

# 3

## Datoteke

Ovo poglavlje posvećeno je radu sa datotekama u programskom jeziku C. Prvo su uvedeni osnovni pojmovi vezani za spoljašnju memoriju, sistem datoteka, reprezentaciju podataka u datotekama, organizaciju i način pristupa podacima. Zatim su prikazane mogućnosti za rad sa datotekama u programskom jeziku C. Opisana je manipulacija tekstualnim i binarnim datotekama i ilustrovana primjena osnovnih funkcija iz standardne biblioteke za rad sa datotekama. Na kraju su još prikazani i standardni tokovi i ilustrovane mogućnosti njihovog korišćenja.

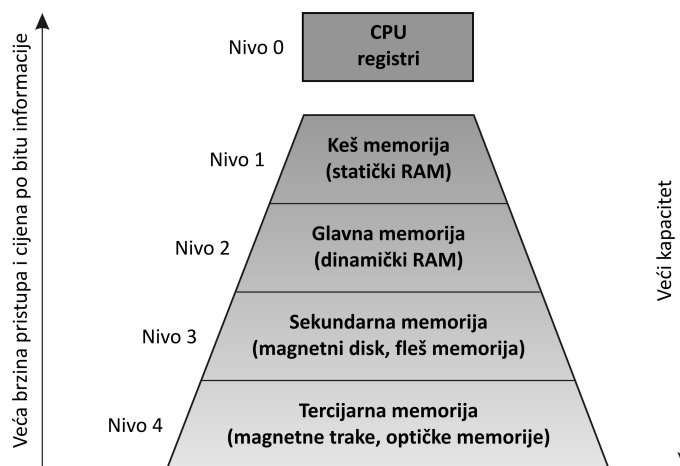
### 3.1. Osnovno o datotekama

#### 3.1.1. Hijerarhijska organizacija memorije

Tokom izvršavanja programa podaci prolaze kroz različite nivoe **hijerarhijske organizacije memorije**. Kao što se vidi sa sl. 3.1, prema višim memorijskim nivoima kapacitet memorije se smanjuje, a brzina pristupa i jedinična cijena memorije (cijena po bitu) povećavaju. Cilj hijerarhijske organizacije jeste da se u toku rada postigne brzina pristupa podacima što bliža prvom nivou, a da ukupna cijena i kapacitet memorijskog sistema budu što bliže cijeni i kapacitetu sekundarnog memorijskog nivoa.

Na najvišem hijerarhijskom nivou nalaze se **registri** procesora. Iako predstavljaju memorijske elemente, registri nisu dio **memorijskog podsistema računara** već **centralne procesne jedinice**<sup>1</sup>, pa je iz tog razloga registarska memorija označena nultim nivoom.

<sup>1</sup> engl. *Central Processing Unit* – CPU



Slika 3.1: Hijerarhijska organizacija memorije u računarskom sistemu

CPU direktno pristupa podacima na prva dva memorijska nivoa, referenciranjem odgovarajuće memorijske adrese u okviru mašinskih instrukcija procesora. Da bi se ubrzao pristup podacima u **glavnoj memoriji**, kopije tih podataka drže se u brzim **keš memorijama** i procesor, zapravo, pristupa tim kopijama u kešu, a ne stvarnim podacima u glavnoj memoriji. Keš može biti realizovan na jednom ili na više nivoa. Glavna memorija takođe može da se organizuje u više modula i u više podnivoa. Prva dva memorijska nivoa često se označavaju terminom **primarna memorija**. Ove memorije gube sadržaj sa isključenjem napajanja, pa se nazivaju i **memorije sa privremenim sadržajem** (engl. *volatile memory* ili *volatile storage*).

Da bi procesor pristupio podacima na **sekundarnim** (nivo 3) i **tercijarnim** memorijama (nivo 4), neophodno je da se prvo napravi transfer podataka sa tih medija u glavnu memoriju. Ove memorije ne gube sadržaj sa isključenjem napajanja, pa se nazivaju i **memorije sa stalnim sadržajem** (engl. *non-volatile memory*). Često se još za ova dva nivoa koristi termin **spoljašnja** ili **spoljna memorija**. Sekundarne memorije nazivaju se i *on-line* memorijama, jer se podacima može pristupiti direktno i u bilo koje vrijeme rada računarskog sistema, dok tercijarne memorije zahtijevaju određenu akciju operatera ili drugog uređaja prije nego što podaci postanu raspoloživi (*off-line* memorije). Tercijarne memorije koriste se za zaštitne i arhivske kopije podataka.

Memorije sa stalnim sadržajem realizuju se različitim tehnologijama i postoje u obliku različitih komponenata, pri čemu način pristupa podacima zavisi od tehničko-tehnoloških karakteristika komponente. S obzirom na način pristupa, memorije sa stalnim sadržajem dijelimo na **memorije sa direktnim pristupom** i **memorije sa sekvencijalnim pristupom**.

**Memorije sa direktnim pristupom** omogućavaju direktan pristup željenoj lokaciji, bez potrebe da se prije toga pristupi nekoj drugoj lokaciji. Takve komponente su tvrdi diskovi, optički diskovi, memorijske kartice itd.

**Memorije sa sekvencijalnim pristupom** ne omogućavaju direktan pristup proizvoljnoj lokaciji na kojoj je upisan podatak, već je neophodno pristupiti svim prethodno upisanim podacima. Takve su npr. magnetne trake.

### 3.1.2. Organizacija spoljašnje memorije

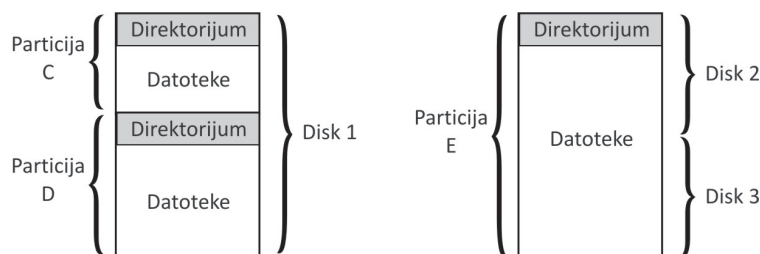
Računar tipično posjeduje više fizičkih jedinica spoljašnje memorije. Standardno računari posjeduju bar jednu jedinicu magnetnih diskova (engl. *hard disk drive*). Diskovi se uobičajeno dijele na particije (engl. *partition*), kojima se pristupa kao nezavisnim jedinicama. Svaka takva jedinica obično se naziva volumen (engl. *volume*). Svaka jedinica (volumen) ima odgovarajuće simboličko ime u računarskom sistemu. U različitim operativnim sistemima, jedinice se imenuju na različite načine. Na primjer, u Windows operativnim sistemima jedinice se označavaju u obliku X:, gdje je X veliko slovo abecede. Tako prva particija hard diska tipično ima oznaku C:, a druga oznaku D:. U UNIX/Linux operativnim sistemima jedinice se označavaju u obliku /dev/volumen, gdje je volumen (višeslovni) naziv jedinice, npr. /dev/voll, /dev/hdisk0 i sl.

#### Sistem datoteka

Podaci u spoljašnjoj memoriji organizovani su u **sistem datoteka** (engl. *file system*). Sistem datoteka sačinjavaju: kolekcija datoteka i direktorijumska struktura. Datoteke sadrže podatke od interesa za korisnika, a direktorijumi omogućavaju organizovanje i sadrže informacije o datotekama u sistemu. Organizacija sistema datoteka ilustrovana je na sl. 3.2.

**Datoteka** (engl. *file*) je imenovana kolekcija uzajamno povezanih podataka u spoljašnjoj memoriji. Posmatrano iz perspektive korisnika, datoteka je najmanja logička cjelina spoljašnje memorije – svaka manipulacija spoljašnjom memorijom svodi se na manipulaciju datotekama. Korisnik ne može da upiše podatke u spoljašnju memoriju, a da oni ne pripadaju nekoj datoteci, niti može da pročita podatke iz spoljašnje memorije koji ne pripadaju nekoj datoteci.

Datoteka predstavlja sekvencu bajtova u spoljašnjoj memoriji, koja sadrži odgovarajuću informaciju relevantnu za njenog kreatora. Podaci upisani u



Slika 3.2: Ilustracija organizacije sistema datoteka u sekundarnoj memoriji

neku datoteku mogu da se interpretiraju na različite načine. U datoteku mogu biti upisane različite vrste informacionog sadržaja. Prevashodno, datoteke sadrže programski kôd i podatke. Programske datoteke mogu da sadrže izvorni, objektni ili izvršni kôd, dok datoteke sa podacima mogu da sadrže različite vrste podataka (npr. tekstualne, numeričke, multimedijalne). Format podataka u datotekama takođe mogu da budu različiti – od slobodnog (npr. slobodan tekst) do strogo definisanog (npr. u nekim bazama podataka).

Svaka datoteka ima naziv. Naziv datoteke je obično niz alfanumeričkih znakova (npr. `program1.c`). U različitim operativnim sistemima važe različita pravila za imenovanje datoteka, kao što su: maksimalan broj znakova, dozvoljeni i nedozvoljeni znakovi, razlikovanje velikih i malih slova, itd. U nekim operativnim sistemima pravi se razlika između velikih i malih slova (npr. UNIX), tako da `Program.c` i `program.c` predstavljaju nazive dvije različite datoteke. Postoje i operativni sistemi (npr. Windows) u kojima se ne pravi ta razlika, tako da `Program.c` i `program.c` referenciraju istu datoteku.

Mnogi operativni sistemi podržavaju dvodjelne nazive datoteka, pri čemu su dijelovi razdvojeni tačkom (npr. `program.c`). Dio iza tačke naziva se *ekstenzija* i obično ukazuje na tip, odnosno vrstu informacionog sadržaja datoteke. Obično ekstenzija sadrži od jedan do tri znaka (`c`, `exe`, `mp3`, ...), ali postoje i sistemi (npr. UNIX i novije verzije Windows operativnog sistema) koji dozvoljavaju da ekstenzija ima veći broj znakova ili da postoje višestruke ekstenzije (npr. `program.c.Z`).

