SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Seminarski rad iz kolegija Programiranje II

Implementacija dinamičkih struktura podataka

Autor rada: Darko Golner
Br. indeksa: [redacted]

U Varaždinu, siječanj 2001.

Sadržaj:

1. Uvod

Mnogi programski jezici koriste apstraktne tipove podataka. Njih definira programer kako bi unaprijedio postojeće tipove podataka. Takvi tipovi podataka se sastoje od vrijednosti i skupa operacija koji su nad njima definirani. To nam omogućuje pisanje složenih programa pomoću relativno jednostavnih struktura. Njihovo razumijevanje je potrebno svakom profesionalnije orijentiranom programeru, a kada se usvoje, predstavljaju prilično jako oruđe za rješavanje kompleksnih problema. Kako bi se dinamičke strukture podataka približile svima, napisan je ovaj seminarski rad koji osim teorijskog dijela sadrži i praktični dio. U ovom seminarskom radu pobliže su opisane strukture liste, reda, stoga i stabla. Uz opis svake strukture, priložen je i praktični dio. On se sastoji od objašnjenja implementacije dotične strukture kao i od priloženog koda programa koji sadrži operacije nad tom strukturom.

Iako je glavnina priloženog koda napisana u jeziku Pascal, to nije nimalo slučajno. Naime, jezik Pascal ima vrlo mnogo sličnosti s pseudojezikom. To dodatno doprinosi razumljivosti. Osim toga, uz kod u Pascalu je priložen i kod u jeziku C++ koji predstavlja korak naprijed naprednom i profesionalnijem programiranju.

2. Definicija dinamičkih struktura podataka

Ako se struktura podataka za vrijeme izvođenja programa ne mijenja u memoriji, tada se naziva statičkom strukturom. Kao primjer možemo navesti nizove, slogove, skupove i druge. Statičke varijable se deklariraju i pozivaju po imenu. Za vrijeme izvođenja programa statičkoj varijabli se dodjeljuje određena lokacija u memoriji i ta lokacija ostaje rezervirana za sve vrijeme izvođenja programa. Dinamičke varijable se ne deklariraju i ne pozivaju po imenu. Za dinamičke varijable je karakteristično da se mogu kreirati i uklanjati za vrijeme izvođenja programa. Pristupanje dinamičkoj varijabli vrši se pomoću odgovarajućeg pokazivača koji je nastao usporedno s kreiranjem dinamičke varijable. Dinamička varijabla vezana s odgovarajućim pokazivačem naziva se pokazivačka varijabla. Takve se varijable koriste za opis složenih dinamičkih struktura podataka kao što su liste i stabla. Na stranicama što slijede opisat ću strukture liste, stog, red i binarno stablo.

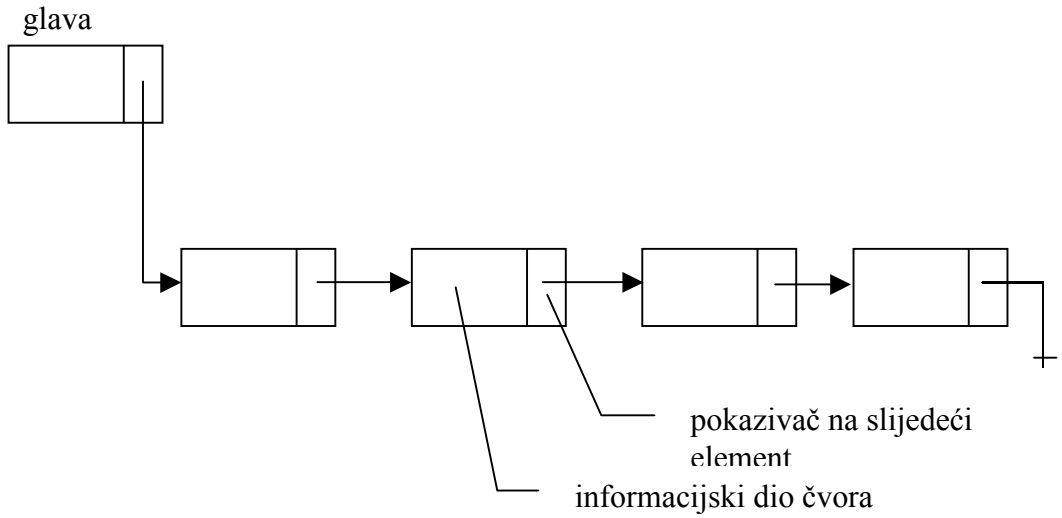
3. Liste

Obrada podataka često uključuje pohranu i obradu podataka organiziranih u linearne liste. Jedan od načina pohrane takvih podataka je pomoću polja. Linearna ovisnost podataka u polju odraz je njihove pozicije u memoriji a ne neke informacije sadržane u samim elementima polja. Zbog toga se može jednostavno izračunati adresa elementa polja. Međutim, upotreba polja za pohranu liste ima i nedostatke : umetanje i brisanje elementa je vremenski zahtjevno. Drugi nedostatk upotrebe polja je taj što unaprijed trebamo znati veličinu polja te se ta veličina ne može mijenjati tokom izvođenja programa.

Drugi način pohrane liste je upotrebom dinamičke strukture podataka. Svaki element liste sastoji se od sadržaja elementa i pokazivača koji sadrži adresu slijedečeg elementa liste. Kod tog načina zapisa liste, susjedni elementi liste ne trebaju biti smješeni na susjednim memorijskim adresama. Takav zapis liste omogućava jednostavnije brisanje i umetanje elemenata u listu.

