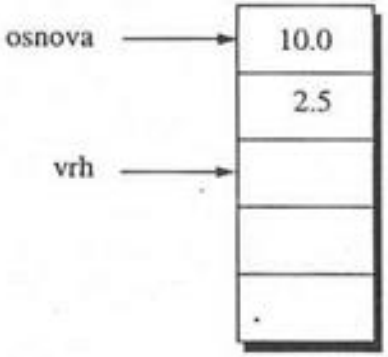
a) Prazan stek



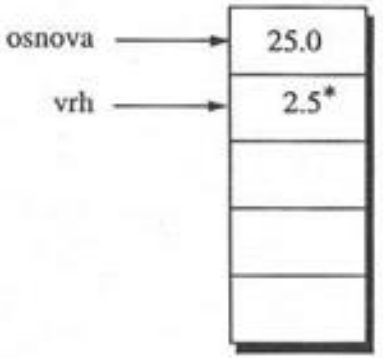
b) Posle push(10.0)



c) Posle push(2.5)



d) Posle push(pop()*pop())



*Vrednost ostaje na steku, ali će pri sledećem pozivu funkcije push() biti prepisana novim sadržajem.

Tip podataka - stek

- Stek ima osobinu tipa podataka.
- Realizacija steka može biti sekvencijalna ili spregnuta.
- Najjednostavnija implementacija steka je pomoću niza i pratećih pokazivača ili indeksa koji pokazuju gde podatak treba da se upiše i odakle da se pročita, sekvencijalna realizacija.

  (**z39.c**)

Spregnuta realizacija steka

- Spregnuta realizacija steka celobrojnih vrednosti upotrebom jednostruko spregnute liste.(**z40.c**)

SEKVENCA

Definicija

- Sekvenca se definiše kao uređeni par:

$$D=(S(D),r(D))$$

sa sledećim osobinama:

- struktura je linearna
- dozvoljen je pristup svakom elementu
- ukloniti se mogu samo svi elementi odjednom
- novi se dodaje na kraj

Upotreba sekvence

- Sekvenca se koristi kao mehanizam organizovanja podataka na masovnim, sekundarnim, memorijama.
- Recimo film se smešta kao sekvenca blokova.
- Prilikom pisanja, novi blok se dodaje iza poslednjeg.
- Brisanje, ako je moguće, obavlja se odjednom nad svim elementima, a pristupa se od bloka do bloka.

Opreacije nad elementima sekvence

- **Pristup** je moguć bilo kom elementu sekvence, ali krećući se od elementa kome se prethodno pristupalo. (ako se prethodno pristupalo trećem elementu a sada je potrebno pristupiti 25. elementu, obaviće se sekvencijalni prolazak 4., 5., ..., 23. i 24. elementa da bi se stiglo do 25. elementa, slično kretanju kabine lifta).
- **Uklanjanje** je moguće samo nad svim elementima sekvence odjednom, brisanje sekvence.
- **Dodavanje elementa** je moguće na kraju sekvence, iza poslednjeg elementa.

RED

Red

- Kod steka poruke se čitaju obrnutim redosledom u odnosu na redosled dolaska.
- Ako upotrebimo red umesto steka, poruke mogu da se čitju onim redom kojim su primane.
- Baferi koji omogućavaju rad sa uređajima kod kojih se brzine transfera podataka jako razlikuju, kao što su CPU i diskovi, obično se implementiraju u obliku redova.

RED (QUEUE)

- Red je po svojoj organizaciji implementira situaciju u kojoj više korisnika čeka servis sa jednog izvora: red automobila na benzinskoj pumpi, red klijenata pred šalterom...
- Red je linearna struktura podataka čiji se terminal gde se vrši dodavanje novog elementa naziva **kraj reda**, a suprotni terminal je **početak reda** tamo se obavljaju operacije pristupa i uklanjanja elementa.

