VII – Tipovi podataka u C jeziku

- U C-u se mora voditi više računa o tipovima podataka jer je to opširna i kompleksna stvar kojoj C programeri posvećuju ogromnu važnost
- Posebna oblast koja se bavi samo tipovima podataka - **Tipologija**
- Razlikujemo sledeće tipove podataka u programskom jeziku:

1. **Osnovni tipovi**
    a. **Standardni i prošireni celobrojni tipovi** (***char, int, short, long, long long***)
    b. **Realni i kompleksni tipovi sa pokretnim zarezom** (***float, double, long double, float _Complex, double _Complex, long double _Complex*****)**
2. **Nabrojivi tipovi -** celobrojni tipovi koje programer definiše sam Nabrajanje počinje definisanjem ključne reči ***enum*** iza koga sledi identifikator tipa i lista mogućih vrednosti.
3. **Tip void -** specifikator tipa ***void*** označava da nema dostupnih vrednosti tog tipa. Ne možemo deklarisati konstantu ili promenjivu tipa ***void***. Međutim, tip ***void*** se koristi kad funkcija ne vraća nikakvu vrednost ili nema parametre.

VII – Izvedeni tipovi podataka

- Pored osnovnih tipova podataka u C-u postoje i izvedeni tipovi podataka koji se izvode od osnovnih tipova podataka.

4. **Izvedeni tipovi** :
   1. **Pokazivači** – u promenljivoj ovog tipa pamti se adresa nekog podatka u memoriji, gde se pokazivač izvodi iz osnovnog tipa podatka na koji on pokazuje
   2. **Nizovi** – homogena stuktura kod koje su svi podaci istog tipa
      Stringovi – posebna klasa nizova kod kojih su svi elementi karakter podaci i koji se u C-u završavaju specijalnim znakom **`\0`**
   3. **Strukture** – spadaju u korisničke izvedene tipove podataka a koriste se za grupisanje podataka koji su u nekoj semantičkoj vezi
   4. **Unije** – predstavljaju specijalan tip struktura kod kojih se umesto odvajanja mem.prostora za svaki elemenat strukture kreira samo jedna univerzalna mem.lokacija
   5. **Funkcije** - delovi programa koji se definišu samo jednom dok ih možemo izvršavati više puta u programu.

VII – Pokazivači (*Pointers*)

- Pokazivač je promenljiva koja sadrži adresu nekog objekta ili funkcije
- Pokazivači se intenzivno koriste u programskom jeziku C i to zbog:
  1. nekad je to jedini put ka rešavanju nekog problema,
  2. vode kompaktnijem i efikasnijem kodu od bilo kog drugog načina.
- Pokazivači imaju **veoma značajnu ulogu** u C jeziku jer teško je napisati bilo koji kompleksniji program bez njihove upotrebe.
- Programer, u radu sa pokazivačima, ima veliku slobodu i veoma širok spektar različitih mogućnosti.
- To otvara i veliki prostor za efikasno programiranje ali i za greške.
- Naročito to dolazi do izražaja ako se program piše neoprezno, pa je sasvim verovatna bila pojava pokazivača koji su se pogrešno koristili.
- Međutim, uz samo malo pažnje, pokazivači se mogu iskoristiti da bi se postigla **jasna čitljivost** i **jednostavnost** programskog koda.
- Njihova primena omogućuje napredne programske tehnike: dinamičko alociranje memorije, apstraktni tip podataka i polimorfizam funkcija.

