SLOŽENE STRUKTURE PODATAKA

Dragan dr Dinu

STATIČKE STRUKTURE PODATAKA

- U okviru kursa koriste se statičke strukture podataka koje se čuvaju u operativnoj memoriji
- Statičke jer im se ne dodaju novi elementi, niti im se uklanjaju stari
- Dozvoljeno je samo menjati i čitati vrednost osnovnog elementa
- Pristup elementu statičke strukture je direktan i proizvoljan
- U zavisnosti od načina pristupanja elementima:
  - **Indeksirane strukture**, elementima se pristupa na osnovu njihove pozicije u statičkoj strukturi (indeksa) – nizovi, matrice, isl.
  - **Slogovi**, elementima se pristupa na osnovu njihovog naziva u strukturi – struct, union, polja bitova
- Korišćene strukture u kursu do sada su statičke strukture – **skalari**

NIZ

NIZ – DEKLARISANJE

- Niz predstavlja kolekciju elemenata istog tipa
- Zauzima kontinuirani niz bajtova u memoriji
- Deklariše se na sledeći način:

  tip identifikator [br_elem];
  - tip određuje - kog tipa će biti elementi niza
  - identifikator - definiše naziv niza
  - br_elem – definiše koliko će elemenata niz sadržati
- Količina memorije koju će niz zauzeti računa se kao
  sizeof(tip) * br_elem
  sizeof - operator za određivanje količine memorije koju zauzima neki tip podatka

NIZ – DEKLARISANJE PRIMERI

- Primer deklarisanja niza:

  int nizI[5];
  niz od 5 elemenata tipa int

  float nizII[12];
  niz od 12 elemenata tipa float

- **Broj elemanata niza mora biti neposredni operand ili konstantna vrednost ! Zašto?**

  #define MAXINT 100

  int nizIII[MAXINT];

NIZ – DEFINISANJE

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

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

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

NIZ – DEFINISANJE PRIMERI

- Primer definisanja niza:

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

  float nizII[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 nizIII[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 nizIII[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

  0 ≤ ind < br_elem

- 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:

```
intI[0] = 12;

nizI[1] = 2;

nizI[3] = nizI[0] * nizI[0];

a = nizI[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 < br_elem; i++)
    nizI[ i ] = ei;
```

- **for** iteraciju se koristi i za prolaz kroz sve elemente niza (ili jednog dela niza)

UNOS ELEMENATA NIZA SA TASTATURE

- Mora se unositi svaki element 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 minimum 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 MAXELNIZA 50

int main()
{
   int N = 1, i, suma = 0;
   int NizPrirordnihBrojeva[MAXELNIZA];

   printf("\n Program za racunanje sume elemenata niza N prirodnih brojeva.\n\n");

   printf("Unesite N:\t");                                  // prihvati broi 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", &NizPrirordnihBrojeva[ i ]);
   }

   for (i = 0; i < N; i++)                                  // izracunaj sumu elemenata niza
      suma += NizPrirordnihBrojeva[ 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 kao pokazivači
- Naziv niza je pokazivač na prvi element niza (prvu lokaciju u nizu kontinuiranih lokacija)
- Tako da:

int x[10]; **int *px;**	**deklarisanje niza x**
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;**

MATRICA

MATRICA – DEKLARISANJE

- C programski jezik podržava rad sa multidimenzionalnim nizovima
- Matrica je niz nizova, deklariše se na sledeći način:

  tip identifikator [br_vrsta] [br_kolona];

  – tip određuje - kog tipa će biti elementi matrice
  – identifikator - definiše naziv matrice
  – br_vrsta – definiše koliko će vrsta sadržati matrica
  – br_kolona – definiše koliko će kolona sadržati matrica

