Programiranje 2

Predavanje 10 - datoteke, prvi dio

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

20. svibnja 2023.

Osnovno o datotekama

Datoteka je prostor (područje) za trajno spremanje podataka u vanjskoj (sporijoj) memoriji. Ta memorija je različita od standardne radne memorije računala (RAM). Tipični mediji za spremanje datoteka su: trake, kazete, disketa, optički mediji (CD, DVD), flash-memorije, disk (HDD, SSD) itd.
Trajno spremanje znači da podaci u datoteci ostaju sačuvani i nakon izvršavanja programa (gašenja računala). Zbog toga se ista datoteka može koristiti u više programa. Operacijski sustav računala koristi sustav datoteka (*file-system*) za organizaciju datoteka u računalu. Unutar tog sustava datoteka, svaka datoteka ima svoje ime koje koristimo za pristup datoteci.

Pravila za pisanje (tvorbu) imena datoteka specifična su za pojedini operacijski sustav. To posebno vrijedi za puno ime datoteke, koje sadrži i putanju (*path*) do te datoteke u sustavu datoteka.

Ime datoteke

Osnovno ime datoteke (bez putanje) do nje, standardno ima oblik `ime.ekstenzija` (Unix, Windows). Ekstenzija označava vrstu sadržaja datoteke.

- `sort.c` - izvorni kod C programa,
- `math.lib` - biblioteka prevedenih funkcija pripremljena za linker,
- `sort.exe` - izvršni (binarni) kod programa (Windows).

Točka na početku imena datoteke na Unix sustavu označava skrivenu datoteku.

Operacije s datotekama

Osnovne operacije s datotekama su (iz perspektive programa koji obrađuje datoteku):

- čitanje podataka iz datoteke - ulaz podataka u program
- pisanje podataka u datoteku - izlaz podataka iz programa

Postoje dvije podjele datoteka, prema tome što se događa u gore navedenim operacijama:

- po načinu pristupa podacima u datoteci: a) slijedne (sekvencijalne), b) direktne datoteke
- po interpretaciji sadržaja podataka u datoteci pri čitanju i pisanju: a) formatirane, b) neformatirane.

Podjela datoteka - po načinu pristupa

Postoje dva bitno različita načina pristupa podacima u datoteci pri čitanju i pisanju (dva načina realizacije ovih operacija).

- Slijedni ili sekvencijalni pristup čita i piše samo u jednom smjeru (unaprijed), podatak za podatkom, kao na traci. To je standardni način pristupa podacima u C-u.
- Direktnim pristupom čitamo i pišemo bilo gdje u datoteci, slično kao u polju. Realizira se posebnim funkcijama za pozicioniranje u datoteci.

Podjela datoteka - po interpretaciji sadržaja

Neki podatak, npr. cijeli broj, možemo zapisati u datoteku na dva bitno različita način:

- formatirano - u obliku tekstualnog zapisa podatka, kao da pišemo funkcijom `printf`,
- neformatirano - u obliku interne reprezentacije tog podatka u računalu (kopiranjem sadržaja memorije koju taj podatak zauzima).

Identična stvar vrijedi i kod čitanja. Stoga, po načinu zapisa ili po interpretaciji sadržaja, datoteke možemo podijeliti na **formatirane** i **neformatirane**.

U Fortranu je veća razlika između formatiranih i neformatiranih datoteka od C-a.

Formatirane i neformatirane datoteke

Datoteka je konačan niz byte-ova. Kod operacija pisanja i čitanja podataka, te byte-ove možemo zapisati, odnosno interpretirati na dva različita načina.

- **Formatirani** zapis - sadržaj se interpretira kao **tekstualni** zapis podataka.
- **Neformatirani** zapis - sadržaj se interpretira kao **interna reprezentacija** podataka u tom računalu i operacijskom sustavu.

Obje vrste zapisa možemo realizirati u C-u odgovarajućim funkcijama za **čitanje** i **pisanje** (može i na istoj datoteci).

Datoteke u C-u

U C-u nema izravne podjele na formatirane i neformatirane datoteke. Po ANSI standardu, postoje dvije vrste datoteka: **tekstualne** i **binarne**.

- Binarna datoteka - niz podataka tipa `char` (niz znakova).
- Tekstualna datoteka ima dodatnu strukturu, kao tekst. Reprezentira se kao niz znakova podijeljenih u linije (redove), a svaka linija sadrži nula ili više znakova, iza kojih slijedi znak `\n` za kraj linije (reda).

