1. Promenljive, tipovi podataka, operatori

Tipovi podataka

Četiri osnovna tipa pored kojih se mogu pojaviti **modifikatori**

1. char (mod. Unsigned)
2. int (mod. short long unsigned)
3. float
4. double(mod. long)

Konstante

1. **Celobrojne const** – 2, 200000L
2. **Razlomljene const** – 3.14
3. **Heksadecimalne const** – 0xF, 0xFF
4. **Oktalne konstante** – 012
5. **Znakovne konstante** - ‚a' ‚\n' '\xxx' gde je xxx oktalni ASCII kod karaktera
6. **String konstanta** – "neki tekst"

Tip	Dužina	Opseg
unsigned char	8 bits	0 to 255
char	8 bits	-128 do 127
enum	16 bits	-32,768 do 32,767
unsigned int	16 bits	0 do 65,535
short int	16 bits	-32,768 do 32,767
int	16 bits	-32,768 do 32,767
unsigned long	32 bits	0 do 4,294,967,295
long	32 bits	-2,147,483,648 do 2,147,483,647
float	32 bits	3.4 * (10**-38) do 3.4 * (10**+38)
double	64 bits	1.7 * (10**-308) do 1.7 * (10**+308)
long double	80 bits	3.4 * (10**-4932) do 1.1 * (10**+4932)

#define direktiva služi i za definiciju konstanti npr **#define PI 3.14**

Nabrojivi tip

Ključna reč je **enum**
Sintaksa: enum ime {prvi_član[=vrednost], drugi_član[=vrednost]}
npr
enum mesec {JAN = 1, FEB …. , DEC}
upotreba
enum mesec tekuci;
tekuci = JAN;

Promenljive

- Deklarišu se kao instance nekih od navedenih tipova.
Npr int a.
Naziv lokacije za skladištenje podataka.

Operatori

1. Aritmetički operatori: +. -. *, / , %. Možemo umesto x=x+1 koristiti i x+=1
Automatski inkrement i dekrement pre i posle promenljive

2. Relacioni i logički operatori:
- Ne postoji boolean tip
Logički je sve tačno ono što je veće ili jednako sa 1, sve što je manje od 1 je netačno
Relacioni: > <, <= , >=, ==, !=
Logički: &&, ||, !
- Short – circuiting: Ako je operacija && i prvi izraz je netačan ostale neće ni računati, za ili ako je prvi tačan ostale neće ni računati(ceo izraz je tačan)

3. Bit operatori
1. **Logičko I** nad bitovima: &
2. **Logičko ILI** nad bitovima: |
3. **Ekskluzivno ILI(XOR)** nad bitovima: ^
4. **Logička negacija nad bitovima**: - Unarni operator ~
5. **Kombinacija sa** =: &= |= ^=
6. **Shiftovanje:** a>>b – pomera bitove **u a za b mesta**, ako je a pozitivan ubacuje 0, ako je negativan 1, a << b, pomera bitove ulevo i ubacuje 0

4. Konverzije tipova
- Možemo raditi prilikom dodele vrednosti ili u toku računanja
Kast operator za eksplicitnu konverziju tipova
(drugi tip)izraz
npr
int i;
i = (int)3.74 + (int)2.6;

2. Funkcije printf() i scanf()

printf()
- Funkcija za štampanje na ekran i iz biblioteke stdio.h je koja mora da se #include
- **Prvi parametar** je **specifikator formata ispisa**, a ostali parametri su varijable čija se vrednost štampa.
Specifikator formata:
%[indikator][širina][.preciznost][F|N|h|l|L]tip

1. [indikator] – definiše kako se štampa vrednost
- : Levo poravnanje rezultata, popunjava desnu stranu prazninama. Ako nije dato, poravnanje je desno a popunjava levu stranu sa nulama ili prazninama
+: Označena(signed) konverzija rezultata uvek počinje sa + ili sa – znakom

2.[širina] – definiše broj cifara
n: štampa zadati broj cifara
0n: štampa zadati broj cifara, ako nema toliko cifara sa leve strane dodaje nule

3. [.preciznost] – broj decimala
(none) – Preciznost se postavlja na default – 6 za sve e, E, F tipove, Štampa do kraja stringa…
.n – n decimala se štampa
-Ako ima više od n decimala, rezultat je zaokružen, Ako je string u pitanju štampa n karaktera.

