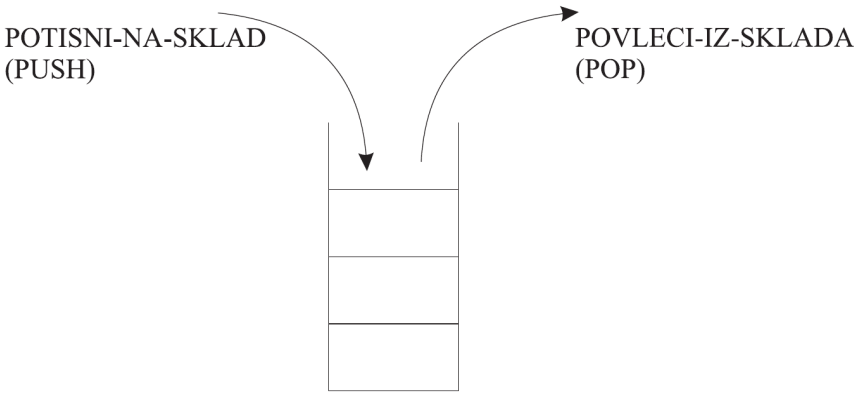
Sklad in krožna vrsta

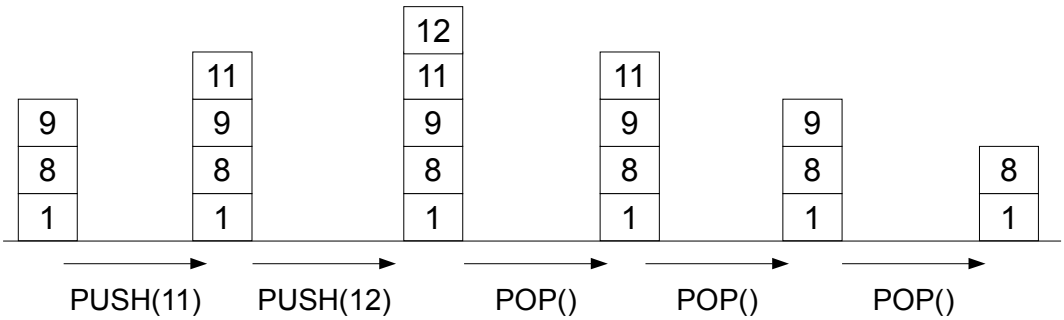
1 Sklad

1.1 Splošna predstavitev problema

Sklad je ena najpomembnejših podatkovnih struktur v računalništvu. Implementira se s pomočjo politike LIFO (*Last In First Out*), kar pomeni, da lahko iz sklada vzamemo samo nazadnje vstavljeni element (slika 1 in 2).

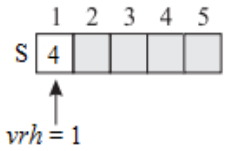


Slika 1: Primer sklada



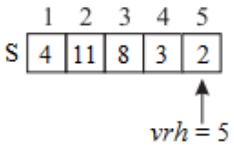
Slika 2: Primer operacij PUSH in POP na skladu

Pomemben je predvsem kot začasna shramba podatkov ter za izvedbo rekurzije. Sklad se lahko uspešno implementira z uporabo navadnega polja.
Vstavljanje v sklad izvedemo z operacijo POTISNI-NA-SKLAD (*PUSH*). Sklad ima lahko največ MAX elementov. Poleg polja potrebujemo še indeks *vrh*. Ko je *vrh* = MAX, vstavljanje v sklad ni več možno. Odstranjevanje elementa s sklada izvedemo z operacijo POVLECI-IZ-SKLADA (*POP*). V primeru, da je *vrh* = 0, je sklad prazen in odstranjevanje elementov ni več možno.
Kot primer vzemimo sklad, kjer ima MAX vrednost 5. Na sklad bi radi shranili vrednosti 4, 11, 8, 3 in 2. Na začetku je sklad prazen, *vrh* ima torej vrednost 0. Nov element lahko vstavljamo samo na vrh sklada. Za tem, ko vstavimo v sklad prvo število, ima *vrh* vrednost 1, kar je vidno na sliki 3.



Slika 3: Sklad S za tem ko vstavimo prvi element

Ko vstavimo v sklad vsa predvidena števila, je indeks *vrh* = 5. Ker je *vrh* dosegel vrednosti MAX, je prišlo do prekoračitve in dodajanje na sklad ni več možno.



Slika 4: Sklad S za tem ko vstavimo vse želene elemente

Iz sklada lahko odstranimo samo element, ki je na vrhu. Če v našem primeru želimo odstraniti element iz seznama, se indeks vrh zmanjša na 4, iz sklada pa se odstrani na zadnje vstavljena vrednost, torej 2.

