

Tema 9

Jednostruko spregnuta lista -
dodatne operacije i rad sa
datotekama, rekurzivne funkcije,
Make

Sortirani unos podataka u listu

- Odabiranje jednog obeležja unutar informacionog dela čvora liste na osnovu koga će se sortirati podaci
- Sortiranje se vrši tokom unosa podataka
 - Vrši se umetanje čvora na određeno mesto u listi

Primer 1

Primer funkcije za sortirani unos u jednostruko spregnutu listu čiji je informacioni deo celobrojna vrednost.

```
void dodaj_sortirano(CVOR **pglava, CVOR *novi)
{
    if (*pglava == NULL || novi->el < (*pglava)->el)
    {
        novi->sledeci = *pglava;
        *pglava = novi;
    }
    else
    {
        CVOR *tekuci = *pglava;

        while (tekuci->sledeci != NULL &&
               novi->el > tekuci->sledeci->el)
        {
            tekuci = tekuci->sledeci;
        }

        novi->sledeci = tekuci->sledeci;
```

Programski jezici i strukture podataka - Tema 9

```
    tekuci->sledeci = novi;  
  }  
}
```

Brisanje čvora

- Prolazi se kroz listu i traži se čvor koji sadrži prosleđenu vrednost
- Ukoliko je pronađen element, proverava se da li je glava liste
 - Ako jeste, nova glava postaće trenutno drugi element liste
 - Ako nije, čvor će se ukloniti tako što se kao sledeći element prethodnog čvora postavi sledeći element tekućeg čvora
- Raskida se veza sa ostatkom liste tako što se sledeći brisanog čvora postavlja na NULL, dok dinamički alocirana memorija uklonjenog čvora se oslobađa pomoću funkcije free

```
void obrisi_cvor(CVOR **pglava, int el)
{
    CVOR *tekuci = *pglava, *prethodni = *pglava;
    while (tekuci != NULL && tekuci->el != el)
    {
        prethodni = tekuci;
        tekuci = tekuci->sledeci;
    }
}
```

Brisanje čvora

```
if (tekuci != NULL)
{
    if (tekuci == *pglava)
    {
        *pglava = (*pglava)->sledeci;
    }
    else
    {
        prethodni->sledeci = tekuci->sledeci;
    }

    tekuci->sledeci = NULL;
    free(tekuci);
}
```

- Napomena: Traženje i brisanje čvora imaju pronalaženje čvora implementirano na dva različita načina (prva while petlja u funkcijama)
- Za domaći, zameniti načine traženja čvora između ove dve funkcije

Zadatak 1

Napisati program koji učitava osvojene poene u igrici. Ispisati učitane podatke u vidu "highscore" tabele, odnosno od najvećeg ka najmanjem. Poeni se učitavaju u jednostruko spregnutu listu. Napisati test program koji učitava poene u proizvoljnom redosledu i prikazuje ih u sledećem formatu "%2d. %d":

```
1. 824
2. 798
3. 777
4. 767
5. 743
6. 692
7. 686
8. 624
9. 533
10. 521
```

Proširenje informacionog dela čvora liste

- Informacioni deo čvora se može sastojati od više polja
- U slučaju sortiranja podataka, bira se jedno od polja na osnovu kog će podaci biti sortirani

Primer čvora koji sadrži informacije o studentu:

```
typedef struct student_st
{
    char broj_indeksa[MAX_INDEX];
    char ime[MAX_IME];
    char prezime[MAX_PREZIME];
    unsigned ocena1;
    unsigned ocena2;
    unsigned ocena3;
    unsigned ocena4;
    double prosek;
    struct student_st *sledeci;
} STUDENT;
```

Pravljenje novog čvora

```
STUDENT *napravi_cvor(char *broj_indeksa, char *ime, char *prezime,
                      unsigned ocena1, unsigned ocena2, unsigned ocena3, unsigned ocena4)
{
    STUDENT *novi = (STUDENT *)malloc(sizeof(STUDENT));

    if (novi == NULL)
    {
        printf("Nije moguće zauzeti memoriju!\n");
        exit(EXIT_FAILURE);
    }

    strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
    novi->ocena1 = ocena1;
    novi->ocena2 = ocena2;
    novi->ocena3 = ocena3;
    novi->ocena4 = ocena4;
    novi->prosek = 0;
    novi->sledeci = NULL;

    return novi;
}
```

Učitavanje iz ulazne datoteke

```
void ucitaj_studente(FILE *ulazna, STUDENT **pglava)
{
    char tmp_broj_indeksa[MAX_INDEX];
    char tmp_ime[MAX_IME];
    char tmp_prezime[MAX_PREZIME];
    unsigned tmp_ocena1, tmp_ocena2, tmp_ocena3, tmp_ocena4;
    STUDENT *novi;

    while (fscanf(ulazna, "%s %s %s %u %u %u %u",
                  tmp_broj_indeksa,
                  tmp_ime,
                  tmp_prezime,
                  &tmp_ocena1,
                  &tmp_ocena2,
                  &tmp_ocena3,
                  &tmp_ocena4) != EOF)
    {
```

