# Programiranje 2
# 10. predavanje

Saša Singer

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

PMF – Matematički odsjek, Zagreb

# Sadržaj predavanja

- Datoteke (prvi dio):
  - Uvod.
  - Vrste datoteka — tekstualne i binarne.
  - Otvaranje i zatvaranje datoteke.
  - Standardne datoteke — `stdin`, `stdout`, `stderr`.
  - Funkcije za čitanje i pisanje — znak po znak, liniju po liniju, formatirani ulaz i izlaz.
  - Funkcije za prepoznavanje greške.

# Informacije

Trenutno nema bitnih informacija.

# Datoteke

# Sadržaj

- Datoteke:
  - Uvod.
  - Vrste datoteka — tekstualne i binarne.
  - Otvaranje i zatvaranje datoteke.
  - Standardne datoteke — `stdin`, `stdout`, `stderr`.
  - Funkcije za čitanje i pisanje — znak po znak, liniju po liniju, formatirani ulaz i izlaz.
  - Funkcije za prepoznavanje greške.
  - Binarni ulaz i izlaz.
  - Direktni pristup podacima.
  - Čitanje i pisanje u istoj datoteci.
  - Primjeri i zadaci.

# Osnovno o datotekama

Datoteka je prostor (ili područje) za

- trajno spremanje podataka u nekoj vanjskoj, sporijoj memoriji,

a ne u standardnoj radnoj memoriji računala (RAM).

Tipični mediji za spremanje datoteka su:

- disk, disketa, optički mediji (CD, DVD), "flash"–memorije, trake, kazete, i sl.

Trajno spremanje znači da

- podaci u datoteci "žive" dulje od trajanja izvršavanja programa.

Posljedica. Ista datoteka može se koristiti u više programa.

# *Osnovno o datotekama (nastavak)*

U sklopu operacijskog sustava računala,

- datoteke su organizirane u posebni "sustav datoteka" (engl. "file–system").

Unutar tog sustava datoteka,

- svaka datoteka ima svoje ime — koje koristimo za pristup datoteci.

Napomena: Pravila za pisanje (tvorbu) imena datoteka

- specifična su za pojedini operacijski sustav.

To posebno vrijedi za tzv. "puno" ime datoteke, koje sadrži i

- "stazu" ili "put" (engl. "path") do te datoteke u cijelom sustavu datoteka.

# Ime datoteke

Osnovno ime datoteke, bez "puta" do nje, standardno ima sljedeći oblik (Unix, Windows):

```
ime.nastavak
```

s tim da "`nastavak`" označava vrstu sadržaja datoteke.

Primjer.

- `sort.c` — tekst programa pisanog u C-u,
- `blabla.txt` — obični tekst,
- `math.lib` — biblioteka već prevedenih funkcija pripremljena za "linker".
- `sort.exe` — izvršni (binarni) kôd programa (Windows).

Stvarno, znak "točka" u imenu datoteke ima ponešto različito značenje na Windowsima i na Unixima.

# Operacije s datotekama

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

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

Postoje dvije "grube" podjele datoteka, prema tome što se "zbiva" u ovim operacijama:

- prva podjela — po načinu pristupa podacima u datoteci,
  - na slijedne (ili sekvencijalne) i direktne datoteke;
- druga podjela — po interpretaciji sadržaja podataka u datoteci, pri čitanju i pisanju,
  - na formatirane i neformatirane datoteke.

# Podjela datoteka — po načinu pristupa

Postoje dva bitno različita načina

- pristupa podacima u datoteci pri čitanju i pisanju,

tj. dva načina realizacije ovih operacija. Stvarno, datoteka može biti ista, samo je način pristupa podacima u njoj različit.

Slijedni ili sekvencijalni pristup (odnosno, datoteka):

- čita i piše se samo u jednom smjeru — unaprijed, podatak za podatkom, kao na traci.

Ovo je standardni način pristupa podacima u C-u.

Direktni pristup (odnosno, datoteka):

- čita i piše se bilo gdje u datoteci, slično kao u polju.

Realizira se posebnim funkcijama za pozicioniranje u datoteci.