Binarne i tekstualne datoteke

Razlika između binarnih i tekstualnih datoteka ovisi samo o standardnoj oznaci za kraj linije u odgovarajućem operacijskom sustavu.

- Unix - znak `\n` (line feed ili newline).
- Windows - dva znaka: `\r` (carriage return) i `\n`.
- Mac OS - znak `\r`.

Funkcije koje prepoznaju kraj linije (poput `fgets` i `fputs`) korektno pretvaraju standardni kraj linije (u datoteci) u znak `\n` (u C programu) i obratno. Zato na Unixu nema nikakve razlike između binarnih i tekstualnih datoteka, a na ostalim navedenim sustavima ima.

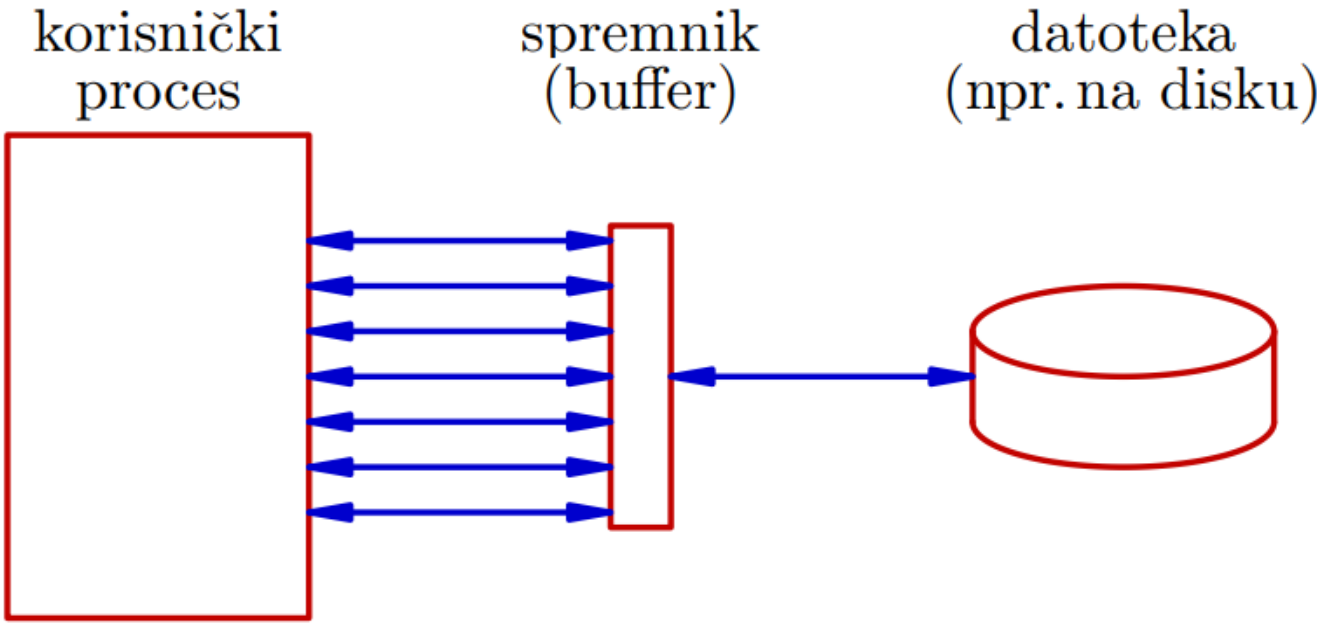
Rad s datotekama u C-u

Sve operacije s datotekama u C-u, uključujući i ulazno-izlazne, tj. stvarno čitanje i pisanje podataka, realizirane su standardnim funkcijama, deklariranim (prototipovima) u standardnoj datoteci zaglavlja `<stdio.h>`. Te funkcije su visoka razina ulazno-izlaznih operacija, jer skrivaju niz detalja vezanih uz konkretni operacijski sustav i lako se prenose s jednog sustava na drugi.

Za upotrebu tih funkcija treba nam osnovni pojam: spremnik (*buffer*).

Spremnik (*buffer*)

Stvarna komunikacija između korisničkog programa i datoteke vrši se preko posebnog prostora u memoriji računala kojeg zovemo spremnik (*buffer*). Razlog je razlika u brzini između centralnih dijelova računala (procesor, memorija) i vanjskih ulazno-izlaznih jedinica (npr. disk). Posljedica hijerarhijske građe memorije. U spremnik se privremeno pohranjuju sve informacije koje se šalju u datoteku ili primaju iz datoteke.