Operacije nad redom

- **FRONT(prvi)**, pristupa elementu na početku reda i vraća njegovu vrednost bez destrukcije sadržaja reda.
- **OUTQUEUE(iz_reda)**, pristupa elementu na početku reda i vraća njegovu vrednost uklanjajući ga iz reda. Operacija je destruktivna jer se njenom primenom smanjuje broj elemenata reda pri čemu element koji je bio drugi postaje element na početku reda.
- **INQUEUE(u_red)**, smešta na kraj reda novi element pri čemu do tada poslednji element postaje pretposlednji. Operacija je destruktivna jer se njenom primenom povećava broj elemenata reda pri čemu se mora voditi računa o memorijskom prostoru koji je dodeljen redu.

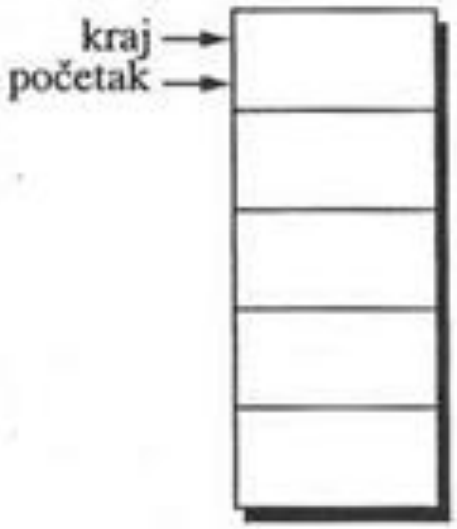
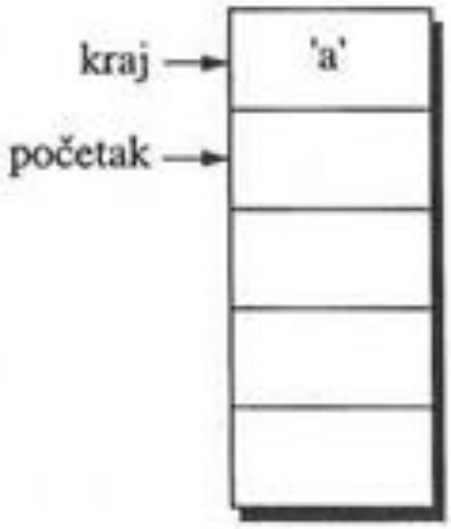
Tip podataka - red

- Red je mehanizam smeštanja u memoriju tipa prvi-u, prvi-iz (na engleskom "first-in, first-out" - FIFO).
- Element koji je prvi smešten u red, prvi se uzima iz reda. Red možemo implementirati pomoću niza znakova isto kao kod steka.
- Indeks početka pokazuje gde treba da se obavi sledeca operacija upiši
- Indeks kraja pokazuje odakle će se obaviti sledeća operacija pročitaj.

Tip podataka - red

- Za manipulaciju redom se realizuju dve funkcije **ured**() za upis elementa u red i **izreda**() za čitanje elementa iz reda.
- Kada se pokuša upis u red, proverava se najpre indeks početka.
- Ako lokacija na koju indeks početka ukazuje nije iza kraja reda, element se upisuje na tu lokaciju, zatim se inkrementira početak.
- Ako je početak već iza kraja reda, ništa ne može da se upiše.
- Na element koji se čita ukazuje indeks kraja.
- Ako je indeks kraja jednak indeksu početka, red je prazan i nema šta da se pročita.
- U suprotnom, obavlja se operacija pročitaj i indeks kraja se inkrementira.

RED (prvi u, prvi iz)

Prazan red	Posle ured(a)	Posle izreda()
		

Logička struktura **reda**

$$F=(S(F),r(F))$$

pri čemu se skup elemenata može urediti tako da da bude predstavljen sa:

$$S(F)=\{x_1,x_2,\ldots,x_n\}$$

realizacija **r** definiše se sa:

$$r(F) = \begin{cases} \{(x_i, x_{i+1}) \mid i = 1,\ldots,n-1\} & \text{za } n > 1 \\ 0 & \text{za } n \leq 1 \end{cases}$$

Za element x_1 se kaže da je na početku reda, a za element x_n da je na kraju reda.

Logička struktura reda

Da bi se ostvarila logička struktura reda potrebno je definisati dve primitivne funkcije:

1. Funkcija **prvi** koja daje element na početku (dakle x_1).
2. Funkcija **poslednji** za izdvajanje elementa x_n na kraju reda.

