# Programiranje I

Nizovi (vektori)

Stringovi

Funkcije

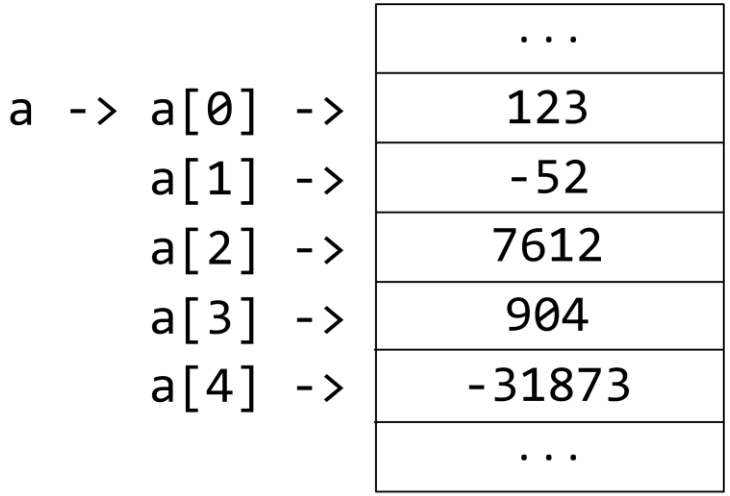
# Nizovi (vektori)

- Deklaracija niza se obavlja sa

  tip a[5];

  gdje je tip neki od C tipova (float, int, double, itd.).

- Na ovaj način je zauzeta memorija za 5 promjenljivih, redom – jedna za drugom.




- Te promjenljive su a[0], a[1], a[2], a[3] i a[4] (obratite pažnju na indekse) i sa njima možemo vršiti sve operacije dozvoljene kod datog tipa, npr. a[0] += a[3] - a[4].

# Nizovi

- Adrese članova niza su: &a[0], &a[1], &a[2], &a[3] i &a[4], ali postoji i alternativni oblik a, a+1, a+2, a+3 i a+4.
- Ime niza ima vrijednost početne adrese niza (adrese prvog člana niza, tačnije prvog bajta prvog člana niza).
- a+1 nije adresa narednog bajta u memoriji, već adresa narednog člana niza.
- a+k predstavlja pomjeraj za k*sizeof(tip) bajta u memoriji.
- Ovo omogućava pristup članovima niza i sa ***(a+1) = -3**, čime bi drugi član niza (onaj sa indeksom 1) postao -3.
- Uočite da ***a+1** ne znači **a[1]**, već **a[0]+1**, zbog većeg prioriteta * u odnosu na +.

# Nizovi i pokazivači

- Veliki broj obrada nizova se sastoji u obilasku svih članova uz njihovo korišćenje ili mijenjanje po nekom kriterijumu.
- Kod nizova, puni smisao ima korišćenje pokazivača i primjena raznih operacija sa njima. Na primjer:

```
tip *b;
b = a + 3;
```

- Sada b pokazuje na član a[3], i razlika b-a iznosi 3 (ne broj bajtova, već broj elemenata datog tipa između članova određenih pokazivačima a i b).

# Višedimenzioni nizovi

- Deklaracija 2D niza se obavlja kao:

  float m[4][5];

  gdje m predstavlja niz od 4 elementa, od kojih svaki predstavlja niz od 5 elemenata tipa float.
- Dakle, u C-u: **matrica = niz nizova**.
- Elementi niza **m** su: **m[0]**, **m[1]**, **m[2]** i **m[3]**, gdje je **m[0]** adresa nulte vrste matrice, **m[1]** adresa prve vrste ...
- Mogući broj dimenzija višedimenzionog niza zavisi od kompajlera.
- Ako se deklaracije nizova i matrica obogate pokazivačima može da nastane prava zbrka vezana za to koji je zapravo tip podataka deklarisan.