VII – Pokazivači i adrese

- Memorija se sastoji se od niza uzastopnih bitova koji su organizovani u vidu adresibilnih lokacija kojima se pristupa pojedinačno ili grupno
- Na 16-bitnim sistemima adrese zauzimaju dva bajta, na 32-bitnim sistemima četiri bajta, na 64-bitnim osam bajta, i slično.
- Bilo koji oktet bitova(*byte*)može biti **char**, par okteta može se smatrati kao **short**, a četiri susedna okteta čine jedan **long** celobrojni tip.
- Pokazivači predstavljaju tip podataka čije su vrednosti memor. adrese.
- Iako su pokazivačke vrednosti (adrese) celi brojevi, pokazivački tipovi se razlikuju od celobrojnih i ne mogu se mešati sa njima.
- Jezik C razlikuje više pokazivačkih tipova i za svaki tip podataka (i osnovni i korisnički) postoji odgovarajući pokazivački tip.
- Pokazivači implicitno čuvaju informaciju o tipu onoga na šta ukazuju.
- Informacija o tipu pokazivača ne postoji tokom izvršavanja programa
- Tu informaciju koristi kompajler tokom kompilacije, da u mašinskom kodu generiše instrukcije koje odgovaraju konkretnim tipovima.
- U toku izvršavanja pokazivačka promenljiva zauzima samo onoliko bajtova koliko zauzima podatak o adresi na koju ukazuje.

VII – Pokazivači

- Rad sa pokazivačkim promenljivama teško je razumeti bez poznavanja koncepta **indirekcije**, odnosno koncepta posrednog pristupa
- Bilo koji objekat ili promenljiva se pamti na tačno određenoj lokaciji.
- Pokazivači omogućuju indirektni pristup vrednostima tih programskih promenljivih korišćenjem njihovih memorijskih adresa.
- Pokazivači se često koriste u radu sa nizovima kako bi se lakše dolazilo do određenog člana niza.
- U neposrednoj vezi sa pokazivačima su **dva specijalna operatora**:

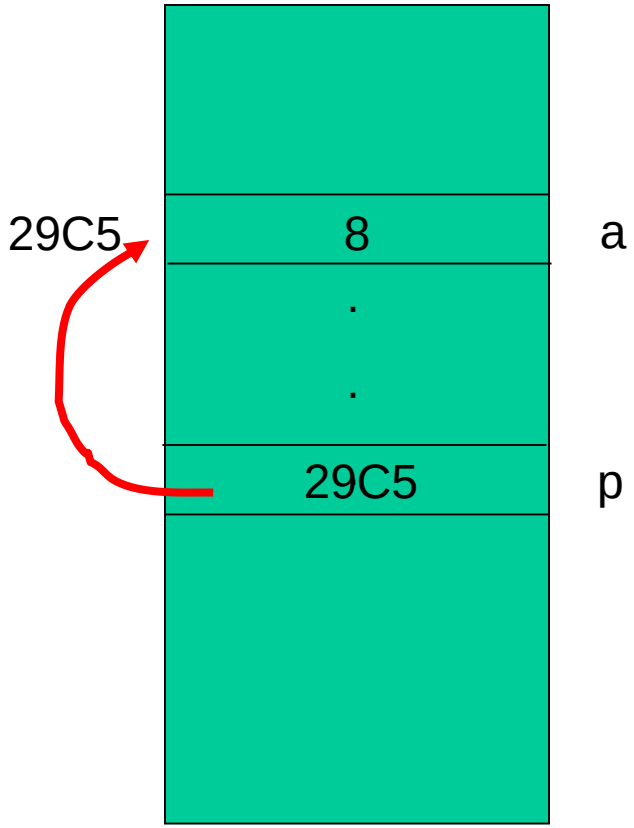
1. **Adresni operator referenciranja** **&**(*ampersand*)ili operator ***adresa-od*** predstavlja unarni operator vrlo visokog prioriteta koji daje mem. adresu objekta na koji je primenjen. Predstavlja način da se sazna mem.lokacija gde je taj objekat smešten u memoriji računara.
2. **Indirektni operator** ili operator ***dereferenciranja*** * omogućava deklarisanje pokazivačke promenljive i indirektni pristup do vrednosti za koje su korišćene te pokazivačke promenljive. To znači da kada se on primeni na neku adresu u memoriji on vraća ono što je zapisano na toj adresi.

