Algoritmi i strukture podataka

Studijski programi:
Softversko inženjerstvo, Računarska tehnika, Informatika i matematika
Nastavnik: doc. dr Ulfeta Marovac, umarovac@np.ac.rs

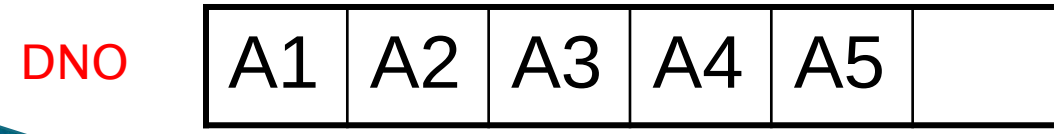
STEKOVI

Uvod - Stekovi

- Stek je stuktura podataka koja u pogledu pristupa nameće odgovarajuća ograničenja
- Operacije sa stekom su konceptualno i implementaciono vrlo jednostavne
- Stek se jako često sreće u računarskim primenama
- Stek se može implementirati u oba načina reprezentacije: sekvencijalna i ulančana

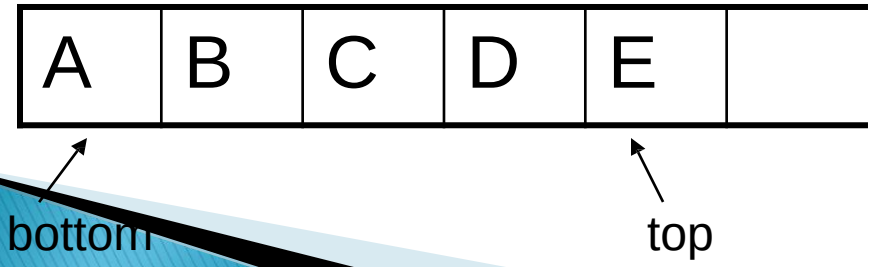
Šta je stek?

- Stek(stack) je struktura koja predstavlja jednu vrstu linearne liste sa specifičnom disciplinom pristupa: *element se na stek može umetati ili sa njega uklanjati samo na jednom kraju linearne stukture.*
- Mesto na kojem se elementi ubacuju i izbacuju se zove vrh(top), dok se drugi kraj zove dno(bottom)
- Element se može uvek ubaciti na stek ali se može brisati samo ako stek ima bar jedan element



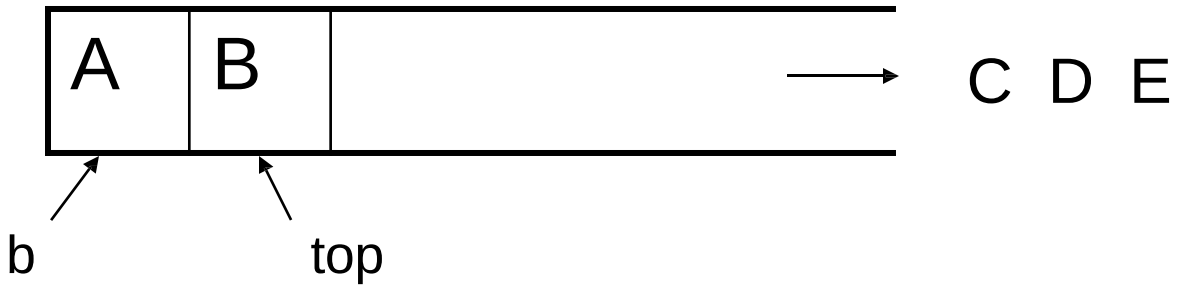
LIFO

- Obzirom da se sa steka uklanja elemenat sa vrha, a on je poslednji koji je umetnut, za stek se kaže da da nameće LIFO(Last In First Out) disciplinu pristupa
- Npr. ako se na prazan stek redom stavljaju elementi A, B, C, D i E, onda je na vrhu poslednju umetnuti elemenat E

