Baze podataka 1

Osnovni pojmovi struktura podataka

sa realizacijom u programskom jeziku C

Klasifikacija struktura podataka

Klasifikacija 1

- prema dozvoljenom broju neposrednih prethodnika i sledbenika jednog čvora strukture
  - linearne strukture
  - strukture stabla
  - mrežne strukture
- strukture bez konteksta (semantike)
  - bitne samo karakteristike grafa koji može biti
    - linearan, stablo, mreža

Klasifikacija struktura podataka

Klasifikacija 2

- prema nivou apstraktnosti skupa čiji elementi snabdevaju čvorove i ivice grafa semantikom
  - strukture nad skupom obeležja (strukture obeležja)
    - nazivaju i logičkim strukturama
    - visok nivo apstraktnosti
  - strukture nad skupom podataka (strukture podataka) - mogu biti
    - logičke strukture podataka
    - fizičke strukture podataka

Linearne strukture podataka

- linearne strukture podataka - LinSP
- svaki čvor grafa
  - može da ima najviše jednog direktnog prethodnika
  - može da ima najviše jednog direktnog sledbenika
- klasifikacija
  - aciklične – lanci, otvorene liste i nizovi
  - ciklične – zatvorene liste ili prsten

Predstavljanje LinSP

- u zavisnosti od postupka dodavanja i brisanja elemenata, aciklične linearne strukture se nazivaju:
  - n-torka ili vektor (ako se dodavanje i brisanje ne vrši)
  - stek (ako se dodavanje i brisanje vrši samo na jednom kraju)
  - red (ako se dodavanje vrši na jednom, a brisanje na drugom kraju)
  - dek (ako se i dodavanje i brisanje vrši na oba kraja)

Predstavljanje LinSP

- dva osnovna postupka za predstavljanje linearnih struktura podataka u operativnoj memoriji:
  - sekvencijalna reprezentacija - niz
  - spregnuta reprezentacija – lista

Sekvencijalna predstava LinSP

- lokacije operativne memorije su uređene putem celobrojnog adresnog mehanizma
- čvorovi linearne strukture podataka se mogu tako memorisati da se sledećem čvoru može pristupiti jednostavnim povećanjem adrese lokacije tekućeg čvora

Stek

- stek se može sekvencijalno reprezentovati na sličan način kao i niz fiksne dužine
- problem: ograničen broj elemenata
- Metode:
  - pop (skini element sa vrha steka)
  - push (stavi element na vrh steka)
  - peek (pročitaj element na vrhu steka)
  - empty (da li je stek prazan)

Red

- realizacija pomoću niza fiksne dužine
- ponovo se koriste lokacije oslobođene pri čitanju elemenata iz reda, tako da do prekoračenja može doći tek ako treba upisati N-ti element u red
- metode:
  - ubaci
  - čitaj
  - nađi
- referentni pokazivači: *levi* i *desni*
  - *levi*: pokazuje na poziciju sa koje treba pročitati sledeći element
  - *desni*: pokazuje na prvu slobodnu poziciju na koju je moguće upisati sledeći element

Spregnuta predstava LinSP

- svaki čvor strukture se smešta u jednu lokaciju
- lokacija ima jedno polje za smeštanje pokazivača ka lokaciji neposredno narednog čvora
- neposredno susedni čvorovi nisu, u opštem slučaju, u fizički neposredno susednim lokacijama

Spregnuta predstava LinSP

- Stek
  - pokazivač na vrh steka
- Red
  - dva referentna pokazivača: *čelo* i *kraj*
    - *čelo* sadrži adresu lokacije čvora koji treba prvi da bude opslužen (pročitan i eliminisan iz reda)
    - *kraj* sadrži adresu lokacije čvora koji je poslednji upisan u red
  - elementi reda su spregnuti od čela ka kraju