Nakon kreiranja, datoteka postaje nezavisna od programa kojim je kreirana ili korisnika koji je kreirao. Na primjer, neki drugi korisnik može da je otvori i pročita njen sadržaj u istom ili u nekom drugom programu, ili da je iskopira na neki drugi fizički medijum, ili pošalje elektronskom poštom. U svim navedenim situacijama datoteka zadržava svoj naziv.

### Organizacija sistema datoteka

Jedinice spoljašnje memorije mogu da sadrže velik broj datoteka (na hiljade, milione ili čak milijarde). Da bi se olakšao pristup, odnosno pronalaženje datoteke na fizičkom medijumu, svaki volumen tipično ima **direktorijum** (engl. *directory*) koji sadrži osnovne informacije o datotekama u pripadajućem volumenu, kao što su njen naziv, lokacija, veličina i tip.

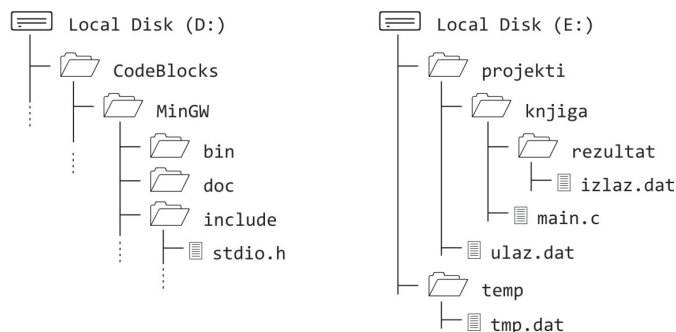
Postoje različiti načini za organizaciju direktorijuma. U savremenim operativnim sistemima primjenjuje se hijerarhijska organizacija, koja podrazumijeva postojanje korijenskog (engl. *root*) direktorijuma ili foldera (engl. *folder*), unutar kojeg korisnik može da kreira korisničke direktorijume koji omogućavaju grupisanje datoteka prema različitim kriterijumima. Na primjer, tipični direktorijumi u Windows operativnim sistemima su `Program Files` (podrazumijevano sadrži podatke o datotekama koje pripadaju instalisanim korisničkim programima) i `Windows` (sistemske datoteke). Korisnik dalje može da u okviru direktorijuma formira poddirektorijume (engl. *subdirectory*) ili podfoldere (engl. *subfolder*).

Posmatrano iz perspektive korisnika, direktorijum može da sadrži pod-direktorijume i datoteke, a svaki poddirektorijum dodatne poddirektorijume i datoteke itd. Zbog toga se uobičajeno kaže da neki direktorijum sadrži datoteke, iako direktorijum zapravo predstavlja hijerarhijski organizovanu simboličku tabelu koja sadrži samo podatke o datotekama u volumenu, a ne i same datoteke. Tako se, na primjer, kaže da je *MinGW* prevodilac instalisan u direktorijumu *MinGW*, dok se njegove komponente nalaze u odgovarajućim poddirektorijumima: razvojni alati u poddirektorijumu *bin*, zaglavlja (*.h* datoteke) u poddirektorijumu *include* itd.

### Putanje

Svaka datoteka ima **putanju** (engl. *path*) pomoću koje se specifikuje njena lokacija u sistemu datoteka (posmatrano iz perspektive korisnika). **Apsolutna putanja** reprezentuje put od korijenskog direktorijuma do željene datoteke, pri čemu se komponente putanje razdvajaju odgovarajućim separatorom. U različitim operativnim sistemima koriste se različiti separatori – u Windows sistemima to je znak `\` (*backslash*), a u UNIX/Linux sistemima to je znak `/` (*slash*). Dodatno, na početku može da se specifikuje i volumen, pa je opšti oblik apsolutne putanje u Windows sistemima `X:\put\do\datoteke\datoteka`, a u UNIX/Linux sistemima `/dev/vol/put/do/datoteke/datoteka`. Na primjer (sl. 3.3), ako je razvojno okruženje *CodeBlocks* instalisano u korijenskom direktorijumu na `D:` particiji, tada datoteka `stdio.h` ima apsolutnu putanju `D:\CodeBlocks\MinGW\include\stdio.h`.

**Relativna putanja** specifikuje lokaciju datoteke u odnosu na **radni direktorijum** (engl. *working directory* ili *current directory*). Korisnik može da izabere neki direktorijum kao radni. To je tipičan slučaj sa razvojnim okruženjima. Na primjer, prilikom kreiranja projekta u *CodeBlocks* razvojnom okruženju, programer specifikuje radni direktorijum – direktorijum u kojem će se nalaziti sve datoteke koje pripadaju datom projektu. Pretpostavimo da je korisnik izabrao `E:\projekti\knjiga` kao radni direktorijum (sl. 3.3). Sve datoteke koje se nalaze u tom direktorijumu jednostavno se referenciraju navođenjem njihovog imena. Tako `main.c` predstavlja relativnu putanju te datoteke.



Slika 3.3: Primjer hijerarhijske organizacije direktorijuma

Relativna putanja datoteke `izlaz.dat`, koja se nalazi u poddirektorijumu `rezultat`, specifikuje se u obliku `rezultat\izlaz.dat`. Za specifikaciju nadređenog (roditeljskog) direktorijuma koristi se simbol `..`, pa `..\ulaz.dat` predstavlja relativnu putanju datoteke `ulaz.dat` koja se nalazi u roditeljskom direktorijumu radnog direktorijuma, dok je `..\..\temp\tmp.dat` relativna putanja datoteke `tmp.dat` koja se nalazi u direktorijumu `temp`.

Treba imati u vidu da znak `\` u programskom jeziku C ima specijalnu ulogu. Zato u stringu, koji reprezentuje putanju datoteke, svaki znak `\` mora da se zamijeni sa `\\`, npr. "`..\..\temp\tmp.dat`".

### 3.1.3. Organizacija i pristup podacima u datotekama

Organizaciju datoteke možemo posmatrati na različitim nivoima apstrakcije. Na nivou operativnog sistema, datoteke se uobičajeno posmatraju kao **nestrukturisani nizovi (sekvence) bajtova**. Tako se datoteke tretiraju u Windows i UNIX/Linux operativnim sistemima. Ovakva organizacija pruža veliku fleksibilnost – operativni sistem vidi datoteku samo kao sekvencu bajtova, dok korisnički programi mogu da manipulišu datotekom na sebi svojstven način, tj. mogu da umeću, dodaju, mijenjaju i interpretiraju sadržaj. Postoje i drugačiji načini organizacije datoteke na nivou operativnog sistema, u kojima se podacima ne pristupa na nivou bajta nego u većim blokovima ili zapisima fiksne dužine.

#### Načini pristupanja podacima u datoteci

Postoje dva bitno različita načina pristupa datoteci prilikom čitanja i pisanja podataka: **direktni** i **sekvencijalni**.

**Direktni pristup** podrazumijeva da se podatak upisuje na proizvoljnu lokaciju u datoteci ili se čita sa proizvoljne lokacije u jednom koraku. Često se datoteke, u kojima se podacima pristupa direktno, nazivaju **direktne datoteke** ili **datoteke sa direktnim pristupom**. Za direktni pristup podacima neophodno je pozicioniranje u datoteci na odgovarajuću lokaciju. Direktno pozicioniranje u jeziku C realizuje se posebnim funkcijama za pozicioniranje.

**Sekvencijalni (ili slijedni) pristup** podrazumijeva da se podaci u datoteku upisuju redom, jedan iza drugog, u istom smjeru (unaprijed, kao na traci). Prilikom čitanja mora se pristupiti svim podacima koji su na fizički medijum upisani prije željenog podatka, da bi se na kraju pristupilo željenom podatku. Datoteke, u kojima se podacima pristupa sekvencijalno, nazivaju se **datoteke sa sekvencijalnim pristupom** ili **sekvencijalne datoteke**. Ovo je standardni način pristupa i organizacije datoteka u jeziku C.

#### Interpretacija sadržaja datoteke

Postoje dva bitno različita načina upisa podataka u datoteku, odnosno interpretacije upisanog sadržaja. Podatke u datoteku možemo da upisujemo u **formatiranom obliku** ili **neformatiranom obliku**.

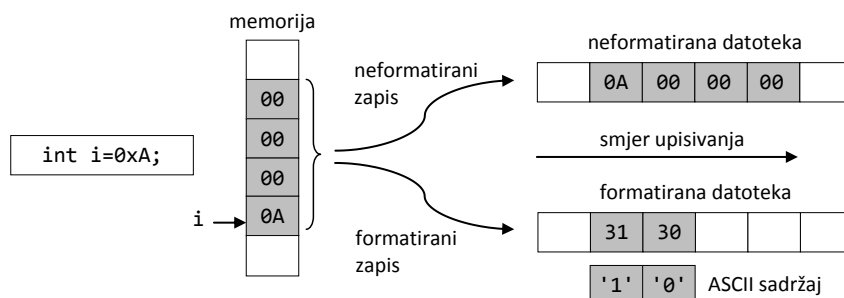
**Neformatirani zapis** podrazumijeva upis podatka u obliku kojim se podaci interno reprezentuju u računar doslovnim prepisivanjem binarnog sadržaja iz memorije u datoteku. Analogno neformatiranom upisu, neformatirano čitanje podrazumijeva da se sadržaj iz datoteke kopira bajt po bajt u memoriju. Neformatirano zapisivanje podataka u datoteku tipično rezultuje sadržajem koji je čovjeku nečitljiv (ako sadržaj prikazemo u nekom programu za rad sa tekstom). Jezik C raspolaže funkcijama koje omogućavaju neformatirano upisivanje i čitanje podataka.

**Formatirani zapis** podrazumijeva upis podatka u obliku koji je razumljiv čovjeku, slično ispisu teksta na standardnom izlazu pomoću funkcije `printf`. Analogno upisu, formatirano čitanje iz datoteke podrazumijeva čitanje podataka slično čitanju podataka sa standardnog ulaza pomoću funkcije `scanf`. S obzirom na to da se podaci u datoteku upisuju u obliku niza znakova (upisuju se odgovarajući binarni ekvivalenti u skladu sa ASCII kodom), svi podaci koji već nisu znakovi ili stringovi, prije upisa moraju da se konvertuju u odgovarajuću sekvencu znakovnih podataka. Analogno, i prilikom čitanja iz datoteke može biti potrebna konverzija znakovnih podataka. Formatirano upisivanje i čitanje podataka u jeziku C realizuje se standardnim funkcijama koje realizuju potrebne konverzije.

