Programiranje 2

Predavanje 11 - datoteke, drugi dio

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

24. svibnja 2023.

Prepoznavanje greške — funkcije `ferror`, `feof`

Funkcije za čitanje podataka iz datoteke vraćaju `EOF` ili `NULL` (kod `fgets`) u dva slučaja:

- ako je došlo do greške prilikom čitanja,
- ako se kod čitanja (odmah) naišlo na kraj datoteke.

Funkcije: `int ferror(FILE *fp);` i `int feof(FILE *fp);` služe za razlikovanje ta dva slučaja.

Funkcija `ferror` vraća broj različit od nule (istina) ako je došlo do greške, a nulu (laž) ako nije. Funkcija `feof` vraća broj različit od nule (istina) ako smo naišli na kraj datoteke prilikom čitanja, a nulu (laž) u suprotnom. Provjeru vrijednosti ovih funkcija treba napraviti odmah **nakon operacije čitanja iz datoteke**. Pisanje nije uspješno samo u slučaju greške, tada `ferror` vraća istinu.

Funkcije `ferror` i `feof`

Svaka otvorena datoteka ima dva indikatora (zastavice, *flag*) statusa u pripadnoj strukturi tipa `FILE`: a) `eof` (end of file) - označava kraj datoteke, b) `error` - označava grešku prilikom operacije (npr. disk se napuni prilikom pisanja).

Funkcije `ferror` i `feof` testiraju stanje zastavica (indikatora) i vraćaju odgovarajuću logičku vrijednost. Otvaranje datoteke `fopen(...)` briše stanje oba indikatora. Odmah nakon uspješnog otvaranja datoteke je: `feof(...) == 0` i `ferror(...) == 0`.

Indikatore postavljaju funkcije za ulazne i izlazne operacije na datoteci (stanje indikatora odnosi se na prethodnu operaciju). Zbog toga provjeru stanja treba napraviti odmah nakon operacije. Indikator `error` signalizira grešku i kod čitanja i kod pisanja. `feof` treba testirati samo nakon čitanja.

Indikatore datoteke na koju pokazuje pokazivač `fp` možemo obrisati funkcijom: `void clearerr(FILE *fp);`

Funkcija feof — primjeri

Kopiramo sadržaj datoteke in u datoteku out, znak po znak. Pretpostavimo da nema grešaka pri čitanju i pisanju (ne provjeravamo ferror, već samo feof).
Ukoliko kopiranje realiziramo:

```
do {
    c = fgetc(in);
    fputc(c, out);
} while (!feof(in));
```

Znak c == EOF upišemo prije testa !feof(int) u while petlji.

Istu pogrešku dobijemo i implementacijom:

```
while (!feof(in)) {
        c = fgetc(in);
        fputc(c, out); }
```

Prepoznavanje greške - primjer

Pišemo funkciju koja kopira sadržaj jedne datoteke u drugu liniju po liniju (obje datoteke su otvorene). Funkcija treba prepoznati i javiti greške prilikom čitanja i pisanja. Za prepoznavanje greške prilikom čitanja koristimo funkciju `ferror` (vraćamo 0 ili kod greške).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_LINE 129
4
5  int copy_file(FILE *in, FILE *out){
6      char buf[MAX_LINE]; /* Ulazni buffer. */
7      while (fgets(buf, MAX_LINE, in) != NULL)
8          if (fputs(buf, out) == EOF) {
9              fprintf(stderr,
10                 "\nGreska u pisanju.\n");
11             return 1; }
```

Prepoznavanje greške - primjer

