



Programiranje I

Fajlovi

Enumeracija

Strukture - Uvodno

Rad sa fajlovima

- Programski jezik C poznaje tip podatka "pokazivač na fajl".
- Da bi se radilo sa ovim tipom u C-u, mora biti uključena programska biblioteka `stdio.h`.
- Pokazivači na fajl se deklarišu kao:
`FILE *a;`
- Do trenutka inicijalizacije ove promjenljive (inicijalizacija se obavlja pomoću funkcije `fopen`), pokazivač je neupotrebljiv.

Funkcija **fopen**

- Funkcija **fopen** ima sintaksu:
a = **fopen**("fajl koji se otvara", "način otvaranja");
- Ako se fajl koji se otvara nalazi u tekućem folderu, može se navesti samo njegovo ime, npr. "Test.txt".
- Ako se fajl nalazi u nekom drugom folderu na disku, mora se navesti putanja do njega: "C:\\Temp\\Proba.txt". Zbog čega se koristi \\, a ne samo \?
- Ako se fajl otvara za čitanje, drugi argument **fopen**-a je "**r**".
- Ako je otvaranje obavljeno uspješno, pokazivač **a** se pozicionira na početak postojećeg fajla.

Funkcija **fopen**

- Ako se fajl otvara za upis, drugi argument je "**w**". Ako fajl ranije nije postojao biće kreiran, a ako je postojao biće izbrisan. U oba slučaja će pokazivač biti pozicioniran na početak otvorenog fajla.
- Nadovezivanje na kraj fajla (bez brisanja postojećeg sadržaja) se obavlja sa "**a**" (pokazivač se sada pozicionira na kraj fajla).
- Opcija "**+**" omogućava otvaranje i za čitanje i za upisivanje.
- Svaki od ovih karaktera može biti praćen slovom "**t**" ili slovom "**b**".

Funkcija **fopen**

- Kombinacija **"rt"** označava da se fajl otvara za čitanje u tekstualnom modu, dok, recimo, **"wb"** znači da se fajl otvara za upis u binarnom modu.
- Po pravilu, fajl se otvara u tekstualnom modu, tako da se slovo **t** može izostaviti.
- Kod binarnog režima podaci se u fajl smještaju u zavisnosti od binarne reprezentacije tipa podatka, dok se u tekstualnom režimu svi podaci smještaju u formi ASCII koda (jedan karakter-jedan bajt).

Funkcija **fopen**

- Na primjer, broj **1234567** se u binarnom režimu zapisuje kao **4 bajta** (pod uslovom da **int** zauzima **4 bajta**), dok u **tekstualnom** režimu zauzima **7 bajta**, za svaki karakter po jedan ('1', '2', '3', '4', '5', '6' i '7').
- Ako se fajl ne može otvoriti iz bilo kog razloga (ne postoji za čitanje, nema prostora za upis, zabranjen pristup), **fopen** vraća **NULL** pokazivač.
- Svaki poziv funkcije **fopen** mora biti praćen provjerom da li je fajl otvoren i preduzimanjem eventualnih korektivnih radnji.

Funkcija **fopen**

- Primjer otvaranja fajla sa provjerom

```
a = fopen("abc.txt", "r");  
if(a == NULL) exit(1);
```

- Postoji više funkcija za upis i čitanje iz fajla. Uglavnom liče na već poznate naredbe iz **stdio.h** biblioteke (i fajl funkcije pripadaju toj biblioteci).
- Razlikuju se po dodatnom argumentu koji predstavlja pokazivač na fajl.
- Na primjer, ako je fajl otvoren za upis, funkcija **fprintf(a, "%d \n", 2);** upisuje broj 2 i znak za novi red u fajl na koji pokazuje **a** (pozicionira se na početak narednog reda fajla).

Upis i čitanje iz fajla

- Sve funkcije za upis i čitanje iz fajla vrše pomjeranje pozicije za čitanje i upis tako da je naredna pozicija tamo gdje je stala prethodna funkcija.
- Funkcije `fputc(c, a)` i `putc(c, a)` upisuju karakter `c` u fajl `a`.
- Funkcija `fputs(s, a)` upisuje string `s` u fajl `a`, ali za razliku od `puts` ne prelazi u novi red.
- Funkcije `fgetc(a)` i `getc(a)` vraćaju tekući karakter iz fajla.

