Baze podataka 1

Vežbe – uvod u C

Sadržaj

- Osnovna struktura programa
- Rad sa strukturama
- Rad sa pokazivačima
- Dinamička alokacija memorije
- Rad sa tekstualnim datotekama
- Rad sa binarnim datotekama
- Najčešći uzroci *segmentation fault*-a

Osnovna struktura programa

Glavni elementi strukture programa

- Ulazna tačka programa - funkcija *main*
- Deklaracije funkcija
  - Moraju biti definisane pre *main* funkcije
  - Mogu biti izdvojene u zasebne datoteke zaglavlja (*header files*)
- Deklaracije globalno dostupnih tipova
  - Moraju biti definisane pre prvog korišćenja
  - Mogu biti izdvojene u zasebne datoteke zaglavlja
- Implementacije funkcija
  - Ako postoji deklaracija, mogu se naći u programu i nakon *main*-a
  - Mogu biti izdvojene u zasebne izvorne datoteke (*source files*)
- Pored ovoga, mogu se definisati i ostali standardni elementi:
  - Makro direktive, alijasi (*typedef*), konstante...
  - Takođe mogu biti definisani u okviru datoteka zaglavlja

Datoteke zaglavlja

- Imenovanje treba da bude dosledno da bi se lako moglo reći koje zaglavlje ide uz koji izvornu datoteku
- Koristiti "" umesto <> prilikom uvlačenja sopstvenih zaglavlja u `#include` direktivi
  - Navodnici govore kompajleru da potraži zaglavlja u direktorijumu programa
- Koristiti uslovnu zaštitu uključivanja u svakoj datoteci zaglavlja (`#ifndef`)
- Što se kompajlera tiče, ne postoji apsolutno nikakva razlika između datoteke zaglavlja i izvorne datoteke
  - I pored ovoga, ne bi trebalo raditi `#include` izvornih datoteka, već isključivo datoteka zaglavlja

Rad sa strukturama

Rad sa strukturama

- Struktura predstavlja skup promenljivih pod jednim imenom.
- Podrazumevano korišćenje - struct StructName variableName
  - Radi jednostavnijeg korišćenja, može se uvesti alijas
- Primer definicije uz alijasing:

```
struct structureName {
   dataType member1;
   dataType member2;
 ...
};

typedef struct structureName structureNameAlias;
```

Rad sa strukturama - Primer

```c
#include <stdio.h>

struct book{
    char title[10];
    char author[20];
    double price;
    int pages;
};

int main(){
    struct book book1 = {"Learn C", "Dennis Ritchie", 675.50, 325};

    printf("Title:  %s \n", book1.title);
    printf("Author: %s \n", book1.author);
    printf("Price:  %lf\n", book1.price);
    printf("Pages:  %d \n", book1.pages);
    printf("Size of book struct: %d", sizeof(struct book));
    return 0;
}
```

Zadatak 1

Sa standardnog ulaza učitati (statički) niz *n* (unosi se sa tastature) tačaka koordinatnog sistema, a zatim naći tačku koja se nalazi najbliže koordinatnom početku.

Svaka tačka je opisana strukturom tacka_st koja ima sledeća polja:

X osa (realan broj),
Y osa (realan broj).

Primer zaglavlja funkcije za računanje blizine koordinatnom početku:

```
struct tacka_st najbliza_t(struct tacka_st*, int);
```

Pri implementaciji razdvojiti fajlove zaglavlja, fajlove izvornog koda i fajl sa ulaznom tačkom programa.

Rad sa pokazivačima

Rad sa pokazivačima

Deklaracija:

```
data_type *ptr_name;
```

data_type - tip na koji pokazuje pokazivač, a ne tip pokazivača

ptr_name - naziv promenljive pokazivača

Dodeljivanje vrednosti pokazivaču:

```
int variable;
int *ptr1 = &variable;
int *ptr2 = 0x0215ff70 ;
int *ptr3 = NULL;  // Neophodno korišćenje stdlib biblioteke!
```