# Indeksiranje članova niza

- Ako imamo deklarisan niz:

  int a[7];

  a u programu koristimo element a[10], neće doći do prekida rada programa, a vjerovatno ni do bilo kakvog upozorenja, što znači da se nesmetano može pristupiti memorijskim lokacijama van niza, tj. gdje su neke druge promjenljive ili dio sistemskog koda. Čak se mogu koristiti i negativni indeksi, npr. a[-5].

- Nije potrebno naglašavati da se ove situacije moraju izbjeći.

# Inicijalizacija nizova

- Nizovi se mogu inicijalizovati zajedno sa deklarisanjem (definisati) kao:

  int a[5] = {0, 1, 2, 3, 4};

  čime smo članove niza redom postavili na vrijednosti navedene u vitičastim zagradama. Dozvoljena je varijanta:

  int a[5] = {0, 1, 2};

  ali se sada eksplicitno neinicijalizovane vrijednosti (a[3] i a[4]) niza postavljaju na 0.

- Varijanta

  int a[5] = {0};

  postavlja sve članove niza na 0.

# Inicijalizacija nizova

- Dozvoljena je i sljedeća inicijalizacija:
  `int a[] = {0, 1, 2, 3, 4};`
  čime se implicitno zauzima 5 pozicija za cijele brojeve.
- Kod inicijalizacije
  `int a[2] = {0, 1, 2, 3, 4};`
  samo se prva dva člana niza inicijalizuju, ostali brojevi unutar zagrada se zanemaruju.
- Kod višedimenzionih nizova inicijalizacija se može obaviti sa:
  `int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};`
  `int a[ ][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};`
- Prvi metod je, nadamo se, jasan, dok smo drugim metodom implicitno zauzeli potreban broj vrsta.

# Inicijalizacija nizova

- Dozvoljena je i inicijalizacija tipa:
  int a[ ][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  jer će se podaci sada smiještati vrstu po vrstu, a računar ima podatak koliko je elemenata u svakoj vrsti.
- Od standarda C99, koristi se označeni inicijalizator:
  int x[7] = {[2]=9, [4]=12}; // ostali 0
  int x[] = {[2]=9, [4]=12}; // maksimalni index je 4
- Nije dozvoljena inicijalizacija oblika:
  int a[ ][ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  jer ne postoji podatak kako smjestiti elemente iz vitičastih zagrada u matricu (tj. koliko elemenata ide u vrste).

# Stringovi

- Niz karaktera (podataka tipa `char`) se naziva **string**.
- Sa stringovima smo se već upoznali kroz funkcije printf:

```
printf("Suma je %d", suma);
```

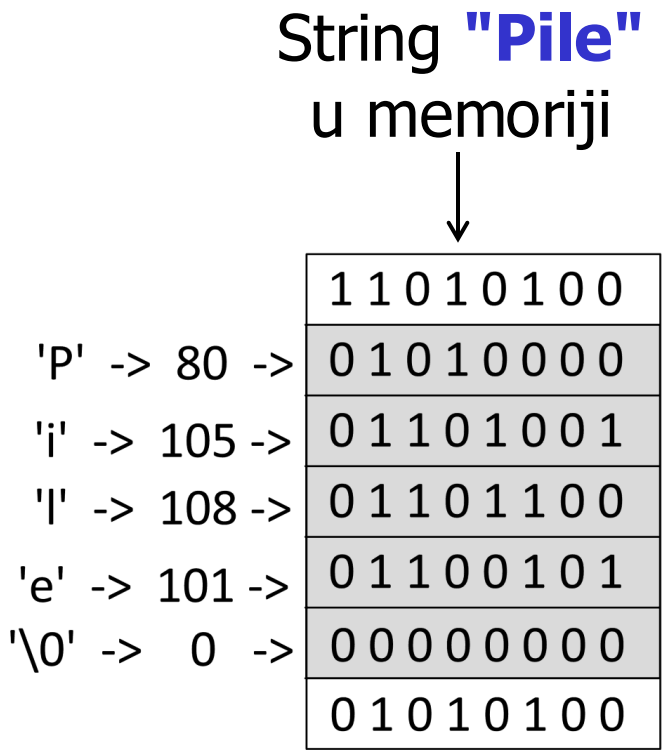
String

- String je niz karaktera pod znacima navoda, npr. "Program".
- Za razliku od stringa, pojedinačni karakteri se navode koristeći apostrofe: 'P', 'r', ... , 'a', 'm'.
- String se deklariše kao svaki drugi niz, koristeći uglaste zagrade [ ] i broj elemenata unutar zagrada:

```
char s[30];
```

