Programiranje 1

Milena Vujošević Janičić

www.matf.bg.ac.rs/~milena

Programiranje 1

Pregled

Pregled

Nizovi

- Često je u programima potrebno korišćenje velikog broja srodnih promenljivih.
- Umesto velikog broja pojedinačno deklarisanih promenljivih, moguće je koristiti *nizove*.
- Obrada elemenata nizova se onda obično vrši na uniforman način, korišćenjem petlji.

Primer

```
#include <stdio.h>

int main() {
  int b0, b1, b2, b3, b4, b5, b6, b7, b8, b9;
  scanf("%d%d%d%d%d%d%d%d%d%d",
        &b0, &b1, &b2, &b3, &b4, &b5, &b6, &b7, &b8, &b9);
  printf("%d %d %d %d %d %d %d %d %d %d",
        b9, b8, b7, b6, b5, b4, b3, b2, b1, b0);
}
```

Primer

```
#include <stdio.h>

int main() {
  int b[10], i;
  for (i = 0; i < 10; i++)
    scanf("%d", &b[i]);
  for (i = 9; i >= 0; i--)
    printf("%d ", b[i]);
  return 0;
}
```

Deklaracija niza

Deklaracija

```
tip ime_niza[dimenzija];
```

- Dimenzija predstavlja broj elemenata niza.
- Na primer, deklaracija

  `int a[10];`

  uvodi niz a od 10 celih brojeva. Prvi element niza ima indeks 0, pa su elementi niza:

  `a[0], a[1], a[2], a[3], a[4],`
  `a[5], a[6], a[7], a[8], a[9]`

Deklaracija i inicijalizacija

- Prilikom deklaracije se može izvršiti i inicijalizacija

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

- Nakon ove deklaracije, sadržaj niza a jednak je:

1	2	3	4	5

- Ako se koristi inicijalizacija, moguće je navesti veću dimenziju od broja navedenih elemenata (kada se inicijalizuju samo početni elementi deklarisanog niza).
- Neispravno je navođenje manje dimenzije od broja elemenata inicijalizatora.

Deklaracija i inicijalizacija

- Veličina memorijskog prostora potrebnog za niz određuje se u fazi prevođenja programa, pa broj elemenata niza (koji se navodi u deklaraciji) mora biti konstantan izraz.
- U narednom primeru, deklaracija niza `a` je ispravna, dok su deklaracije nizova `b` i `c` neispravn

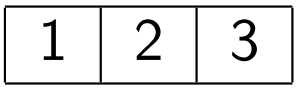
```
int x;
char a[100+10];
int b[];
float c[x];
```

Inicijalizacija

- Dimenziju niza je moguće izostaviti samo ako je prilikom deklaracije izvršena i inicijalizacija niza i tada se dimenzija određuje na osnovu broja elemenata u inicijalizatoru. Na primer, nakon deklaracije

```
int b[] = { 1, 2, 3 };
```

sadržaj niza `b` jednak je:



sizeof

- Kada se operator `sizeof` primeni na ime niza, rezultat je veličina niza u bajtovima.
- Broj elemenata može se izračunati na sledeći način:

```
sizeof(ime niza)/sizeof(tip elementa niza)
```

Pristup elementima niza

- Kada se pristupa elementu niza, indeks može da bude proizvoljan izraz celobrojne vrednosti, na primer:

  `a[2*2+1]`
  `a[i+2]`
- U fazi prevođenja (pa ni u fazi izvršavanja ne vrši se nikakva provera da li je indeks pristupa nizu u njegovim granicama i moguće je bez ikakve prijave greške ili upozorenja od strane prevodioca pristupati i lokaciji koji se nalazi van opsega deklarisanog niza (na primer, moguće je koristiti element `a[13]`, pa čak element `a[-1]` u prethodnom primeru).
- Ovo najčešće dovodi do fatalnih grešaka prilikom izvršavanja programa.