# *Podjela datoteka — po interpretaciji sadržaja*

Neki podatak, na pr. cijeli broj, možemo zapisati u datoteku na dva bitno različita načina:

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

Potpuno ista stvar vrijedi i kod čitanja.

U skladu s tim, po načinu zapisa ili po interpretaciji sadržaja, datoteke možemo podijeliti na

- formatirane i neformatirane.

U C-u ova podjela nije "tvrda" (za razliku od na pr. Fortrana).

# Formatirane i neformatirane datoteke

Stvarno, datoteka je naprosto (konačan) niz byteova. Kod operacija pisanja i čitanja podataka,

- te byteove možemo zapisati, odnosno, interpretirati

na dva različita načina.

Formatirani zapis (odnosno, datoteka):

- sadržaj se interpretira kao tekstualni zapis podataka.

Neformatirani zapis (odnosno, datoteka):

- 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 (smije 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 je, naprosto, niz podataka tipa `char`, tj.

- niz znakova.

Tip `char` standardno odgovara jednom byteu.

Tekstualna datoteka ima dodatnu strukturu, kao tekst. To je

- 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. Na primjer:

- Unix — baš znak `\n` (line feed ili newline),
- Windows — paket od dva znaka `\r` (carriage return) i `\n`,
- Mac OS — samo 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 tzv. 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 još jedan osnovni pojam:

- spremnik (engl. "buffer").

## *Spremnik (buffer)*

Stvarna komunikacija između korisničkog programa i datoteke vrši se preko

- posebnog prostora u memoriji računala, kojeg zovemo spremnik (engl. buffer).

Zašto? Zbog razlike u brzini između

- centralnih dijelova računala (procesor, memorija) i
- vanjskih ulazno–izlaznih jedinica (na pr., disk).

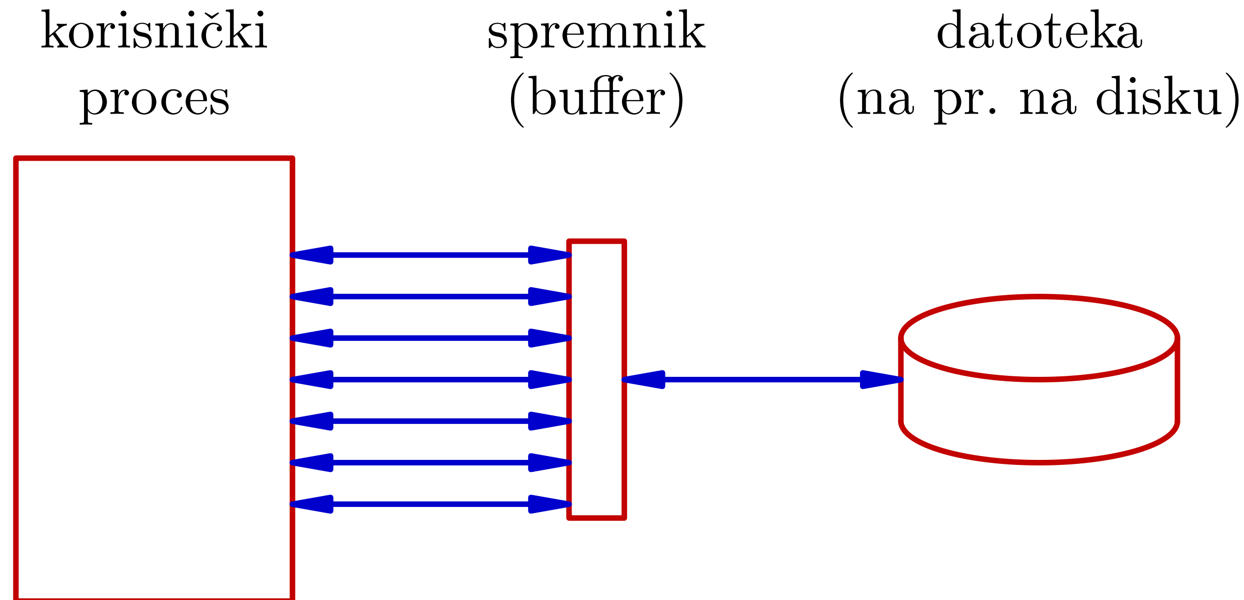
Sjetite se hijerarhijske građe memorije!

