Datoteke

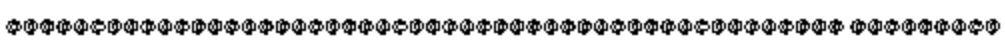
Za početak par napomena vezanih uz datoteke!

Formatirane datoteke:
- obavezno korištenje fscanf i fprintf naredbi za čitanje i pisanje, može i fgets i fputs
- korištenje fread i fwrite naredbi kod formatiranih datoteka vraća nula bodova na tom zadatku na ispitu
- strukture možete koristit, ali ne morate, lakše vam je bez njih. Vama na izbor.

Neformatirane datoteke:
- obavezno korištenje fread i fwrite naredbi za čitanje i pisanje
- korištenje fscanf i fprintf ili fgets i fputs naredbi kod neformatiranih datoteka vraća nula bodova na tom zadatku na ispitu
- obavezno korištenje struktura (struct .. { ... };).

Evo nadam se da će vam svima ovo pomoći da bolje shvatite datoteke. Ako imate bilo kakvih pitanja, pošaljite mi na PM! Također bih vam preporučio da potražite tu na forumu riješene domaće zadaće, obratite pažnju na zadatke za koje se zna da su dobili 2/2, jer samo za takve postoji nekakva garancija da su točni, a dobar su materijal za učenje.

Sretno svima!



Malo više o strukturama...

Možemo ih pisati na nekoliko načina.

```
struct gradovi {
int a;
char b;
} grad;
```

Dakle varijablu grad strukturi gradovi pridužite odmah. Ako ste strukturu napisli ovako i unutar main-a, to se smije napravit.

A ako pišete strukturu izvan main-a, onda ide

```
struct gradovi {
int a;
char b;
};

int main () {
gradovi2 grad2;
...
}
```

Varijablu grad strukturi gradovi dodijelite unutar main-a!

Ovo su dakle 3 načina (a ima ih i još) na koja možete pisati strukture. Sva tri su točna i svejedno je koji od njih koristite. Dovoljno vam je znati samo jedan način, pa si odaberite onaj koji vam se čini najlakši i naučite njegovu sintaksu. Primjetite da uvije na kraju strukture dolazi ;

Ovo struct gradovi grad; i int a; je vrlo slična stvar
a je varijabla tipa int
grad je varijabla tipa struct gradovi

U svim ovim slučajevima, varijablama unutar strukture pristupa se na način grad.a ili grad.b, dakle
varijabla_strukture.varijabla_unutar_strukture

Malo i o randu:

> Citiranje:
>
> Izvorni post od **ljesnjak**:
> *ima li itko tko mi moze malo pojasnit funkcioniranje f-cije random??*
> *recimo da zelim [-3,89] i <5,5] kako to napisat?? -> uz malo objasnjenje*

Ok, objasnit ću ti to na tvojim primjerima.

Funkcija rand() % a; vraća slučajno generirani broj u intervalu [0,a>. Važno je primjetiti da **a** nije uključen i interval, dakle, ako je a jednak 5, funkcija može generirati 0 ili 1 ili 2 ili 3 ili 4.

Za primjer [-3,89], vidimo da su i donja i gornja granica uključene. **Potrebno je prebrojati koliko različitih vrijednosti smijemo generirati!!!** Kako to prebrojati?

Formula je slijedeća:

x = Broj različitih brojeva koje smijemo generirati
gg = gornja granica intervala
dg = donja granica intervala

Slučaj kad su obje granice uključene ([-3,89]):
x = gg - dg + 1; (x je 93)

Znači smijemo generirati 93 različite vrijednosti. Znamo da
rand() % x; vraća vrijednost u intervalu [0,92], pa je potrebno to "naštimati" na interval [-3,89]. To naštimavamo ovako:

rand() % x + dg;

Dakle dodali smo donju granicu intervala, što je u našem slučju -3, te smo tako osigurali da nam se broj nalazi u zadanom intervalu [-3,89].

Slučaj kad je samo jedna granica uključena:
x = gg - dg;

tu postoje dvije mogućnosti:
a) uključena je donja granica [-5,5>
b) uključena je gornja granica <-5,5]

Slučaj a)
rand() % x + dg;
Ovdje je donja granica uključena, pa minimalni rezultat koji smijemo dobit naštimavamo na donju granicu.

Slučaj b)
rand() % x + dg + 1;
Ovdje je donja granica isključena, pa nam minimalni rezultat koji smijemo dobit mora bit za 1 veći od donje granice.

Slučaj kad su obje granice isključene:
Uzmimo npr <0,6>, dakle smijemo generirati samo brojeve od 1 do 5.
Lako se uvjeriti da ukupno ima 5 različitih brojeva koje smijemo generirati i da donja formula vrijedi.

x = gg - dg- 1 ;

rand() % x vraća vrijednosti od 0 do 4.

rand() % x + dg + 1;

Još jedna važna napomena!
Prije korištenja funkcije rand (), potrebno je (najbolje u glavnom programu), napisati slijedeće:

srand((unsigned)time(NULL));

To naučite napamet!!!

Pogledajmo sada što bi bilo kada toga ne bi bilo, tj. kada bismo koristili rand(), a da prethodno nisamo napisali srand((unsigned)time(NULL));
Uzmimo, na primjer, da želimo generirati 5 brojeva u intervalu [0, 10].

Dakle kod za to bi izgledao ovako:

```
for(i=0; i<5; i++) {
polje[i]=rand() % 11;
}
```

Kada bi mi sad ispisali to polje, ispisalo bi se npr:
4 2 7 3 3

Pokrenemo ponovno program. Kakvi će se brojevi generirati ovaj put?
ISTI!!!
4 2 7 3 3

Dakle, svaki put bi program generirao isti niz brojeva, a to onda baš i nije slučajno. Upisivanjem srand((unsigned)time(NULL)); prije korištenja rand() funkcije, postižemo da se prilikom svakog pokretanja programa generira stvarno slučajni niz brojeva.

FILE *fopen (const char *filename, const chart *mode);
funkcija obavlja dvije operacije: definira međuspremnik i povezuje međuspremnik sa datotekom. Argument FILENAME predstavlja eksterno ime datoteke. Ako se podaci nalaze na disku., tada se FILENAME odnosi na dio diska na kojem su zapisani podaci koji se indetificiraju argumentom FILENAME. U slučaju datoteka, koje ne mogu djeliti svoje resurse na više logičkih dijelova, argument FILENAME predstavlja samo datoteku. Argument MODE određuje U/I operaciju. Prema definiciji funkcija vraća pokazivač na strukturu tipa FILE koja se još naziva i pokazivač na datoteku. Pokazivač es na datoteku deklarira kao FILE *fp;. Ako se u postupku otvaranja datoteke javi pogreška , funkcija vraća null pokazivač. Greške koje nastaju otvaranjem datoteke mogu biti prouzročene otvaranjem nepostojeće datoteke za čitanjem, pokušaj otvaranjem datoteke koje su sistemski zaštićene. Greške koje nastaju otvaranjem datoteke ispisuju se preko pokazivača.