l vrednosti

- Pojedinačni elementi nizova su l-vrednosti (tj. moguće im je dodeljivati vrednosti).
- Međutim, nizovi nisu l-vrednosti i nije im moguće dodeljivati vrednosti niti ih menjati.

```
int a[3] = {5, 3, 7};
int b[3];
b = a;      /* Neispravno - nizovi se ne mogu dodeljivati
a++;        /* Neispravno - nizovi se ne mogu menjati. */
```

Primer

Gde je greška u narednom kodu?

```
int main(){
int n,i;
int a[10] = {0};
int b[10] = {0};
scanf("%d", &n);
printf("\na = \n");
for(i=0; i<10; i++)
    printf("%d ", a[i]);
printf("\nb = \n");
for(i=0; i<10; i++)
    printf("%d ", b[i]);
printf("\n");
for(i=0; i<n; i++)
    a[i] = i;
printf("\na = \n");
for(i=0; i<10; i++)
    printf("%d ", a[i]);
printf("\nb = \n");
for(i=0; i<10; i++)
    printf("%d ", b[i]);
printf("\n");
return 0;
}
```

Primer

Izlaz za n=5:

Primer

Izlaz za n=5:

```
a =
0  0  0  0  0  0  0  0  0  0
b =
0  0  0  0  0  0  0  0  0  0

a =
0  1  2  3  4  0  0  0  0  0
b =
0  0  0  0  0  0  0  0  0  0
```

Primer

Izlaz za n=10:

Primer

Izlaz za n=10:

```
a =
0  0  0  0  0  0  0  0  0  0
b =
0  0  0  0  0  0  0  0  0  0

a =
0  1  2  3  4  5  6  7  8  9
b =
0  0  0  0  0  0  0  0  0  0
```

Primer

Izlaz za n=15:

Primer

Izlaz za n=15:

Izmena niza b

```
a =
0  0  0  0  0  0  0  0  0  0
b =
0  0  0  0  0  0  0  0  0  0

a =
0  1  2  3  4  5  6  7  8  9
b =
10  11  12  13  14  0  0  0  0  0
```

Primer

Izlaz za n=25:

Primer

Izlaz za n=25:

Primer

Izlaz za n=25:

Izmena niza b i Segmentation fault!

```
a =
0 0 0 0 0 0 0 0 0 0
b =
0 0 0 0 0 0 0 0 0 0

a =
0 1 2 3 4 5 6 7 8 9
b =
10 11 12 13 14 15 16 17 18 19
Segmentation fault (core dumped)
```

Nizovi i funkcije

- Niz se ne može preneti kao argument funkcije — prenosi se samo adresa početka niza koja se zadaje njegovim imenom
- Kada se ime niza navede kao argument funkcije, onda do te funkcije stiže informacija o adresi početka niza i o tipu elemenata (ali ne i o imenu niza niti o broju elemenata niza)
- Kako izgleda memorija
- Pošto funkcija koja je pozvana dobija informaciju o adresi početka originalnog niza, ona može da neposredno menja njegove elemente (i takve izmene će biti sačuvane nakon izvršenja funkcije), što je drugačije u odnosu na sve ostale tipove podataka.
- Prenos adrese početka niza vrši se — kao i uvek — po vrednosti.

Niz kao argument funkcije

- Funkcija koja prima niz može biti deklarisana na neki od narednih načina:

```
tip ime_funkcije(tip ime_niza[dimenzija]);
tip ime_funkcije(tip ime_niza[]);
```

- S obzirom na to da se u funkciju prenosi samo adresa početka niza, a ne i dimenzija niza, prvi oblik deklaracije nema puno smisla tako se znatno ređe koristi.

