Programiranje 2
11. predavanje

Saša Singer

singer@math.hr
web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Datoteke (nastavak):
  - Binarni ulaz i izlaz.
  - Direktni pristup podacima.
  - Čitanje i pisanje u istoj datoteci.
  - Primjeri i zadaci.

Informacije

Trenutno nema bitnih informacija.

Binarno čitanje i pisanje

Binarne datoteke — uvod

U praksi često trebamo datoteke koje sadrže

- niz struktura određenog tipa, ili
- niz podataka standardnog tipa (poput `int` ili `double`),

u internoj binarnoj reprezentaciji — bez pretvaranja u tekst.

Na primjer, tako

- izbjegavamo greške zaokruživanja koje nastaju pri formatiranom čitanju i pisanju realnih vrijednosti.

Takve datoteke otvaramo kao binarne. Neformatirane ulazno–izlazne operacije realiziraju se posebnim funkcijama

- `fread` i `fwrite`, koje doslovno kopiraju sadržaj zadanog bloka byteova (kao niz znakova — niz podataka tipa `unsigned char`).

Binarno čitanje i pisanje

Funkcije za binarno (ili neformatirano) čitanje i pisanje su:

```
size_t fread(void *ptr, size_t size,
             size_t nobj, FILE *fp);
size_t fwrite(const void *ptr, size_t size,
              size_t nobj, FILE *fp);
```

Argumenti funkcija su:

- `ptr` — pokazivač na varijablu (ili polje) u koju `fread` upisuje, odnosno, iz koje `fwrite` čita,
- `size` — veličina pojedinog objekta,
- `nobj` — broj objekata koje treba učitati/ispisati,
- `fp` — pokazivač na datoteku iz koje se čita (`fread`), ili u koju se piše (`fwrite`).

Binarno čitanje — funkcija `fread`

Funkcija `fread`

- čita iz datoteke na koju pokazuje `fp`
- niz od `nobj` objekata (svaki veličine `size`)
- i sprema ih u varijablu (polje) na koju pokazuje `ptr`.

Izlazna vrijednost funkcije je:

- broj učitanih objekata iz datoteke,
- koji može biti i manji od `nobj`, ako je došlo do greške ili kraja datoteke.

Treba koristiti funkcije `feof` i `ferror` za provjeru statusa nakon operacije.

Binarno pisanje — funkcija `fwrite`

Funkcija `fwrite`

- piše u datoteku na koju pokazuje `fp`
- niz od `nobj` objekata (svaki veličine `size`)
- iz varijable (polja) na koju pokazuje `ptr`.

Izlazna vrijednost funkcije je:

- broj napisanih objekata u datoteku,
- koji može biti i manji od `nobj`, ako je došlo do greške.
  (Kod pisanja nema smisla testirati kraj datoteke).

Ove funkcije ne rade konverziju iz binarnog zapisa u znakovni (ASCII) zapis i obratno. Čita/piše se blok od `nobj * size` znakova, kao interna reprezentacija podataka u tom računalu.

Binarno čitanje i pisanje — primjer

Primjer. Čitanje cijelog polja cijelih brojeva iz datoteke:

```
int polje[10];
...
fread(polje, sizeof(int), 10, fp);
```

Primjer. Pisanje cijelog polja cijelih brojeva u datoteku:

```
int polje[10] = { ... };
...
fwrite(polje, sizeof(int), 10, fp);
```

Binarno pisanje strukture — primjer

Primjer. Zapis jedne strukture u datoteku.

```
typedef struct {
    int broj_racuna;
    char ime[80];
    double stanje;
} Racun;
Racun kupac = { 47, "Pero Bacilova", -1234.00 };

fp = fopen("novi.dat", "wb");
...
if (fwrite(&kupac, sizeof(Racun), 1, fp) != 1) {
    fprintf(stderr, "Greska pri upisu.\n");
    exit(1); }
```

Binarno čitanje i pisanje — komentari

Prednosti binarnog ulaza/izlaza:

- brzina — nema pretvaranja u tekst ili iz teksta, i
- (mala) veličina zapisa — na primjer, `int` treba 4 bytea, umjesto i do 10 znamenki (bez predznaka).

Nedostatak binarnog ulaza/izlaza:

- ovisnost o arhitekturi računala i prevoditelju,
- nije čitljiv za ljude — binarna datoteka se ne može editirati običnim tekst–editorom (katkad je to i prednost).

