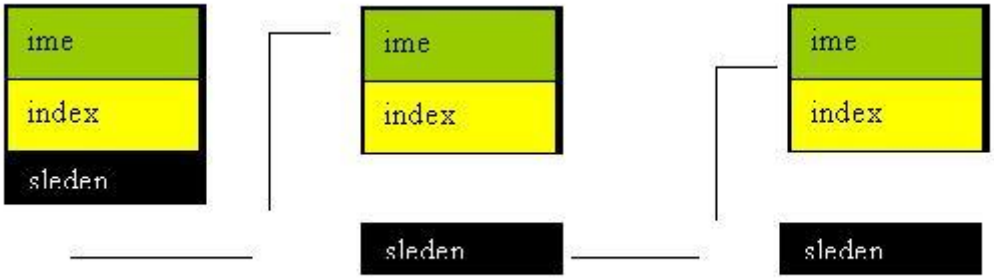
Листи

Динамичката меморија се користи за сместување на големи и сложени типови на податоци, па покажувачките променливи служат за нивна манипулација. Тие можат да покажуваат кон локација од било кој тип на податок вклучувајќи ги и записите.

Запис е множество на податоци. Елементите на записот не се индексирани, туку секој податок има свој идентификатор (име) со кое се пристапува до него. На пример, записот за еден студент нека ги содржи податоците: име, број на индекс и следен. ime е текстуален податок, додека index е целоброен податок. sleden е покажувач од тип pok, кој е дефиниран како покажувач од тип student. Според дефиницијата на student, можеме да креираме листа од записи од ист тип, поврзани со употреба на покажувачи. Листата е илустрирана со следнава слика:



Поврзаната листа е една од основните динамички структури на податоци кои се употребуваат во програмирањето. Се состои од низа од јазли. Секој јазол има дел за сместување податоци и дел за сместување покажувач кој служи како врска кон наредниот јазол од низата. Поврзаните листи овозможуваат вметнување и отстранување на јазли од било кое место во низата за константно време, но не дозволува пристап до јазлите по случаен избор. Односно, секвенцијалниот пристап е ефикасен, меѓутоа директниот не е бидејќи треба да се поминат сите елементи од листата за да се добијат потребните податоци.
Постојат неколку типови на поврзани листи: едностранo поврзани, двострано поврзани и кружни листи.

Низи наспроти поврзани листи

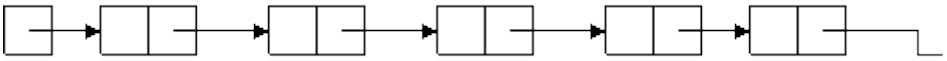
Низите алоцираат меморија за сите нејзини елементи заедно како еден блок. Наспроти тоа листите алоцираат меморија за секој елемент посебно во одделен блок меморија. Тоа значи дека поврзани листи најдобро работат за складирање на низи од податоци на кои не им се знае однапред колкав ќе биде нивниот број, со можност за подоцна да се менува бројот на елементите и нивниот редослед.

Пристапот до елементите кај низите е директен, додека кај листите секвенцијален. Едностраните листи можат да се поминуваат само во еден правец. Ова ги прави поврзаните листи да се неупотребливи каде што е пожелно да се пребаруваат брзо елементите по нивниот индекс. Потребен е дополнителен мемориски простор за референците кај поврзаните листи, што ги прави непрактични за листи со едноставни податоци како char или Boolean.

	Низи	Листи
Пребарување	O(1)	O(n)
Вметнување	O(n)	O(1)
Бришење	O(n)	O(1)

Едностранo поврзани листи

Наједноставниот тип на поврзани листи се едностано поврзаните листи кои имаат по еден покажувач за секој јазол, кој покажува кон наредниот јазол од листата, кон null вредност ако станува збор за последниот јазол од листата или кон празна листа. Анкер (anchor) на листата е покажувач кој покажува кон првиот јазол. За да се промени редоследот на листата доволно е само да се промени вредноста на покажувачот.



Секој јазол од листата има свој *следбеник* (освен последниот) и свој *предходник* (освен првиот). *Должината* на поврзана листа се изразува преку бројот на јазли од кои е составена. Листа која не содржи јазли се нарекува *празна листа*.

Креирањето на поврзани листи во Pascal бара одреден напор, но придобивката од користењето на поврзаните листи е зголемена брзина на работа и поефикасно користење на меморијата.

Да разгледаме практичен пример на креирање едностранo поврзана листа и нејзина манипулација. Нека јазлите содржат записи како тип на податоци, дефинирани на начин што предходно го објаснивме со записот `student`.