U taj spremnik privremeno se pohranjuju

- sve informacije koje se šalju u datoteku ili primaju iz datoteke.

# Spremnik (nastavak)

Slikica spremnika i komunikacije:




Svrha spremnika je

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

## Spremnik (nastavak)

Ovaj spremnik nalazi se “na strani” računala — kao dio operacijskog sustava za rad s datotekama.

Većina ulazno–izlaznih uređaja ima još i

- svoj vlastiti spremnik (katkad, čak i cache),

s potpuno istom 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, naravno, ovise o operacijskom sustavu.

Ovu strukturu, katkad, isto zovemo "spremnik" ili "file buffer" — po jednom dijelu njezinog sadržaja.

Tu su "skriveni" svi detalji implementacije datoteka, koje korisnici (srećom) ne moraju znati!

Ipak, ako vas zanima:

- Što se sve pamti u strukturi tipa `FILE`?

# *Sadržaj strukture tipa* `FILE`

Sadržaj strukture tipa `FILE` je “hrpa” toga:

- osnovne informacije o datoteci (ime i sl.),
- vrsta operacije — da li se čita ili piše (tzv. `file_mod`),
- status operacija — je li došlo do neke greške ili do kraja datoteke (v. `ferror`, `feof`),
- stvarna trenutna pozicija u datoteci (nula je početak) — tamo gdje ide sljedeće čitanje ili pisanje (v. `ftell`),
- stvarna lokacija spremnika za komunikaciju,
- trenutna pozicija u spremniku (koliki dio smo “pojeli”), jer, kad stignemo do kraja — mora se fizički učitati ili napisati novi blok podataka,

i još ponešto toga, što trenutno nije bitno.

## Spremnik — otvaranje i zatvaranje datoteke

Svakoj datoteci s kojom radimo u programu

- pridružen je odgovarajući objekt tipa `FILE`.

To je "spremnik" za tu datoteku.

Bitno: Spremnik je dinamički objekt! Dakle, do tog objekta tipa `FILE` dolazimo

- preko pokazivača na njega (tzv. "file pointer")!

Na početku rada s datotekom, taj objekt 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 pokazivati na spremnik za tu datoteku, nakon otvaranja.

Primjer.

```
FILE *fp;
```

Sljedeći korak je otvaranje datoteke, tj.

- kreiranje pripadnog spremnika (alokacija memorije) i uspostavljanje komunikacije sa stvarnom datotekom u operacijskom sustavu.

Napomena. 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), a
- `tip` — string koji kaže kako treba otvoriti tu datoteku, tzv. "način rada" ili `file_mod` (v. malo kasnije).

Funkcija `fopen` vraća:

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

Savjet: Provjerite vraćeni pokazivač!

# *Otvaranje datoteke — funkcija* `fopen` *(nastavak)*

Primjer. 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");
    ...
}
```

Ovdje je `ime` pravo ime datoteke, onako kako se datoteka zaista “zove” u operacijskom sustavu,

- na primjer: `"podaci.dat"`.

Drugi string `tip` je jedan od sljedećih stringova.

# *Tipovi (`file_mod`) — tekstualne datoteke*

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

| tip | značenje |
|---|---|
| `"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+"`):

- ako se postojeća datoteka otvori s `"w"` ili `"w+"`, njezin sadržaj se briše (oprez!) i pisanje će početi od početka.

Dodavanje (`"a"` ili `"a+"`):

- ako datoteka koju otvaramo s `"a"` ili `"a+"` ne postoji, bit će kreirana i pisanje onda ide od početka,
- ako postoji, novi tekst bit će dodavan 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`.

| tip | značenje |
|---|---|
| `"rb"` | binarno čitanje iz postojeće, |
| `"wb"` | binarno pisanje, kreiranje nove, |
| `"ab"` | binarno dodavanje, |
| `"rb+"` ili `"r+b"` | binarno čitanje i pisanje iz postojeće, |
| `"wb+"` ili `"w+b"` | binarno čitanje i pisanje, kreiranje nove, |
| `"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.

Napomena. Zatvaranje je nužno — u protivnom, kod pisanja može doći do gubitka podataka, ako program završi greškom!

# Zatvaranje datoteke — funkcija `fclose` (nast.)

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,
- i oslobađa memoriju za spremnik.

Primjer.