U zavisnosti od načina upisa podataka u datoteku i interpretacije sadržaja datoteke, razlikujemo **formatirane datoteke** i **neformatirane datoteke**. Ova podjela je striktna u nekim programskim jezicima, ali u jeziku C nije – na primjer, podatke možemo upisati formatirano, a čitati neformatirano.

#### Primjer 3.1:

Pretpostavimo da je u primarnoj memoriji smještena cjelobrojna promjenljiva `i` čija je vrijednost `0xA`, kao što je ilustrovano na sl. 3.4 (koristi se LE konvencija).



Slika 3.4: Formatirani i neformatirani zapis u datoteku

Na slici je prikazan sadržaj datoteke kao rezultat neformatiranog, odnosno formatiranog upisa. U neformatiranoj datoteci bajtovi iz memorije redom su prepisani (počevši od najniže adrese), a u formatiranoj datoteci redom su upisani bajtovi koji reprezentuju cifre 1 (`0x31`) i 0 (`0x30`), jer je dekadaska vrijednost promjenljive `i` jednaka 10.

## 3.2. Podrška za rad sa datotekama u jeziku C

### 3.2.1. Klasifikacija datoteka u jeziku C

Datoteka predstavlja sekvencu ili niz bajtova koji su zapisani formatirano (čovjeku čitljivo) ili neformatirano (čovjeku nečitljivo). U jeziku C ne postoji striktna podjela na formatirane i neformatirane datoteke. Prema ANSI standardu, datoteke se dijele na **tekstualne** i **binarne**.

**Binarna datoteka** je niz jednobajtnih podataka (niz podataka tipa `char`). Iako se svaka datoteka može posmatrati kao binarna, obično se pod binarnim datotekama podrazumijevaju datoteke koje su formirane **neformatiranim načinom** zapisivanja podataka. S obzirom na to da su podaci u datoteku upisani neformatirano, samo program (programer) koji je formirao datoteku može uspješno da interpretira njen binarni sadržaj, jer jedino on zna šta je upisano i kako. Neki primjeri binarnih datoteka su:

- izvršni programi (`exe`, `com`, ...), npr. `program.exe`;
- audio (`wav`, `m4a`, `raw`, ...), npr. `audio.wav`;
- video (`avi`, `mpeg`, `vob`, ...), npr. `video.avi`;
- slike (`jpg`, `tif`, `eps`, `bmp`, ...), npr. `slika.jpg`.

**Tekstualne datoteke** formiraju se **formatiranim upisivanjem** podataka. Tekstualna datoteka ima strukturu koja karakteriše tekst – sadrži redove (linije) "čitljivog" (korisniku prepoznatljivog) teksta. Svaki red predstavlja niz znakova koji završava znakom za kraj reda (engl. *End-Of-Line* – EOL). Odranije znamo da se u jeziku C terminator reda reprezentuje znakom `'\n'`. Stvarna reprezentacija terminatora reda u datoteci zavisi od operativnog sistema. Za reprezentaciju terminatora reda koriste se sljedeći znakovi:

- `<CR>` ("Carriage Return", ASCII kod: 13 / `0x0d`),
- `<LF>` ("Line Feed", ASCII kod: 10 / `0x0a`),

pri čemu se EOL reprezentuje sa jednim ili sa dva znaka:

- Windows, DOS: `<CR><LF>`;
- Unix, Linux: `<LF>`;
- Mac OS: `<CR>`.

Primjeri tekstualnih datoteka su:

- "plain-text" datoteke formirane editorima teksta, npr. `readme.txt`;
- datoteke sa izvornim kodom u nekom programskom jeziku, npr. `main.c`;
- XML, HTML i druge datoteke.

Svaka datoteka može da se posmatra kao binarna, jer je svaka datoteka samo niz bajtova zapisanih na neki način. I tekstualna datoteka je binarna, ali binarna ne mora da bude tekstualna.



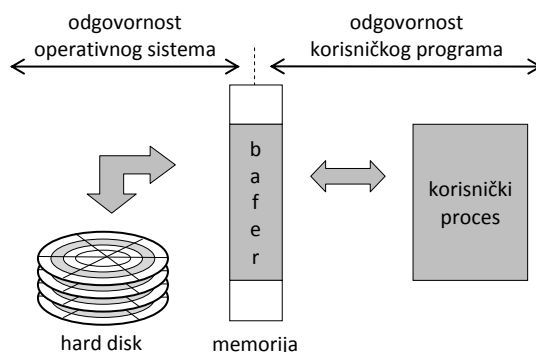
### 3.2.2. Pristup datotekama u jeziku C

Ranije je navedeno da razlikujemo sekvencijalni i direktni pristup datotekama. U programskom jeziku C sve datoteke se tretiraju kao sekvencijalne, tj. kao uzastopni nizovi bajtova koji se uobičajeno nazivaju **tokovi** (eng. *stream*). Komunikacija sa bilo kojim perifernim uređajem reprezentuje se tokovima. Postoji mogućnost direktnog pristupa i ona se realizuje pozicioniranjem na željeni bajt u datoteci.

#### Baferovanje

Komunikacija sa sekundarnom memorijom, u kojoj su fizički smještene datoteke, mnogo je sporija nego komunikacija sa primarnom memorijom. Zbog toga se tipično, iz programa napisanih u višim programskim jezicima, datotekama ne pristupa direktno, upisujući ili čitajući znak po znak, nego se koristi tzv. **baferovani pristup**. Bafer je dio primarne memorije koji se koristi za privremeno smještanje ulaznih ili izlaznih podataka. Prilikom čitanja podataka iz datoteke (sa tastature ili iz nekog drugog ulaznog uređaja), podaci se prvo smještaju u bafer, a zatim su dostupni datom programu. Slično, prilikom upisa podataka u datoteku, podaci se prvo smještaju u bafer, a zatim upisuju u datoteku. Princip baferovanja ilustrovan je na sl. 3.5.

U praksi se primjenjuju dva načina baferovanja: **linijsko baferovanje** (engl. *line buffering*) i **potpuno baferovanje** (engl. *full buffering*). Kod linijskog baferovanja – u baferu se drži jedan red, a kod potpunog baferovanja – u baferu se drži blok podataka proizvoljne veličine. Na primjer, ako se koristi linijsko baferovanje, operativni sistem će podatke upisati u datoteku kad se unese znak za novi red. Većina tokova je potpuno baferovana, i to je obično najbolje rješenje. Međutim, ne može komunikacija sa svim ulazno-izlaznim uređajima da se realizuje potpunim baferovanjem – na primjer, linijsko baferovanje mora da se koristi za komunikaciju sa uređajima kojima se ostvaruje interakcija sa korisnicima, kao što su standardni ulaz i standardni izlaz.



Slika 3.5: Baferovanje ulazno-izlazne komunikacije

Svrha bafera je smanjenje broja pristupa perifernim uređajima i povećanje efikasnosti ulazno-izlaznih uređaja. Bafer se nalazi na strani računara, kao dio operativnog sistema koji je zadužen za komunikaciju sa perifernim uređajima. Pored toga, i većina perifernih uređaja ima sopstveni bafer (ponekad i keš) sa potpuno istom svrhom (ovo se naziva *"double-buffer"* komunikacija).

Uobičajena manipulacija datotekama u jeziku C je baferovana. Međutim, komunikacija sa datotekama ne mora da se zasniva na baferovanju. Tada operativni sistem podatke upisuje u datoteku što je prije moguće. U jeziku C moguć je i nebaferovan pristup, a realizuje se pomoću posebnih funkcija niskog nivoa (engl. *low-level file functions*).

### Pristup baferu

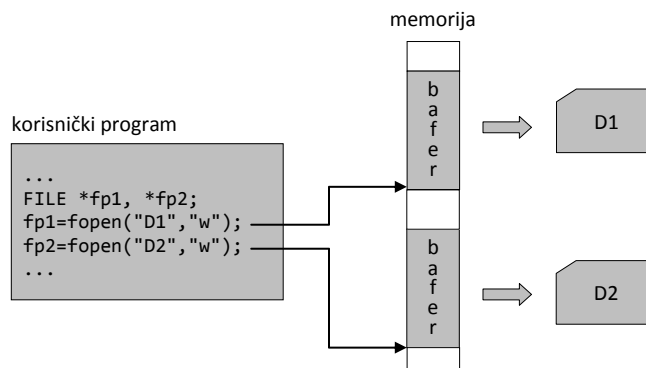
Datoteka zaglavlja za standardni ulaz i izlaz (`<stdio.h>`) sadrži definiciju strukture `FILE`, koja apstrahuje pristup baferu i sadrži sve ostale podatke potrebne za komunikaciju sa datotekom iz korisničkih C programa koji koriste baferovani pristup datotekama.

Za rad sa datotekama nije neophodno poznavanje elemenata strukture `FILE`. U nastavku su ukratko prikazani njeni najznačajniji elementi:

- naziv datoteke;
- režim rada sa datotekom (npr. čitanje i/ili pisanje);
- status nakon izvršavanja operacije (greška ili kraj datoteke);
- pokazivač trenutne pozicije u datoteci;
- stvarna adresa bafera u memoriji;
- trenutna pozicija u baferu (na osnovu koje se odlučuje o novom čitanju ili upisu bafera).

Svaka datoteka, kojoj pristupamo iz C programa, treba da ima svoj bafer – jedan slog tipa `FILE`. Bafer je dinamički objekat (smješten u dinamičkoj zoni memorije) i pristupa mu se indirektno – pomoću pokazivača (tzv. *"file pointer"*). Da bismo koristili datoteku, treba definisati pokazivač na tip `FILE`:

```
FILE *fp;
```



Slika 3.6: Indirektan pristup datoteci preko bafera

Kao što znamo od ranije, odmah po definiciji, ovaj pokazivač ne pokazuje na konkretan bafer. Konkretna bafera biće kreiran operacijom "otvaranja" datoteke – pozivom funkcije `fopen`, kao što je ilustrovano na sl. 3.6. Na kraju rada s datotekom, datoteku je neophodno zatvoriti – pozivom funkcije `fclose`. Tada se sadržaj bafera po potrebi upisuje u datoteku, a zauzeti prostor u dinamičkoj zoni memorije oslobađa.