4. Modifikator [F|N|h|l|L] – definiše kraći ili duži tip podataka h,short int, l – long int double, L long double…
5. tip – brojevi

Tip	Očekivan ulaz	Format rezultat
Brojevi		
d	Integer	Signed decimal integer
i	Integer	Signed decimal integer
o	Integer	Unsigned octal integer
u	Integer	Unsigned decimal integer
x	Integer	Unsigned hexadecimal int (with a, b, c, d, e, f)
X	Integer	Unsigned hexadecimal int (with A, B, C, D, E, F)
f	Floating point	Signed value oblika [-]dddd.dddd.
e	Floating point	Signed value oblika [-]d.dddd or e[+/-]ddd
g	Floating point	Kraći zapis od %f i %e
E	Floating point	Isto kao e; samo ima 'E' za eksponent
G	Floating point	Isto kao e; samo ima 'E' za eksponent ako je e format

```
printf("celobrojni: %d\n", c);      -->     celobrojni: 356
printf("celobrojni: %6d\n", c);     -->     celobrojni:    356
printf("celobrojni: %-6d\n", c);    -->     celobrojni: 356
printf("celobrojni: %+6d\n", c);    -->     celobrojni:   +356
printf("celobrojni: %+6d\n", -c);   -->     celobrojni:   -356

printf("razlomljeni: %f\n", 3.141);     --> razlomljeni: 3.141000
printf("razlomljeni: %6.2f\n", 3.141); --> razlomljeni:   3.14
printf("razlomljeni: %e\n", 3.141);     --> razlomljeni: 3.141000e+00
printf("razlomljeni: %6.2e\n", 3.141); --> razlomljeni: 3.14e+00
printf("razlomljeni: %g\n", 3.141);     --> razlomljeni: 3.141
```

Tip	Očekivan ulaz	Format rezultat
Karakteri		
c	Character	Jedno slovo
s	String	Štampa string do kraja ili do zadatog broja slova
%	Ništa	Štampa znak %

Učitavanje sa tastature
scanf()- takođe iz stdio.h
- **Prvi parametar** je **specifikator formata unosa**, a ostali parametri su **adrese varijabli** u koje se smešta vrednost.
- Specifikator formata:
%[indikator][širina][.preciznost][F|N|h|l|L]tip

- **Ako se učitava celobrojna vrednost**: scanf(“%d”, &i);
- Ako želimo da učitamo više od jedne promenljive: scanf(“%d %f”, &i, &j);
- **Separator** između dva broja je **whitespace** (space, tab, enter).
• Separator se može i navesti: scanf(“%d:%f”, &i, &j); >>1:25

3. Naredbe

1. If else
Opšta sintaksa:
if(uslov_1)
telo_1
else if(uslov_2)
telo_2
else
telo_3

```
if (poeni > 89)
  ocena = 5;
else if (poeni > 74)
  ocena = 4;
else if (poeni > 59)
  ocena = 3;
else if (poeni > 44)
  ocena = 2;
else ocena = 1;
```

2. Ternarni operator
a = (i < 10) **?** i * 100 **:** i * 10;

isto kao:
```
if (i < 10)
        a = i * 100;
else
        a = i * 10;
```

3. Switch
- Izraz u switch() izrazu mora da proizvede **celobrojnu vrednost.**
- Ako ne proizvodi celobrojnu vrednost, ne može da se koristi switch(), već if()!
- Ako se izostavi **break**; propašće u sledeći case.
- Kod default izraza ne mora break - to se podrazumeva

```
switch(ocena)
{
        case 5: printf(“odlican\n”); break;
        case 4: printf(“vrlo dobar\n”); break;
        case 3: printf(“dobar\n”); break;
        case 2: printf(“dovoljan\n”); break;
        case 1: printf(“nedovoljan\n”); break;
        default: printf(“nekorektna ocena”);
}
```

4. while
- Za **cikličnu strukturu** kod koje se samo **zna uslov za prekid**.
- Telo ciklusa ne mora ni jednom da se izvrši
- **Opšta sintaksa:**

```
while (uslov)
        telo
```

```
int r = 0;
while(r < 50) {
        r = random(100);
        printf(“%d\n”, r);
}
```

- Važno: izlaz iz petlje na false

5. Do while
- Za **cikličnu strukturu** kod koje se samo **zna uslov za prekid**.
- Razlika u odnosu na while petlju je u tome što se telo ciklusa izvršava makar jednom.

Opšta sintaksa:

```
do
        telo
while (uslov);
```

```
int i = 0;
do {
        printf(“%d\n”, i++);
} while (i < 10)
```

6. for
- Za organizaciju petlji kod kojih se **unapred zna koliko puta** će se izvršiti telo ciklusa.
- Petlja sa početnom vrednošću, uslovom za kraj i blokom za korekciju.