# Terminacioni karakter

- Pored pojedinačnih karaktera, stringovi u C-u sadrže dodatni karakter **'\0'**. Ovaj karakter se nalazi na kraju stringa i naziva se **terminacioni karakter**.
- Na osnovu terminacionog karaktera, kompajler zna gdje se završava string u memoriji.
- Terminacioni karakter se automatski dodaje na kraj stringa prilikom njegovog učitavanja, bez uticaja programera.
- Terminacioni karakter ima ASCII kod 0.



# strlen i sizeof

- Prije nego pređemo na stringove, objasnimo jedan detalj koji umije da zbuni.
- Funkcija koja se često koristi u radu sa stringovima **strlen("program")**, definisana u biblioteci string.h, daje rezultat **7**, odnosno ona ne broji terminacioni karakter. Sa druge strane, **sizeof** mjeri memoriju potrebnu za smještaj stringa. U ovom slučaju, **sizeof("program")** vraća broj **8**.
- Da bismo koristili funkciju strlen, kao i ostale funkcije za rad sa stringovima, u program se mora uključiti biblioteka string.h.

# Deklaracija i inicijalizacija stringa

- String se može deklarisati kao i svaki niz na sljedeći način:
  ```
  char str[20];
  ```
- Inicijalizacija stringa se može obaviti na sljedeće načine:
  ```
  char str[20] = "cert";
  char str[] = "cert";
  char str[] = {'c', 'e', 'r', 't', '\0'};
  ```
- Koje vrijednosti će vratiti sizeof(str) i strlen(str)?
- Dozvoljena je i sljedeća inicijalizacija (spajanje stringova, pri čemu se stringovi razdvajaju bjelinom):
  ```
  char s[50] = "Dobar" "dan, " "kolega";
  ```

# Učitavanje i štampanje stringova

- Funkcijom scanf se može učitati cio string (ne kao kada su u pitanju numerički nizovi - član po član). Sintaksa je:

  scanf("%s", str); ⟶ String se smješta karakter po karakter, od adrese str nadalje.

- Funkcijom scanf string se učitava samo do prve bjeline tako da ova funkcija ne može služiti npr. za učitavanje imena i prezimena odjednom.
- Funkcijom printf štampa se string, pri čemu se u okviru stringa mogu štampati i bjeline, uključujući i znak za novi red:

  printf("%s", str); ⟶ Kao kod scanf, navodi se ime stringa.

- U stdio.h, postoje i sljedeće funkcije:
  - **gets(str)** učitava string do znaka za novi red (uključuje spejsove i tabove),
  - **puts(str)** štampa string i automatski nakon štampanja prelazi u novi red.

# Korisne funkcije iz string.h

- strcpy(tar, or) kopira string or (od origin) u string tar (od target).
- strcat(tar, or) nadovezuje string or na kraj stringa tar.
- strcmp(a, b) leksikografski poredi stringove a i b.
  - ako je string a veći od stringa b vraća pozitivan broj;
  - ako su stringovi jednaki vraća nulu;
  - ako je string a manji od stringa b vraća negativan broj.
  - String a je leksikografski veći od stringa b ako je:
    - ASCII kod prvog karaktera stringa **a** veći od ASCII koda prvog karaktera stringa b;
    - ukoliko su prvi karakteri isti porede se naredni.
    - Stringovi su jednaki ako su im svi karakteri jednaki.
  - ASCII kod terminacionog karaktera je manji od bilo kog drugog karaktera i u C-u je jednak 0.