### 3.2.3. Operacije nad datotekom

Operacije nad datotekom su:

- **otvaranje**
  - kreiranje bafera,
  - uspostavljanje veze sa fizičkom datotekom,
  - definisanje režima rada (otvaranje nove/postojeće datoteke, dodavanje/prepisivanje zapisa, ...);
- **pristup datoteci**
  - čitanje iz datoteke (sa ili bez konverzije),
  - pisanje u datoteku (sa ili bez konverzije);
- **pozicioniranje na željeni zapis u datoteci**
- **provjera statusa**
  - dobijanje informacije o eventualnim greškama u toku rada,
  - dobijanje informacije o kraju datoteke,
  - dobijanje informacije o trenutnoj poziciji pointera za čitanje/upis;
- **zatvaranje**
  - raskidanje veze sa fizičkom datotekom,
  - dealokacija bafera.

Svaka operacija nad datotekom izvodi se pozivom odgovarajuće standardne funkcije. Deklaracije (prototipovi) standardnih funkcija za baferovani rad sa datotekama, nalaze se u zaglavlju `<stdio.h>`.

### 3.2.4. Otvaranje datoteke

Otvoravanje datoteke je prva operacija koja se izvodi nad datotekom. Ako datoteka nije otvorena, nije joj moguće pristupiti, tj. nije moguće u nju upisati niti iz nje čitati podatke. Otvaranje datoteke je operacija kojom se uspostavlja veza sa fizičkom datotekom i kreira pripadajući dinamički bafer.

Otvoravanje datoteke vrši se pozivom funkcije `fopen`, čiji je prototip:

---

```
FILE *fopen(const char *ime, const char *rezim);
```

---

**Parametri funkcije** `fopen` su:

- **ime** – string koji reprezentuje naziv datoteke, npr.  
`"readme.txt"`,  
`"podfolder\\fajl.txt"` (relativna putanja),  
`"C:\\folder\\podfolder\\fajl.txt"` (apsolutna putanja).
- **režim** – string koji definiše režim rada (način korišćenja datoteke)  
 osnovni režimi (tabela 3.1):  
`"r"` / `"w"` / `"a"` (*read/write/append* = čitanje/pisanje/dodavanje)  
 kombinovani režimi (tabela 3.2):  
`"r+"` / `"w+"` / `"a+"`  
 rad sa binarnim datotekama (dodati 'b'):  
`"rb"` / `"r+b"` / `"wb"` / `"w+b"` / `"ab"` / `"a+b"`

**Rezultat izvršavanja funkcije** `fopen`:

- uspješno otvaranje: adresa bafera koji je pridružen otvorenoj datoteci;
- neuspješno otvaranje: `NULL`.

**Tabela 3.1:** Osnovni režimi rada

Režim	Opis
"r"	Otvaranje datoteke u režimu za čitanje iz postojeće datoteke. Ako datoteka ne postoji, <code>fopen</code> vraća <code>NULL</code> .
"w"	Otvaranje datoteke u režimu za upisivanje (prepisivanje). Ako datoteka postoji, piše preko postojećeg sadržaja. Ako datoteka ne postoji, otvara novu.
"a"	Otvora datoteku u režimu za dodavanje (na kraj datoteke). Ako datoteka ne postoji, otvara novu.

**Tabela 3.2:** Kombinovani režimi rada

Režim	Opis
"r+"	Otvaranje datoteke i za čitanje i za pisanje. Ako datoteka ne postoji, <code>fopen</code> vraća <code>NULL</code> .
"w+"	Otvaranje datoteke i za čitanje i za pisanje. Ako datoteka postoji, piše preko postojećeg sadržaja. Ako datoteka ne postoji, otvara novu.
"a+"	Otvaranje datoteke i za čitanje i za dodavanje (na kraj datoteke). Ako datoteka ne postoji, otvara novu.

**Uobičajeni programski obrazac** za otvaranje datoteke:

---

```
FILE *fp;
if (fp = fopen("ime", rezim))
    // ... datoteka uspješno otvorena
else
    // ... datoteka nije uspješno otvorena
```

---

**Uobičajene greške** kod otvaranja datoteke su:

- pogrešno ime datoteke (u datom operativnom sistemu);
- pokušaj da se nepostojeća datoteka otvori u režimu za čitanje;
- pokušaj da se datoteka otvori u režimu za pisanje, a medijum (npr. CD ROM) to ne omogućava;
- pokušaj da se datoteka otvori u režimu za pisanje, a dati korisnik nema na to pravo;
- ako se ne želi prepisivanje postojećeg sadržaja, a datoteka se ipak otvori u režimu "w", postojeći sadržaj biće izgubljen.

### 3.2.5. Zatvaranje datoteke

Zatvaranje datoteke je posljednja operacija koja se izvodi nad datotekom. Zatvaranjem datoteke raskida se veza sa fizičkom datotekom, sadržaj bafera upisuje u datoteku (ako je potrebno) i dealocira prostor u dinamičkoj zoni memorije. Zatvaranje se vrši pozivom funkcije `fclose`, čiji je prototip:

---

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

---

**Parametar funkcije** `fclose`:

- **fp** – pokazivač na neku otvorenu datoteku.

**Rezultat izvršavanja** funkcije `fclose`:

- uspješno zatvaranje: **0**;
- neuspješno zatvaranje: **EOF**.  
    **EOF** je konstanta definisana u `<stdio.h>` i ima vrijednost -1.  

```
greska = fclose(fp) == EOF;
```

**Uobičajeni programski obrazac** za otvaranje i zatvaranje datoteke:

---

```
FILE *fp;
if (fp = fopen("ime", rezim))
{
    // rad sa otvorenom datotekom
    fclose(fp);
}
```

---

**Primjer 3.2:**

Sa standardnog ulaza učitati naziv datoteke i režim rada, pa otvoriti traženu datoteku u zadatom režimu. U zavisnosti od uspješnosti otvaranja, ispisati odgovarajuću poruku na standardni izlaz i zatvoriti datoteku ako je uspješno otvorena.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    char f_ime[40], f_mode[4];
    printf("Ime datoteke: ");
    scanf("%s", f_ime);
    printf("Mod rada: ");
    scanf("%s", f_mode);
    if ((fp = fopen(f_ime, f_mode)) != NULL)
    {
        printf("Datoteka uspjesno otvorena\n");
        fclose(fp);
        printf("Datoteka %s zatvorena\n", f_ime);
    }
    else printf("Greska u otvaranju\n");
    return 0;
}
```

Rezultat neuspješnog otvaranja datoteke prikazan je lijevo, a uspješnog desno.

```
Ime datoteke: demo.txt
Mod rada: r
Greska u otvaranju
```

```
Ime datoteke: demo.txt
Mod rada: w
Datoteka uspjesno otvorena
Datoteka demo.txt zatvorena
```

### 3.3. Rad sa tekstualnim datotekama

Tekstualne datoteke formiraju se formatiranim upisivanjem podataka. Tekstualna datoteka ima strukturu koja karakteriše tekst – sadrži linije čovjeku "čitljivog" teksta, koje završavaju EOL terminatorom.

Standardna biblioteka raspolaže sljedećim funkcijama koje omogućavaju baferovani upis u tekstualnu datoteku:

- upis jednog znaka:

```
int fputc(int znak, FILE *fp);
```

- neformatirani upis stringa:

```
int fputs(const char *string, FILE *fp);
```

- formatirani upis stringa:

```
int fprintf(FILE *fp, const char *format, ...);
```

Baferovano čitanje iz tekstualne datoteke realizuje se sljedećim funkcijama iz standardne biblioteke:

- čitanje jednog znaka:

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

- neformatirano čitanje stringa:

```
char *fgets(char *string, int n, FILE *fp);
```

- formatirano čitanje:

```
int fscanf(FILE *fp, const char *format, ...);
```

### 3.3.1. Upisivanje u tekstualnu datoteku

#### Upisivanje jednog znaka

Upisivanje znaka u datoteku vrši se pozivom funkcije `fputc`:

---

```
int fputc(int znak, FILE *fp); ;
```

---

Parametri funkcije `fputc` su:

- **znak** – znak koji se upisuje (pretvoren u `unsigned char`);
- **fp** – pokazivač na otvorenu datoteku.

Rezultat izvršavanja funkcije `fputc`:

- uspješan upis: ASCII kôd upisanog znaka;
  - neuspješan upis: **EOF**.
- ```
greska = fputc('A', fp) == EOF;
```

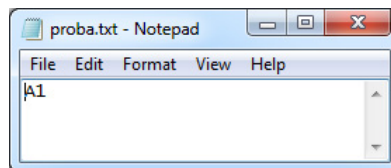
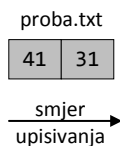
---

#### Primjer 3.3:

U tekstualnu datoteku "proba.txt" treba upisati slovo 'A' i cifru '1'.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt","w")) // otvaranje za upis
    {
        fputc('A',fp); // upis slova 'A'
        fputc('1',fp); // upis cifre '1'
        fclose(fp); // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja programa je tekstualna datoteka pod nazivom proba.txt. Lijevo je prikazan binarni sadržaj kreirane datoteke, a desno interpretacija njenog sadržaja u programu za obradu teksta.



### Neformatirano upisivanje stringa

Neformatirano upisivanje stringa u datoteku vrši se pozivom funkcije `fputs`, čiji je prototip:

---