Programski jezici i strukture podataka - Tema 9

```
novi = napravi_cvor(  
    tmp_broj_indeksa,  
    tmp_ime, tmp_prezime,  
    tmp_ocena1,  
    tmp_ocena2,  
    tmp_ocena3,  
    tmp_ocena4);  
dodaj_na_kraj(pglava, novi);  
}  
}
```

Ispis u izlaznu datoteku

```
void ispisi_studente(FILE *izlazna, STUDENT *glava)
{
    STUDENT *tekuci = glava;

    while (tekuci != NULL)
    {
        fprintf(izlazna, "%s %s %s %.2lf\n",
                tekuci->broj_indeksa,
                tekuci->ime,
                tekuci->prezime,
                tekuci->prosek);

        tekuci = tekuci->sledeci;
    }
}
```

Zadatak 1

Struktura koja predstavlja studenta sastoji se iz sledećih polja:

- broj indeksa, string od maksimalno 11 karaktera
- ime, string od maksimalno 20 karaktera
- prezime, string od maksimalno 30 karaktera
- ocena 1, neoznačeni, ceo broj
- ocena 2, neoznačeni, ceo broj
- ocena 3, neoznačeni, ceo broj
- ocena 4, neoznačeni, ceo broj
- prosek, realna vrednost

Učitati studente iz ulazne datoteke u jednostruko spregnutu listu. Izračunati im proseke i ispisati sadržaj liste u izlaznu datoteku. Pronaći studenta na osnovu broja indeksa koji se zadaje kao argument komandne linije, ispisati njegovo ime, prezime i prosek.

Primer ulazne datoteke:

ra301-2020	Milan	Milovic	10	9	9	9
ee333-2020	Violeta	Simic	8	7	9	8
pr311-2020	Bojan	Antonic	10	8	8	9
ra342-2020	Goran	Kancarevic	7	6	6	8
ee312-2020	Aleksa	Boric	7	8	8	10
ra363-2020	Milica	Janjic	9	10	10	10
pr321-2020	Stefan	Lakovic	6	7	7	7
ee309-2020	Lazar	Marcetic	8	7	8	9
pr343-2020	Pavle	Petrovic	9	9	9	10

Primer poziva programa za studenta koji postoji u listi:

```
./studenti studenti.txt ispis-proseka.txt pr343-2020
```

Očekivani ispis na standardnom izlazu:

```
Student Pavle Petrovic ima prosek 9.25.
```

Primer poziva programa za studenta koji ne postoji u listi:

```
./studenti studenti.txt ispis-proseka.txt pr342-2020
```

Očekivani ispis na standardnom izlazu:

```
Ne postoji student sa zadatim brojem indeksa.
```

Izlazna datoteka ispis-proseka.txt u oba slučaja je:

```
ra301-2020 Milan Milovic 9.25
ee333-2020 Violeta Simic 8.00
pr311-2020 Bojan Antonic 8.75
ra342-2020 Goran Kancarevic 6.75
ee312-2020 Aleksa Boric 8.25
ra363-2020 Milica Janjic 9.75
pr321-2020 Stefan Lakovic 6.75
ee309-2020 Lazar Marcetic 8.00
pr343-2020 Pavle Petrovic 9.25
```

Rekurzivne funkcije

- Funkcija poziva samu sebe
- Svaka rekurzivna funkcija mora imati uslov za izlazak iz rekurzije!
- Pozitivno
 - Razumljivije*
 - Ponekad jedino i moguće (na primer, Akermanova funkcija)
- Mana
 - Opterećuje stek funkcija
 - Brzina

Primer 1

Funkcija za izračunavanje faktoriijela, iterativno i rekurzivno.

```
int fakt_iterativno(int n) {  
    int i, f = 1;  
  
    for(i = 2; i <= n; i++) {  
        f *= i;  
    }  
  
    return f;  
}
```

```
int fakt_rekurzivno(int n) {  
    if(n < 2) {  
        return n;  
    }  
  
    return n * fakt_rekurzivno(n - 1);  
}
```

Napomene

- Rekurzija često izgleda jednostavnije, ali...
 - Koristite je samo ako znate šta tačno radi kod koji pišete
 - Ako su u opticaju iterativno i rekurzivno rešenje, veća je šansa da iterativno ima bolje performanse

Zadatak 1

Napisati program koji izračunava n-ti član Fibonačijevog niza:

$$f_1 = 1, f_2 = 2, f_i = f_{i-1} + f_{i-2}, i = 3, 4, \dots$$

Napisati iterativno i rekurzivno rešenje. Uporediti vreme izvršavanja između implementacija za različite članove n.

Napomena: Program se u Linux terminalu može prekinuti usred izvršavanja pomoću Ctrl+C

Zadatak 2

Sortirati uneti niz od maksimalno 30 celobrojnih elemenata u rastućem redosledu pomoću Merge Sort algoritma. Merge Sort algoritam polovi sadržaj niza dok u svakom podnizu ne ostane 1 (ili 0) elemenata. Polovljenje je rekurzivno i tokom povratka iz rekurzije treba spojiti podnizove tako da rezultujući podniz bude sortiran. Krajnji rezultat je sortirani originalni niz.