VII – Pokazivači i adrese

Primer:

```
void main()
{
   int a;
   int *p;
   p = &a;
   a = 8;
   printf("\n%d %x",*p,p);
}
```

8 29C5



VII – Pokazivači

- Unarni operator **&** daje adresu objekta, tako naredba **p=&a;** određuje adresu **a** prema varijabli **p**, a za **p** možemo reći da "pokazuje na" **a**.
- Operator **&** odnosi se samo na objekte u memoriji, bile to promenljive ili elementi nekog polja i ne može se primeniti na izraze i konstante.
- Adresni operator **&** se može primenjivati samo na jednu vrednost:

Primer: **&(suma+200)** → **Nije dozvoljeno**

- Unarni operator * je indirektni ili dereferentni operator.
- Kad se primeni na pokazivač, on pristupa objektu na koji on pokazuje.

Primer: Neka su **x** i **y** celi brojevi, a neka je **ip** pokazivač na **int**.

```
int x = 1; y = 2, z[10];          /* inicijalizacija */
int *ip;          /* deklaracija pointera */
ip = &x;          /* ip pokazuje na x */
y = *ip;          /* y dobija vrednost promenljive na koju pokazuje ip */
x = 2;            /* neposredna dodela vrednosti na koju pokazuje ip */
*ip = 0;          /* promenljiva na koju pokazuje ip */
                  /* dobija vrednost 0 – posredna dodela vredn.*/
ip = &z[0];       /* ip sada pokazuje na z[0]      ⟶   x=0, y=1
```

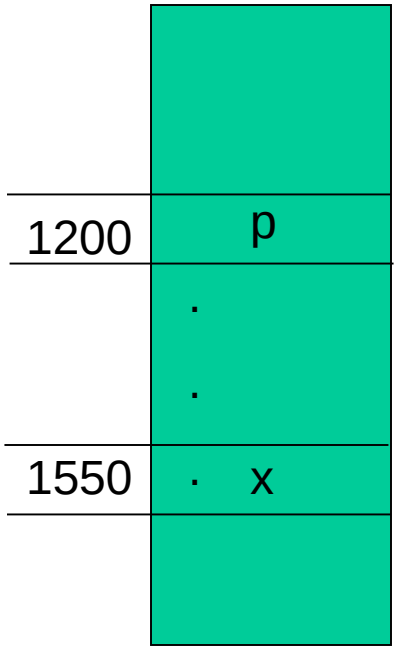
VII – Pokazivači i adrese

➢ **&p, p i *p imaju tri različita značenja**:

1. **&p** - označava adresu promenljive (tipa pokazivača)
2. **p** - označava sadržaj promenljive (tipa pokazivača)
3. ***p** - označava vrednost memorijske lokacije na koju ukazuje promenljiva tipa pokazivač

Primer:

Operativna memorija



Vrednost parametara

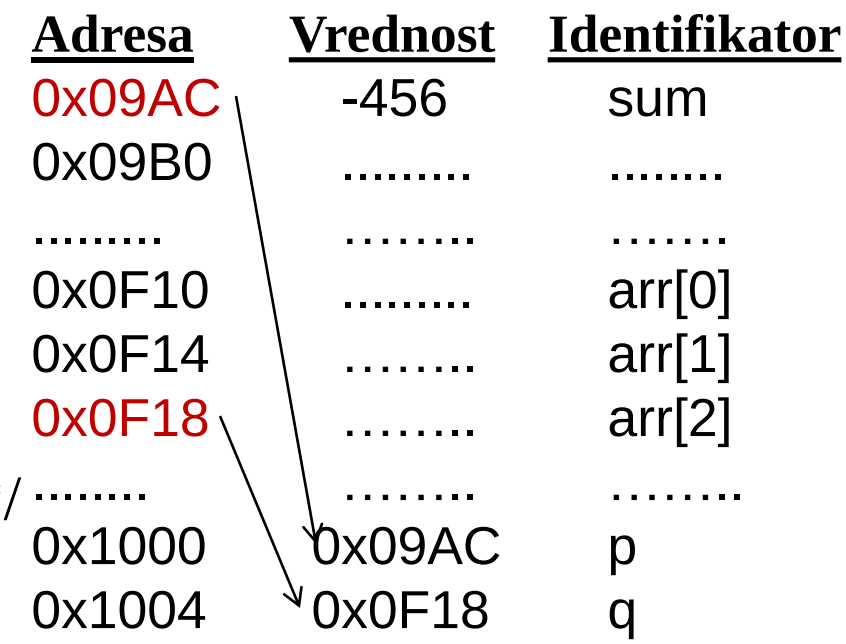
int x; int *p;		x=15;		p = &x;		*p = 13;	
&p	1200	&p	1200	&p	1200	&p	1200
p	?	p	?	p	1550	p	1550
*p	nema	*p	nema	*p	15	*p	13
&x	1550	&x	1550	&x	1550	&x	1550
x	?	x	15	x	15	x	13

VII – Tipovi pokazivača

- Pokazivačkim promenljivama se deklaracijom dodeljuje tip.
- Deklaracija pokazivačke promenljive vrši se tako što se u deklaraciji ispred imena pokazivačke promenljive upisuje zvezdica *.

Primer: **int *p,*q**; /* p i q su pokazivači na int */

- Označava da su deklarisane pokazivačke promenljive **p** i **q**, kojima je namena da sadrže adresu objekata tipa int. Kaže se da su **p** i **q** "**pokazivač na int**".
- Pokazivači, pre upotrebe, moraju biti inicijalizirani na neku realnu memorijsku adresu
- To se ostvaruje tzv. adresnim operatorom **&**:
- **p = ∑** /* p iniciran na adresu varijable sum */
- **q = &arr[2];** /* q iniciran na adresu trećeg elementa niza arr*/



VII – Deklaracija pokazivača

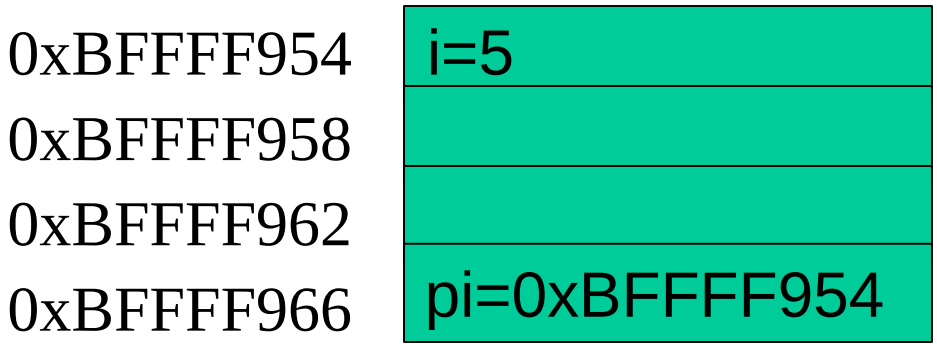
- Da bi se dobila adresa neke promenljive koristi se adresni operator **&**.
- Ako je **v** promenljiva datog tipa, a **pv** pokazivač na taj tip, onda je naredbom **pv=&v;** pokazivaču **pv** pridružena adresa promenljive **v**.
- Pored adresnog operatora koristimo i operator dereferenciranja * koji vraća vrednost koja se nalazi na adresi na koju pokazivač pokazuje
- Tako, ako je **pv=&v**, onda je ***pv** isto što i **v**.
- Pokazivač na neki tip deklariše se na sledeći način: **tip *ime;** gde je **ime** ime pokazivača, a **tip** je tip podatka na koji pokazuje.
- Zvezdica označava da se radi o pokazivaču, a ne o promenljivi tipa **tip**.