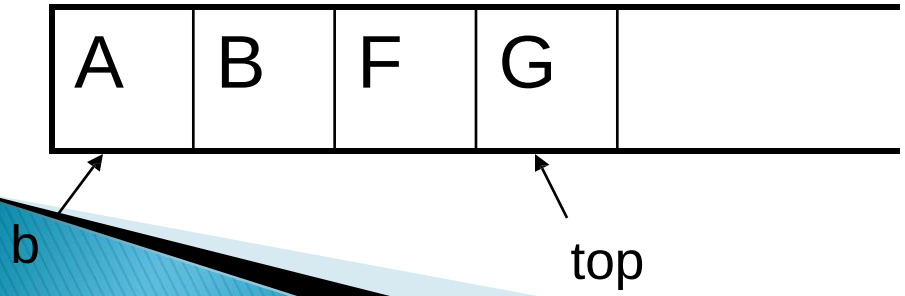


LIFO

- Ako se izvrše tri operacije brisanja onda se elementi uklanjaju u poretku E,D i C



- Naknadne operacije umetanja elemenata F i G daju sledeće stanje



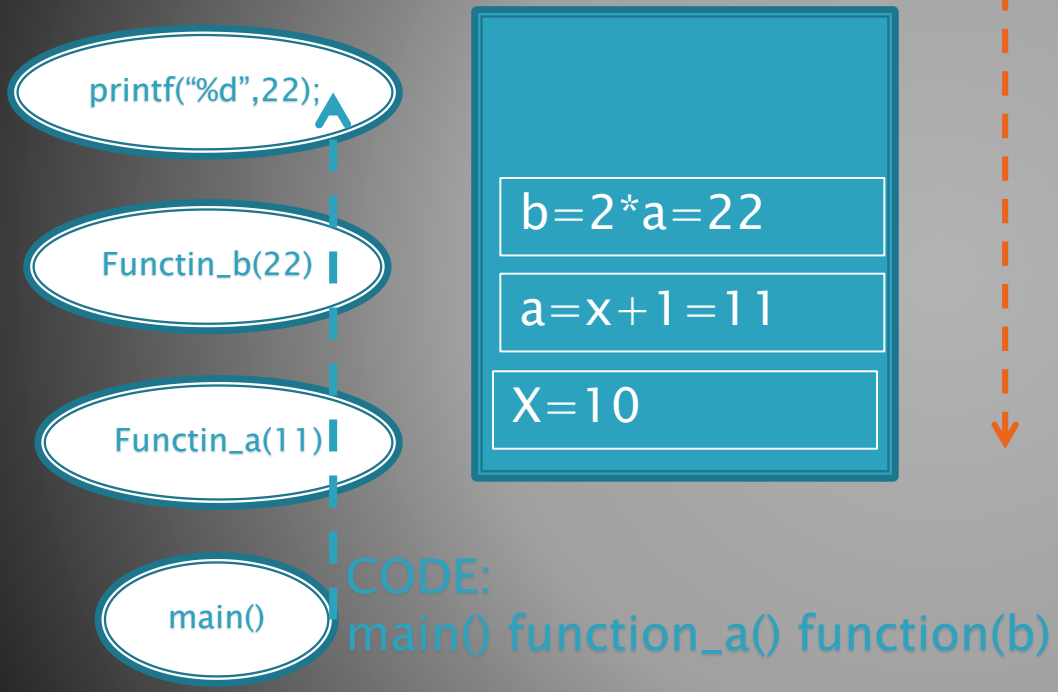
Analogije steka

- Mogu se naći u mnogim realnim situacijama: tanjiri i poslužavnici se slažu kao stek
- U računarskim aplikacijama: pozivanje i povratak iz podprograma, brisanje znakova iz unesene linije u editoru teksta, itd.

Kod steka je **dno** steka *b* fiksno, dok se **vrh** *top* pomera saglasno operacijama umetanja i brisanja elemenata

Pozivanje i povratak iz podprograma

- Promenljive na steku:




```
void function_b(int b)
{ printf("%d",b)
}
  void function_a(int a)
{function_b(2*a);
}
void main()
{ int x=10;
  function_a(x+1)
}
```

Sekvencijalna implementacija steka

- Iako se stek veoma često koristi u računarskim aplikacijama, mnogi programski jezici nemaju standardni ugrađeni tip steka
- Zato ovaj tip treba implementirati simulacijom preko postojećih standardnih tipva
- Pošto stek ne zahteva umetanja i brisanje na proizvoljnoj poziciji, najprirodnije i najčešće se stek implementira u vidu vektora
- Ovakva implementacija zahteva da stek bude **homogena** struktura