Lista

- zadatak 1
  - spregnuta reprezentacija linearne strukture podataka tipa lista

Zadatak 1

- Napisati C program za evidenciju studenata.
  - Svaki student je opisan:
    - imenom (do 20 karaktera),
    - prezimenom (do 20 karaktera),
    - brojem indeksa (do 10 karaktera) i
    - godinom upisa studija.
  - Program treba da obezbedi sledeće:
    - Unos podataka o studentima na proizvoljnu poziciju u odgovarajuću strukturu,
    - Prikaz svih studenata iz odgovarajuće strukture,
    - Brisanje studenta sa zadatim brojem indeksa, iz odgovarajuće strukture
    - Čitanje podataka o studentima iz datoteke *studenti.txt*, i smeštanje u odgovarajuču strukturu
      - podaci su delimitirani jednim blank znakom
    - Snimanje strukture u datoteku *studenti.txt*.

Čvor dinamičke strukture

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BRINDEKS_SIZE 11
#define IME_SIZE 21
#define PREZIME_SIZE 21

 typedef struct student{
    char ime[IME_SIZE];
    char prezime[PREZIME_SIZE];
    char brindeks[BRINDEKS_SIZE];
     int godupis;
     struct student *next;
 }TStudent;

 int main(){
     TStudent *glava = NULL;
     return 0;
 }
```

Prikaz svih studenata iz dinamičke strukture

```
void prikazStudenata(TStudent* glava){

    puts("PRIKAZ STUDENATA");
    puts("-------------------------------------");

    while(glava){
        printf("Student: %s, %s, %s, %d\n",
         glava->ime, glava->prezime,
         glava->brindeks, glava->godupis);
        glava = glava->next;
    }

    puts("-------------------------------------");
}
```

Broj studenata u dinamičkoj strukturi

```
int brojStudenata(TStudent* glava){

    int brojStd = 0;

    while(glava){
        brojStd++;
        glava = glava->next;
    }

    return brojStd;
}
```

Dodavanje novog čvora (studenta) u dinamičku strukturu na proizvoljnu poziciju

```
void dodajStudenta(TStudent **glava,char *ime, char *prezime,
   char* brindeks, int godupis, int pozicija){
    if((pozicija>=0) && (pozicija<=brojStudenata(*glava)+1)){
        int i;
        TStudent *noviS= (TStudent*)malloc(sizeof(TStudent));
        TStudent *tekuci= *glava, *prethodni = *glava;
        strcpy(noviS->ime, ime);
        strcpy(noviS->prezime, prezime);
        strcpy(noviS->brindeks, brindeks);
        noviS->godupis = godupis;
        noviS->next = NULL;
        for(i = 0; i<pozicija; i++){
            prethodni = tekuci;
            tekuci = tekuci->next;
        }
        noviS->next = tekuci;
        if(tekuci == *glava)
            *glava = noviS;
        else
            prethodni->next = noviS;
    }
}
```

Brisanje studenta iz dinamičke strukture po broju indeksa

```
void obrisiStudenta(TStudent **glava, char *indeks){
    TStudent *tekuci = *glava, *prethodni = NULL,
     *temp = NULL;
    while(tekuci){
        if(!strcmp(tekuci->brindeks, indeks)){
            temp = tekuci;
            tekuci = tekuci->next;
            if(temp == *glava){
                *glava = (*glava)->next;
            }else{
                prethodni->next = tekuci;
            }
            free(temp);
        }else{
            prethodni= tekuci;
            tekuci = tekuci->next;
        }
    }
}
```

Brisanje svih čvorova dinamičke strukture

```
void obrisiStudente(TStudent **glava){

    TStudent *temp;

    while(*glava){
        temp = *glava;
        *glava = (*glava)->next;
        free(temp);
    }
}
```

Učitavanje studenata iz tekstualne datoteke *studenti.txt*