3.1. JEDNOSTRUKO VEZANE LISTE

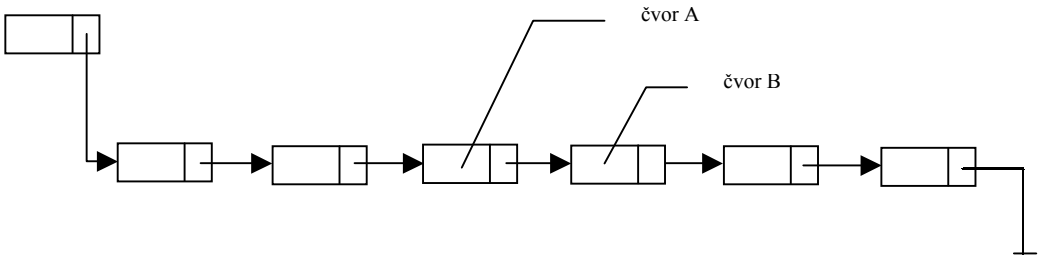
Jednostruko vezana lista je linearni skup podataka, koji se zovu čvorovi, poredanih pomoću pokazivača. Svaki čvor podijeljen je u dva dijela : prvi dio sadržava informacije elementa, a drugi dio sadržava adresu slijedečeg čvora u listi. Pokazivač zadnjeg čvora sadržava null pokazivač. Lista sadrži i pokazivač liste, koji se obično zove glava, koji sadrži adresu prvog člana liste. Posebni slučaj liste je lista bez čvorova. Takva lista se zove null lista ili prazna lista. Jednostruko vezana lista se grafički može prikazati ovako:



Čvor jednostruko vezane liste se najprirodnije realizira kao slog. Jedno polje sloga je pokazivač koji pokazuje na slijedeći čvor vezane liste. Ostala polja određuju sadržaj liste.

Umetanje u linearnu listu

Neka je LISTA linearna lista sa susjednim čvorovima A i B, kao što je prikazano na slici:

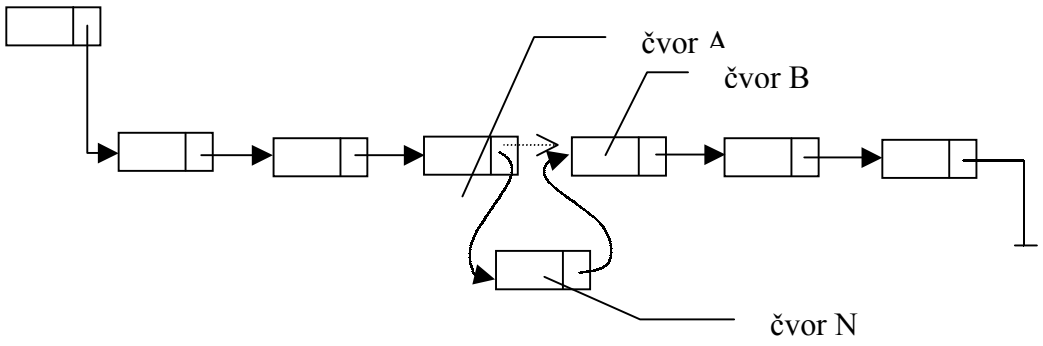


Da bismo ubacili čvor N u listu između čvorova A i B, trebamo:

a) pokazivač čvora A postaviti na adresu čvora N

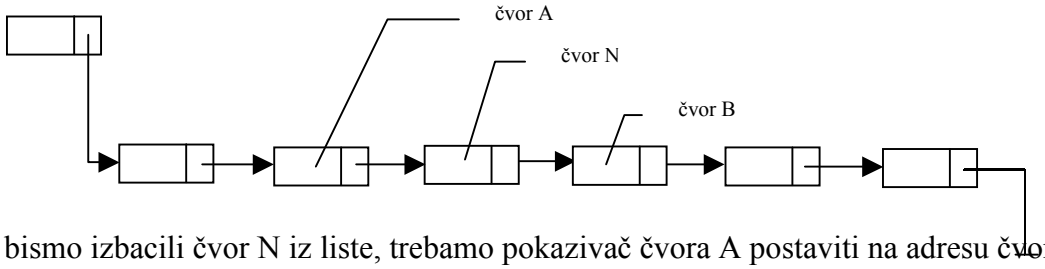
b) pokazivač čvora N postaviti na adresu čvora B

Slika ilustrira umetanje čvora N u listu.



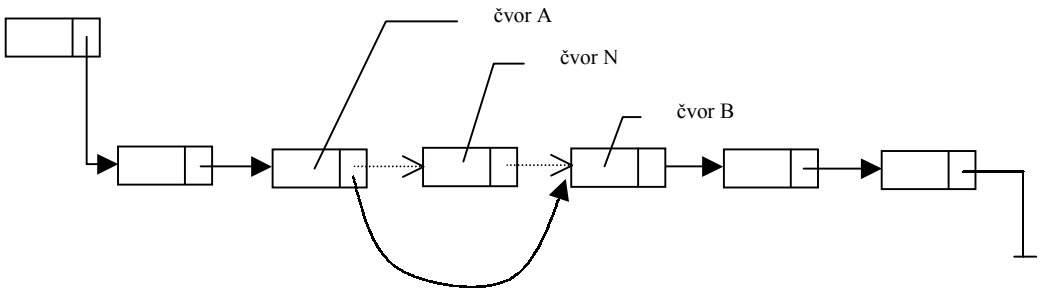
Brisanje iz linearne liste

Neka je LISTA linearna lista sa čvorom N između čvorova A i B, kao što je prikazano na slici:



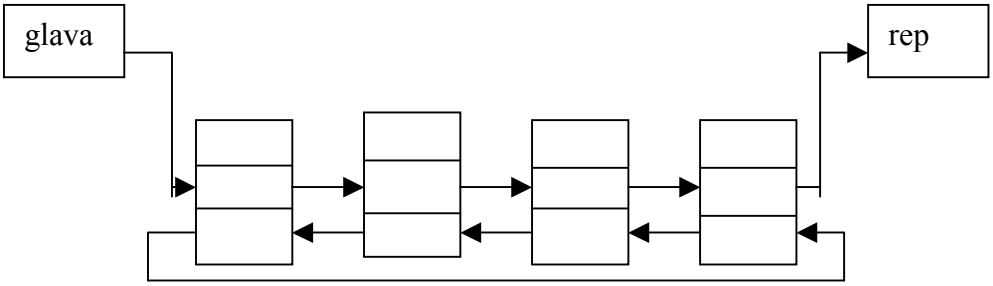
Da bismo izbacili čvor N iz liste, trebamo pokazivač čvora A postaviti na adresu čvora B.