# Korisne funkcije iz string.h

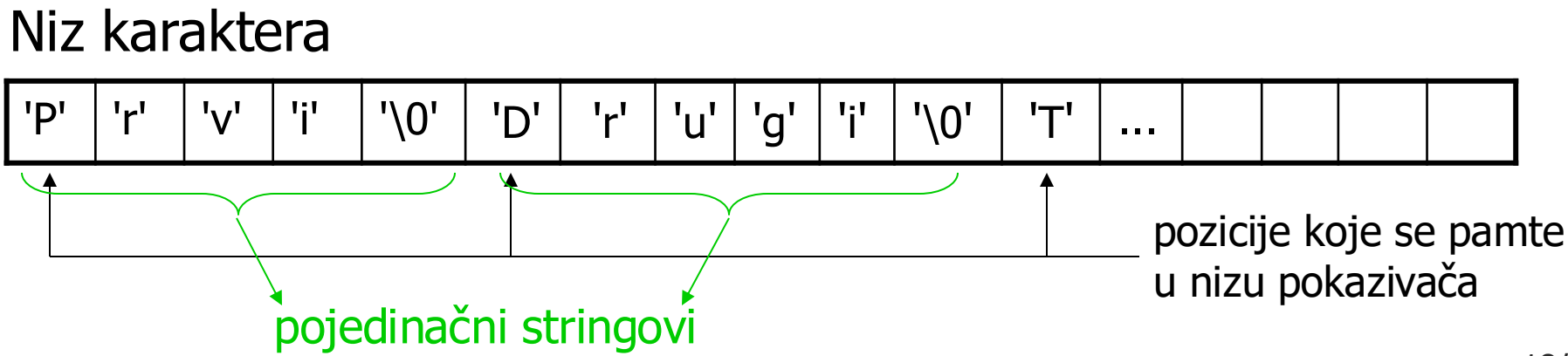
- strstr(tar, or) traži podstring or u stringu tar i vraća pokazivač na prvo pojavljivanje takvog podstringa.
- strchr(tar, c) i strrchr (tar, c) traže prvo i posljednje pojavljivanje karaktera c u stringu tar, respektivno.
- Funkcije atoi(s), atol(s) i atof(s) konvertuju broj sadržan u stringu s u int, long int i double broj, respektivno.
- Funkcije itoa(N, s, B) i ltoa(N, s, B) konvertuju int N i long N u string s u brojnom sistemu sa osnovom B (zaglavlje stdlib).

# Rad sa mnoštvom stringova

- Aplikacije koje rade sa stringovima obično rade sa velikim brojem stringova.
- Svaki string se mora dimenzionisati na najveću očekivanu dužinu.
- Na primjer, baza prezimena se mora dimenzionisati na najveće očekivano prezime od, recimo, 15 slova, dok većina prezimena kod nas ima manje od 10.
- Gora varijanta je za riječi opšte namjene, gdje se dimenzionisanje mora obaviti sa 20 i više slova, dok mnogo riječi, npr. veznici, imaju samo po nekoliko slova.

# Rad sa mnoštvom stringova

- Da bi se izbjegla ova nepotrebna memorijska zahtjevnost, uobičajeno se koristi sljedeći trik.
- U jednom nizu karaktera (namjerno sada koristimo ovaj pojam) se čuvaju svi stringovi razdvojeni terminacionim karakterima. Pozicije početaka pojedinih stringova se pamte preko pokazivača.