1.2 Pomoč pri implementaciji

Pri implementaciji sklada je najprej treba definirati polje dolžine MAX ter indeks vrh, ki ga inicializiramo na 0 (-1 v jeziku C).
Implementirati je potrebno še dve funkciji in sicer funkcijo za vstavljanje elementa v sklad ter funkcijo za odstranjevanje iz sklada.

```
procedure PUSH(S, x)
    begin
        if vrh = MAX then
            return prekoračitev;
         else
            vrh := vrh + 1;
            S[vrh] := x;
    end
```

Izpis 1: Psevdokod funkcije PUSH

Funkcija ima dva vhoda, S in x. Prvi predstavlja kazalec na polje medtem, ko drugi predstavlja vrednost, katero želimo dodati na sklad. Če je polje S tako, kot indeks vrh, definirano globalno, kazalca na polje ni treba vnašati v funkcijo.

```
procedure POP(S)
    begin
        if vrh = 0 then
            return napaka;
        else
            vrh := vrh - 1;
            return S[vrh + 1];
    end
```

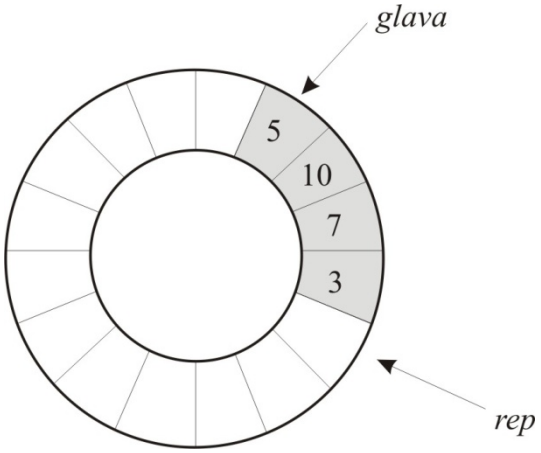
Izpis 2: Psevdokod funkcije POP

Funkcija ima en sam vhod, to je kazalec na polje S. V primeru, da je sklad prazen, funkcija vrne napako, drugače vrne element na vrhu sklada.

2 Krožna vrsta

2.1 Splošna predstavitev problema

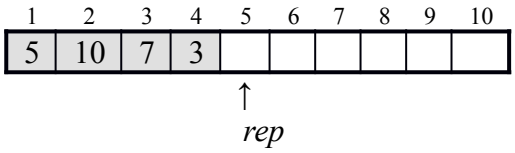
Vrsta je podatkovna struktura tipa FIFO (*First In First Out*), kar pomeni, da iz vrste vedno odstranimo element, ki je bil najdlje v njej. Njena statična implementacija pogosto poteka s pomočjo krožne vrste.
Krožna vrsta je podatkovna struktura, ki se uporablja povsod tam, kjer je pomnilnik omejen. Primer take vrste lahko vidimo na sliki 5.



Slika 5: Primer krožne vrste

Glavna ideja te podatkovne strukture je optimalno izrabiti prostor v pomnilniku, ki ga imamo na voljo. Pri klasični statični implementaciji polja, indeksi tečejo od indeksa 1 pa do indeksa *MAX*. Shranjevanje v polje poteka tako, da hranimo indeks *rep*, to je indeks mesta v polju, v katerega bomo vpisali dano vrednost.

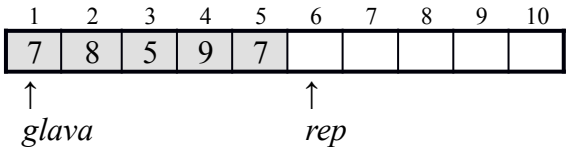
Denimo, da hranimo podatke v klasičnem polju, kjer ima *MAX* vrednost 10. V to polje želimo zaporedoma shraniti vrednosti 5, 10, 7 in 3. Pri tem dobimo situacijo, ki jo prikazuje slika 6.



Slika 6: Hranjenje podatkov v klasičnem polju

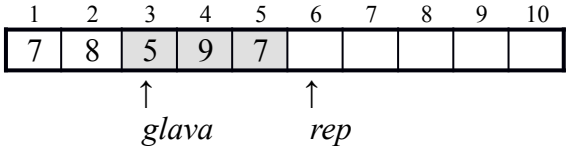
Ko v polje vpišemo 10 števil, je polje polno, *rep* = *MAX* in vpisovanje vanj ni več mogoče. Problem se pojavi, če želimo v polje shranjevati večje število podatkov, ki se skozi čas spreminjajo, pomnilnik pa je omejen. V tem primeru zraven indeksa *rep*, uvedemo še en indeks in sicer indeks *glava*. Pri indeksu *rep* vpisujemo v polje nova števila, pri indeksu *glava* pa števila beremo iz polja. Ko je število iz polja prebrano, ga ne potrebujemo več. Ker nas prebrani podatek ne zanima več, prestavimo indeks *glava* na naslednji element v polju. Za razliko od klasičnega hranjenja podatkov v

polju, kjer je indeks *glava* vedno fiksen in ga, zato zanemarimo, se v primeru krožne vrste le-ta spreminja. Razen indeksa *glava* pa se med uporabo krožne vrste spreminja tudi indeks *rep*, to je indeks prvega praznega elementa v polju, kamor shranimo nov podatek. Ob shranjevanju novega podatka se ta vpiše v prazni element, nato pa se indeks *rep* poveča za ena. Zaradi takega načina shranjevanja podatkov, se lahko območje zasedenosti elementov po polju premika. Poglejmo to na primeru. Denimo, da v prazno krožno vrsto, predstavljeno s poljem (*MAX* = 10), vstavimo števila 7, 8, 5, 9 in še enkrat 7. Dobimo situacijo, ki je prikazana na sliki 7.