Na slici je prikazana lista nakon brisanja čvora N.



3.2. DVOSTRUKO VEZANE LISTE

Jedna dinamička varijabla može imati dva ili više pokazivača. To svojstvo omogućuje kreiranje složenih struktura podataka. Jedna od njih je dvostruko vezana lista. U dvostruko vezanoj listi prvi pokazivač pokazivačke varijable pokazuje na prethodni čvor liste, a drugi pokazivač pokazuje na slijedeći čvor liste. Prvi pokazivač prvog člana obično pokazuje na poslijednji čvor liste. Zbog takve kružne vezanosti, dvostruko vezana lista se kraće naziva prsten. Dvostruko vezana lista se grafički može prikazati ovako:



Umetanje u dvostruko vezanu listu

Neka je LISTA dvostruko vezana lista sa susjednim čvorovima A i B. Da bismo ubacili čvor N u listu između čvorova A i B, trebamo:

a) Prvi pokazivač čvora A postaviti na adresu čvora N.

b) Prvi pokazivač čvora N postaviti na adresu čvora B.

c) Drugi pokazivač čvora N postaviti na adresu čvora A.

d) Drugi pokazivač čvora B postaviti na adresu čvora N.

Brisanje iz dvostruko vezane liste

Neka je LISTA dvostruko vezana lista sa čvorom N između čvorova A i B. Da bismo izbacili čvor N iz liste, trebamo:

a) prvi pokazivač čvora A postaviti na adresu čvora B

b) drugi pokazivač čvora B postaviti na adresu čvora A

4. Stog

Stog (engl. stack) je dinamička struktura podataka koja se realizira pomoću vezane liste. Čvorovi stoga su istog tipa, a njihov je broj varijabilan. Rad sa stogom se zasniva na upravljanju čvorovima koji se nalaze na vrhu stoga. Podatak se može upisati na stog ili ispisati iz stoga. Stog se koristi na slijedeći način: poslijednji podatak unesen u stog uzima se prvi iz njega. To se naziva LIFO princip (od engl. Last In First Out). Rad sa stogom sastoji se od:

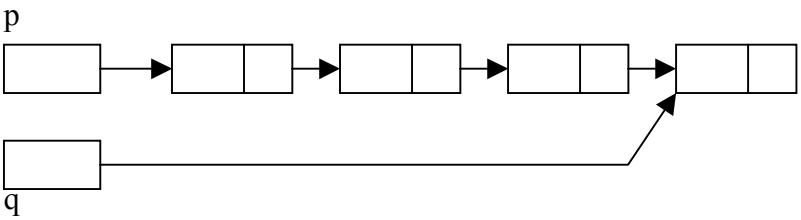
a) definiranju stoga
b) upisu čvorova u stog
c) ispisu čvorova iz stoga

Definiranje stoga sastoji se u definiranju pokazivača stoga i sadržaja stoga. Ako je vrijednost pokazivača stoga NULL (u Pascalu NIL), tada je stog prazan. Ako stog nije prazan, tada pokazivač pokazuje na čvor stoga koji se nalazi na vrhu stoga. Čvorovi stoga su pokazivačke varijable predstavljene slogovima čije jedno polje pokazuje na slijedeći čvor stoga, a drugo polje sadrži vrijednost čvora.

5. Red

Red je struktura podataka slična stogu. Za razliku od stoga, podatak koji je prvi upisan u red prvi se ispisuje iz reda. Takav način upisa i ispisa poznat je kao FIFO princip (od engl. First In First Out). Naziv red za ovakvu strukturu podataka upotrebljen je zbog analogije s redovima iz svakodnevnog života.

Za razliku od stoga, red određuju dva pokazivača. Prvi pokazuje na prvi čvor reda, a drugi na poslijednji čvor reda. Red se grafički može prikazati kao na slici:



6. Binarno stablo

Za razliku od nizova, polja, lista, stoga i reda koji predstavljaju linearne strukture podataka, stablo je nelinearna struktura podataka. *Binarno stablo* T definira se kao konačni skup elemenata, zvanih *čvorovi,* za koje vrijedi :

(1) T je prazno (*prazno stablo* ili *null stablo*)

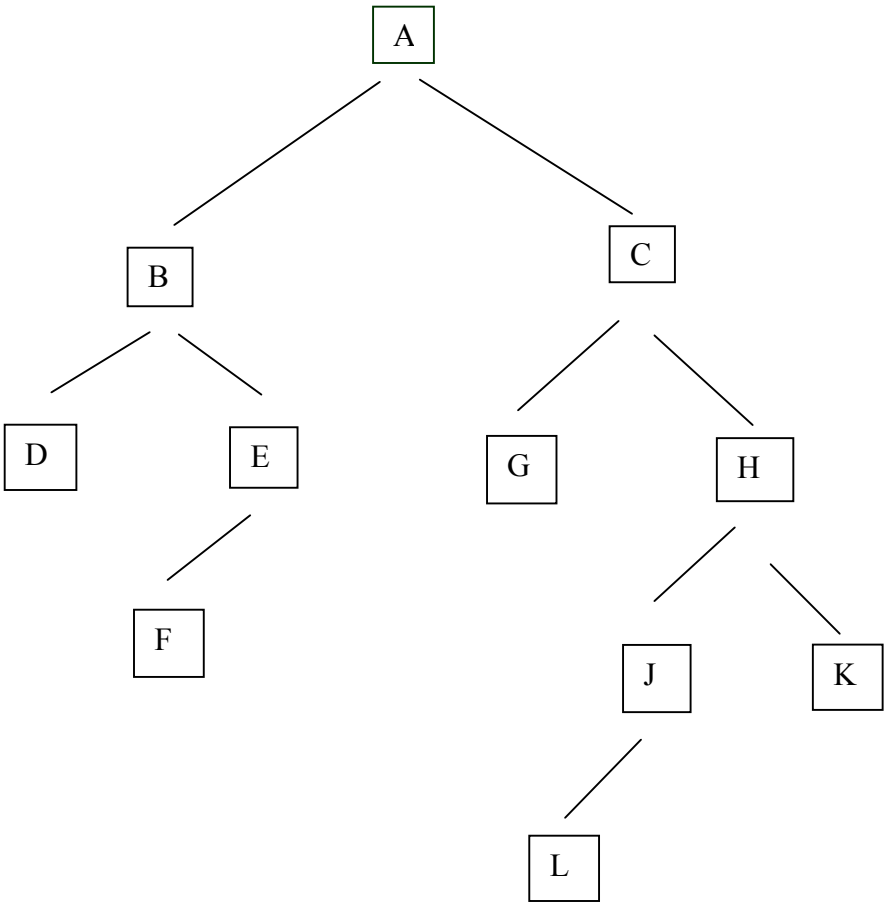
(2) T sadržava posebni čvor R, koji se zove korijen stabla T, i preoste čvorove koji oblikuju uređeni par odvojenih binarnih stabala T_1 i T_2.