# Rad sa mnoštvom stringova

- Realizacija načina rada sa mnoštvom stringova nije stvar kompajlera, već programerske vještine.

```
#include<stdio.h>
int main(){
char s[1000], *p[200], temp[20];
int size = 1, tp = 0, i, j = 0;
p[0] = s;
while(size) {
    gets(temp);
    size = strlen(temp);
    for(i=0; i<size; i++) s[tp+i] = temp[i];
    s[tp+size] = '\0';
    tp += size+1;
    if(size) p[++j] = s+tp;
}
for(i=0; i<j; i++)
    printf("%s\n", p[i]);
}
```

nastavak gore

# Rad sa mnoštvom stringova

- Objasnimo prethodni primjer.
- Deklarisali smo string i niz pokazivača na stringove.
- U while petlji ostajemo sve dok je učitani string dužine veće od nula, odnosno dok korisnik ne unese prazan string.
- Svaki uneseni string se pozicionira na odgovarajuće mjesto, uz ažuriranje promjenljive tp, koja pamti poziciju dokle se stiglo u "velikom stringu".
- Nakon unosa stringa postavlja se terminacioni karakter na odgovarajuće mjesto i podesi da odgovarajući pokazivač iz niza pokazivača pokaže na naredni string čiji se unos očekuje.
- Na kraju ovog demonstracionog programa smo iz "velikog stringa" štampali pojedinačne stringove.
- Program nije optimalan ni direktno praktično upotrebljiv, već je samo ilustrativni primjer.

# String literali

- Pokazivač na tip char se može inicijalizovati na sljedeći način:

  char *p = "Test";

- Na ovaj način je deklarisan pokazivač na read-only memoriju koja sadrži string literal "Test".

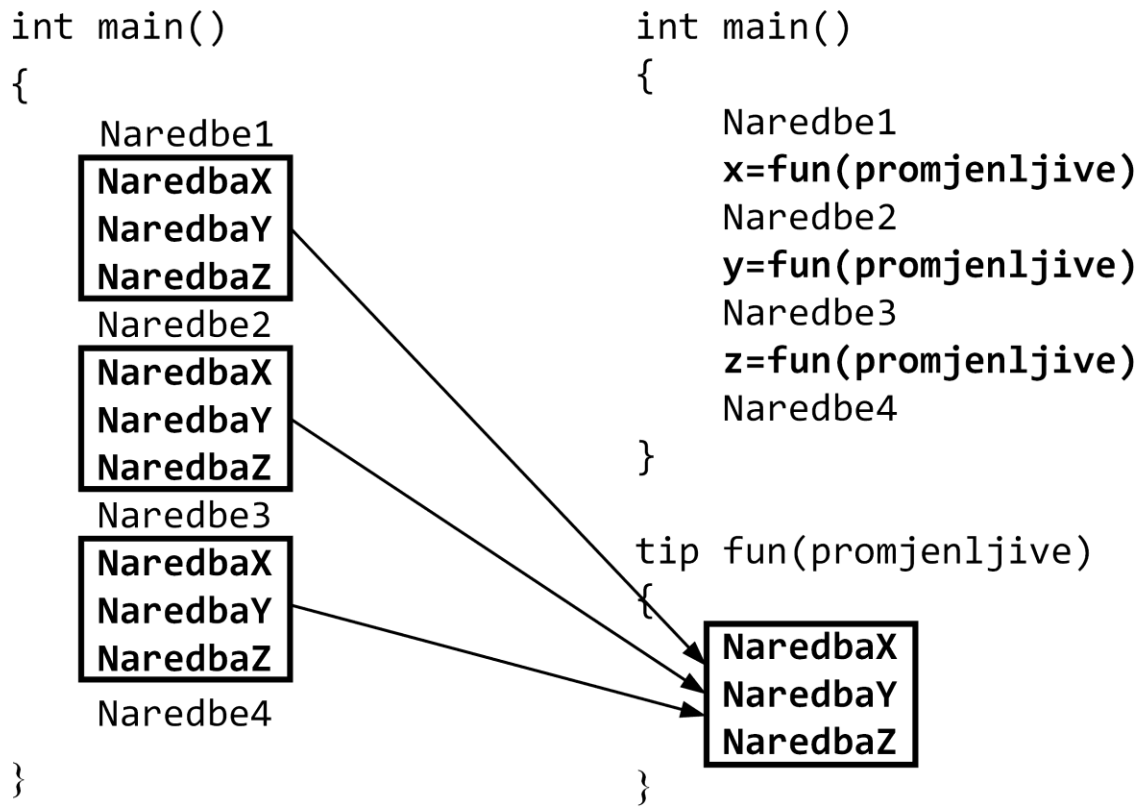
- Pokušaj izmjene ovog stringa, na primjer

  p[0] = 'B';

  dovodi do greške prilikom kompajliranja.