- **Opšta sintaksa:**
for (**inicijalizacija; uslov; korekcija)**
telo

for (i = 0; i < 10; i++)
printf(“%d\n”, i);

- može i višestruka inicijalizacija i step-statement:
for(i = 0, j = 1; i < 10 && j != 11;i++, j++)
• oprez (može da se ne završi):
double x;
for (x = 0; x != 10; x+=0.1) …

break – prekida telo tekuće ciklične strukture (ili case dela) i izlazi iz nje. Može i za izlaz iz ugnježdene petlje
continue – prekida telo tekuce ciklične strukture i otpočinje sledeću iteraciju petlje.

```
for(i = 0; i < 100; i++)
{
        // Out of for loop
        if(i == 74) break;
        // Next iteration
        if(i % 9 != 0) continue;
        printf(“%d\n”, i);
}
```

4. Nizovi i stringovi

Nizovi
- Opšta sintaksa za definiciju:
tip_podatka ime[veličina];

int a[10];
- **Pristup elementima niza** – operator []:
i = a[2];
a[3] = 5;
- **Indeksi** u nizovima kreću od 0!!!
– prvi element ima indeks 0
– poslednji element ima indeks (veličina – 1)

Inicijalizacija niza
- **Opšta sintaksa:** tip_podatka ime[veličina] = {$vred_0$, $vred_1$, ... $vred_{n-1}$}
int dani_u_mesecu[12] = {31, 28, 31, 30, 31, 31, 30, 31, 30, 31};

- Broj elemenata za inicijalizaciju može da bude **manji od veličine niza.**
– ostatak se inicijalizuje nulama

Dimenzija niza se može izostaviti ako se niz inicijalizuje
– veličina niza je broj elemenata za inicijalizaciju

Višedimenzionalni nizovi

- Opšta sintaksa: tip_podatka ime[dimenzija1][dimenzija2]...[dimenzijan]
float tabela[5][12];

Inicijalizacija:
Opšta sintaksa:
tip_podatka ime[dimenzija1][dimenzija2]...[dimenzijax] =

```
int i[2][3] = {
        {1, 2, 3},
        {4, 5 ,6}
};

int i[2][3][4] = {
        {
                {1, 2, 3, 4},
                {5, 6, 7, 8},
                {9, 10, 11, 12}
        },
        {
                {13, 14, 15, 16},
                {17, 18, 19, 20},
                {21, 22, 23, 24}
        }
};
```

Stringovi
- Stringovi su nizovi karaktera.
- Stringovi se **završavaju znakom '\0'**, što je znak sa ASCII kodom 0
- char ime1[] = "Mika";
char ime11[] = {'M', 'i', 'k', 'a', '\0'}

Učitavanje stringova
- Funkcija **scanf("%s", s)** učitava string do **prvog whitespace,** što može dpredstavlja problem ako u rečenici ima razmaka.
- Funkcija **gets(s)** učitava string do znaka "novi red", koji se dobija kada se pritisne ENTER.

```
i = 0;
j = 0;
while(s[i] != 0){
        if (s[i] == c)
                j++;
        i++;
}
```

5. Funkcije

• Motivacija:
– ponavljanje koda
– dekompozicija na manje celine
• **Osnovni elementi:**
1. opciona deklaracija
2. definicija
3. poziv

Opšta sintaksa:

```
povratni_tip ime_funkcije(parametri)
{
...
}
```

1. Povratni tip je bilo koji tip podatka ili void ako funkcija ne vraća vrednost.
– funkcija vraća najviše jednu vrednost!

2. Parametri se deklarišu na isti način kao i promenljive.
– Ako funkcija nema parametara, stavlja se void ili se ostave prazne zagrade (zastarelo).

3. Ako funkcija vraća vrednost, to se postiže **return naredbom:**
– return a;
– return (a)

```
int max(int a, int b)
{
        if (a > b)
                return a;
        else
                return b;
}
```

Deklaracija funkcija

- Ako koristimo funkciju pre nego što je definisana, moramo da je deklarišemo.
- **Opšta sintaksa:**

```
povratni_tip ime_funkcije(parametri);
```

```
double f(int i);
void main(void)
{
        double x;
        x = f(5);
}
double f(int i)
{
        return i*2.0;
}
```

- Ako se **funkcija ne deklariše i ne definiše iznad mesta poziva,** prva upotreba funkcije će se smatrati deklaracijom:
  1. **podrazumevaće se da funkcija vraća int,** jer se na osnovu prve upotrebe ne može zaključiti šta funkcija vraća, pa će se deklarisati bez povratnog tipa, a tada se po definiciji smatra da vraća int
  2. **ne vodi se računa o parametrima funkcije**
  3. ovo ne predstavlja problem ako funkcija vraća int, ali predstavlja problem ako funkcija vraća bilo šta drugo

Parametri funkcija
- Parametri funkcija se prenose po **vrednosti ili po referenci.**
- Prenos parametara po vrednosti je podrazumevani način prenosa u C-u

1. Prenos parametara po vrednosti:
- **prave se kopije parametara** i te kopije se prosleđuju funkciji
- **posledica**: nije moguće promeniti prosleđenu promenljivu iz funkcije

2. Prenos parametara po referenci:
- Funkcija se deklariše da prima pokazivač na parametar:

void f(**int *i**)
{
...
}

- Prilikom poziva funkcije, prosleđuju se adrese parametara, a ne parametri, oblika: f(**&i**)
- Prilikom korišćenja parametara unutar funkcije, koristi se oblik: ***i**

```
void f(int *i)
{
  *i = 3;
}
void main()
{
  int i = 5;
  f(&i);
  printf("%i", i);
}
```

Nizovi kao parametri funkcija
- U listi parametara funkcije ne navode se dimenzije.
• Primer:

void f(**int a[]**)
{
    ...
    a[3] = 5;
}

- Ovo će promeniti a[3] u pozivajućoj funkciji!

