

1. Dinamičke strukture podataka u programskom jeziku PASCAL

Videli smo da se pomoću prostih i strukturiranih tipova podataka može rešiti vrlo široka klasa zadataka. Međutim, postoji i klasa zadataka kod koje tipovi koje smo do sada koristili teško da mogu biti od pomoći. To su zadaci u kojima broj objekata sa kojima se radi nije unapred poznat. Radeći sa nizovima slogova uvek smo morali procenjivati maksimalni broj slogova, da bi prevodilac rezervisao dovoljno memorijskog prostora.

U PASCAL-u postoje takozvane dinamičke promenljive koje se kreiraju i uništavaju u toku izvršavanja programa. One se ne pojavljuju u odeljku za opis promenljivih, tako da im se ne možemo obraćati pomoću unapred definisanih imena. Pristup dinamičkim promenljivim se realizuje samo pomoću pokazivačkih promenljivih.

1.1. Pokazivački tip podataka

Pokazivačka promenljiva - je promenljiva koja "pokazuje" na drugu promenljivu, odnosno sadrži **adresu memorijske lokacije** u kojoj se čuva dinamička promenljiva. Pokazivački tip definišemo na sledeći način:

```
<pokazivacki_tip> ::= ^ <ime_tipa>
```

U samoj definiciji navodi se tip promenljive na koju pokazuje pokazivačka promenljiva. Specijalni znak ^ označava da se radi o pokazivačkom tipu. U sledećem primeru definišemo pokazivački tip:

```
type intpok = ^ integer;
```

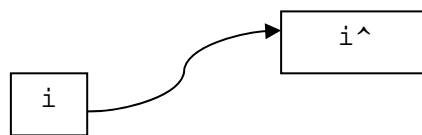
Promenljive tipa `intpok` pokazuju na `integer` promenljive. Ove promenljive deklariramo na potpuno isti način kao i bilo koje druge promenljive:

```
var i: intpok;
```

Možemo takodje pisati:

```
var i:^integer;
```

Da bi pristupili dinamičkoj promenljivoj na koju pokazuje promenljiva `i` koristimo oznaku `i^`. Dinamička promenljiva `i^` ima sve osobine koje ima svaka promenljiva tipa `integer`. Prema tome, njoj se može dodeliti vrednost, može se koristiti u aritmetičkim izrazima, kao parametar prilikom poziva podprograma, itd... Na sledećoj slici je prikazan odnos pokazivačke promenljive `i` (tipa `intpok`) i dinamičke promenljive `i^`.



Nad promenljivima pokazivačkog tipa moguće je vršiti jedino operaciju dodele vrednosti kao i poredjenje pomoću relacije `=`. Nije moguće ispisati vrednost ove promenljive niti je učitati.

Za kreiranje dinamičkih promenljivih se koristi standardna procedura `new`, koja se često naziva procedura dinamičkog razmeštanja. Prilikom poziva procedure `new(i)` izvršavaju se dve akcije:

- rezerviše memorijski prostor za dinamičku promenljivu `i^`;
- dodeljuje pokazivačkoj promenljivoj `i` adresu te promenljive.

Na početku programa promenljiva `i`, kao i sve druge promenljive ima nedefinisanu vrednost. To znači da je njena vrednost neodređena i može biti bilo šta. Pošto je vrednost promenljive `i` adresa, to znači da ova promenljiva na početku programa pokazuje na neku lokaciju u memoriji koja može biti bilo gde.

Ako se pokazivačkoj promenljivoj dodeli konstanta `nil` (`nil` je rezervisana reč), tada ona ne pokazuje ni na koju memorijsku lokaciju.

Treba imati na umu da se pokazivačka promenljiva čija je vrednost `nil` bitno razlikuje od promenljive čija je vrednost nedefinisana. Pokazivačka promenljiva čija je vrednost `nil` može se porediti na jednakost ili nejednakost sa drugom pokazivačkom promenljivom, što je nedozvoljeno za pokazivačku promenljivu čija je vrednost nedefinisana, jer će doći do greške u toku izvršavanja programa.

U PASCAL-u se mora poštovati pravilo: ime bilo kog tipa se mora prvo definisati, pa tek onda- koristiti. Jedini izuzetak je u slučaju definisanja pokazivačkog tipa, koji se može definisati korišćenjem već uvedenog tipa, kao i tipa koji će se kasnije definisati. Na primer,

```
type pokazivac=^t;
    t=...;
```

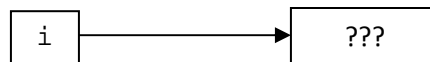
Ako je potrebno da se ukloni beskorisna dinamička promenljiva, i tako oslobodi memorijski prostor, koristi se procedura `dispose`. Na primer, `dispose(pok1)` stavlja na raspolaganje memorijski prostor koji je zauzimala dinamička promenljiva `pok1` na koju pokazuje promenljiva `pok1`. Ovo ne treba shvatiti kao ukidanje promenljive `pok1`. Ona je samo postala nedefinisana, jer više ne pokazuje na neku memorijsku lokaciju.