Primer: promenljiva **pi** deklarisana naredbom **int *pi;** je pokazivač na **int**

- Prilikom definicije promenljive ona može biti direktno inicijalizirana ali promenljiva čija se adresa uzima mora biti definisana pre nje

Primer:

```
int i=5;
int *pi=&i;
```



VII – Deklaracija pokazivača

- Adresni operator **&** može se primeniti samo na operande kojima je pridružena jedinstvena adresa.
- Zato ga ne možemo primeniti na npr. aritmetičke izraze i slično.
- Operator dereferenciranja ***** deluje samo na pokazivačke promenljive.
- Adresni operator i operator dereferenciranja su unarni operatori i imaju isti prioritet kao ostali unarni operatori.
- Njihov prioritet je veći od prioriteta aritmetičkih operatora tako da u aritmetičkim izrazima *pi nije potrebno stavljati u zagradu.

Primer: Uz deklaraciju (**int i=5; int *pi=&i;**) izraz **i=2*(*pi+6);** daje **i=22** jer se ***pi** izračunava i daje 5 pre aritmetičkih operacija.

- Operator dereferenciranja može se pojaviti na levoj strani jednakosti tj. možemo imati ***pi=6**; što je ekvivalentno sa **pi=6**, ali se zato adresni operator ne može pojaviti na levoj strani jednakosti.
- Deklaracija **int *pi** indicira da je ***pi** objekt tipa **int**.
- Sintaksa deklaracije promenljive prihvata sintaksu izraza u kome se ona pojavljuje a to isto se odnosi i na deklaraciju funkcija.

VII – Deklaracija pokazivača

Primeri:

- U deklaraciji pokazivača

  int *f(char *);

  f je funkcija koja uzima pokazivač na **char** i vraća pokazivač na **int**.
- Deklaracija sugeriše da ***f(s)** mora biti tipa **int** (**s** je pokazivač na **char** ili jednostavno niz znakova).
- Pokazivač se u printf naredbi ispisuje sa kontrolnim znakom **%p**:

```
#include <stdio.h>
int main(void) {
int i=5;
int *pi=&i;
printf("i= %d, adresa od i= %p\n",i,pi);
return 0;
}
```

VII – Operacije sa pokazivačima

1. ***Deklarisanje pokazivača*** - postupak kojim se deklariše identifikator pokazivača, tako što se između oznake tipa na koji pokazivač pokazuje i identifikatora pokazivača upisuje operator indirekcije *.

   Primer: **int x, *p**; /* deklaracija varijable x i pokazivača p */

2. ***Inicijalizacija pokazivača*** - operacija kojom se pokazivaču dodeljuje vrednost koja je jednaka adresi objekta na koji on pokazuje. Za dobijanje adrese objekta koristi se unarni adresni operator '**&**' .

   Primer: **p = &x;** /*p sadrži adresu od x */

3. ***Dereferenciranje pokazivača*** -operacija kojom se pomoću pokazivača pristupa mem.objektu na koji on pokazuje, odnosno, ako se u izrazima ispred identifikatora pokazivača zapiše operator indirekcije *, dobija se dereferencirani pokazivač (*p). On se može koristiti kao promenljiva, tj. referenca memorijskog objekta na koji on pokazuje.

   Primer: **y = *p;** /* **y** dobija vrednost promenljive na koju **p** */
   /* pokazuje, isti učinak kao **y = x** */
   ***p = y;** /* **y** se dodeljuje promenljivoj na koju **p** */
   /* pokazuje, isti učinak kao **x = y** */

VII – Operacije sa pokazivačima

- Delovanje adresnog operatora je komplementarno delovanju operatora indirekcije, i važi da naredba **y = *(&x);** ima isti učinak kao i **y = x**;.
- Unarni operatori ***** i **&** su iznad većine operatora u izrazima.