```
int fputs(const char *string, FILE *fp);
```

---

**Parametri funkcije** `fputs` su:

- **string** – string koji se upisuje u datoteku;
- **fp** – pokazivač na otvorenu datoteku.

**Rezultat izvršavanja** funkcije `fputs`:

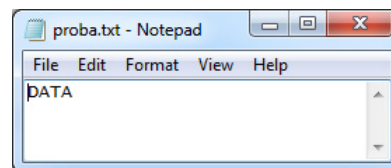
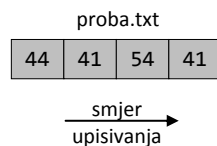
- uspješan upis: nenegativan broj;
  - neuspješan upis: **EOF**.
- ```
greska = fputs(string,fp) == EOF;
```
- 

#### Primjer 3.4:

U tekstualnu datoteku "proba.txt" treba upisati string "DATA".

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt","w")) // otvaranje za upis
    {
        fputs("DATA",fp); // upis stringa "DATA"
        fclose(fp);       // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja programa je tekstualna datoteka pod nazivom `proba.txt`. Lijevo je prikazan binarni sadržaj kreirane datoteke, a desno interpretacija njenog sadržaja u programu za obradu teksta.



Ako je u istom direktorijumu već postojala istoimena datoteka, njen sadržaj je u potpunosti izgubljen, bez obzira na to koliko je ranije bilo znakova u toj datoteci i koliko je novih znakova upisano.

---



### Formatirano upisivanje stringa

Formatirani upis stringa u datoteku vrši se pozivom funkcije `fprintf`, čiji je prototip:

---

```
int fprintf(FILE *fp, const char *format, ...);
```

---

**Obavezni argumenti funkcije `fprintf` su:**

- **fp** – pokazivač na otvorenu datoteku;
- **format** – konverzioni string (kao kod funkcije `printf`).

**Neobavezni argumenti funkcije `fprintf`:**

- lista izraza čije se izračunate vrijednosti upisuju u datoteku u skladu sa zadatim formatom.

**Rezultat izvršavanja funkcije `fprintf`:**

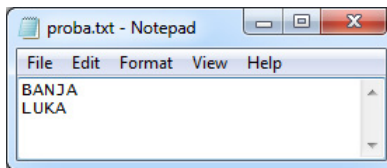
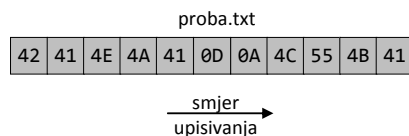
- uspješan upis: broj upisanih znakova u datoteku;
  - neuspješan upis: negativna cjelobrojna vrijednost
- ```
greska = fprintf(fp, "%d", a+2) < 0;
```
- 

#### Primjer 3.5:

U tekstualnu datoteku "proba.txt" treba upisati string "BANJA LUKA", tako da svaka riječ bude upisana u zasebnom redu.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt","w")) // otvaranje za upis
    {
        fprintf(fp,"BANJA\nLUKA"); // upis stringa
        fclose(fp); // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja programa je tekstualna datoteka pod nazivom `proba.txt`. Lijevo je prikazan binarni sadržaj kreirane datoteke, a desno interpretacija njenog sadržaja u programu za obradu teksta. Isti rezultat dobili bismo i da je upis izvršen pozivom funkcije `fputs("BANJA\nLUKA", fp)`, umjesto pozivom `fprintf(fp, "BANJA\nLUKA")`. Treba uočiti da je terminator reda (`\n`) u datoteku upisan kao kombinacija dva bajta `0D` i `0A` (vidjeti reprezentaciju EOL u odjeljku 3.2.1).



### 3.3.2. Čitanje iz tekstualne datoteke

#### Čitanje jednog znaka

Čitanje jednog znaka iz datoteke vrši se pozivom funkcije `fgetc`, čiji je prototip:

---

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

---

**Parametar funkcije** `fgetc`:

- **fp** – pokazivač na otvorenu datoteku iz koje se čita.

**Rezultat izvršavanja** funkcije `fgetc`:

- uspješno čitanje: ASCII kôd pročitano znaka;
- neuspješno čitanje (greška ili kraj datoteke): **EOF**.  
 kraj = (znak = fgetc(fp)) == EOF;

---

#### Primjer 3.6:

Iz tekstualne datoteke "proba.txt", koja je formirana u primjeru 3.5, treba pročitati sadržaj znak po znak i ispisati ga na standardnom izlazu.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt","r")) // otvaranje za citanje
    {
        char c;
        while ((c = fgetc(fp)) != EOF) // citanje znak po znak
            printf("%c", c);          // ispis na standardni izlaz
        fclose(fp);                  // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja programa prikazan je na slici lijevo. Treba uočiti da su bajtovi 0D i 0A (vidjeti sliku u primjeru 3.5) pročitani i interpretirani kao znak '\n'. Da je umjesto `printf("%c", c)` bilo `printf("%02X ", c)`, tada bismo dobili izlaz koji je na slici prikazan desno.

|               |
|---------------|
| BANJA<br>LUKA |
|---------------|

|                               |
|-------------------------------|
| 42 41 4E 4A 41 0A 4C 55 4B 41 |
|-------------------------------|

---

### Neformatirano čitanje stringa

Neformatirano čitanje stringa iz datoteke vrši se pozivom funkcije `fgets`, čiji je prototip:

---

```
char *fgets(char *string, int n, FILE *fp);
```

---

Parametri funkcije `fgets` su:

- **string** – adresa u memoriji gdje se upisuje pročitani string iz datoteke;
- **n** – maksimalan broj znakova u stringu (uključujući terminator);
- **fp** – pokazivač na otvorenu datoteku iz koje se čita.

Rezultat izvršavanja funkcije `fgets`:

- uspješno čitanje: adresa učitano stringa;
- neuspješno čitanje (kraj datoteke ili greška): **NULL**.

```
greska = fgets(string, 100, fp) == NULL;
```

Čitanje se završava ako je učitano maksimalnih  $n-1$  znakova iz datoteke ili ako je dostignut kraj reda ili kraj datoteke. Treba imati u vidu da se terminator reda, bez obzira na reprezentaciju u datoteci, interpretira kao znak `'\n'` (`'\n'` je sastavni dio učitano stringa).

---

#### Primjer 3.7:

Iz tekstualne datoteke `"proba.txt"`, koja je formirana u primjeru 3.5, treba pročitati sadržaj red po red i ispisati ga na standardnom izlazu.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt","r")) // otvaranje za citanje
    {
        char s[100];
        while (fgets(s,100,fp)) // citanje red po red
            printf("%s", s);    // ispis na standardni izlaz
        fclose(fp);           // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja programa prikazan je na slici. Treba uočiti da bismo isti rezultat dobili i ako bismo umjesto broja 100 (i kapacitet niza i argument funkcije `fgets`), stavili bilo koji drugi prirodan broj veći od 1. Smanjivanjem veličine niza, povećava se broj čitanja stringova iz bafera i usporava rad.

|               |
|---------------|
| BANJA<br>LUKA |
|---------------|

### Formatirano čitanje

Formatirano čitanje iz datoteke vrši se pozivom funkcije `fscanf`, čiji je prototip:

---

```
int fscanf(FILE *fp, const char *format, ...);
```

---

**Obavezni argumenti funkcije** `fscanf` su:

- **fp** – pokazivač na otvorenu datoteku;
- **format** – konverzioni string (kao kod funkcije `scanf`).

**Neobavezni argumenti funkcije** `fscanf`:

- lista adresa na koje se smještaju učitani podaci.

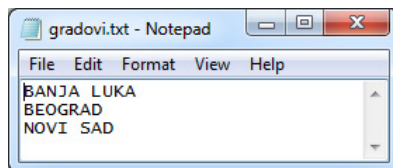
**Rezultat izvršavanja funkcije** `fscanf`:

- uspješno čitanje: broj učitanih podataka;
- neuspješno čitanje (kraj datoteke ili greška): **EOF**.  
`greska = fscanf(fp, "%d", &prom) == EOF;`

---

#### Primjer 3.8:

Na standardnom izlazu ispisati sve riječi iz datoteke "gradovi.txt".



```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("gradovi.txt","r")) // otvaranje za citanje
    {
        char s[100];
        while ( fscanf(fp,"%s",s)>0 ) // cita po jednu rijec
            printf("%s\n", s);      // ispis na standardni izlaz
        fclose(fp);                // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

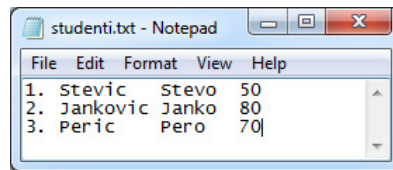
Rezultat izvršavanja programa prikazan je na slici.

```
BANJA
LUKA
BEOGRAD
NOVI
SAD
```

Funkcija `fscanf` omogućava veoma jednostavno učitavanje većeg broja podataka različitih tipova i formata iz tekstualne datoteke, slično učitavanju podataka sa standardnog ulaza pomoću funkcije `scanf`.

**Primjer 3.9:**

U datoteci "studenti.txt" upisani su podaci o studentima i njihovim rezultatima na nekom ispitu. Zna se da svaki red sadrži podatke o jednom studentu u sljedećem formatu: redni broj, prezime, ime, broj bodova. Na standardnom izlazu treba prvo ispisati sve rezultate iz datoteke, a zatim podatke o studentu sa najvećim brojem bodova.