Upis i čitanje iz fajla

- Funkcija `fgets(s, n, a)` učitava string `s` iz fajla na koji pokazuje pokazivač `a`, ali ne više od `n` karaktera. Ovo je mala razlika u odnosu na `gets(s)` koja učitava string do znaka Enter, jer se u fajlovima može dogoditi da je sve štampano u jednom redu, tako da bi praktično čitav fajl bio učitao odjednom.
- Napominjemo da se na kraju fajla nalazi poseban simbol koji je definisan preko simboličke konstante `EOF` (End-Of-File). Prilikom upisa na kraj fajla vrši se pomjeranje ovog karaktera, dok se čitanje ne može vršiti nakon njega.

Upis i čitanje iz fajla

- Dakle, prilikom čitanja treba uvijek vršiti provjeru da li je dostignut kraj fajla.
- **Primjer:** Svaki red fajla sadrži po jedan cio broj. Formirati novi fajl koji se razlikuje od početnog po tome što su prvi, treći, peti itd. broj udvostručeni.

```
#include<stdio.h>
int main() {
int broj, i;
FILE *a, *b;
```

```
if((a = fopen("CioBroj.txt", "r")) == NULL) {
    puts("Greska pri otvaranju fajla");
    exit(1); }
if((b = fopen("NoviCio.txt", "w")) == NULL) {
    puts("Greska pri otvaranju fajla");
    exit(1); }
```

Primjer

```
i=1;
while(fscanf(a, "%d", &broj) != EOF) {
    fprintf(b, "%d\n", (i%2==1) ? 2*broj : broj);
    i++;
}
fclose(a);
fclose(b);
}
```

- Prvi fajl smo otvorili za čitanje, a drugi za upis. U istoj naredbi vršimo provjeru da li je otvaranje uspjelo poređenjem sa **NULL**. U slučaju da nije uspjelo, izlazimo iz programa.
- U petlji, zatim, učitavamo broj po broj iz fajla na koji pokazuje **a**, sve do kraja fajla.

- Pojasnimo ukratko program. Koristimo dvije cjelobrojne promjenljive **i** i **broj**. **i** služi za brojanje redova fajla, dok u **broj** učitavamo brojeve iz početnog fajla.
- Koristimo dva pokazivača na fajlove.

Primjer - pojašnjenje

- U slučaju da je broj *i* paran, u fajl na koji pokazuje pokazivač *b* se vrši njegov upis, a ako je neparan upisuje se njegova duplirana vrijednost.
- Na kraju se fajlovi zatvaraju (ove će naredbe biti kasnije objašnjene).
- Još jedan par funkcija se može koristiti za upis i čitanje iz fajla. Za upis podataka iz memorije u fajl može poslužiti funkcija

```
int fwrite(void *p, int d, int n, FILE *a);
```

fwrite i fread

Parametri funkcije `fwrite` su:

- `p` pokazivač na zonu u memoriji sa koje počinje prepisivanje u fajl
- `d` dužina (u bajtima) objekta koji se prepisuje
- `n` broj objekata koji se prepisuju
- `a` pokazivač na fajl u koji se vrši prepisivanje

- Funkcija za prepisivanje iz fajla u memoriju je `fread` i ima prototip:

```
int fread(void *p, int d, int n, FILE *a);
```

Značenje argumenata je kao i kod `fwrite`, ali se sada prepisuju podaci iz fajla u memoriju.

- Upis i čitanje se po pravilu uvijek vrše pomoću parova funkcija (`fscanf`, `fprintf`), (`fgets`, `fputs`) i (`fread`, `fwrite`).

Pomjeranje u fajlu

- Pored standardnog načina za pomjeranje pokazivača na fajl funkcije za upis i čitanje, postoje i druge funkcije koje vrše pomjeranje po fajlu bez upisa i čitanja.
- Prva od njih je

`rewind(a)`

koja "premotava" fajl `a` na početak, odnosno pokazivač `a` nakon ove naredbe pokazuje na početak fajla. Podsjeća na komandu za premotavanje trake na kasetofonu, a ima i slično porijeklo, jer je nekad najznačajniji memorijski medijum bila magnetna traka koja se s vremena na vrijeme morala premotati.

Pomjeranje u fajlu

- Druga značajna funkcija za pomjeranje u fajlu je `fseek(a,n,k)`. Argumenti funkcije su:
 - `a` pokazivač na fajl
 - `n` udaljenost od referentne pozicije (u bajtima)
 - `k` referentna pozicija
- Pretpostavljamo da su prva dva argumenta jasna. Treći argument definiše poziciju u odnosu na koju se pomjeramo. Obično se zadaje preko simboličke konstante koja može uzeti jednu od sljedeće tri vrijednosti:
 - `SEEK_SET` (ili numerički 0 → početak fajla)
 - `SEEK_CUR` (ili numerički 1 → tekuća pozicija)
 - `SEEK_END` (ili numerički 2 → kraj fajla).