Primer: y = *p + 1; ⇔ y = (*p) + 1;

- Jedino postfiksni unarni operatori (**--**, **++**, **[]**, **()**) imaju veći prioritet od * i & prefiks operatora:

Primer: y = *p++; ⇔ y = *(p++);

- Pokazivač kome je vrednost nula (NULL) naziva se nul pokazivač.

Primer: p = NULL; /* p pokazuje na ništa */

- Sa dereferenciranim pokazivačem (*p) se može manipulisati kao i sa promenljivom pripadnog tipa kao što je prikazano u primerima:

```
int x, y, *px, *py;
px = &x;          /* px sadrži adresu od x – ne utiče na x */
*px = 0;          /* vrednost x postaje 0 - ne utiče na px */
py = px;          /* py takođe pokazuje na x - ne utiče na px ili x */
*py += 1;         /* uvećava y za 1 - ne utiče se na px ili py */
y = (*px)++;      /* y = 1, a x =2 - ne utiče na px ili py */
```

VII – Operacije sa pokazivačima

- Nad pokazivačima moguće je izvoditi sledeće računske operacije:

1. Dodela vrednosti jednog pokazivača drugom
2. Dodavanje celobrojnog podatka na vrednost pokazivača
3. Oduzimanje celobrojnog podatka od vrednosti pokazivača
4. Upoređivanje vrednosti dva pokazivača
5. Upoređivanje vrednosti pokazivača sa nulom

VII – Operacije sa pokazivačima

- Aritmetičke operacije dozvoljene nad pokazivačima konzistentne su sa svrhom pokazivača da pokazuju na promenljivu određenog tipa.
- Ako je **pi** pokazivač tipa **int**, onda će **pi+1** biti pokazivač na sedeću promenljivu tipa **int** u memoriji.
- To znači da dodavanje **1** pokazivaču se ne povećava adresa koju on sadrži za jedan, već za onoliko koliko je potrebno da nova vrednost pokazuje na sledeću promenljivu istog tipa u memoriji.

Primer:

```
#include <stdio.h>
int main(void)
{
float x[]={1.0,2.0,3.0},*px;
px=&x[0];
printf("Vrednosti: x[0]=%g, x[1]=%g, x[2]=%g\n“, x[0],x[1],x[2]);
printf("Adrese : x[0]=%x, x[1]=%x, x[2]=%x\n“,px,px+1,px+2);
return 0;
}
```

VII – Operacije sa pokazivačima

- U ovom primeru vidimo da će pokazivač biti inkrementiran dovoljno da pokaže na sedeću **float** vrednost.
- Uočimo da smo pokazivače ispisali u formatu **%x** kao heksadecimalni ceo broj (uporedite s ispisom u formatu **%p**).
- Svakom pokazivaču moguće je dodati i oduzeti ceo broj.
- Zato ako je **px** pokazivač i **n** promenljiva tipa **int**, onda su dozvoljene operacije nad pokazivačima: **++px --px px+n px-n**
- Pokazivač px+n ukazuje na *n-ti* objekt nakon onog na koji pokazuje px
- Unarni operatori **&** i ***** imaju viši prioritet od aritmetičkih operatora i operatora pridruživanja.
- Zato u izrazu ***px += 1**; dolazi do povećanja za jedan vrednosti na koju px pokazuje, a ne samog pokazivača.
- Isti izraz bismo mogli napisati kao **++*px;** zato što važi pravilo da je asocijativnost unarnih operatora zdesna na levo, pa se prvo primenjuje dereferenciranje, a zatim inkrementiranje.
- Izraz ***px++** inkrementirao bi pokazivač nakon što bi vratio vrednost na koju **px** pokazuje pa zato treba koristiti zagrade : **(*px)++;**.

