Složeni tipovi podataka u programskom jeziku C

Složeni tipovi podataka u jeziku C

Složeni (izvedeni) tipovi u programskom jeziku C su:

- **nizovi**
- **strukture**
- **unije**
- **polja bitova**

U zavisnosti od načina pristupanja elementima dele se na:

- **indeksirane** - elementu se pristupa na osnovu njegove pozicije (indeksa) – **nizovi - [ ]**
- **strukture, unije, polja bitova** - elementima se pristupa na osnovu njihovog naziva u strukturi – **struct**, **union**

Niz – deklarisanje

Niz predstavlja kolekciju elemenata istog tipa

Zauzima kontinuirani niz bajtova u memoriji

Deklariše se na sledeći način:

tip identifikator[brojElem];

- **tip** određuje kog tipa će biti elementi niza
- **identifikator** definiše naziv niza
- **brojElem** definiše koliko će elemenata niz sadržati

Količina memorije koju će niz zauzeti računa se kao

sizeof(tip) * brojElem

sizeof - operator za određivanje količine memorije koju zauzima neki tip podatka

Niz – deklarisanje primeri

Primer deklarisanja niza:

```
int nizA[5];
```
niz od 5 elemenata tipa int

```
float nizB[12];
```
niz od 12 elemenata tipa float

Broj elemenata niza mora biti neposredni operand ili konstantna vrednost ! Zašto?

```
#define MAXINT 100
int nizC[MAXINT];
```

Niz – definisanje

Vrednosti elemenata niza se inicijalizuju tokom deklaracije, tj. niz se definiše na sledeći način:

tip identifikator [brojElem] = {e_1, e_2, ... , $e_{brojElem}$};

- **tip** određuje kog tipa će biti elementi niza
- **identifikator** definiše naziv niza
- **brojElem** – definiše koliko će elemenata niz sadržati, može se izostaviti jer se broj elemenata niza može izbrojati iz **{ }**
- **e_1, e_2, ..., $e_{brojElem}$** – vrednosti koje će se zapamtiti u nizu; ako je deklarisan *brojElem*, broj vrednosti mora jednak ili manji od *brojElem*

Niz – primeri definisanja

Primeri definisanja niza:

int nizA[ ] = {1, 2, 3, 4, 5};
niz od 5 elemenata tipa **int** čiji su elementi 1, 2, 3, 4, 5

float nizB[5] = {1.1, 2.1, 3.4, 4.8, 5.9};
niz od 5 elemenata tipa **float** čiji su elementi 1.1, 2.1, 3.4, 4.8, 5.9

double nizC[100] = {2.2,14.3,17.33}
niz od 100 elemenata tipa **double** čiji su prva tri elementa 2.2, 14.3, 17.33 (**šta je sa preostalim elementima?**)

double nizD[100] = {0}
niz od 100 elemenata tipa **double** čiji svi elementi imaju vrednost 0

Pristupanje elementima niza

Elementima niza se pristupa preko operatora **[ind]**
gde je **ind** indeks (pozicija) elementa u nizu

$$\mathbf{0 \leq ind < brojElem}$$

Indeks niza u C programskom jeziku **uvek ide od 0**, a ne od 1!

Identifikator niza predstavlja **adresu prvog elementa niza!**

Primeri pristupanja elementima niza:

```
nizA[0] = 12;
nizA[1] = 2;
nizA[2] = nizA[0] * nizA[0];
broj = nizA[0];
```

Niz – inicijalizacija

Niz se ne može inicijalizovati odjednom, već elemenat po elemenat

Idealno je koristiti **for** iteraciju kada se zna tačan broj elemenata niza

Niz se može inicijalizovati na sledeći način:

```
int i;
for (i = 0; i < brojElem; i++)
    nizA[ i ] = ei;
```

for iteracija se koristi i za prolaz kroz sve elemente niza (ili jednog dela niza)

Unos elemenata niza sa tastature

Mora se unositi svaki elemenat zasebno

Navodi se adresa indeksiranog elementa niza (ne mora jedino za prvi element)

Primer unosa vrednosti elemenata niza sa standardnog ulaza (pretpostavlja se niz celih brojeva od 5 elemenata):

```
for (i = 0; i < 5; i++)
    scanf("%d", &niz[i]);
```

Ispis elemenata niza

Mora se ispisivati svaki element zasebno

Primer ispisa vrednosti elemenata niza na standardni izlaz (pretpostavlja se niz celih brojeva od 5 elemenata):

```
for (i = 0; i < 5; i++)
    printf("%d", niz[i]);
```

Niz – primer

Zadatak:
Implementirati program za računanje sume vrednosti elemenata niza prirodnih brojeva koji sadrži najviše 50 elemenata. Program prihvata od korisnika broj elemenata ($0 < n \leq 50$) i vrednost svakog pojedinačnog elementa.