Sekvencijalna implementacija steka

- Sekvencijalna implementacija steka obično imlicira konačnu dužinu vektora, koja se određuje po maksimalnoj očekivanoj veličini
- Dno steka se veže za početak a vrh steka se dinamički pomera u opsegu indeksa samog vektora
- Kako bi se pratio vrh steka, uvodi se **pokazivač steka** (stack pointer) gde se pamti indeks elementa vektora koji trenutno predstavlja vrh steka realizovanog ovim vektorom

Sekvencijalna implementacija steka

- U razmatranju sekvencijalne reprezentacije steka, pretpostavićemo da je:
  - Stek predstavljen vektorom S[1:n]
  - Stek raste na gore
  - Celobrojni pokazivač top[S] sadrži indeks vrha steka
  - U svakom trenutku stek se sastoji od elemenata S[1],…,S[top[s]]
  - Top[S] predstavlja razliku broja prethodno izvršenih operacija umetanja i brisanja, pa time i trenutni broj elemenata steka.

Operacije sa stekom- sekvencijalna reprezentacija

- Tipične operacije sa stekom su:
  - Kreiranje steka
  - Umetanje elemenata
  - Brisanje elemenata
  - Čitanje vršnog elementa bez njegovog uklanjanja
  - Provera da li je stek prazan

Operacija kreiranja steka

- Operacija kreiranja steka S veličine n, INT-STACK(S, n), podrazumeva rezervisanje prostora za niz date veličine
- Kako je stek na početku prazan, pokazivač steka se inicijalizuje na nulu

```
INIT-STACK(S, n)
ALLOCATE(S[1:n])
top[S]=0
```

Provera da li je stek prazan

- Pošto pokazivač steka pokazuje i trenutni broj elemenata na steku, ispitivanje da li je stek S prazan se vrši pozivom funkcije

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

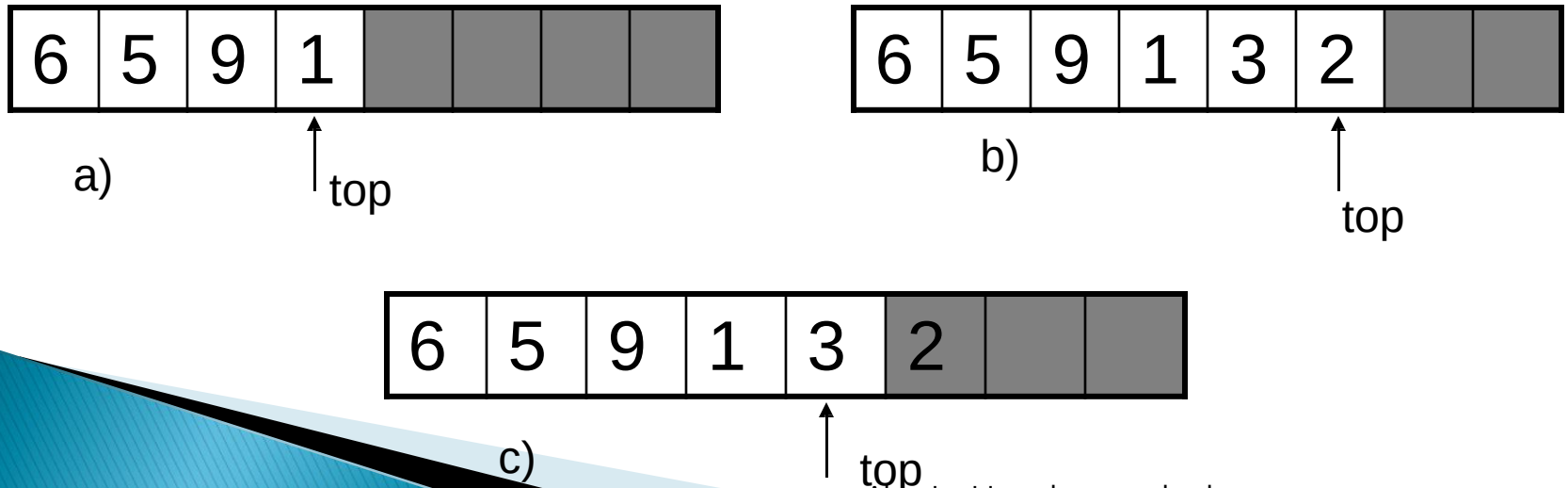
Operacija umetanje novog elementa

- Ova operacija prvo proverava da li je stek pun. Ako pokazivač steka ukazuje da je i poslednja lokacija zauzeta, signalizira se prekoročenje kapaciteta i novi elemenat ne može da se umetne.