Referenciranje i dereferenciranje

- Referenciranje - koristi se unarni operator “&” koji daje adresu promenljive
  `ptr = &a;` - dodeljivanje vrednosti adrese promenljive a, pokazivačkoj promenljivoj
- Vrednost počinje sa "0x" i predstavlja celobrojnu vrednost memorijske lokacije u heksadecimalnom brojnom sistemu
- Dereferenciranje - koristi se unarni operator “*”, omogućuje posredan pristup podatku pomoću adrese u pokazivačkoj promenljivoj
- Pristup poljima struktura preko pokazivača - može da se koristi operator strelica operator –>

Česte greške pri radu sa pokazivačima

- Izostavljanje operatora "*" kada se deklariše više pokazivača u istoj deklaraciji

```
int* prt1, prt2;
```

- Korišćenje neinicijalizovanih pokazivača

```
int* prt1;
```

- Dodeljivanje pokazivača neinicijalizovanoj promenljivoj
- Dodeljivanje vrednosti promenljivoj pokazivača

```
int *prt = 10;
```

- Netačna sintaksa za povećanje dereferenciranih vrednosti pokazivača

```
int* ptr1;   int x = 100;        ptr1 = &x;       *ptr1++;
```

- Pokušaj promene adrese promenljive

```
&ptr1 = &ptr2;
```

Alokacija memorije (malloc, calloc, realloc)

Alokacija memorije (malloc, calloc, realloc)

- Dinamička alokacija memorije je dodela memorije za upotrebu u računarskom programu tokom čitavog vremena rada tog programa.
  Koristi se u slučaju da potrebna količina memorije koju program koristi nije unapred poznata.
- Upravljanje preko pokazivača.
- Dinamički dodeljenu memoriju treba eksplicitno osloboditi.

```
void *malloc(size_t size);
void *calloc(int n, size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
```

Sizeof i dinamičko zauzimanje memorije

- *Sizeof* je unarni operator koji se koristi za izračunavanje veličine tipova podataka u bajtovima.
- Može se primeniti na:
  - integrisane tip podataka,
  - korisnički-definisane tipove podataka.
- Prilikom dinamičkog zauzimanja memorije, neophodno izvršiti eksplicitnu konverziju onoga što funkcije vraćaju u pokazivačku promenljivu odgovarajućeg tipa
  - Najčešće tip koji odgovara tipu koji je korišćen unutar *sizeof* operatora

Alokacija memorije (malloc, calloc, realloc) - Primer

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

int main()
{
    // This pointer will hold the base address of the block created
    int* ptr;
    int n, i;
    // Get the number of elements for the array
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("Entered number of elements: %d\n", n);

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));

    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");
        return 0;
}
```

Greške pri alokaciji memorije

- Dereferenciranje NULL pokazivača
  - *malloc*() će vratiti NULL pokazivač ako nema raspoložive memorije, ili ako je program premašio dozvoljenu količinu memorije.
  - Povratne vrednosti uvek treba proveravati jer dereferenciranje NULL pokazivača može dovesti do greške u programu.
- Curenje memorije
  - Neophodan poziv funkcije *free*() kako bi se alocirana memorija oslobodila
  - Takođe, može biti uzrokovano gubitkom traga pokazivača
    - Na primer, prilikom korišćenja privremenog pokazivača za povratnu vrednost *realloc*()
      - Može dovesti do toga da originalni pokazivač bude pregažen sa NULL pokazivačem.
        - Nakon ovoga, ne postoji mogućnost da se pristupi dodeljenoj memoriji

Greške pri alokaciji memorije - Primeri

```
void *ptr;
size_t size = BUFSIZ;
ptr = malloc(size);
/* some further execution happens here... */
/* now the buffer size needs to be doubled */
size *= 2;
ptr = realloc(ptr, size);
    if (ptr == NULL)
        { /* ptr pointer lost !!! */
return 1;
```