MODOVI:
«w» pisanje (ako datoteka ne postoji, stvara se; ako postoji, briše se sadržaj; nije dozvoljeno čitanje)
«a» pisanje (ako datoteka ne postoji, stvara se; ako postoji, podaci se dodaju na kraj; nije dozvoljeno čitanje)
«r» čitanje (ako datoteka ne postoji , NULL; nije dozvoljeno pisanje)
«r +» čitanje i pisanje (ako datoteka ne postoji, NULL)
«w+» čitanje i pisanje (ako datoteka ne postoji, stvara se)
«a +» čitanje i pisanje (ako datoteka ne postoji, stvara se; podaci se dodaju na kraj)
Napomena: Na DOS-u za neformatirane (binarne) datoteke se dodaje b (binary).

int fclose (FILE *fp);
pomoću funkcije zatvaramo datoteku. File fp je pokazivač na datoteku koji vraća funkcija fopen(). Upisuje preostale podatke iz međuspremnika u datoteku na disku , upisuje oznaku kraja datoteke EOF i prekida vezu između međuspremnika i datoteke. Zatvaranjem datoteke međuspremnik se oslobađa.Ako je uspjelo zatvaranje vraća 0 a EOF je greška.

int fgetc (FILE *stream);
funkcija omogućuje učitavanje jednog znaka. Argument predstavlja pokazivač na datoteku koji se vraća pozivom funkcije fopen. Funkcija vraća cijelobrojnu vrijednost pročitanog znaka i pomiče pokazivač na sljedeći znak u međuspremniku.To znači da se sadržaj datoteke čita sekvencijalno.Nakon svih učitanih podataka iz međuspremnika, u međuspremnik se učitava novi blok podataka iz datoteke. O očitvanju novog bloka u međuspremnik brinu se same funkcije. U slučaju greške funkcija vraća EOF ili kraj datoteke.Budući da EOF označava kraj datoteke, funkcija će i u slučaju redovno pročitanog kraja datoteke vratiti EOF. Budući da fgetc( ) učitava jedan znak, dakle jedan bajt, može se iskoristiti za definiranje složenih funkcija.

int fscanf (FILE *stream, const char *format, arg1,.....,arg n);
funkcija za učitvanje podataka iz međuspremnika. Ona omogućuje formatirano učitavanje podataka, gdje je FILE pokazivač na datoteku koji se vraća pozivom funkcije fopen. Argumenti kontrloni niz i lista argumenata imaju isto značenje i kod funkcije scanf. Tj funkcija fscanf () je specijalni slučaj scanf funkcije gdje je datoteka iz koje se podaci učitavaju standardni ulaz. Funkcija se koristi najčešće u paru s fprint funkcijom. Funkcija vraća broj učitanih polja a greška EOF.

char *fgets (char *s, int n, FILE *stream);
s --- područje u memoriji gdje će biti smješteni podaci
n --- maksimalan broj znakova koji se može smjestiti u s
vraća pokazivač na učitani niz, Null ako je greška ili kraj datoteke.
Funkcija sprema učitane podatke na adresu na kojoj pokazuje argument funkcije s i vraća pokazivač na pročitani niz znakova.

int fputc (int c, FILE *stream);
omogućuje upis jednog znaka u međuspremnik, gdje je *stream pokazivač na datoteku kjoeg vraća funkcija fopen. Argument c zadaje se cjelobrojna vrijednost znaka koji se upisuje. Ako se u toku upisa podatka ne pojavi greška , funkcija vraća cjelobrojnu vrijednost upisanog znaka. U protivnom funkcija vraća EOF.

int fprintf (FILE *stream, const char *format, arg1, arg2, ...,arg n);
funkcija omogućuje formatirani upis podataka. Argument funkcije stream je pokazivač na datoteku, koji se vraća pozivom funkcije fopen. U slučaju da smo stream zamjenili sa stdout obje bi funkcije obavljale potpuno iste operacije, formatirani ispis podataka na standardni izlaz . Ostali argumenti funkcije imaju isto značenje kao i funkcija printf ( ). Funkcija kao rezultat vraća broj ispisanih znakova,a ako dođe do greške tada vraća EOF

int fputs (char *s, FILE *stream);
omogućuje upis niza znakova u međuspremnik, gdje s predstavlja adresu niza znakova. Ako je došlo do greške funkcija vraća EOF. U suprotnom funkcija vraća cjelobrojnu vrijednost zadnjeg upisanog znaka iz niza. U slučaju da se upisuje prazan niz funkcija vraća vrijednost nula.

size_t fread (void *ptr, size_t size, size_t n, FILE *stream);

ptr ----------- adresa u memoriji u kojoj će se smjestiti učitani podaci
size ----------- veličina jednog objekta koji će se učitati
n ----------- broj objekata koji će se učitati pozivom funkcije; vraća broj učitanih objekata.
Ako je kraj datoteke ili pogreška, rezultat je <n.(broj učitanih blokova je manji od broja zadanih blokova). Zbog toga je kod poziva funkcije dobro provjeriti broj učitanih grupa i kraj datoteke.
Pozivom funkcije učitava se blok podataka zadanih različitim duljinama. Funkcijom može učitati bilo koji tip podatka , različitih duljina. Duljina grupe podataka koja se učitava zadaje se argumentom size. Najčešća primjena funkcije je kod učitavanja polja, polja struktura, dinamičkih struktura podataka... Najčešće se upotrebljava u paru s funkcijom fwrite ( ).

size_t fwrite (void *ptr, size_t size, size_t n, FILE *stream);
ptr ------------ adresa u memoriji s koje se zapisuju podaci
size ----------- veličina jednog objekta koji će se zapisati
n ----------- broj objekata koji će se zapisati pozivom funkcije
Vraća broj zapisanih objekata . Ako je bila greška, rezultat je <n.
Pokazivač je deklariran kao void čime je omogučen prijenos pokazivača bilo kojem tipa podatka. Osnovni blok podataka je promjenjive duljine i ovisi o tipu podatka. Duljina osnovnom bloka podatka izražava se brojem bajtova. Argument *stream je pokazivač na datoteku. Funkcija vraća cjelobrojnu vrijednost koja predstavlja broj uspješno upisanih grupa. Ako se ta vrijednost razlikuje od broja zadanih grupa došlo je do pogreške. Funkcija je pogodna za upis niza podataka različith duljina: polje, polje struktura, dinamičke strukture podataka...

int seek (FILE *stream, long offset, int whence);

offset ---------- pomak u byte
whence:
SEEK_SET - od početka
SEEK_CUR - od trenutne pozicije
SEEK_END - od kraja datoteke
Vraća 0 ako je uspjelo, <>0 ako je pogreška
Podacima u datoteci se može pristupiti direktno, pozicioniranjem pokazivača u međuspremniku.
U slučaju pravilnom pozicioniranja pokazivača u datoteci vraća 0. a offset-broj bajtova određuje sljedeću poziciju pokazivača u datoteci koja ovisi o argumentu whence (odakle). Argument odakle određuje mjesto od kojeg će se pokazivač pomaknuti za zadani broj bajtova .

long ftell (FILE *stream);
trenutna pozicija u datoteci
vraća poziciju u datoteci ili -1 u slučaju pogreške

@@ @@@@@@@@@@

@@ @@@@@@@@@@

Citiranje:

> Izvorni post od **Drizzt**:
> *Slijedna neformatirana datoteka "mobiteli.dat" sadrži sljedeće podatke o mobitelima:*
> *sifra_mobitela int*
> *proizvodjac char[50]*
> *model char[20]*
> *pri čemu redni broj zapisa u datoteci odgovara šifri mobitela.*
>
> *Direktna neformatirana datoteka "cijene.dat" sadrži podatke o cijenama:*
> *sifra_mobitela int*
> *cijena float*
>
> *Datoteka "cijene.dat" je direktna, dakle, redni broj zapisa odgovara šifri (1,2,3,4,5....).*
> *Potrebno je napisati program u kojem treba učitati proizvođača i model mobitela, te koristeći ove dvije datoteke, u prvoj datoteci pronaći šifru za taj mobitel, a u drugoj datoteci preko šifre naći i ispisati cijenu mobitela!*

Source kod:

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

struct mobitell {
        int sifra_mobitela;
        char proizvodjac[50];
        char model[20];
};

typedef struct {
        int sifra;
        float cijena;
} mobitel2;

void napravi_dat() {
        FILE *f1, *f2;

        f1=fopen("mobiteli.dat","wb");
        f2=fopen("cijene.dat","wb");
        /*Otvaranje datoteka za upis vrijednosti u njih, Datoteke su
          neformatirana pa koristimo wb */
        mobitell mobl;
```