T_1 i T_2 nazivamo lijevo i desno podstablo, a čvorove T_1 i T_2 lijevi i desni naslijednik (dijete) čvora T. Binarno stablo obično se prikazuje pomoću dijagrama. Dijagram na slici prikazuje binarno stablo koje : (1) se sastoji od 11 čvorova, prikazanih slovima od A do L. (2) korijen stabla T je čvor A na vrhu dijagrama. (3) lijeva prema dolje usmjerena linija čvora N pokazuje na lijevog naslijednika od N, a desna na desnog naslijednika. Npr.

(a) B je lijevi a C desni naslijednik čvora A.

(b) lijevo podstablo čvora A sastoji se od čvorova B,D,E i F, a desno podstablo od čvorova C,G,H,J,K i L.

Svaki čvor N u binarnom stablu T ima 0,1 ili 2 naslijednika. Čvorovi A, B, C i H imaju dva naslijednika, čvorovi E i J samo jednog, a čvorovi D, F, G i L nemaju naslijednika. Čvorovi bez naslijednika zovu se *krajnji čvorovi* (listovi). Gornja definicija binarnog stabla je rekurzivna budući da je T definirano pomoću podstabla T_1 i T_2.



PRIKAZ BINARNIH STABLA U MEMORIJI

Neka je T binarno stablo. Postoje dva načina za zapis binarnog stabla u memoriji računala. Prvi način zove se vezni prikaz (eng. link representation) i on je analogan načinu prikaza linearnih lista. Drugi način, upotrebom samo jednog polja, zove se sekvencijalni prikaz. Glavni zahtjev kod bilo kojeg prikaza stabla T je da on omogućuje direktan pristup korijenu R i da za bilo koji čvor N omogučuje direktan pristup djeci od N.

Vezni prikaz binarnih stabala

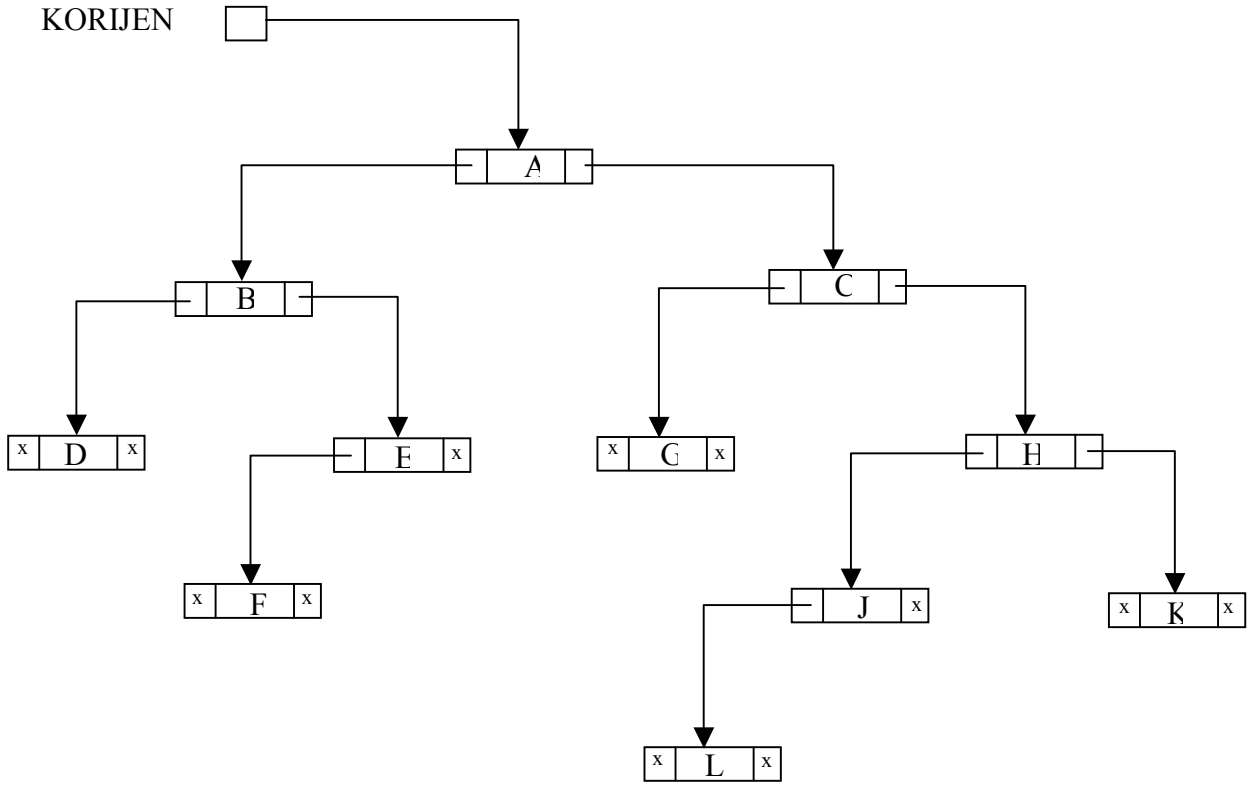
Vezni prikaz koristi tri paralelna polja, INFO, LIJEVI i DESNI i pokazivačku varijablu KORIJEN na slijedeći način :

(1) INFO[K] sadrži podatak čvora N

(2) LIJEVI[K] sadrži adresu lijevog djeteta čvora N

(3) DESNI[K] sadrži adresu desnog djeteta čvora N

KORIJEN sadrži adresu korijena stabla. Ako je bilo koje podstablo prazno, tada odgovarajući pokazivač sadrži null vrijednost. Ako je stablo T prazno, KORIJEN sadrži null vrijednost.



