

# *Programiranje (C)*

## *12. predavanje*

Saša Singer

[singer@math.hr](mailto:singer@math.hr)  
[web.math.hr/~singer](http://web.math.hr/~singer)

PMF – Matematički odjel, Zagreb

# *Sadržaj predavanja*

- Datoteke (nastavak):
  - Binarni ulaz i izlaz.
  - Direktan pristup podacima.
  - Primjeri.

# Binarno čitanje i pisanje

# **Binarne datoteke — uvod**

U praksi često trebamo **datoteke** koje sadrže

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

u internoj **binarnoj** reprezentaciji — bez pretvaranja u tekst.

Na primjer, tako

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

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

- **fread** i **fwrite**, koje **doslovno kopiraju** sadržaj zadalog **bloka** byte-ova, tj. znakova — podataka tipa **char**.

# *Binarno čitanje i pisanje*

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

---

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

---

Argumenti funkcija su:

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

# **Binarno čitanje — funkcija fread**

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

## Funkcija **fread**

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

Izlazna vrijednost funkcije je:

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

# *Binarno pisanje — funkcija fwrite*

## Funkcija `fwrite`

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

Izlazna vrijednost funkcije je:

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

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

# *Binarno čitanje i pisanje — primjer*

Primjer. Čitanje cijelog **polja** cijelih brojeva iz datoteke:

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

Primjer. Pisanje cijelog **polja** cijelih brojeva u datoteku:

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

## *Binarno pisanje strukture — primjer*

Primjer. Zapis jedne strukture u datoteku.

---

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

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

---

# *Binarno čitanje i pisanje — komentari*

Prednost binarnog ulaza/izlaza:

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

Nedostatak binarnog ulaza/izlaza:

- ovisnost o arhitekturi računala i prevoditelju.

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

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

# Direktan pristup podacima

# *Sekvencijalni pristup podacima*

Sve ulazno-izlazne operacije koje smo radili do sada koristile su

- tzv. sekvencijalni pristup podacima u datoteci.

Što to znači?

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

- čitanje i pisanje ide od početka datoteke, a
- dodavanje ide na kraj datoteke, iza svega što već postoji u datoteci.

Nakon toga, svaka sljedeća ulazno-izlazna operacija

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

## *Trenutna pozicija u datoteci*

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

- **trenutna pozicija** u datoteci (tzv. **file\_pos**),  
do koje smo “stigli” s prethodnim **operacijama** na toj datoteci.

Trenutna pozicija se “mjeri”

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

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

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

- ovisno o dozvoljenoj **veličini** datoteke.

## *Trenutna pozicija u datoteci (nastavak)*

Kako se mijenja trenutna pozicija?

Svaka pojedina ulazno-izlazna operacija uvijek

- ide “unaprijed” u datoteci, od trenutne pozicije.

Zato, kad god napravimo neku operaciju čitanja ili pisanja,

- trenutna pozicija se povećava za broj pročitanih ili napisanih znakova.

Ako “nasilno” ne mijenjamo trenutnu poziciju, onda dobivamo

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

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

# *Direktan pristup podacima — uvod*

Međutim, dozvoljeno je

- promijeniti vrijednost trenutne pozicije u datoteci.

Kad to napravimo, onda

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

Na taj način možemo

- čitati i pisati podatke bilo gdje u datoteci,  
tj. svakom znaku (podatku) u datoteci pristupamo direktno,  
slično kao u polju.

Zato se ovaj način rada s datotekom zove

- direktan ili slučajan pristup podacima.

## *Direktan pristup podacima — realizacija*

Realizacija **direktnog** pristupa slična je **indeksiranju** kod **polja**:

- prije operacije, **zadajemo trenutnu poziciju u datoteci**.

To se radi posebnom funkcijom za **pozicioniranje** u datoteci.

Zadavanje pozicije je malo složenije nego kod polja, jer

- **promjena** mesta u datoteci može i malo **dulje** potrajati.

Zato imamo nekoliko mogućnosti za “relativno” zadavanje **nove trenutne pozicije** u datoteci.

Za **direktan** pristup podacima koristimo **dvije** funkcije:

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

## *Trenutna pozicija u datoteci — funkcija ftell*

Deklaracija (prototip):

---

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

---

Funkcija **ftell** vraća:

- trenutnu poziciju u već otvorenoj datoteci na koju pokazuje **fp** — tj. broj znakova od početka te datoteke.

Napomena. Odmah nakon **otvaranja** datoteke (može i s "a"), dobivamo da je **pozicija = 0L**.