```
12        if (ferror(in)) {
13           fprintf(stderr, "\nGreska␣u␣citanju.\n");
14           return 2; }
15         return 0;
16 }
```

Napomene

Operacije čitanja i pisanja možemo raditi i s tekstualnim i s binarnim datotekama. Potencijalna razlika kod operacija znak po znak: funkcije `fgetc` i `fputc`.

Izbor tipa datoteke (način otvaranja) ovisi o potrebama (efikasno spremanje i pristup ili mogućnost vizualnog pregledavanja).

Ulazno-izlazne operacije koje se izvode liniju po liniju (`fgets`, `fputs`, formatirano `fscanf`, `fprintf`) se prirodno izvode s tekstualnim datotekama.

Spomenutim funkcijama možemo čitati i pisati samo znakove (formatirano ili neformatirano), vrijednosti ostalih standardnih tipova (samo formatirano). Čitanje i pisanje objekata ostalih tipova ne možemo raditi na taj način već moramo zapisivati po komponentama (često nepraktično). Složene tipove možemo učinkovito spremati i čitati koristeći binarno čitanje i pisanje.

Binarne datoteke

U praksi često trebamo datoteke koje sadrže niz struktura određenog tipa ili niz podataka standardnog tip (`int`, `double` itd.) u internoj binarnoj reprezentaciji (bez pretvaranja u tekst). Tako izbjegavamo greške zaokruživanja koje nastaju pri formatiranom čitanju i pisanju realnih vrijednosti.

Opisanu funkcionalnost ostvarujemo otvaranjem datoteke kao binarne. Neformatirane ulazno-izlazne operacije realiziraju se posebnim funkcijama `fread` i `fwrite`, koje kopiraju sadržaj zadanog bloka byte-ova (kao niz znakova - niz podataka tipa `unsigned char`).

Binarno čitanje i pisanje

Funkcije za binarno (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` if `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

Čitanje cijelog polja cijelih brojeva iz datoteke:

```
1 int polje[10];
2 ...
3 fread(polje, sizeof(int), 10, fp);
```

Pisanje cijelog polja cijelih brojeva u datoteku:

```
1 int polje[10] = { ... };
2 ...
3 fwrite(polje, sizeof(int), 10, fp);
```

Binarno pisanje strukture — primjer

```
1  typedef struct {
2        int broj_racuna;
3        char ime[80];
4        double stanje;
5  } Racun;
6
7  Racun kupac = { 47, "Pero␣Peric", 200.00 };
8  fp = fopen("novi.dat", "wb");
9  ...
10 if (fwrite(&kupac, sizeof(Racun), 1, fp) != 1) {
11        fprintf(stderr, "Greska␣pri␣upisu.\n");
12        exit(1); }
```

Binarno čitanje i pisanje — komentari

Prednosti binarnog ulaza/izlaza:

- brzina - nema pretvaranja u tekst ili iz teksta,
- manja veličina zapisa - npr. zapis `int`-a u 4 byte-a, umjesto do 10 znakova i potencijalno predznak.

Nedostatak binarnog ulaza/izlaza:

- Ovisnost o arhitekturi računala i prevoditelju,
- nije čitljiv ljudima - binarna datoteka se ne može editirati običnim tekst-editorom.

U kombinaciji s funkcijama za pozicioniranje u datoteci (`ftell`, `fseek`), funkcije `fread` i `fwrite` služe i za direktni pristup podacima.

Sekvencijalni pristup podacima

Ulazno-izlazne operacije koje smo radili do sada, koristile su sekvencijalni pristup podacima u datoteci. Nakon otvaranja datoteke, početna pozicija ovisi o načinu otvaranja datoteke. Kod pisanja (`"w"`) i čitanja (`"r"`), pozicionirani smo na početak datoteke, a kod dodavanja (`"a"`) na kraj datoteke (iza svega što postoji u datoteci - ovo je ovisno o implementaciji). Nakon toga, svaka sljedeća ulazno-izlazna operacija nastavlja raditi točno tamo gdje je prethodna operacija završila - tj. stalno se krećemo u datoteci unaprijed.

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 prethodnim operacijama na toj datoteci. Trenutna pozicija se računa kao broj znakova (byte-ova) od početka datoteke (slično kao indeksi polja znakova). Nula označava početak datoteke (ispred prvog znaka). 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.

Svaka ulazno-izlazna operacija pomiče poziciju u datoteci unaprijed (u odnosu na trenutnu). Kad god napravimo operaciju čitanja ili pisanja, trenutna pozicija se povećava upravo za broj pročitanih ili napisanih znakova (byteova). Ako sami ne mijenjamo trenutnu poziciju, dobivamo sekvencijalno čitanje i pisanje (svaka operacija počinje gdje je prethodna stala). Trenutna pozicija se mijenja sama.

Direktni pristup podacima

Dozvoljeno je promijeniti vrijednost trenutne pozicije u datoteci. To radimo zadavanjem mjesta 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 (byte-u) u datoteci pristupamo direktno, slično kao u polju. Zato se ovaj način rada s datotekom zove direktni ili slučajni pristup podacima.

Realizacija direktnog pristupa slična je indeksiranju kod polja. Prije operacije zadajemo poziciju u datoteci, posebnom funkcijom za pozicioniranje u datoteci. Za direktni pristup podacima u C-u koristimo dvije funkcije: `ftell` - vraća trenutnu poziciju u datoteci i `fseek` - mijenja trenutnu poziciju u datoteci na zadanu poziciju.

Trenutna pozicija u datoteci - funkcija `ftell`

Deklaracija funkcije: `long int ftell(FILE *fp);`

Funkcija `ftell` vraća trenutnu poziciju u već otvorenoj datoteci na koju pokazuje `fp` (broj znakova, byte-ova, od početka te datoteke).

Izlazna vrijednost je:

- nenegativan broj u slučaju uspjeha
- $-1L$ u slučaju greške.

Vrijednost $0L$ znači da se nalazimo na početku datoteke. Dakle, povratna vrijednost je udaljenost od početka datoteke (u znakovima, byte-ovima).

Kod 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 Windowsima (Intel C, Code::Blocks) - trenutna pozicija je početak datoteke ($0L$), na Linuxu - trenutna pozicija je kraj datoteke (iza zadnjeg znaka, duljina datoteke u byte-ovima).

Promjena pozicije u datoteci — funkcija `fseek`

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

Argumenti funkcije `fseek` su:

- `fp` - pokazivač na već otvorenu datoteku,
- `offset` - zadani pomak u broju znakova (byte-ova),
- `origin` - indikator položaja ili ishodište od kojeg se broji pomak. Zadaje se jednom od 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` postavlja trenutnu poziciju u datoteci na koju pokazuje `fp` na `offset` znakova od zadanog ishodišta `origin`.

Funkcija fseek

Izlazna vrijednost funkcije je:

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

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