Dijagram prikaza binarnog stabla

Na slici je prikazan shematski dijagram veznog prikaza liste. Svaki čvor sastoji se od tri polja a prazno podstablo označeno je sa x. Slika prikazuje kako bi ta lista bila pohranjena u memoriji računala.

Opaska 1. U danom primjeru svaki čvor saržava samo jedan informacijski element (slovo). U praksi to može biti i čitavi zapis.

Opaska 2. Budući da se čvorovi mogu ubacivati u listu i brisati iz liste, pretpostavljamo da su prazne lokacije u polju INFO, LIJEVO i DESNO povezane u linearnu listu čija je glava DOST. Polje LIJEVO sadrži pokazivač na slijedeći prazni čvor.

Opaska 3. Bilo koja nevažeća adresa može se koristiti kao NULL pokazivač. U praksi se koristi 0 ili negativni broj.

7. Implementacija dinamičkih struktura podataka u Pascalu i C++-u

7.1. Implementacija jednostruko vezane liste

Neka je čvor vezane liste realiziran kao slog od dva polja. Prvo polje sadrži cijeli broj, a drugo sadrži pokazivač na slijedeći čvor. Tip čvorova liste i pokazivača liste definira se na slijedeći način:

```
pokazivac=^cvor;
element=RECORD
            sadrzaj:integer;
            veznik:pokazivac;
        END;
```

Ovdje je pokazivač definiran preko čvora, a čvor preko pokazivača. Tako nam se može činiti da se radi o okretanju u krug, ali navedene kružne definicije su dozvoljene. Kreiranje jedne jednostruko vezane liste pokazat će slijedeći primjer.

Neka su deklarirane slijedeće cjelobrojne varijable i pokazivači tipa pokazivac.

```
VAR a,b,c,u:integer;
    p,q:integer;
```

Ako želimo kreirati vezanu listu s tri čvora (elementa), tada ćemo raditi po slijedećem postupku:

- Najprije se pokazivaču p dodjeljuje vrijednost NIL.

```
p:=NIL;
```

- Kreira se pokazivačka varijabla q pomoću procedure new.

```
new(q);
```

- Zatim se dodjeljuju vrijednosti poljima pokazivačke varijable q.

```
q^.sadrzaj:=a;
q^.veznik:=p;
```

- Ako ponovimo postupak dobit ćemo listu s dva elementa.

```
new(q);
q^.sadrzaj:=b;
q^.veznik:=p;
p:=q;
```

- Nastavimo li taj postupak, dobit ćemo listu s tri elementa.

```
new(q);
q^.sadrzaj:=c;
```

```
q^.veznik:=p;
p:=q;
```

Nakon kreiranja vezane liste, oba pokazivača pokazuju poslijednji element. Nakon što smo kreirali listu, postavlja se pitanje kako pristupiti pojedinim elementima, odnosno čvorovima. Ako pokazivač p pokazuje na prvi čvor liste, tada to znači da pokazuje na poslijednji uneseni element. Naredbom dodjeljivanja:

```
u:=p^.sadrzaj;
```

varijabli se dodjeljuje sadržaj prvog čvora. Sadržaj drugog čvora se dobiva na slijedeći način:

```
u:=p^.veznik^.sadrzaj;
```

Ako je lista dugačka, tada ovakvo pristupanje nema smisla. Potrebno je pretraživati listu.

Neka se npr. traži čvor liste čija je vrijednost d. Uvodi se logička varijabla indikator kojoj je početna vrijednost true. Ako se u listi nađe čvor s istim sadržajem kao varijabla d, tada se indikatoru dodjeljuje vrijednost false. Testiranjem indikatora, provjerava se da li postoji tražena vrijednost. Odgovarajući dio programa izgledao bi tada ovako:

```
q:=p;
indikator:=true;
WHILE (q<>NIL) AND indikator DO
        IF q^.sadrzaj=d THEN indikator:=false;
                        ELSE q:=q^.veznik;
```

Vezana lista ima svojstvo lakog ubacivanja novih elemenata u listu kao i njihovog brisanja. Novi element se ubacuje iza r-tog elementa na slijedeći način:

```
new(q);
q^.sadrzaj:=u;
q^.veznik:=r^.veznik;
r^.veznik:=q;
```

Postupak izbacivanja r-tog elementa je još jednostavniji, samo je potrebno napisati:

```
r^.veznik:=r^.veznik^.veznik;
```