U kombinaciji s funkcijama za pozicioniranje u datoteci (`ftell`, `fseek`), funkcije

- `fread` i `fwrite` služe i za direktni pristup podacima.

Direktni pristup podacima

Sekvencijalni pristup podacima

Ulazno–izlazne operacije koje smo radili do sada, koristile su

- tzv. sekvencijalni pristup podacima u datoteci.

Što to znači?

Gdje počinje prva ulazno–izlazna operacija — ovisi o načinu otvaranja datoteke:

- čitanje (`"r"`) i pisanje (`"w"`) ide od početka datoteke, a
- dodavanje (`"a"`) ide na kraj datoteke, iza svega što već postoji u datoteci.

Nakon toga, svaka sljedeća ulazno–izlazna operacija

- nastavlja raditi točno tamo gdje je prethodna operacija završila — tj. stalno idemo "unaprijed" u datoteci.

Trenutna pozicija u datoteci

Za svaku datoteku, u pripadnoj `FILE` strukturi, pamti se i

- trenutna pozicija u datoteci (tzv. `file_pos`),

do koje smo "stigli" s prethodnim operacijama na toj datoteci.

Trenutna pozicija se "mjeri" na isti način kao i indeksi kod polja znakova:

- u broju znakova (byteova) od početka datoteke,
- s tim da nula znači da smo na početku datoteke — ispred prvog znaka = onog s "indeksom" nula (ako ga ima).

Standardni tip za tu vrijednost je `long`, odnosno, `long int`.

Na nekim sustavima, taj tip može biti i veći od `long`,

- ovisno o dozvoljenoj veličini datoteke.

Trenutna pozicija u datoteci (nastavak)

Kako se mijenja trenutna pozicija?

Svaka pojedina ulazno–izlazna operacija uvijek

- ide "unaprijed" u datoteci, od trenutne pozicije.

Zato, kad god napravimo neku operaciju čitanja ili pisanja,

- trenutna pozicija se povećava upravo za broj pročitanih ili napisanih znakova (byteova).

Ako sami ne mijenjamo trenutnu poziciju, onda dobivamo

- sekvencijalno čitanje i pisanje,
- tj. svaka operacija starta tamo gdje je prethodna stala.

Trenutna pozicija u datoteci se "uredno" mijenja "sama" i ne trebamo voditi brigu o njoj.

Direktni pristup podacima — uvod

Međutim, dozvoljeno je

- promijeniti vrijednost trenutne pozicije u datoteci.

Kad to napravimo, onda

- zadajemo mjesto u datoteci na kojem želimo da počne sljedeća ulazno–izlazna operacija.

Na taj način, možemo

- čitati i pisati podatke bilo gdje u datoteci,

tj. svakom znaku (byteu) u datoteci pristupamo direktno, slično kao u polju.

Zato se ovaj način rada s datotekom zove

- direktni ili slučajni pristup podacima.

Direktni pristup podacima — realizacija

Realizacija direktnog pristupa slična je indeksiranju kod polja:

- prije operacije, zadajemo trenutnu poziciju u datoteci.

To se radi posebnom funkcijom za pozicioniranje u datoteci.

Zadavanje pozicije je malo složenije nego kod polja, jer

- promjena mjesta u datoteci može i malo dulje potrajati.

Zato imamo nekoliko mogućnosti za "relativno" zadavanje nove trenutne pozicije u datoteci (obzirom na staru).

Za direktni pristup podacima koristimo dvije funkcije:

- `ftell` — koja daje (vraća) trenutnu poziciju u datoteci, i
- `fseek` — koja mijenja trenutnu poziciju u datoteci na zadanu poziciju.

Trenutna pozicija u datoteci — funkcija `ftell`

Deklaracija (prototip):

```
long int ftell(FILE *fp);
```

Funkcija `ftell` vraća:

- trenutnu poziciju u već otvorenoj datoteci na koju pokazuje `fp`,
- = broj znakova (byteova) od početka te datoteke.

Izlazna vrijednost je:

- nenegativan broj (≥ 0) — u slučaju uspjeha, ili
- `-1L` — u slučaju greške.

Vrijednost `0L` znači da se nalazimo na početku datoteke!