POSLEDICA:
• **Niz se može promeniti iz funkcije!**
• Nizovi se ne prosleđuju po vrednosti, tj. **ne pravi se kopija niza**
- Ako je potrebna veličina niza, ona se mora proslediti – funkcija ne može da zna koliko je velik niz

Višedimenzionalni nizovi kao parametri funkcija
- U listi parametara funkcije t**reba navesti najmanje n-1 dimenziju** (n je broj dimenzija)

void f(int a[][10])
{
    ...
    a[3][3] = 5;
}

Opseg vidljivosti promenljivih
- Promenljive deklarisane **unutar funkcije** se "vide" samo u funkciji.
- Pomenljive deklarisane **izvan funkcije se** "vide" i u **ostalim funkcijama** koje su **definisane ispod nje** – to su **spoljašnje promenljive**

Ključna reč static
- Ako je promenljiva **spoljašnja i statička**, onda se "vidi" samo **unutar tekuće datoteke (modula).**
  - svaka **spoljašnja promenljiv**a može da se "vidi" iz drugih modula, ako nije static
  – **static** u ovom slučaju predstavlja **zaštitu pristupa spoljašnjoj promenljivoj.**

- Ako je **promenljiva lokalna (unutar funkcije) i statička**, njena se **vrednost čuva i po izlasku iz funkcije**
  posledica: po ponovnom ulasku u funkciju, **statička promenljiva ima vrednost iz prethodnog poziva.**

Rekurzivne funkcije
- Rekurzija: funkcija koja poziva samu sebe.
- Svaka rekurzivna funkcija mora da ima uslov za izlaz iz rekurzije!

Pozitivno:
– razumljivije
– ponekad i jedino moguće(Akermanova funkcija)

Mana:
– opterećuje stek
– brzina

```
int fakt(int n)
{
  if (n < 2)
    return 1;

  return n * fakt(n-1);
}
```

II grupa

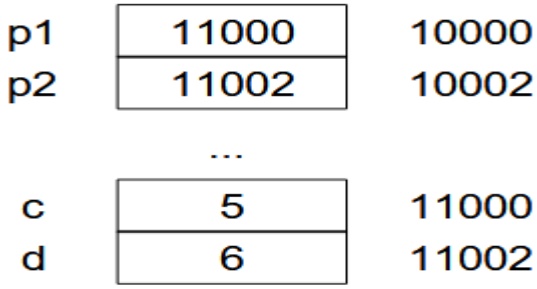
1. Pokazivači
- Pokazivač je adresa u memoriji, broj.
  1. ukazuje na lokaciju u memoriji
  2. sadržaj pokazivačke promenljive je adresa

Pokazivačke promenljive pokazuju na **druge promenljive** ili na **početak memorijskog bloka**
  - pokazivači na promenljive tipa int, float, itd

Deklaracija:
int ***p1, *p2;

Dodela vrednosti: dodelile su se adrese
p1 = &c; p2 = &d

Pristup lokaciji:
  a = *p1;
dereferenciranje *p2 = 6; na adresi koja se nalazi u pokazivacu p2 se smešta vrednost 6, tj promenljivoj c se dodeli vrednost 6



Specijalna konstanta NULL
- Ako pokazivačka promenljiva ima vrednost NULL, onda ne pokazuje ni na šta.

```
int *p;
p = NULL;
```

Neinicijalizovana vrednost NEMA NULL!
– mora se eksplicitno inicijalizovati na NULL

Operacije sa pokazivačima
Sabiranje/oduzimanje sa brojem:

```
int *p;
p = p + 10; adresa plus deset
p++;
```

• **Poređenje:**

```
int *p1, *p2;
...
if (p1 == p2) ako adrese koje su upisane u memoriji na koju pkazuje su iste
```

Operator dodele vrednosti
Dodelom vrednosti dobijamo da oba **pokazivača ukazuju na isto.**

```
int a = 5;
int *p1, *p2;
p1 = &a;
p2 = p1;
```

p1	11000	10000
p2	11000	10002
	...	
a	5	11000

Pokazivači i argumenti funkcija
Za **prosleđivanje parametra po referenci** koriste se pokazivači:

```
void swap(int *a, int *b)
{
        int t = *a;
        *a = *b;
        *b = t;
}
...
int x = 5, y = 6;
swap(&x, &y);
```