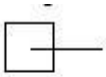
```
Type
     pok=^student;
     student: record
                 ime: string;
                 index: integer;
               sleden: pok;
    end;
```

Ќе започнеме со **процедура за додавање елементи од назад,** која подоцна ќе ја искористиме во програмата и за креирањето на листата. Кодот во Pascal е:

```
Procedure DodadiNazad (Var anker : PLista; nov : PLista);
begin
  if anker = nil then
    anker := nov
  else
    begin
      pom := anker;
      while pom^.sleden <> nil do
        begin
          pom := pom^.sleden;
        end;
      pom^.sleden := nov;
    end;
end;
```

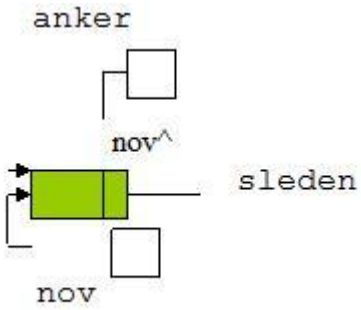
Разгледуваме случај на додавање елементи од назад на: *празна листа* и додавање елементи на *листа со повеќе елементи*.

Дали листата е празна проверуваме со *if anker = nil then*

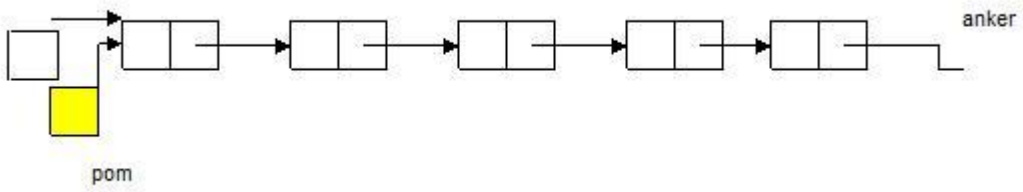


Anker

т.е. ако листата е ***празна,*** *тогаш anker := nov*



Ако не е празна воведуваме нова покажувачка променлива `pom` и ја поставуваме да покажува онаму каде што покажува `anker`, односно на почетокот на листата.

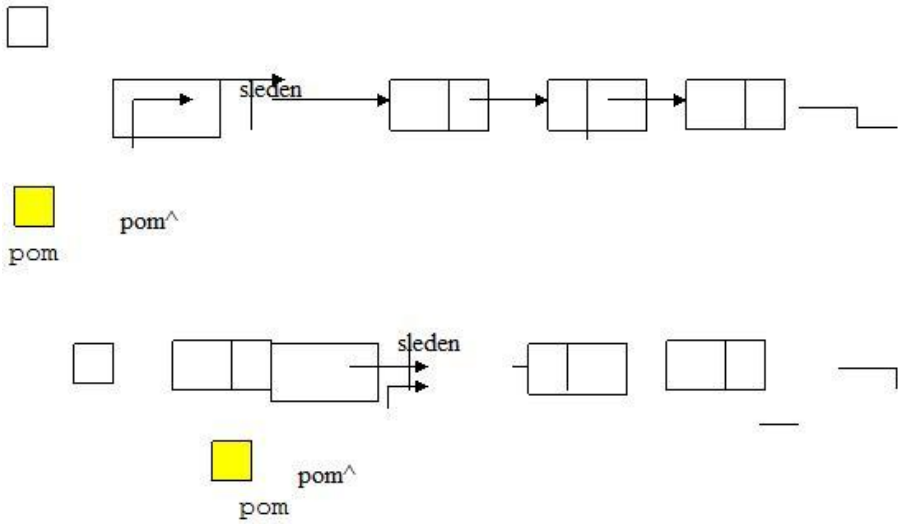


И започнуваме со “шетање” низ листата за да стигнеме до нејзиниот крај, каде ќе го