```
fclose(fp);
```

Oprez: pokazivač `fp` se ne mijenja, ali pokazuje "u ništa" i "zdravo" ga je staviti na `NULL` — isto kao kod `free`.

# Otvaranje i zatvaranje datoteke — primjer

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

```
#include <stdio.h>
#include <stdlib.h>
    ...
    FILE *fp;
    if ((fp = fopen("primjer.dat", "w")) == NULL) {
        printf("Ne mogu otvoriti datoteku!\n");
        exit(EXIT_FAILURE);    /* exit(1); */
    }
        /* Rad s datotekom (pisanje u nju). */
    ...
    fclose(fp);
```

# Standardne datoteke

Svakom C programu, u trenutku kad počinje izvršavanje, 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 (engl. “redirection”).

Pri pozivu programa, na komandnoj liniji, možemo standardne datoteke `stdin` i `stdout` preusmjeriti na neke druge datoteke.

Primjer.

```
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

# *Funkcije za čitanje i pisanje — pregled*

Funkcije za rad s datotekama odgovaraju već poznatim funkcijama za čitanje i pisanje:

| standardni ulaz/izlaz | rad s datotekom |
|---|---|
| `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);
int getc(FILE *fp);
```

Funkcije `fgetc` i `getc` vraćaju:

- sljedeći znak iz datoteke na koju pokazuje `fp`, ili
- `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` i zato je izlazni tip `int`.

Naime, "oznaka" za `EOF` ne smije biti legalni znak u datoteci.

## *Funkcije* `fgetc`, `getc` *(nastavak)*

Jedina razlika između `fgetc` i `getc` je sljedeća:

- `fgetc` mora biti "prava" funkcija, dok
- `getc` može biti implementirana i kao makro naredba.

  U tom slučaju, `getc` smije "evaluirati" argument `fp` i više puta, a ne samo jednom (što onda može izazvati "neželjene" efekte — pogledati zadnje predavanje o makro naredbama s parametrima).

Funkcija `getchar()` za standardni ulaz implementira se kao

- `getc(stdin)`.

# Obrada datoteke znak po znak — primjer

Primjer. Obrada datoteke čitanjem znak po znak, tipično se radi na sljedeći način ("predložak"):

```
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

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

---

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

  /* Broj znakova u datoteci. */

int main(int argc, char *argv[])
{
    FILE *fp;
    int ch, brojac = 0;
```

# Broj znakova u datoteci (nastavak)

```
if (argc == 1) {    /* Nema imena datoteke! */
    printf("Uporaba: %s ime_datoteke\n",
           argv[0]);
    exit(1);
}

if ((fp = fopen(argv[1], "r")) == NULL) {
    printf("Ne mogu otvoriti datoteku %s!\n",
           argv[1]);
    exit(2);
}
```

# Broj znakova u datoteci (nastavak)

```
    while ((ch = fgetc(fp)) != EOF) ++brojac;
    fclose(fp);

    printf("Broj znakova u datoteci %s = %d\n",
           argv[1], brojac);
    return 0;
}
```

# *Broj znakova u datoteci — rezultat*

Ovaj program spremljen je u datoteci `br_zn.c`, koja ima 623 znaka (Windows). Kad ga izvršimo baš na toj datoteci

```
br_zn br_zn.c >br_zn.out
```

dobijemo izlaz:

- `Broj znakova u datoteci br_zn.c = 594`.

Razlog: datoteka iz koje čitamo otvorena je s `"r"`, tj. kao tekstualna datoteka, pa funkcija `fgetc`

- pretvara standardni kraj reda `\r`, `\n` (na Windowsima)
- u znak `\n` za naš program.

Zato nam “fali” 29 znakova (toliko je linija u programu).

# Broj znakova u binarnoj datoteci — rezultat

Ako istu datoteku otvorimo kao binarnu ("rb"),