Izlazna vrijednost je:

- nenegativan broj ( $\geq 0$ ) — u slučaju **uspjeha**, ili
- **-1L** — u slučaju **greške**.

# *Promjena pozicije u datoteci — funkcija fseek*

Deklaracija (prototip):

---

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

---

Argumenti funkcije **fseek** su:

- **fp** — pokazivač na već otvorenu **datoteku**,
- **offset** — zadani **pomak** u broju **znakova** (byte-ova),
- **origin** — indikator položaja ili “**ishodište**” od kojeg se broji **pomak**. Zadaje se jednom od sljedeće **tri simboličke konstante** (definirane u **<stdio.h>**):
  - **SEEK\_SET** — od **početka** datoteke,
  - **SEEK\_CUR** — od **trenutne** pozicije u datoteci,
  - **SEEK\_END** — od **kraja** datoteke.

## *Funkcija fseek (nastavak)*

Funkcija **fseek** postavlja **trenutnu poziciju**

- u **datoteci** na koju pokazuje **fp**,
- na **offset znakova** od zadanog “**ishodišta**” **origin**.

Izlazna vrijednost funkcije je:

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

## *Funkcija fseek — primjeri*

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

---

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

---

## Funkcija `fseek` — za tekstualne datoteke

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

- `offset` mora biti — nula, ili vrijednost koju vratí poziv funkcije `ftell`.

To znači da su dobro definirani **jedino** pozivi oblika:

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

## *Pozicioniranje na početak — funkcija rewind*

Pozicioniranje na **početak** datoteke možemo napraviti i pozivom funkcije

---

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

---

Ovaj poziv **ekvivalentan** je s:

---

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

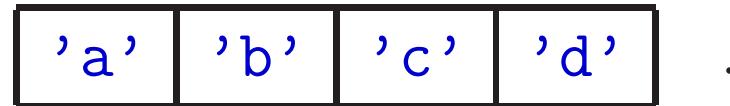
---

tj. osim pozicioniranja na **početak** datoteke,

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

## *Primjer za funkciju ftell*

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



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

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

Što je **rezultat**?

## *Primjer za ftell — dio programa*

Sljedeći dio programa realizira čitanje, nakon otvaranja:

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

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

## **Primjer za ftell — rezultati**

Izlaz tog dijela programa je:

---

Pozicija: 0, znak = a

Pozicija: 1, znak = b

Pozicija: 2, znak = c

Pozicija: 3, znak = d

Pozicija: 4, znak = -1

Pozicija: 4, znak = -1

---

“Znak” **-1** je standardna vrijednost za **EOF**.

Pripadni program je **fpos.c**. Taj program

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

## **Primjer za ftell — varijacije na temu**

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

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

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

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

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

- trenutna pozicija = 0L.

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

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

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

Pričuvni program je `fpos_ap.c`. Uočite da tik prije dodavanja znaka 'e' vrijedi:

- trenutna pozicija = 4L.

# Naopako kopiranje (*invertiranje*) datoteke

Primjer. Napisati program koji

- naopako kopira sadržaj jedne datoteke u drugu.

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

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

onda druga (izlazna) datoteka mora imati sljedeći oblik:

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

Kopiranje radimo znak po znak, tako da po prvoj datoteci

- idemo unatrag — od kraja datoteke, prema početku, koristeći direktni pristup podacima.

## **Naopako kopiranje datoteke (nastavak)**

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

Varijabla **pomak** broji pomak “unazad” od **kraja** datoteke (“ishodište” **SEEK\_END**).

---

```
#include <stdio.h>

int main()
{
    char *in_name = "freverse.in";
    char *out_name = "freverse.out";
    FILE *in, *out;
    long file_pos, pomak = 0L;
```

## *Naopako kopiranje datoteke (nastavak)*

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

if ((out = fopen(out_name, "wb")) == NULL) {
    fprintf(stderr, "Ne mogu pisati u: %s!\n",
            out_name);
    exit(1);
}
```

## *Naopako kopiranje datoteke (nastavak)*