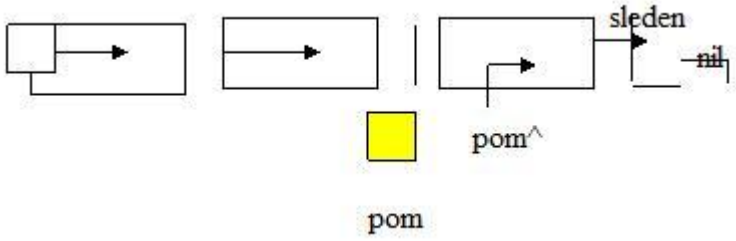
додадеме новиот елемент. Тоа се изведува со програмските линии

while pom^.sleden nil do

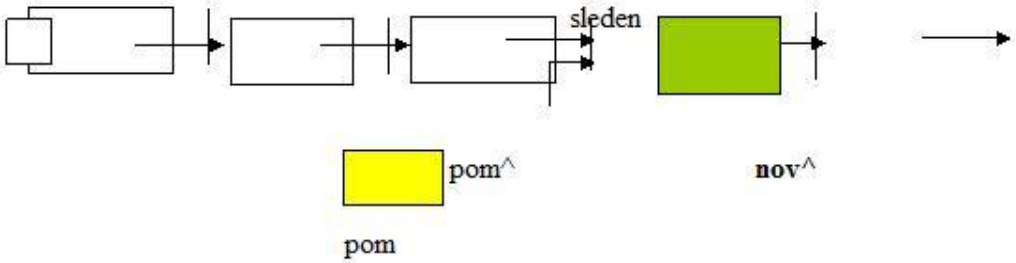
pom := pom^.sleden;



Кога `pom^.sleden=nil` *односно помошниот покажувач ќе покажува на последниот елемент од листата (чиј* `sleden` *покажувач има* nil *вредност),*



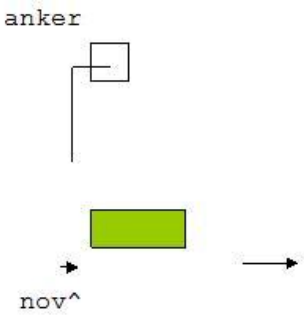
Тогаш со pom^.sleden:=nov ; *се поставува новиот елемент на крајот од поврзаната листа.*



Процедура за додавање на елементи на почетокот на листата:

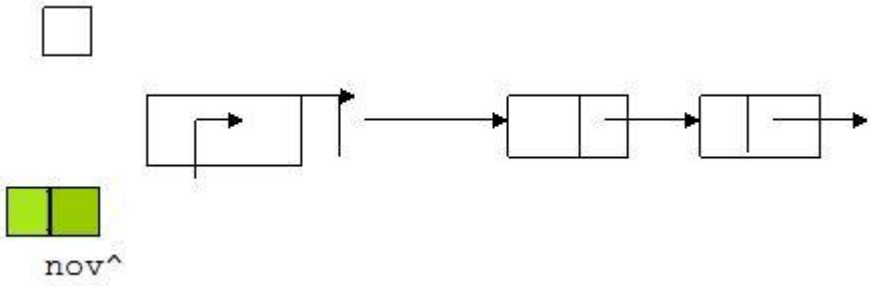
```
Procedure DodadiNapred (Var anker : PLista; nov : PLista);

begin
  if anker = nil then
    anker := nov
  else
    begin
      nov^.sleden := anker;
      anker := nov;
    end;
end;
```

Ако листата е празна, односно главата на листата има вредност nil, ***anker:=nov ;***

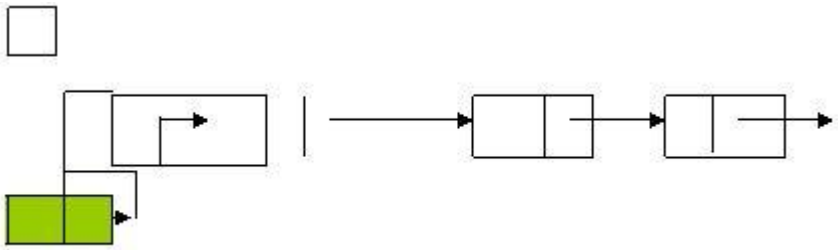


Ако листата има елементи:

nov^.sleden:=anker



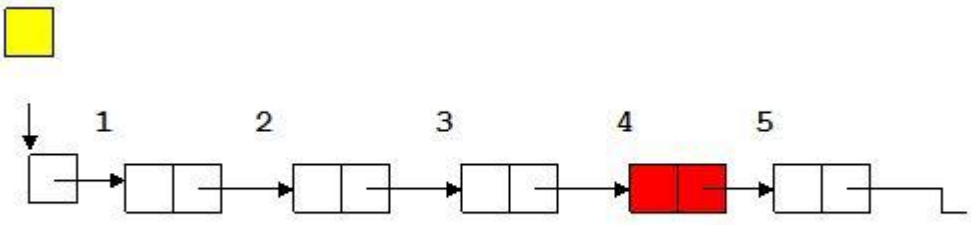
anker:=nov;