```
...
if ((fp = fopen(argv[1], "rb")) == NULL) {
...
```

onda

- nema pretvaranja kraja linije u `fgetc`,

pa dobivamo korektan broj znakova!

Pripadni program `br_zn_b.c` ima 624 znaka (Windows) i uredno (na samom sebi) daje izlaz:

- `Broj znakova u datoteci br_zn_b.c = 624`.

## *“Vraćanje” učitanog znaka — funkcija* `ungetc`

Deklaracija (prototip):

```
int ungetc(int c, FILE *fp);
```

Funkcija `ungetc` “vraća”

- znak `c` (pretvoren u `unsigned char`) “natrag” u `*fp`, u pripadni “buffer”, u `FILE` strukturi na koju pokazuje `fp`,
- i čini taj znak dostupnim za ponovno čitanje.

Taj znak će se ponovno pročitati kod sljedećeg čitanja.

Izlazna vrijednost je:

- znak `c` — ako je uspješno “vraćen” u spremnik, ili
- `EOF` — u slučaju greške.

## *Funkcija* `ungetc` *(nastavak)*

Po standardu, u svakom trenutku, dozvoljeno je “vratiti”

- najviše jedan znak “natrag” u spremnik za datoteku,
- i taj znak ne smije biti `EOF`.

Što je svrha funkcije `ungetc` ili “vraćanja” znakova?

Kod obrade “gramatički” strukturiranog teksta (poput riječi), vrlo često se događa da

- kraj neke vrste objekata prepoznamo tako da učitamo prvi znak sljedećeg objekta.
- Umjesto da ga posebno pamtimo, naprosto ga “vratimo” u spremnik za nastavak čitanja!

Tipični primjer je praznina (blank) kao kraj riječi.

# *Funkcija* `ungetc` *(nastavak)*

Primjer.

```
int c;                  /* Bolje od char c. */

...                     /* Citamo znak po znak. */
c = fgetc(fp);
...                     /* Prepoznamo kraj. */
ungetc(c, fp);          /* Vratimo c u spremnik. */
...
c = fgetc(fp);          /* Opet procitamo c. */
```

## *Pisanje znak po znak — funkcije* `fputc`, `putc`

Deklaracija (prototip):

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

- znak `c` — ako je uspješno napisan u datoteku, ili
- `EOF` — u slučaju greške.

## *Funkcije* `fputc`, `putc` *(nastavak)*

Jedina razlika između `fputc` i `putc` je sljedeća:

- `fputc` mora biti “prava” funkcija, dok
- `putc` može biti implementirana i kao makro naredba.

  U tom slučaju, `putc` smije “evaluirati” argument `fp` i više puta, a ne samo jednom (što onda može izazvati “neželjene” efekte — pogledati zadnje predavanje o makro naredbama s parametrima).

Funkcija `putchar(c)` za standardni izlaz implementira se kao

- `putc(c, stdout)`.

# Kopiranje datoteke znak po znak

Primjer. Napisati 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`. Program se zove `fcopy_1.c`.

---

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

int main(int argc, char *argv[])
{
    FILE *in, *out;
    int c;    /* Ne: char c! */
```

# *Kopiranje datoteke znak po znak (nastavak)*

```
if (argc != 3) {      /* Nema imena datoteka! */
    fprintf(stderr, "Uporaba: %s ime1 ime2\n",
            argv[0]);
    exit(1);
}
if ((in = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Ne mogu citati: %s!\n",
            argv[1]);
    exit(2);
}
if ((out = fopen(argv[2], "w")) == NULL) {
    fprintf(stderr, "Ne mogu pisati: %s!\n",
            argv[2]);
    exit(3);
}
```

## Kopiranje datoteke znak po znak (nastavak)

```
    while ((c = fgetc(in)) != EOF)
        fputc(c, out);

    fclose(in);
    fclose(out);

    return 0;
}
```

---

Komentar. Ovakvo kopiranje datoteke byte–po–byte je najsporiji način kopiranja.

Prednost: jednostavno, i (uglavnom) najsigurnije.

Bolje (bitno brže) je kopirati u većim blokovima.

# *Kopiranje datoteke znak po znak — funkcija*

Primjer. Napisati 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).

```
void copy_file(FILE *in, FILE *out)
{
    int c;    /* Ne: char c! */

    while ((c = fgetc(in)) != EOF)
        fputc(c, out);

    return;
}
```

Pripadni program: `fcopy_1f.c` (koristi ovu funkciju).