```
/* Datoteku kopiramo naopako. */

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

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

    /* Sad je pozicija narasla za 1L. */
} while (file_pos != 0L);
```

## *Naopako kopiranje datoteke (nastavak)*

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

---

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

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

## **Naopako kopiranje datoteke — rezultat**

Ulagana datoteka **freverse.in** ima točno 33 znaka:

---

Ja sam mala Ruza, mamima sam kci.

---

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

---

.ick mas amimam ,azuR alam mas aJ

---

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

**Zadatak.** Napravite **istu** stvar

- sekvensijalnim pristupom podacima.

Složenost je tada **kvadratna** u **duljini** datoteke!

## **Naopako kopiranje datoteke — napomene**

Napomena. Obje datoteke moraju biti otvorene kao binarne.

U protivnom,

- čim ulazna datoteka ima bar jedan znak za kraj reda, stvar ne radi dobro na sustavima kod kojih
- postoji razlika između binarnih i tekstualnih datoteka (poput Windowsa).

Dovoljno je “naopako” kopirati datoteku na `stdout`.

Probajte program `frev_out.c`, koji

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

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

# *Naopako kopiranje datoteke — kraj napomene*

Uzrok greske je:

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

Upravo zato postoje ranije navedena ograničenja

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

# *Dodavanje bonusa na račun — početak*

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

---

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

int size = sizeof(Racun);
```

---

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

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

## Dodavanje bonusa na račun — zadatak

Treba napisati funkciju `dodaj_bonus` sa zaglavljem oblika:

---

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

---

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

Funkcija treba `n`-tom zapisu u datoteci

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

Brojanje zapisu u datoteci počinje od 1.

Za rješenje koristimo direktni pristup podacima u datoteci.

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

# Dodavanje bonusa na račun — pristup podacima

Za čitanje **n**-tog zapisa:

- treba “preskočiti” prvih **n - 1** zapisa od početka datoteke, tj. od “ishodišta” SEEK\_SET.

Odgovarajuće pozicioniranje je:

---

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

---

Nakon dodavanja bonusa, za pisanje “novog” **n**-tog zapisa:

- treba se “vratiti” natrag za jedan zapis od trenutne pozicije u datoteci, tj. od “ishodišta” SEEK\_CUR.

---

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

---

## *Dodavanje bonusa na račun — funkcija*

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

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

## *Dodavanje bonusa na račun — funkcija (nast.)*

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

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

## *Dodavanje bonusa na račun — funkcija (nast.)*

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

## *Dodavanje bonusa na račun — funkcija (nast.)*

```
if (kor.stanje > 0) {  
    kor.stanje = kor.stanje + bonus;  
    fseek(racuni, -size, SEEK_CUR);  
    if (fwrite(&kor, size, 1, racuni) != 1) {  
        fprintf(stderr, "Greska u pisanju.\n");  
        exit(1);  
    }  
}  
  
fclose(racuni);  
  
return;  
}
```

---

## *Dodavanje bonusa na račun — rezultati*

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

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

Polazna datoteka **racuni.dat** ima ovaj sadržaj:

---

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

---

## *Dodavanje bonusa na račun — rezultati (nast.)*

Zatim dodajemo bonus sljedećim zapisima:

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

Nova datoteka **racuni.dat** ima ovaj sadržaj:

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

# *Čitanje i pisanje u istoj datoteci*

Datoteku možemo **otvoriti** tako da je **dozvoljeno**:

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

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

- u tzv. **file\_mod** stringu, koji zadaje “**načina**” rada s datotekom — navedemo **znak +**.

U tom slučaju, treba biti **oprezan**

- pri **prijelazu** s **čitanja** na **pisanje** i **obratno**,  
zbog toga što postoji **spremnik** za komunikaciju između programa i datoteke.

## *Čitanje i pisanje u istoj datoteci (nastavak)*

Između jedne i druge vrste operacija na istoj datoteci, treba pozvati:

- funkciju za pozicioniranje u datoteci (`ftell`, `fseek`, `rewind`, a ima ih još), ili
- funkciju `fflush` za pražnjenje (pisanje) spremnika u datoteku (samo pri prijelazu s pisanja na čitanje).

# *Pisanje spremnika u datoteku — funkcija fflush*

Deklaracija (prototip):

---

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

---

Ako **fp** pokazuje na “izlaznu” datoteku (tj. zadnja operacija je bila **pisanje** u tu datoteku), onda **fflush**:

- piše u tu datoteku onaj dio sadržaja spremnika koji do tad nije bio “fizički” napisan u nju,  
tj. “prazni” spremnik u **datoteku**.

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

- nije definiran (nema smisla).

## *Funkcija fflush (nastavak)*

Poziv oblika:

---

```
fflush(NULL);
```

---

“prazni” spremnike za **sve** izlazne **datoteke** (u tom trenutku).

Izlazna vrijednost funkcije **fflush** je:

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