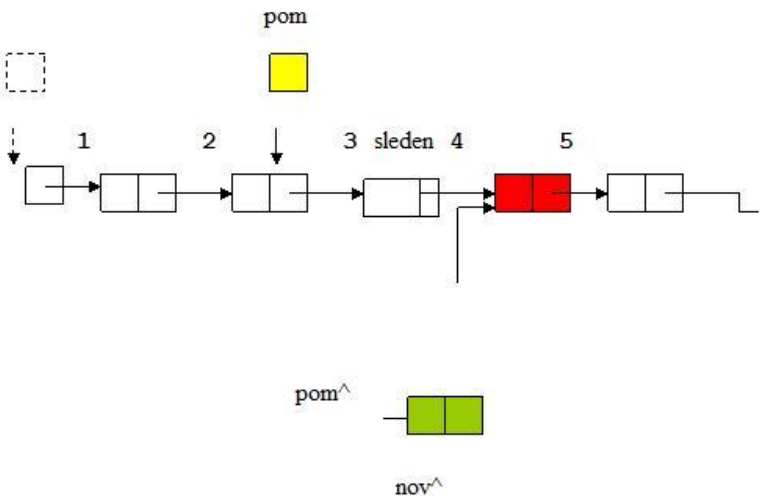
Процедура за додавање на елемент на n-та позиција

```
Procedure DodadiSredina (Var anker: PLista; nov: PLista);
begin

 Writeln('Vnesi go brojot na elementot pred koj sakas da go vmetnes noviot element: ');
 Readln(m);

    pom:=anker;
    for i:=1 to (m-2) do
        pom:=pom^.sleden;

    nov^.sleden:=pom^.sleden;
    pom^.sleden:=nov;
end;
```

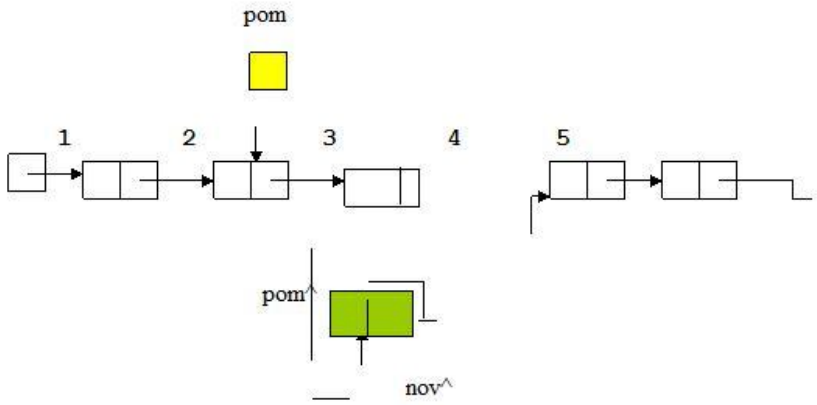
Ако сакаме да го додадеме нов елемент пред m-тиот елемент од поврзаната листа, помошниот покажувач `pom` го поставуваме на почетокот, и почнуваме да „шетаме“ за (m-2) чекори по нејзината должина



Ако сакаме да вметнеме нов пред 4-тиот елемент, го поместуваме помошниот покажувач два пати по должината на листата и застануваме кај неговиот предходник, т.е. на 3-тиот елемент.



Новиот елемент го поставуваме да покажува на исто место со `pom^.sleden` со програмската линија `nov^.sleden:=pom^.sleden`. И потоа правиме "преврзување" на елементите од листата со тоа што `pom^.sleden` го ставаме да покажува на исто место со `nov (pom^.sleden:= nov;)`.