Funkcija `ftell` (nastavak)

Za lakše snalaženje, jer početak = `0L`, dobivena vrijednost je:

- udaljenost od početka datoteke (u znakovima = byteima),
- kao da smo tik ispred prvog sljedećeg znaka (ako ga ima).

Napomena. Odmah nakon otvaranja datoteke s `"r"` ili `"w"`

- dobivamo da je trenutna pozicija = `0L`, tj. početak.

Ako datoteku otvorimo za dodavanje (`"a"`) — dobivena vrijednost ovisi o implementaciji! Na primjer,

- na Windowsima (Intel C, Code::Blocks) — trenutna pozicija je početak datoteke (`0L`),
- na Linuxima — trenutna pozicija je kraj datoteke, iza zadnjeg znaka (`ftell` daje duljinu datoteke u byteima).

Promjena pozicije u datoteci — funkcija `fseek`

Deklaracija (prototip):

```
int fseek(FILE *fp, long offset, int origin);
```

Argumenti funkcije `fseek` su:

- `fp` — pokazivač na već otvorenu datoteku,
- `offset` — zadani pomak u broju znakova (byteova),
- `origin` — indikator položaja ili "ishodište" od kojeg se broji pomak. Zadaje se jednom od sljedeće tri simboličke konstante (definirane u `<stdio.h>`):
  - `SEEK_SET` — od početka datoteke,
  - `SEEK_CUR` — od trenutne pozicije u datoteci,
  - `SEEK_END` — od kraja datoteke.

Funkcija `fseek` (nastavak)

Funkcija `fseek` postavlja trenutnu poziciju

- u datoteci na koju pokazuje `fp`,
- na `offset` znakova od zadanog "ishodišta" `origin`.

Izlazna vrijednost funkcije je:

- nula — ako je uspješno postavila zadanu poziciju, ili
- broj različit od nule — u slučaju greške.

Opet, za lakše snalaženje: `offset` = udaljenost. Zamislite

- da je pozicija tik ispred zadanog znaka (ako ga ima),
- ili da "gledamo" u zadani znak (ako ga ima).

Probajte zamisliti stanje u ishodištima `SEEK_SET` = početak datoteke i `SEEK_END` = kraj datoteke (= iza zadnjeg znaka)!

Funkcija `fseek` — primjeri

Primjer. Nekoliko poziva funkcije `fseek` za pozicioniranje u datoteci zadanoj pokazivačem `fp`.

```
fseek(fp, 0L, SEEK_SET);   /* Na POCETAK
                              datoteke. */
fseek(fp, 0L, SEEK_END);   /* Na KRAJ
                              datoteke. */
fseek(fp, 2L, SEEK_SET);   /* 2 znaka IZA
                              pocetka datoteke. */
fseek(fp, 2L, SEEK_CUR);   /* 2 znaka IZA
                              trenutne pozicije. */
fseek(fp, -2L, SEEK_END); /* 2 znaka ISPRED
                              kraja datoteke. */
```

Zadana pozicija mora biti unutar "granica" datoteke.

Funkcija `fseek` *— za tekstualne datoteke*

Kod poziva funkcije `fseek` za tekstualne datoteke, standard postavlja sljedeće ograničenje:

- `offset` mora biti — nula, ili vrijednost koju vrati poziv funkcije `ftell` (označimo ju s `ftell_pos`).

To znači da su dobro definirani jedino pozivi oblika:

argumenti funkcije `fseek`	značenje
`fp, 0L, SEEK_SET`	idi na početak datoteke,
`fp, 0L, SEEK_END`	idi na kraj datoteke,
`fp, 0L, SEEK_CUR`	ostani na trenutnoj poziciji, (Nema puno smisla!)
`fp, ftell_pos, SEEK_SET`	idi na poziciju koju je dao prethodni poziv `ftell`.

Pozicioniranje na početak — funkcija `rewind`

Pozicioniranje na početak datoteke možemo napraviti i pozivom funkcije

```
void rewind(FILE *fp);
```

Ovaj poziv ekvivalentan je s:

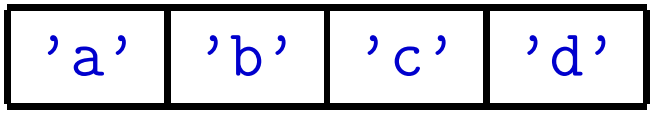
```
fseek(fp, 0L, SEEK_SET);
clearerr(fp);
```

tj., osim pozicioniranja na početak datoteke,

- još brišemo i indikatore — za kraj datoteke i za grešku.