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

Operacija umetanje novog elementa

- Ako imamo početno stanje steka, slika a)
- Posle izvršavanja dve operacije umetanja, PUSH(S,3) i PUSH(S, 2), slika b)
- Posle uklanjanja elementa sa vrha, POP(S), slika c)



Operacija uklanjanja elementa

- Funkcija POP(S) uklanja i vraća vrednost elementa sa steka S, što predstavlja inverznu operaciju operaciji umetanja
- Element koji je uklonjen fizički ostaje u nizu, ali je logički uklonjen jer je pokazivač steka pomeren ispod njega

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

Operacija čitanja vršnog elementa

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

- Ova operacija se može dobiti kombinacijom dve uzastopne operacije x=POP(S) i PUSH(S, x) koje u zbiru ne menjaju stanje steka, a vrednost vršnog elementa ostavljaju u promenljivoj x

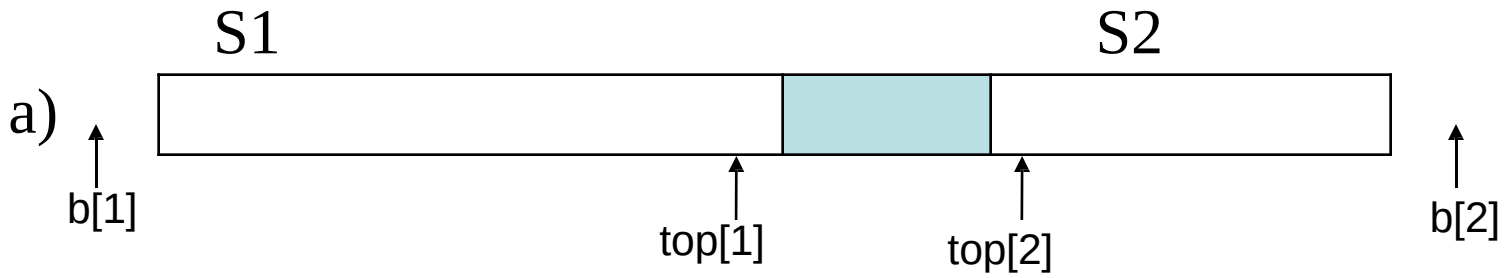
Sekvencijalna implementacija više stekova

- Nekad se javlja potreba za korišćenjem više stekova istovremeno
- Nekad može da se desi prekoračenje na jednom steku dok su drugi daleko od toga da budu puni
- Ovakav slučaj ukazuje da je nepraktično implementirati svaki stek u zasebnom vektoru čija dužina odgovara maksimalnom očekivanom kapacitetu, jer se prostor ne koristi efikasno
- Rešenje koje poboljšava efikasnost korišćenja prostora:

  Smeštanje svih stekova u jedan zajdnički blok memorije(vektor) i dinamička preraspodela ovog prostora u skladu sa potebama pojedinih stekova, jer je malo verovatno da će se svi oni istovremeno napuniti

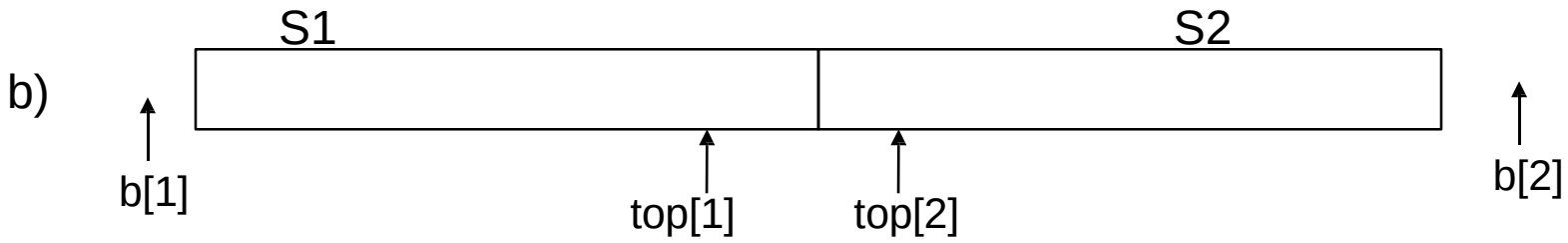
Sekvencijalna implementacija više stekova