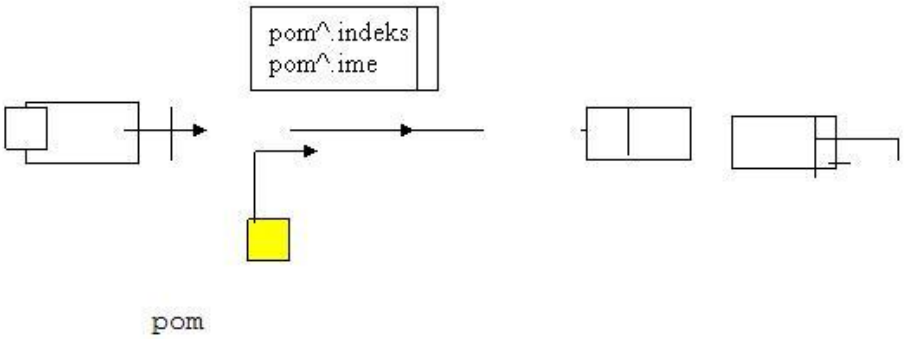


Процедура за листање на елементите од поврзаната листа:

```
procedure BrisiRedenbroj (Var anker : PLista);

begin
 readln(m);
 if m=1 then
   begin
     pom := anker;
     anker := pom^.sleden;
     dispose (pom);
   end
 else
   begin
     pom := anker;
     for i:=1 to (m-2) do
       pom := pom^.sleden;
     pom2 := pom^.sleden;
     pom^.sleden := pom2^.sleden;
     dispose (pom2);
   end;
end;
```

Го поставуваме помошниот покажувач да покажува на почетокот од листата, односно на исто место каде што покажува anker со pom:=anker;

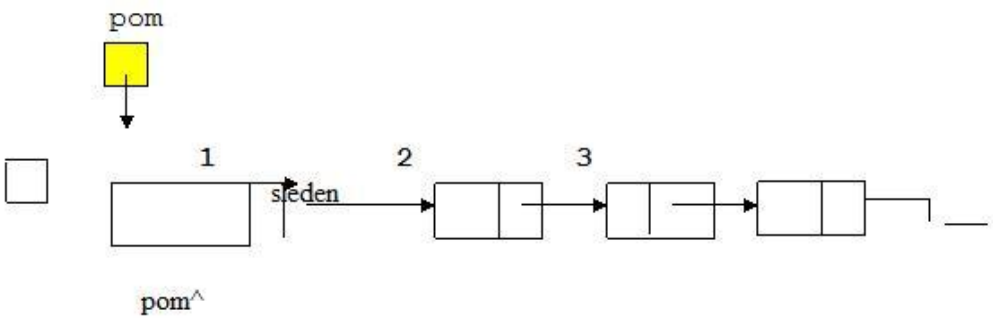
Шетаме низ листата на ист начин објаснет во процедурата DodadiNazad. На секој елемент до кој дошол помошниот покажувач, му пристапуваме до податоците кои ги содржи со pom^.indeks, pom^.ime (во нашиов случај јазлите од листата содржат име и презиме од записот student) и ги печатиме истите.



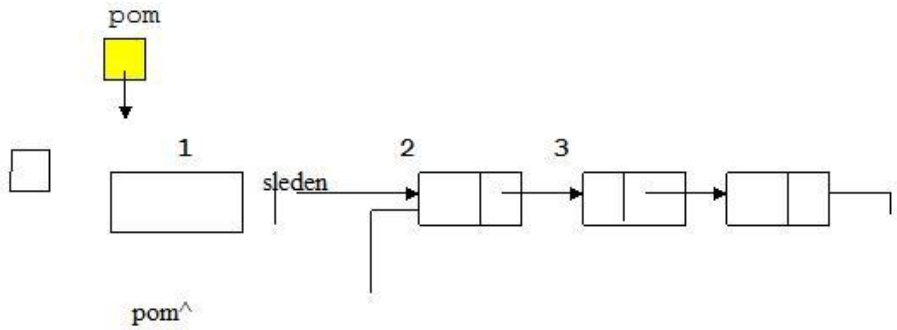
Процедура за бришење на m-ти елемент од поврзана листа:

```
procedure BrisiRedenbroj (Var anker : PLista);

begin
 readln(m);
 if m=1 then
   begin
    pom := anker;
    anker := pom^.sleden;
    dispose (pom);
   end
 else
   begin
    pom := anker;
    for i:=1 to (m-2) do
      pom := pom^.sleden;
    pom2 := pom^.sleden;
    pom^.sleden := pom2^.sleden;
    dispose (pom2);
   end;
end;
```

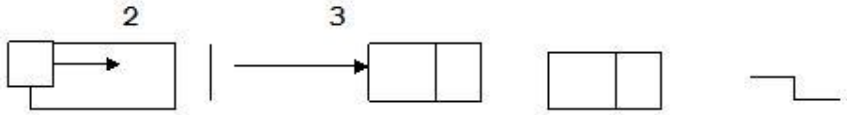
Ако сакаме да го избришеме првиот елемент од листата (m=1), поставуваме помошен покажувач да покажува на почетокот на листата `pom := anker;`