Primjer za funkciju `ftell`

Primjer. Pretpostavimo da imamo već postojeću datoteku koja sadrži točno 4 znaka:



.

Prvo otvorimo tu datoteku za (tekstualno ili binarno) čitanje, a zatim 6 puta ponovimo sljedeće:

- nađemo trenutnu poziciju u toj datoteci (funkcija `ftell`) i ispišemo ju (na `stdout`),
- učitamo sljedeći znak iz te datoteke i ispišemo ga (opet, na `stdout`).

Što je rezultat?

Primjer za `ftell` *— dio programa*

Sljedeći dio programa realizira čitanje, nakon otvaranja:

```
        /* Sekvencijalno citamo tu datoteku. */

    for (i = 0; i < 6; ++i) {
        printf("   Pozicija: %ld,", ftell(fp));
/*
        printf(" znak = %2d\n", fgetc(fp));
*/
        if ((c = fgetc(fp)) >= 0)
            printf(" znak = %2c\n", c);  /* Znak. */
        else
            printf(" znak = %2d\n", c);  /* Broj. */
    }
```

Primjer za `ftell` — rezultati

Izlaz tog dijela programa je:

```
Pozicija: 0, znak =  a
Pozicija: 1, znak =  b
Pozicija: 2, znak =  c
Pozicija: 3, znak =  d
Pozicija: 4, znak = -1
Pozicija: 4, znak = -1
```

"Znak" `-1` je uobičajena vrijednost za `EOF`.

Pripadni program je `fpos.c`. Taj program

- prvo kreira takvu datoteku (s imenom `fpos.dat`),
- a zatim ju čita na zadani način.

Primjer za `ftell` — varijacije na temu

Modificirajte program tako da ispisuje trenutne pozicije i prilikom kreiranja datoteke — prije pisanja svakog znaka.

Varijacija 1. Nakon kreiranja zadane datoteke s 4 znaka, treba:

- otvoriti tu datoteku za dodavanje (`"a"`),
- u nju napisati još 2 znaka: `'e'`, `'f'` (na kraj),
- i zatvoriti datoteku.

Zatim treba otvoriti tu datoteku za čitanje i 8 puta ponoviti operaciju čitanja sljedećeg znaka (kao u primjeru).

Pripadni program je `fpos_a.c`. Uočite da odmah nakon otvaranja za dodavanje, na Windowsima vrijedi:

- trenutna pozicija = `0L`.

Primjer za `ftell` — *varijacije (nastavak)*

Varijacija 2. Nakon kreiranja zadane datoteke s 4 znaka, treba:

- otvoriti tu datoteku za čitanje i dodavanje (`"a+"`),
- 6 puta ponoviti operaciju čitanja sljedećeg znaka (kao u primjeru),
- u datoteku napisati još 2 znaka: `'e'`, `'f'` (na kraj).

Pripadni program je `fpos_ap.c` (`ap` = aplus = `"a+"`).

Uočite da tik prije dodavanja znaka `'e'`, vrijedi:

- trenutna pozicija = `4L`, tj. stigli smo na kraj datoteke.

Napomena. Da čitanje nije stiglo do kraja datoteke,

- pri prijelazu s čitanja na pisanje, trebalo bi pozvati neku funkciju za pozicioniranje u datoteci (v. malo kasnije)!

Naopako kopiranje (invertiranje) datoteke

Primjer. Napišite program koji

- naopako kopira sadržaj jedne datoteke u drugu.

Na primjer, ako prva (ulazna) datoteka ima oblik:

'a'	'b'	'c'	'd'

onda druga (izlazna) datoteka mora imati sljedeći oblik:

'd'	'c'	'b'	'a'

Kopiranje radimo znak po znak, tako da po prvoj datoteci

- idemo unatrag — od kraja datoteke, prema početku,

koristeći direktni pristup podacima.

Naopako kopiranje datoteke (nastavak)

Napomena. Zbog načina pristupa podacima, obje datoteke treba otvoriti kao binarne, a ne kao tekstualne!

Varijabla `pomak` broji pomak "unazad" od kraja datoteke, tj. "ishodište" je `SEEK_END`.