- Najjednostavniji slučaj je kod postojanja dva steka
- Tada se za njih rezerviše kontininualni zajednički memorijski prostor u vektoru V[1:m]
- Dno prvog steka se veže za prvu adresu ispred početka vektora, a dno drugog steka za prvu adresu posle kraja vektora, slika a)
- Prvi stek raste nagore a drugi stek raste nadole



Sekvencijalna implementacija više stekova

- Prekoračenje kapaciteta pri umetanju se javlja tek kada je ukupna veličina oba steka jednaka veličini alociranog prostora
- Vrhovi stekova su tada u susednim lokacijama (top[2]=top[1]+1), slika b)



Sekvencijalna implementacija više stekova

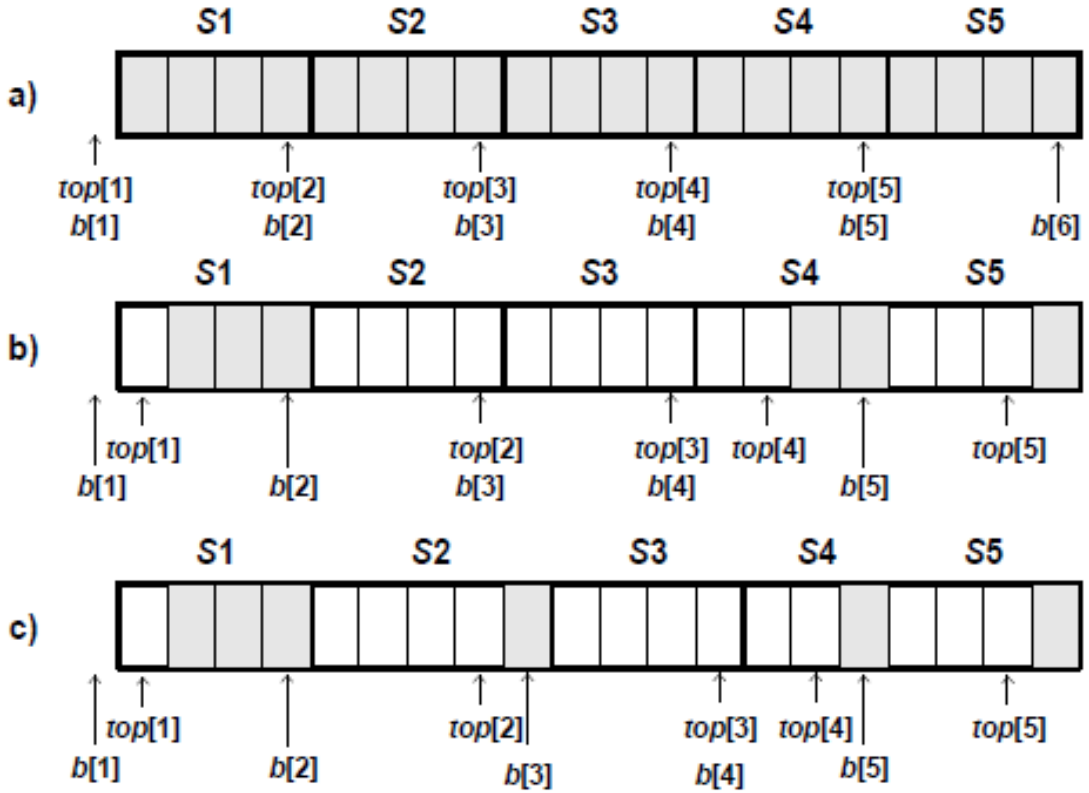
- Za više od dva steka, implementacija u zajedničkom memorijskom bloku postaje dosta složenija
- Nije moguće, kao u slučaju dva steka, vezati dna svih stekova za fiksne pozicije a pritom dinamički raspodeljivati prostor i obezbediti da prekoračenje pri umetanju na bilo kojem steku nastane tek kad ukupna veličina svih stekova postane jednaka veličini alociranog zajedničkog prostora
- Zato je portebno povremeno vršiti pomeranje stekova i time održavati njohovo svojstvo sekvencijalne alokacije
- Ovo rezultuje preraspodelom prostra pa se odlaže pojava prekoračenja