Zadatak 1. Neka je promenljiva `i` deklarisan kao `intpok`. Šta se dešava posle izvršenja sledećih naredbi:

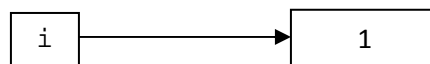
```
new(i);
i^:=1;
i:=nil;
```

Rešenje:

Najpre promenljiva `i` ima nedefinisanu vrednost. Ona pokazuje na neku memorijsku lokaciju.



Naredbom `new` rezerviše se memorijska lokacija za promenljivu `i` i formira odgovarajuća promenljiva `i^`. Njoj se dodeljuje vrednost 1. Prema tome, imamo sledeću situaciju:



Promenljivoj `i` se zatim dodeli vrednost `nil`. Prema tome, ova promenljiva sada ne pokazuje ni na jednu dinamičku promenljivu. Dinamička promenljiva na koju je pre ove naredbe pokazivala promenljiva `i` postaje nedostupna.

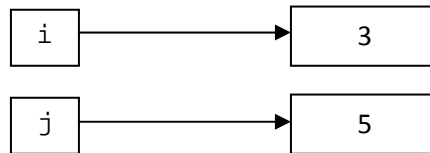


Zadatak 2. Predpostavimo da su promenljive `i` i `j` deklarisan kao promenljive tipa `intpok` i neka su vrednosti promenljivih `i^` i `j^` redom jednake 3 i 5. Razmotriti šta će da se dogodi prilikom izvođenja sledeće dve operacije dodele vrednosti:

- a) $i := j;$
 b) $i^{\wedge} := j^{\wedge};$

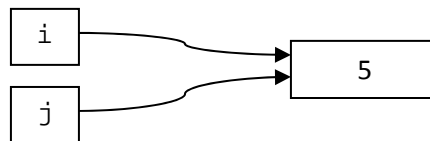
Rešenje:

Na početku imamo sledeću situaciju:



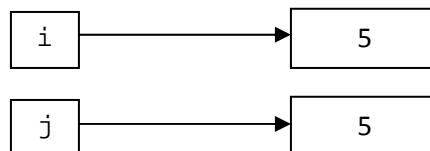
Pokazivači i i j pokazuju na dve različite memorijske lokacije i pritom su vrednosti promenljivih i^{\wedge} i j^{\wedge} redom 3 i 5.

U slučaju a) vrednosti pokazivačkih promenljivih i i j biće jednake. To znači da će ove dve promenljive pokazivati na jednu istu dinamičku promenljivu odnosno na istu memorijsku lokaciju.



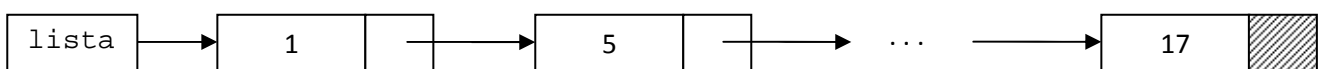
Promenljiva na koju je pre ove operacije pokazivala promenljiva i i dalje postoji (tj postoji rezervisan prostor u memoriji za ovu promenljivu), međutim nijedna pokazivačka promenljiva na nju ne pokazuje. To znači da se ovoj promenljivoj ne može pristupiti. Sa druge strane, na promenljivu na koju je pokazivao pokazivač j sada pokazuju i i i j . Prema tome i^{\wedge} i j^{\wedge} predstavljaju identične promenljive.

U slučaju b) vrednost promenljive i^{\wedge} biće 5 pri čemu će i dalje promenljive i i j pokazivati na dve različite dinamičke promenljive, odnosno na različite lokacije u memoriji.



1.2. Jednostruko povezane liste

Jednostruko povezane liste imaju sledeću strukturu:



Dakle svaki element liste se sastoji iz dva dela. Prvi deo je informacioni deo i sadrži vrednost podatka koji se pamti u listi. Drugi deo predstavlja pokazivač na sledeći element liste. Drugi deo poslednjeg elementa liste ne pokazuje nigde i jednak je `nil`. Na početku liste je pokazivač `lista` koji pokazuje na prvi element.