Pokazivači i nizovi
- Svaki niz je zapravo pokazivač na prvi element u memoriji.
- int a[20];

```
int *pa;
```

pa = a; je isto što i: **pa = &a[0]; adresa prvog elementa u nizu**

ovo je samo po adresama
a+2 == &a[2] == pa+2

pristupanje vrednosti dereferenciranje skidamo referencu sa *
***(a+3) == a[3] == *(pa+3)**

Pokazivači i nizovi kao parametri funkcija

- Niz kao parametar funkcije se **prosleđuje po referenci** time što je ime niza zapravo adresa prvog elementa

```
int strlen(char a[])          int strlen(char *a)
{                             {
   ...                           ...
}                             }

      strlen("tekst"); /* string konstanta */
      strlen(niz);     /* char niz[100]    */
      strlen(ptr);     /* char *ptr        */
```

Pokazivači i višedimenzionalni nizovi

• Višedimenzionalni nizovi se u memoriji čuvaju u linearnom poretku (npr.dvodimenzionalni nizovi se čuvaju kao niz vrsta, tj. vrstu po vrstu)

```
int a[3][2];
int *p;
p = a;
```

```
p    == &a[0][0]
p+1  == &a[0][1]
p+2  == &a[1][0]
p+3  == &a[1][1]
p+4  == &a[2][0]
p+5  == &a[2][1]
```

Nizovi pokazivača

Opšta sintaksa:

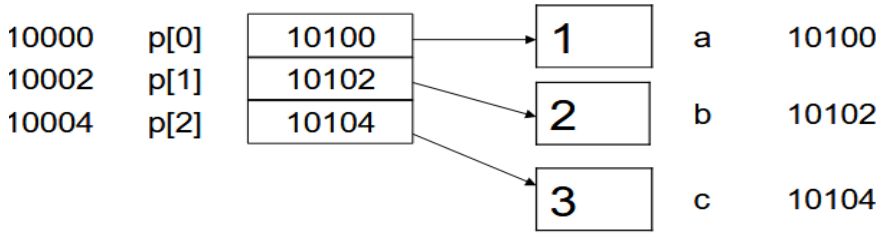
tip *pok[vel];

• Primer:

int a=1, b=2, c=3;

int ***p[3]***; p[0] = &a;

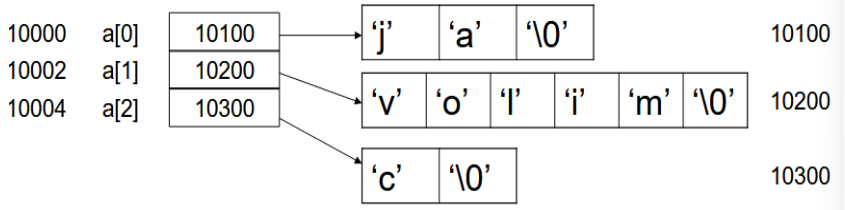
p[1] = &b; p[2]=&c;



Niz pokazivača na karakter

- Pokazivač na karakter je string.
- Niz pokazivača na karakter je niz pokazivača na string

```
char *a[3] = {"ja", "volim", "c"};
```



Pokazivači na pokazivače

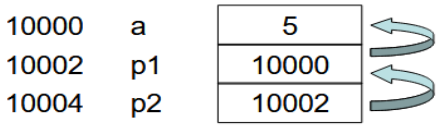
Opšta sintaksa:

tip_podatka **ime;

Sadržaj takve promenljive je broj – **adresa u memoriji** gde se nalazi **pokazivač**

• Primer:

```
int a = 5;
// u p1 je adresa od a
int *p1 = &a;
// u p2 je adresa od p1
int **p2 = &p1;
```



Pokazivači na funkcije
- Opšta sintaksa za deklaraciju:
povratni_tip (*ime)(parametri);

- Poziv funkcije preko pokazivača: pokfunc.c
(*ime)(parametri);
Zagrade kod imena su obavezne!
– Ako se izostave, smatraće se da je u pitanju funkcija koja vraća pokazivač na povratni tip.

Alokacija memorije
- Pokazivačke promenljive pokazuju na blok alocirane memorije
    - pokazuju na prvi bajt bloka alocirane memorije
    - u zavisnosti od tipa pokazivača, blok se tretira kao blok int-ova, char-ova, itd.

- Koristi se kada se u toku pisanja programa ne zna koliko memorije je potrebno.
- Alocirana memorija se mora **dealocirati!**
- Alokacija se vrši sistemskom funkcijom **malloc(br_bajtova).**
- **Dealokacija** funkcijom **free(pokazivac).**

- Memorija se **alocira sa heap-a.**
– **heap** je blok memorije dodeljen svakom programu.