Sekvencijalna implementacija više stekova

- Neka je u zajedničkom vektoru V[1:M] smešteno k stekova S1,...,Sk
- Neka se u vektoru top[1:k] čuvaju pokazivači vrhova stekova, a u vektoru b[1:k] pozicije dna stekova
- Znači, top[i] pokazuje na vrh a b[i] na dno steka Si, $1 \leq i \leq k$
- Ako se pretpostavi da su stekovi iste veličine tada je na početku

  $b[i]=top[i]=\lfloor m/k \rfloor (i-1)$, $1 \leq i \leq k$

Sekvencijalna implementacija više stekova



Sekvencijalna implementacija više stekova

- Neka je u zajedničkom vektoru V[1:M] smešteno k stekova S1,...,Sk
- Neka se u vektoru top[1:k] čuvaju pokazivači vrhova stekova, a u vektoru b[1:k] pozicije dna stekova
- Znači, top[i] pokazuje na vrh a b[i] na dno steka Si, $1 \leq i \leq k$
- Ako se pretpostavi da su stekovi iste veličine tada je na početku

  $b[i]=top[i]=\lfloor m/k \rfloor (i-1)$, $1 \leq i \leq k$

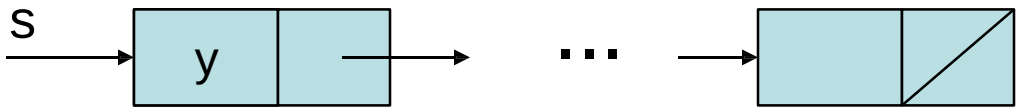
Ulančana reprezentacija steka

- Nedostaci sekvencijalne reprezentacije steka
  - Za stek se alocira prostor ograničene veličine, pa postoji mogućnost prekoračenje
  - Ako stek raste nepredvidljivo, teško je maksimizirati ovaj prostor tako da se smanji verovatnoća prekoračenja
  - Alokacija na maksimalnu očekivanu veličinu steka uslovljava slabo iskorišćenje prostora
  - Imšlementacija više stekova može izazvati zahtevne operacije pomeranja u cilju boljeg iskorišćenja prostora
- Ove teškoće se otklanjaju implementacijom steka u vidu ulančane liste

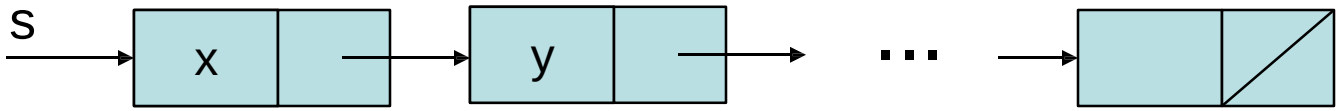
Implementacija steka u vidu ulančane liste

- Ovde pokazivač na listu predstavlja pokazivač steka top, slika a) .
- Posto se pristupa samo prvom čvoru liste, dovoljno je da lista bude jednostuko ulančana
- Slika b), stanje nastalo umatanjem elementa x

a)



b)



Operacije sa stekom implementiranim u vidu ulančane liste

```
STACK-EMPTY-L(s)
if (s=nil) then
return true
else
  return false
end_if
```

```
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)
end_if
```

s je pokazivač na listu
p je pomoćni pokazivač

Primena stekova

- Zbog svoje specifične discipline pristupa, stek se koristi u računarskim programina (prevodioci, operativni sistemi, itd) tako i u aplikativnim
- Razmatraćemo dve karakteristične primene:
  - Obrada aritmetičkih izraza
  - Rad sa podprogramima

Obrada aritmetičkih izraza

- Uobičajeni zadatak prevodilaca viših programskih jezika je razvoj aritmetičkih izraza i generisanje koda za njihovo izračunavanje
- Aritmetički izrazi se u programskim jezicima obično predstavljaju tako da se u binarnim operacijama operator nalazi između pripadajućih operanada, a unarni operator se stavlja ispred operanda
- Ovaj način predstavljanja se naziva **infiksna notacija**
- **Ova notacija se tradicionalno koristi kao** najprirodniji način, vizuelno