# Čitanje liniju po liniju — funkcija `fgets`

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

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

gdje su:

- `str` — pokazivač na dio memorije (engl. 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` *(nastavak)*

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, na primjer, sljedećim pozivom funkcije `fgets`.

Izlazna vrijednost funkcije `fgets` je:

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

## *Funkcija* `gets` *— za standardni ulaz (NE!)*

Funkcija `gets` čita string sa standardnog ulaza `stdin`.

```
char *gets(char *str);
```

Uočite da `gets` ne prima veličinu buffera `str` kao argument. Može se dogoditi da je

- ulazna linija dulja od za nju rezervirane memorije u `str`.

Savjet: umjesto `gets(str)`,

- bolje je koristiti `fgets(str, n, stdin)`.

Dodatna razlika između `fgets` i `gets`:

- `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 `'\0'` na kraju stringa se ne ispisuje.

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

Izlazna vrijednost je:

- nenegativan broj — ako je ispis uspio, ili
- `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, dok
- `puts` dodaje znak `'\n'` na kraj (umjesto znaka `'\0'`).

Ove razlike u ponašanju na znak `'\n'`, između

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

potpuno 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 `format` string i ostale argumente su ista kao prije!

## *Funkcije* `fscanf` *i* `fprintf` *(nastavak)*

To znači da je

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

Funkcija `fscanf` vraća:

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

Funkcija `fprintf` vraća:

- nenegativan broj napisanih znakova, ili
- negativan broj — u slučaju greške.

Napomena. Isplati se provjeriti izlaznu vrijednost!

# *Funkcije* `fscanf` *i* `fprintf` *— primjer*

Primjer. Formatirano pisanje u datoteku.

```
int kolicina = 50;
double cijena = 7.50;
...
fprintf(fp, "%5d, %10.2f\n", kolicina, cijena);
```

Primjer. Formatirano čitanje takvog teksta iz datoteke.

```
int kolicina;
double cijena;
...
fscanf(fp, "%d, %lf\n", &kolicina, &cijena);
```

Razmak iza `%d,` nije nužan.

# 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*

Primjer. Formatirano čitanje teksta iz stringa i pisanje u string (v. program `fio_str.c`).

```
#include <stdio.h>

int main(void)
{
    char *ulaz = " 50,  7.50\n";     /* Ulaz! */
    char izlaz[80];
    int kolicina;
    double cijena;

    sscanf(ulaz, "%d,%lf\n", &kolicina, &cijena);
    printf("%5d, %10.2f\n", kolicina, cijena);
```

## *Funkcije* `sscanf` *i* `sprintf` *— primjer (nastavak)*

```
    sprintf(izlaz, " kolicina = %5d\n"
            " cijena = %10.2f\n", kolicina, cijena);
    printf("%s\n", izlaz);

    return 0;
}
```

Izlaz programa ima 4 linije (zadnja je prazna, piše ju `printf`):

```
   50,        7.50
 kolicina =    50
 cijena =       7.50

```

Korist: Za brzo pretvaranje broja u niz znakova i obratno!

## Zadatak — Uvjetno kopiranje riječi ("filter")

Primjer. Napisati 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 sve riječi

- s više ($>$) od 4 znaka, koje ne sadrže niti jedno slovo.

Svaku riječ treba napisati u novi red.

Napomena. Posao = "filtriranje" ulaza po nekom pravilu.

## Uvjetno kopiranje riječi (nastavak)

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

int main(int argc, char *argv[])
{
    char s[129];
    FILE *ulaz, *izlaz;
    int uvjet, n, i;

        /* Kontrolni ispis na stdout. */
    printf("Ulazna datoteka: %s\n", argv[1]);
    printf("Izlazna datoteka: %s\n", argv[2]);
```

## *Uvjetno kopiranje riječi (nastavak)*