Primer

Šta ispisuje naredni program

```
#include <stdio.h>

void f(int a[]) {
  printf("f: %d\n", sizeof(a));
}

int main() {
  int a[] = {1, 2, 3, 4, 5};
  printf("main: %d\n", sizeof(a));
  f(a);
  return 0;
}
```

Primer

Šta ispisuje naredni program

```
#include <stdio.h>

void f(int a[]) {
  printf("f: %d\n", sizeof(a));
}

int main() {
  int a[] = {1, 2, 3, 4, 5};
  printf("main: %d\n", sizeof(a));
  f(a);
  return 0;
}
```

```
main: 20
f: 4
```

Niz kao argument funkcije

- Prilikom prenosa niza (tj. adrese njegovog početka) u funkciju, pored imena niza, korisnik obično treba da eksplicitno prosledi i broj elemenata niza kao dodatni argument (da bi pozvana funkcija imala tu informaciju).
- Povratni tip funkcije ne može da bude niz. Funkcija ne može da kreira niz koji bi bio vraćen kao rezultat, a rezultat može da vrati popunjavanjem niza koji joj je prosleđen kao argument.

Primer — Šta ispisuje naredni program?

```
#include <stdio.h>

void ucitaj_broj(int a)
{
  printf("Ucitaj broj: ");
  scanf("%d", &a);
}
void ucitaj_niz(int a[], int n)
{
  int i;
  printf("Ucitaj niz: ");
  for (i = 0; i < n; i++)
     scanf("%d", &a[i]);
}
```

```
int main() {
  int x = 0;
  int y[3] = {0, 0, 0};
  ucitaj_broj(x);
  ucitaj_niz(y, 3);
  printf("x = %d\n", x);
  printf("y = %d %d %d\n",
       y[0], y[1], y[2]);
}
```

Pregled

Niske

- Posebno mesto u programskom jeziku C zauzimaju nizovi koji sadrže karaktere — *niske karaktera* ili kraće *niske* (engl. *strings*)
- Konstantne niske navode se između dvostrukih navodnika (na primer, `"ja sam niska"`).
- U okviru niski, specijalni karakteri navode se korišćenjem specijalnih sekvenci (na primer, `"prvi red\ndrugi red"`).
- Niske su interno reprezentovane kao nizovi karaktera na čiji desni kraj se dopisuje karakter `'\0'`, tzv. završna, terminalna nula (engl. *null terminator*)

Niske

- Niske mogu da se koriste i prilikom inicijalizacije nizova karaktera.

```
char s1[] = {'Z', 'd', 'r', 'a', 'v', 'o'};
char s2[] = {'Z', 'd', 'r', 'a', 'v', 'o', '\0'};
char s3[] = "Zdravo";
```

- Nakon navedenih deklaracija, sadržaj niza `s1` jednak je:

0	1	2	3	4	5
'Z'	'd'	'r'	'a'	'v'	'o'

a sadržaj nizova `s2` i `s3` jednak je:

0	1	2	3	4	5	6
'Z'	'd'	'r'	'a'	'v'	'o'	'\0'

Niske

- Niz `s1` sadrži 6 karaktera (i zauzima 6 bajtova).
- Deklaracije za `s2` i `s3` su ekvivalentne i ovi nizovi sadrže po 7 karaktera (i zauzimaju po 7 bajtova).
- Prvi niz karaktera nije terminisan nulom i ne može se smatrati ispravnom niskom.
- Potrebno je jasno razlikovati karakterske konstante (na primer, `'x'`) koje predstavljaju pojedinačne karaktere i niske (na primer, `"x"` — koja sadrži dva karaktera, `'x'` i `'\0'`).

string.h

- Standardna biblioteka definiše veliki broj funkcija za rad sa niskama.
- Prototipovi ovih funkcija se nalaze u zaglavlju `string.h`.
- Na primer, funkcija `strlen` služi za izračunavanje dužine niske.

```
int i = 0;
while (s[i] != '\0')
  i++;
```

- Pri računanju dužine niske, ne računa se završna nula.

string.h

- Pored strlen, postoje i druge interesantne funkcije, npr
  - `strcmp` — poređenje dve niske,
  - `strcpy` — kopiranje druge niske u prvu,
  - `strstr` — proverava da li je druga niska podniska prve niske,
  - `strcat` — nadovezuje drugu nisku na prvu,
  - ...