Ovu strukturu možemo definisati na sledeći način:

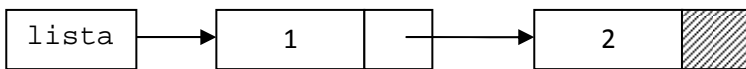
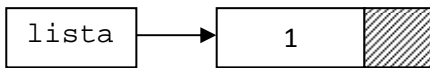
```

type   pok=^slog;
         slog=record
           vr:integer;
           sl:pok;
         end;

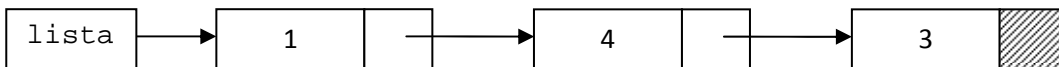
```

Dakle svaki element liste je predstavljen pomoću sloga `slog`. Deo `vr` predstavlja vrednost ovog elementa koja je u našem slučaju tipa `integer` (umesto tipa `integer` može biti bilo koji drugi tip, ili više promenljivih) dok deo `sl` predstavlja pokazivač na sledeći element liste. Sledi primer prazne liste, liste koje ima jedan i liste koja ima 2 elementa:

```
lista=nil
```



Zadatak 3. Neka je data sledeća lista:



Naći vrednost sledećih izraza:

- | | |
|-----------------------------------|-----------------------------------|
| a) <code>lista^.vr</code> | b) <code>lista^.sl^.vr</code> |
| c) <code>lista^.sl^.sl^.vr</code> | d) <code>lista^.sl^.sl^.sl</code> |

Rešenje:

a) Imamo da je `lista^` dinamička promenljiva tipa `slog`. Sa slike vidimo da je njen `vr` deo jednak 1. Prema tome, na ovaj način pristupamo prvom članu liste.

b) Dalje imamo da je `lista^.sl` pokazivač tipa `pok` i da pokazuje na drugi element liste. Prema tome, zaključujemo da je `lista^.sl^.vr=4`.

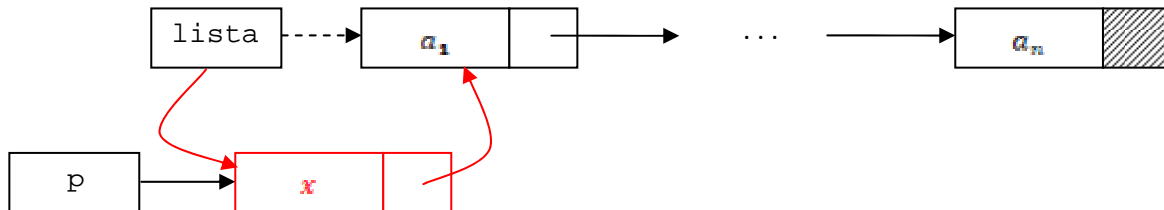
c) Slično dobijamo da je `lista^.sl^.sl^.vr=5`, vrednost trećeg elementa liste.

d) Pošto je treći element liste ujedno i poslednji imamo da je `lista^.sl^.sl^.sl=nil`.

Zadatak 4. Napisati proceduru za dodavanje elementa na pocetak liste. Koristeći ovu proceduru napisati program kojim se formira lista čiji su članovi redom brojevi 10, 9, 8, ..., 2, 1.

Rešenje:

Opišimo proceduru za dodavanje novog elementa x na početak jednostruko povezane liste $lista$. Najpre je potrebno kreirati slog u kome će biti smešten element x . Pomoću naredbe `new` kreira se novi slog u memoriji i njegova adresa upisuje u pokazivač p . U vr deo ovog sloga se upisuje vrednost x . U `sl` deo ovog sloga upisuje se adresa prvog elementa liste koji na ovaj način postaje drugi element. Njegova adresa se pamti u promenljivoj $lista$. Pošto novokreirani slog predstavlja novi početak liste, u pointer $lista$ upisuje se njegova adresa (tj vrednost pointera p). Na sledećoj slici grafički je prikazan postupak ubacivanja novog elementa u listu. Isprekidanim linijama su označene nove vrednosti pokazivača.



Na početku je lista prazna, tj vrednost pokazivača $lista$ je `nil`. Medjutim, predhodno opisana procedura je validna i u ovom slučaju pa ne moramo da ga razmatramo posebno.

Listu čiji su članovi brojevi 10, 9, 8, ..., 2, 1 formiramo primenjujući proceduru `dodajnapoc`. Sledi implementacija ovog rešenja:

```

type pok=^slog;
      slog=record
        vr:integer;
        sl:pok;
      end;

procedure dodajnapoc(var lista:pok;
                    x:integer);
var p:pok;
begin
  new(p);
  p^.vr:=x;
  p^.sl:=lista;
  lista:=p;
end;

var l1:pok;
    i:integer;
begin
  l1:=nil;
  for i:=1 to 10 do
    dodajnapoc(l1,i);
  end.

```

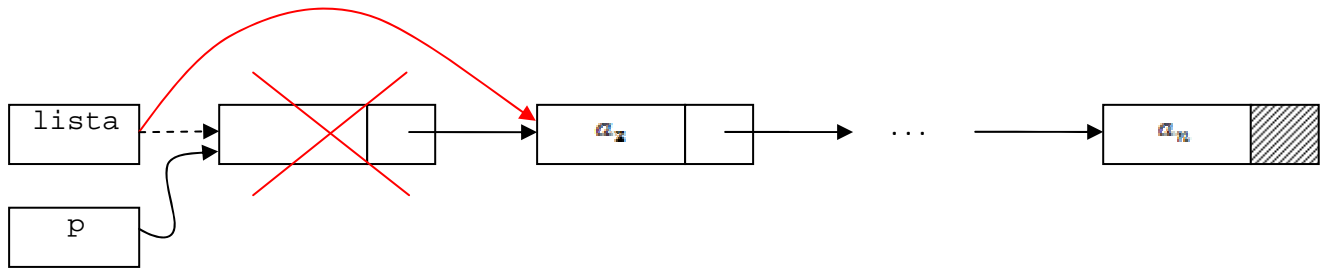
Zadatak 5. Napisati program za formiranje, ispisivanje i brisanje liste svih prostih brojeva od 1 do n .