```
if ((ulaz = fopen(argv[1], "r")) == NULL) {
    printf(" Greska na ulazu!\n");
    exit(2);
}
if ((izlaz = fopen(argv[2], "w")) == NULL) {
    printf(" Greska na izlazu!\n");
    exit(3);
}
    /* Uociti test u while: " ... == 1". */

while (fscanf(ulaz, "%128s", s) == 1) {
    n = strlen(s);
        /* Kontrolni ispis na stdout. */
    printf("duljina = %d, rijec = %s\n", n, s);
```

# Uvjetno kopiranje riječi (nastavak)

```
        if (n > 4) {
            uvjet = 1;
            i = 0;

                /* U uvjetu petlje, umjesto i < n,
                   moze se staviti s[i] != '\0'. */
            while (i < n && uvjet) {
                uvjet = uvjet && !isalpha(s[i]);
                ++i;
            }
            if (uvjet)
                fprintf(izlaz, "%s\n", s);
        }
    }
```

## Uvjetno kopiranje riječi (nastavak)

```
    fclose(ulaz);
    fclose(izlaz);

    return 0;
}
```

Program se zove `f_zad_1.c`. Uzmimo da se

- ulazna datoteka zove `f_zad_1.in`, a
- izlazna `f_zad_1.out`.

Komandna linija za izvršavanje programa ima ovaj oblik:

```
    f_zad_1 f_zad_1.in f_zad_1.out
```

# Uvjetno kopiranje riječi — rezultat

Ulazna datoteka `f_zad_1.in` ima točno dvije linije (7 riječi):

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

Kraj svake linije je znak `'\n'` (na Windowsima: `'\r'`, `'\n'`).

Pripadna izlazna datoteka `f_zad_1.out` je:

```
12345
333333
222222
```

Napomena. Test u vanjskoj petlji glasi `fscanf(...) == 1`. U ovom zadatku može i `fscanf(...) != EOF`, zbog čitanja stringova i preskakanja bjelina (v. `f_zad_1n.c`). Inače, oprez!

# Uvjetno kopiranje riječi — zadaci

Zadatak. Prepravite unutarnju `while` petlju tako da ima samo jedan uvjet i koristi naredbu `break` (ubrzanje). Ideja:

- Ako je `s[i]` slovo, stavimo `uvjet` na `0` i prekinemo petlju.

Zadatak. 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;
- takva linija dolazi na samom kraju ulazne datoteke.

# Funkcije za prepoznavanje greške

# Prepoznavanje greške — funkcije `ferror`, `feof`

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

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

Zato postoje funkcije:

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

koje služe za

- razlikovanje ta dva slučaja.

# *Funkcije* `ferror`, `feof` *(nastavak)*

Funkcija `ferror` vraća:

- broj različit od nule (istina) — ako je došlo do greške, a
- nulu (laž) — ako nije.

Funkcija `feof` vraća:

- broj različit od nule (istina) — ako smo naišli na kraj datoteke prilikom čitanja (pokušaj čitanja preko kraja), a
- nulu (laž) — u suprotnom.

Provjeru vrijednosti ovih funkcija treba napraviti

- odmah nakon operacije čitanja iz datoteke.

Može i nakon pisanja, ali pisanje nije uspješno samo u slučaju greške (`ferror` je tada istina).

# *Funkcije* `ferror`, `feof` *(nastavak)*

Svaka otvorena datoteka ima dva indikatora (ili zastavice, engl. flag) statusa te datoteke u pripadnoj strukturi tipa `FILE`:

- `eof` (end–of–file) "flag" — jesmo li došli do kraja te datoteke,
- `error` "flag" — je li došlo do greške prilikom operacije (na primjer, pišemo, a disk se napuni do kraja).

Funkcije `ferror` i `feof` samo testiraju stanje tih indikatora i vraćaju odgovarajuću logičku vrijednost.

Otvaranje datoteke `fopen(...)` briše stanje oba indikatora.

Dakle, odmah nakon uspješnog otvaranja datoteke vrijedi:

- `feof(...) == 0` i `ferror(...) == 0`.

# *Funkcije* `ferror`, `feof` *(nastavak)*

Napomena. Ove “flagove” postavljaju funkcije za ulazne i izlazne operacije na datoteci, tj.

- stanje indikatora odnosi se na prethodnu operaciju.

Zato provjeru stanja treba napraviti odmah nakon operacije.

Indikator `error` signalizira grešku i kod čitanja i kod pisanja.

Za razliku od toga, `feof` treba testirati samo nakon čitanja!

- Čitanje postavlja `eof` indikator samo ako smo pokušali čitati preko kraja datoteke — i to je jedini način za detekciju kraja datoteke.
- Kod pisanja — nije precizno definirano! Obično, pisanje briše `eof` indikator, tj. dobivamo `feof(...) == 0`. Dakle, nema smisla testirati `feof` nakon pisanja.

# “Brisanje” indikatora — funkcija `clearerr`

Brisanje indikatora možemo napraviti i sami. Funkcija

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

- briše stanje oba indikatora za datoteku na koju pokazuje `fp` — postavlja ih na nulu.

# *Funkcija* `feof` *— primjeri*

Primjer. Treba kopirati sadržaj datoteke `in` u datoteku `out`, znak po znak, kao u programu `fcopy_1.c`.

Pretpostavimo, na trenutak, da nema grešaka pri čitanju i pisanju, tj. ne treba provjeravati `ferror`, već samo `feof`.

Zatim, kopiranje realiziramo ovako — `do-while` petljom:

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

Što će se dogoditi (v. program `fcopy_2a.c`)?

Odgovor. Kad je `c == EOF`, napišemo ga, prije `feof` testa!

## *Funkcija* `feof` *— primjeri (nastavak)*

Može i ovako — `while` petljom (po ugledu na Pascal):

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

Opet, što će se dogoditi (v. program `fcopy_2b.c`)?

Odgovor. Ista greška kao i malo prije! Dodatno, ovdje još

- testiramo `feof` prije čitanja, a tada je `feof(in) == 0`. To vrijedi čak i kad je `in` prazna datoteka (od 0 znakova) — tek nakon prvog čitanja, `feof(in)` postaje istina!

U oba slučaja, zbog pisanja `EOF` (ovdje $= -1$) kao `char`, dobijemo dodatni byte 255 na kraju `out` datoteke.

## *Primjer za prepoznavanje greške*

Primjer. Napisati funkciju koja kopira sadržaj jedne datoteke u drugu, liniju po liniju (obje datoteke su otvorene). Funkcija treba prepoznati i javiti greške prilikom čitanja i pisanja.

Za prepoznavanje greške prilikom čitanja koristimo funkciju `ferror`. Vraćamo 0 ili kôd greške (v. program `fcopy_3f.c`).

---

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

#define MAX_LINE 129

int copy_file(FILE *in, FILE *out)
{
    char buf[MAX_LINE];     /* Ulazni buffer. */
```