```
mobitel2 mob2;

mobl.sifra_mobitela=3;
strcpy(mobl.proizvodjac,"Nokia");
strcpy(mobl.model,"6080");
/*Tu smo upisali vrijednosti u strukturu mobl. Za pridruzivanje
  znakova stringovima mobl.proizvodjac koristimo strcpy, kao prvi
  argument pisemo string u koji kopiramo znakove, a kao drugi
  argument direktno zadajemo niz (pišemo ga pod " ") Ne mozemo
  pisati mobl.proizvodjac=Nokia, to nece raditi. Isto vrijedi i za model,
  tj. mobl.model */
fwrite(&mobl,sizeof(mobl),1,fl);
/*Kada smo pohranili određene vrijednosti u varijable unutar strukture
  jednostavno cijelu strukturu zapisemo u datoteku */

mobl.sifra_mobitela=1;
strcpy(mobl.proizvodjac,"Nokia");
strcpy(mobl.model,"3100");
fwrite(&mobl,sizeof(mobl),1,fl);

mobl.sifra_mobitela=4;
strcpy(mobl.proizvodjac,"Sony Ericsson");
strcpy(mobl.model,"K750i");
fwrite(&mobl,sizeof(mobl),1,fl);

mobl.sifra_mobitela=2;
strcpy(mobl.proizvodjac,"Sony Ericsson");
strcpy(mobl.model,"Z300");
fwrite(&mobl,sizeof(mobl),1,fl);

fclose(fl);
/*Gotovi smo s upisom u datoteku mobiteli.dat, sad pocinje upis u
  cijene.dat, a mobiteli.dat zatvaramo */

mob2.sifra=1;
mob2.cijena=1700.12;
fwrite(&mob2,sizeof(mob2),1,f2);

mob2.sifra=2;
mob2.cijena=899.90;
fwrite(&mob2,sizeof(mob2),1,f2);

mob2.sifra=3;
mob2.cijena=1299.00;
```

```
        fwrite(&mob2,sizeof(mob2),1,f2);

        mob2.sifra=4;
        mob2.cijena=1540.49;
        fwrite(&mob2,sizeof(mob2),1,f2);

        fclose(f2);
}

int main(){
        char trazeni_proizvodjac[50];
        char trazeni_model[20];
        int trazena_sifra=0;

        mobitell mobl;
        mobitel2 mob2;

        FILE* fl;
        FILE* f2;

        napravi_dat();
        /*Pozivamo funkciju koja ce nam stvoriti datoteke mobiteli.dat i
        cijene.dat. Ako pokrećete program nekoliko puta za redom, ovu je
        funkciju funkciju dovoljno pozvati samo prvi put. Datoteke će ostati
        zapisane na disku, dakle, nije ih potrebno stalno iznova stvarati.*/

        fl=fopen("mobiteli.dat","rb");
        f2=fopen("cijene.dat","rb");
        /*Otvaramo datoteke u formatu za citanje, neformatirane su pa ide rb */
    puts("Upisi proizvodjaca --> ");
        gets(trazeni_proizvodjac);
        /*Ime proizvodjaca moze sadrzavati praznine (Sony Ericsson) pa koristimo gets,
        jer ako koristimo scanf i %s, učitao bi nam samo do prvog razmaka,
        dakle, u varijablu traženi_proizvođač bi učitao samo Sony*/
        puts("Upisi model --> ");
        /*puts je slican kao i printf, ispisuje stringove pod navodnicima na ekran, ili
          moze ispisati stringove iz tocno odredjenog niza, npr. puts(niz); */
        scanf("%s",trazeni_model);
        /*Model ne sadrzi praznine pa mozemo koristiti scanf*/


        while(fread(&mobl,sizeof(mobl),1,fl)==1){
                /*Citamo iz datoteke jednu strukturu, dok je broj procitanih struktura jednak 1,
                  procitani podatci se spremaju u strukturu mobl*/
                if(!(strcmp(mobl.proizvodjac,trazeni_proizvodjac))&&!(strcmp(mobl.model,trazeni_model))){
                        /*Ako je proizvodjac kojeg smo ucitali sa tipkovnice jednak proizcodjacu kojeg
                          smo ucitali iz datoteke i ako je model kojeg smo ucitali sa tipkovnice jednak
                          modelu kojeg smo ucitali iz datoteke, citamo sifru iz datoteke i spremamo ju
                          u varijablu trazena_sifra, strcmp vraća 0 ako su dva niza koja uspoređujemo
                          identična, pa zato tu funkciju negiramo (uskličnik ispred poziva funkcije) */
                        trazena_sifra = mobl.sifra_mobitela;
                }
        }
        if(trazena_sifra==0) {
                printf("Trazeni mobitel ne postoji.");
                exit(0);
        }
        /*Ako nismo nasli sifru u datoteci, za mobitel koji smo upisali sa tipkovnice, onda
          taj mobitel ne postoji u nasoj datoteci i izlazimo iz programa*/

        fseek(f2,(trazena_sifra-1)*sizeof(mob2),SEEK_SET);
        /*Datoteka cijene.dat je direktna pa se pomocu sifre pozicioniramo na pocetak zapisa
          sa trazenom sifrom i cijenom. Budući da šifre počinju od 1 i predstavljaju redni broj zapisa,
          a pokazivač na datoteku nam se na otvaranja datoteke nalazi na početku datoteke, da bismo
          došli do zapisa sa šifrom 1, moramo se pomaknuti za 0 bajtova, do zapisa sa šifrom 2 moramo se
          pomaknuti za jedan blok podataka, sa šifrom 3 za 2 bloka podataka, pa zato imamo
         (trazena_sifra-1)*sizeof(mob2) */
        fread(&mob2,sizeof(mob2),1,f2);
        /*Citamo zapis sa trazenom sifrom, procitani podatci se spremaju u strukturu mob2*/
        if(mob2.sifra==trazena_sifra){
                /*Jos jednom provjeravamo da li je sifra na tom mjestu u datoteci jednaka
                  trazenoj sifri i ako je, ispisujemo cijenu */
                printf("\nCijena je:%.2f\n",mob2.cijena);
        }

        fclose(fl);
        fclose(f2);
        /*Zatvaramo datoteke, gotovi smo*/
        return 0;
}
```

Citiranje:

> Izvorni post od **PsyBurn**:
> *Slijedna formatirana datoteka gradovi.txt sa zapisima oblika:*
> *ime_grada[20+1]#postanski_broj\n*
>
> *Druga, slijedna formatirana datoteka izbaci.txt sa zapisima oblika:*
> *postanski_broj\n*
>
> *Kreirati novu, slijedno formatiranu datoteku s imenom nova.txt, gdje su zapisani svi gradovi iz gradovi.txt , OSIM onih ciji se postanski broj nalazi u datoteci izbaci.txt .*

Source kod:

```
#include <stdio.h>
struct prva {
        char grad[20+1];
        int posta;
};
struct druga {
        int brposte;
};

int main(){
        struct prva jedan;
        struct druga dva;
        int a=1, trazenibroj=0;
        /*a smo na pocetku postavili u 1, koristit cemo a kao pomocnu varijablu
          pomocu koje odredujemo dali nastavljamo upis u datoteku*/
        FILE *gradovi;
        FILE *brojposte;
        FILE *treca;
        gradovi=fopen("gradovi.txt", "w");
        /*Otvaramo datoteku u formatu za pisanje*/
        brojposte=fopen("izbaciti.txt", "w+");
        /*Otvaramo datoteku u formatu za pisanje*/

        while (a){
                printf("Upisi ime grada: ");
                gets(jedan.grad);
                /*Ucitavamo ime grada, ime moze sadrzavati praznine, pa koristimo
                  gets(), jer bi scanf ucitao samo do prvoga razmaka*/
                printf("Upisi postanski broj: ");
                scanf ("%d", &jedan.posta);
                /*Ucitamo postanski broj*/
                fprintf (gradovi,"%s#%d\n", jedan.grad, jedan.posta);
                /*Podaci u datoteku moraju biti zapisani u obliku:
                  ime grada#postanski broj\n, pa ih pomocu fprintf-a tako i upisujemo*/
                printf("1 za nastavak, 0 za kraj -->");
                scanf ("%d", &a);
                /*Ako u varijablu a ovdje unesemo 1, ponovno cemo pisati u ovu istu
                  datoteku, ako unesemo 0, prestajemo s pisanjem u datoteku*/
                getchar();
                /*Nakon sto smo ucitali vrijednost u a, moramo potvrditi unos sa enter.
                  Ako smo unijeli 1, while petlja se nastavlja izvrsavati i da nam taj
                  "enter" program ne bi spremio pod onaj gets na vrhu, koristimo prazan
                  getchar. Dakle getchar() čita jedan znak i ne sprema ga nigdje*/
                printf("\n");/*Da napravimo razmak izmedju 2 unosa*/
        }
        fclose(gradovi);
        /*Unos u datoteku je gotov, zatvaramo datoteku*/
        a=1;
        /*a ponovno postavljamo u 1, te koristimo za kontrolu upisa u datoteku*/
        while (a){
                printf("Upisi postanski broj: ");
                scanf ("%d", &dva.brposte);
                /*Isto kao i u gornjoj while petlji*/
                fprintf (brojposte,"%d\n", dva.brposte);
                /*Podaci u datoteku moraju biti zapisani u obliku: postanski broj\n,
                  pa ih pomocu fprintf-a tako i upisujemo*/
                printf("1 za nastavak, 0 za kraj -->");
                scanf ("%d", &a);
                getchar(); /*Isto kao i gore*/
                printf("\n");
        }
        fclose(brojposte);
        /*Unos u datoteku je gotov, zatvaramo datoteku*/

        gradovi=fopen("gradovi.txt", "r");
        /*Otvaramo prvu datoteku u formatu za citanje*/
        brojposte=fopen("izbaciti.txt", "r");
        /*Otvaramo drugu datoteku u formatu za citanje*/
        treca=fopen("nova.txt", "w");
        /*Otvaramo trecu datoteku u formatu za pisanje*/

        while(fscanf(gradovi,"%[^#]#%d%*c", jedan.grad, &jedan.posta )!=EOF) {
                /*Citamo prvi redak iz datoteke gradovi i podatke spremamo u varijable
                  strukture prva
                  %[^#] --> cita sve znakove dok ne dodje do znaka #, koristimo
```

```
                        ovaj nacin kada treba citati znakove s razmakom, jer
                        bi %s ucitalo samo do prvog razmaka
                   %*c --> znaci da treba preskociti jedan znak, mi preskacemo oznaku za novi
                       red na kraju retka. Mogli smo pisati i \n umjesto %*c, no %*c je
                               univerzalni nacin za preskakanje znakova, preskocit ce bilo koji znak,
                               dok \n mozemo koristiti jedino ako smo sigruni da je bas to znak koji
                       trebamo preskociti, kao što smo koristili direktno znak #, poslije %[^#]*/
                while(fscanf(brojposte,"%d%*c", &dva.brposte)!=EOF) {
                        /*Za svaki procitani redak prve datoteke, citamo cijelu drugu
                          datoteku i trazimo da li se u njoj nalazi isti postanski broj
                          kao i u prvoj datoteci*/
                        if(dva.brposte==jedan.posta) trazenibroj=1;
                        /*Ako se isti postanski broj nalazi u obje datoteke, pomocnu
                          varijablu trazenibroj postavljamo u 1*/
                }
                if (trazenibroj==0) {
                        /*Ako je trazeni broj ostao nula, onda ne postoji u obje datoteke
                          isti postanski broj, pa ucitani redak iz prve datoteke prepisujemo
                          u trecu datoteku*/
                        fprintf(treca,"%s#%d\n", jedan.grad, jedan.posta);
                }
                fseek(brojposte,0L,SEEK_SET);
                /*Nakon sto se izvrti ovaj unutarnji while, pokazivac na drugu datoteku
                  ce se nalaziti na kraju te datoteke, pa kad ucitamo slijedeci redak iz
                  prve datoteke iz druge nece citati vise nista. A buduci da treba provjeriti
                  cijelu drugu datoteku, pokazivac te datoteke treba vratiti na njen pocetak*/
                trazenibroj=0;
                /*U slucaju da je isti postanski broj nadjen u obje datoteke trazeni broj ce biti
                  postavljen u 1. Moramo ga vratiti u 0 kako bi program mogao ispravno provjeriti
                  i sve ostale retke u prvoj datoteci*/
        }

        printf("Ispis prve datoteke: \n");
        fseek(gradovi,0L,SEEK_SET);
        while(fscanf(gradovi,"%[^#]#%d%*c", jedan.grad, &jedan.posta)!=EOF){
                printf("%s#%d\n", jedan.grad, jedan.posta);
        }
        printf("\n");

        printf("Ispis druge datoteke: \n");
        fseek(brojposte,0L,SEEK_SET);
        while(fscanf(brojposte,"%d%*c", &dva.brposte)!=EOF){
                printf("%d\n",dva.brposte);
        }
        printf("\n");

        fclose (gradovi);
        fclose (brojposte);
        fclose (treca);
        /*Gotovi smo, zatvaramo sve 3 datoteke. Rezultate upisa u datoteke mozete pogledati
          u direktoriju u kojem je smješten vaš program. Tamo ce biti kreirane datoteke.
          Npr. u mom slucaju: C:\Program Files\Microsoft Visual Studio\MyProjects\Gradovi*/

        treca=fopen("nova.txt","r");
        printf("Ispis trece datoteke: \n");
        while(fscanf(treca,"%[^#]#%d%*c", jedan.grad, &jedan.posta)!=EOF){
                printf("%s#%d\n", jedan.grad, jedan.posta);
        }
        printf("\n");
        /*Otvaramo trecu datoteku u formatu za citanje, te ju ispisujemo da vidimo dal program radi*/
        fclose(treca);

        return 0;
}
```

@@@ @@@@@@@@@@@@@@@@@@@@@@

Citiranje:

> Napisati funkciju prebroji koja će prebrojati broj znamenki u formatiranoj datoteci. Ime datoteke prenosi se u funkciju kao argument. Potrebno je provjeravati uspješnost operacija nad datotekom.

Source kod:

```
#include<stdio.h>
#include<stdlib.h>
/*stdlib.h nam treba zbog naredbe (funkcije) exit*/
#include<ctype.h>
/*ctype.h nam treba zbog isdigit(znak)*/

int prebroji(char *ime) {
        FILE *f;
        char c;
        int br = 0;
```