Nakon što smo opisali principe koji vrijede za jednostruko vezanu listu, potrebno je i napisati primjer programa koji to provodi u djelo. Navodim primjer takvog koda:

```
PROGRAM jednostruko_vezana_lista;
{kreira sređenu listu brojeva}
TYPE pokazivac=^element;
   element=RECORD
        veza:pokazivac;
            broj:integer
            END;
VAR glava:pokazivac;
  izbor:char;

PROCEDURE pripremi(VAR iza,ispred:pokazivac; x:integer);
BEGIN {nađi odgovarajuće mjesto u listi}
```

```
iza:=glava; {iza pokazuje na prvi element liste}
WHILE (x>iza^.broj) AND (iza<>NIL) DO
     BEGIN (* putuj kroz listu *)
     ispred:=iza;
     iza:=iza^.veza;
     END;
END;

PROCEDURE ubaci;
VAR novaveza,iza,ispred:pokazivac;
   x:integer;
BEGIN {dodaj novi element u listu}
write('novi broj --> ');
readln(x);
pripremi(iza,ispred,x);
new(novaveza); {novi pokazivač}
novaveza^.broj:=x;
novaveza^.veza:=iza; {poveži novi sa sljedećim}
IF iza=glava THEN glava:=novaveza {ubaci na početak liste}
             ELSE ispred^.veza:=novaveza; {poveži prethodni sa novim}
END;

PROCEDURE brisi;
VAR iza,ispred:pokazivac;
   x:integer;
BEGIN {briši element iz liste}
write('brisem broj -->  ');
readln(x);
pripremi(iza,ispred,x);
IF x=iza^.broj THEN IF iza=glava THEN glava := iza^.veza {izbaci prvi element liste}
                                 ELSE ispred^.veza := ispred^.veza^.veza
          ELSE writeln(' Nisam nasao broj ',x)
END;

PROCEDURE nadji;
VAR kazaljka:pokazivac;
   x,n:integer;
BEGIN {traži zadani broj u listi}
write('trazim broj --> ');
readln(x);
n:=1;
kazaljka:=glava; {kazaljka na početak liste}
WHILE (kazaljka<>NIL) AND (x>kazaljka^.broj) DO
     BEGIN
     kazaljka:=kazaljka^.veza; {putuj kroz listu}
     n:=n+1;
     END;
IF x = kazaljka^.broj THEN {ako si našao}
                writeln(n:3,'--> ',kazaljka^.broj)
                    ELSE writeln('Nisam nasao broj ',x)
END;

PROCEDURE pisi_listu;
VAR kazaljka:pokazivac;
   i:integer;
BEGIN {ispiši sve elemente liste}
i:=0;
writeln;
writeln('Izlazna lista ');
kazaljka:=glava; {kazaljka na početak liste}
```

```
WHILE kazaljka<>NIL DO
    BEGIN {ispiši element liste}
    i:=i+1;
    writeln(i:3,'. --> ',kazaljka^.broj);
    kazaljka:=kazaljka^.veza;
    END;
END;

BEGIN {glavni}
glava:=NIL; {prvi pokazivač}
REPEAT
 writeln;
 REPEAT
 write(' Zelite : B(risati, D(odati,');
 write(' I(spisati, N(aci, Z(avrsiti ? ');
 readln(izbor)
 UNTIL izbor IN ['B','b','D','d','I','i','N','n','Z','z'];
 CASE izbor OF
    'B','b' : IF glava = NIL THEN writeln('Lista je prazna')
                  ELSE brisi;
    'D','d' : ubaci;
    'I', 'i','Z','z' : pisi_listu;
    'N','n' : nadji;
 END;
UNTIL izbor IN ['Z','z'];
END.
```

7.2. Implementacija dvostruko vezane liste

Tip pokazivačke varijable u slučaju dvostruko vezane liste definira se kao:

```
pokazivac=^element;
element=RECORD
        sadrzaj:tipsadrzaja;
        prethodni,slijedeci:pokazivac;
        END;
```

Pokazivač s imenom prethodni pokazuje na prethodni čvor liste. Analogno tome, pokazivač slijedeci pokazuje na slijedeći čvor. Ako se čvor na koji pokazuje pokazivač p upisuje ispred čvora na koji pokazuje pokazivač q, procedura je slijedeća:

```
PROCEDURE upisi_ispred_p(p,q:pokazivac);
BEGIN
p^.prethodni:=q;
p^.slijedeci:=q^.slijedeci;
q^.slijedeci.prethodni^:=p;
q^.slijedeci:=p;
END;
```

Ako se element na koji pokazuje pokazivač p upisuje iza elementa na koji pokazuje pokazivač q, tada je procedura slijedeća:

```
PROCEDURA upisi_iza_q(p,q:pokazivac);
BEGIN
p^.prethodni:=q^.prethodni;
```

```
        p^.slijedeci:=q;
        q^.prethodni.slijedeci^:=q;
        q^.prethodni:=p;
        END;
```

Kao primjer programa koji koristi dvostruko vezanu listu, navodim ovaj:

```
PROGRAM dvostruko_vezana_lista;
TYPE pokazivac=^element;
    element=RECORD
        ispred,iza:pokazivac;
            znak:char;
        END;
VAR prvi,zadnji:pokazivac;

PROCEDURE slozilistu;
VAR novi,p:pokazivac;
   noviznak:char;
BEGIN
writeln('Upisite recenicu: ');
new(novi);
prvi:=novi; {prvi element}
p:=NIL;
REPEAT
read(noviznak);
novi^.znak:=noviznak;
novi^.ispred:=p;
IF p<>NIL THEN p^.iza:=novi;
p:=novi;
new(novi);
UNTIL noviznak='.';
p^.iza:=NIL;
zadnji:=p^.ispred; {zadnji element}
END;

PROCEDURE pisinormalno;
VAR p:pokazivac;
BEGIN
writeln;
writeln('Upisana je recenica: ');
p:=prvi; {pokazivač na prvi element liste}
WHILE p<>NIL DO       BEGIN
                      write(p^.znak);
                      p:=p^.iza; {slijedeći element}
                      END;
writeln;
END;

PROCEDURE pisiobrnuto;
VAR p:pokazivac;
BEGIN
writeln;
writeln('Recenica obrnutim redoslijedom :');
p:=zadnji; {pokazivač na zadnji element liste}
WHILE p<>NIL DO       BEGIN
                      write(p^.znak);
                      p:=p^.ispred {prethodni element}
                      END;
writeln;
END;
```

```
BEGIN {glavni}
slozilistu;
pisinormalno;
pisiobrnuto;
readln;
readln;
END.
```