# Primjer za prepoznavanje greške (nastavak)

```
    while (fgets(buf, MAX_LINE, in) != NULL)
        if (fputs(buf, out) == EOF) {
            fprintf(stderr,
                    "\nGreska u pisanju.\n");
            return 1;    /* Necu exit(1); */
        }

    if (ferror(in)) {
        fprintf(stderr, "\nGreska u citanju.\n");
        return 2;    /* Necu exit(2); */
    }

    return 0;
}
```

# Napomene uz čitanje i pisanje

Operacije čitanja i pisanja možemo (naravno) raditi

- i s tekstualnim i s binarnim datotekama.

Moguću razliku smo već ilustrirali kod operacija

- znak po znak — funkcije `fgetc` i `fputc`.

Izbor "tipa" datoteke, tj. načina otvaranja, ovisi o tome što točno treba napraviti.

S druge strane, ostale ulazno–izlazne operacije koje smo dosad napravili:

- liniju po liniju — funkcije `fgets` i `fputs`,
- formatirano — funkcije `fscanf` i `fprintf`,

prirodno se rade s tekstualnim datotekama.

# Napomene uz čitanje i pisanje (nastavak)

Dosad spomenutim funkcijama možemo čitati i pisati samo:

- znakove — formatirano ili neformatirano,
- vrijednosti ostalih standardnih tipova — i to samo formatirano.

Čitanje i pisanje objekata svih ostalih tipova,

- posebno, složenih tipova — poput struktura ili polja,

ne možemo napraviti na taj način,

- osim po komponentama — što je izrazito nepraktično.

Za takve tipove možemo (moramo) koristiti

- binarno čitanje i pisanje.