Spremnik (*buffer*)

Svrha spremnika je:

- smanjiti komunikaciju s vanjskom memorijom (npr. diskom) - transfer podataka ide u blokovima,
- povećati efikasnost ulazno–izlaznih funkcija.

Ovaj spremnik je dio operacijskog sustava za rad s datotekama. Većina ulazno-izlaznih uređaja ima još i svoj vlastiti spremnik (katkad i *cache*) s identičnom svrhom (tzv. *double-buffer* komunikacija).

Pristup spremniku u C-u, struktura tipa FILE

Standardna datoteka zaglavlja `<stdio.h>` sadrži deklaraciju strukture posebnog tipa koji se zove `FILE`. U strukturi tipa `FILE` opisan je spremnik i svi ostali podaci potrebni za komunikaciju s datotekom, koji ovise o operacijskom sustavu. Ovu strukturu katkad isto zovemo **spremnik** ili **file buffer** po jednom dijelu njezinog sadržaja. Tu se nalaze svi detalji implementacije datoteka koje korisnici ne moraju znati.

Struktura `FILE` sadrži: a) osnovne informacije o datoteci, b) vrstu operacije - čitanje/pisanje (tzv. `file_mod`), c) status operacija - je li došlo do greške ili smo došli do kraja datoteke (`ferror`, `feof`), d) stvarna trenutna pozicija u datoteci (nula je početak) - pozicija gdje ide sljedeće čitanje/pisanje (`ftell`), e) stvarna lokacija spremnika za komunikaciju, f) trenutna pozicija u spremniku - kada stignemo do kraja datoteke trebamo fizički učitati ili napisati novi blok podataka.

Spremnik - otvaranje i zatvaranje datoteke

Svakoj datoteci s kojom radimo u programu pridružen je odgovarajući objekt tipa `FILE`. To je spremnik za tu datoteku. Pošto je on dinamički objekt, do njega dolazimo preko pokazivača (tzv. *file pointer*). Na početku rada s datotekom, *file pointer* moramo kreirati operacijom otvaranja datoteke - funkcija `fopen`. Na kraju rada s datotekom, moramo osloboditi memoriju za spremnik operacijom zatvaranja datoteke - funkcija `fclose`.

Otvaranje datoteke - pokazivač na datoteku

Prvi korak, prije samog otvaranja datoteke je deklaracija pripadnog pokazivača na `FILE` (*file pointer*). On će, nakon otvaranja, pokazivati na spremnik za tu datoteku (`FILE *fp`). Sljedeći korak je otvaranje datoteke, tj. kreiranje pripadnog spremnika (alokacija memorije) i uspostavljanje komunikacije sa stvarnom datotekom u operacijskom sustavu.

Datoteka mora biti otvorena prije prve operacije pisanja ili čitanja.

Otvaranje datoteke - funkcija `fopen`

Otvaranje datoteke vrši se pozivom funkcije `fopen` `FILE *fopen(const char *ime, const char *tip);` gdje je:

- `ime` - pravo ime datoteke koja se otvara (string)
- `tip` - string koji kaže kako treba otvoriti tu datoteku (način rada ili `file_mod`)

Funkcija `fopen` vraća:

- Pokazivač na strukturu `FILE`, povezanu s tom datotekom ako je datoteka uspješno otvorena
- `NULL`, ako datoteka nije mogla biti otvorena (greška).

Nakon otvaranja datoteke treba uvijek provjeriti vraćeni pokazivač.

Otvaranje datoteke — funkcija `fopen`

Otvaranje datoteke tipično se radi na sljedeći način:

```
FILE *fp;
...
fp = fopen(ime, tip);
if (fp == NULL) { /* Reakcija na gresku. */
printf("Greska u otvaranju datoteke!\n");
...
}
```

`ime` je pravo ime datoteke (ime u operacijskom sustavu), npr. *podaci.dat*. Drugi string `tip` je jedan od predefiniranih stringova oznake tipa.