Rešenje:

Lista se formira na sličan način kao u predhodnom zadatku.

Ispisivanje liste vršimo na sledeći način. Najpre pomoćnom pokazivaču p dodelimo adresu prvog elementa liste. Vrednost prvog elementa liste sada dobijamo kao $p^.vr$. Zatim prelazimo na sledeći element. Pošto je njegova adresa je $p^.sl$, sve što treba je dodeliti promenljivoj p ovu vrednost. Ovaj postupak se nastavlja sve dok se ne dodje do kraja liste odnosno dok je vrednost promenljive p različita od `nil`. Napomenimo da na ovaj način redom pristupamo svakom elementu liste. Način pristupa kod koga da bi pristupili nekom elementu potrebno je pristupiti svim predhodnim elementima nazivamo sekvencijalnim pristupom. Za liste je karakterističan ovaj način pristupa, dok se elementima nizova pristupa direktno.

Sada ćemo opisati postupak kojim se briše jedan element sa početka liste. Posle brisanja, drugi element liste postaje prvi i njegova adresa se upisuje u promenljivu `lista`. Sada je potrebno osloboditi memoriju koja je bila rezervisana za prvi element. Najpre pomoću pokazivača `p` zapamtimo adresu ovog elementa, zatim



promenimo vrednost pokazivača `lista` i na kraju pomoću funkcije `dispose` oslobodimo memorijsku lokaciju koja je rezervisana za prvi element. Primenjujući ovu proceduru više puta, brišemo celu listu. Pritom proceduru primenjujemo sve dok je vrednost pokazivača na početak liste različita od `nil`.

U glavnom programu koristimo funkciju `prost` da bi ispitati da li je broj prost. Ukoliko jeste dodajemo ga na početak liste. Listu zatim ispisujemo i na kraju brišemo. Sledi implementacija ovog rešenja:

```

type pok=^slog;
      slog=record
        vr:integer;
        sl:pok;
      end;

procedure dodajnapoc(var lista:pok;
                    x:integer);
var p:pok;
begin
  new(p);
  p^.vr:=x;
  p^.sl:=lista;
  lista:=p;
end;

procedure ispis(lista:pok);
var p:pok;
begin
  p:=lista;
  while p<>nil do begin
    writeln(p^.vr);
    p:=p^.sl;
  end;
end;

```

```

procedure brisi(var lista:pok);
var p,l:pok;
begin
  p:=l;
  l:=l^.sl;
  dispose(p);
end;

function prost(x:integer):boolean;
var i:integer;
begin
  if x mod 2 <> 0 then
    for i:=3 to trunc(sqrt(x)) do
      if x mod i <> 0 then
        prost:=true
      else
        prost:=false;
  else prost:=false;
end;

begin {main}
  readln(n);
  for i:=1 to n do
    if i=prost then dodajnapoc(i);
  ispis(lista);
  bris(lista);
end {main}.

```

Zadatak 6. Data je jednostruko povezana lista prirodnih brojeva. Napisati proceduru za nalaženje maksimalnog elementa liste i pokazivača na taj element.

Rešenje:

Kao i u predhodnom zadatku i ovde pristupamo svakom elementu liste i tražimo maksimalnu vrednost na uobičajeni način. Pritom, osim same vrednosti maksimalnog elementa, pamtimo i njegovu adresu. Sledi implementacija ovog rešenja:

```

procedure maxp(lista:pok; var max:integer; var pmax:pok);
var p:pok;
begin
    max:= -1
    pmax:=nil;
    p:=lista;
    while p<>nil do begin
        if p^.vr>max then begin
            max:=p^.vr;
            pmax:=p;
        end;
        p:=p^.sl;
    end;
end;

```

Zadatak 7. Implementirati metodu BubbleSort za sortiranje elemenata liste u opadajućem poretku. Kod ove metode, najpre se pronadje maksimalni element niza (liste) i smesti na početak. Zatim se od ostatka niza takodje pronadje maksimalni element pa se on smesti na drugo mesto, itd...