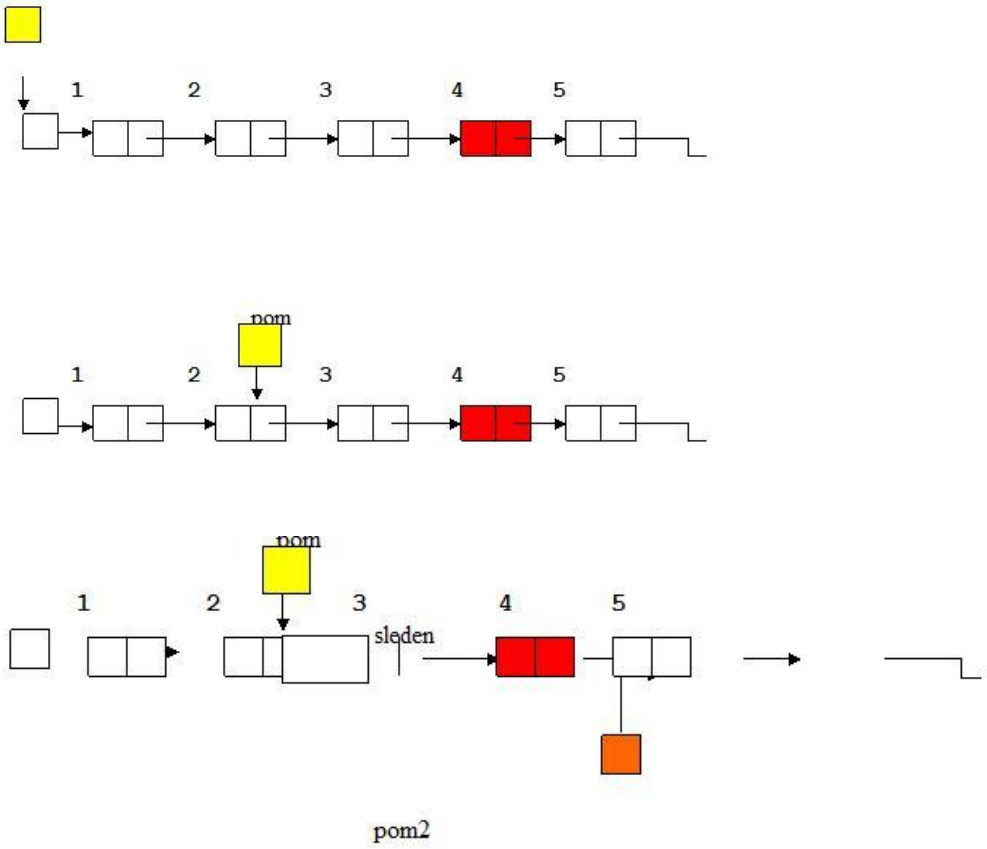
Анкерот се поставува да покажува на исто место со pom^.sleden



И со `dispose(pom)` се остранува првиот елемент со што се ослободува мемориската локација која ја зафаќал тој.

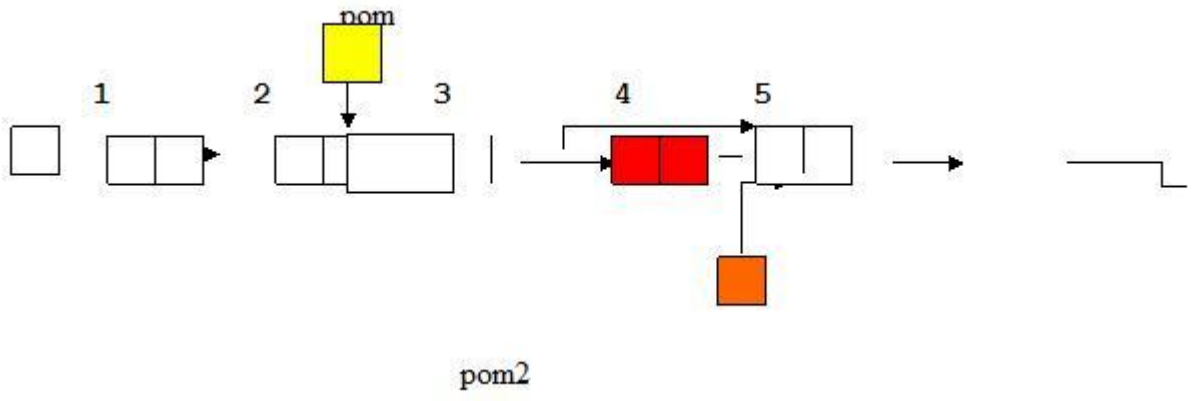


Ако сакаме да го избришеме **m-тиот елемент** од поврзаната листа, помошниот покажувач `pom` го поставуваме на почетокот, и почнуваме да „шетаме“ за (m-2) чекори по нејзината должина. На пример, ако сакаме да го избришеме 4-тиот елемент, од почетокот на листата правиме 2 поместувања на `pom` по должината на листата и застануваме со покажувачот да покажува на еден елемент пред елементот кандидат за бришење. И воведуваме нов помошен покажувач `pom2` кој ќе ни покажува на исто место `co pom^.sleden (pom2:=pom^.sleden).`