Vežba:
Proširiti prethodni zadatak tako da računa poziciju maksimalne vrednosti niza. Korisniku se prikazuje na kojoj poziciji u nizu se nalazi maksimalna vrednost, kao i sama vrednost.

Šta ako ima više istih vrednosti koje su maksimalne?

Niz – zadatak

```
#include <stdio.h>
#include <stdlib.h>
#define MAXEL 50
int main()
{
   int n = 1, i, suma = 0;
   int nizPrirodnihBrojeva[MAXEL];
   printf("\n Program za racunanje sume elemenata niza N prirodnih brojeva.\n\n");
   printf("Unesite N:\t");                                  // prihvati broj elemenata niza
   scanf("%d", &n);
   for (i = 0; i < n; i++)                                  // prihvati elemente niza od korisnika
   {
      printf("Unesite %d. element niza:\t", i+1);
      scanf("%d", &nizPrirodnihBrojeva[i]);
   }
   for (i = 0; i < n; i++)                                  // izracunaj sumu elemenata niza
      suma += nizPrirodnihBrojeva[i];
   printf("\n Suma elemenata niza izosi %d.\n\n", suma);    // prikazi rezultat
   return 0;
}
```

- **Dodati proveru korektnosti unetih vrednosti**
- **Da li se može koristiti manji broj FOR iteracija?**

Nizovi i pokazivači

Nizovi se mogu posmatrati i kao pokazivači

Naziv niza je pokazivač na prvi element niza (prvu lokaciju u nizu kontinuiranih lokacija)

Tako da:

`int x[10];`	**deklarisanje niza `x`**
`int *px;`	**deklarisanje pokazivača `px`**
`px = x;`	**je isto što i `px = &x[0];`**
`px + 3`	**je isto što i `&x[3]`**
`*(px + 4)`	**je isto što `x[4]`**
`*(px + 4) = 12;`	**je isto što i `x[4] = 12;`**

Matrice

Matrica – deklarisanje

Programski jezik C podržava rad sa višedimenzionalnim nizovima

Matrica je niz nizova, deklariše se na sledeći način:

tip identifikator[brojVrsta][brojKolona];

- **tip** određuje kog tipa će biti elementi matrice
- **identifikator** definiše naziv matrice
- **brojVrsta** definiše koliko vrsta sadrži matrica
- **brojKolona** definiše koliko kolona sadrži matrica

Ukupan broj elemenata matrice je:
brojVrsta * brojKolona

Matrica – deklarisanje primeri

Slično nizu:

int matrica[3][4];
matrica koja sadrži 12 elemenata smeštenih u 3 vrste i 4 kolone

Kao i kod niza, broj vrsta i kolona mora biti ili **konstanta** ili **neposredni operand**

Matrica zauzima kontinuiranu količinu memorije na sledeći način:

1	2	3	4
5	6	7	8
9	10	11	12

adresa(m[i][j]) =adresa_prvog_m+ i * broj_elem_vrste_m + j

Matrica – definisanje

Vrednosti elemenata niza se inicijalizuju tokom deklaracije (definicija) na sledeći način:

tip identifikator [brVr][brKol] = {{e_{11}, e_{12}, …, e_{1brKol}},
{e_{21}, e_{22}, …, e_{2brKol}},
…
{e_{brVr1}, e_{brVr2}, …,$e_{brVr\ brKol}$}};

- **tip** određuje kog tipa će biti elementi matrice
- **identifikator** definiše naziv matrice
- **brVr** – definiše koliko će vrsta sadržati matrica, može se izostaviti jer se može izbrojati
- **brKol** – definiše koliko će kolona sadržati matrica
- **e_{11}, e_{12}, …, e_{1brKol}** – vrednosti koje će se memorisati u jednu vrstu