Niske

- Kôd koji obrađuju niske obično ih obrađuje karakter po karakter i obično sadrži petlju oblika:

```
while (s[i] != '\0')
   ...
```

ili oblika:

```
for (i = 0; s[i] != '\0'; i++)
    ...
```

Niske

- Kako završna nula ima numeričku vrednost 0 (što predstavlja istinitosnu vrednost *netačno*), poređenje u prethodnoj petlji može da se izostavi:

```
for (i = 0; s[i]; i++)
    ...
```

- Izuzetno neefikasan kôd dobija se pozivanjem funkcije `strlen` za izračunavanje dužine niske u svakom koraku iteracije:

```
for (i = 0; i < strlen(s); i++)
    ...
```

Niske i funkcije

- Kako su niske zapravo nizovi, to za prenos niski u funkcije važi sve što važi i za nizove
- Međutim, za nizove brojeva se ne može u funkciji odrediti dužina, i zbog toga je neophodno preneti dužinu niza kao argument funkcije
- S druge strane, za niske je moguće utvrditi njihovu dužinu (na osnovu konvencije da se završavaju sa `'\0'`) pa se zbog toga niske obično prenose bez dodatnog argumenta koji se odnosi na dužinu niske

Pregled

Višedimenzioni nizovi

Pored jednodimenzionih mogu se koristiti i višedimenzioni nizovi

Višedimenzioni nizovi

```
tip ime_niza[dimenzija_1]...[dimenzija_2];
```

- Dvodimenzioni nizovi (matrice) tumače se kao jednodimenzioni nizovi čiji su elementi nizovi. Zato se elementima dvodimenzionog niza pristupa sa:

  `ime_niza[vrsta][kolona]`

  a ne sa `ime_niza[vrsta, kolona]` (ovde je zapravo primena operatora ,)

Matrice

- Elementi se u memoriji smeštaju po vrstama
- Niz se može inicijalizovati navođenjem liste inicijalizatora u vitičastim zagradama; pošto su elementi opet nizovi, svaki od njih se opet navodi u okviru vitičastih zagrada (mada je unutrašnje vitičaste zagrade moguće i izostaviti).

```
char a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Primer

- Dvodimenzioni niz sadrži broj dana za svaki mesec, pri čemu su u prvoj vrsti vrednosti za obične, a u drugoj vrsti za prestupne godine:

```
char broj_dana[][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
```

- Šta izračunava naredni kod:

```
int prestupna = (godina % 4 == 0 && godina % 100 != 0) ||
                godina % 400 == 0;
char bd = broj_dana[prestupna][mesec];
```

Odnos sa funkcijama

- Ukoliko se ime dvodimenzionalnog niza koristi kao argument u pozivu funkcije, deklaracija parametra u funkciji mora da uključi broj kolona; broj vrsta je nebitan, jer se i u ovom slučaju prenosi samo adresa (tj. pokazivač na niz vrsta, pri čemu je svaka vrsta niz).
- U slučaju niza sa više od dve dimenzije, samo se prva dimenzija može izostaviti (dok je sve naredne dimenzije neophodno navesti).

```
tip ime_funkcije(tip ime_niza[bv][bk]);
tip ime_funkcije(tip ime_niza[][bk]);
```

Pregled

Korisnički definisani tipovi

- Jezik C ima svega nekoliko ugrađenih osnovnih tipova.
- Već nizovi i niske predstavljaju složene tipove podataka.
- Osim nizova, postoji još nekoliko načina izgradnje složenih tipova podataka.
- Promenljive se mogu organizovati u strukture, unije, nabrojive tipove i polja bitova

Strukture

- Osnovni tipovi jezika C često nisu dovoljni za pogodno opisivanje svih podataka u programu.
- Ukoliko je neki podatak složene prirode tj. sastoji se od više delova, ti njegovi pojedinačni delovi mogu se čuvati nezavisno (u zasebnim promenljivama), ali to često vodi programima koji su nejasni i teški za održavanje.
- Za razliku od nizova koji objedinjuju jednu ili više promenljivh istog tipa, struktura objedinjuje jednu ili više promenljivih, ne nužno istih tipova.
- Definisanjem strukture uvodi se novi tip podataka i nakon toga mogu da se koriste promenljive tog novog tipa, na isti način kao i za druge tipove.

Primer

Ključna reč `struct` započinje definiciju strukture. Nakon nje, navodi se ime strukture, a zatim, između vitičastih zagrada, opis njenih *članova*

```
struct razlomak {
    int brojilac;
    int imenilac;
};
```

Primer

Strukture mogu sadržati promenljive proizvoljnog tipa. Na primer, moguće je definisati strukturu koja sadrži i niz.