```
void ucitajStudente(TStudent** glava){

    FILE *f = fopen("studenti.txt", "r");
    char ime[IME_SIZE], prezime[PREZIME_SIZE],
     brindeks[BRINDEKS_SIZE];
    int godupis;

    while(fscanf(f, "%s %s %s %d", ime, prezime,
          brindeks,  &godupis) != EOF)
         dodajStudenta(glava, ime, prezime,
          brindeks, godupis,brojStudenata(*glava));

    fclose(f);
}
```

Čuvanje podataka o studentima iz dinamičke strukture u tekstualnoj datoteci *studenti.txt*

```
void sacuvajStudente(TStudent *glava){

    FILE *f = fopen("studenti.txt", "w");

    while(glava != NULL){
       fprintf(f, "%s %s %s %d\n",glava->ime,
        glava->prezime,glava->brindeks,
        glava->godupis);
       glava = glava->next;
    }

    fclose(f);
}
```

Zadatak 2

- Zadata je binarna datoteka *tacke.bin*, koja sadrži koordinate tačke u ravni. Program treba da omogući:
  - Unos nove tačke u odgovarajuću strukturu,
    - Tačke se dodaju u strukturu u neopadajućem redosledu prema udaljenosti od koordinatnog početka.
  - Prikaz svih tacaka iz odgovarajuće strukture,
  - Brisanje tačaka iz strukture koje imaju manju udaljenost od zadate, od koordinatnog početka,
  - Učitavanje koordinata tačaka iz datoteke *tacke.bin* u odgovarajuću strukturu,
  - Snimanje strukture u datoteku *tacke.bin.*

Čvor dinamičke strukture

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct tacka{
    double x, y;
    struct tacka *next;
}Tacka;

int main()
{
    Tacka *glava = NULL;

   return 0;
}
```

Rastojanje tačke od koordinatnog početka

```
double rastojanje (double x, double y){
    return sqrt(x*x + y*y);
}
```

Unos nove tačke u odgovarajuću strukturu

```
void dodajTacku(Tacka** glava, double x, double y){
    Tacka *novaTacka = (Tacka*)malloc(sizeof(Tacka));
    novaTacka->x = x;
    novaTacka->y = y;
    novaTacka->next = NULL;
    Tacka *tek = *glava, *pre = *glava;
    if(*glava){
        while(tek){
            if(rastojanje(novaTacka->x, novaTacka->y)<
               rastojanje(tek->x, tek->y) ){
                novaTacka->next = tek;
                if(tek == *glava)
                    *glava = novaTacka;
                else
                    pre->next = novaTacka;
                break;
            }
            pre = tek;
            tek = tek->next;
            if(tek == NULL)
                pre->next = novaTacka;
        }
    }
    else
        *glava = novaTacka;
}
```

Brisanje tačaka iz strukture koje imaju manju udaljenost od zadate, od koordinatnog početka

```
void obrisiTacku(Tacka **glava, double ras){
    Tacka *tekuci = *glava, *prethodni = NULL, *temp = NULL;
    printf("BRISANJE  TACAKA SA RASTOJANJEM OD (0,0) MANJIM OD  %.2lf\n",
           ras);
    puts("-----------------------------------------------");
    while(tekuci){
        if(rastojanje(tekuci->x, tekuci->y) <= ras){
            temp = tekuci;
            tekuci = tekuci->next;
            if(temp == *glava){
                *glava = (*glava)->next;
            }else{
                prethodni->next = tekuci;
            }
            printf("Brisanje tacke : (%.2lf,%.2lf)\n",tekuci->x,
            tekuci->y);
            free(tekuci);
        }else{
            prethodni= tekuci;
            tekuci = tekuci->next;
        }
    }
    puts("-----------------------------------------------");
}
```