Tipovi (`file_mod`) tekstualne datoteke

Za otvaranje tekstualne datoteke, tj. za tekstualni način rada s datotekom koriste se sljedeći tipovi:

- `r` - otvaranje postojeće datoteke samo za čitanje,
- `w` - kreiranje nove datoteke samo za pisanje,
- `a` - otvaranje postojeće datoteke za dodavanje teksta,
- `r+` - otvaranje postojeće datoteke za čitanje i pisanje,
- `w+` - kreiranje nove datoteke za čitanje i pisanje,
- `a+` - otvaranje postojeće datoteke za čitanje i dodavanje teksta.

`r = read`, `w = write`, `a = append` (dodavanje na kraj).

Tipovi za otvaranje datoteke — osnovna pravila

Kod tipova za otvaranje datoteka vrijede sljedeća pravila:

- Čitanje (`r` ili `r+`) očekuje postojeću datoteku, ne kreira novu (greška).
- Pisanje (`w` ili `w+`) briše sadržaj postojeće datoteke (ukoliko postoji) i pisanje počinje od početka.
- Dodavanje (`a` ili `a+`) kreira datoteku ukoliko ona ne postoji i pisanje kreće od početka, ako datoteka postoji novi tekst će biti dodan na kraj te datoteke.

Tipovi (`file_mod`) - binarne datoteke

Za otvaranje binarne datoteke, tj. za binarni način rada s datotekom, u odgovarajući tekstualni tip treba dodati slovo `b`.

- `rb` - binarno čitanje iz postojeće datoteke,
- `wb` - binarno pisanje, kreiranje nove binarne datoteke,
- `ab` - binarno dodavanje,
- `rb+` ili `r+b` - binarno čitanje i pisanje iz postojeće binarne datoteke,
- `wb+` ili `w+b` - binarno čitanje i pisanje, kreiranje nove binarne datoteke,
- `ab+` ili `a+b` - binarno čitanje i dodavanje.

Unix ima samo jedan tip datoteka (binarno = tekstualno).

Zatvaranje datoteke — funkcija `fclose`

Na kraju rada s datotekom, datoteku treba obavezno zatvoriti pozivom funkcije `fclose`.
`int fclose(FILE *fp);`, gdje je: `fp` - pokazivač na strukturu `FILE`, povezanu s tom datotekom (pripadni spremnik).

Funkcija `fclose` vraća:

- nulu, ako je datoteka uspješno zatvorena,
- `EOF` u slučaju greške.

Zatvaranje datoteke je nužno, u protivnom kod pisanje može doći do gubitka podataka, ako program završi greškom. Moguće je i da se dio sadržaja datoteke ne učita iz spremnika datoteke.

Zatvaranje datoteke — funkcija `fclose`

Funkcija `fclose` radi sljedeće:

- prazni spremnik i ako treba piše u datoteku ono što dotad nije napisano iz spremnika,
- završava komunikaciju s datotekom u operacijskom sustavu,
- oslobađa memoriju za spremnik.

Korištenje: `fclose(fp);`
Nakon poziva funkcije `fclose`, pokazivač `fp` se ne mijenja, ali pokazuje na dealociranu memoriju, stoga je korektno taj pokazivač postaviti na `NULL` (kao što to radimo i nakon poziva funkcije `free`).

Otvaranje i zatvaranje datoteke - primjer

Otvaranje (za pisanje) i zatvaranje datoteke `primjer.dat` možemo napraviti ovako:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  ...
4  FILE *fp;
5
6  if ((fp = fopen("primjer.dat", "w")) == NULL) {
7              printf("Ne␣mogu␣otvoriti␣datoteku!\n");
8              exit(EXIT_FAILURE); /* exit(1); */
9  }
10 /* Rad s datotekom (pisanje u nju). */
11 ...
12 fclose(fp);
```

Standardne datoteke

Svakom C programu u trenutku izvršavanja stoje na raspolaganju tri standardne, automatski otvorene datoteke (njih otvara operacijski sustav računala):

- **standardni ulaz** - tipkovnica računala,
- **standardni izlaz** - ekran računala,
- **standardni izlaz za greške** - ekran računala.

U datoteci zaglavlja `<stdio.h>` deklarirani su konstantni pokazivači na `FILE` strukture, povezane s tim datotekama. Ti pokazivači imaju sljedeća imena:

- `stdin` - za standardni ulaz,
- `stdout` - za standardni izlaz,
- `stderr` - za standardni izlaz za greške.

Standardne datoteke - preusmjeravanje

Neki operacijski sustavi (Unix, DOS, Windows, ...) imaju mogućnost preusmjeravanja datoteka (*redirection*). Pri pozivu programa, na komandnoj liniji, možemo standardne datoteke `stdin` i `stdout` preusmjeriti na neke druge datoteke.