```
//Use a pointer after using the free() function

int *ptr = malloc(sizeof(int));
free(ptr);
*ptr = 123; //undefined behavior
```

Zadatak 2

Izmeniti rešenje Zadatka 1 tako da se za čuvanje tačaka koristi dinamički niz.

Rad sa tekstualnim datotekama

Otvaranje i zatvaranje datoteke

- `FILE *fopen (const char *naziv, const char *rezim);`
  - naziv - naziv datoteke koja treba da bude otvorena u skladu sa pravilima OS
  - režim - oznaka načina korišćenja datoteke
  - povratna vrednost
    - pokazivač na otvorenu datoteku ili
    - NULL vrednost ako otvaranje nije uspešno izvršeno
      - obavljati proveru povratne vrednosti
- `int fclose(FILE *f);`
  - f - pokazivač na prethodno otvorenu datoteku koja treba da bude zatvorena
  - automatsko zatvaranje - kraj main funkcije

Preporučuje se eksplicitno zatvaranje svake otvorene datoteke

Mogući režimi rada - tekstualne datoteke

Oznaka	Opis
r	Čitanje postojeće datoteke.
r+	Čitanje i pisanje u postojećoj datoteci.
w	Pisanje u već postojećoj datoteci (prethodni sadržaj biva uništen), dok se automatski kreira nova datoteka ako ne postoji neka sa datim nazivom.
w+	Pisanje i čitanje u već postojećoj datoteci (prethodni sadržaj biva uništen), dok se automatski kreira nova datoteka ako ne postoji neka sa datim nazivom.
a	Dodavanja sadržaja na kraj postojeće datoteke; ako datoteka sa datim nazivom ne postoji, biće kreirana.
a+	Čitanje i dodavanja sadržaja na kraj postojeće datoteke; ako datoteka sa datim nazivom ne postoji, biće kreirana.

Čitanje i pisanje tekstualnih datoteka

- Čitanje i pisanje
  - obavlja se sekvencijalno (od početka datoteke)
  - svaki sledeći pristup (iza poslednjeg mesta kojem je pristupano u prethodnom čitanju ili pisanju)
- Rad sa znakovima - čitanje
  - `int fgetc(FILE *f);`
  - `char *fgets(char *tekst, int n, FILE *f);`

- Rad sa znakovima - pisanje
  - `int fputc(int c, FILE *f);`
  - `int fputs(const char *tekst, FILE *f);`

Čitanje i pisanje tekstualnih datoteka

- Rad sa znakovima - ulazna i izlazna konverzija
  - `int fscanf(FILE *f, const char *form, ...);`
  - `int fprintf(FILE *f, const char *form, ...);`
    - Parametri:
      - f - pokazivač na prethodno otvorenu datoteku
      - form - specifikacija konverzije
      - ... - adrese za preuzimanje/smeštanje sadržaja
    - Povratna vrednost
      - `fscanf`:
        - broj uspešno konvertovanih i smeštenih podataka ili konstanta EOF ako se pojavio kraj datoteke ili desila neka greška
      - `fprintf`:
        - broj ispisanih znakova ili negativna vrednost (-1) ako se desila neka greška pri konverziji

Rad sa tekstualnim datotekama - Primer

```
#include <stdio.h>

#define MAX 20

struct student {
    char br_indeksa[10];
    char ime[20];
    char prezime[20];
};

int main() {
    struct student studenti[MAX];
    int n = 0;
    FILE *f = fopen("studenti.txt", "r");
    while (n < MAX && fscanf(f, "%9s %19s %19s", studenti[n].br_indeksa, studenti[n].ime, studenti[n].prezime) == 3) {
        n++;
    }
    for (int i = 0; i < n; i++) {
        printf("Indeks: %s, Ime: %s, Prezime: %s\n", studenti[i].br_indeksa, studenti[i].ime, studenti[i].prezime);
    }