Zadatak 3

Napisati program u kome se korišćenjem rekurzivne funkcije izračunava najveći zajednički delilac prirodnih brojeva x i y . Najveći zajednički delilac se rekurzivno definiše sledećom formulom:

$$\text{nzd}(x, y) = \begin{cases} y, & x = 0 \\ \text{nzd}(y \% x, x), & x \neq 0 \end{cases}$$

Kompajliranje programa iz više datoteka sa izvornim kodom

- Projekti u stvarnom svetu sastoje se iz više datoteka
 - Podela na datoteke sa zaglavljem (*header file*, ekstenzija .h) i implementacijom (*source file*, ekstenzija .c)
 - Grupišu se funkcionalnosti i srodni delovi izvornog koda
- gcc podržava kompajliranje ovako struktuiranih projekata
 - Jedan par datoteka zaglavlja (opciono) i implementacije čini jednu objektnu datoteku (*object file*, ekstenzija .o)
 - Linker (program koji je u sklopu gcc kompajlera) je zadužen za spajanje .o datoteka u jednu, izvršnu datoteku, a.out
 - gcc -c primer.c će napraviti objektnu datoteku primer.o, u slučaju uspešnog kompajliranja

Sadržaj zaglavlja i implementacije

Zaglavlje sadrži:

- Deklaracije funkcija (prototipove)
- Makro definicije (`#define` pretprocesorska direktiva)
- Definicije tipova (strukture, unije, enumeracije, tipovi definisani pomoću `typedef`)
- Konstante

Implementacija sadrži:

- Definicije funkcija
- Definicije promenljivih
- Implementaciju programa

Implementacija uvek uključuje barem svoje odgovarajuće zaglavlje pomoću pretprocesorske direktive `#include`

Make

- Alat, koji pomaže pri generisanju izvršnih datoteka pomoću jedne komande, make
 - Moguće je uraditi i ručno, zvanjem, na primer, gcc kompajlera, ali nije učinkovito
- Opis šta je potrebno uraditi u posebnoj datoteci, koja se zove Makefile
- Na predmetu se koristi GNU Make, postoje i druge implementacije

Struktura Makefile-a

- Pravila (*rules*): Objašnjavaju make programu kako da napravi ciljnu datoteku od izvorne, uz navođenje zavisnosti da bi se to postiglo
- Ciljevi (*targets*): Predstavljaju krajnji proizvod nastao od komandi definisanim u pravilima

Primer jedne definicije pravilo-cilj:

```
cilj: zavisnosti...  
    komande  
    ...
```

Primer 1 - funkcija.h

```
#ifndef FUNKCIJA_H
#define FUNKCIJA_H

int fun();

#endif
```

- Provera postojanja makroa FUNKCIJA_H i definicija ako ne postoji
 - Radi se iz razloga ako se zaglavlje uključuje u više različitih datoteka pomoću `#include` direktive
 - Obezbeđuje da se sadržaj zaglavlja pojavi isključivo jednom
- Definicija zaglavlja funkcije fun

Primer 1 - funkcija.c

```
#include "funkcija.h"

int fun()
{
    return 42;
}
```

- Uključuje se zaglavlje funkcija.h pomoću #include direktive
- Konkretna implementacija funkcije fun

Primer 1 - main.c

```
#include <stdio.h>
#include "funkcija.h"

int main()
{
    printf("Koji je smisao zivota, svemira i svega?\n");
    printf("Odgovor: %d\n", fun());
    return 0;
}
```

- Uključuje se `stdio.h` i `funkcija.h` pomoću `#include` direktive
 - Razlika između korišćenja navodnika i znaka manje-veće kod `#include` direktive (traženje datoteke)
- Implementacija `main` funkcije

Primer 1 - Makefile

```
.PHONY: clean

a.out: funkcija.o main.o
    gcc funkcija.o main.o

main.o: main.c
    gcc -c main.c

funkcija.o: funkcija.c
    gcc -c funkcija.c

clean:
    rm *.o a.out
```

- Pravljenje objektnih datoteka pomoću funkcija.o i main.o ciljeva
- Pravljenje izvršne datoteke a.out na osnovu objektnih datoteka
- Pomoću make se pravi a.out, a pomoću make clean se brišu objektne i izvršna datoteka

Zadatak 1

```
.PHONY: clean

a.out: lista.o zad1.o
    gcc lista.o zad1.o

zad1.o: zad1.c
    gcc -c zad1.c

lista.o: lista.c
    gcc -c lista.c

clean:
    rm *.o a.out
```

Na osnovu zadatog Makefile-a, izvršiti izmeštanje izvornog koda prethodno rešenog zadatka 1 (iz sekcije "dodatne operacije" ili "proširenje informacionog dela čvora liste"). Izmestiti sve funkcije koje imaju vezu isključivo sa listom (inicijalizacija, dodavanje, brisanje liste itd.) u par zaglavlje/implementacija lista.h i lista.c. Ponašanje programa nakon reorganizacije treba da ostane očuvano.