```
        f=fopen(ime, "r");
        if(f == NULL) {
                printf("Neuspjelo otvaranje datoteke!");
                exit(7);
        }
        /*fopen vraća pokazivač na početak datoteke ako je uspješno otvorio datoteku
          u suprotnom vraća NULL. Ispisujemo poruku o pogrešci. Mnogi se pitaju čemu
          služi ovaj broj kod exit naredbe. Ako se dogodi da otvaranje datoteke nije
          uspjelo, na ekranu će se pojaviti poruka o grešci, a osim nje biti će još
          jedna poruka otprilike ovakvog sadržaja: "The program exited with code 7."
          Dakle, uopće nije bitno koji broj tu piše. Taj broj samo služi programeru,
          kada u programu postoji više takvih exit naredbi, kako bi znao na kojem je
          točno mjestu program izašao, tj. gdje se dogodila pogreška.*/
        while((c = fgetc(f)) != EOF) {
                /*fgetc(f) čita po jedan znak iz datoteke na koju pokazuje pokazivač f.
                  Ako je znak uspješno pročitan, vraća taj znak, tj. pohranjuje taj znak
                  u varijablu c.*/
                if( isdigit(c) ) br++;
                /*Za pročitani znak provjeravamo da li je taj znak znamenka i ako je
                  povećavamo brojač za 1*/
        }

        fclose(f);
        return br;
        /*Vraćamo u glavni program broj znamenki, tj vrijednost varijable br.
          Program očekuje da tu povratnu vrijednost mora vratiti u neku varijablu
          u glavnom programu, mora je nekud pohraniti. Pa bi nam poziv fje iz glavnog
          programa izgledao ovako:
          br_znam = prebroji(naziv);
          br_znam je varijabla deklarirana u glavnom programu koja služi za pohranjivanje
          povratne vrijednosti iz funkcije. naziv je string (niz znakova) koji sadrži ime
          datoteke*/
}
```

Citiranje:

U svakom retku slijedne formatirane datoteke nalazi se zapis o studentu koji je oblika:
ssssNNpppoppppopppo...

gdje je:
ssss - sifra studenta, 4 znamenke, cijeli broj
NN - broj polozenih ispita, cijeli broj
ppp - sifra predmeta, cijeli broj
o – ocjena, cijeli broj

Parova pppo ima onoliko koliko je student polozio ispita.
napisati funkciju koja za svakog studenta ispisuje sifru, te prosjecnu ocjenu!

Ime datoteke prenosi se u fju kao argument funkcije!
Nije potrebno provjeravati uspjesnost operacija!

Source kod:

```
#include<stdio.h>

void fja(char *ime) {

        FILE *f;
        int sifStud, n, sifPred, ocjena, i;
        double prosOcj;

        f = fopen(ime, "r");

        while(fscanf(f, "%4d%2d", &sifStud, &n) == 2) {
                /*Mi ne znamo koliko nam zapravo svaki redak ima znakova. Znamo da
                  prva dva podatka u svakom retku sadrže 4 + 2 znakova, prvi podatak
                  je sifra studenta, drugi podatak je broj polozenih ispita. Nakon toga
                  slijede parovi pppo. ppp su 3 znamenke koje predstavljaju sifru predmeta
                  a o predstavlja ocjenu. Takvih parova ima onoliko koliko je student
                  polozio ispita, u nasem slucaju n*/
                prosOcj = 0;
                /*Prije nego krenemo racunati ocjene prosOcj postavljamo na nulu.
                  Kada ove naredbe ne bi bilo, onda bi u prosječnu ocjenu računao sumu ocjena
```

```
                onog studenta čiji zapis trenutno čitamo + prosječnu ocjenu prethodnog studenta.*/
              for( i=0; i<n; i++) {
                      fscanf(f, "%3d%ld", &sifPred, &ocjena);
                      /*Treba nam for petlja koja će se izvršiti n puta, jer moramo pročitati
                        n zapisa koji izgledaju ovako: pppo. Šifra predmeta nas ne zanima, bitna
                        nam je samo ocjena, te nakon sto ocjenu ucitamo u varijablu ocjena, dodajemo
                        je u sumu ocjena.*/
                      prosOcj += ocjena;
                      /*Računamo sumu svih ocjena dotičnog studenta*/
              }
              prosOcj /= n;
              /*Računamo prosječnu ocjenu*/

              printf("%d %lf", sifStud, prosOcj);
              /*Ispisujemo sifru studenta i njegovu prosjecnu ocjenu*/

              fscanf(f, "%*c");
              /*Iako to nigdje nije explicitno navedeno, moramo biti svjesni činjenice da
                se na kraju svakog retka nalazi oznaka za novi red, osim možda na kraju
                zadnjeg retka. Na kraju zadnjeg retka ne mora postojati \n.
              %*c --> znaci da treba preskociti jedan znak, mi preskacemo oznaku za novi
              red na kraju retka. Mogli smo pisati i \n umjesto %*c, no %*c je
      univerzalni nacin za preskakanje znakova, preskocit ce bilo koji znak,
              dok \n mozemo koristiti jedino ako smo sigruni da je bas to znak koji
              trebamo preskociti */

      }
      fclose(f); /*Zatvaramo datoteku*/
      return;/*Prazan return, jer je funkcija tipa void.
                       Program bi radio i bez te naredbe.*/
}
```

Citiranje:

> Napisati funkciju koja će u slijednoj formatiranoj datoteci "datoteka.txt" prebrojati koliko ima praznih redaka, te koliko ima redaka koji imaju samo jedan znak. Funkcije mora biti tipa void i rezultate vratiti preko argumenata. Pretpostavite da je max duljina retka 512 znakova. Operacije nad datotekom nije potrebno provjeravati.

Source kod:

```
#include<stdio.h>

void fja(int *br_praznih, int *br_1_znak) {

      FILE *f;
      char redak[513];
      /*512 je broj znakova koliko ih može biti u jednom retku (u tih 512
        je također uračunat i znak za novi red \n) + još jedno mjesto za \0*/
      *br_praznih = 0;
      *br_1_znak = 0;

      f = fopen("datoteka.txt","r");

      while(fgets(redak, 512, f) != NULL) {
              /*Sada malo o fgets i zašto nam je on za ovaj zadatak bolji izbor od fscanf?
                Format naredbe fgets izgleda ovako:
                char *fgets (char *s, int n, FILE *stream);
                s --- područje u memoriji gdje će biti smješteni podaci
                n --- maksimalan broj znakova koji se može smjestiti u s
                stream ---- pokazivač na datoteku iz koje čitamo

                fgets čita najviše n znakova (u našem zadatku to je 512) ili dok ne dođe
                do znaka \n, pri čemu također pročita i znak \n. fgets također dodaje znak
                \0 na kraj stringa u koji sprema podatke.
               fgets koristimo zbog automatskog dodavanja \0 na kraju, a i nije nam bitno
               kako izgledaju podaci u retku, tako da ga je za ovaj primjer jednostavnije koristit*/
              if(strlen(redak) == 1) *br_praznih++;
              /*Zašto je duljina praznog retka jedan a ne nula?
                Zato što se na kraju svakog retka nalazi oznaka za novi red. Pa isto tako
                se ta oznaka nalazi i u praznom retku, fgets ju čita i na kraj stringa
                redak dodaje \0. strlen radi tako da vraća duljinu niza znakova bez znaka
                za kraj niza \0. Sve ostale znakove računa u duljinu niza. Dakle, zato što
                se na kraju svakog retka nalazi \n (ako je redak prazan to je zapravo jedini
                znak), duljina praznog retka biti će 1, a duljina retka s jednim znakom 2 */
              if(strlen(redak) == 2) *br_1_znak++;
      }