- Ukupan broj elemenata matrice je **br_vrsta * br_kolona**

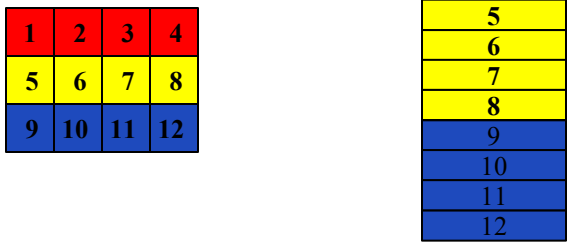
MATRICA – DEKLARISANJE PRIMERI

- Slično nizu:

  int matrica[3][4];
  matrica koja sadrži 12 elemenata 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:



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 [br_vr][br_kol] = {{e_{11}, e_{12}, ..., e_{1br_kol}},
{e_{21}, e_{22}, ..., e_{2br_kol}},
...
{e_{br_vr1}, e_{br_vr2}, ..., $e_{br\_vrbr\_kol}$}};

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

MATRICA – DEFINISANJE PRIMERI

- Primer definisanja matrice:

```
int matrical[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 matrical[ ][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
  [indeks_vrste] [indeks_kolone]
  gde je

  0 ≤ indeks_vrste < br_vrsta
  0 ≤ indeks_kolone < br_kolona

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

- Primeri pristupanja elementima matrice:

```
matricaI[0][0] = 12;

nizI[1][0] = 2;

nizI[0][3] = nizI[0][0] * nizI[1][0];

a = nizI[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 < br_vrst; i++)
    for (j = 0; j < br_kolona; j++)
        matrical[ i ][ j ] = e_ij;
```

- **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 koji sadrži minimum 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


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

#define MAXVRSTA 10
#define MAXKOLONA 20

int main()
{
   int Nvr = 1, Nkol = 1, i, j;
   float srvr = 0;
   int MatricaPrirordnihBrojeva[MAXVRSTA][MAXKOLONA];

   printf("\n Program za racunanje srednje vrednosti elemenata matrice prirodnih brojeva.\n\n");

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

   for (i = 0; i < Nvr; i++)                                 // prihvati elemente matrice od korisnika
      for (j = 0; j < Nkol; j++)
      {
         printf("Unesite element matrice na koordinati [%d][%d]:\t", i+1, j+1);
         scanf("%d", &MatricaPrirordnihBrojeva[i][j]);
      }
   for (i = 0; i < Nvr; i++)                                 // izracunaj sumu elemenata matrice
      for (j = 0; j < Nkol; j++)
         srvr += MatricaPrirordnihBrojeva[i][j];

   srvr /= (Nvr*Nkol);                                       // izracunaj srednju vrednost elemenata matrice

   printf("\n Srednja vrednost elemenata matrice izosi %f.\n\n", srvr);   //Prikazi rezultat

   return 0;
}
```

MULTIDIMENZIONALNI NIZOVI

- Logika koja se primenjuje kod nizova i matrica proširuje se 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

STRING

STRING

- Posebna vrsta jednodimenzionalnog niza tipa **char**
- Null terminisani što znači da se svakom stringu (nizu karaktera) na kraju dopisuje završni, NULL znak: **'\0'** koji je sastavni deo string
- Zbog NULL znaka za stringg od 10 karaktera treba deklarisati niz karaktera od 11 elemenata
- Rad sa stringovima predstavlja specifičan način korišćenja niza znakova
- Za C kompajler string konstanta je svaki niz znakova između znaka navoda. Znaci između navodnika plus završni znak memorišu se u niz susednih lokacija
- Zbog završnog znaka **'C'** i **"C"** nisu isti jer je **"C"** niz od dva znaka: **'C'** i **'\0'**
- Prazan string **""** sadrži samo završni znak

STRING – DEKLARISANJE/INICIJALIZACIJA

- Prvo se mora deklarisati niz karaktera:

  char identifikator [br_elem];