Prikaz svih tačaka

```
void ispisiTacke(Tacka *glava){
    puts("TACKE:");
    puts("--------------------------------");
    while(glava!=NULL){
        printf("Tacka (%.2lf, %.2lf) sa ", glava->x,
         glava->y);
        printf("rastojanjem od  tacke (0,0): %.2lf\n",
         rastojanje(glava->x, glava->y));

        glava = glava->next;
    }
    puts("--------------------------------");
}
```

Brisanje dinamičke strukture

```
void obrisiTacke(Tacka **glava){
    Tacka *temp;
    while(*glava){
        temp = (*glava);
        *glava = (*glava)->next;
        printf("Brisanje tacke : ");
        printf("(%.2lf,%.2lf)\n", temp->x,
         temp->y);
        free(temp);
    }
}
```

Snimanje strukture u datoteku *tacke.bin*

```
void sacuvajTacke(Tacka *glava){
    FILE *f = fopen("tacke.bin", "w");

    while(glava != NULL){
        fwrite(&glava->x, sizeof(double), 1, f);
        fwrite(&glava->y, sizeof(double), 1, f);
        glava = glava->next;
    }

    fclose(f);
}
```

Učitavanje koordinata tačaka iz datoteke tacke.bin u odgovarajuću strukturu

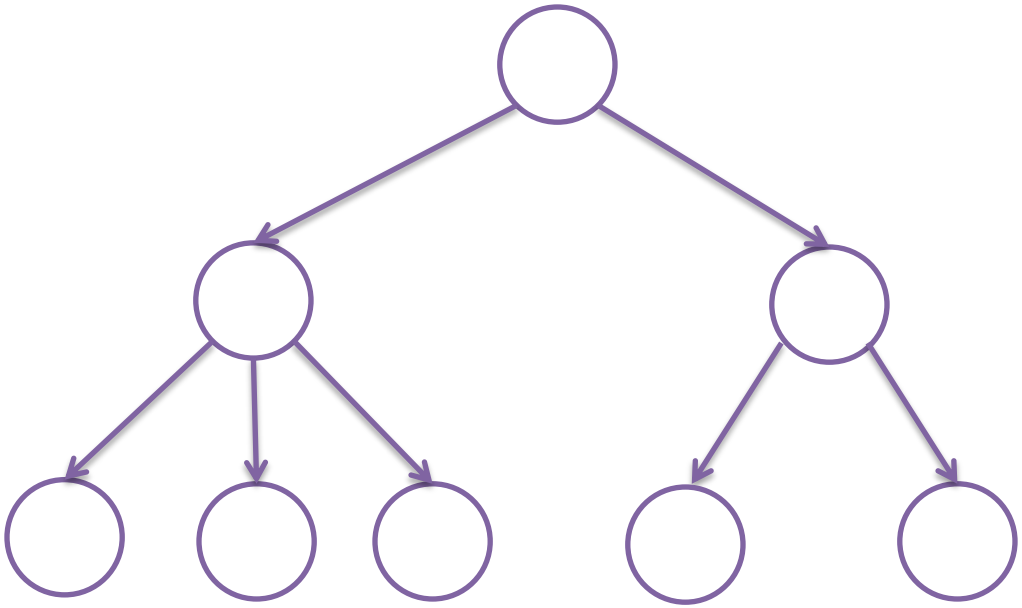
```
void ucitajTacke(Tacka** glava){
    FILE *f = fopen("tacke.bin", "r");

    double posX, posY;

    while(1){
        if((!fread(&posX, sizeof(double), 1, f)) ||
           (!fread(&posY, sizeof(double), 1, f)))
            break;
        dodajTacku(glava, posX, posY);
    }

    fclose(f);
}
```

Zadatak 3

- Proširiti zadatak 1 sledećim funkcionalnostima
  - Prilikom dodavanja novog studenta, omogućiti proveru da li već postoji student sa tim brojem indeksa,
  - Čuvanje podataka o studentima u binarnoj datoteci *studenti.bin,*
  - Čitanje podataka o studentima iz binarne datoteke *studenti.bin* i njihovo smeštanje u odgovarajuću strukturu,
  - Za svakog studenta potrebno je evidentirati broj položenih ispita,
    - omogućiti izmenu broja položenih ispita za studente, na osnovu broja indeksa.