Matrica – definisanje primeri

Primer definisanja matrice:

```
int matrica[3 ][4] = {  {1,  2,  3,  4},
                        {5,  6,  7,  8}
                        {9, 10, 11, 12} };
```

matrica od 12 elemenata tipa **int** raspoređenih u 3 vrste i 4 kolone

```
int matrica[ ][4] = {  {1,  2,  3,  4},
                       {5,  6,  7,  8}
                       {9, 10, 11, 12} };
```

matrica od 12 elemenata tipa **int** raspoređenih u 3 vrste i 4 kolone

Pristupanje elementima matrice

Elementima matrice se pristupa preko operatora
[indeksVrste] [indeksKolone]
gde je

$$0 \leq \textbf{indeksVrste} < \textbf{brojVrsta}$$
$$0 \leq \textbf{indeksKolone} < \textbf{brojKolona}$$

Identifikator matrice predstavlja **adresu prvog elementa matrice!**

Primeri pristupanja elementima matrice:

```
matrica[0][0] = 12;
matrica[1][0] = 2;
matrica[0][3] = matrica[0][0] * matrica[1][0];
a = matrica[0][3];
```

Matrica – inicijalizacija

Kao i niz, ne može se inicijalizovati odjednom, već elemenat po elemenat

Idealno je koristiti dve **for** iteracije - kada se zna tačan broj elemenata matrice

Matrica se može inicijalizovati na sledeći način:

```
for (i = 0; i < brojVrsta; i++)
    for (j = 0; j < brojKolona; j++)
        matrica[ i ][ j ] = eij;
```

Dve for iteracije se koriste i za prolaz kroz sve elemente matrice (ili jednog njenog dela)

Matrica – primer

Zadatak:

Implementirati program za računanje srednje vrednosti elemenata matrice prirodnih brojeva koja sadrži najviše 10 vrsta i 20 kolona. Program prihvata od korisnika broj vrsta i kolona i vrednost svakog pojedinačnog elementa.

Vežba:

Proširiti prethodni zadatak tako da računa sumu elemenata svake pojedinačne vrste. Korisniku se prikazuje par vrsta – suma njenih elemenata.

Matrica – zadatak

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

#define MAXVRSTA 10
#define MAXKOLONA 20

int main()
{
    int brojVrsta = 1, brojKolona = 1, i, j;
    float srednjaVrednost = 0;
    int matricaPrirodnihBrojeva[MAXVRSTA][MAXKOLONA];

    printf("\n Program za racunanje sr. vred. elem. matrice prirodnih brojeva.\n\n");

    printf("Unesite broj vrsta:\t");                    // prihvati broj vrsta i kolona
    scanf("%d", &brojVrsta);
    printf("\nUnesite broj kolona:\t");
    scanf("%d", &brojKolona);


    ...
```

Matrica – zadatak

```
...

for (i = 0; i < brojVrsta; i++)            // prihvati elemente matrice od korisnika
    for (j = 0; j < brojKolona; j++)
    {
        printf("Unesite element matrice na koordinati [%d][%d]:\t", i+1, j+1);
        scanf("%d", &matricaPrirodnihBrojeva[i][j]);
    }

for (i = 0; i < brojVrsta; i++)
    for (j = 0; j < brojKolona; j++)
        srednjaVrednost += matricaPrirodnihBrojeva[i][j];    // izracunaj sumu
                                                             //elemenata matrice

srednjaVrednost /= (brojVrsta*brojKolona);          // izracunaj srednju vrednost
                                                    // elemenata matrice

printf("\n Srednja vrednost elemenata matrice je %f.\n\n", srednjaVrednost);

return 0;
}
```

Multidimenzionalni nizovi

Logika koja se primenjuje kod nizova i matrica proširuje se i na više dimenzija

Opšti oblik deklaracije višedimenzionalnog niza izgleda:

tip identifikator [dim1] [dim2] ... [dimN];

- **tip** određuje kog tipa će biti elementi višedimenzionalnog niza
- **identifikator** definiše naziv višedimenzionalnog niza
- **dim1** definiše koliko će elemenata biti u prvoj dimenziji višedimenzionalnog niza
- **dim2** definiše koliko će elemenata biti u drugoj dimenziji višedimenzionalnog niza
- **dimN** definiše koliko će elemenata biti u N-toj dimenziji višedimenzionalnog niza

Stringovi

String

Posebna vrsta jednodimenzionalnog niza tipa **char**

Null terminisani - što znači da se svakom stringu (nizu karaktera) na kraju dopisuje završni NULL znak: **'\0'** (ASCII vrednost 0) koji je sastavni deo stringa

Zbog NULL znaka za string od 10 karaktera treba deklarisati niz karaktera od 11 elemenata

Rad sa stringovima je specifičan način korišćenja niza znakova

Za kompajler string konstanta je svaki niz znakova između znaka navoda. Znaci između navodnika plus završni znak memorišu se u niz susednih lokacija

'C' i **"C"** se razlikuju, **"C"** je niz od dva znaka: **'C'** i **'\0'**

Prazan string **""** sadrži samo završni znak **'\0'**

String – deklarisanje/inicijalizacija

Prvo se mora deklarisati niz karaktera:

```
char identifikator [brojElem];
```

Treba voditi računa da brojElem bude jednak broju karaktera koji se želi preuzeti plus završni karakter

Ako se string ne definiše, inicijalizacija se vrši na isti način kao i za standardne nizove

```
char niz [brojElem];
for (i = 0; i < brojElem; i++)
    niz[ i ] = ei;
```

gde $\mathbf{e_i}$ mora biti karakter

String – definisanje

Stringovi se mogu definisati korišćenjem string konstanti:

```
char nizS[brojElem] = "Neki string";
```

- definisan je statički niz **nizS** datom konstantom
- moguće je izostaviti **brojElem** jer se to može odrediti iz string konstante

```
char nizS[ ] = "Neki string";
```

Definisanje string konstantom je ekvivalentno sa:

```
char nizS[ ] = {'N', 'e', 'k', 'i', ' ', 's', 't', 'r', 'i', 'n', 'g', '\0'};
```

Ako bi se izostavio završni znak, to više ne bi bio string, već običan niz

Kao i kod običnih nizova, **naziv je adresa prvog elementa**!

Rad sa stringovima

U jeziku C ne postoji poseban osnovni tip za stringove, međutim glavni problem je nedostatak operatora za rad sa stringovima

Rešenja:

1. Operatore realizovati algoritamski, kao da se radi sa numeričkim nizovima
2. Koristiti gotove funkcija iz zaglavlja **string.h** standardne biblioteke

Unos stringa sa tastature

Mora se rezervisati dovoljno memorije i za završni znak

Učitavanje stringa može se realizovati sledećim funkcijama:

- **scanf** – za formatirani unos teksta (mora se voditi računa da će scanf prihvati sve karaktere do prvog praznog mesta, sem ako se ne koristi poseban format za opis unosa)
- **gets** – prihvata čitav string sa ulaza – tastature, pa i prazna mesta, **ali je vrlo nebezbedna, pa se ne preporučuje!**
- **fgets** – prihvata niz karaktera ograničene dužine iz ulazne datoteke tj. toka (može biti i ***stdin***),
- **gets_s** – prihvata niz karaktera ograničene dužine sa ulaza - tastature
- **fgets** i **gets_s** su daleko sigurnije alternative **gets** metode

Funkcija *gets*

Deklaracija ima sledeći izgled:

```
char *gets(char *str);
```

Funkcija učitava niz karaktera (broj učitanih karaktera nije definisan!) i vraća pokazivač na početak stringa ili NULL u slučaju kada dođe do greške ili nijedan karakter nije učitan

Funkcija nije bezbedna!

Napad korišćenjem prepunjavanja bafera (engl. *buffer overflow*) - Morris-ov crv – 2. novembar 1988.

Izbačena iz C 2011 standarda

Funkcija *fgets*

Deklaracija ima sledeći izgled:

char *fgets(char *str, int velicinaStr, FILE *datoteka);

Funkcija učitava ograničen niz karaktera (broj učitanih karaktera je < **velicinaStr**) u promenljivu **str** iz datoteke **datoteka**

Maksimalan broj učitanih karaktera može biti **velicinaStr–1**.
Zašto?

Karakteri se učitavaju ili do maksimalnog broja ili dok se ne naiđe na **EOF** (kraj datoteke) ili oznaku za novi red (**'\n'**)

Ako se podaci učitavaju sa tastature, **datoteka == stdin**

Funkcija *fgets*

Ako su podaci uspešno učitani, **fgets** vraća pokazivač na učitani string, tj. sam string (**str**)

U slučaju da se prilikom učitavanja stiglo do kraja datoteke (a da mu ne prethodi ni jedan karakter), ili da je došlo do greške, **fgets** vraća **NULL**

Ako se učita i end-of-line karakter, on će biti uključen u string **str**

Ako ima više od **velicinaStr** karaktera, oni ostaju na ulaznom toku

Kombinacija **scanf** i **fgets** u istom programu mora da se koristi pažljivo

Funkcija *gets_s*

Deklaracija ima sledeći izgled:

char *gets_s(char *str, int velicinaStr);

Predstavlja sigurniju verziju **gets** metode

Funkcija učitava ograničen niz karaktera (broj učitanih karaktera je < **velicinaStr**) u promenljivu **str** sa standardnog ulaza

Maksimalan broj učitanih karaktera može biti **velicinaStr – 1**. **Zašto?**

Karakteri se učitavaju ili do maksimalnog broja ili dok se ne naiđe na **EOF** (kraj datoteke) ili oznaku za novi red (**'\n'**) koji se sam ne dodaje u str

Funkcija *gets_s*

Ako su podaci uspešno učitani, **gets_s** vraća pokazivač na učitani string, tj. sam string (**str**)

U slučaju da se prilikom učitavanja stiglo do kraja datoteke (a da mu ne prethodi ni jedan karakter), ili da je došlo do greške, **gets_s** vraća **NULL**

Sledeće greške se mogu detektovati pomoću **gets_s**:

- **velicinaStr** je 0
- **velicinaStr** je veća od RSIZE_MAX (makro koji računa maksimalnu dozvoljena vrednost veličine stringa)
- **str** je NULL pointer
- nije otkriven **end-of-line/new-line** ili **EOF** unutar niza karaktera veličine **velicinaStr**

fgets/gets_s primeri

Opšti deo:

```
#define VELBUF 81;
...
char* str;
...
```

fgets primer

```
if (fgets(str, VELBUF, stdin) == NULL) {
    printf("Doslo je do greske prilikom ucitavanja!");
}
```

gets_s primer

```
if (gets_s(str, VELBUF) == NULL) {
    printf("Doslo je do greske prilikom ucitavanja!");
}
```

Ispisivanje stringa na izlazu

Ispisivanje stringa može se realizovati sledećim funkcijama:

- **printf** – za formatirani prikaz teksta (može prikazivati mešovite podatke)
- **puts** – ispisuje samo stringove

Da bi se prešlo u novi red kod **printf()** funkcije mora se navesti '\n', dok **puts()** to radi automatski

Primer:

```
printf("%s \n", ime1);
puts(ime2);
```

Funkcija *puts*

Koristi se za ispis teksta, tj. stringa

Ima samo jedan argument, pokazivač na početak stringa (ili podstringa) koji se ispisuje

Ispisuje sve znakove, počev od karaktera na koji pokazuje parametar funkcije, pa do završnog znaka

U ispisu završni znak zamenjuje se znakom za prelazak u novi red. **Koji je završni znak stringa?**

Posle ispisa stringa prelazi se u novi red

Funkcija **puts()** ima sledeće zaglavlje:

```
int puts(const char *s)
```

Funkcije za manipulaciju stringovima

Za manipulaciju sadržajem stringa koristi se skup za to posebno implementiranih funkcija

Nalaze se u **<string.h>**

Neke od funkcija za manipulaciju stringovima	
strcpy(s1, s2);	Kopira string s2 u string s1
strcat(s1, s2);	Dodaje string s2 na kraj stringa s1
strlen(s1);	Vraća dužinu stringa s1
strcmp(s1, s2);	Vraća 0 ako su s1 i s2 identični, < 0 ako s1 sadrži "manji" karakter od s2, > 0 ako s1 sadrži "veći" karaktera od s2
strchr(s1, ch);	Vraća pokazivač na prvo pojavljivanje znaka ch u s1
strstr(s1, s2);	Vraća pokazivač na prvo pojavljivanje stringa s2 u stringu s1

Rad sa stringovima – primer

Zadatak:
Napraviti program koji prihvata string i karakter od korisnika, a kao rezultat prikazuje koliko se puta taj karakter pojavljuje u stringu.

Vežba:
Učitati string, zatim omogućiti korisniku da bira neku od sledećih radnji: ispis stringa, određivanje dužine stringa, ispis stringa obrnutim redosledom, brojanje velikih slova u stringu (char se može tumačiti kao broj, tako da je **'a' < 'b' < 'c'**). Omogućiti korisniku da bira izvršavanje datih radnji sve dok ne odabere kraj programa.

String – zadatak

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUF 255
int main()
{
    char strIzraz[BUF];
    char karakter;
    int i = 0, brojPojavStr = 0, duzStringa = 0;
    printf("\n Program za trazenje ukupnog broja pojavaljivanja karaktera u nekom stringu.\n\n");
    printf("Unesite string:\t");
    gets_s(strIzraz, BUF);
    printf("\nUnesite karakter:\t");
    scanf("%c", &karakter);
    duzStringa = strlen(strIzraz);
    for (i = 0; i < duzStringa; i++)
      if (strIzraz[i] == karakter)
        brojPojavStr++;
    printf("\nKarakter \'%c\' se pojavljuje %d puta u stringu\n-\"%s\".\n", karakter,
                                                                    brojPojavStr, strIzraz);
    return 0;
}
```