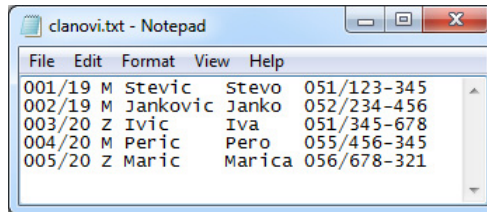
```
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *fp;
    if (fp=fopen("studenti.txt","r")) // otvaranje za citanje
    {
        int rb, sb, maxb=0;
        char sp[30], si[30];
        char maxp[30], maxi[30];
        while (fscanf(fp,"%d. %s %s %d", &rb,sp,si,&sb)==4)
        {
            printf("%d. %s %s %d\n", rb, sp, si, sb);
            if (sb>maxb)
            {
                maxb=sb;
                strcpy(maxp,sp);
                strcpy(maxi,si);
            }
        }
        printf("Podaci o najboljem studentu:\n");
        printf("%s %s: %d bodova\n", maxp, maxi, maxb);
        fclose(fp); // zatvaranje datoteke
    }
    else
        printf("Greska kod otvaranja datoteke.\n");
    return 0;
}
```

Rezultat izvršavanja programa prikazan je na slici.

```
1. Stevic Stevo 50
2. Jankovic Janko 80
3. Peric Pero 70
Podaci o najboljem studentu:
Jankovic Janko: 80 bodova
```

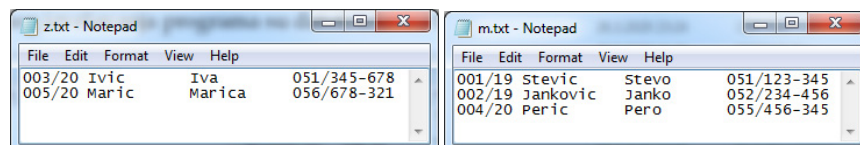
**Primjer 3.10:**

U datoteci "clanovi.txt" upisani su podaci o članovima sportskog društva. Zna se da svaki red sadrži podatke o jednom članu u sljedećem formatu: broj članske karte (string), pol (znak), prezime (string), ime (string), broj telefona (string). Podatke o ženskim članovima treba prepisati u datoteku "z.txt", a podatke o muškim članovima u datoteku "m.txt".



```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("clanovi.txt","r")) // otvaranje "clanovi.txt"
    {
        FILE *fpm = fopen("m.txt","w"); // otvaranje "m.txt"
        FILE *fpz = fopen("z.txt","w"); // otvaranje "z.txt"
        if (fpm && fpz)
        {
            char sb[7], pol, sp[10], si[10], st[15];
            while (fscanf(fp,"%s %c %s %s %s", sb,&pol,sp,si,st)==5)
            {
                if (pol=='M' || pol=='m')
                    fprintf(fpm, "%-6s %-10s %-10s %s\n", sb,sp,si,st);
                if (pol=='Z' || pol=='z')
                    fprintf(fpz, "%-6s %-10s %-10s %s\n", sb,sp,si,st);
            }
        }
        if (fpm)
            fclose(fpm); // zatvaranje "m.txt" ako je otvorena
        if (fpz)
            fclose(fpz); // zatvaranje "z.txt" ako je otvorena
        fclose(fp); // zatvaranje "clanovi.txt"
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja programa su datoteke z.txt (lijevo) i m.txt (desno), čiji je sadržaj prikazan u programu za obradu teksta.



## 3.4. Rad sa binarnim datotekama

Svaka datoteka može da se posmatra kao binarna datoteka. Da bi neka datoteka bila tretirana kao binarna, prilikom otvaranja treba specificovati režim koji uključuje znak 'b' (vidjeti odjeljak 3.2.4).

Osnovna manipulacija binarnim datotekama uključuje operacije čitanja i pisanja pojedinačnih bajtova, kao i operacije blokovskog prenosa podataka iz datoteke u memoriju i obrnuto.

### 3.4.1. Čitanje i pisanje pojedinačnih bajtova

Za čitanje i pisanje pojedinačnih jednobajtnih podataka koriste se funkcije `fgetc` i `fputc`. Ove funkcije koriste se i za rad sa tekstualnim datotekama i detaljno su opisane i ilustrovane u prethodnom odjeljku.

---

```
int fgetc(FILE *fp);  
int fputc(int znak, FILE *fp); ;
```

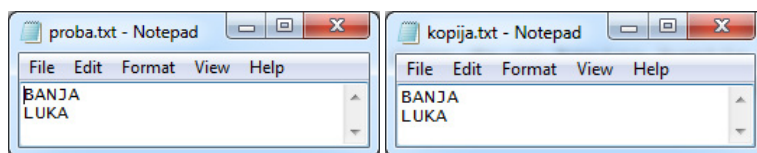
---

#### Primjer 3.11:

Program koji kopira datoteku "proba.txt" (kreiranu u primjeru 3.5).

```
#include <stdio.h>  
int main()  
{  
    FILE *fpo, *fpk;  
    if (fpo=fopen("proba.txt","rb"))    // otvaranje originala  
    {  
        if (fpk=fopen("kopija.txt","wb"))    // otvaranje kopije  
        {  
            int c;  
            while ((c = fgetc(fpo)) != EOF) // cita bajt po bajt  
                fputc(c,fpk);  
            fclose(fpk);                // zatvaranje kopije  
            printf("Kopiranje je uspjesno zavrшено.\n");  
        }  
        else printf("Ne može da se kreira kopija.\n");  
        fclose(fpo);    // zatvaranje originala  
    }  
    return 0;  
}
```

Na slici su prikazani sadržaj originala (lijevo) i sadržaj kopije (desno) dobijene uspješnim izvršavanjem prethodnog programa. Treba uočiti da je promjenljiva `c`, u koju učitavamo bajt iz datoteke, tipa `int` da ne bi došlo do greške prilikom poređenja sa konstantom `EOF`.



Treba imati u vidu da funkcije `fgetc` i `fputc` "vode računa" o reprezentaciji terminatora reda (EOL) u tekstualnim datotekama. Naime, ako se u tekstualnu datoteku upisuje znak `'\n'`, u datoteku će stvarno biti upisan njegov ekvivalent koji zavisi od konkretnog operativnog sistema (vidjeti reprezentaciju EOL u odjeljku 3.2.1), dok će prilikom upisa znaka `'\n'` u binarnu datoteku biti upisan samo jedan bajt (`0x0a`) nezavisno od operativnog sistema. Isto tako, prilikom čitanja iz tekstualne datoteke, terminator reda biće pročitan i protumačen kao `'\n'`, bez obzira na operativni sistem, dok će prilikom čitanja iz binarne datoteke biti pročitan jedan ili dva bajta, zavisno od operativnog sistema.

---

**Primjer 3.12:**

Datoteku "proba.txt", formiranu u primjeru 3.5, treba otvoriti kao binarnu, a zatim pročitati njen sadržaj bajt po bajt i ispisati ga na standardnom izlazu.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt", "rb")) // citanje iz binarne dat.
    {
        int c;
        while ((c = fgetc(fp)) != EOF) // citanje bajt po bajt
            printf("%02X ", c);
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja programa prikazan je na sljedećoj slici:

|                                  |
|----------------------------------|
| 42 41 4E 4A 41 0D 0A 4C 55 4B 41 |
|----------------------------------|

Ako bismo datoteku otvorili kao tekstualnu:

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("proba.txt", "r")) // citanje iz tekstualne dat.
    {
        int c;
        while ((c = fgetc(fp)) != EOF) // citanje znak po znak
            printf("%02X ", c);
        fclose(fp);
    }
    return 0;
}
```

dobili bismo sljedeći rezultat:

|                               |
|-------------------------------|
| 42 41 4E 4A 41 0A 4C 55 4B 41 |
|-------------------------------|

---



### 3.4.2. Blokovo pisanje i čitanje

Pored funkcija za rad sa pojedinačnim bajtovima, standardna biblioteka raspolaže i funkcijama za blokovski prenos podataka iz datoteke u memoriju i obrnuto.

#### Blokovo pisanje u datoteku

Blokovski upis u binarnu datoteku vrši se pozivom funkcije `fwrite`, čiji je prototip:

---

```
size_t fwrite(const void *mem, size_t v, size_t n, FILE *fp);
```

---

Argumenti funkcije `fwrite` su:

- **mem** – adresa početka memorijskog bloka koji se kopira u datoteku;
- **v** – veličina svakog pojedinačnog podatka koji se upisuje u datoteku;
- **n** – broj podataka koji treba da se upiše u datoteku;
- **fp** – pokazivač na otvorenu datoteku u koju se upisuju podaci.

Rezultat izvršavanja funkcije `fwrite` je broj uspješno upisanih podataka.

---

#### Primjer 3.13:

U binarnu datoteku "bin.dat" treba upisati broj 1 predstavljen kao četvorobajtni cjelobrojni podatak te kao podatak u pokretnoj tački u običnoj preciznosti.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("bin.dat", "wb")) // pisanje u binarnu datoteku
    {
        int i=1;
        float f=1;
        fwrite(&i, sizeof(i), 1, fp);
        fwrite(&f, sizeof(f), 1, fp);
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja je datoteka "bin.dat", čiji je sadržaj heksadecimalno, bajt po bajt, prikazan na slici. Ovaj sadržaj može lako da se ispiše pomoću programa iz primjera 3.12 (za naziv datoteke treba zadati "bin.dat"). Prva četiri bajta u datoteci (01 00 00 00) predstavljaju rezultat upisa promjenljive `i`, a druga četiri bajta (00 00 80 3F) predstavljaju rezultat upisa promjenljive `f`. Treba uočiti da su podaci u memoriji bili reprezentovani u skladu sa LE konvencijom (najlakši bajt je smješten na najnižoj adresi). Zbog toga je četvorobajtna cjelobrojna vrijednost 00 00 00 01 upisana kao 01 00 00 00, a podatak u pokretnoj tački 3F 80 00 00 upisan kao 00 00 80 3F.

|                         |
|-------------------------|
| 01 00 00 00 00 00 80 3F |
|-------------------------|

---

**Primjer 3.14:**

U binarnu datoteku "razlomci.dat" treba upisati niz razlomaka (razlomak je reprezentovan odgovarajućom strukturom).

```
#include <stdio.h>
typedef struct razlomak { short int b,n; } RAZLOMAK;
int main()
{
    RAZLOMAK niz[2]={1,2},{-1,3};
    FILE *fp;
    if (fp=fopen("razlomci.dat","wb"))
    {
        fwrite(niz,sizeof(RAZLOMAK),2,fp);
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja je datoteka "razlomci.dat", čiji je sadržaj heksadecimalno, bajt po bajt, prikazan na slici. Ovaj sadržaj može da se ispiše pomoću programa iz primjera 3.12 (za naziv datoteke treba zadati "razlomci.dat"). Prva četiri bajta (01 00 02 00) predstavljaju rezultat upisa prvog razlomka (niz[0]), čija je vrijednost 1/2. Prva dva bajta (01 00) predstavljaju brojilac, a druga dva bajta (02 00) predstavljaju imenilac. Na isti način, druga četiri bajta (FF FF 03 00) predstavljaju rezultat upisa drugog razlomka (-1/3).

|                         |
|-------------------------|
| 01 00 02 00 FF FF 03 00 |
|-------------------------|

**Primjer 3.15:**

U binarnu datoteku "tacka.dat" treba upisati podatke o tački A(1.0,2.0) koja je reprezentovana odgovarajućom strukturom.