Izvedene, operacije nad redom:

1. provera da li je red prazan
2. uklanjanje svih elemenata
3. određivanje broja elemenata

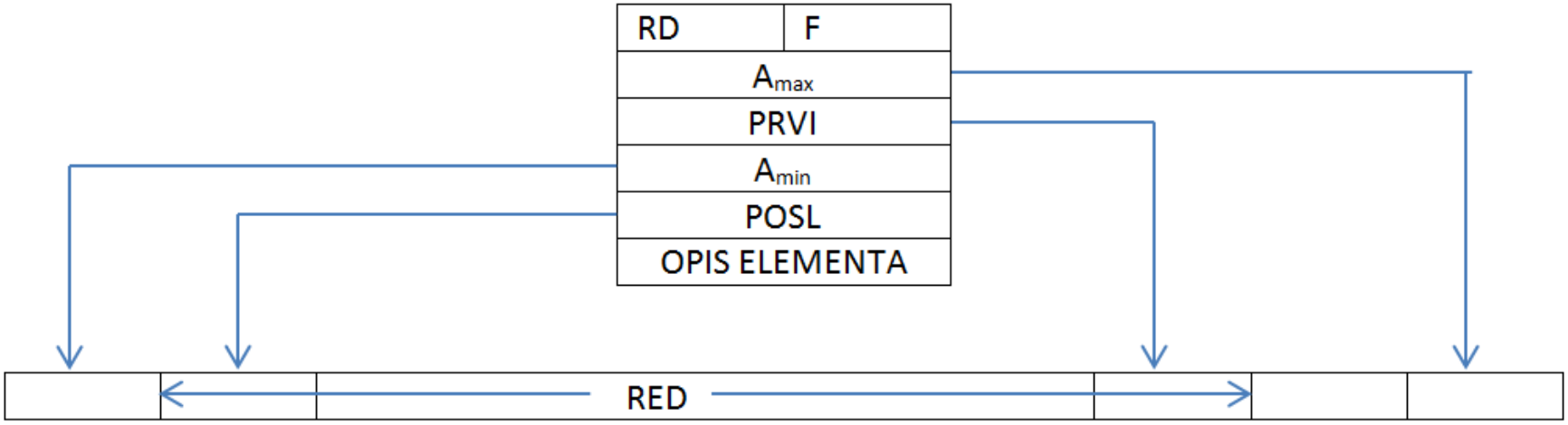
Operacije se realizuju na sličan način kao kod steka.

- Funkcija **poslednji**, sama po sebi nije kompleksna, takođe fizička realizacija je jednostavna.
- Problem sa ovom funkcijom se odnosi na samu definiciju reda, naime,red je definisan kao struktura podataka kod koje se pristup ostvaruje samo na njegovom početku.
- Nasuprot tome, funkcija **rear** nije ništa drugo no pristup elementu na kraju strukture i kao takva nije primenljiva na red.
- Daljom analizom vidimo da ova operacija ne koristi niti jednu drugu osnovnu operaciju (koristi samo primitivnu funkciju) i samim time je osnovna, što usložnjava analizu prava njene egzistencije.
- No, rešenje postoji, i vezano je za kontekst. Odnosno ako je kontekst u kojem se pojavljuje struktura takav da se predviđa pristup na oba kraja, onda to treba i omogućiti, uz ogradu da se u tom slučaju ne radi o redu već o strukturi koja je slična redu (ali i steku).

Fizička struktura reda

- Kao i kod steka, realizacija je sekvencijalna ili spregnuta.
- Sekvencijalna fizička realizacija podrazumeva postojanje deskriptora sličnog deskriptoru steka, s tom razlikom da ovde postoji osim adrese prvog, i adresa poslednjeg koja je takođe promenljiva.
- Putem ove dve adrese se definišu dve primitivne funkcije prvi i poslednji.