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

int main(void)
{
    char *in_name = "freverse.in";
    char *out_name = "freverse.out";
```

Naopako kopiranje datoteke (nastavak)

```
FILE *in, *out;
long file_pos, pomak = 0L;

if ((in = fopen(in_name, "rb")) == NULL) {
    fprintf(stderr, "Ne mogu citati iz: %s!\n",
            in_name);
    exit(1);
}

if ((out = fopen(out_name, "wb")) == NULL) {
    fprintf(stderr, "Ne mogu pisati u: %s!\n",
            out_name);
    exit(2);
}
```

Naopako kopiranje datoteke (nastavak)

```
    /* Datoteku kopiramo naopako. */

do {
        /* Pomak unazad od kraja. */
    if (fseek(in, --pomak, SEEK_END)) break;

        /* Zapamti poziciju i ucitaj znak.
           Bas tim redom! */
    file_pos = ftell(in);
    fputc(fgetc(in), out);

        /* Sad je pozicija narasla za 1L. */

} while (file_pos != 0L);
```

Naopako kopiranje datoteke (nastavak)

```
    fclose(in);
    fclose(out);
    return 0;
}
```

Program se zove `freverse.c`. Radi jednostavnosti, program koristi fiksna imena datoteka:

- ulazna datoteka — `freverse.in`,
- izlazna datoteka — `freverse.out`.

Napomena. Kod pomaka unatrag od kraja datoteke, test

```
if (fseek(in, --pomak, SEEK_END)) break;
```

je zaštita od prazne ulazne datoteke. Bez tog testa — ne radi.

Naopako kopiranje datoteke — rezultat

Ulazna datoteka `freverse.in` ima točno 33 znaka:

```
Ja sam mala Ruza, mamima sam kci.
```

Izlazna datoteka `freverse.out` ima, također, 33 znaka:

```
.ick mas amimam ,azuR alam mas aJ
```

Složenost ovog programa je linearna u duljini datoteke, zbog direktnog pristupa podacima.

Zadatak. Napravite istu stvar

- sekvencijalnim pristupom podacima.

Složenost je tada kvadratna u duljini datoteke!

Naopako kopiranje datoteke — napomene

Napomena. Obje datoteke moraju biti otvorene kao binarne.

U protivnom,

- čim ulazna datoteka ima bar jedan znak za kraj reda,

stvar ne radi dobro na sustavima kod kojih

- postoji razlika između binarnih i tekstualnih datoteka (poput Windowsa).

Dovoljno je naopako "kopirati" datoteku na tekstualni `stdout`.

Probajte program `frev_out.c`, koji

- naopako ispisuje (binarnu) datoteku `frev_out.in` na `stdout`,

a rezultat je preusmjeren u datoteku `frev_out.out`.

Naopako kopiranje datoteke — kraj napomene

Uzrok greške je:

- pretvaranje kraja linije kod pisanja (i čitanja) znakova!

Na Windowsima, kraj linije na ulazu je "paket" od dva znaka `\r`, `\n`. Kod čitanja unatrag (za binarnu ulaznu datoteku)

- prvo se pročita `\n` (bez pretvaranja),
- ali se napiše kao paket `\r`, `\n` (zbog pretvaranja).
- Zatim se učita i napiše `\r` (bez pretvaranja).

Upravo zato postoje ranije navedena ograničenja

- na pozive funkcije `fseek` za tekstualne datoteke,
- da se izbjegnu ovakve "čarolije" na kraju svakog reda.

Čitanje i pisanje u istoj datoteci

Čitanje i pisanje u istoj datoteci

Datoteku možemo otvoriti tako da je dozvoljeno:

- i čitanje iz te datoteke,
- i pisanje u tu istu datoteku.

Takav "način" rada s datotekom (engl. "update mode") dobivamo tako da, kod otvaranja datoteke,

- u tzv. `file_mod` stringu, koji zadaje "način" rada s datotekom — navedemo znak `+`.

U tom slučaju, treba biti oprezan

- pri prijelazu s čitanja na pisanje i obratno,

zbog toga što postoji spremnik za komunikaciju između programa i datoteke — treba ga korektno "isprazniti".

Čitanje i pisanje u istoj datoteci (nastavak)

U principu, na istoj datoteci, nije dozvoljen izravni prijelaz s jedne vrste operacija na drugu.

Pri prijelazu s čitanja na pisanje, između ovih operacija, treba

- pozvati neku funkciju za pozicioniranje u datoteci (`fseek`, `rewind` ili `fsetpos`),
- osim ako je čitanje stiglo do kraja datoteke.

Razlog: standardno pisanje uvijek ide na kraj datoteke.

Pri prijelazu s pisanja na čitanje, između ovih operacija, treba

- pozvati neku funkciju za pozicioniranje u datoteci, ili
- pozvati funkciju `fflush` za pražnjenje (pisanje) sadržaja spremnika u datoteku.

Pisanje spremnika u datoteku — funkcija `fflush`

Deklaracija (prototip):