Rešenje:

Koristićemo proceduru maxp iz predhodnog zadatka. Na početku postavljamo vrednost pomoćnog pointera p na adresu prvog element liste. Najpre pronadjemo maksimalni element i zamenimo ga sa prvim elementom. Pritom menjamo vrednosti polja vr prvog i maksimalnog elementa. Zatim p pomeramo jedno mesto udesno tako da on pokazuje na drugi element liste i primenjujemo isti postupak. Postupak prekidamo kada dodjemo do kraja liste, odnosno kada je p=nil. Sledi implementacija ovog rešenja:

```

procedure bubblesort(lista:pok)
var p,pmax:pok;
    tmp,max:integer;
begin
    p:=lista;
    while p<>nil do begin
        maxp(p,max,pmax);

        {maksimum ide napred}
        tmp:=p^.vr;
        p^.vr:=pmax^.vr;
        pmax^.vr:=tmp;

        p:=p^.sl;
    end;
end.

```

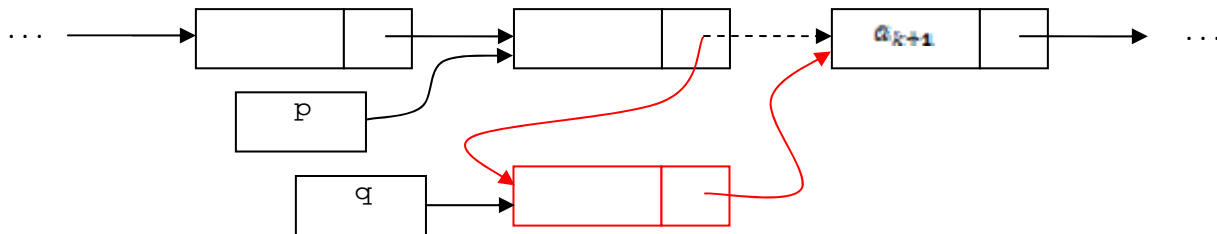
Zadatak 8. Data je sortirana lista (u rastućem poretku) celih brojeva. Napisati proceduru za umetanje datog elementa x u listu tako da ona ostane sortirana.

Rešenje:

Najpre je potrebno naći poziciju elementa x u sortiranoj listi. Ako listu sačinjavaju brojevi $a_1 \leq a_2 \leq \dots \leq a_n$ i ako je tada je potrebno naći element takav da je $a_k < x \leq a_{k+1}$. Element x treba umetnuti između elemenata a_k i a_{k+1} . Pritom moguća su dva slučaja u kojima odgovarajući element a_k ne postoji. Tada je ili $x > a_n$ ili $x < a_1$. U prvom slučaju, element treba dodati na kraj liste, tj posle elementa a_n . U drugom slučaju element dodajemo na početak liste. Ukoliko je lista prazna element možemo ubaciti pomoću procedure `dodajnapoc`.

Prema tome najpre proveravamo da li je lista prazna i da li važi $x \leq a_1$. Ukoliko je bilo koji od predhodna dva uslova ispunjen element x dodajemo na početak liste. U suprotnom tražimo element a_k krećući se pointerom p kroz listu. Sve dok je $p^{.sl}^{.vr} < x$, odnosno dok je x veći i od a_{k+1} tražimo dalje. Ukoliko je postoji. Tada je ili $x > a_n$ doći ćemo do poslednjeg elementa liste odnosno važiće $p^{.sl} = \text{nil}$.

Kada nadjemo element posle koga treba umetnuti x (odnosno pointer na odgovarajući element), vršimo umetanje. Najpre formiramo novi slog (adresu ovog sloga pamtimo u promenljivoj q) u čiji vr deo upisujemo vrednost elementa x . U deo sl novog sloga upisujemo adresu elementa a_{k+1} , odnosno $p^{.sl}$. Na kraju povezujemo element a_k i novi slog tako što u pointer $p^{.sl}$ upisujemo adresu novog elementa.



Primitimo da se slučaj $p^{.sl} = \text{nil}$ odnosno $k = n$ uopšte ne razlikuje od predhodnog i da je izložena procedura i ovde validna. Sledi implementacija ovog rešenja:

```

procedure umetni (var lista:pok; x:integer);
var p,q:pok;
begin
  if lista=nil then begin
    dodajnapoc(lista,x);
    exit;
  end;

  if lista^{.vr} >= x then begin
    dodajnapoc(lista,x);
    exit;
  end;

  p:=lista;
  while (p^{.sl} <> nil) and (p^{.sl}^{.vr} < x) do
    p:=p^{.sl};