На овој начин сме се осигурале дека и со отстранувањето на врската која води од 3-тиот елемент кон 4-тиот, сеуште ќе имаме пристап до сите елементи од листата. Сега извршуваме „преврзување“ со тоа што ќе го

поставиме pom^.sleden да покажува на исто место со pom2^.sleden. Односно врската од 3-тиот елемент која покажуваше кон 4-тиот, сега покажува онаму каде што покажува 4-тиот елемент, а тоа е на 5-тиот.

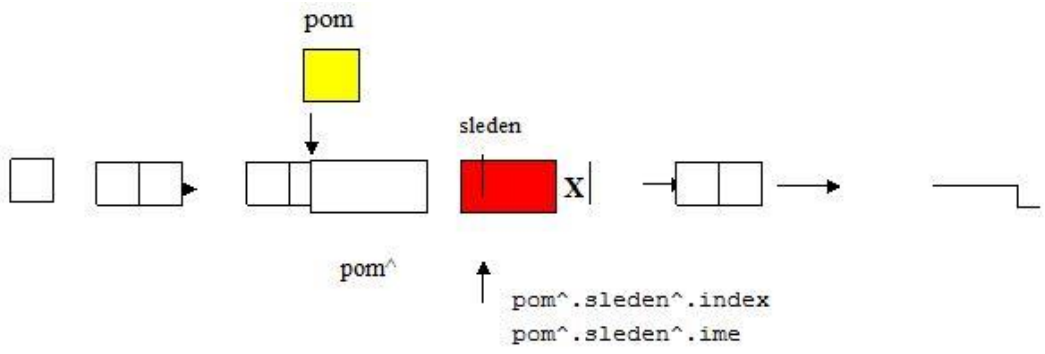


Процедурата за бришење на елемент од листа пребарувајќи го според податокот што го содржи тој елемент е слична со предходно објаснетата процедура:

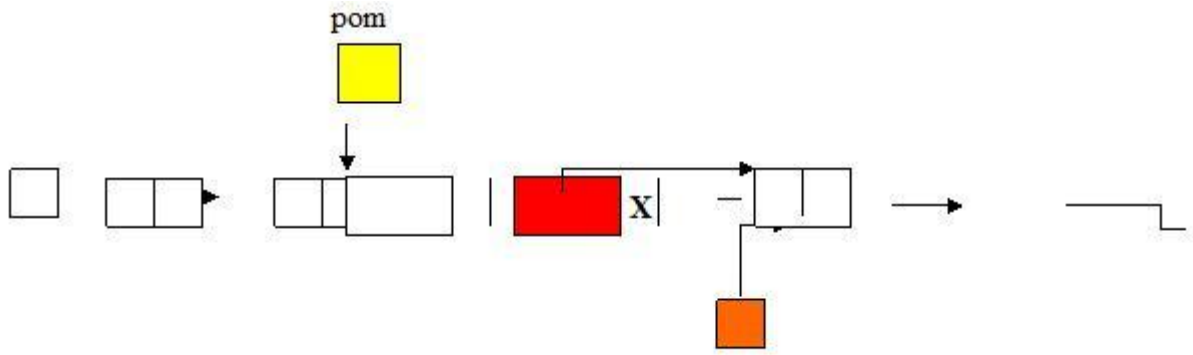
```
Procedure BrisiPodatok (Var anker : PLista; x : integer);
begin

 readln(x);
 writeln;
 if anker^.indeks = x then
   begin
     pom := anker;
     anker := pom^.sleden;
     dispose (pom);
   end
 else
   begin
     pom := anker;
     while pom^.sleden^.indeks <> x do
       pom := pom^.sleden;
     pom2 := pom^.sleden;
     pom^.sleden := pom2^.sleden;
     dispose (pom2);
   end;
end;
```

Помошен покажувач се поставува на почетокот на листата и се шета по листата се додека

pom^.sleden^.indeks x



Кога pom^.sleden^.index ќе биде бараниот податок воведуваме нов помошен покажувач pom2 и постапката е иста како и во предходниот случај.