Rad sa stringovima – *buffer overflow*

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

int main(void)
{
   char buff[15];
   int pass = 0;
   printf("\n Unesite lozinku: \n");
   gets(buff);
   if(strcmp(buff, "tajnaSifra42"))
   {
      printf ("\n Pogresna lozinka! \n");
   }
   else
   {
      printf ("\n Ispravna lozinka! \n");
      pass = 1;
   }
   if(pass)
   {
     /* Dodeli root ili admin privilegije korisniku*/
      printf ("\n Root privilegije dodeljene korisniku! \n");
   }
   return 0;
}
```

Rad sa stringovima – *buffer overflow*

Interakcija sa programom – **ispravna lozinka**:

$ ./bfrovrflw

Unesite lozinku: tajnaSifra42

Ispravna lozinka!

Root privilegije dodeljene korisniku!

Rad sa stringovima – *buffer overflow*

Interakcija sa programom – **neispravna lozinka**:

```
$ ./bfrovrflw
Unesite lozinku:  sifra123
Neispravna lozinka!
```

Interakcija sa programom – **neispravna lozinka, ali duža od bafera za čuvanje ulaznog stringa**:

```
$ ./bfrovrflw
Unesite lozinku:  hhhhhhhhhhhhhhhhhhhh
Neispravna lozinka!
Root privilegije dodeljene korisniku!
```

Buffer overflow!!!

Složeni tipovi podataka – strukture, unije, polja bitova

Složeni tipovi podataka - strukture

Postoje 3 vrste složenih tipova podataka - struktura u programskom jeziku C:

- **STRUKTURA** – obično se definiše nad podacima koji su u semantičkoj vezi, "nepromenljiva" struktura
- **UNIJA** – omogućava da se u istoj lokaciji pamte različiti tipovi podataka, "promenljiva" struktura
- **POLJE BITOVA** - varijanta strukture koja omogućava pristup na nivou bitova (kao kod jezika druge generacije)

Zahteva se kontinualni memorijski prostor za njihovo čuvanje

Struktura

Statička struktura kod koje je **pozicija elemenata određena njegovim nazivom**

Koristi se za predstavljanje skupa podataka koji opisuju neka bitna svojstva objekta

Elementi strukture mogu biti različitog tipa i obično su **semantički povezani**

Zahtevaju kontinuiran memorijski prostor za svoje smeštanje

Rezervisana reč **struct**

Često se koriste u kombinaciji sa nizovima, pri čemu formiraju tabele

Struktura - deklarisanje

Struktura se deklariše na sledeći način:

```
struct [identifikator]
{
    deklaracija elementa1;
    deklaracija elementa2;
    ...
    deklaracija elementaN;
} [jedna ili više promenljivih date strukture];
```

Moguće je izostaviti identifikator, pri čemu se mora deklarisati barem jedna promenljiva date strukture

Moguće je izostaviti promenljive, pa ih kasnije deklarisati

Struktura – deklarisanje primer

```
struct racun
{
   unsigned int brojRacuna;
   char imePrezime[51];
   char imeBanke[31];
   float stanje;
}
```

– Struktura za koju nije deklarisana promenljiva, te se ona mora naknadno deklarisati (kombinacija **struct identifikator + naziv_promenljive**):

```
struct racun mojRacun;
```

```
struct racun
{
   unsigned int brojRacuna;
   char imePrezime[51];
   char imeBanke[31];
   float stanje;
} mojRacun;
```

– Struktura za koju je deklarisana i promenljiva

Struktura – deklarisanje primer

```
struct
{
   short x;
   short y;
} promTacka;
```

– Struktura za koju nije deklarisan identifikator (posledice?)