Primjer programa napisanog u C++-u:

```
//dvostruko vezana lista
#include <iostream.h>

struct element{
        element *ispred,*iza;
        char znak;
};

element *prvi,*zadnji;

void slozi_listu(){
        element *novi,*p;
        char noviznak;
        cout <<"Upisite recenicu: ";
        novi=new element();
        prvi=novi; //prvi element
        p=NULL;
        do{
                cin >>noviznak;
                novi->znak=noviznak;
                novi->ispred=p;
                if (p!=NULL) p->iza=novi;
                p=novi;
                novi=new element();
        } while (noviznak!='.');
        p->iza=NULL;
        zadnji=p->ispred; //zadnji element
};

void pisi_normalno(){
        element *p;
        cout <<"Upisana je recenica: ";
        p=prvi; //pokazivač na prvi element liste
        while (p!=NULL){
                cout <<p->znak;
                p=p->iza; //slijedeći element
        };
        cout <<endl;
};

void pisi_obrnuto(){
        element *p;
        cout <<"Recenica obrnutim redoslijedom: "<<endl;
        p=zadnji;
        while (p!=NULL){
                cout <<p->znak;
                p=p->ispred;
        };
```

```
};

//glavni program
void main(){
        slozi_listu();
        pisi_normalno();
        pisi_obrnuto();
        cout <<endl;
}
```

7.3. Implementacija stoga

Rad sa stogom se sastoji od:

- definiranja stoga
- upisu elemenata u stog
- ispisu elemenata iz stoga

Definiranje stoga se sastoji od definiranja pokazivača stoga i sadržaja stoga. Ako je vrijednost pokazivača NIL, stog je prazan. Ako nije prazan, pokazivač stoga pokazuje na element koji se nalazi na vrhu stoga. Elementi stoga su pokazivačke varijable predstavljene slogovima čije jedno polje pokazuje na slijedeći element stoga, a drugo polje sadrži vrijednost elementa stoga. Elementi stoga mogu se definirati na slijedeći način:

```
TYPE pokazivac=^element;
     element=RECORD
                sadrzaj:tip;
                veza:pokazivac;
                END,
VAR pok:pokazivac;
```

Primjer programa koji rukuje stogom je slijedeći:

```
PROGRAM primjer_stoga;
TYPE pokazivac=^stog;
   stog=RECORD
      podatak:integer;
      veza:pokazivac;
      END;
VAR vrh:pokazivac;
  izbor:char;
  x:integer;
FUNCTION prazan:Boolean;
{provjeri ima li podataka u stogu}
BEGIN
prazan:=(vrh=NIL);
END;

PROCEDURE push(x:integer);
{dodaj podatak na vrh stoga}
VAR p:pokazivac;
BEGIN
new(p);
```

```
p^.podatak:=x;
p^.veza:=vrh;
vrh:=p;
END;

FUNCTION pop:integer;
{uzmi podatak s vrha stoga}
BEGIN
pop:=vrh^.podatak;
vrh:=vrh^.veza;
END;

BEGIN {glavni}
vrh:=NIL;
writeln('Last In First Out');
REPEAT
writeln;
write('D(odaj, U(zmi, K(raj  --> ');
readln(izbor);
CASE izbor OF
    'd','D' : BEGIN
              write(' Podatak : ');
              readln(x);
              push(x);
              END;
    'u','U' : IF NOT prazan THEN writeln('Uzimam : ',pop)
                        ELSE writeln('Nema podataka!!!');
END;
UNTIL izbor IN ['k','K'];
END.
```

Primjer programa napisanog u C++-u:

```
//primjer stoga
#include <iostream.h>

struct element{
          int podatak;
          element *veza;
};

element *vrh;
char izbor;
int x;

//provjeri da li ima podataka na stogu
int prazan(){
          if (vrh==NULL) return(1);
                    else return(0);
};

//dodaj podatak na vrh stoga
void push(int x){
          element *p;
          p=new element();
          p->podatak=x;
          p->veza=vrh;
          vrh=p;
};
```

```
//uzmi podatak s vrha stoga
int pop(){
        element *tmp;
        tmp=new element();
        tmp->podatak=vrh->podatak;
        vrh=vrh->veza;
        return (tmp->podatak);
};

//glavni program
void main(){
        vrh=NULL;
        cout <<"Last In First Out"<<endl;
        do{
                cout <<endl;
                cout <<"D(odaj, U(zmi, K(raj --> ";
                cin >>izbor;
                switch (izbor){
                case 'D' : {
                                                cout <<"Podatak: ";
                                                cin >>x;
                                                push(x);
                                };
                                break;
                case 'U' : if (!prazan()) cout <<"Uzimam: "<<pop()<<endl;
                                                else cout <<"Nema podataka!"<<endl;
                                break;
                };

        }while (izbor!='K');
}
```

7.4. Implementacija reda

Primjer programa koji se služi strukturom reda:

```
PROGRAM primjer_reda;
TYPE pokazivac=^red;
    red=RECORD
       podatak:integer;
       veza:pokazivac;
       END;
VAR prvi,zadnji:pokazivac;
   izbor:char;
   x:integer;
FUNCTION prazan:Boolean;
{provjeri ima li podataka u redu}
BEGIN
prazan:=(prvi=NIL);
END;

PROCEDURE dodaj(x:integer);
{dodaj podatak na kraj reda}
VAR p:pokazivac;
BEGIN
new(p);
p^.podatak:=x;
p^.veza:=NIL;
```

```
IF prvi=NIL THEN prvi:=p; {prvi podatak}
zadnji^.veza:=p;
zadnji:=p;
END;

FUNCTION uzmi:integer;
{uzmi podatak s pocetka reda}
BEGIN
uzmi:=prvi^.podatak;
prvi:=prvi^.veza;
END;

BEGIN {glavni}
prvi:=NIL;
zadnji:=NIL;
writeln('First In First Out');
REPEAT
writeln;
write('D(odaj, U(zmi, K(raj  --> ');
readln(izbor);
CASE izbor OF
   'd','D' : BEGIN
             write(' Podatak : ');
             readln(x);
             dodaj(x);
             END;
   'u','U' : IF NOT prazan THEN writeln('Uzimam: ',uzmi)
                        ELSE writeln('Nema podataka!!!');
END;
UNTIL izbor IN ['k','K'];
END.
```