      fclose(f); /*Zatvaramo datoteku*/
}
```

Citiranje:

> Za studenta se vodi evidencija: sifra (int), ime i prezime (sadrzi praznine, max 40 znakova), broj bodova (int). 50 bodova smatra se granicom za prolaz. U svakom retku datoteke "studenti.txt" nalaze se podaci za jednog studenta u slijedećem formatu:
>
> sifra#ime prezime#broj bodova
>
> Napisati glavni program u kojem svim studentima koji imaju manje od 50 bodova, treba povećati broj bodova za 10.
>
> Napomena: Ovo je tip zadatka kada se u formatiranoj datoteci koristi fseek. Dakle, kada treba nešto mijenjati u već postojećoj formatiranoj datoteci, tada se i kod formatiranih datoteka koristi fseek.

Source kod:

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

int main () {
        int sifra, br_bod, n;
        char ime_prez[41];
        FILE *f;

        f = fopen("studenti.txt","r+");
        /*Otvaramo datoteku u r+ foramtu jer iz nje čitamo, a po
          potrebi i pišemo u nju.*/

        if (f == NULL) {
                printf("Neuspjelo otvaranje datoteke!");
                exit(0);
        }

        n=ftell(f);
        /*Funkcija ftell vraća broj bajtova od početka datoteke do naše trenutne
          pozicije u datoteci. Ovaj prvi ftell radimo prije čitanja iz datoteke
          zbog mogućnosti da prvi student (student u prvom retku) ima manje od 50
          bodova.*/

        while(fscanf(f, "%d#%[^#]#%d%*c", &sifra, ime_prez, &br_bod) != EOF) {
                /*Ovdje nam se javlja problem, budući da ime i prezime mogu sadržavati
                  praznine, a fscanf, kao i scanf, kod korištenja %s čita znakove samo
                  do prvog razmaka. To se rješava tako da kao format za čitanje u fscanf
                  naredbi koristimo slijedeće: %[^#]. To znači: čitaj sve znakove (bilo
                  kakve) sve dok ne dođeš do znaka #. Na taj način je moguće i sa fscanf-om
                  čitati praznine i pohranjivati ih u string. U našem slučaju, pohranjujemo
                  sve te znakove ih u ime_prezime. %*c znači preskoči jedan znak, u našem
                  slučaju oznaku za novi red. Naredba fscanf će pročitati taj znak, ali
                  program ne očekuje da taj pročitani znak negdje treba i pohraniti.*/
                if(br_bod < 50) {
                        /*Ako je broj bodova manji od 50, bodove treba povećati za 10
                          zatim ih treba upisati na odgovarajuće mjesto u redak studenta
                          kojem pripadaju. Nakon što se gore izvršila naredba fscanf, podatke
                          iz tog retka smo učitali u varijable, no pokazivač na datoteku se
                          sada nalazi na početku slijedećeg retka.*/
                        br_bod += 10;
                        fseek(f, n, SEEK_SET);
                        /*Budući da nam u formatiranoj datoteci retci mogu biti različite duljine
                          nije dobro izračunati duljinu retka, pa pretpostaviti da su svi retci
                          jednake duljine. U varijabli n se nalazi broj bajtova od početka datoteke
                          do početka našeg retka iz kojeg smo čitali podatke i u koji, ako smo
                          povećali bodove trebamo pisati. Pa krenimo s pisanjem.*/
                        fprintf(f, "%d#%s#%d\n", sifra, ime_prez, br_bod);
                        /*Podatke ponovno upisujemo u taj redak u formatu u kojem je u zadatku
                          određeno da trebaju biti upisani (sifra#ime prezime#broj bodova).
                          Pažnja: Voditi računa o oznaci za novi red. printf i fprintf nemaju
                          problema sa prazninama kod stringova.*/
                }

                n = ftell(f);
                /*Nakon izvršavanja fprintf naredbe nam se pokazivač na datoteku ponovno
                  nalazi na početku slijedećeg retka. U slijedećem prolasku kroz while petlju
                  to će nam biti početak našeg aktivnog retka na koji ćemo se morati vraćati
                  ako budemo u taj redak morali pisati. Samo za prvi redak u datoteci se
                  n = ftell(f) piše prije ulaska u while petlju, a za svaki slijedeći redak prije
                  ponovnog izvršvanja naredbe fscanf iz uvjeta while petlje. Tako će u varijabli
                  n nakon izvršavanja fscanf-a biti broj bajtova od početka datoteke do početka
                  retka kojeg smo pročitali naredbom fscanf.Nadam se da je ovo razumljivo */

        }
        fclose(f);
        return 0;
}
```

Ovaj program možete vrlo jednostavno testirati i na svojim računalima. Potrebno je samo stvoriti datoteku kojoj će svaki redak izgledati ovako:
sifra#ime prezime#broj bodova

Npr.

123#Ivo Ivic#45
234#Marko Maric#78
23456#Bosiljka#22
56#Edwin Van Der Saar#62 (Primjetite da ime i prezime mogu sadržavati i više od jedne praznine ili nijednu prazninu, no mi smo ubili sve muhe jednim udarcem pomoću [^#])

itd.

Poželjno je da se takva datoteka nalazi u radnom direktoriju vašeg programa. Možete ju dodati čak direktno iz Visual Studia, tako da odete na File -> New i kao novi file dodate txt file, pri čemu vodite računa o imenu datoteke!
Ako se takva datoteka ne nalazi u radnom direktoriju vašeg programa potrebno je u fopen naredbi napisati cijeli path do vaše datoteke. Npr:
f = fopen("D:\moj direktorij\studenti.txt","r+");

A također ju možete stvoriti i iz C-a, korištenje fprintf-a. A za neformatirane datoteke imate primjer u prvom postu kako stvoriti neformatiranu datoteku. Pa sretno!

NEFORMATIRANE DATOTEKE

Isprika, znam da sam obećao za nedjelju, ali nisam stigao do danas, stavit ću još par primjera večeras. Uživajte!

Citiranje:

> U slijednoj neformatiranoj datoteci "cijene.bin" nalaze se podaci o cijenama određenih proizvoda. **Prvi podatak** u datoteci je broj podataka (long) koji se nalaze u datoteci, a zatim slijedi toliki broj parova šifra (int) i cijena (float). Napisati program koji će izračunati prosječnu cijenu proizvoda.

Source kod:

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

struct cijene {
        int sifra;
        float cijena;
} ;

int main () {
        FILE *f;
        long broj;
        /*broj nismo stavili u strukturu, jer nam se taj podatak nalazi samo na početku datoteke,
          dok nam se ostali podaci ponavljaju, a svaki zapis je oblika sifra - cijena, najlakše je
          sve podatke koji pripadaju jednom zapisu staviti u strukturu, u suprotnom moramo voditi
          računa o tome koji podatak nam dolazi slijedeći i umjesto jednog čitanja iz datoteke u
          kojem pročitamo cijeli jedan zapis, morali bismo čitati onoliko puta koliko se različitih
          podataka nalazi u svakom zapisu.*/
        float prosjek = 0;
        struct cijene cij;
        int i;

        f = fopen("cijene.dat","rb");
        /*Otvaramo u formatu za čitanje jer u datoteku ne moramo ništa pisati.
          Kod svih neformatiranih datoteka, kada pišemo format u kojem ih otvaramo,
          uz slovo koje označava format mora se nalaziti slovo b, da bi prevodioc
          znao da se radi o binarnoj (neformatiranoj) datoteci.*/

        if (f == NULL) {
                printf("Greska kod otvaranja datoteke.");
                exit(1);
        }

        fread(&broj, sizeof(long), 1, f);

        /*Budući da nam broj određuje koliko zapisa u datoteci ima, možemo koristiti i for petlju,
          jer nam je poznat broj ponavljanja, pa nam while petlja nije nužna.*/

        for(i=0; i<broj; i++) {
                fread(&cij, sizeof(cij), 1, f);
                prosjek += cij.cijena;
        }

        prosjek /= broj;
        /*U varijabli prosjek se prije ove naredbe nalazi suma svih cijena, a u varijabli broj nam se nalazi
          ukupan broj zapisa u datoteci, a samim time i ukupan broj cijene koje smo zbrojili u varijablu prosjek.*/

        fclose(f);
        return 0;
}
```

Citiranje:

> Napisati funkciju koja će kao rezultat vratiti broj **popunjenih** zapisa direktne neformatirane datoteke u kojoj su zapisi oblika :
> matični broj - int
> prezime i ime - 40 + 1 znak
> Redni broj zapisa odgovara matičnom broju.
> Napisati glavni program u kojem je potrebno otvoriti datoteku, te pokazivač na istu predati funkciji.Napisati i poziv funkcije iz glavnog programa.
> Operacije nad datotekom nije potrebno provjeravati.

Source kod:

```
#include <stdio.h>

int vratiBrZapisa(FILE *f) {
	/*Funkciju možemo nazavati bilo kako, to je ostavljeno nama na izbor. Kao argument
	  funkcija prima pokazivač na datoteku.*/

	struct {
		int mbr;
		char prezime_ime[41];
	} zapis;

	/*U ovom sam primjeru napisao strukturu unutar funkcije. Više o načinima pisanja
	  struktura imate u prvom postu.*/

	int i = 0, n = 0;
	/*Trebaju nam dva brojača, vidjet ćemo zašto. */

	while(fread(&zapis, sizeof(zapis), 1, f) == 1) {
		/*Čitamo zapis po zapis, while petlja će se izvršavati sve dok je broj pročitanih
		  zapisa jednak 1.*/
		i++;
		/*Budući je datoteka direktna, redni broj zapisa je ujedno i matični broj. To znači
		  da nam redni brojevi zapisa (ilitiga matični brojevi) počinju od 1, a ne od nule.
		  Na to treba paziti. Dakle, nakon svakog pročitanog zapisa povećamo iterator, na taj
		  način bi nam vrijednost iteratora i za svaki pročitani zapis trebala odgovarati
		  njegovom rednom (matičnom) broju. Zašto ovo radimo? Važno je uočiti da u tekstu
		  zadatka piše da moramo prebrojati sve popunjene zapise, što znači da datoteka može
		  sadržavati i prazne zapise. Uobičajena je praksa da se prazni zapisi označavaju
		  nulom. Zato nam redni brojevi zapisa počinju tek od 1, a ne kao indeksi kod polja
		  od nule. Pretpostavimo sada da nam je npr. treći zapis prazan. To znači da je njegov
		  redni broj nula, a ne 3. Pitanje je koji će biti redni broj četvrtog zapisa?
		  Odgovor je 4. Čisto iz razloga (kao što ćete vidjeti na ASPu), što se može dogoditi
		  da na to mjesto ipak moramo nešto upisati.

		  Ilustracije radi, možemo si zamisliti da ova datoteka sadrži JMBAG-ove, te prezime i
		  ime svih studenata (sretnika) koji su prošli PIPI. Također možemo zamisliti da student
		  sa šifrom 3 (Nadajte se da to niste vi ) nije položio PIPI (za sada ). No kako na
		  FER-u postoji i tjedan ponovljenih na kojem taj student može položiti PIPI, postoji
		  mogućnost da  i on postane sretnik i nađe se u ovoj datoteci, pa za njega moramo
		  ostaviti mjesta, jer ako bismo ga dodali na kraj, onda njegova šifra ne bi odgovarala
		  rednom broju zapisa i datoteka u tom slučaju ne bi bila direktna.*/

		if(zapis.mbr == i) n++;
		/*Dakle ako vrijednost iteratora i odgovara rednom broju zapisa, brojač popunjenih zapisa
		  uvećaj za 1. Ovaj uvjet u if naredbi bismo također mogli shvatit na način "ako zapis
		  nije prazan".*/
	}

	return n;
	/*Ovdje je nadam se sve jasno.*/
}

int main () {
	FILE *fp;
	int br;

	fp = fopen("datoteka.dat","rb");
	/*Datoteku otvaramo u formatu za čitanje, datoteka je binarna.*/

	br = vratiBrZapisa(fp);
	/*Primjer poziva funkcije, funkcija vraća broj popunjenih zapisa, te kako taj podatak ne bi
	  bio izgubljen, tj. kako bi nam bio dostupan i u glavnom programu moramo ga pohraniti u
	  neku varijablu.*/

	fclose(fp);
	/*Zatvaramo datoteku.*/

	return 0;
}
```

Citiranje:

> U slijednoj neformatiranoj datoteci su zapisi oblika:
> mat. broj - char[8+1]
> iznos plaće - double
>
> Redni broj zapisa odgovara matičnom broju.
> Napisati funkciju koja će svim osobama kojima je plaća ispod prosjeka povećati plaću za 10 %. Pretpostaviti da je datoteka otvorena u glavnom programu.

Source kod:

```
#include <stdio.h>

struct pla {
        char mbr[8+1];
        double placa;
};
/*Voditi računa o tome da se ime strukture mora razlikovati od imena varijalbi*/

void povecaj(FILE *f) {
        /*Funkcija ništa ne mora vratiti, već samo mijenja postojeću datoteku, pa je tipa void*/

        struct pla pl;
        /*pla je ime strukture, a pl varijabla koja je tipa struct pla*/
        int n = 0;
        double prosjek = 0;

        /*Krenimo prvo izračunati prosjek.*/

        while(fread(&pl, sizeof(struct pla), 1, f) == 1) {
                /*Mogli smo ovdje napisati i sizeof(pl), isto nam dođe, ali nije isto ako napišemo
                  sizeof(struct) ili sizeof(pla).*/
                n++;
                prosjek += pl.placa;
        }

        prosjek /= n;

        /*Važno je primjetiti da nam se svakim izvršavanjem naredbe fread pokazivač pomiče na početak
          slijedećeg zapisa. Nakon što se izvrši while petlja pokazivač f će se nalaziti na kraju
          datoteke, pa ga prvo moramo vratit na početak datoteke, pa tek onda možemo ponovno čitati
          iz datoteke, inače nam se slijedeća while petlja ne bi nikad izvršila. (Broj pročitanih
          zapisa ne bi bio == 1, već nula jer bi nam se pokazivač nalazio na kraju datoteke). Vraćanje
          na početak datoteke možemo izvesti na dva načina: duži i kraći . (Oba vrijede i za
          foramtirane datoteke)*/

        rewind(f);
        /*Ovo je pogađate kraći . Također smo mogli iskoristit i fseek naredbu, jednostavno se
          pomaknemo za nula bajtova od početka datoteke. To bi ovako izgledalo:
          fseek(f, 0, SEEK_SET);*/

        while(fread(&pl, sizeof(pl), 1, f) == 1) {
                /*Ponovno čitamo iz datoteke*/
                if(pl.placa < prosjek) {
                        /*Provjeravamo da li je pročitana plaća manja od prosječne, ako je povećavamo ju
                          i tako povećanu ponovno upisujemo u datoteku.*/
                        pl.placa *= 1.1;
                        fseek(f, -1L * sizeof(pl), SEEK_CUR);
                        /*Čemu sad ovo?
                          Već smo rekli da nam se svakim izvršavanjem naredbe fread pokazivač pomiče na početak
                    slijedećeg zapisa, te kada bismo krenuli upisivati od tog mjesta, izgubili bismo podatke
                          o osobi sa matičnim brojem iz slijedečeg zapisa i njenoj plaći, a u datoteci bismo
                          dobili dvije različite plaće koje vežemo uz isti matični broj. Dakle, moramo se vratit
                          sa početka slijedećeg zapisa, na početak zapisa kojeg smo pročitali i kojem trebamo
                          povećati plaću, da bismo nove podatke upisali na ispravno mjesto.
                          f je pokazivač na datoteku, minus kod sizeof-a označava da se pomičemo u suprotnom smjeru
                          od defaultnog. Defaultni je prema kraju datoteke, ovo bi bilo prema početku. 1L označava
                          da je broj 1 konstanta tipa long. Zašto long? Taj drugi argument fseek naredbe nam označava
                          broj bajtova za koji se želimo pomaknuti. Po sintaksi, fseek očekuje da taj broj bude bude
                          izražen ako tip long. SEEK_CUR je zapravo skraćenica od SEEK CURRENT, u smislenom prijevodu
                          pomakni se od trenutne pozicije.*/
                        fwrite(&pl, sizeof(pl), 1, f);
                        /*Izmijenjene podatke upisujemo u datoteku na odgovarajuće mjesto.*/
                }
        }