Pomjeranje u fajlu

- Jasno je da se od kraja fajla može ići samo unazad (**n** je negativno), a od početka unaprijed, dok se iz tekuće pozicije, podrazumjevajući da je negdje u sredini fajla, možemo kretati i unaprijed i unazad.
- Funkcija **ftell(a)** vraća tekuću poziciju u fajlu na koju pokazuje **a**.
- Funkcija **feof(a)** provjerava da li smo u fajlu stigli do karaktera EOF (vraća **1** ako **jesmo** i **0** ako **nijesmo**).

Zatvaranje fajlova

- Da bi promjene koje su se vršile nad fajlovima bile pravilno ažurirane, fajlovi se moraju, prije završetka programa, **obavezno** zatvoriti.
- Fajl koji je otvoren preko pokazivača **a** može se zatvoriti funkcijom
`fclose(a);`
- Svi fajlovi koji su otvoreni u programu jednovremeno se zatvaraju funkcijom
`_fcloseall();`

Redirekcija

- Postoje situacije kada jedan dio rada sa fajlovima umjesto programa može vršiti operativni sistem. To se obavlja preko tzv. **redirekcije**.
- Posmatrajmo sljedeći elementarni program:

```
#include<stdio.h>
int main() {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d %d", a, b);
}
```

Redirekcija

- Program prima dva cjelobrojna argumenta i štampa ih. Pretpostavimo da se izvršna verzija programa zove `prog.exe`.
- Izvršavanje se obavlja sa komande linije naredbom:
`prog`
- Program sačeka da unesemo dva broja koja zatim odštampa na ekranu.
- Međutim, verzija naredbe:
`prog > izlaz.txt`
će sa komandne linije primiti dva broja, koja neće odštampati na ekranu.

Redirekcija

- Umjesto na ekran, rezultat je preusmjeren (redirektovan) u fajl `izlaz.txt`, a znak `>` obavještava operativni sistem da rezultate treba preusmjeriti sa ekrana (stand. izlaza) u fajl.
- Slično, naredba:
`prog < ulaz.txt`
će iz fajla `ulaz.txt` učitati podatke (neće ih tražiti sa tastature kao standardnog ulaza) i rezultat će odštampati na ekranu.
- Dozvoljene su i kombinacije sa preusmjeravanjem i ulaza i izlaza.

Ovo je samo ilustracija bez potrebe da joj bude posvećena značajna pažnja. U operativnim sistemima UNIX/Linux postoji mnogo složeniji sistem koji dozvoljava operacije tipa: sutra u 6 pokreni program i rezultat smjesti u fajl koji ćeš nakon dva sata poslati na e-mail.

Enumeracija

- Enumeracija (ili nabrojanje) je specijalni tip podatka u C-u koji predstavlja zamjenu za veći broj cjelobrojnih konstanti.
- Konstantama se dodjeljuju imena (tzv. **simboličke konstante**).
- Prije nego objasnimo detaljnije enumeraciju, ilustrujmo jednu njenu primjenu.
- Dani u nedjelji se u programu mogu pamtititi kao stringovi, ali ih je često mnogo bolje zapamtiti kao cijele brojeve. To onda znači da programer treba da zapamti preslikavanje dana u odgovarajući cijeli broj i da kroz program koristi cijeli broj gdje bi se inače pozvao na dan.

Enumeracija

- Čak i da programer ima takav kapacitet da pamti sva preslikavanja koja uvede u program, to ne znači da će to umjeti da prate drugi programeri koji budu nastavljali njegov rad. Stoga je mnogo bolje definisati odgovarajuće nabrojanje i uvesti simboličke konstante koje već po nazivu sugerišu o kom se danu (ili bilo kom drugom podatku) radi.
- Enumeracija se definiše najčešće globalno (van svih funkcija u programu). Primjer:

```
enum dani {PONED, UTOR, SRIJ, CETV, PETAK, SUBOTA, NEDJ};
```

uvedene simboličke konstante

naziv enumeracije, koji se u C-u može izostaviti

ključna riječ (mala slova)

Enumeracija

- Simboličke konstante uzimaju vrijednosti **PONED=0, UTOR=1, SRIJ=2, CETV=3, PETAK=4, SUBOTA=5, NEDJ=6.**
- Pokušaj promjene vrijednosti simboličke konstante dovodi do greške u programu i prekida izvršavanja.
- Nije potrebno simboličke konstante pisati velikim slovima, ali je to konvencija.