# Funkcije – Osnovno

Kada se u programu ponavlja više linija koda, dobra praksa je grupisati te linije u obliku funkcije, čime se pojednostavljuje održavanje i modifikacija programa.



# Funkcije – Osnovno

- Funkcija predstavlja programsku cjelinu koja izvršava određeni zadatak.
- Pomoću funkcija se složeni programski zadaci dijele na jednostavnije cjeline. Time se postiže veća jasnoća programa i olakšava se njegova modifikacija i održavanje.
- Dobro osmišljena funkcija obavlja jedan jasno definisan zadatak.
- Korisnik dobro osmišljene funkcije ne mora poznavati detalje njene implementacije da bi je koristio.
- Funkcija može da ima ulazne podatke, i može da vrati rezultat svog izvršavanja onoj funkciji koja ju je pozvala.

# Definicija funkcije

- Definicija funkcija ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1, ..., tip_n arg_n) {
        tijelo funkcije
}
```

gdje:

  - tip_podatka predstavlja tip podatka koji će funkcija vratiti kao rezultat svog izvršavanja;
  - ime_funkcije mora biti pravilan identifikator;
  - tip_1 arg_1, ... , tip_n arg_n je lista ulaznih parametara funkcije. Deklaracije pojedinih parametara se odvajaju zarezima.

- Unutar vitičastih zagrada se navodi tijelo funkcije koje se sastoji od deklaracije promjenljivih i izvršnih naredbi, isto kao kod funkcije main.

# Vraćanje rezultata funkcije

- Funkcija može da vrati rezultat svog izvršavanja, što se postiže pomoću ključne riječi **return** na sljedeći način:

  return izraz;

  gdje izraz može biti proizvoljan izraz (konstanta, promjenljiva, matematički izraz, poziv druge funkcije).
- Tip izraza treba da odgovara tipu podatka koji funkcija vraća.
- Naredba return predstavlja tačku izlaska iz funkcije, tj. ako ima naredbi nakon return, one se neće izvršiti!
- Ukoliko funkcija ne vraća rezultat, ona se deklariše kao **void**:

```
void ime_funkcije(tip_1 arg_1, ..., tip_n arg_n) {
        tijelo funkcije
}
```

# Primjer tri funkcije

```
int zbir1(int x, int y)
{
        return x+y;
}
```

```
void zbir2(int x, int y)
{
        printf("%d", x+y);
}
```

```
int zbir3(int x, int y)
{
        printf("%d", x+y);
        return 1;
}
```

Crvenom bojom su označena **zaglavlja funkcija**, gdje se, pored imena funkcije, navodi tip njenog rezultata, a u zagradi se definišu parametri funkcije.

Ove tri funkcije su slične, ali rade tri različite stvari:

- Prva funkcija vraća rezultat koji je suma parametara;
- Druga funkcija ne vraća rezultat, već samo štampa zbir;
- Treća funkcija štampa zbir i vraća rezultat 1 koji može predstavljati indikaciju da je operacija obavljena uspješno.

# Primjeri – Napomene

- Rezultati funkcija koje nijesu void mogu da se pridruže dozvoljenoj lijevoj strani izraza, ali i ne moraju:

```
c = zbir1(3,4);
zbir3(c,12);
```

- U drugom slučaju funkcija će vratiti vrijednost, ali pošto nema lijeve strane izraza, privremena promjenljiva u kojoj je sačuvana vrijednost rezultata će biti dealocirana.
- Iz glavnog programa se funkcije tipa void pozivaju bez navođenja promjenljivih na lijevoj strani znaka jednakosti:

```
zbir2(5,c);
```