```
fseek(fp, 0L, SEEK_SET); /* POCETAK datoteke. */
fseek(fp, 0L, SEEK_END); /* 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:

- `fp`, `0L`, `SEEK_SET` - idi na početak datoteke,
- `fp`, `0L`, `SEEK_END` - idi na kraj datoteke,
- `fp`, `0L`, `SEEK_CUR` - ostani na trenutnoj poziciji,
- `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 pozivom funkcije `void rewind(FILE *fp);`
Ovaj poziv ekvivalentan je s:

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

Osim pozicioniranja na početak datoteke, brišemo i indikatore za kraj datoteke i za grešku.

Funkcija `ftell` - primjer

Pretpostavimo da postoji datoteka koja sadrži 4 znaka:

```
a b c d
```

Prvo otvorimo tu datoteku za (tekstualno ili binarno) čitanje, a zatim 6 puta ponovimo sljedeće: a) nađemo trenutnu poziciju u toj datoteci (funkcija `ftell`) i ispišemo ju (na `stdout`), b) učitamo sljedeći znak iz te datoteke i ispišemo ga (na `stdout`).

```
1  /* Sekvencijalno citamo tu datoteku. */
2  for (i = 0; i < 6; ++i) {
3      printf("␣Pozicija:␣%ld,", ftell(fp));
4
5      if ((c = fgetc(fp)) >= 0)
6          printf("␣znak␣=␣%2c\n", c); /* Znak. */
7      else
8          printf("␣znak␣=␣%2d\n", c); /* Broj. */
9      }
```

Funkcija `ftell` - primjer

Izlaz 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`.

`ftell` - varijacije

1) Modificirajte program tako da ispisuje trenutne pozicije prije pisanja svakog znaka (treba zapisati i početnu poziciju).

2) Nakon kreiranja zadane datoteke s 4 znaka, treba:

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

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

4) 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).

Naopako kopiranje (invertiranje) datoteke

Pišemo program koji naopako kopira sadržaj jedne datoteke u drugu.

Npr. koristeći datoteku:

```
a b c d
```

trebamo stvoriti datoteku:

```
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.

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

Naopako kopiranje datoteke

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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void){
4
5    char *in_name = "freverse.in";
6    char *out_name = "freverse.out";
7
8    FILE *in, *out;
9    long file_pos, pomak = 0L;
10   if ((in = fopen(in_name, "rb")) == NULL) {
11       fprintf(stderr, "Ne␣mogu␣citati␣iz:␣%s!\n",
12                   in_name);
13       exit(1); }
```

Naopako kopiranje datoteke

```
1   if ((out = fopen(out_name, "wb")) == NULL) {
2       fprintf(stderr, "Ne␣mogu␣pisati␣u:␣%s!\n",
3                   out_name);
4       exit(2); }
5
6   do {/* Datoteku kopiramo naopako. */
7 /* Pomak unazad od kraja. */
8       if (fseek(in, --pomak, SEEK_END)) break;
9 /* Zapamti poziciju i ucitaj znak. */
10      file_pos = ftell(in);
11      fputc(fgetc(in), out);
12 /* Sad je pozicija narasla za 1L. */
13  } while (file_pos != 0L);
14
15  fclose(in);
16  fclose(out);
17  return 0; }
```

Naopako kopiranje datoteke

Kod pomaka unatrag od kraja datoteke, test:
`if (fseek(in, -pomak, SEEK_END)) break;`
je zaštita od prazne ulazne datoteke. Bez tog testa program ne radi.