- Primjer:

```
enum {AAA=6, BBB=-1, CCC, DDD=3, EEE};
```

Konstante kojima su dodijeljene vrijednosti uzimaju te vrijednosti, dok one koje nijesu specificirane uzimaju vrijednost za 1 veću od prethodne. Ako prva nije definisana, njena vrijednost je 0.

Strukture

- Do sada smo se srijetali sa kolekcijama podataka istog tipa i te kolekcije smo nazivali nizovima (sa specijalnim slučajevima - matricama i stringovima).
- Modelovanje podatka iz realnog svijeta je često teško ostvarivo preko do sada naučenih tipova podataka, uključujući i nizove.
- Na primjer, RADNIK, MAŠINA, STUDENT, PROZOR se teško modeluju ijednim do sada uvedenim tipom podataka.
- Stoga su definisane **strukture** (alternativni nazivi su zapis ili slog), koje mogu da sadrže više podataka raznorodnog tipa.

Strukture

- Strukture se u programskom jeziku C uvode pomoću ključne riječi **struct**:

```
struct naziv_strukture {  
    deklaracija promjenljivih članica strukture  
} promjenljive_strukturnog_tipa;
```

- Primjer:

```
struct str {  
    char ch;  
    int i;  
} s1, s2;
```

Sada promjenljive **s1** i **s2** imaju tip strukture **str** i obe sadrže cijeli broj **i** i karakter **ch**.
Memorija se za jedan podatak strukturnog tipa zauzima redom (karakter, pa cijeli broj).

Strukture – definicija i deklaracija

- Struktura je vidljiva tamo gdje je definisana.
- Po pravilu se strukture definišu van svih funkcija (globalno), ali se istovremeno ne deklarišu (ako nema potrebe za globalnim strukturama) promjenljive odgovarajućeg tipa.
- Promjenljive strukturnog tipa se mogu deklarirati unutar funkcija (mogu biti i argumenti funkcija):

```
struct str {  
    char ch;  
    int i;  
};
```

Definicija
strukture

```
struct str s1, s2;
```

Deklaracija promjenljivih
strukturnog tipa

Strukture – memorija i inicijalizacija

- Sama struktura predstavlja mustru (šemu, šablon) kako će promjenljive strukturnog tipa biti smještene u memoriji računara. **Stoga, struktura ne zauzima memoriju!**
- Promjenljive strukturnog tipa zauzimaju memoriju koja je jednaka zbiru memorija potrebnih za smještanje promjenljivih članica strukture.
- Podaci strukturnog tipa se mogu inicijalizovati navođenjem odgovarajućih promjenljivih unutar vitičastih zagrada. Za strukturu iz prethodnog primjera, to se radi sa:

```
struct str s1 = {'a', 4}, s2 = {.ch = '#', .i = 345};
```

↑
bitan redosljed

Od C99, možemo koristiti označene inicijalizatore
(eng. *designated initializers*) – nije bitan redosljed

Strukture – pristupanje članovima

- Promjenljivim unutar strukturnog tipa se može pristupiti preko operatora tačka.

```
s1.ch = 'A';  
s1.i = 123;
```

- Unutar strukture se može nalaziti niz kao podatak član i tom nizu trebaju biti poznate sve dimenzije kako bi se mogla obaviti alokacija memorije:

```
struct Radnik {  
    char ImePrezime[40];  
    int GodineStaza;  
} rad = {"Mitar Pavlović", 4};
```

Strukture i nizovi

- Sada se elementima ovog niza može pristupiti kao:
`rad.ImePrezime[7] = 'a';`
Učitavanje imena i prezimena bi bilo:
`scanf("%d%s", &rad.GodineStaza, rad.ImePrezime);`
- Moguće je deklarirati niz struktura (tačnije je reći niz podataka strukturnog tipa):
`struct Radnik Pogon[40];`
- Elementima niza se pristupa na standardni način:
`Pogon[7].GodineStaza = 11;`
`strcpy(Pogon[3].ImePrezime, "Petar Pavlovic");`

Strukture i typedef

- Neki programeri ne vole da "vuku" ključnu riječ **struct** kroz program, već umjesto nje uvode "sopstveni tip podatka" preko **typedef**. Ovo se, po pravilu, vrši globalno. Na primjer:

```
typedef struct {  
    char ImPrez[40];  
    int godUpis, godStud;  
} Student;
```

- Deklaracija i upotreba promjenljivih ovog tipa se obavlja navođenjem samo novodefinisanog tipa, npr.:

```
Student DrugaGod[90];
```