```
demo <demo.in >demo.out
```

Znak $<$ preusmjerava `stdin` na datoteku `demo.in`, pa se čitanje vrši iz datoteke `demo.in`. Znak $>$ preusmjerava `stdout` na datoteku `demo.out`, pa se pisanje vrši u datoteku `demo.out`.

Funkcije za čitanje i pisanje - pregled

Standardni ulaz/izlaz	Datoteke
`getchar`	`fgetc`, `getc`
`putchar`	`fputc`, `putc`
`gets`	`fgets`
`puts`	`fputs`
`printf`	`fprintf`
`scanf`	`fscanf`

Sve funkcije u desnom stupcu kao argument primaju pokazivač na `FILE`. To je zadnji argument u prva četiri reda, a prvi argument za zadnje dvije funkcije.

Čitanje znak po znak - funkcije `fgetc`, `getc`

Deklaracija (prototip): `int fgetc(FILE *fp);` i `int getc(FILE *fp);`.
Funkcije `fgetc` i `getc` vraćaju:

- sljedeći znak iz datoteke na koju pokazuje `fp`,
- `EOF` u slučaju greške ili kraja datoteke.

Vraćeni znak je tipa `unsigned char`, pretvoren u `int`. `EOF` je simbolička konstanta definirana u `<stdio.h>`. Najčešće je `EOF = -1` (ne smije biti legalni znak u datoteci) i zato je izlazni tip `int`.

Razlika između `fgetc` i `getc` je da se `getc` može implementirati i kao makro naredba dok isto nije moguće za `fgetc`. `getc` treba oprezno upotrebljavati ako se definira kao makro naredba (mogućnost višestrukog evaluiranja argumenta `fp`).

Funkcija `getchar()` za standardni ulaz implementira se kao `getc(stdin)`.

Obrada datoteke znak po znak — primjer

Obrada datoteke čitanjem znak po znak, tipično se radi na sljedeći način:

```
FILE *fp; int ch; /* Ne: char ch! */
if ((fp = fopen("podaci.txt", "r")) == NULL) {
        printf("Ne mogu otvoriti datoteku!\n");
        exit(EXIT_FAILURE); /* exit(1); */
}

/* Obrada datoteke - znak po znak). */
while ((ch = fgetc(fp)) != EOF) {
... /* Obradi znak ch. */
}
...
fclose(fp);
```

Broj znakova u datoteci

Pišemo program koji broji znakove u datoteci. Ime datoteke zadaje se kao argument komandne linije.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  /* Broj znakova u datoteci. */
4  int main(int argc, char *argv[]){
5   FILE *fp;
6   int ch, brojac = 0;
7
8   if (argc == 1) { /* Nema imena datoteke! */
9      printf("Uporaba: %s ime\n",argv[0]);
10     exit(1); }
11
12  if ((fp = fopen(argv[1], "r")) == NULL) {
13     printf("Neuspjesno otvaranje %s!\n",argv[1]);
14     exit(2); }
```

Broj znakova u datoteci

```
15  while ((ch = fgetc(fp)) != EOF) ++brojac;
16          fclose(fp);
17  printf("Broj␣znakova␣%s:␣%d\n",argv[1], brojac);
18  return 0; }
```

Zbog razlika u označavanju kraja reda na različitim operacijskim sustavima, možemo dobiti nešto drugačiji broj znakova. Funkcija `fgetc` će, kada datoteku čitamo kao tekstualnu, dva znaka `\r`, `\n`, koji označavaju kraj reda na Windowsima, pretvoriti u znak `\n`. Ukoliko datoteku promatramo i otvorimo kao binarnu, tada navedene konverzije nema pa dobijemo isti rezultat bez obzira na korišteni operacijski sustav.

Vraćanje učitanog znaka - funkcija `ungetc`

Prototip: `int ungetc(int c, FILE *fp);`
Funkcija `ungetc` vraća znak *c* (pretvoren u `unsigned char`) natrag u spremnik iz `FILE` strukture na koju pokazuje `fp` i čini taj znak dostupnim za ponovo čitanje. Taj znak će se ponovo pročitati kod sljedećeg čitanja. Izlazna vrijednost funkcije je:

- *c* ako je uspješno vraćen u spremnik,
- `EOF` u slučaju greške.