Sekvencijalna fizička realizacija reda

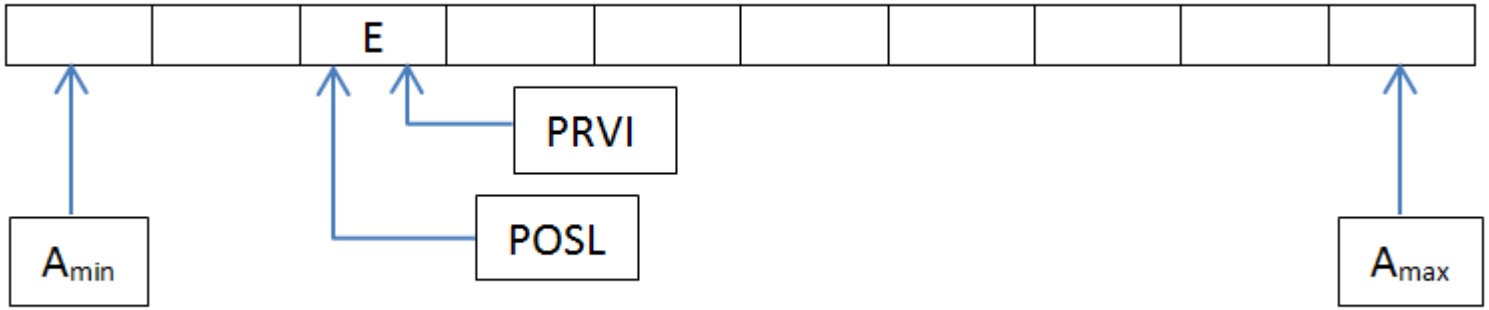


Deskriptor reda obuhvata:

- indikator strukture reda, RD
- ime reda, F
- najveću adresu memorijskog prostora, A_{max}
- adresu početka reda, PRVI
- najmanju adresu memorijskog prostora, A_{min}
- adresu elementa na kraju reda, POSL
- opis elemenata reda.

- Na osnovu prethodnog prikaza se može primetiti da se fizička struktura **steka**, može smatrati **posebnim** slučajem fizičke strukture **reda** kod kojeg je POSL=A_{min}=const.
- Sekvencijalna struktura reda karakteristična je po jednoj pojavi: Stanje prepunjenosti reda čiji memorijski prostor nije do kraja zauzet.

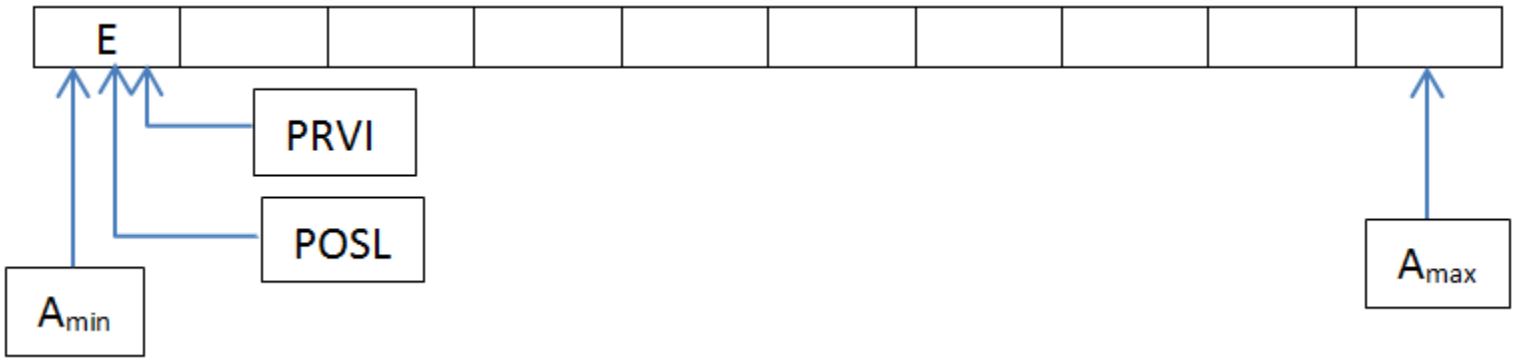
Posmatrajmo red R sa jednim elementom koji je smešten u memorijski prostor koji može da primi 10 elemenata, neka je stanje reda u nekom trenutku sledeće:



Nakon primene sledeće sekvence:

InQueue(F,a), OutQueue(F), InQueue(F,b), OutQueue(F),

stanje reda će biti:



- Kao što se može videti prva primena operacije **InQueue** izazvaće prepunjenost reda, iako u njemu postoji samo jedan element.
- Zapravo ova ilustracija opisuje sledeći paradoks: usled promenljivosti obe granične adrese red kao da se pomera po memoriji ka minimalnoj adresi i kada je dostigne ulazi u stanje prepunjenosti, ne zato što nema mesta za upis novog elementa, nego zato što nema mesta za pomeranje reda.