Комплетна програма за манипулација со едностране поврзана листа:

```
Program PovrzanaLista;
Type
PLista = ^TLista;
TLista = Record
indeks : integer;
ime : String;
sleden : PLista;
end;
Var
anker, nov, pom, pom2 : PLista;
n, i,m, indeks, x, izbor : integer;
ime : String;

Procedure KreirajLista (Var anker:PLista; nov: PLista);
Begin
anker:=nil;
writeln('Kolu elementi ke vnesuvas vo listata:');
readln(n);
writeln;
for i:= 1 to n do
begin
new(nov);
writeln ('Vnesi go brojot na index na ',i, '-tiot element, pa potoa negovoto ime:');
readln(indeks);
readln(ime);
writeln;
nov^.indeks:=indeks;
nov^.ime:=ime;
nov^.sleden:=nil;
if anker = nil then
anker := nov
else
begin
pom := anker;
while pom^.sleden nil do
begin
pom := pom^.sleden;
end;
pom^.sleden := nov;
```

```
end;
end;
end;
Procedure KreirajNov(Var nov: PLista);
begin
new(nov);
writeln('Vnesi go indexot na noviot element:');
readln(indeks);
writeln('Vnesi go imeto na noviot element:');
readln(ime);
writeln;
nov^.indeks:=indeks;
nov^.ime:=ime;
nov^.sleden:=nil;
end;

Procedure DodadiSredina (Var anker: PLista; nov: PLista);
begin
Writeln('Vnesi go brojot na elementot pred koj sakas da go vmetnes noviot element: ');
Readln(m);

pom:=anker;
for i:=1 to (m-2) do
pom:=pom^.sleden;
nov^.sleden:=pom^.sleden;
pom^.sleden:=nov;
end;


Procedure Listaj;
begin
writeln;
writeln('Elementite na listata se:');
writeln;
pom := anker;
while pom nil do
begin
writeln (pom^.indeks, ' ', pom^.ime);
pom := pom^.sleden;
end;
writeln;
end;


Procedure DodadiNapred (Var anker : PLista; nov : PLista);

begin
if anker = nil then
anker := nov
else
begin
nov^.sleden := anker;
anker := nov;
end;
end;
```

```
Procedure DodadiNazad (Var anker : PLista; nov : PLista);

begin
if anker = nil then
anker := nov
else
begin
pom := anker;
while pom^.sleden nil do
begin
pom := pom^.sleden;
end;
pom^.sleden := nov;
end;
end;

procedure BrisiPodatok (Var anker : PLista; x : integer);

begin
writeln('Vnesi go brojot na index na elemenotot sto ke go brises:');
readln(x);
writeln;
if anker^.indeks = x then
begin
pom := anker;
anker := pom^.sleden;
dispose (pom);
end
else
begin
pom := anker;
while pom^.sleden^.indeks x do
pom := pom^.sleden;
pom2 := pom^.sleden;
pom^.sleden := pom2^.sleden;
dispose (pom2);
end;
end;




procedure BrisiRedenbroj (Var anker : PLista);

begin
writeln('Na koe mesto se naogja elementot sto ke go brises:');
readln(m);
Writeln('Izbravte da go brisete ',m,'-tiot element od listata');
writeln;
if m=1 then
begin
```

```
pom := anker;
anker := pom^.sleden;
dispose (pom);
end
else
begin
pom := anker;
for i:=1 to (m-2) do
pom := pom^.sleden;
pom2 := pom^.sleden;
pom^.sleden := pom2^.sleden;
dispose (pom2);
end;
end;



begin
KreirajLista(anker, nov);
Listaj;
writeln;
Readln;
n:=0;
while n<1 do
begin
writeln;
writeln('OBERI OPCIJA OD MENITO:');
writeln;
writeln('1. Kreiraj nov');
writeln('2. Vmetni na pocetok');
writeln('3. Vmetni na kraj');
writeln('4. Vmetni vo sredina');
writeln('5. Listaj');
writeln('6. Brisi element spored sodrzinata');
writeln('7. Brisi element spored mestoto');
writeln('8. KRAJ');

readln(izbor);
writeln;
case izbor of
1: KreirajNov(nov);
2: DodadiNapred(anker, nov);
3: DodadiNazad(anker, nov);
4: DodadiSredina(anker,nov);
5: Listaj;
6: BrisiPodatok(anker, x);
7: BrisiRedenbroj(anker);
8: n:=2;
end;
end;
writeln;
end.
```