Memorija se uvek alocira zadatim brojem bajtova.
• Funkcija **malloc() vraća adresu bloka** u memoriji
– ako tu adresu prihvatimo u pokazivač na int, onda se taj blok memorije tretira kao niz int-ova

2. Strukture i typedef operator

- Podaci se u stvarnosti češće javljaju u složenom obliku nego samo u obliku brojeva ili slova
- Podaci o nečemu se obično sastoje iz više elemenata.

Strukture

Opšti oblik strukture:
struct oznaka_strukture
{
    tip el1;
    tip el2;
...
};
• Primer:
struct osoba
{
    char ime[50];
    char adresa[150];
    int starost;
};

Deklaracija promenljive tipa strukture:
struct osoba osoba1, osoba2;
• Korišćenje:

osoba1.starost=15;
strcpy(osoba1.ime, “Petar Petrovic”);
osoba2.starost=60;

- Promenljiva tipa strukture **zauzima onoliko memorije** koliko **svi elementi zauzimaju zajedno**.

Inicijalizacija struktura

```
struct osoba osoba1 = {
        “Petar Petrovic”,
        “Petrova ulica 3”,
        15
};
```

Operator dodele vrednosti kopira ceo sadržaj strukture u drugu.

```
struct osoba osoba2;
...
osoba2=osoba1;
```

Istovremena deklaracija i definicija

Moguća je istovremena deklaracija i definicija:

```
struct osoba {
        char ime[50];
        char adresa[150];
        int starost;
} os;
```

Nizovi struktura

Opšti oblik:

```
struct str ime[veličina];
```

```
struct osoba osobe[3];
```

osobe[2].ime pristup imenu druge treće osobe u nizu

• Inicijalizacija:

```
struct osoba osobe[3] =
{
        { "Petar Petrovic", "Petrova ulica 3", 15},
        { "Mika Mikic", "Mikina ulica 4", 25},
        { "Djura Djuric", "Djurina ulica br. 3", 35}
};
```

Pokazivač na strukturu

• Pokazivač na strukturu se deklariše kao i svi drugi pokazivači:

```
struct osoba *pok;
```

• Pristup elementima strukutre na koju ukazuje pokazivač:

```
pok->starost
```

Niz pokazivača na strukturu

Slično nizu običnih pokazivača, opšta sintaksa je:
struct str *ime[dužina];
• Primer:
struct osoba *pokosobe[3];

• Pristup elementu niza:
pokosobe[i]->ime

Hijerarhijske strukture
• Unutar jedne strukture može da postoji druga.
• Primer:

```
struct datum
{
        int dan;
        int mesec;
        int godina;
};
struct osoba {
        char ime[50];
        char adresa[150];
        int starost;
        struct datum datum_rodjenja;
};
...
struct osoba osoba1 = { "Petar Petrovic", "Petrova ulica 3", 15,
{10, 1, 1990}};
```

Struktura može da ima element **tipa iste strukture:**

```
struct cvor
{
        int informacija;
        struct cvor *sledeci;
};
```

Operator typedef
- Za **definiciju imena novog tipa podatka**, na osnovu postojećeg ili izvedenog tipa podatka.
- **Opšti oblik: typedef** postojeci_tip novo_ime;
• Primer:

```
typedef float real;
...
real a, b;
ili
typedef char *string;
...
string a;
ili
typedef char string [50];
...
string a;
```

• Operator typedef i struct:

```
typedef struct
{
  string ime;
  string adresa;
  int starost;
} osoba;
...
osoba a, b, *c;
```

3. Unije i polja bitova

- Unije liče na strukture,

– svi elementi zauzimaju isti memorisjki prostor,

– u jednom trenutku može da se koristi samo jedan element.

• Primer:

union promenljivo
{
int i;
float f;
};
...
union promenljivo u;
u.i = 3;
ili
u.f = 3.14;

Češća primena unija je u kombinaciji sa strukturama:

• Još preciznije:

```
struct krug
{
  float r;
};
struct pravougaonik
{
  float a, b;
};
struct trougao
{
  float a, b, c;
};
```

```
struct figura
{
  int tip;
  float obim;
  union
  {
    struct krug k;
    struct pravougaonik p;
    struct trougao t;
  } podaci;
};
```

UnijaStruktura2.c

• Upotreba:

```
fig2.tip = PRAVOUGAONIK;
fig2.podaci.p.a   = 10;
fig2.podaci.p.b   = 20;
fig2.obim = 2 * fig2.podaci.p.a + 2 * fig2.podaci.p.b;
```

Polja bitova

- Omogućuje definisanje struktura koje imaju zadati broj bitova.