    fclose(f);
    return 0;
}
```

Zadatak 3

- Fudbalski klub Red Bul Lajpcig (RBL) uveo je novi informacioni sistem i elektronsko praćenje poseta navijača utakmicama koje se igraju na njihovom stadionu.
- U okviru informacionog sistema RBL-a, svi podaci o utakmicama i navijačima čuvaju se u CSV datotekama, gde je znak razdvajanja - razmak.
- U CSV datoteku se upisuju sledeći podaci o ulascima (za sve utakmice evidencija se vodi u jednoj CSV datoteci):
  - redni broj utakmice,
  - datum i vreme početka utakmice (string u formatu DD/MM/YYYY_HH:mm),
  - ime gostujućeg tima (razmaci zamenjeni donjom crtom),
  - tip ulaznice (O - obična, S - sezonska, P - sa popustom za penzionere i studente),
  - datum i vreme ulaska navijača sa datom ulaznicom (tekst u formatu DD/MM/YYYY_HH:mm).

Zadatak 3

- Vas su unajmili da napišete C program koji omogućava kreiranje datoteke izveštaja za utakmice na kojima su domaćini.
- U pripremljenoj datoteci izveštaja, za svaki tip ulaznice prikazano je sledeće:
  - ukupan broj evidentiranih ulazaka za dati tip ulaznice,
  - ukupna zarada za dati tip ulaznice (O – 50e, S – 30e, P – 25e),
  - lista ulazaka za dati tip ulaznice,
    - Dodatno: lista uzlazno sortirana po nazivu protivničkog tima
- Napomena: sami odredite strukturu datoteke izveštaja

Rad sa binarnim datotekama

Otvaranje i zatvaranje binarne datoteke

- `FILE *fopen (const char *naziv, const char *rezim);`
  - naziv - naziv datoteke koja treba da bude otvorena u skladu sa pravilima OS
  - režim - oznaka načina korišćenja datoteke
  - povratna vrednost
    - pokazivač na otvorenu datoteku ili
    - NULL vrednost ako otvaranje nije uspešno izvršeno
      - obavljati proveru povratne vrednosti
- `int fclose(FILE *f);`
  - f - pokazivač na prethodno otvorenu datoteku koja treba da bude zatvorena
  - automatsko zatvaranje - kraj main funkcije

Preporučuje se eksplicitno zatvaranje svake otvorene datoteke

Mogući režimi rada - tekstualne datoteke

Oznaka	Opis
rb	Čitanje postojeće binarne datoteke.
rb+	Čitanje i pisanje u postojećoj binarnoj datoteci.
wb	Pisanje u već postojećoj binarnoj datoteci (prethodni sadržaj biva uništen), dok se automatski kreira nova datoteka ako ne postoji neka sa datim nazivom.
wb+	Pisanje i čitanje u već postojećoj binarnoj datoteci (prethodni sadržaj biva uništen), dok se automatski kreira nova datoteka ako ne postoji neka sa datim nazivom.
ab	Dodavanja sadržaja na kraj postojeće binarne datoteke; ako datoteka sa datim nazivom ne postoji, biće kreirana.
ab+	Čitanje i dodavanja sadržaja na kraj postojeće binarne datoteke; ako datoteka sa datim nazivom ne postoji, biće kreirana.

Rad sa podacima - ulaz

- `int fread(void *sadrzaj, int duz, int n, FILE *f);`
  - učitavanje zadatog broja podataka u memoriju iz datoteke od prethodne pozicije
  - nova pozicija u datoteci na mestu iza poslednjeg učitanog bajta
    - *sadrzaj* - memorijska adresa od koje počinje smeštanje učitanih podataka
    - *duz* - dužina pojedinačnog podatka za učitavanje (u bajtovima)
    - *n* - broj podataka koji treba učitati iz datoteke
    - *f* - pokazivač na prethodno otvorenu datoteku

  - povratna vrednost - broj uspešno pročitanih podataka
  - indikator greške učitavanja - *feof(), ferror()*

Rad sa podacima - izlaz

- `int fwrite(const void *sadrzaj, int duz, int n, FILE *f);`
  - upisivanje zadatog broja podataka od mesta završetka poslednjeg pristupa
  - ako pozicija nije kraj datoteke, dolazi do prepisivanja sadržaja
  - ako kapacitet datoteke nije dovoljan, datoteka se povećava
  - nova pozicija u datoteci na mestu iza poslednjeg upisanog bajta
    - *sadrzaj* - memorijska adresa od koje počinje čitanje podataka
    - *duz* - dužina pojedinačnog podatka za smeštanje (u bajtovima)
    - *n* - broj podataka koji treba smestiti u datoteku
    - *f* - pokazivač na prethodno otvorenu datoteku

  - povratna vrednost - broj uspešno upisanih podataka,
  - ako je došlo greške onda je ta vrednost manja od n

Rad sa podacima - pozicioniranje

- `int fseek(FILE *f, long offset, int ref);`
  - postavljanje trenutne pozicije unutar datoteke
  - može i za tekstualne datoteke
  - neophodna kada posle radnje čitanja dolazi pisanje (i obrnuto)
    - *f* - pokazivač na prethodno otvorenu datoteku
    - *offset* - udaljenost nove pozicije od referentne tačke (u bajtovima)
    - *ref* - referentna tačka u odnosu na koju se posmatra udaljenost nove pozicije
      - *SEEK_SET* - početak datoteke
      - *SEEK_CUR* - trenutna pozicija unutar datoteke
      - *SEEK_END* - kraj datoteke
  - povratna vrednost - oznaka greške - ako vrednost različita od 0

Rad sa podacima - pozicioniranje

- `long ftell(FILE *f);`
  - nalaženje trenutne pozicije unutar datoteke
  - *f* - pokazivač na prethodno otvorenu datoteku
  - povratna vrednost
    - trenutna pozicija u bajtovima u odnosu na početak datoteke
    -1L - oznaka greške
- `void rewind(FILE *f);`
  - postavljanje pozicije na početak datoteke
  - *f* - pokazivač na prethodno otvorenu datoteku
  - nema povratnu vrednost

Indikacija greške

- Vrednosti indikatora se postavljaju kroz bočne efekte mnogih prethodnih funkcija
- Dva indikatora
    - indikator kraja datoteke
    - indikator greške
- Funkcije
    - `void clearerr(FILE *f);`
        - briše vrednosti oba indikatora
    - `int feof(FILE *f);`
        - provera indikatora kraja datoteke
    - `int ferror(FILE *f);`
        - provera indikatora greške za datoteku
- Globalna promeniljiva
    - *errno* u *<errho.h>* - kod greške

Rad sa binarnim datotekama - Primer