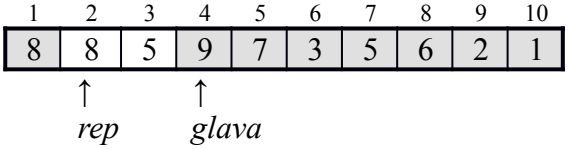
Slika 7: Rezultat vstavljanja petih elementov v krožno vrsto s statično predstavitvijo

Kot lahko vidimo na sliki 7, ima po opravljenih vnosih indeks *glava* vrednost 1, *rep* pa 6. Če sedaj opravimo dve branji iz krožne vrste dobimo situacijo, ki jo prikazuje slika 8.



Slika 8: Stanje v vrsti po branju dveh elementov

Po branju dveh elementov ima indeks *glava* vrednost 3, medtem ko ostaja indeks *rep* nespremenjen. Vidimo pa lahko tudi to, da vrednosti 7 in 8 iz polja fizično nismo brisali, le indeks *glava* smo prestavili za dve mesti.
Da bi predstavili ostale lastnosti krožne vrste predpostavimo, da imamo še eno branje iz vrste, nato pa vpišemo še števila 3, 5, 6, 2, 1 in 8. Rezultat teh operacij je prikazan na sliki 9.



Slika 9: Stanje po brisanju in vstavljanju šestih novih elementov

Na sliki 9 lahko opazimo, da se indeks *rep* po tem, ko doseže vrednost *MAX* spet zmanjša na vrednost 1 in nato ob vnosih spet narašča do vrednosti *MAX*. Pri tem je treba paziti le, da *rep* ne doseže ali celo preseže indeksa *glava*, saj bi v tem primeru izgubljali podatke. Zaradi tega ob vsakem vnosu preverjamo ali je *glava* enaka *rep* + 1. Če to velja, pravimo, da je vrsta polna.

Iz slik 7, 8, 9 pa vidimo, to, da se pri branju povečuje tudi indeks *glava*. In sicer se pri vsakem branju poveča za vrednost 1. V tem primeru pa moramo paziti, da vrednost indeksa *glava* ne doseže vrednosti indeksa *rep*, saj bi to pomenilo napako. Pri tem je treba povedati, da tudi za indeks *glava* velja enako kot za indeks *rep*, da se po prekoračitvi vrednosti *MAX* zmanjša na 1.

2.2 Pomoč pri implementaciji

Pri implementaciji krožne vrste s statičnimi podatkovnimi strukturami je najprej treba definirati polje dolžine *MAX* ter dva indeksa *glava* in *rep*, ki jih inicializiramo na začetno vrednost 1 (vrednost 0 v jeziku C). Nato je potrebno definirati samo še dve funkciji in sicer eno za branje iz krožne vrste, eno pa za pisanje vanjo. Funkcijo za branje prikazuje psevdokod v izpisu 1.

```
procedure BERI(Q)
 begin
   if glava = rep then
     return napaka;
   else
     x := Q[glava];
     glava := (glava mod MAX) + 1;
     return x;
 end
```

Izpis 1: Psevdokod funkcije BERI

Kot lahko vidimo v izpisu 1, ima funkcija en sam vhod, to je kazalec na polje *Q*, v katerem hranimo krožno vrsto. Če je polje, tako kot indeksa *glava* in *rep*, definirano globalno, tega naslova ni treba vnašati v funkcijo. Kot izhod funkcija vrne vrednost, na katero kaže indeks *glava*, katere vrednost je nato potrebno na novo izračunati. Pri tem izračunu uporabimo funkcijo **mod**, ki pomeni ostanek pri deljenju. S tem dosežemo, da se ob prekoračitvi vrednosti *MAX*, vrednost indeksa *glava* zmanjša na 1.

Implementirati je potrebno tudi še funkcijo VPIŠI, ki je predstavljena s psevdokodom v izpisu 2. Ta funkcija ima dva vhodna parametra *Q* in *x*, pri čemer je prvi kazalec na polje z vrsto, drugi pa je vrednost, ki jo v vrsto vpisujemo. Funkcija nima nobenega regularnega izhoda, vrne lahko le napako v primeru, ko je vrsta polna in vpis ni bil mogoč. Tudi tukaj povečujemo indeks *rep* s pomočjo funkcije **mod** in sicer iz enakega razloga kot pri izpisu 1.

```
procedure VPIŠI(Q,x)
 begin
   novi_rep := (rep mod MAX) + 1
   if glava = novi_rep then
     return prekoračitev;
   else
     Q[rep] := x;
     rep := novi_rep;
  end
```

Izpis 2: Psevdokod funkcije VPIŠI

Z implementacijo teh dveh funkcij smo zaključili tudi z implementacijo krožne vrste.