Po standardu, u svakom trenutku dozvoljeno je vratiti najviše jedan znak (različit od `EOF`) u spremnik za datoteku.

Vraćanje unatrag u spremniku za čitanje je često korisno kod obrade gramatički strukturiranog teksta (npr. riječi). Kraj neke vrste takvih objekata prepoznamo učitavanjem prvog znaka sljedećeg objekta. Tada, umjesto da posebno pamtimo suvišno učitani znak, vratimo se za poziciju unatrag u spremniku za čitanje te učitavamo sljedeći objekt.

Vraćanje učitanog znaka - funkcija ungetc

Praznina (blank) je tipičan primjer znaka koji označava kraj riječi u tekstu.

```
1 int c; /* Bolje od char c. */
2 ... /* Citamo znak po znak. */
3 c = fgetc(fp);
4 ... /* Prepoznamo kraj. */
5 ungetc(c, fp); /* Vratimo c u spremnik. */
6 ...
7 c = fgetc(fp); /* Opet procitamo c. */
```

Pisanje znak po znak - funkcije `fputc`, `putc`

Prototip: `int fputc(int c, FILE *fp);` i `int putc(int c, FILE *fp);`

Funkcije `fputc` i `putc` upisuju zadani znak *c* (pretvoren u `unsigned char`) u datoteku na koju pokazuje `fp`. Izlazna vrijednost je:

- *c* ako je uspješno upisan u datoteku,
- `EOF` u slučaju greške.

Jedina razlika između `fputc` i `putc` je što `putc` možemo koristiti i kao makro naredbu dok `fputc` ne možemo. Kao i kod `getc`, `putc` treba oprezno upotrebljavati ako se definira kao makro naredba (mogućnost višestrukog evaluiranja argumenta `fp`).

Funkcija `putchar(c)` za standardni izlaz implementira se kao `putc(c,stdout)`.

Kopiranje datoteke znak po znak

Pišemo program koji kopira sadržaj jedne datoteke u drugu, znak po znak. Imena datoteka zadaju se kao argumenti komandne linije (`odakle`, `kamo`). Poruke o greškama pišemo na `stderr` (standardni izlaz za greške) funkcijom `fprintf`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]){
5    FILE *in, *out;
6    int c; /* Ne: char c! */
7    if (argc != 3) { /* Nema imena datoteka! */
8        fprintf(stderr, "Uporaba: %s ime1 ime2\n",
9                                          argv[0]);
10       exit(1); }
```

Kopiranje datoteke znak po znak

```
11  if ((in = fopen(argv[1], "r")) == NULL) {
12      fprintf(stderr, "Ne␣mogu␣citati:␣%s!\n",
13                                      argv[1]);
14      exit(2); }
15
16  if ((out = fopen(argv[2], "w")) == NULL) {
17      fprintf(stderr, "Ne␣mogu␣pisati:␣%s!\n",
18                                      argv[2]);
19      exit(3); }
20
21  while ((c = fgetc(in)) != EOF)
22          fputc(c, out);
23
24  fclose(in);
25  fclose(out);
26  return 0; }
```

Kopiranje datoteke znak po znak - funkcija

Pišemo funkciju koja kopira sadržaj jedne datoteke u drugu znak po znak. Obje datoteke, tj. pokazivači na njih su argumenti funkcije (smatramo da su obje već otvorene).