Da bi se prevazišao paradoks, potrebno je obezbediti mehanizam za pomeranje reda, ovo se postiže cirkularnom strukturom.

Neka je l(F) broj memorijskih lokacija određen za smeštanje jednog elementa.

Prilikom realizacije operacije InQueue adresa za upis novog elementa izračunava se na sledeći način:

$$\text{POSL}=\begin{cases} POSL - l(F) & POSL > Amin \\ Amax & POSL = Amin \end{cases}$$

odnosno, ako je najniža aresa popunjena, novi element se kružno upisuje na najvišu adresu Amax.

Slično prilikom uklanjanja elementa pomoću operacije OutQueue adresa se preračunava na sledeći način:

$$\text{PRVI}=\begin{cases} PRVI - l(F) & PRVI > Amin \\ Amax & PRVI = Amin \end{cases}$$

Na taj način, sukcesivna primena operacija InQueue i OutQueue izaziva svojevrsno kruženje reda po memorijskom prostoru bez opasnosti da dođe do lažne prepunjenosti, izuzev slučaja kada je memorija zaista popunjena.

Tehnika cirkularne realizacije stvara novi negativni efekat, koji se mora uzeti u obzir.

Definišimo funkciju:

$$\text{prethodni}(x,F)=\begin{cases} x-l(F) & x > Amin \\ Amax & x = Amin \end{cases}$$

gde je x element reda F.

Pretpostavimo da red ima jedan jedini element, te operacija OutQueue može da se izvrši još samo jednom.

Neposredno pre izvršavanja je PRVI=POSLEDNJI, tako da će posle toga red biti prazan i važiće:

prethodni(POSL,F)=PRVI

Lako je uočiti da ista realizacija važi i za slučaj da su elementi reda poređani u redosledu (po rastućim adresama)

POSL,..., Amax,Amin,...,PRVI,

a to je očigledno red koji je pun.

Odavde sledi da ako se cirkularna tehnika primeni bez dodatnih mehanizama, nema mogućnosti da se utvrdi da li je memorijski prostor potpuno zauzet ili potpuno slobodan.

Jedan od načina za prevazilaženje ovog problema jeste mehanizam kojim se lokacija koja je iza (u smislu funkcije prethodni) aktuelnog poslednjeg tretira drugačije od ostalih i to tako što se u nju ne upisuje element.

Između adrese te lokacije (nazovimo je PRAZNA) i adrese kraja reda po definiciji se uspostavlja veza:

```
prethodni(POSL,F)=PRAZNA
```

Sada je uslov za prazan red da bude:

$$PRVI=PRAZNA$$

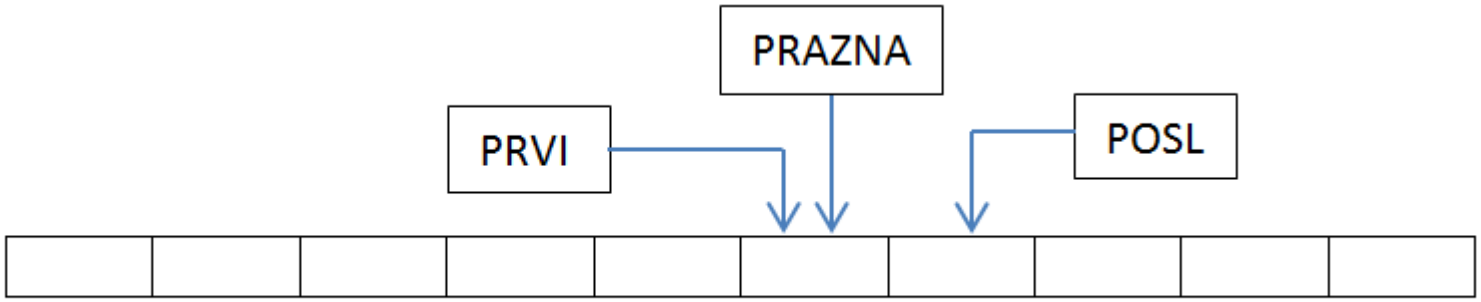
ili posredstvom funkcije prethodni:

$$prethodni(POSL,F)=PRAZNA$$

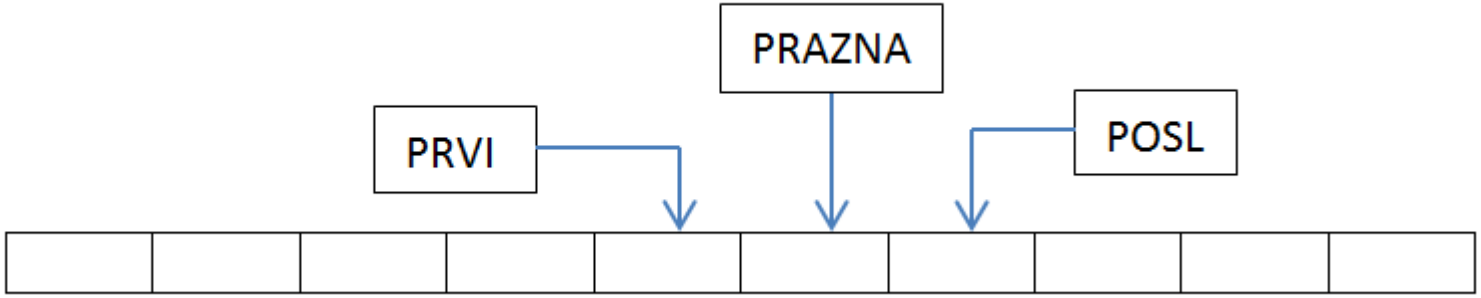
a za pun red:

$$prethodni(PRAZNA,F)=PRVI$$

Prazan red:



Pun red:



Kružni red

- Linearni redovi ograničeni su dužinom reda. Zbog toga se ne koriste često.
- Jedna varijacija linearnih redova - kružni red koristi se često, naročito za baferovanje pri prenosu podataka.
- Kružni red je takođe pravolinijski deo memorije, ali se prilikom pristupa vraćamo nazad na početak kada se dosegne kraj reda.

Kružni red

- Dobar primer kružnog reda je bafer za tastaturu personalnog računara.
- Bafer tastature prima znakove brzinom kojom ih kucate, a procesu ih predaje na zahtev.
- Ako je proces zauzet nekim drugim poslom, ovi baferi omogućuju vam da nastavite sa kucanjem, a da se ništa od toga što otkucate ne izgubi.

Kružni red

- Ako su indeksi početka i kraja jednaki, da li je red pun ili prazan? To se ne može znati.
- Za razlikovanje ove dve situacije žrtvujemo jedno mesto u redu.
- Indeks kraja se inicijalizuje na jedno mesto iza indeksa početka i nije mu dozvoljeno da ga stigne. (**z41.c**)