```
struct student {
  char ime[50];
  float prosek;
};
```

Primer

Strukture mogu biti ugnježdene, tj. članovi struktura mogu biti druge strukture. Na primer:

```
struct dvojni_razlomak {
    struct razlomak gore;
    struct razlomak dole;
};
```

Strukture

- Definicija strukture uvodi novi tip i nakon nje se ovaj tip može koristiti kao i bilo koji drugi.
- Prilikom deklarisanja promenljivih ovog tipa, kao deo imena tipa, neophodno je korišćenje ključne reči `struct`, na primer:

```
struct razlomak a, b, c;
```

- Definicijom strukture je opisano da se razlomci sastoje od brojioca i imenioca, dok se navedenom deklaracijom uvode tri konkretna razlomka koja se nazivaju `a`, `b` i `c`.

Strukture

- Moguća je i deklaracija sa inicijalizacijom, pri čemu se inicijalne vrednosti za članove strukture navode između vitičastih zagrada:

```
struct razlomak a = { 1, 2 };
```

- Redosled navođenja inicijalizatora odgovara redosledu navođenja članova strukture.
- Dakle, navedenom deklaracijom je uveden razlomak a čiji je brojilac 1, a imenilac 2.

Strukture

- Članu strukture se pristupa preko imena promenljive (čiji je tip struktura) iza kojeg se navodi tačka a onda ime člana, na primer:

  `a.imenilac`

- Na primer, vrednost promenljive `a` tipa `struct razlomak` može biti ispisan na sledeći način:

  ```
  printf("%d/%d", a.brojilac, a.imenilac);
  ```

- Naglasimo da je operator `.`, iako binaran, operator najvišeg prioriteta (istog nivoa kao male zagrade i unarni postfiksni operatori).

Nizovi struktura

- Kao što se prosti tipovi podataka mogu grupisati u nizove, tako može postojati i potreba za grupisanjem struktura u nizove
- Na primer

```
struct opis_meseca {
    char ime[10];
    int broj_dana;
};
...
struct opis_meseca meseci[13];
```

(deklarisan je niz dužine 13 da bi se meseci mogli referisati po svojim rednim brojevima, pri čemu se početni element niza ne koristi).

Primer

Moguća je i deklaracija sa inicijalizacijom (u kojoj nije neophodno navođenje broja elemenata niza)

```
struct opis_meseca meseci[] = {
  { "",0 },
  { "januar",31 },
  { "februar",28 },
  { "mart",31 },
  ...
  { "decembar",31 }
};
```

Primer

- Nakon navedene deklaracije, ime i-tog meseca u godini se može dobiti sa `meseci[i].ime`, njegov broj dana sa `meseci[i].broj_dana`
- Broj elemenata ovako inicijalizovanog niza može se izračunati na sledeći način:

```
sizeof(meseci)/sizeof(struct opis_meseca)
```

- Kako izgleda memorija za jednu strukturu, a kako za niz struktura?

Strukture i funkcije

- Parametri funkcija mogu biti i strukture i drugi korisnički definisani tipovi.
- Pored toga, funkcije kao tip povratne vrednosti mogu imati tip strukture.
- Prenos argumenta se i u ovom slučaju vrši po vrednosti.

Primer

Funkcija `kreiraj_razlomak` od dva cela broja kreira i vraća objekat tipa `struct razlomak`:

```
struct razlomak kreiraj_razlomak(int brojilac, int imenilac) {
    struct razlomak rezultat;
    rezultat.brojilac = brojilac;
    rezultat.imenilac = imenilac;
    return rezultat;
}
```

Primer

Struktura moze biti parametar funkcije

```
struct razlomak saberi_razlomke(struct razlomak a,
                                struct razlomak b) {
    struct razlomak c;
    c.brojilac = a.brojilac*b.imenilac + a.imenilac*b.brojilac;
    c.imenilac = a.imenilac*b.imenilac;
    return c;
}
```

Primer

Šta nije u redu sa narednim kodom?