- Treba voditi računa da br_elem 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 nizI [br_elem];
for (i = 0; i < br_elem; i++)
    nizI[ i ] = ei;
```

gde e_i mora biti karakter

STRING – DEFINISANJE

- Stringovi se mogu definisati korišćenjem string konstanti:

  char nizS[br_elem] = "Neki string";

  – definisan je statički niz **nizS** datom konstantom

  – moguće je izostaviti **br_elem** jer se to može odrediti iz string konstante

  char nizS[ ] = "Neki string";

- Definisanje string konstantom je ekvavelnto sledećem:

  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 nizova, naziv je adresa prvog elementa

UNOS STRINGA SA TASTATURE

- Mora se voditi računa da se rezerviše 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 nesiguran pa se ne preporučuje!**)
  - **fgets** – prihvata niz karaktera ograničene dužine iz ulazne datoteke (može i ***stdin***),
  - **gets_s** – prihvata niz karaktere ograničene dužine sa ulaza, tastature
  - **fgets** i **gets_s** 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 učitaju 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 **fscanf** i **fgets** u istom programu mora da se koristi pažljivo – **fflush (stdin)**

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, **fgets** vraća **NULL**
- Sledeće greške **gets_s** detektuje:
  - **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 8;
...
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

- 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 , 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 znake počev od karaktera na koji pokazuje parametar funkcije 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 stringovim	
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 manje karaktere od **s2**, **> 0** ako **s1** sadrži više karaktera od **s2**
strchr(s1, ch);	Vraća pokazivač na prvo pojavljivanje znaka **ch** u **s1**
strstr(s1, s2);	Vraća pokazivač na prvoj pojavljivanje podstringa **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 karakteri pojavljuje u stringu.

- **VEŽBA:**
  Učitati string, zatim omogućiti korisniku da bira neku od sledećih radnji: ispis stringa, određivanje dužine stringe, 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>
int main()
{
    char strIzraz[255];
    char karakter;
    int i = 0, brPojavljivanjaStr = 0, duzStringa = 0;
    printf("\n Program za trazenje ukupnog broja pojavaljivanja karaktera u nekom stringu.\n\n");
    printf("Unesite string:\t");
    gets(strIzraz);
    printf("\nUnesite karakter:\t");
    scanf("%c", &karakter);
    duzStringa = strlen(strIzraz);
    for (i = 0; i < duzStringa; i++)
      if (strIzraz[i] == karakter)
        brPojavljivanjaStr++;
    printf("\nKarakter \'%c\' se pojavljuje %d putan u stringu\n-\"%s\".\n", karakter,
                                                    brPojavljivanjaStr, strIzraz);
    return 0;
}
```

RAD SA STRINGOVIMA – BUFFER OVERFLOW …


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

int main()
{
    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!!!**

SLOGOVI

SLOGOVI

- Postoje 3 vrste slogova u C programskom jeziku:
  - **STRUKTURA**, klasični, nepromenljivi slogovi
  - **UNIJA**, promenljivi (varijantni) slog
  - **POLJE BITOVA**, varijanta C strukture koja omogućava pristup bitovima (flert sa programskim jezicima druge generacije)
- Zahtevaju kontinualni memorijski prostor za svoje smeštanje

STRUCT

STRUCT

- 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
- Zahtevaju kontinuiran momorijski prostor za svoje smeštanje
- Rezervisana reč **struct**
- Često se koriste u kombinaciji sa nizovima, pri čemu formiraju tabele

STRUCT - 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, te se one kasnije deklarišu

STRUCT – DEKLARISANJE PRIMER ...


```
struct racun
{
   unsigned int broj_racuna;
   char ime_prezime[51];
   float stanje;
}
```

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

```
struct racun mojRacun;
```

```
struct racun
{
   unsigned int broj_racuna;
   char ime_prezime[51];
   float stanje;
} mojRacun;
```

– Struktura za koju je daklarisana i promenljiva

... STRUCT – DEKLARISANJE PRIMER


```
struct
{
   short x;
   short y;
} tacka1;
```

– Struktura za koju nije daklarisan identifikator (posledice?)

```
typedef struct tacka
{
   short x;
   short y;
} TTacka;
```

– Deklaracija novog tipa strukture, pa da se preko njega deklarišu nove instance (promenljive) tipa date strukture:

```
TTacka koordinatniPocetak, t1;
```

STRUCT – PRISTUPANJE ELEMENTU

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

- Sintaksa:

  strucPromenljiva.element

- Primer:

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

STRUCT – INICIJALIZACIJA

- Svaki element inicijalizuje se zasebno
- Pošto svaki element može biti drugačijeg tipa a pristupa im se preko naziva, ne koriste se iteracije
- Sintaksa:

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

STRUCT – PRIMER INICIJALIZACIJE


```
mojRacun.broj_racuna = 777;

strcpy(mojRacun.ime_prezime, "Dinu Dragan");

mojRacun.stanje = 11111.00;
```

STRUCT – PRIMER

- **ZADATAK:**
  Napraviti program koji prihvata podatke o polaznicima kursa (ime, prezime, jmbg, grad). Podaci se smeštaju u niz, može biti maksimalno 40 polaznika. Korisniku se omogućuje prikaz svih polaznika iz istog grada koji zadaje korisnik. Omogućiti korisniku da ponavlja datu radnju sve dok ne odabere kraj programa.

- **VEŽBA:**
  Proširiti prethodni primer tako da se omogući pretraživanje i po drugim kriterijumima (ime, prezime, jmbg).

STRUCT-URA – ZADATAK ...


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

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

struct Polaznik
{
    char Ime[MAXCHAR];
    char Prezime[MAXCHAR];
    char jmbg[MAXJMBG];
    char grad[MAXCHAR];
};

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

    struct Polaznik polaznici[MAXPOLAZNIKA];

    int brPolaznika = 0;
    int i = 0;

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

... STRUCT-URA – ZADATAK ...


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

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("-----------------------------------");
   }
```

... STRUCT-URA – ZADATAK


```
do
   {
      printf("\n Unesite grad za koji zelite da izlistate polaznike:\t");
      scanf(" %[^\t\n]s",traziGrad);
      fflush(stdin);

      printf("\nIz datog grad 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);
      fflush(stdin);

   } while ((da == 'D')||(da == 'd'));

   return 0;
}
```

STRUKTURE I POKAZIVAČI

- Vrlo čest slučaj da se deklariše pokazivač na strukturu:

  struct strucPromenljiva *pokStrukt;

- Te se za pristup elementu koristi zaseban operator, **->**:

  pokStrukt->elementStrukture1

- Što je skraćeni zapis sledećeg:

  (*pokStrukt).elementStrukture1

- Primer:

```
struct racun *mojRacun;
mojRacun->broj_racuna = 777;
strcpy(mojRacun->ime_prezime, „Dinu Dragan");
mojRacun->stanje = -7.7f;
```

UNION

UNION

- Statička struktura varijantnog (promenljivog) polja
- Omogućuje da se različiti tipovi skladište unutar iste memorijske lokacije
- Može da sadrži više elemenata (članova), ali u jednom trenutku samo jedan član može da sadrži vrednost
- Predstavlja efikasan način da se jedna memorijska lokacija koristi u različite svrhe
- Rezervisana reč **union**
- Memorijska lokacija u kojoj se čuva unija zauzima onoliko prostora koliko i najveći tip elementa unije, tako da ako unija sadrži **char**, **int** i **float** unija će zauzimati onoliko bajtova koliko je potrebno za memorisanje **float** tipa (**nije char+int +float**)

UNION - DEKLARISANJE

- Unija se deklariše na sledeći način:

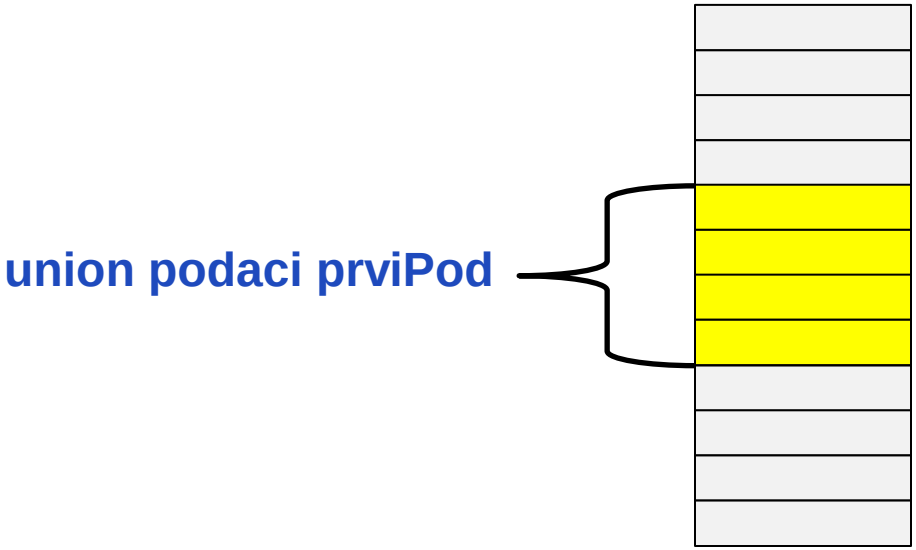
```
union [identifikator]
{
    deklaracija elementa1;
    deklaracija elementa2;
    ...
    deklaracija elementaN;
} [jedna ili više promenljivih date strukture];
```

- Sve što je važilo za upotrebu identifikatora, deklarisanje promenljivih i deklarisanje tipova strukture, važi i za uniju

UNION – DEKLARISANJE

- Ako se pretpostavi da char zauzima 1 bajt, int zauzima 2 bajta, a float 4 bajta, u tom slučaju će unija zauzimati 4 bajta:

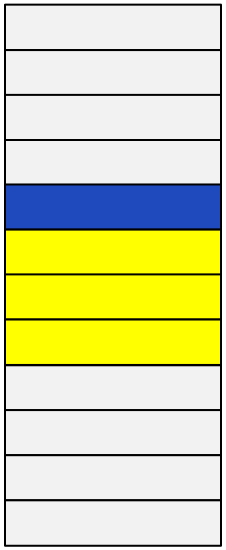
```
union podaci
{
    char znak;
    int ceo_broj;
    float raz_broj;
} prviPod;
```



UNION – INICIJALIZACIJA

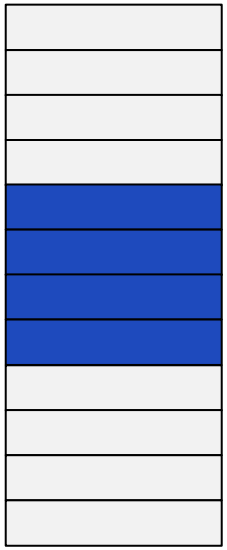
- Elementu unije pristupa se na isti način kao i elementu sktrukture preko operatora . (tačka)
- Samo jedan element u jednom trenutku može da bude inicijalizovan

prviPod.znak = 'a';



prviPod.ceo_broj = 10;

prviPod.raz_znak = 9,9f;



POLJA BITOVA

TIP POLJA BITOVA

- Omogućuje da se u strukturi označi koliko koje polje zauzima bita
- Pristupa se na nivou bita
- Kao i kod obične strukture navode se elementi i njihov tip
  - svaki elemenat se može podeliti na više polja koja zauzimaju određen broj bitova
  - broj bitova svakog polja eksplicitno se navodi
  - ukupna suma bitova koje zauzimaju polja jednog elementa može biti manja ili jednaka veličini tipa elementa kojem polja pripadaju
  - podacima se pristupa preko naziva polja
- Predstavlja vezu prema programskim jezicima prve i druge generacije
- Treba voditi računa o prenosivosti koda (portability issue)

TIP POLJA BITOVA – SINTAKSA …

- Prvi način za deklarisanje polja bitova je sledeći:

```
struct [identifikator]
{
   tip     Polje1 : n1,
           Polje2 : n2,
           …
           PoljeN : nN;
}[jedna ili više promenljivih date strukture];
```

- Suma mora (**n1+n2+nN**) biti jednaka ili manja od **veličine tipa**

… TIP POLJA BITOVA – SINTAKSA

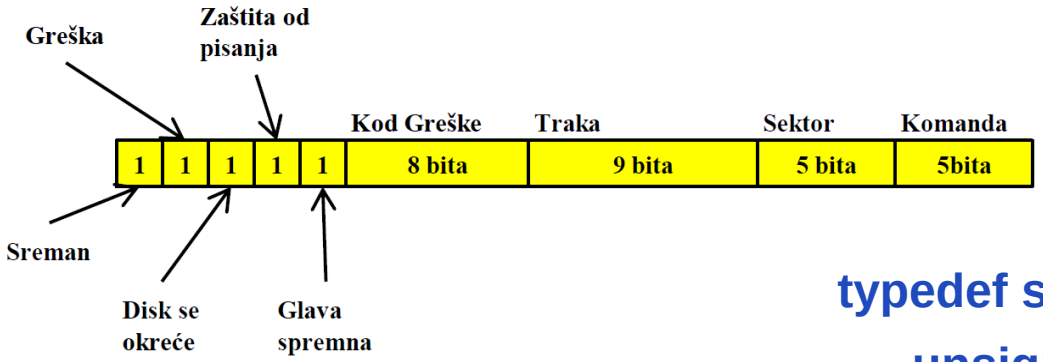
- Drugi način za deklarisanje polja bitova je sledeći:

```
struct [identifikator]
{
   tip     Polje1 : n1;
   tip     Polje2 : n2;
           …
   tip     PoljeN : nN;
}[jedna ili više promenljivih date strukture];
```

- U ovom slučaju ne mora se voditi računa o tome da li je suma (**n1+n2+nN**) jednaka ili manja od veličine **tipa**, ali ako je suma veća od veličine jednog **tipa**, zauzeće se u memoriji dovoljno prostora da stanu **dva tipa** do svoje pune veličine

TIP POLJA BITOVA – PRIMER …

- Primer registra UI kontrolera hard diska čija ukupna veličina ne prelazi 32 bita (jedan integer u MS VS...), npr.




```
typedef struct RegistarHD {
    unsigned    spreman : 1;
    unsigned    greska : 1;
    unsigned    disk_se_okrece : 1;
    unsigned    zastita_od_pisanja : 1;
    unsigned    glava_spremna : 1;
    unsigned    kod_greske : 8;
    unsigned    traka : 9;
    unsigned    sektor : 5;
    unsigned    komanda : 5;
} THDRegistar;
```

… TIP POLJA BITOVA – PRIMER

- Ako se pretpostavi da postoji DISK_REGISTER_MEMORY u operativnom sistemu (u kojoj se nalazi adresa registra za upravljanjem diska), onda se za pristup disku može napraviti promenljiva **disk_reg**

```
THDRegistar *disk_reg = (THDRegistar *) DISK_REGISTER_MEMORY;
// definisati sektor i traku iz koje se čitaju podaci, kao i tip operacije – READ
disk_reg–>sektor = novi_sektor;
disk_reg–>traka = nova_traka;
disk_reg–>komanda = READ;
// čekanje na kraj operacije, kada disk postaje spreman
while (! disk_reg–>spreman);
// Provera da li je došlo do greške
if (disk_reg–>greska) {
    // otkrivanje vrste greške na osnovu njenog koda
    switch(disk_reg–>kod_greske)
    .....
}
```

DINAMIČKA POLJA

Dragan dr Dinu

DINAMIČKA POLJA U C JEZIKU ...

- Kada postoji potreba za dinamičkom alokacijom homogenog prostora
- Veličina niza se alocira dinamički na zahtev programa u skladu sa potrebama korisnika
- Alokacija dinamičkog niza išla bi slično primeru:

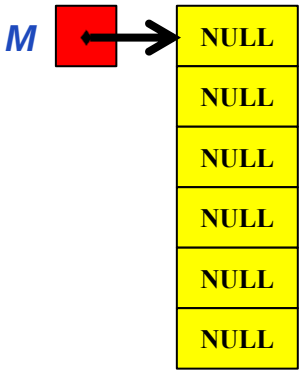
```
int i, vel = 5;
int* Niz = (int*) calloc(vel, sizeof(int) );