Strukture tipa stabla

- svaki čvor
  - može imati najviše jednog direktnog prethodnika i do n direktnih sledbenika,
    - $0 \leq n \leq N-1$,
    - N - kardinalni broj skupa koji se snabdeva strukturom



Strukture tipa stabla

- strukture stabla su aciklične strukture
- postoji put od najmanjeg do svakog drugog čvora
- postoji bar jedan maksimalan čvor
- broj grana u stablu iznosi N-1

Strukture tipa stabla

- najmanji čvor – koren
- maksimalni čvor – list
- nivovska hijerarhija:
  - koren je čvor prvog nivoa hijerarhije
  - proizvoljan čvor s_i nalazi se na k-tom nivou hijerarhije, ako se nalazi na kraju puta dužine k-1, a put počinje u korenu stabla
- broj nivoa hijerarhije h – visina stabla

Strukture tipa stabla

- red stabla:
  - za stablo se kaže da je n-arno, odnosno reda n, ako svakom čvoru koji nije list odgovara maksimalno n direktno potčinjenih čvorova
  - n=2 – binarno stablo
- puno stablo:
  - svi listovi se nalaze na istom rastojanju od korena (h-1)

Strukture tipa stabla

- kompletno stablo:
  - svi čvorovi, koji ne predstavljaju listove, imaju svih n odlaznih potega, odnosno svih n direktno podređenih čvorova
- balansirano stablo
  - za svaki čvor važi da se broj čvorova u svakom njegovom podstablu ne razlikuje za više od jedan
- optimalno balansirano stablo
  - stablo reda n, čiji su svi čvorovi na nivoima od 1 do h-2 kompletni

Predstava binarnog stabla

- semantika pridružena čvoru stabla smešta se u memorijsku lokaciju
- grana stabla se može reprezentovati bilo putem fizičkog pozicioniranja, bilo putem pokazivača

Sek. predstava binarnog stabla

- ako je binarno stablo statično (čvorovi se ne brišu, a novi se ne dodaju)
- potrebno je rezervisati niz od 2^h-1 memorijskih lokacija
- ako neki čvor na nivou h-1 stabla nema levi ili desni podređeni čvor, odgovarajuća lokacija se ostavlja praznom

Spr. predstava binarnog stabla

- lokacija za smeštanje svakog čvora mora imati najmanje dva polja pokazivača, koja sadrže adrese lokacija u koje su smeštena dva direktno podređena čvora
- za efikasnu realizaciju nekog od postupaka prolaska kroz stablo, potreban je još jedan pokazivač (ka narednom čvoru)

Zadatak 4

- Napisati C program za evidenciju studenata.
  - Svaki student je opisan:
    - imenom (do 20 karaktera),
    - prezimenom (do 20 karaktera),
    - brojem indeksa (do 10 karaktera) i
    - godinom upisa studija.
  - Program treba da obezbedi sledeće:
    - Unos podataka o studentima u niz
    - Sortiranje niza po broju indeksa
    - Formiranje spregnutog balansiranog binarnog stabla na osnovu sortiranog niza, pri čemu se raspoređivanje slogova po čvorovima vrši prema broju indeksa
    - Oslobađanje lokacija zauzetih za potrebe niza i stabla

Čvor dinamičke strukture

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TREE_SIZE_INCREMENT 5

#define BRINDEKS_SIZE 11
#define IME_SIZE 21
#define PREZIME_SIZE 21

typedef struct content{
    char brindeks[BRINDEKS_SIZE];
    char ime[IME_SIZE];
    char prezime[PREZIME_SIZE];
    int godupis;
} TCONTENT;
```

Čvor dinamičke strukture