```
int fflush(FILE *fp);
```

Ako `fp` pokazuje na "izlaznu" datoteku (tj. zadnja operacija je bila pisanje u tu datoteku), onda `fflush`

- piše u tu datoteku, onaj dio sadržaja spremnika koji do tad nije bio "fizički" napisan u nju,

tj. "prazni" spremnik u datoteku.

Ako `fp` pokazuje na "ulaznu" datoteku (tj. zadnja operacija je bila čitanje iz te datoteke), onda efekt poziva funkcije `fflush`

- nije definiran (nema smisla).

Funkcija `fflush` (nastavak)

Poziv oblika:

```
fflush(NULL);
```

"prazni" spremnike za sve izlazne datoteke (u tom trenutku).

Izlazna vrijednost funkcije `fflush` je:

- nula — ako je uspješno "ispraznila" spremnik(e), ili
- `EOF` — ako je prilikom pisanja došlo do greške.

"Uredni" završetak programa (može i pozivom funkcije `exit`) automatski prazni spremnike za sve otvorene (izlazne) datoteke.

Dodavanje bonusa na račun — početak

Primjer. Telefonski račun opisan je strukturom tipa `Racun`:

```
typedef struct {
    int tel_broj;
    char vlasnik[20];
    double stanje;
} Racun;

int size = sizeof(Racun);
```

Podaci o računima korisnika spremljeni su u binarnoj datoteci koja sadrži niz takvih struktura. Svaki "zapis" u datoteci je

- jedna struktura tipa `Racun` — veličine `size`.

Dodavanje bonusa na račun — zadatak

Treba napisati funkciju `dodaj_bonus` sa zaglavljem oblika:

```
void dodaj_bonus(const char *f_name, int n);
```

String `f_name` je ime (postojeće) binarne datoteke koja sadrži niz struktura tipa `Racun`.

Funkcija treba `n`-tom zapisu u datoteci

- dodati bonus od 100.0 na `stanje` računa, ako je `stanje` prije toga bilo pozitivno.

Brojanje zapisa u datoteci počinje od 1.

Za rješenje koristimo direktni pristup podacima u datoteci.

Uzmimo da se pokazivač na tu datoteku zove `racuni`.

Dodavanje bonusa na račun — pristup podacima

Za čitanje `n`-tog zapisa:

- treba "preskočiti" prvih `n - 1` zapisa od početka datoteke, tj. od "ishodišta" `SEEK_SET`.

Odgovarajuće pozicioniranje je (pretvaranje u `long`):

```
file_pos = (long) (n - 1) * size;
fseek(racuni, file_pos, SEEK_SET);
```

Nakon dodavanja bonusa, za pisanje "novog" `n`-tog zapisa:

- treba se "vratiti" natrag — za jedan zapis od trenutne pozicije u datoteci, tj. od "ishodišta" `SEEK_CUR`.

```
fseek(racuni, -size, SEEK_CUR);
```

Dodavanje bonusa na račun — funkcija

```
void dodaj_bonus(const char *f_name, int n)
{
    FILE *racuni;
    Racun kor;
    long file_pos;
    const double bonus = 100.0;

    if ((racuni = fopen(f_name, "r+b")) == NULL) {
        fprintf(stderr, "Ne mogu otvoriti: %s!\n",
                f_name);
        exit(1);
    }
```

Dodavanje bonusa na račun — funkcija (nast.)

```
    /* Pozicioniranje ispred n-tog zapisa. */
file_pos = (long) (n - 1) * size;

if (fseek(racuni, file_pos, SEEK_SET)) {
    fprintf(stderr,
            "Greska u fseek, n = %d.\n", n);
    printf("Greska u fseek, n = %d.\n", n);
    fclose(racuni);
    return;    /* Necu exit, za demo! */
    }
```

Dodavanje bonusa na račun — funkcija (nast.)