if (Niz == NULL)
   printf("Nije uspela alokacija memorije.\n")

else{
   // rad sa dinamičkim nizom
   for (i = 0; i < vel; i++)
        Niz[ i ] = /* nesto */ ...
        ...
}
...
free(Niz);
```

... DINAMIČKA POLJA U C JEZIKU

- Dinamička matrica (primer iz knjige prof. dr D. Ivetića):

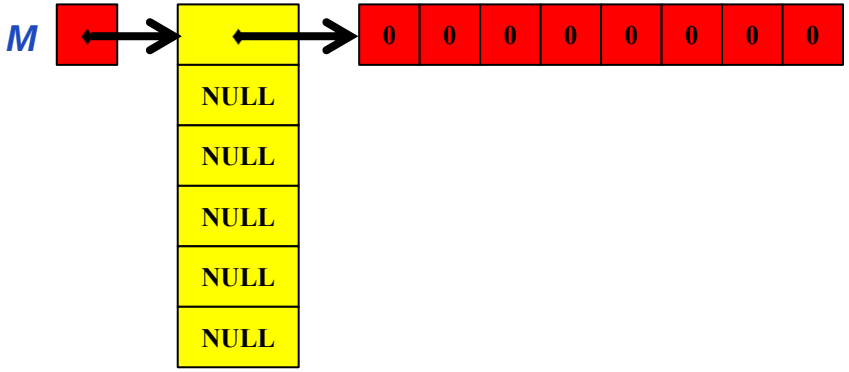
(a) Kreiranje osnove za vel=6

M = (int) calloc (vel, sizeof (int*));**



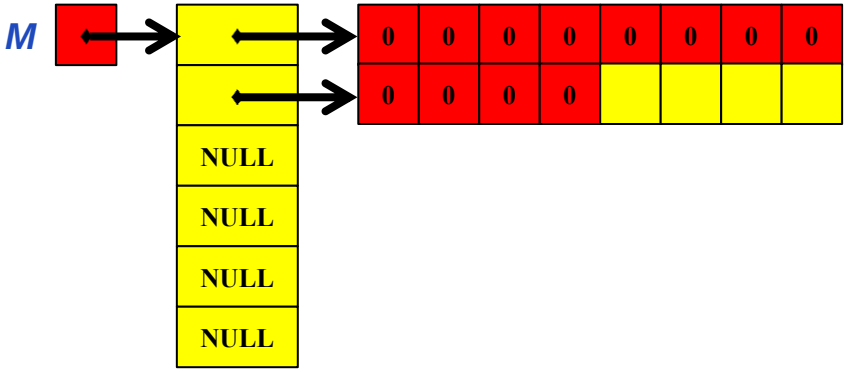
(b) Prva vrsta od K1=8 elemenata

M[0] = (int*) calloc (K1, sizeof (int));



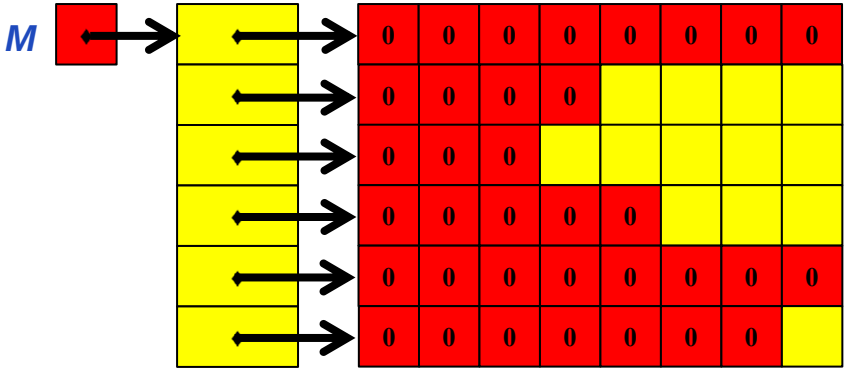
(c) Druga vrsta od K2=4 elementa

M[1] = (int*) calloc (K2, sizeof (int));



(d) vel-ta vrsta od Kvel=7 elemenata

M[vel–1] = (int*) calloc (Kvel, sizeof (int));



DINAMIČKA POLJA – PRIMER VEKTOR ...