Opšti oblik:

struct oznaka
{
[unsigned] int ime1 : br_bitova;
[unsigned] int ime2 : br_bitova;
...
};

- Redosled polja bitova u memoriji zavisi od implementacije.

```
struct char_attribute {
  unsigned int boja      : 3;
  unsigned int intensity : 1;
  unsigned int pozadina  : 3;
  unsigned int flash     : 1;
};
```

4.Datoteke

Osnovni način rada sa datotekama je preko **FILE strukture:**
– **datoteka se otvara,** dobija se **pokazivač na FILE strukturu**
– sve **funkcije za rad sa datotekama** primaju kao **parametar i pokazivač na FILE strukturu**
– **datoteka se zatvara**, sa jedinim parametrom – pokazivačem na FILE strukturu.
• Sve **funkcije** se nalaze u **stdio.h biblioteci.**

Otvaranje datoteke
Funkcija za otvaranje datoteke je:
FILE *fopen(char *ime_datoteke, char *mod)

Drugi parametar označava mod otvaranja datoteke:
– **“r” – read,** odn. samo čitanje
– **“w” – write,** odn. samo pisanje
– **“a” – append**, odn. dodavanje na kraj
– na nekim sistemima se dodaje i “b” na kraj moda da se naglasi binarno otvaranje datotek

Funkcija **fopen vraća NULL** ako ne može da otvori zadatu datoteku
• Primer:
FILE *f;
f = fopen(“pera.dat”, “r”);

Zatvaranje datoteke
• Datoteka se zatvara funkcijom
fclose(FILE *f)

Rad sa datotekom
Funkcije za rad sa datotekom su:
1. **getc(FILE *f)** i **putc(char c, FILE *f) –** učitavanje/pisanje jednog karaktera u datoteku
2. **fprintf(FILE *f, ...)** i **fscanf(FILE *f, ...)** – isto kao i printf/scanf, samo sa datotekom
3. **fgets(char *s, int maxl, FILE *f)** i **fputs(char *, FILE *f) -** učitavanje/snimanje jednog stringa iz datoteke
4. **feof(FILE *f)** – vraća 1 ako smo stigli do kraja datoteke
u toku čitanja
5. **ferror(FILE *f)** – vraća 1 ako je bilo greške tokom rada sa datotekom

---
getc i putc
• getc(FILE *f) i putc(char c, FILE *f) – učitavanje/pisanje jednog karaktera u datoteku.
• Ako se kao rezultat učitavanja getc funkcijom **dobije konstanta EOF**, onda smo došli do **kraja datoteke.**

---
fprintf i fscanf

fprintf(FILE *f, ...) i fscanf(FILE *f, ...) – isto kao i printf/scanf, samo sa datotekom.
• Prvi parametar je pokazivač na FILE strukturu, a ostali su kao kod standardnih printf i scanf funkcija.
---

fgets i fputs
• fgets(char *s, int maxl, FILE *f) i fputs(char*, FILE *f) – učitavanje/snimanje stringa iz datoteke.
• **Funkcija fgets:**
– čita ili do zadate dužine ili do kraja reda
– vraća NULL ako dođe do kraja datoteke
– učitani string sadrži i znak za novi red!
• **Funkcija fputs** snima string**, bez znaka za novi red**
---
feof i ferror
• feof(FILE *f) – vraća 1 ako smo stigli do kraja datoteke u toku čitanja.
• **Funkcija feof** će vratiti 1 na kraju datoteke tek nakon što kraj datoteke bude detektovan pri pokušaju čitanja iz datoteke nekom standardnom funkcijom!
• **ferror(FILE *f)** – vraća 1 ako je bilo greške tokom rada sa datotekom.
• Funkcija **clearerr**(FILE *f) briše status greške.

Sistemski pozivi
• Rad sa datotekama je moguć i upotrebom **OS-specifičnih funkcija**
– logična posledica ovakvog pristupa: gubitak portabilnosti.

Rad sa blokovima memorije
• Funkcije fread i fwrite čitaju/pišu blokove memorije.
• **Funkcija fread:**
fread(void *ptr, size_t size, size_t n, FILE *stream);
• **Funkcija fwrite:**
fwrite(const void *ptr, size_t size, size_t n, FILE*stream)

NEVEZANO ZA PITANJA
Dinamičke strukture podataka
• Dinamičke strukture podataka menjaju svoju veličinu tokom vremena.
• Koriste memoriju koja se dinamički alocira i oslobađa
– za to se koriste malloc i free funkcije
• Jednostruko povezane liste – uređene jednostruko povezane liste
• Binarna stabla
• Stek
Alokacija memorije

5. Liste

Jednostruko povezane liste (jednostruko spregnute liste)

• **Skup čvorova povezanih u jednom smeru**

• Svaki čvor se sastoji iz dva elementa:

1. **informacija koju čvor nosi** (informacioni deo) i
2. **pokazivač na sledeći element**.