```
    /* Ucitaj zapis u strukturu. */
if (fread(&kor, size, 1, racuni) != 1)
    if (ferror(racuni)) {
        fprintf(stderr, "Greska u citanju.\n");
        exit(2);
    }
    else if (feof(racuni)) {
        fprintf(stderr,
                "Kraj datoteke, n = %d.\n", n);
        printf("Kraj datoteke, n = %d.\n", n);
        fclose(racuni);
        return;    /* Necu exit, za demo! */
    }
```

Dodavanje bonusa na račun — funkcija (nast.)

```
        /* Azuziraj stanje i napisi novi zapis. */
    if (kor.stanje > 0) {
        kor.stanje = kor.stanje + bonus;
        fseek(racuni, -size, SEEK_CUR);
        if (fwrite(&kor, size, 1, racuni) != 1) {
            fprintf(stderr, "Greska u pisanju.\n");
            exit(3);
        }
    }

    fclose(racuni);

    return;
}
```

Dodavanje bonusa na račun — rezultati

Cijeli program za kreiranje i obradu računa zove se `racuni.c`.

- Program kreira datoteku računa s imenom `racuni.dat` (ime se učitava s komandne linije, u `argv[1]`),
- a zatim dodaje bonus nekim zapisima u toj datoteci.

Polazna datoteka `racuni.dat` ima ovaj sadržaj (pogledajte izlaz `racuni.out`):

```
zapis  1: 384907,      Tihana Glasnovic,     92.00
zapis  2: 622744,           Goga Trubic,    456.27
zapis  3: 918235,        Josip Mobitelic,  -234.49
zapis  4: 436702,       Martina Lajavic,     74.12
zapis  5: 739417,         Pero Bacilova, -1017.12
zapis  6: 208143,       Mirna Sutljivic,     48.50
```

Dodavanje bonusa na račun — rezultati (nast.)

Zatim, dodajemo bonus sljedećim zapisima:

```
dodaj_bonus(argv[1], 3);
dodaj_bonus(argv[1], 6);
dodaj_bonus(argv[1], 1);
```

Nova datoteka `racuni.dat` ima ovaj sadržaj (v. `racuni.out`):

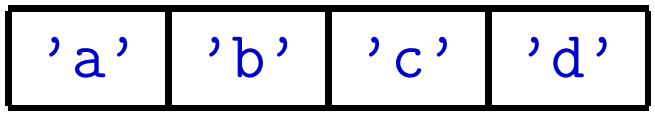
```
zapis  1: 384907,     Tihana Glasnovic,    192.00
zapis  2: 622744,          Goga Trubic,    456.27
zapis  3: 918235,       Josip Mobitelic,  -234.49
zapis  4: 436702,      Martina Lajavic,     74.12
zapis  5: 739417,        Pero Bacilova, -1017.12
zapis  6: 208143,      Mirna Sutljivic,    148.50
```

Naopako okretanje (invertiranje) jedne datoteke

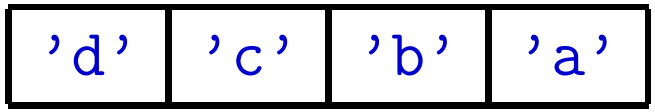
Primjer. Napišite program koji naopako okreće ili invertira

- sadržaj jedne zadane datoteke — "na licu mjesta", tj. ista datoteka je ulazna i izlazna.

Na primjer, ako datoteka na početku ima oblik:



onda na kraju mora imati sljedeći oblik:



Datoteku treba otvoriti tako da je dozvoljeno

- istovremeno čitanje i pisanje u toj datoteci.

Invertiranje datoteke

Dalje postupamo slično kao u polju ili stringu.

- Jedini problem je naći "polovište" datoteke, tj. prepoznati kad smo gotovi!

Bitno najlakši način je

- naći duljinu datoteke = otići na kraj,

i raspoloviti tu duljinu!