Primjer programa napisanog u C++-u:

```
//primjer reda
#include <iostream.h>

struct element{
          int podatak;
          element *veza;
};

element *prvi,*zadnji;
char izbor;
int x;

int prazan(){
          if (prazan==NULL) return(1);
                    else return(0);
};

//dodaj podatak na kraj reda
void dodaj(int x){
          element *p;
          p=new element();
          p->podatak=x;
          p->veza=NULL;
          if (prvi==NULL) prvi=p; //prvi podatak
          zadnji->veza=p;
          zadnji=p;
```

```
};

//uzmi podatak s početka reda
int uzmi(){
        element *tmp;
        tmp=new element();
        tmp->podatak=prvi->podatak;
        prvi=prvi->veza;
        return(tmp->podatak);
};

//glavni program
void main(){
        prvi=zadnji=NULL;
        cout <<"First In First Out"<<endl;
        do{
                cout <<"D(odaj, U(zmi, K(raj --> ";
                cin >>izbor;
                switch (izbor){
                case 'D' : {
                                                cout <<"Podatak: ";
                                                cin >>x;
                                                dodaj(x);
                                };
                                break;
                case 'U' : if (prazan()) cout <<"Uzimam: "<<uzmi();
                                                else cout <<"Nema podataka!"<<endl;
                                break;
                };
        } while(izbor!='K');
}
```

7.5. Implementacija binarnog stabla

Jedan čvor uređenog binarnog podstabla može se definirati na slijedeći način:

```
TYPE pokazivac=^cvor;
        cvor=RECORD
                objekt:tippodatka;
                lijevi,desni:pokazivac;
                END;
```

Prikaz programa koji koristi strukturu binarnog stabla bio bi ovakav:

```
PROGRAM binarno_stablo;
TYPE veza=^stablo;
    stablo=RECORD
        broj:integer;
            desno,lijevo:veza;
        END;
VAR korijen,p:veza;
   n:integer;
   izbor:char;
   t:text;
PROCEDURE novicvor (VAR cvor:veza; n:integer);
{stvori novi čvor - podatak}
BEGIN
new(cvor);
```

```
WITH cvor^ DO BEGIN
            broj:=n;
                desno:=NIL;
                lijevo:=NIL
                END;
END;

PROCEDURE gradistablo (VAR p:veza);
{dodaj novi podatak na slobodno mjesto}
BEGIN
IF p=NIL THEN novicvor(p,n)
        ELSE WITH p^ DO
                IF n<broj      THEN gradistablo(lijevo)
                               ELSE gradistablo(desno);
END;

PROCEDURE izlaz (p:veza);
{ispisi elemente binarnog stabla}
BEGIN
IF p<>NIL THEN BEGIN
              izlaz(p^.lijevo);
                write(t, p^.broj:8);
                izlaz(p^.desno);
                END;
END;

BEGIN {glavni}
assign(t,'CON');
rewrite(t);  {ispis na ekran}
korijen:=NIL; {ulaz u stablo}
REPEAT
writeln;
write(' D(odaj, I(spisi, Z(avrsi : ');
readln(izbor);
CASE izbor OF
    'D', 'd' :          BEGIN
                        write(' broj : ');
                        readln(n);
                        novicvor (p,n);
                    gradistablo(korijen)
                        END;
    'I', 'i' : izlaz (korijen);
    'Z', 'z' : BEGIN
               close(t);  {zatvori 'CON'}
               assign(t,'rezultat.dat');
               rewrite(t);  {izlaz u datoteku}
               izlaz(korijen);
               close(t);
               END;
END;
UNTIL izbor IN ['Z','z'];
END.
```

Primjer programa napisanog u C++-u:

```
//primjer binarnog stabla
#include <iostream.h>

struct element{
            int broj;
```

```
        element *desno,*lijevo;
};

element *korijen,*p;
char izbor;
int n;

//stvori novi čvor - podatak
void novi_cvor(element *cvor, int n){
        cvor=new element();
        cvor->broj=n;
        cvor->desno=NULL;
        cvor->lijevo=NULL;
};

//dodaj novi podatak na slobodno mjesto
void gradi_stablo(element *p){
        if (p==NULL) novi_cvor(p,n);
                else if (n<p->broj) gradi_stablo(p->lijevo);
                                else gradi_stablo(p->desno);
};

//ispiši elemente binarnog stabla
void izlaz(element *p){
        if (p!=NULL) {
                izlaz(p->lijevo);
                cout <<p->broj<<endl;
                izlaz(p->desno);
        };
};

//glavni program
void main(){
        korijen=NULL;
        do{
                cout <<"D(odaj, I(spisi, Z(avrsi: ";
                cin >>izbor;
                switch (izbor){
                case 'D' : {
                                                cout <<"broj: ";
                                                cin >>n;
                                                novi_cvor(p,n);
                                                gradi_stablo(korijen);
                                        };
                                        break;
                case 'I' : izlaz(korijen);
                                        break;
                case 'Z' : izlaz(korijen);
                                        break;
                };
        } while(izbor!='Z');
}
```

8. Zaključak vezan uz temu seminarskog rada

Pišući ovaj seminar, imao sam namjeru pobliže objasniti dinamičke strukture podataka kao važan dio znanja o programiranju. Kao što sam u uvodu rekao, dinamičke su strukture važne za rješavanje složenih zadataka. Njihovom upotrebom ne samo da možemo smanjiti količinu koda u programu, nego možemo povećati brzinu rada, odnosno smanjiti vrijeme izvršavanja. Iako se ovdje opisane strukture nazivaju apstraktnima, pokušao sam ih objasniti na manje apstraktan način, tj. kroz primjere. Tako se pridonosi njihovoj razumljivosti, a kada se jednom savladaju, predstavljaju veliku pomoć u programiranju.

Premda nisam opisao baš sve dinamičke strukture podataka, nastojao sam raščlaniti one najvažnije, a nadam se da mi je u tome uspjelo.

9. Literatura

1. B. Motik, J. Šribar: Demistificirani C++, Element, Zagreb, 2000.
2. M. Čubrilo, N. Crnko: Visual Basic Vizualni pristup programiranju, DRIP, Zagreb, 1994.
3. Z. Vlašić: Pascal, Tehnička knjiga, Zagreb, 1991.
4. Bilješke s predavanja iz kolegija Programiranje II, Strukture podataka i Operacijski sustavi
5. Primjeri s Interneta