```
1  void copy_file(FILE *in, FILE *out){
2      int c; /* Ne: char c! */
3
4      while ((c = fgetc(in)) != EOF)
5              fputc(c, out);
6
7      return; }
```

Kopiranje datoteke `byte-po-byte` je najsporiji način kopiranja. Prednost je što je jednostavno, i sigurno. Bitno brže je kopirati u većim blokovima.

Čitanje liniju po liniju — funkcija `fgets`

Funkcija za čitanje podataka iz datoteke, liniju po liniju, je:

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

- `str` - pokazivač na dio memorije (*buffer*) u koji će ulazna linija biti spremljena kao string,
- `n` - veličina memorije na koju pokazuje prvi argument = maksimalni broj znakova koji želimo spremiti u polje `str`,
- `fp` - pokazivač na datoteku iz koje se učitava.

Funkcija `fgets` će iz datoteke na koju pokazuje `fp` pročitati liniju od najviše $n-1$ znakova, (najdalje) do prvog sljedećeg znaka '`\n`' za kraj linije, uključujući i njega, ili do kraja datoteke, i na kraj učitanog stringa dodati nul-znak '`\0`'. Ako je ulazna linija dulja od $n-1$ znakova, ostatak se ne čita. Može se pročitati kasnije, npr. sljedećim pozivom funkcije `fgets`. Izlazna vrijednost je:

- pokazivač `str` ako je sve uspješno pročitano,
- `NULL` u slučaju greške ili ako se na početku čitanja odmah došlo do kraja datoteke.

Funkcija gets — za standardni ulaz

Funkcija `gets` čita string sa standardnog ulaza `stdin`.
`char *gets(char *str);`

`gets` ne prima veličinu *buffer*-a `str` kao argument. Može se dogoditi da je ulazna linija dulja od za nju rezervirane memorije u `str`. Iz tog razloga se ne preporuča korištenje ove funkcije!. `gets` je izbačena iz jezika C od standarda *C*11.

Umjesto `gets(str)` bolje je koristiti `fgets(str, n, stdin)`. Dodatna razlika između `fgets` i `gets` je u tome što `fgets` učitava i znak '`\n`' (bez zamjene) dok `gets` učita '`\n`' i zamjenjuje ga znakom '`\0`'.

Pisanje liniju po liniju - funkcija `fputs`

Funkcija za pisanje podataka u datoteku, liniju po liniju, je:

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

Funkcija `fputs`:

- ispisuje znakovni niz (string) na kojeg pokazuje `str` u datoteku na koju pokazuje `fp`,
- nul-znak na kraju stringa se ne ispisuje.

Ako želimo prijelaz u novi red, string mora sadržavati znak '`\n`' (ne piše se automatski na kraju stringa).

Izlazna vrijednost je:

- nenegativan broj - ako je ispis uspio,
- `EOF` u slučaju greške.

Funkcija puts — za standardni izlaz

Funkcija `puts` piše string na standardni izlaz `stdout`:

```
int puts(const char *str);
```

Razlika između `fputs` i `puts`:

- `fputs` ne dodaje znak '`\n`' na kraju ispisa
- `puts` dodaje znak '`\n`' na kraj (umjesto znaka '`\0`').

Razlike u ponašanju s obzirom na znak '`\n`', između:

- `fputs(str, stdout)` i `puts(str)`
- `fgets(str, n, stdin)` i `gets(str)`

odgovaraju jedna drugoj, tako da kopiranje datoteke liniju po liniju, odgovarajućim parom funkcija radi korektno.

Formatirano čitanje i pisanje za datoteke

Za formatirano čitanje iz datoteke koristimo funkciju:

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

Za formatirano pisanje u datoteku koristimo funkciju:

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

Ove funkcije rade identično kao ranije funkcije `scanf`, `printf`, s tim da je ovdje prvi argument: pokazivač `fp` - na datoteku s kojom se radi operacija (čitanje ili pisanje). Pravila za string formata i ostale argumente su ista kao prije.

- `fscanf(stdin, ...)` je ekvivalentno sa `scanf(...)`,
- `fprintf(stdout,...)` je ekvivalentno s `printf(...)`.

Funkcija `fscanf` vraća:

- nenegativan broj učitanih objekata,
- `EOF` ako je došlo do greške ili do kraja datoteke, prije prve konverzije, tj. čitanja vrijednosti prvog objekta.

Formatirano čitanje i pisanje za datoteke

Funkcija `fprintf` vraća:

- nenegativan broj napisanih znakova,
- negativan broj - u slučaju greške

Preporučljivo je ispitivati povratne vrijednosti funkcija `fscanf` i `fprintf`.

```
1  //formatirano pisanje u datoteku
2  int kolicina = 50;
3  double cijena = 7.50;
4  ...
5  fprintf(fp, "%5d, %10.2f\n", kolicina, cijena);
6
7  //formatirano citanje iz datoteke
8  int kolicina;
9  double cijena;
10 ...
11 fscanf(fp, "%d, %lf\n", &kolicina, &cijena);
```

Formatirano čitanje i pisanje za stringove

Formatirano čitanje i pisanje možemo raditi i sa stringovima, a ne samo s datotekama. Odgovarajuće funkcije su:

```
int sscanf(char *s, const char *format, ...);
int sprintf(char *s, const char *format, ...);
```

Ove funkcije rade identično kao i funkcije `fscanf`, `fprintf`, s tim da je ovdje prvi argument **pokazivač** *s* na string s kojim se radi operacija.

Kod pisanja, funkcija `sprintf` dodaje nul-znak '\0' na kraj stringa, ali ga ne broji kad vraća broj napisanih znakova. String *s* mora biti dovoljno velik za cijeli ispis (nema kontrole).

Funkcije sscanf i sprintf — primjer

Radimo formatirano čitanje teksta iz stringa i pisanje u string.