Za ulaznu datoteku od 33 znaka:

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

izlazna datoteka ima 33 znaka:

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

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

Napravite istu stvar sekvencijalnim pristupom podacima. Složenost je tada kvadratna u duljini datoteke.

Naopako kopiranje datoteke — napomene

Obje datoteke moraju biti otvorene kao binarne. U protivno, čim ulazna datoteka ima bar jedan znak za kraj reda, program ne radi dobro kod sustava kod kojih postoji razlika između binarnih i tekstualnih datoteka (npr. Windows).

Neće dobro raditi niti program koji naopako ispisuje sadržaj binarne datoteke na `stdout`, te preusmjerava ispis u izlaznu datoteku.

Na Windowsu se znak za kraj linije'`\n`' zapisuje kao dva znaka '`\r`' '`\n`', stoga dolazi do umjetnog dodavanja znakova u izlaznu datoteku.

Čitanje i pisanje u istoj datoteci

Datoteku možemo otvoriti tako da je dozvoljeno i čitanje iz datoteke i pisanje u tu datoteku. Takav način rada s datotekom definiramo tako da u `file_mod` stringu navedemo znak `+`. Treba biti oprezan pri prijelazu s čitanja na pisanje i obratno (treba korektno isprazniti spremnik za komunikaciju između programa i datoteke).

Pri prijelazu s čitanja na pisanje (između operacija) treba pozvati funkciju za pozicioniranje u datoteci (`fseek`, `rewind` ili `fsetpos`), osim ako je čitanje stiglo do kraja datoteke ili pozvati funkciju `fflush` za pražnjenje (pisanje) sadržaja spremnika u datoteku.

Pisanje spremnika u datoteku — funkcija `fflush`

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

Ako `fp` pokazuje na izlaznu datoteku (tj. zadnja operacija je bila pisanje u tu datoteku), `fflush` piše u tu datoteku onaj dio sadržaja spremnika koji do tada nije bio fizički zapisan u nju (prazni spremnik u datoteku). Ako `fp` pokazuje na ulaznu datoteku (tj. zadnja operacija je čitanje iz te datoteke), efekt poziva funkcije `fflush` nije definiran (nema smisla).

Poziv oblika `fflush(NULL);` prazni spremnike za sve izlazne datoteke u tom trenutku.

Izlazna vrijednost funkcije `fflush` je:

- nula - ako je uspješno ispraznila spremnik,
- `EOF` - ako je prilikom pisanja došlo do greške.

Regularan završetak programa (uključujući pozivom funkcije `exit`) automatski prazni spremnike za sve otvorene (izlazne) datoteke.

Dodavanje bonusa na račun

Telefonski račun opisan je strukturom tipa `Racun`:

```
1  typedef struct {
2      int tel_broj;
3      char vlasnik[20];
4      double stanje;
5  } Racun;
6
7  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

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`.

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`):

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

Dodavanje bonusa na račun

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);
```

Funkcija dodaj_bonus glasi:

```
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

```
11  /* Pozicioniranje ispred n-tog zapisa. */
12    file_pos = (long) (n - 1) * size;
13
14    if (fseek(racuni, file_pos, SEEK_SET)) {
15          fprintf(stderr,
16                  "Greska u fseek, n = %d.\n", n);
17    printf("Greska u fseek, n = %d.\n", n);
18    fclose(racuni);
19    return;}
20
21   /* Ucitaj zapis u strukturu. */
22    if (fread(&kor, size, 1, racuni) != 1)
23        if (ferror(racuni)) {
24            fprintf(stderr, "Greska u citanju.\n");
25            exit(2); }
```

Dodavanje bonusa na račun

```
26          else if (feof(racuni)) {
27              fprintf(stderr,
28              "Kraj datoteke, n = %d.\n", n);
29          printf("Kraj datoteke, n = %d.\n", n);
30          fclose(racuni);
31          return; }
32
33  /* Azuziraj stanje i napisi novi zapis. */
34     if (kor.stanje > 0) {
35        kor.stanje = kor.stanje + bonus;
36        fseek(racuni, -size, SEEK_CUR);
37     if (fwrite(&kor, size, 1, racuni) != 1) {
38        fprintf(stderr, "Greska u pisanju.\n");
39        exit(3); } }
40     fclose(racuni);
41     return; }
```

Dodavanje bonusa na račun — rezultati

Pretpostavimo da u datoteci `racuni.dat` imamo sadržaj:

```
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
```

Zatim, dodajemo bonus sljedećim zapisima:

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

Dodavanje bonusa na račun — rezultati

Nova datoteka `racuni.dat` ima sadržaj:

```
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

Pišemo program koji okreće naopako (invertira) sadržaj jedne zadane datoteke, međutim invertirani sadržaj upisuje u tu istu datoteku (bez korištenja pomoćne datoteke).

Ako datoteka na početku ima oblik:

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

nakon izvođenja programa treba imati oblik:

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

Datoteku treba otvoriti tako da je dozvoljeno istovremeno čitanje i pisanje u toj datoteci.

Invertiranje datoteke

Postupamo slično kao u polju ili stringu. Jedini problem je naći polovište datoteke, tj. prepoznati kada smo gotovi. Najlakši način je pronaći duljinu datoteke (odlaskom na kraj) te raspoloviti dobivenu duljinu.

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

Uz to trebamo raditi pažljivo pozicioniranje s odgovarajućim pomakom.

Invertiranje datoteke

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void) {
5      char *dat_name = "finvert.dat";
6      FILE *dat;
7      long file_pola, pomak = 0L;
8      int ch_1, ch_2;
9
10     if ((dat = fopen(dat_name, "rb+")) == NULL) {
11         fprintf(stderr, "Ne␣mogu␣citati␣iz:␣%s!\n",
12                                          dat_name);
13       exit(1); }
14 /* Duljina i poloviste datoteke. */
15     fseek(dat, 0L, SEEK_END);
16     file_pola = ftell(dat) / 2L;
```

Invertiranje datoteke

```
17 /* Datoteku invertiramo. */
18    while (pomak < file_pola) {
19 /* Pomak unaprijed od pocetka.
20 Ucitaj prvi znak. */
21       fseek(dat, pomak, SEEK_SET);
22       ch_1 = fgetc(dat);
23       /* Pomak unazad od kraja.
24 Ucitaj drugi znak. */
25       fseek(dat, -pomak - 1L, SEEK_END);
26       ch_2 = fgetc(dat);
27       /* Pomak za jedno mjesto unazad od
28 trenutnog. Napisi prvi znak. */
29       fseek(dat, -1L, SEEK_CUR);
30       fputc(ch_1, dat);
```

Invertiranje datoteke

```
31  /* Pomak unaprijed od pocetka.
32  Napisi drugi znak. */
33        fseek(dat, pomak, SEEK_SET);
34        fputc(ch_2, dat);
35        ++pomak; /* Povecaj pomak! */
36       }
37        fclose(dat);
38        return 0; }
```

Za ulaznu datoteku sadržaja:

`Ja sam mala Ruza, mamima sam kci.`

Promijenjena datoteka ima sadržaj:

`.ick mas amimam ,azuR alam mas aJ`

Ukoliko kod izvršimo dva puta, dobijemo polaznu datoteku.

Složenost programa je linearna u duljini datoteke.

Zadaci

1) 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!

- Pronađite broj računa u datoteci (duljina datoteke podijeljena s veličinom svake strukture),
- Daljnji postupak je kao kod okretanja datoteke (pojedinačnih znakova) osim što svaki pojedini objekt ima duljinu `size`, pa čitanje vršimo funkcijom `fread` (a ne `fgetc`), a pisanje vršimo funkcijom `fwrite` (a ne `fputc`).

Zadaci

2) Napišite program koji (uzlazno) sortira datoteku `racuni.dat`, koja sadrži niz struktura tipa `Racun` 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.

Izaberite neki algoritam sortiranja na polju (proizvoljan) koji koristi jedno polje, zatim sve osnovne operacije realizirajte malim funkcijama na datoteci.

Potencijalno korisne funkcije: a) broj objekata u zadanoj datoteci, b) citanje i-tog objekta iz datoteke (u strukturu), c) pisanje i-tog objekta (iz strukture) u datoteku, d) usporedba dva objekta (strukture), e) zamjena dva objekta u datoteci (pažljivim čitanjem i pisanjem - može zasebnom funkcijom kao u invertiranju).
Napredno: implementirajte `QuickSort` algoritam na datoteci.