```
struct tacka
{
   short x;
   short y;
} promTacka;
```

– Deklaracija novog tipa strukture zajedno sa promenljivom, preko novog tipa strukture mogu se i kasnije deklarisati nove promenljive:

```
struct tacka koordinatniPocetak, t1;
```

Pristupanje elementu strukture

Elementu strukture pristupa se preko njegovog imena i operatora . (tačka)

Sintaksa:

```
strukProm.element
```

Primer:

```
t1.x = 0;
t1.y = 0;
if (t1.x == koordinatniPocetak.x &&
    t1.y == koordinatniPocetak.y)
   printf("Tacka se nalazi u koordinatnom
               pocetku.");
```

Struktura – inicijalizacija

Svaki element inicijalizuje se zasebno

Pošto svaki element može biti drugačijeg tipa a pristupa im se preko naziva, ne koriste se iteracije

Sintaksa:

```
strukProm.element1 = vrednost1;
strukProm.element2 = vrednost2;
...
strukProm.elementN = vrednostN;
```

Struktura – primer inicijalizacije

```
mojRacun.brojRacuna = 982;

strcpy(mojRacun.imePrezime, "Dusan Gajic");

strcpy(mojRacun.imeBanke, “Credit Suisse");

mojRacun.stanje = 111111.0;
```

struct i *typedef*

Često korišćena kombinacija

Imenuje se novi složeni korisnički tip podataka

Izbegava se stalno ponavljanje rezervisane reči **struct** – kraći i "čistiji" kod

```
typedef struct
{
    unsigned int brojRacuna;
    char imePrezime[51];
    char imeBanke[31];
    float stanje;
} racun;
```

Rad sa strukturama – primer

Zadatak:
Napraviti program koji prihvata podatke o polaznicima kursa (ime, prezime, JMBG, grad). Podaci se smeštaju u niz. Može biti maksimalno 40 polaznika. Realizovati funkcionalnost za prikaz svih polaznika iz istog grada koji zadaje korisnik. Omogućiti korisniku da ponavlja prethodnu radnju sve dok ne odabere kraj programa.

Vežba:
Proširiti prethodni primer tako da se omogući pretraživanje i po drugim kriterijumima (ime, prezime, JMBG)

Struktura – zadatak

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

#define MAXPOLAZNIKA 40
#define MAXCHAR 31
#define MAXJMBG 14

typedef struct
{
    char ime[MAXCHAR];
    char prezime[MAXCHAR];
    char jmbg[MAXJMBG];
    char grad[MAXCHAR];
} polaznik;

int main()
{
    char traziGrad[31];
    char da = 'N';

    polaznik polaznici[MAXPOLAZNIKA];

    int brPolaznika = 0;
    int i = 0;

    printf("\n Program za unos i pretrazivanje polaznika.\n\n");
```

Struktura – zadatak

```
printf("Unesite broj polaznika (maksimalno 40):\t");
scanf("%d", &brPolaznika);

for (i = 0; i < brPolaznika; i++){
        printf("\nUnesite podatke za %d. polaznika:\n", i+1);
        printf("\nIme:\t\t");
        gets_s(polaznici[i].ime, MAXCHAR);
        printf("\nPrezime:\t");
        gets_s(polaznici[i].prezime, MAXCHAR);
        printf("\nJMBG:\t\t");
        gets_s(polaznici[i].jmbg, MAXJMBG);
        printf("\nGrad:\t\t");
        gets_s(polaznici[i].grad, MAXCHAR);
        printf("-----------------------------------");
    }
```

Struktura – zadatak

```
do{
        printf("\n Unesite grad za koji zelite da prikazete polaznike:\t");
        scanf(" %[^\t\n]s",traziGrad);              //regex - ucitava string dok se ne naidje \n ili \t

        printf("\nIz trazenog grada su polaznici:\n");
        printf("-----------------------------------");
        for (i = 0; i < brPolaznika; i++){
            if (!strcmp(polaznici[i].grad, traziGrad)){
                printf("\nRedni broj polaznika:\t%d", i+1);
                printf("\nIme:\t\t\t%s", polaznici[i].ime);
                printf("\nPrezime:\t\t%s", polaznici[i].prezime);
                printf("\nJMBG:\t\t\t%s", polaznici[i].jmbg);
                printf("\n-----------------------------------");
            }
        }
        printf("\n Zelite li da nastavite (D/d)?\t");
        scanf("%c", &da);
    } while ((da == 'D')||(da == 'd'));
    return 0;
}
```