        /*Budući da je datoteka otvorena u glavnom programu, ovdje je nećemo zatvarati!*/
}
```

Citiranje:

> Zadana je slijedna formatirana datoteka "mjesta.txt" koja sadrži zapise sljedećeg oblika:
>
> Ada 31214 Laslovo
> Adamovec 10363 Belovar
> Adzamovci 35422 Zapolje
> Alaginci 34000 Pozega
> Alan 53271 Krivi Put
> Bankovci 34000 Pozega
> ...
>
> Prvi dio zapisa je naziv mjesta, zatim slijedi poštanski broj i na kraju je naziv poštanskog ureda, podaci su odvojeni razmakom (NAPOMENA: više mjesta može pripadati jednom poštanskom uredu, u nazivu mjesta ne pojavljuju se brojke, najmanji poštanski broj je 10000 a najveći je 53296, niti jedan naziv nije dulji od 64 slova, naziv poštanskog ureda može sadržavati praznine).
> Potrebno je napisati program kojim će se poštanski brojevi i nazivi poštanskih ureda prepisati u direktnu neformatiranu datoteku "mjesta.dat". Redni broj zapisa u datoteci odgovara poštanskom broju umanjenom za 10000 (zbog uštede prostora jer ne postoje manji poštanski brojevi).

Source kod:

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

typedef struct {
        int pbr;
        char naziv[64];
} sMjesto;

/*Očito je da podatke čitamo iz formatirane datoteke, čemu struktura?
  Struktura iz razloga, što te pročitane podatke trebamo upisati u neforamtiranu datoteku,
  pa je najlakše odmah te podatke učitavati u strukturu.
  Naziv mjesta ne prepisujemo u neformatiranu datoteku, pa nam on nije potreban.*/

int main() {
        FILE * fUlaz, * fIzlaz;
        /*fUlaz - formatirana datoteka, iz nje čitamo
          fIzlaz - neformatirana datoteka, u nju pišemo, to je naš izlazni proizvod */

        sMjesto mjesto;

        /*Pogledati napomene u prvom postu o načinima korištenja struktura. Ja sam u ovim primjerima
          koristio sva tri načina koja sam naveo, čisto da bi vidjeli na primjeru kako to izgleda,
          preporučam da naučite jedan od ta tri i njega se držite, jer inače, ako niste sigurni u
          svoje znanje, dogodit će vam se to da ćete se zbuniti i zabrljati na ispitu.*/
       if ((fUlaz = fopen("mjesta.txt", "r")) == NULL)
               exit(-1);
       /*Formatiranu datoteku otvaramo u formatu za čitanje, a  neformatiranu u foramtu za pisanje.*/
       if ((fIzlaz = fopen("mjesta.dat", "wb")) == NULL)
               exit(-1);

       while (fscanf(fUlaz, "%*[^0-9]%d %[^\n]%*c", &mjesto.pbr, mjesto.naziv) == 2) {
               /*Za početak hajdemo objasniti ovu "%*[^0-9]%d %[^\n]%*c"
                 Rekli smo na početku da nas naziv mjesta ne zanima, jer ga ne prepisujemo.
                 Naziv mjesta može sadržavati praznine, pa ne možemo koristit %s.
                 Znamo da naziv mjesta sasvim sigurno ne sadrži znamenke.
                 Prvi podatak koji nas zanima počinje znamenkom.
                 Znači treba nam nešto što će pročitati sve skupa sa razmakom, dok ne dođe do neke znamenke.

                 Kod fscanf-a čitanje stringa skupa s razmacima možemo napraviti samo ako znamo koji znak
                 (ili skup znakova) nam dolazi poslije tog podatka koji čini taj string.
                 To označavamo na način %[^neki znak]. Oznaka %[^neki znak] znači čitaj sve dok ne dođeš
                 do nekog znaka (ili skupa znakova). U našem primjeru nam je dovoljno staviti %[^1-5],
                 jer su to jedine znamenke kojima može započeti poštanski broj, a %[^1-5] znači:
                 čitaj sve dok ne dođeš do bilo koje znamaenke iz intervala 1 - 5, pri čemu su 1 i 5 uključeni
                 u interval. Zbog općenistosti smo stavili %[^0-9], čime su obuhvaćene sve znamenke.
                 fscanf će dakle pročitati sve znakove dok ne dođe do znamenke, ali ne i znamenku.

                 Što sada znači %*[^0-9]? Čemu * ?
                 Rekli smo već kako nam naziv mjesta nije potreban, jer ga ne moramo prepisati. Između ostaloga
                 cilj programera je i izbjeći korištenje nepotrebnih resursa, u našem slučaju to bi bila rezervacija
                 memorije za naziv grada.
                 %*[^0-9] signalizira prevodiocu (compileru) da treba pročitati sve znakove dok ne dođe do znamenke
                 ali ih ne treba pohraniti, pa fscanf neće očekivati neku varijblu, u našem slučaju string u koji bi
                 pohranio te znakove, jednom čarobnom riječju - PRESKOČI. Zvijezdica je dakle preskok. A što
                 preskačemo, to ovisi o tome s čim u kombinaciji koristimo zvijezdicu.

                 Nakon toga slijedi %d. Dakle, čitamo poštanski broj i spremamo ga u mjesto.pbr.
                 Zatim znamo da slijedi razmak, koji direktno upišemo u fscanf, a nakon razmaka slijedi naziv
                 poštanskog ureda koji također može sadržavati praznine, i na kraju retka se nalazi \n.

                 Znači, opet ne možemo koristiti %s, no znamo da je prvi znak koji slijedi poslije naziva
                 poštanskog ureda \n. Pa čitamo sve do tog znaka %[^\n], i spremamo u mjesto.naziv.

                 Na kraju je još potrebno pročitati taj znak za novi red, kako bismo došli na početak slijedećega
                 retka. Za naziv mjesta nam mogu dolaziti različiti nizovi znakova, no ovdje znamo točno s kojim
                 znakom imamo posla pa ga možemo upisati direktno ( \n ), ili ga možemo jednostavno preskočiti
                 kao što smo i učinili (%*c - preskoči jedan znak)
```

```
                Vidimo da fscanf naredba čita jedan po jedan redak iz formatirane datoteka, a zatim pročitane podatke
                upisujemo u neformatiranu datoteku. Na kraju while uvjeta stoji == 2, zato jer u svakom retku čitamo dva
                podatka i spremamo ih u varijable. Petlja će se izvršavati sve dok je broj podataka učitanih u varijble
                jednak 2. Umjesto == 2, možemo staviti i != EOF. Oba načina su ispravna.*/

            fseek(fIzlaz, (mjesto.pbr - 10000L) * sizeof(mjesto), SEEK_SET);

            /*Neformatirana datoteka mora biti direktna, dakle redni broj zapisa mora odgovarati poštanskom
              broju umanjenom za 10 000. Najmanji poštanski broj je 10 000, pa da bismo njega upisali na početak
              datoteke moramo se od početka pomaknuti za nula bajtova. Da bismo mogli upisati zapis s poštanskim
              brojem 10 001 moramo se od početka pomaknuti za jednu veličinu bloka podataka (jednu veličinu zapisa),
              kako bismo ga upisali na ispravno mjesto, itd. Nadam se da je jasan princip.
              Primjetite da ovdje redni broj zapisa i poštanski broj nemaju istu vrijednost kao u prethodnim
              primjerima.*/

            fwrite(&mjesto, sizeof(mjesto), 1, fIzlaz);
            /*Upisujemo jedan zapis u neformatiranu datoteku*/
        }
        fclose (fUlaz);
        fclose (fIzlaz);

        /*Zatvaramo datoteke.*/

        return 0;
}
```