```
typedef struct linkedtreenode {
   TCONTENT sadrzaj;
   struct linkedtreenode *levi;
   struct linkedtreenode *desni;
} TLINKEDTREENODE;
```

Sortiranje niza

```
void sortirajNiz(TCONTENT *niz, int nniz){
    int i, j;
    int min;
    TCONTENT temp;
    for (i = 0; i < nniz-1; i++){
        min = i;
        for (j = i+1; j < nniz; j++){
            if ( strcmp(niz[j].brindeks, niz[min].brindeks) < 0 )
                min = j;
        }
        temp = niz[i];
        niz[i] = niz[min];
        niz[min] = temp;
    }
}
```

Prikaz sadržaja

```
void stampajSlog(TCONTENT *slog, int i){
    printf("\n%d:", i+1);
    printf("\n\t %s %s %s %d", slog->brindeks,
     slog->ime, slog->prezime, slog->godupis);
}

void stampajNiz(TCONTENT *niz, int nniz){
    int i;
    for (i = 0; i < nniz; i++)
        stampajSlog(niz+i,i);
}
```

Formiranje stabla

```
TLINKEDTREENODE* formirajStablo(TCONTENT *niz,
  int nniz, TLINKEDTREENODE **stablo){
    formirajCvorStabla(niz, 0, nniz-1, stablo);
    return *stablo;
}
```

Formiranje stabla

```
void formirajCvorStabla(TCONTENT *niz, int levi,
  int desni, TLINKEDTREENODE **cvorstabla){
    if( levi > desni){
        *cvorstabla = NULL;
        return;
    }
    int srednji = (levi+desni)/2 + (levi+desni)%2;
    *cvorstabla =
     (TLINKEDTREENODE*)malloc(sizeof(TLINKEDTREENODE));
    (**cvorstabla).sadrzaj = niz[srednji];
    formirajCvorStabla(niz, levi, srednji-1,
     &(**cvorstabla).levi);
    formirajCvorStabla(niz, srednji+1, desni,
     &(**cvorstabla).desni);
}
```

Prikaz čvorova stabla

```
void prikaziStablo(TLINKEDTREENODE *stablo){
    //BREADTH FIRST
    TLINKEDTREENODE **cvorovi=
     (TLINKEDTREENODE**)malloc(TREE_SIZE_INCREMENT*
     sizeof(TLINKEDTREENODE*));
    int *roditelji=
     (int*)malloc(TREE_SIZE_INCREMENT*sizeof(int));
    int *nivoi=
     (int*)malloc(TREE_SIZE_INCREMENT*sizeof(int));
    int n=TREE_SIZE_INCREMENT;
    int start=0,end=0;
    cvorovi[start]=stablo;
    roditelji[start]=-1;
    nivoi[start]=0;
    end++;
    //...
```

Prikaz čvorova stabla

```
// ... void prikaziStablo(TLINKEDTREENODE *stablo){
    int k=0;
    TLINKEDTREENODE *ltn=NULL;
    TCONTENT *c=NULL;
    while(start!=end){
        ltn=cvorovi[start];
        c=&ltn->sadrzaj;
        stampajSlog(c,k++);
        printf("\n\t adresa: %p",ltn);
        printf("\n\t adresa-levi: %p",ltn->levi);
        printf("\n\t adresa-desni: %p",ltn->desni);
        printf("\n\t adresa-roditelj: %p",
         roditelji[start]!=-1?
         cvorovi[roditelji[start]]:NULL);
        printf("\n\t nivo: %d",nivoi[start]);
        //...
```

Prikaz čvorova stabla