Obrada aritmetičkih izraza

- Infiksna notacija ima neke nedostatke koji takve izraze čine nepogodnim za obradu od strane prevodilaca
- U složenijim izrazima sa više operatora i operanada nije jednostavno odrediti pripadnost operanada operatorima, a time i poredak njihovog izvršavanja
- Ako se u izrazu A+B*C zaborave poznate konvencije, postavlja se pitanje koji operand pripada kojem operatoru
- Problem pripadnosti se može rešiti zagradama

Obrada aritmetičkih izraza

- Ugnježdanjem zagrada se postiže da je pripadnost operanda operatorima nedvosmisleno određena
- Ovakva forma se naziva **izraz sa potpunim zagradama**
  (A-((B*C)/(D+E)))
- U izrazu sa potpunim zagradama je poredak izvršavanja operacija jednoznačno određen
- Ako se nivo ugnježdavanja definiše kao broj parova zagrada koji okružuju neki operator, onda se on može koristiti za određivanje redosleda izvršavanja pojedinih operatora

Obrada aritmetičkih izraza

- U izrazu sa potpunim zagradama operatori se izvršavaju po opadajućem nivou ugnježđavanja
- Prvo oni sa najvećim nivoom, pa sa sledećim manjim a poslednji se izvršavaju operatori sa nivoom 1
- Ukoliko više operatora ima isti nivo, izvršavaju se jedan za drugim, s leva u desno
- Za izraz (A-((B*C)/(D+E))), prvo ide na podizraz na nivou 3 (B*C) pa (D+E) pa onda podizraz na nivou 2 ((B*C)/(D+E)) i na kraju ceo izraz koji je na nivou 1 (A-((B*C)/(D+E)))

Obrada aritmetičkih izraza

- Da bi se smanjio broj zagrada uvodi se pojam prioriteta operacija. Za izraz A-B*C/(D+E*F) – napisati poredak izvršavanja
- Određivanje poretka izvršavanja operatora u podizrazima na osnovu prioriteta i nivoa ugnježdavanja u infiksnom izazu zahteva da prevodilac obično više puta ispita izraz s leva na desno
- Ovo je glavni nedostatak infiksne notacije
- Ovaj nedostatak može da se izbegne upotrebom dva manje uobičajena načina za predstavljanje aritmetičkih izraza sa drugačijim rasporedom operatora i operanada a to su: **prefiksna** i **postfiksna** notacija

Prefiksna i postfiksna notacija

- U **prefiksnoj** notaciji operator se nalazi ispred operanada, a u **postfiksnoj** notaciji oprator se nalazi iza operanada
- Infiksni izray A+B postaje +AB u prefiksnoj notaciji i AB+ u postfiksnoj notaciji
- Prefiksna forma se naziva i **poljskom notacijom**, po poljskom matematičaru Lukasiewicz-u, a postfiksna forma **inverznom poljskom** notacijom
- Razmatraćemo samo postfiksnu notaciju, zbog slčnih osobina ove dve notacije

Infiksna forma u postfiksnu

- Afitmetički izraz u infiksnoj notaciji se pretvara u postfiksni izraz tako sto se mesta između dva operanda premeste iza drugog operanda
- Ovo se najlakše radi ukoliko izraz ima potpune zagrade, jer svaki par zagrada tačno okružuje binarni operator i njegova dva operanda ili unarni operator i jedan njegov operand
- U tom slučaju samo treba svaki operator prebaciti na mesto njemu odgovarajuće desne zagrade, a zatim ukloniti sve zagrade iz izraza

Infiksna forma u postfiksnu

- Npr. Izraz **A+B*(C-D)+(E/F)*G/H** se treba prvo konvertovati u izraz sa potpunim zagradama

  ((A+(B*(C-D)))+(((E-F)*G)/H))

  postaje

  ABCD-*+EF-G*H/+

Infiksna forma u postfiksnu

Za vežbu

1. a+b*c-d/e*f
2. (a+b*c-d)/(e*f)

Izračunavanje postfiksnog 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
```