```
#include <stdio.h>

#define MAX 20

struct student {
    char br_indeksa[10];
    char ime[20];
    char prezime[20];
};

int main() {
    struct student studenti[MAX];
    int n = 0;
    /*Prethodni primer - ucitavanje iz txt fajla*/
    FILE *fb = fopen("studenti.bin", "wb");
    for (int i = 0; i < n; i++) {
        fwrite(&studenti[i], sizeof(struct student), 1, fb);
    }
    fseek(fb, 0, SEEK_SET);
    struct student s;
    while (fread(&s, sizeof(struct student), 1, fb) == 1) {
        printf("Indeks: %s, Ime: %s, Prezime: %s\n", s.br_indeksa, s.ime, s.prezime);
    }
    fclose(fb);
    return 0;
}
```

Zadatak 4

- Fudbalski klub RBL vas je unajmio kako biste implementirali definisane operacije nad podacima o poseti navijača utakmicama koje se igraju na njihovom stadionu.
- Zbog potencijalno velike količine podataka kojom je potrebno rukovati, podatke ne bi trebalo da uvlačite u radnu memoriju radi obrade, već je neophodno da ih čuvate u binarnoj datoteci i da definisane operacije vršite direktno nad tom datotekom.
- Podaci o ulascima imaju sledeću strukturu:
  - redni broj ulaznice
  - redni broj utakmice,
  - datum i vreme početka utakmice (string u formatu DD/MM/YYYY_HH:mm),
  - ime gostujućeg tima (razmaci zamenjeni donjom crtom),
  - tip ulaznice (O - obična, S - sezonska, P - sa popustom za penzionere i studente),
  - datum i vreme ulaska navijača sa datom ulaznicom (tekst u formatu DD/MM/YYYY_HH:mm).

Zadatak 4

- Omogućiti sledeće operacije:
  - Kreiranje nove binarne datoteke sa zadatim nazivom
    - Dodati slog koji će kao vrednost za polje *rbr_ulaznice* imati -1 - indikator kraja datoteke
  - Prenos podataka iz date *BP1_03_Uvod_u_C_Z4_RBL_UlasciNaStadion.csv* datoteke
    - Prilikom prenosa, obezbediti sledeće:
      - Svaki pročitani red CSV datoteke će se dodati na kraj binarne datoteke
      - Jedinstvenost polja *rbr_ulaznice* (duplikati mogu nastati višestrukim očitavanjem ulaznice)
  - Prikaz celokupnog sadržaja binarne datoteke
  - Prikaz podataka o ulasku sa zadatom vrednošću polja rbr_ulaznice

Zadatak 5

- Kao proširenje rešenja Zadatka 4, omogućiti sledeće operacije:
  - Ručni unos novog ulaska na stadion, ručnim unosom podataka u realnom vremenu
    - Obezbediti jedinstvenost polja rbr_ulaznice
  - Izmenu vrednosti polja tip_ulaznice za ulazak sa zadatom vrednošću polja rbr_ulaznice
  - Logičko brisanje podatka sa zadatom vrednošću polja rbr_ulaznice
    - U ovu svrhu, u strukturu dodati polje podatak_aktivan, sa mogućim vrednostima 0 i 1

Zadatak 6

- Kao proširenje rešenja Zadatka 5, omogućiti sledeće operacije:
  - Fizičko brisanje podatka sa zadatom vrednošću pola rbr_ulaznice
    - Vrši se po principu pomeranja svih slogova koji su mu naredni za jedno mesto u levo
    - Dodatno: osloboditi memoriju koja je ostala upražnjena nakon pomeranja slogova

Najčešći uzroci segmentation fault-a

Segmentation fault

- Segmentation fault, je kritična greška koja se javlja kada program pokuša da pristupi memorijskoj lokaciji kojoj nije dozvoljen pristup. To je čest problem u C programskom jeziku i može biti nezgodan grešaka za otklanjanje.
- Razumevanje segmentation fault greške i načina njenog sprečavanja je od suštinskog značaja za pisanje stabilnog i pouzdanog C koda.

Razlozi zbog kojih nastaje segmentation fault

- **Dereferenciranje NULL pokazivača**

  NULL pokazivač ima rezervisanu vrednost koja se naziva konstantom NULL pokazivača koja služi za indikaciju da pokazivač ne pokazuje ni na jedan važeći objekat ili funkciju.

  Kada je NULL pokazivač dereferenciran, program pokušava da pristupi memorijskoj lokaciji koja ne postoji. Ovo može uzrokovati grešku segmentation fault.