```
#include <stdio.h>
typedef struct tacka { char t; float x,y; } TACKA;
int main()
{
    TACKA a={'A',1.0,2.0};
    FILE *fp;
    if (fp=fopen("tacka.dat","wb"))
    {
        fwrite(&a,sizeof(TACKA),1,fp);
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja je datoteka "tacka.dat", čiji je sadržaj heksadecimalno, bajt po bajt, prikazan na slici. U datoteku je ukupno upisano 12 bajtova, jer je to veličina podatka tipa TACKA. Prvi bajt (41) predstavlja rezultat upisa oznake tačke (element a.t). Nakon toga, upisana su tri bajta (00 00 00) koji predstavljaju *padding*, pa četiri bajta (00 00 80 3F) koji reprezentuju *x* koordinatu tačke (a.x) pa četiri bajta (00 00 00 40) koji reprezentuju *y* koordinatu tačke (a.y).

|                                     |
|-------------------------------------|
| 41 00 00 00 00 00 80 3F 00 00 00 40 |
|-------------------------------------|

### Blokovsko čitanje iz datoteke

Blokovsko čitanje iz binarne datoteke vrši se pozivom funkcije `fread`, čiji je prototip:

---

```
size_t fread(void *mem, size_t v, size_t n, FILE *fp);
```

---

Argumenti funkcije `fread` su:

- **mem** – memorijska adresa gdje se smještaju pročitani podaci;
- **v** – veličina svakog pojedinačnog podatka koji se čita iz datoteke;
- **n** – broj podataka koji treba da se pročita iz datoteke;
- **fp** – pokazivač na otvorenu datoteku iz koje se čitaju podaci.

Rezultat izvršavanja funkcije `fread` je broj uspješno učitanih podataka.

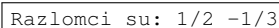
---

#### Primjer 3.16:

Iz binarne datoteke "razlomci.dat" (kreirane u primjeru 3.14) treba pročitati sve razlomke i ispisati ih na standardnom izlazu.

```
#include <stdio.h>
typedef struct razlomak { short int b,n; } RAZLOMAK;
int main()
{
    FILE *fp;
    if (fp=fopen("razlomci.dat","rb"))
    {
        RAZLOMAK r;
        printf("Razlomci su:");
        while ( fread(&r,sizeof(RAZLOMAK),1,fp) )
            printf(" %d/%d", r.b,r.n);
        fclose(fp);
    }
    else
        printf("Greska kod otvaranja ulazne datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja prikazan je na slici.



```
Razlomci su: 1/2 -1/3
```

Treba uočiti da su podaci iz datoteke čitani pojedinačno, razlomak po razlomak. Svakim uspješnim izvršavanjem funkcije `fread` učitava se po jedan blok podataka veličine četiri bajta, što odgovara veličini jednog podatka tipa `RAZLOMAK`, a učitani blok se, bajt po bajt, upisuje u memoriju na adresu promjenljive `r`. Učitavanje se ponavlja sve dok funkcija `fread` kao rezultat vraća vrijednost 1 (uspješno pročitao jedan razlomak).

---

Funkcije za blokovski prenos podataka omogućavaju veoma jednostavno kopiranje podataka iz memorije u datoteku i obrnuto – jednim pozivom moguće je upisati ili učitati blok podataka proizvoljne veličine. Budući da se kopiranje izvodi bajt po bajt, u slučaju upisivanja strukture u datoteku, biće iskopirani i bajtovi *padding*-a (kao što je ilustrovano u primjeru 3.15). To može značajno da uveća veličinu datoteke pa se ponekad, u cilju smanjenja veličine datoteke, upis u datoteku ne vrši upisivanjem strukture kao cjeline, već se svaki element strukture upisuje zasebno (primjer 3.17). Pri učitavanju tako upisanih (pakovanih) podataka, može doći do pogrešne interpretacije ako se učitavanje ne vrši na odgovarajući način.

#### Primjer 3.17:

Pakovani upis tačaka A(1, 2) i B(3, 4) u datoteku "tacke.dat".

```
#include <stdio.h>
typedef struct tacka { char t; float x,y; } TACKA;
int main()
{
    FILE *fp;
    if (fp=fopen("tacke.dat","wb"))
    {
        TACKA tacke[]={{'A',1.0,2.0},{'B',3.0,4.0}};
        for (int i=0; i<2; i++)
        {
            fwrite(&tacke[i].t,sizeof(tacke[i].t),1,fp);
            fwrite(&tacke[i].x,sizeof(tacke[i].x),1,fp);
            fwrite(&tacke[i].y,sizeof(tacke[i].y),1,fp);
        }
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja je datoteka "tacke.dat", čiji je sadržaj prikazan na slici. Prvih devet bajtova predstavlja prvu tačku (prvi bajt – oznaka, četiri bajta – x koordinata, četiri bajta – y koordinata). Na isti način je upisana i druga tačka.

|                                                       |
|-------------------------------------------------------|
| 41 00 00 80 3F 00 00 00 40 42 00 00 40 40 00 00 80 40 |
|-------------------------------------------------------|

Pretpostavimo da se podaci iz datoteke čitaju struktura po struktura:

```
#include <stdio.h>
typedef struct tacka { char t; float x,y; } TACKA;
int main()
{
    FILE *fp;
    if (fp=fopen("tacke.dat","rb"))
    {
        TACKA t;
        while ( fread(&t,sizeof(TACKA),1,fp) )
            printf("%c(%4.2f,%4.2f) ", t.t,t.x,t.y);
        fclose(fp);
    }
    return 0;
}
```

Rezultat izvršavanja dat je na slici. Treba uočiti da je iz datoteke učitana samo jedna tačka (nema dovoljno bajtova za dvije tačke) te da je samo naziv učitane tačke tačan (ostali učitani bajtovi nisu smješteni na odgovarajuće adrese i nema *padding-a*).

A(0.00, 0.00)

Imajući u vidu primjer 3.17, za upis i čitanje struktura treba primjenjivati konzistentan pristup – ili upisivanje i čitanje kompletne strukture ili upisivanje i čitanje pojedinačnih elemenata strukture.

## 3.5. Pozicioniranje u datoteci

U jeziku C, sve datoteke tretiraju se kao sekvencijalne, ali postoji i mogućnost direktnog pristupa pozicioniranjem na željeni bajt u datoteci.

### Indikator trenutne pozicije

Jedan od atributa otvorene datoteke jeste indikator trenutne pozicije, koji pokazuje lokaciju sa koje može da bude pročitana ili na koju može da bude upisan sljedeći bajt. To je cjelobrojna vrijednost (broj bajtova) koja predstavlja pomjeraj trenutne pozicije u odnosu na početak datoteke. Očitavanje trenutne pozicije u datoteci vrši se pozivom funkcije `ftell`, čiji je prototip:

---

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

---

Prilikom otvaranja datoteke, bez obzira na režim rada, indikator trenutne pozicije pokazuje na početak datoteke i ima vrijednost 0. Indikator trenutne pozicije automatski se inkrementuje sa svakim čitanjem ili pisanjem (podrazumijevani pristup datoteci je sekvencijalan).

---

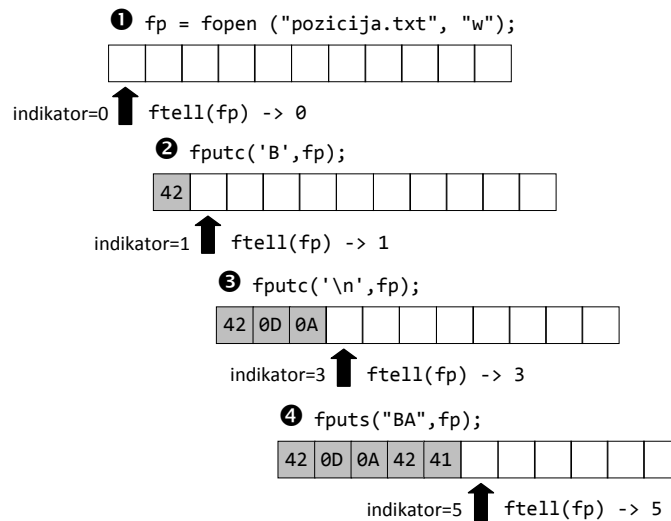
### Primjer 3.18:

Program koji ilustruje očitavanje trenutne pozicije u datoteci.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("pozicija.txt", "w"))
    {
        printf("Nakon otvaranja: %d\n", ftell(fp));
        fputc('B', fp);
        printf("Nakon upisa B: %d\n", ftell(fp));
        fputc('\n', fp);
        printf("Nakon upisa EOL: %d\n", ftell(fp));
        fputs("BA", fp);
        printf("Nakon upisa BA: %d\n", ftell(fp));
        fclose(fp);
    }
    return 0;
}
```

Rezultat uspješnog izvršavanja programa prikazan je na sljedećoj slici. Nakon otvaranja, indikator trenutne pozicije ima vrijednost 0. To je pozicija na koju je upisan znak 'B', nakon čega se indikator inkrementuje i ima vrijednost 1. Na toj poziciji započeo je upis terminatora reda (OD OA), nakon čega indikator ima vrijednost 3. Na kraju je upisan string "BA", nakon čega indikator ima vrijednost 5.

```
Nakon otvaranja: 0
Nakon upisa B: 1
Nakon upisa EOL: 3
Nakon upisa BA: 5
```



Izuzetak je otvaranje datoteke u režimu za dodavanje ("a"). I tada je na početku vrijednost indikatora pozicije 0 (da bi se omogućilo čitanje), ali neće doći do prepisivanja preko postojećeg sadržaja, nego će podaci biti upisani iza posljednjeg bajta. Nakon toga će indikator pozicije pokazivati na kraj datoteke.

#### Primjer 3.19:

Program koji ilustruje očitavanje trenutne pozicije u režimu za dodavanje.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("pozicija.txt", "a"))
    {
        printf("Nakon otvaranja: %d\n", ftell(fp));
        fputc('1', fp); printf("Nakon upisa 1: %d\n", ftell(fp));
        fclose(fp);
    }
    return 0;
}
```

```
Nakon otvaranja: 0
Nakon upisa 1: 6
```

### Pozicioniranje na željenu poziciju

Direktno pozicioniranje u datoteci na željenu poziciju postiže se pozivom funkcije `fseek`, čiji je prototip:

---

```
int fseek(FILE *fp, long pomjeraaj, int reper);
```

---

Argumenti funkcije `fseek` su:

- **fp** – pokazivač na otvorenu datoteku;
- **pomjeraaj** – broj bajtova pomjeraja u odnosu na reper;
- **reper** – orijentir u odnosu na koji se određuje željena pozicija.