```

```

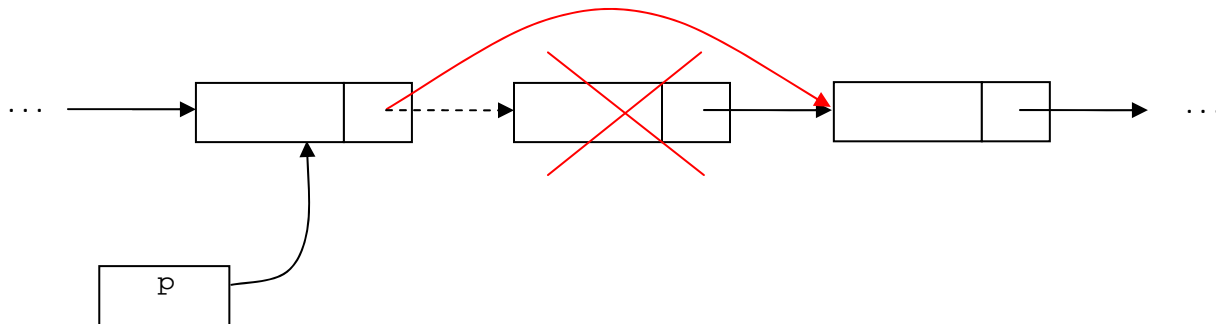
new(q);
q^.vr:=x;
q^.sl:=p^.sl;
p^.sl:=q;
end;

```

Zadatak 9. Napisati proceduru za brisanje elementa iz liste. Procedura ima jedan parametar p koji je pokazivač na predhodni element u listi. Na osnovu ove procedure napisati proceduru kojom se iz date liste list izbacuju svi parni brojevi.

Rešenje.

Tražena procedura je vrlo slična onoj za izbacivanje elementa sa početka liste. Dakle pokazivaču $p^.sl$ (pokazuje na element koji se briše) promenimo vrednost u $p^.sl^.sl$. Predhodno, pokazivač q dobije vrednost $p^.sl$ (pamtimo element koji treba da obrišemo). Nakon toga pozivom funkcije `dispose` oslobadjamo memorijski prostor na koji pokazuje pokazivač q (brišemo element).



Primitimo da $p^.sl$ nikako ne može biti `nil`, dok $p^.sl^.sl$ može. Medjutim, direktnom proverom zaključujemo da čak i u tom slučaju opisana procedura radi. Sledi implementacija ovog rešenja.

```

procedure Izbaci(p:pok);
var q:pok;
begin
  q:=p^.sl;
  p^.sl:=p^.sl^.sl;
  dispose(q);
end;

```

1.3. Stek i red

1.3.1. Stek

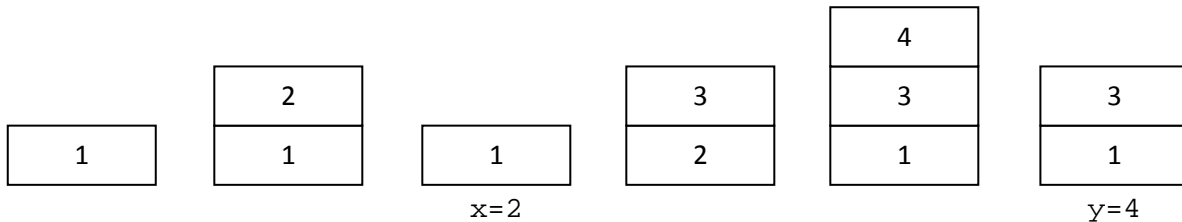
Stek (magacin) je takva struktura podataka iz koje se prvi uzima element koji je poslednji ubačen. To znači da se sa stekom manipuliše prema LIFO principu (Last In First Out). Nad stekom mogu da se izvrše dve

operacije `push` i `pop`. Prva operacija smešta novi element na vrh steka dok druga izbacuje element sa vrha steka.

Primer. Posmatrajmo šta se dešava izvršenjem sledećih operacija nad stekom:

`push(1); push(2); pop(x); push(3); push(4); pop(y);`

Na početku je stek prazan. Prve dve operacije smeštaju redom brojeve 1 i 2 u stek. Treća operacija izbacuje element sa vrha steka (u ovom slučaju to je 2) iz steka i promenljivoj `x` dodeljuje vrednost 2. Naredne dve operacije ubacuju elemente 3 i 4 u stek, dok poslednja operacija izbacuje element sa vrha (sada je to 4) i dodeljuje ga promenljivoj `y`.



Stek možemo implementirati kao jednostruko povezanu listu pri čemu su operacije `push` i `pop` redom operacije dodavanja i brisanja elementa na početak liste odnosno sa početka liste.

Zadatak 9. Definisati odgovarajuću strukturu i procedure `push` i `pop` kojima se implementira struktura podataka stek. Zatim učitati N celih brojeva ($N \leq 100$), smestiti ih u stek, a zatim ih izbacivanjem iz steka ispisati u obrnutom redosledu. Broj N nije unapred poznat a niz brojeva na ulazu se završava 0.

Rešenje:

Definicija tipa stek je potpuno ista kao i definicija jednostruko povezane liste. Takodje operacija `push` je ista kao operacija dodavanja elementa na početak liste a operacija `pop` je ista kao operacija brisanja elementa sa početka liste. Pointer koji označava početak liste sada označava vrh steka.

Za učitavanje brojeva koristi se `while` petlja pošto se ne zna koliko ukupno ima brojeva na ulazu. Brojevi se najpre ubacuju u stek a ona izbacuju. Na taj način se vrši ispisivanje u obrnutom redosledu.

