# Programski jezik I

Pokazivači

Nizovi (vektori)

Stringovi (uvod)

# Pokazivači

- Prvi složeni tip podataka kojeg uvodimo je pokazivač (eng. pointer). U pitanju je moćan, ali često i nerazumljiv koncept.
- Nemoguće je izbjeći rad pokazivačima u programskom jeziku C, kao ni u aplikacijama koje rade sa hardverskim uređajima.
- Pokazivač se deklariše sa int *p;
- Zvjezdica ispred imena promjenljive p označava da je u pitanju pokazivač, u ovom slučaju pokazivač na cijeli broj.
- Pokazivač predstavlja **adresu** (prvog bajta) promjenljive u memoriji računara.
- Na primjer, nakon deklaracije
  int x, *p;
  naredbom p=&x; u p upisujemo adresu promjenljive x.
  Operacija uzimanja adrese se naziva referenciranje.

# Pokazivači

- Preko pokazivača **p** se sada može pristupiti podatku koji se nalazi na adresi **p** (ova operacija se zove dereferenciranje):
  ***p = 20;** ⟶ uočite da je * sada unarni operator
  Upiši broj **20** na adresu **p**, u ovom slučaju u promjenljivu **x**.
- Koncept vam vjerovatno još uvijek djeluje apstraktno, ali sa njegovom neophodnošću ćemo se upoznati kasnije, posebno kad budemo radili nizove i funkcije.
- Tokom rada pokazivač može da pokaže na drugu, proizvoljnu, promjenljivu.
- Dozvoljene operacije sa pokazivačima su: poređenje pokazivača, sabiranje sa konstantom, inkrementiranje i dekrementiranje pokazivača.

# Inicijalizacija pokazivača

- Pokazivačke promjenljive se mogu inicijalizovati i deklarisati baš kao i ostale promjenljive. Primjer:

```
int x;
int *p = &x;          // moglo je i int x,*p = &x;
```

- Drugi primjer je:

```
int *p = NULL;     // ili int *p=0
```

gdje je NULL simbolička konstanta definisana u biblioteci stdio.h, koja kaže da pokazivačka promjenljiva u datom trenutku ne pokazuje ni na jednu promjenljivu.

- Takođe, inicijalizacija se može vršiti i konkretnom adresom:

```
int *p = (int *) 0x000000006FA126BD;    // 64-bitna arhitek.
```

# Inicijalizacija pokazivača - Značaj

- Ako se pokazivačka promjenljiva ne inicijalizuje, ona pokazuje na bilo koju adresu u memoriji, a to može biti i adresa podataka potrebnih za rad operativnog sistema i sistemskih programa, proizvoljnih promjenljivih našeg programa, pa, recimo, čak i adresa preko kojih se komunicira sa periferijama.
- U velikim programima možete incidentno koristiti ovu promjenljivu, što može dovesti do "pucanja" programa.
- Stoga je značaj inicijalizacije pokazivača veći nego inicijalizacije ostalih promjenljivih.
- Grubo pravilo: ako pokazivač ne pokazuje na željenu promjenljivu tokom izvršenja programa, vratiti ga na NULL.

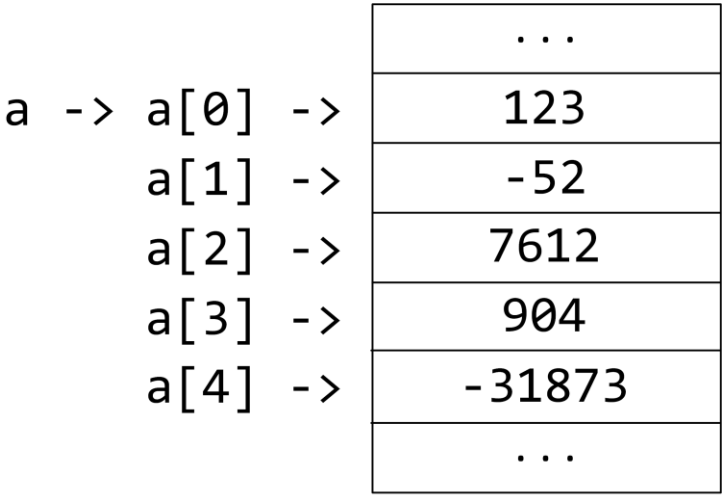
# Nizovi (vektori)

- Deklaracija niza se obavlja kao:

  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 tim promjenljivim su dozvoljene sve operacije kao sa datim tipom, 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_a) 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.
- Pokazivači su izuzetno pogodni za obilazak jer se mogu koristiti operatori inkrementiranja i dekrementiranja.
- 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 ...
- Moguć 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:

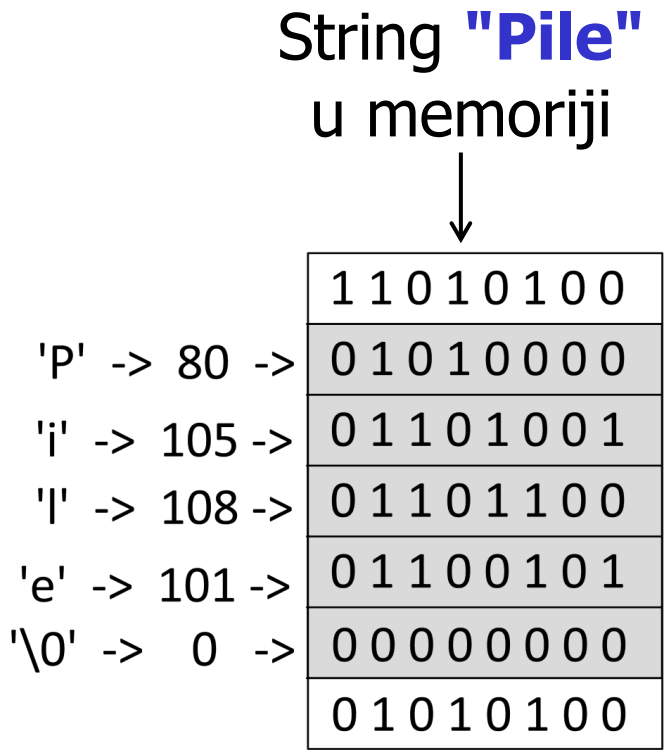
```
printf("Suma je %d", suma);
```


- String 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.

# Učitavanje i štampanje karaktera

- Funkcija **getch()** vraća karakter učitan sa tastature. Ova funkcija ne baferuje podatke koje učitava, odnosno ne čeka pritisak tastera Enter, već ono što učita odmah proslijedi u odgovarajuću promjenljivu:

```
char a;
a = getch();
```

- Funkcija **putch(a)** štampa karakter argument a.
- Iz bezbjednosnih razloga, umjesto gets() funkcije, za učitavanje stringova sa tastature preporučuje se upotreba char s[20]

```
fgets(s, sizeof(s), stdin);
```

→ stdin označava standardni ulaz – tastaturu, obavezan argument.