• Primer:

```
typedef char TIP;
typedef struct cvor_st
{
        TIP inf;
        struct cvor_st *sledeći;
} LCVOR;
...
LCVOR *pocetak_liste;
```

Stanja jednostruko povezane liste

• **Prazna lista**

– pocetak_liste == NULL

• **Lista ima jedan čvor (element)**

– pocetak_liste → prvi (i jedini) element → NULL

• **Lista ima više čvorova** (elemenata)

– pocetak_liste → prvi el → ... → poslednji → NULL

podsetnik

→ pokazivač na strukturu - struktura cvor_st *sledeci

Operacije sa listama

1. Dodavanje na početak

• Kreira se novi čvor

– **novi = (LCVOR *)malloc(sizeof(LCVOR));**

• novi->sledeci = pocetak_liste;

• pocetak_liste = novi;

---

2. Dodavanje na kraj liste

• Kreira se novi čvor

– **novi = (LCVOR *)malloc(sizeof(LCVOR));**

• krene se od početka liste i sve dok je

```
while(tekuci != NULL){
```

– prethodni = tekuci;

– tekuci = tekuci → sledeci;

```
}
```

• kada je tekuci == NULL, tada prethodni ukazuje na poslednji cvor

```
novi->sledeci = NULL;
prethodni->sledeci = novi;
```

---

3. Pronalaženje čvora
• krene se od početka liste i sve dok je **tekuci != NULL**
– ako je **tekuci->inf == trazena_inf,** onda **vrati tekuci,** inače
– **tekuci = tekuci ->sledeci;**
• ako se do kraja ne nađe traženi čvor, tekuci će biti jednak NULL, pa funkcija to vraća
---
4. Brisanje čvora - brisi.c
• Ako se briše **prvi čvor**, onda se **pocetak_liste preveže na drugog**
• Inače se traži čvor, ali se pamti i pokazivač na prethodnog
– preveže se prethodni na sledećeg od nađenog
---
5. Brisanje liste
• Krene se od početka liste
– tekuci = pocetak_liste;
• dokle god je **tekuci razlicit od NULL,**
– pamti se tekuci
• tmp = tekuci
– pređe se na sledeći
• tekuci = tekuci->sledeci;
– obriše se tmp čvor
• free (tmp);

Bisanje liste

```
tekuci = pocetak_liste;
while (tekuci != NULL)
{
        tmp = tekuci;
        tekuci = tekuci->sledeci;
        printf("brisem: %c\n", tmp->inf);
        free(tmp);
}
pocetak_liste = NULL
```

Uređena lista
• Ideja – dodavati nove čvorove tako da lista ostane uređena (sortirana)
– to se postiže tako što se sadržaj novog čvora poredi sa tekućim, i ako je manji ili jednak, dodaje se ispred tekućeg
– inače se pomerimo na sledeći čvor i ponovimo poređenje

6. Stabla

Svaki čvor sadrži **informaciju i dva pokazivača:**
– **levi** ukazuje na **podstablo** koje se sastoji iz **čvorova manjih od tekućeg**
- desni ukazuje na **podstablo** koje se sastoji iz čvorova **većih od tekućeg**
• Postoji i osnovni čvor – **koren**
• Čvor koji **nema ni jedan podčvor se zove list**
• **Dobre osobine**
– binarno stablo očuvava elemente sortirane
– brza pretraga

```
typedef int TIP;
typedef struct cvor_st
{
        TIP inf;
        struct cvor_st *desni;
        struct cvor_st *levi;
} BCVOR;
...
BCVOR *koren;
```

Operacije sa stablima

1. Ubacivanje u stablo - ubacist.c

• Kreira se novi čvor

– novi = (BCVOR *)malloc(sizeof(BCVOR));

• prođe se kroz stablo i kada se stigne do kraja stabla (naiđe na list), umetne se ispod njega (levo ili desno zavisi od sadržaja)

---

2. Pronalaženje čvora - pronst.c

• Krene se od korena i gleda da li je sadržaj traženog čvora manji ili veći od tekućeg
• Ako je manji, ide se u levo podstablo
• Ako je veći, ide se u desno podstablo
• Ako je jednak, vrati se pokazivač na tekući

---

3. Brisanje čvora - brisist.c

• Krene se od korena i gleda se da li je sadržaj traženog čvora manji ili veći od tekućeg
• Ako je manji, ide se u levo podstablo
• Ako je veći, ide se u desno podstablo
• **Ako je jednak,** obriše se čvor i obavi se **prevezivanje**

a) ako je čvor koji se briše list, samo se obriše, a onaj čvor koji je ukazivao na njega se ažurira (NULL)

b) ako čvor koji se briše ima samo jedan podčvor, samo se obriše, a onaj koji je ukazivao na obrisanog se preveže na podčvor

brisist2.c
brisist3.c

c) ako čvor koji se briše ima oba podčvora, on se obriše, a na njegovo mesto se uveže najmanji element u njegovom desnom podstablu

a) najmanji element koji je uvezan umesto izbrisanog se uklanja sa originalne lokacije i radi se isto prevezivanje.

---

4. Brisanje stabla

• Krene se od korena i briše se levo i desno podstablo, pa onda koren