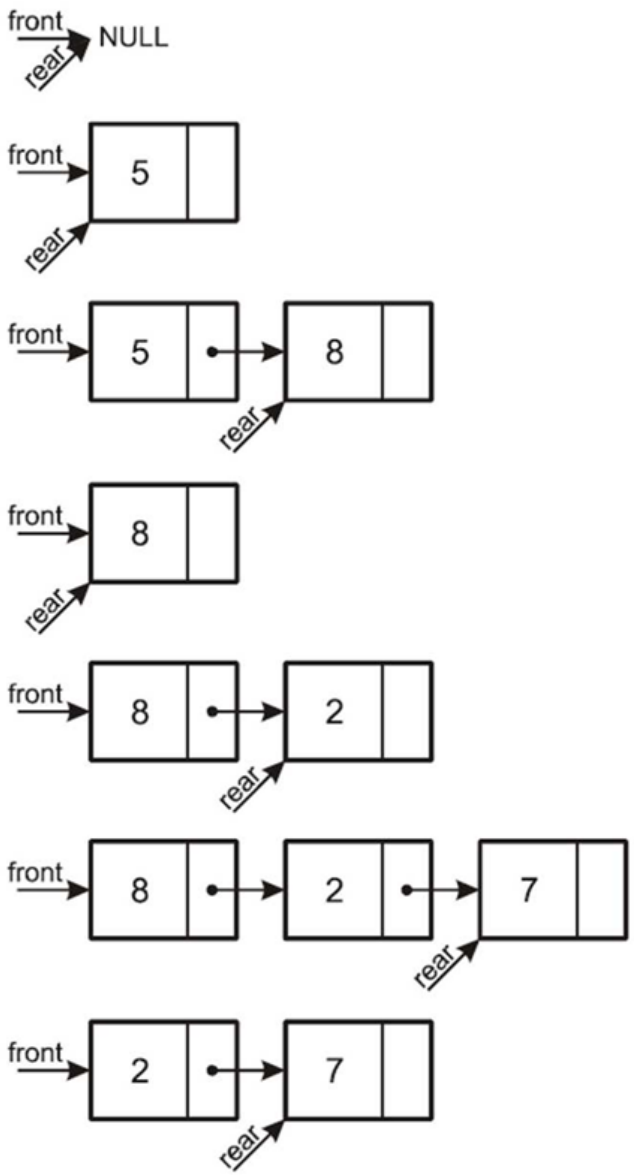
Spregnuta realizacija reda

- Red se može veoma efikasno implementirati korišćenjem spregnutih lista tako što bi se novi element dodavao uvek na kraj liste.
- Takode, u slučaju brisanja elementa iz reda skidao bi se uvek prvi element u listi, odnosno onaj koji je prvi dodat.

Spregnuta realizacija reda

- Na početku, lista je prazna, pa su i pokazivači na početak i kraj liste *(**front*** i ***rear**)* jednaki NULL.
- Funkcija ***insert*** kreira novi element i dodaje ga na kraj liste.
- Funkcija ***delete*** vraća vrednost prvog elementa u listi i pomera pokazivač *front* na sledeći element u listi.
- Na kraju se uništava element koji je bio na početku liste.

- Prazan red
- Insert(5)
- Insert(8)
- Delete()
- Insert(2)
- Insert(7)
- Delete()



Z42.c Spregnuta realizacija reda

DEK

DEK (DEQUE)

- Ime je akronim od Double Ended Queue, predstavlja uopštenje steka i reda u smislu načina pristupa, dodavanja i uklanjanja elemenata.
- Dek zapravo predstavlja, po osobinama, sintezu osobina reda i steka.

DEK (DEQUE)

- Struktura je linearna
- Pristup, uklanjanje i dodavanje dozvoljeno na oba kraja.
- Nema prvog niti poslednjeg, ravnopravni su, stoga govorimo o krajnjem desnom i krajnjem levom elementu.

DEK(DEQUE)

- Struktura podataka klase dek nadograđuje strukturu reda mogućnošću obavljanja svake operacije na oba kraja deka.
- Time se dovodi u pitanje gde je početak, a gde kraj. Nad dekom su definisane operacije:
  - pristup elementu na levom, odnosno desnom kraju deka,
  - dodavanja elementa na levom, odnosno desnom kraju deka,
  - uklanjanja elementa sa levog, odnosno desnog kraja deka.

Fizička realizacija deka

- Sekvencijalna realizacija slična realizaciji reda, javlja se pseudopopunjenost koja se rešava cikličnom strukturom kao kod reda.
- Spregnuta (potrebno dvostruko sprezanje)

- Spregnuta realizacija je drugačija u odnosu na spregnutu realizaciju reda, zbog neophodnosti pristupanja, dodavanja i uklanjanja elemenata na oba kraja.
- Prva razlika je u deskriptoru, on mora da obezbedi pokazivače na levi i desni.
- Druga razlika je u sprezanju, mora postojati propagacija svesnosti u oba smera, to podrazumeva dvostruko sprezanje.