```
void skrati(struct razlomak r) {
  int n = nzd(r.brojilac, r.imenilac);
  r.brojilac /= n; r.imenilac /= n;
}
```

Unije i polja bitova

- Unije i polja bitova predstavljaju načine uštede memorije
- Unije su donekle slične strukturama, ali podaci koji se čuvaju u njima dele isti memorijski prostor. Na primer:

```
union vreme {
  int obicno;
  float precizno;
}
...
union vreme a;
a.obicno=17;
```

Kako izgleda memorija u ovom slučaju?

- Polja bitova se koriste da bi vrednosti za koje se unapred zna da imaju veoma mali opseg, zauzimale samo dogovarajuci broj bitova, a ne i cele bajtove.

Nabrojivi tipovi

- U nekim slučajevima korisno je definisati tip podataka koji ima mali skup dopuštenih vrednosti.
- Ovakvi tipovi se nazivaju *nabrojivi tipovi*
- Nabrojivi tipove se definišu korišćenjem ključne reči `enum`. Na primer:

```
enum znak_karte {
  KARO,
  PIK,
  HERC,
  TREF
};
```

Nabrojivi tipovi

- Nakon navedene definicije, u programu se mogu koristiti imena `KARO`, `PIK`, `HERC`, `TREF`, umesto nekih konkretnih konstantnih brojeva, što popravlja čitljivost programa.
- Pri tome, obično nije važno koje su konkretne vrednosti pridružene imenima `KARO`, `PIK`, `HERC`, `TREF`, već je dovoljno znati da su one sigurno međusobno različite i celobrojne.
- U navedenom primeru, `KARO` ima vrednost 0, `PIK` vrednost 1, `HERC` vrednost 2 i `TREF` vrednost 3.

Nabrojivi tipovi

- Moguće je i eksplicitno navođenje celobrojnih vrednosti. Na primer:

```
enum znak_karte {
  KARO = 1,
  PIK  = 2,
  HERC = 4,
  TREF = 8
};
```

Primer

Nabrojivi tipovi mogu učestvovati u izgradnji drugih složenih tipova.

```
struct karta {
  unsigned char broj;
  enum znak_karte znak;
} mala_dvojka = {2, TREF};
```

typedef

- U jeziku C moguće je kreirati nova imena postojećih tipova koristeći ključnu reč `typedef`. Na primer, deklaracija

  ```
  typedef int Length;
  ```

  uvodi ime `Length` kao sinonim za tip `int`. Ime tipa `Length` se onda može koristiti u deklaracijama, eksplicitnim konverzijama i slično, na isti način kao što se koristi ime `int`:

  ```
  Length len, maxlen;
  ```

- Obično se novouvedena imena tipova pišu velikim početnim slovima da bi se istakla.

typedef

- Deklaracijom `typedef` se ne kreira novi tip već se samo uvodi novo ime za potojeći tip. Staro ime za taj tip se može koristiti i dalje.
- Veoma često korišćenje `typedef` deklaracija je u kombinaciji sa strukturama da bi se izbeglo pisanje ključne reči `struct` pri svakom korišćenju imenu strukturnog tipa. Na primer:

```
struct point {
  int x, y;
};
typedef struct point Point;

Point a, b;
```

typedef

- Definicija strukture i pridruživanje novog imena se mogu uraditi istovremeno. Na primer:

```
typedef struct point {
  int x, y;
} Point;

Point a, b;
```

Literatura

Slajdovi su pripremljeni na osnovu materijala iz šestog i osmog poglavlja knjige:
Filip Marić, Predrag Janičić: Programiranje 1
Za pripremu ispita nisu dovoljni slajdovi, potrebno je koristiti knjigu!