U zaglavlju `<stdio.h>` definisani su sljedeći reperi:

`SEEK_SET (=0)` – početak datoteke,  
`SEEK_CUR (=1)` – trenutna pozicija,  
`SEEK_END (=2)` – kraj datoteke.

Rezultat izvršavanja funkcije `fseek`:

- uspješno pozicioniranje: 0;
- neuspješno pozicioniranje: cjelobrojna vrijednost različita od nule.

---

#### Primjer 3.20:

Program koji određuje veličinu datoteke očitavanjem pozicije kraja datoteke.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("pozicija.txt", "r"))
    {
        if (fseek(fp, 0, SEEK_END))
            printf("Greska prilikom pozicioniranja.\n");
        else
            printf("Velicina datoteke je %d[B].\n", ftell(fp));
        fclose(fp);
    }
    else
        printf("Greska prilikom otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja prikazan je na slici (u pitanju je datoteka "pozicija.txt" iz primjera 3.19). Treba uočiti paradoks da funkcija `ftell` vraća nulu (laž) u slučaju uspješnog pozicioniranja, a nenultu vrijednost (istina) u slučaju neuspješnog pozicioniranja.

Velicina datoteke je 6[B].

**Primjer 3.21:**

Ilustracija pozicioniranja u datoteci i čitanja sadržaja unazad (od kraja prema početku).

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if (fp=fopen("pozicija.txt", "rb"))
    {
        if (fseek(fp, -1, SEEK_SET))
            printf("Ne moze se pozicionirati ispred pocetka.\n");

        printf("Trenutna pozicija: %d\n", ftell(fp));

        fseek(fp, 0, SEEK_END);
        printf("Pozicija kraja datoteke: %d\n", ftell(fp));

        fseek(fp, -1, SEEK_END);
        printf("Pozicija posljednjeg znaka: %d\n", ftell(fp));

        do
        {
            printf("Pozicija %d: ", ftell(fp));
            printf("%02X\n", fgetc(fp));
        }
        while (!fseek(fp, -2, SEEK_CUR));

        fclose(fp);
    }
    else
        printf("Greska prilikom otvaranja datoteke.\n");
    return 0;
}
```

Rezultat uspješnog izvršavanja prikazan je na slici (u pitanju je datoteka "pozicija.txt" iz primjera 3.19).

```
Ne moze se pozicionirati ispred pocetka.
Trenutna pozicija: 0
Pozicija kraja datoteke: 6
Pozicija posljednjeg znaka: 5
Pozicija 5: 31
Pozicija 4: 41
Pozicija 3: 42
Pozicija 2: 0A
Pozicija 1: 0D
Pozicija 0: 42
```

Pomjeraj se može zadati kao pozitivna ili kao negativna vrijednost, pri čemu pozitivna vrijednost omogućava napredovanje kroz datoteku (kretanje prema kraju datoteke), a negativna omogućava nazadovanje (kretanje prema početku datoteke). Treba uočiti da nije moguće pozicioniranje ispred početka datoteke. Pozicioniranje iza kraja datoteke je moguće i neće rezultovati greškom.



## 3.6. Signalizacija grešaka

Svakoj otvorenoj datoteci pridružena su dva logička indikatora: **indikator kraja datoteke** i **indikator greške**. Većina funkcija za manipulaciju datotekom kao bočni efekat postavlja ove indikatore.

Standardna biblioteka raspolaže sljedećim funkcijama koje omogućavaju rad sa indikatorom kraja datoteke i indikatorom greške:

- **funkcija za provjeru da li je kraj datoteke:**

---

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

---

Funkcija `feof` provjerava stanje indikatora kraja datoteke i vraća nenultu vrijednost ako je dostignut kraj datoteke, inače vraća nulu.

- **funkcija za ispitivanje indikatora greške:**

---

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

---

Funkcija `ferror` provjerava stanje indikatora greške i u slučaju greške vraća nenultu vrijednost koja predstavlja kôd greške, inače vraća nulu. Kodovi i opis grešaka dati su u `<errno.h>`.

- **funkcija za brisanje indikatora greške i indikatora kraja datoteke:**

---

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

---

---

### Primjer 3.22:

Ilustracija očitavanja greške i brisanja indikatora.

```
#include <stdio.h>
int main ()
{
    FILE *fp;
    int greska;
    if (fp=fopen("pozicija.txt", "r"))
    {
        fputc('X', fp);
        if (greska=ferror(fp))
        {
            printf("Greska: %d\n", greska);
            clearerr(fp);
            if (greska=ferror(fp))
                printf("Greska: %d\n", greska);
            else
                printf("Greska je obrisana.\n");
        }
    }
    return 0;
}
```

Rezultat izvršavanja prikazan je na slici. Treba uočiti da je datoteka otvorena u režimu za čitanje i da je pokušaj upisa znaka 'X' rezultovao greškom čiji je kôd 32. Nakon toga su indikatori obrisani, što je dodatno provjereno.

```
Greska: 32
Greska je obrisana.
```

Posebnu pažnju treba posvetiti korištenju funkcije `feof`. Ova funkcija provjerava stanje indikatora kraja datoteke, a ne da li pokazivač trenutne pozicije stvarno pokazuje na kraj datoteke. Budući da na stanje indikatora kraja datoteke utiču operacije čitanja ili pisanja, funkciju `feof` ne treba koristiti prije čitanja ili pisanja iz datoteke.

---

**Primjer 3.23:**

Pretpostavimo da sljedeći program pristupa praznoj datoteci "feof.dat".

```
#include <stdio.h>
int main ()
{
    FILE *fp;
    if (fp=fopen("feof.dat", "rb"))
    {
        printf("Provjera kraja datoteke: %d\n", feof(fp));
        fseek(fp,0,SEEK_END);
        printf("Pozicija kraja: %d\n", ftell(fp));
        fclose(fp);
    }
    else
        printf("Greska otvaranja ulazne datoteke.\n");
    return(0);
}
```

Rezultat izvršavanja prikazan je na slici. Treba uočiti da je funkcija `feof` vratila vrijednost 0 (nije dostignut kraj datoteke), a datoteka je prazna (pokazivač pozicije pokazuje na kraj datoteke).

```
Provjera kraja datoteke: 0
Pozicija kraja: 0
```

---

Neočekivano ponašanje programa može da se desi ako se kontrola toka (npr. petlje) vrši na osnovu rezultata funkcije `feof`. Pošto funkcija `feof` ne provjerava trenutnu poziciju u datoteci, nego indikator kraja datoteke, postoji mogućnost da se desi suvišno učitavanje koje će završiti neuspješno, kao što je ilustrovano u primjeru 3.24. Zbog toga treba biti pažljiv, odnosno izbjegavati kontrolu petlji pomoću funkcije `feof`.

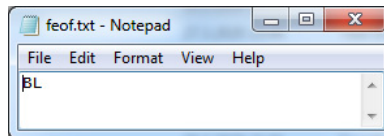
**Primjer 3.24:**

Ilustracija problematične kontrole toka zasnovane na rezultatu funkcije `feof`.

```
#include <stdio.h>
int main ()
{
    FILE *fp;
    if (fp=fopen("feof.txt", "rb"))
    {
        int rb=0;
        while (!feof(fp))
            printf("%d: %02X\n", rb++, fgetc(fp));
        fclose(fp);
    }
    return(0);
}
```

Rezultat izvršavanja programa prikazan je lijevo na slici, a sadržaj datoteke "feof.txt" prikazan je desno. Treba uočiti da je zbog pogrešne kontrole toka, koja je zasnovana na rezultatu funkcije `feof`, izvršeno jedno suvišno čitanje. Suvišno čitanje završilo je neuspješno – funkcija je vratila EOF (-1), zbog čega je na standardnom izlazu ispisano FFFFFFFF.

```
0: 42
1: 4C
2: FFFFFFFF
```



## 3.7. Standardni tokovi

Pri pokretanju programa, tri predefinisane datoteke (standardni ulazno-izlazni tokovi) otvaraju se automatski:

- **stdin** = standardni ulaz (podrazumijevano tastatura),
- **stdout** = standardni izlaz (podrazumijevano displej),
- **stderr** = standardni izlaz za greške (podrazumijevano displej).

Standardni tokovi automatski se otvaraju i mogu da se koriste kao i bilo koja druga datoteka, kao što je ilustrovano u sljedećem primjeru.

**Primjer 3.25:**

Ilustracija pristupa standardnim tokovima kao datotekama.

```
#include<stdio.h>
int main()
{
    char s[100];
    fputs("Unesite tekst: ", stdout);
    fgets(s, 100, stdin);
    fputs(s, stdout);
    return 0;
}
```

Primjer izvršavanja programa prikazan je na slici.

```
Unesite tekst: Programski jezik C
Programski jezik C
```

### Preusmjeravanje standardnih tokova

U nekim okruženjima moguće je iz komandne linije, prilikom poziva programa, izvršiti redirekciju (preusmjeravanje) standardnih tokova:

- preusmjeravanje standardnog ulaza sa tastature na ulaznu datoteku (učitavanje se ne vrši sa tastature, nego iz datoteke):

```
C:\>program <ulaz.txt
```

- preusmjeravanje standardnog izlaza sa displeja na izlaznu datoteku (ispis se ne vrši na displej, nego u datoteku):

```
C:\>program >izlaz.txt
```

#### Primjer 3.26:

Ilustracija redirekcije standardnog ulaza i standardnog izlaza.

```
#include <stdio.h>
int main()
{
    char s[100];
    fgets(s, 100, stdin);
    fputs(s, stdout);
    return 0;
}
```

Pretpostavimo da je izvršni program ("redirekcija.exe") pokrenut iz komandne linije, kao što je ilustrovano na slici.



Umjesto sa standardnog ulaza, program učitava podatke iz datoteke "ulaz.txt" i rezultat ne ispisuje na standardnom izlazu, nego u izlaznu datoteku "izlaz.txt", kao što je ilustrovano na slici.