Sledi implementacija ovog rešenja

```

type stek=^slog;
      slog=record
        vr:integer;
        sl:stek;
      end;

procedure push(var vrh:stek;
               x:integer);
var p:stek;
begin
  new(p); p^.vr:=x;
  p^.sl:=vrh;
  vrh:=p;
end;

procedure pop(var vrh:stek;
              var x:integer);
var p:stek;
begin
  if vrh=nil then begin
    x:=-1;
    exit;
  end;
  p:=vrh;
  x:=vrh^.vr;
  vrh:=vrh^.sl;

  dispose(p);
end;

```

```

var x:integer;
    s:stek;
begin
  s:=nil;
  x:=1;
  while (x>0) do begin
    read(x);
    if x>0 then push(s,x);
                                end;
                                readln;
                                while (s<>nil) do begin
                                  pop(s,x);
                                  writeln(x);
                                end;
  end.
end.

```

Zadatak 10*. Sa tastature učitava aritmetički izraz sastavljen od cifara, operacija + i – i zagrada. Napisati program kojim se izračunava vrednost izraza. Smatrati da je uneti aritmetički izraz korektan.

Rešenje:

Zadatak 11. Napisati program koji učitava niz karaktera koji se sastoji od otvorenih i zatvorenih, malih, srednjih i velikih zagrada i proverava da li su zagrade korektno zapisane u unetom nizu.

Rešenje:

1.3.2. Red

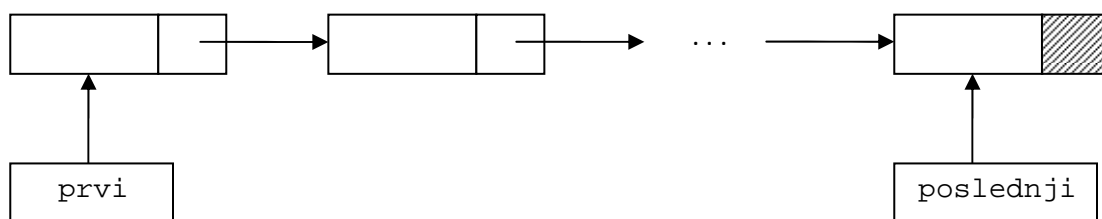
Red je struktura podataka iz koje se prvi uzima element koji je prvi ubačen. Princip za manipulaciju redovima je FIFO (First In First Out). Nad redom mogu da se izvrše operacije put i get. Operacija put ubacuje element u red a operacija get izbacuje jedan element iz reda (tj element koji je prvi ubačen u red).

Primer. Posmatrajmo šta se dešava izvršenjem sledećih operacija nad stekom:

```
put(1); put(2); put(3); get(x); get(y); put(4);
```

Operacijama put ubacujemo odgovarajući element u red. Posle treće operacije put sadržaj reda je: 1 2 3. Operacijom get izbacuje se element iz reda koji je najpre ubačen. U ovom slučaju je to element 1. Posle izbacivanja ovog elementa sadržaj reda je 2 3. Sledeća operacija get izbacuje element 2 pa u redu ostaje samo element 3. Na kraju se dodaje 4 pa je konačni sadržaj reda 3 4.

Strukturu red možemo implementirati pomoću jednostruko povezane liste i grafički predstaviti na sledeći način:



Pokazivač prvi pokazuje na početak reda, tj. na element koji je prvi ubačen u red i koji je sledeći za izbacivanje. Drugi pokazivač, poslednji pokazuje na poslednji element u redu. Prilikom izbacivanja elementa iz reda, izbacuje se element na koga pokazuje prvi i ovaj pokazivač se pomera za jedno mesto udesno. Prilikom

odavanja novog elementa, ovaj element se dodaje iza poslednjeg (tj elementa na koji pokazuje pokazivač poslednji). Ukoliko je red prazan, onda je prvi=poslednji=nil.

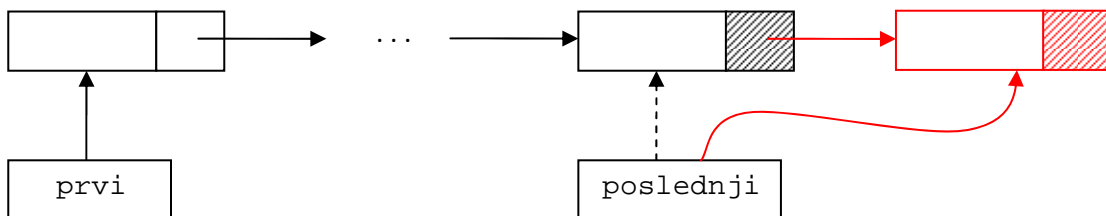
Zadatak 12. Definirati odgovarajuću strukturu i operacije put i get kojima se implementira struktura podataka red. Napisati program kojim se najpre ubacuju u red brojevi u 1, 2, ..., 10 a zatim se izbacuju iz reda i ispisuju na ekran.