```
1  #include <stdio.h>
2  int main(void){
3      char *ulaz = "␣50,␣7.50\n"; /* Ulaz! */
4      char izlaz[80];
5      int kolicina;
6      double cijena;
7      sscanf(ulaz, "%d,%lf\n", &kolicina, &cijena);
8      printf("%5d,␣%10.2f\n", kolicina, cijena);
9      sprintf(izlaz, "␣kolicina␣=␣%5d\n"
10         "␣cijena␣=␣%10.2f\n", kolicina, cijena);
11     printf("%s\n", izlaz);
12     return 0; }
```

Kod možemo upotrebljavati za brzo pretvaranje broja u niz znakova i obratno.

Zadatak — Uvjetno kopiranje riječi ("filter")

Pišemo program koji s komandne linije učitava imena dviju datoteka: ulazne i izlazne. Broj argumenata ne treba provjeravati.

Pretpostavljamo da ulazna datoteka već postoji i:

- sastoji se iz riječi odvojenih bjelinama,
- svaka riječ može sadržavati bilo koje druge znakove,
- riječi nisu dulje od 128 znakova.

Program treba iz ulazne datoteke u novu izlaznu datoteku prepisati riječi s više od 4 znaka, koje ne sadrže niti jedno slovo. Svaku riječ treba napisati u novi red.

Opisani zadatak je **filtriranje** ulaza po nekom pravilu.

Zadatak — Uvjetno kopiranje riječi ("filter")

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5  int main(int argc, char *argv[]){
6      char s[129];
7      FILE *ulaz, *izlaz;
8      int uvjet, n, i;
9  /* Kontrolni ispis na stdout. */
10     printf("Ulazna␣datoteka:␣%s\n", argv[1]);
11     printf("Izlazna␣datoteka:␣%s\n", argv[2]);
12     if ((ulaz = fopen(argv[1], "r")) == NULL) {
13         printf("␣Greska␣na␣ulazu!\n");
14         exit(2); }
15     if ((izlaz = fopen(argv[2], "w")) == NULL) {
16         printf("␣Greska␣na␣izlazu!\n");
17         exit(3); }
```

Zadatak — Uvjetno kopiranje riječi ("filter")

```
18 /* Uociti test u while: " ... == 1". */
19    while (fscanf(ulaz, "%128s", s) == 1) {
20            n = strlen(s);
21 /* Kontrolni ispis na stdout. */
22    printf("duljina = %d, rijec = %s\n", n, s);
23    if (n > 4) {
24       uvjet = 1;
25       i = 0;
26 /* U uvjetu petlje moze i s[i] != '\0'. */
27       while (i < n && uvjet) {
28             uvjet = uvjet && !isalpha(s[i]);
29             ++i; }
30       if (uvjet)
31          fprintf(izlaz, "%s\n", s); } }
32    fclose(ulaz);
33    fclose(izlaz);
34    return 0; }
```

Zadatak — Uvjetno kopiranje riječi ("filter")

Uz pretpostavke:

- ulazna datoteka se zove `fzad1.in`,
- izlazna datoteka se zove `fzad1.out`.

Komandna linija za izvršavanje programa ima oblik:

```
fzad1 fzad1.in fzad1.out
```

Za ulaz:

```
12345a 12345 a 1111
333333 aaaaa 222222
```

Pripadna izlazna datoteka je:

```
12345
333333
222222
```

Uvjetno kopiranje riječi - zadaci

1. Prepravite unutarnju `while` petlju tako da ima samo jedan uvjet i koristi naredbu `break` (ubrzanje). Ideja: ako je `s[i]` slovo, stavmio uvjet na 0 i prekinemo petlju.
2. Probajte staviti drugačije uvjete i naredbe za čitanje u vanjskoj petlji. Pažljivo testirajte za razne ulazne podatke:
   - bjeline ispred prve riječi u redu (liniji),
   - bjeline iza zadnje riječi u redu (liniji),
   - prazne linije,
   - linija koja ima samo bjeline,
   - linija koja ima samo bjeline i javlja se na samom kraju ulazne datoteke.