```
// ... void prikaziStablo(TLINKEDTREENODE *stablo){
        if( ltn->levi != NULL || ltn->desni != NULL){
           if(end==n){
               n+=TREE_SIZE_INCREMENT;

               cvorovi=(TLINKEDTREENODE**)realloc
                (cvorovi,n*sizeof(TLINKEDTREENODE*));

               roditelji=(int*)realloc
                (roditelji,n*sizeof(int));

               nivoi=(int*)realloc(nivoi,n*sizeof(int));
           }

           //...
```

Prikaz čvorova stabla

```
// ... void prikaziStablo(TLINKEDTREENODE *stablo){
            if(ltn->levi != NULL){
                cvorovi[end]=ltn->levi;
                roditelji[end]=start;
                nivoi[end]=nivoi[start]+1;
                end++;
            }
            if(ltn->desni != NULL){
                cvorovi[end]=ltn->desni;
                roditelji[end]=start;
                nivoi[end]=nivoi[start]+1;
                end++;
            }
        }
        start++;
    }
    free(cvorovi);
}
```

Oslobađanje stabla

```
void oslobodiStablo(TLINKEDTREENODE **stablo){

   if(*stablo==NULL)
       return;

   //BREADTH FIRST
     TLINKEDTREENODE **ar=
      (TLINKEDTREENODE**)malloc(
      TREE_SIZE_INCREMENT*sizeof(TLINKEDTREENODE*));
     int n=TREE_SIZE_INCREMENT;
     int start=0,end=0;
     ar[start]=*stablo;
     end++;

     TLINKEDTREENODE *ltn=NULL;
     //...
```

Oslobađanje stabla

```
//... void oslobodiStablo(TLINKEDTREENODE **stablo){
    while(start!=end){
        ltn=ar[start];
        if( ltn->levi != NULL || ltn->desni != NULL){
            if(end==n){
                n+=TREE_SIZE_INCREMENT;
                ar=(TLINKEDTREENODE**)realloc(
                    ar,n*sizeof(TLINKEDTREENODE*));
            }
            if( ltn->levi != NULL) ar[end++]=ltn->levi;
            if( ltn->desni != NULL) ar[end++]=ltn->desni;
        }
        free(ltn);
        start++;
    }
    *stablo=NULL;
    free(ar);
}
```

Glavni program

```
int main(){

    int nniz = 0;
    printf("Unesite duzinu niza: ");
    scanf("%d", &nniz);

    TCONTENT *niz = (TCONTENT*)malloc(nniz*sizeof(TCONTENT));

    if(!niz){
        printf("Nemogucnost formiranja niza putem malloc()");
        exit(EXIT_FAILURE);
    }

    //...
```

Glavni program

```
//... int main(){
    int i;
    for(i=0;i<nniz;i++){
        printf("%d:",i+1);
        printf("\n\t unesite broj indeksa: ");
        fflush(stdin);
        gets(niz[i].brindeks);
        printf("\t unesite ime: ");
        fflush(stdin);
        gets(niz[i].ime);
        printf("\t unesite prezime: ");
        fflush(stdin);
        gets(niz[i].prezime);
        printf("\t unesite godinu upisa studija: ");
        scanf("%d", &niz[i].godupis);
    }
    //...
```

Glavni program

```
//... int main(){

    sortirajNiz(niz,nniz);
    printf("\nNiz sortiran!\n");
    stampajNiz(niz,nniz);

    TLINKEDTREENODE *stablo;
    formirajStablo(niz, nniz, &stablo);

    printf("\n\nPrikaz spregnutog stabla u memoriji");
    prikaziStablo(stablo);
    printf("\n");

    free(niz);
    oslobodiStablo(&stablo);

    return 0;
}
```

Zadatak 5

- Proširiti program iz Zadatka 4 dodavanjem mogućnosti
  - serijalizacije binarnog stabla u binarnu datoteku
  - deserijalizacije binarnog stabla iz binarne datoteke
  - snimanja grafičkog prikaza binarnog stabla u tekstualnu datoteku
    - u čvorovima prikazati broj indeksa

```
17000
|--------------------|
16500                17500
|----------|         |----------|
16300      16700     17200      17700
```