Rešenje:

Da bi definisali tip red potrebna su nam dva pointera (prvi i poslednji) i struktura linearne liste. Prema tome, tip red je slogovni tip i sastoji se od dve promenljive tipa pok.

Operacija get predstavlja blagu modifikaciju operacije izbacivanja elementa sa početka linearne liste (ili ekvivalentno operacija pop kod steka). Modifikacija se sastoji u tome da ukoliko je u redu samo jedan element, osim pointera prvi (koji se u tom slučaju postavlja na nil) potrebno je promeniti i pointer poslednji tako da i on pokazuje na nil.

Operacija put se implementira kao operacija dodavanja elementa na kraj jednostruko povezane liste. Da bi dodali element na kraj liste, potrebno je najpre rezervirati memorijski prostor za novi element. To radimo pomoću naredbe new i pomoćnog pokazivača p. Posle naredbe new(p), pokazivač p pokazuje na novi slog. Pošto je ovaj slog sada poslednji u listi (tj redu) njegovom sl delu dodeljuje se vrednost nil, a vr delu vrednost elementa koji ubacujemo. Slog na koji pokazuje pokazivač poslednji sada je predposlednji i njegovom sl delu dodeljuje se adresa novoformiranog sloga (odnosno vrednost pokazivača p). Na kraju se ažurira vrednost pokazivača poslednji tako što mu se dodeli adresa novog poslednjeg elementa liste, u ovom slučaju to je vrednost pokazivača p. Na sledećoj slici je ova operacija predstavljena grafički:



Sledi implementacija strukture, procedura put i get i glavnog programa:

```

type pok=^slog;
  slog=record
    vr:integer;
    sl:pok;
  end;

  red=record
    prvi,poslednji:pok;
  end;

procedure get(var r:red;
              var x:integer);
var p:pok;
begin
  x:=r.prvi^.vr;
  p:=r.prvi;
  r.prvi:=r.prvi^.sl;
  dispose(p);
  if r.prvi=nil then
    r.poslednji:=nil;
  end;

procedure put(var r:red;
              x:integer);
var p:pok;
begin
  new(p);
  p^.vr:=x;
  p^.sl:=nil;
  if r.prvi=nil then begin
    r.prvi:=p;
    r.poslednji:=p;
  exit;

```

```
end;  
r.poslednji^.sl:=p;  
r.poslednji:=p;  
end;
```

```
var r:red;  
    i,x:integer;  
begin
```

```
r.prvi:=nil; r.poslednji:=nil;  
for i:=1 to 10 do  
    put(r,i);  
  
for i:=1 to 10 do begin  
    get(r,x);  
    writeln(x);  
end;  
end.
```

1.4. Dvostruko povezane liste

1.5. Kružno povezane liste

1.6. Zadaci za vežbu

Zadatak 1. Napisati procedure za dodavanje elementa na početak jednostruko povezane liste, kao i za brisanje elementa sa početka. U glavnom programu učitati listu u obrnutom poretku, ispisati prvi i poslednji element liste i obrisati listu.

Zadatak 2. Napisati proceduru za dodavanje elementa na kraj dvostruko povezane liste. U glavnom programu učitati jednu dvostruko povezanu listu, ispisati njene elemente i obrisati listu.

Zadatak 3. Napisati program koji učitava niz karaktera koji se sastoji od otvorenih i zatvorenih, malih, srednjih i velikih zagrada i proverava da li su zagrade korektno zapisane u unetom nizu.

Zadatak 4. Napisati proceduru za sortiranje elemenata jednostruko povezane liste u rastućem poretku. U glavnom programu učitati listu u obrnutom poretku, sortirati je, ispisati elemente sortirane liste i obrisati listu.

Zadatak 5*. (Topološko sortiranje) Neki projekat se sastoji iz N elementarnih operacija. Ove operacije su medjusobno zavisne. Za i -tu operaciju ($i = 1, 2, \dots, N$) poznata je lista operacija koje ne mogu da se izvrše dok se i -ta ne završi (na primer, krov kuće ne može da se sazida pre svih spratova i temelja). Napisati program koji određuje redosled izvršenja operacija koji poštuje sve zavisnosti. Ukoliko postoji više različitih redosleda, potrebno je ispisati bilo koji.

Na ulazu je dat broj N u prvom redu. U $i + 1$ -vom redu je dat niz operacija koje zavise od i -te operacije. Na izlazu je potrebno ispisati jedan od mogućih redosleda izvršenja operacija.

Zadatak 6. Napisati funkciju koja izračunava zbir elemenata na parnim pozicijama date linearne liste. U glavnom programu, učitati elemente liste (u obrnutom poretku) ispisati zbir elemenata na parnim pozicijama i obrisati listu.

Zadatak 7. Napisati funkciju koja izbacuje sve elemente liste koji su prosti brojevi.

Zadatak 8. Napisati funkciju koja testerasto uredjuje datu linearnu listu.