```
// vektor.h
// vektor koji u sebi sadrzi celebrojeve

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

#define VEKTOR_INICIJALNA_VELICINA 100

typedef struct {
  unsigned int velicina;      // velicina vektor
  unsigned int kapacitet;
  int *podaci;  // niz celih brojeva
} TVektor;

unsigned short vektor_inicijalizacija(TVektor *vektor);
unsigned short vektor_prosiri(TVektor *vektor, int vrednost);
int vektor_get(TVektor *vektor, unsigned int indeks);
unsigned short vektor_set(TVektor *vektor, unsigned int indeks, int vrednost);
unsigned short vektor_dovoljan_kapacitet(TVektor *vektor);
void vektor_oslobodi(TVektor *vektor);
```

... DINAMIČKA POLJA – PRIMER VEKTOR ...


```
// vector.c
#include "vektor.h"

unsigned short vektor_inicijalizacija(TVektor *vektor) {
  // inicijalna velicina i kapacitet
  vektor->velicina = 0;
  vektor->kapacitet = VEKTOR_INICIJALNA_VELICINA;

  // alociraj memoriju za vektor
  if (vektor->podaci = (int*)calloc(vektor->kapacitet, sizeof(int)))
      return 1; // uspeo je
  else
    return 0; // nije uspeo
}


unsigned short vektor_prosiri(TVektor *vektor, int vrednost) {
  if (vektor_dovoljan_kapacitet(vektor)){
      vektor->podaci[vektor->velicina++] = vrednost;
          return 1;
  }
  else
      return 0;
}
```

... DINAMIČKA POLJA – PRIMER VEKTOR ...


```
// vector.c

int vektor_get(TVektor *vektor, unsigned int indeks) {
  if (indeks >= vektor->velicina || indeks < 0) {
      printf("Opseg vektora je %d\n", vektor->velicina);
      exit(1);
  }
  return vektor->podaci[indeks];
}


unsigned short vektor_set(TVektor *vektor, unsigned int indeks, int vrednost) {
  // postavi na nule do zeljenog indeksa
  unsigned short uspeh = 1;
  while (indeks >= vektor->velicina) {
      uspeh = vektor_prosiri(vektor, 0);
          if (!uspeh)
              break;
  }
  // ? while ((indeks >= vektor->velicina)&&(uspeh = vektor_prosiri(vektor, 0));

  // postavi vrednost elementa na zeljenom indeksu
  if (uspeh)
      vektor->podaci[indeks] = vrednost;
  return uspeh;
}
```

... DINAMIČKA POLJA – PRIMER VEKTOR ...


```
// vector.c

unsigned short vektor_dovoljan_kapacitet(TVektor *vektor) {
  unsigned short ima_mesta = 1;
  if (vektor->velicina >= vektor->kapacitet) {
     // stiglo se do kraja vektora, dupliraj kapacitete
     vektor->kapacitet *= 2;
     if ((vektor->podaci = (int*)realloc(vektor->podaci, sizeof(int) * vektor->kapacitet))==NULL)
        ima_mesta = 0;
     }
  return ima_mesta;
}


void vektor_oslobodi(TVektor *vektor) {
  free(vektor->podaci);
};
```

DINAMIČKA POLJA – PRIMER VEKTOR ...


```
// main.c

#include "vektor.h"


int main() {
  TVektor vektor;

  unsigned int i;

  if (vektor_inicijalizacija(&vektor)){

     for (i = 105; i > 95; i--) {
        vektor_prosiri(&vektor, i);
     }

     if (vektor_set(&vektor, 5, 1023532))
        printf("\nUspelo dodavanje novog elementa (%d) u vektor.", vektor_get(&vektor, 5));
     else
        printf("\nNije uspelo dodavanje novog elementa u vektor.");

     vektor_oslobodi(&vektor);
  }

  return 0;
};
```