```c
#include<stdio.h>

int main() {
    int *ptr = NULL;
    // Ovo će uzrokovati segmenatition fault jer je ptr null
    *ptr = 100;
    return 0;
}
```

Razlozi zbog kojih nastaje segmentation fault

- **Pristup memoriji van granica**

  Pristup memoriji izvan dodeljenih granica niza može dovesti do segmentation fault. Neophodno je potvrditi indekse niza i osigurati da ostanu unutar važećeg opsega.

```c
#include<stdio.h>

    int main() {
    int niz[10];
    int i;
        for (i = 0; i < 11; i++) {   // Ovo će uzrokovati segmenatition fault
                                     //jer brojač i je veći od veličine niza
            niz[i]= i;
        }
    return 0;
}
```

Razlozi zbog kojih nastaje segmentation fault

- **Pristup neinicijalizovanom ili oslobođenom memorijskom bloku**

  Pojavljuje se kada se pokazivač koristi za pristup memoriji koja nije inicijalizovana ili koja je prethodno oslobođena.

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

int main() {
    int *ptr;
    int value;  // Pristupanje neinicijalizovanoj memoriji
    printf("Vednost pre inicijalizacije: %d\n", *ptr);
    return 0;
}
```

Razlozi zbog kojih nastaje segmentation fault

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

int main() {
    int *ptr;
    int value;

    ptr = (int *)malloc(sizeof(int));
        if(ptr != NULL){
            *ptr = 10;
        printf("Value after initialization: %d\n", *ptr);

        free(ptr);
        printf("Value after greeing memory: %d\n", *ptr);
        }
    return 0;
}
```

Razlozi zbog kojih nastaje segmentation fault

- **“Stack overflow”**

  Ako stek poziva programa postane prevelik zbog velikog broja ugnežđenih poziva funkcija ili rekurzivnih poziva, to može izazvati Stack overflow i rezultirati greškom segmentation fault.

```c
#include<stdio.h>

void recursiveF(){
    int array[1024];
    recursiveF();
    printf("This line will not be reached due to stack overflow");
}
int main(){

    recursiveF();

    return 0;
}
```

Razlozi zbog kojih nastaje segmentation fault

- **“Buffer overflows”**

  Pisanje izvan granica dodeljenog memorijskog bloka može dovesti do se segmentation fault.

```c
#include<stdio.h>
#include<string.h>

void foo(char* input){
    char buffer[8];
    strcpy(buffer, input); // Potential buffer overflow
    printf("Buffer contents: %s\n ", buffer);
}
int main(){
    char input[12] = "Hello, World!";
    foo(input);
    return 0;
}
```

Kraj!

Hvala na pažnji!