```
fseek(dat, 0L, SEEK_END);
file_pola = ftell(dat) / 2L;
```

Sve ostalo je samo

- pažljivo pozicioniranje s odgovarajućim pomakom.

Invertiranje datoteke — program

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

int main(void) {
    char *dat_name = "finvert.dat";
    FILE *dat;
    long file_pola, pomak = 0L;
    int ch_1, ch_2;

    if ((dat = fopen(dat_name, "rb+")) == NULL) {
        fprintf(stderr, "Ne mogu citati iz: %s!\n",
                dat_name);
        exit(1);
    }
```

Invertiranje datoteke — program (nastavak)

```
    /* Duljina i poloviste datoteke. */

fseek(dat, 0L, SEEK_END);
file_pola = ftell(dat) / 2L;

    /* Datoteku invertiramo. */

while (pomak < file_pola) {

        /* Pomak unaprijed od pocetka.
           Ucitaj prvi znak. */

    fseek(dat, pomak, SEEK_SET);
    ch_1 = fgetc(dat);
```

Invertiranje datoteke — program (nastavak)

```
        /* Pomak unazad od kraja.
           Ucitaj drugi znak. */

fseek(dat, -pomak - 1L, SEEK_END);
ch_2 = fgetc(dat);

        /* Pomak za jedno mjesto unazad od
           trenutnog. Napisi prvi znak. */

fseek(dat, -1L, SEEK_CUR);
fputc(ch_1, dat);
```

Invertiranje datoteke — program (nastavak)

```
            /* Pomak unaprijed od pocetka.
               Napisi drugi znak. */

        fseek(dat, pomak, SEEK_SET);
        fputc(ch_2, dat);

        ++pomak;     /* Povecaj pomak! */
    }

    fclose(dat);

    return 0;
}
```

Invertiranje datoteke — rezultat

Program se zove `finvert.c`. Radi jednostavnosti, program koristi fiksno ime datoteke:

- ulazna i izlazna datoteka je `finvert.dat`.

Polazna datoteka `finvert.dat` ima točno 33 znaka (v. `*.in`):

```
Ja sam mala Ruza, mamima sam kci.
```

Izlazna datoteka `finvert.dat` ima, također, 33 znaka:

```
.ick mas amimam ,azuR alam mas aJ
```

Kad dva puta izvršimo program, dobijemo polaznu datoteku.

Složenost programa je linearna u duljini datoteke.

Primjeri i zadaci

Zadatak — okretanje datoteke računa

Zadatak. Napišite program koji okreće ili invertira datoteku `racuni.dat`, koja sadrži niz struktura tipa `Racun`, iz ranijeg primjera.

Nemojte učitati cijeli niz iz datoteke u neko polje, tamo ga okrenuti, a onda napisati niz u datoteku!

Uputa. Prvo treba naći broj računa u datoteci — to je duljina datoteke podijeljena s veličinom svake strukture (`size`).

Dalje postupamo potpuno isto kao u okretanju datoteke pojedinačnih znakova (byteova),

- osim što svaki pojedini objekt ima duljinu `size`,
- pa čitanje treba napraviti funkcijom `fread` (a ne `fgetc`),
- a pisanje treba napraviti funkcijom `fwrite` (a ne `fputc`).

Zadatak — sortiranje datoteke računa

Zadatak. Napišite program koji (uzlazno) sortira datoteku `racuni.dat`, koja sadrži niz struktura tipa `Racun`, iz ranijeg primjera — po nekom zadanom kriteriju (telefonskom broju, imenu vlasnika, ili stanju računa).

Nemojte učitati cijeli niz iz datoteke u neko polje, tamo ga sortirati nekim algoritmom, a onda napisati niz u datoteku!

Uputa. Izaberite neki algoritam sortiranja na polju,

- bilo koji, po želji, samo da se sortiranje radi "na licu mjesta" — u jednom polju,

a zatim sve osnovne operacije realizirajte "malim" funkcijama!

Dakle, ideja je da postupamo potpuno isto kao da smo u polju, a sve što nam "fali" — realiziramo funkcijama.

Zadatak — sortiranje datoteke računa (nast.)

Funkcije koje mogu "zatrebati" (neke, možda, i ne trebaju):

- Broj objekata (ovdje su to računi) u zadanoj datoteci;
- Čitanje i–tog objekta iz datoteke (u strukturu);
- Pisanje i–tog objekta (iz strukture) u datoteku;
- Usporedba dva objekta (strukture) — da se zna koji je "manji" po zadanom kriteriju, po ugledu na `strcmp`;
- Zamjena dva objekta — i–tog i j–tog u datoteci,
  - što se svodi na "pažljivo" čitanje i pisanje,
  - no, može se realizirati i zasebnom funkcijom (slično kao u invertiranju).

Izazov. Napravite QuickSort algoritam na datoteci!