VII – Operacije sa pokazivačima

Primer:

```
ip = &x;
*ip = *ip + 10; /* uvećava *ip za 10 */
y = *ip + 1;    /* vrednost promenljive na koju pokazuje ip */
                /* se uvećava za 1 i dodeljuje y */
*ip += 1;       /* povećava promenljivu na koju pokazuje ip za 1 */
++*ip;          /* isto */
(*ip)++;        /* isto */
int *iq;        /* deklaracija pokazivača iq */
iq = ip;        /* iq sada pokazuje na istu promenljivu kao i ip */
int **ir;       /* deklaracija pokazivača na pokazivač na int */
ir = &iq;       /* ir sada pokazuje na pokazivač iq */
**ir += 3;      /* x += 3 */
```

x = 16 i y = 11

VII – Operacije sa pokazivačima

Upoređivanje pokazivača

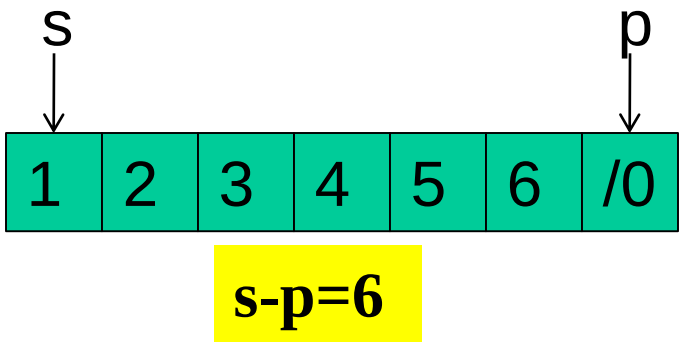
- Pokazivači istog tipa mogu se uporediti pomoću relacijskih operatora
- Takva operacija ima smisla ako pokazivači pokazuju na isto polje.
- Ako su **px** i **py** dva pokazivača istog tipa, onda je moguće koristiti izraze **px < py** **px > py** **px == py** **px != py**
- Rezultat tih operacija je 1 ili 0, zavisno da li je relacija tačna ili ne

Oduzimanje pokazivača

- Pokazivač se može oduzeti od drugoga ako oni pokazuju na isto polje
- Ako su **px** i **py** dva pokazivača na isto polje te ako je **py > px**, tada je **py-px+1** broj elemenata izmedu **px** i **py**, uključujući krajeve.
- Uočimo da je **py-px** vrednost celobrojnog tipa (a ne pokazivačkog).

Primer: Funkcija koja daje broj znakova u stringu:

```
int strlen(char *s)
{
char *p=s;
while(*p != '\0') p++;
return p-s;
}
```



VII – Operacije sa pokazivačima

Pokazivači i celi brojevi

- Pokazivaču nije moguće pridružiti vrednost celobrojnog tipa.
- Izuzetak jedino predstavlja nula jer C garantuje da nula nije legalna adresa i omogućava da se nula pridruži bilo kojoj pokazivackoj promenljivoj sa ciljem da se signalizira kako promenljiva ne sadrži legalnu adresu.
- Ispravno je da se piše **double *p=0;**
- To je naročito korisno kod automatskih promenljivih koje pri pozivu funkcije imaju nepoznatu vrednost.
- Često se u ovu svrhu koristi konstanta NULL (nalazi se u <stdio.h>).

```
#define NULL 0
.....
double *p=NULL;
```

- Pokazivače je osim sa drugim istovrsnim pokazivačem moguće upoređivati i sa nulom, tako da je ispravno da se piše **if(px != 0) ...**
- Upoređivanje s drugim celim brojevima nije dozvoljeno:

```
if(px == 0xBFFFF986) ... //
```

POGREŠNO

VII – Operacije sa pokazivačima

Najčešće greške:

- ✓ Nemoguće je definisati pointer na konstantu ili izraz.
- ✓ Nije moguće promeniti adresu promenljive (jer to ne određuje programer već operativni sistem).
- ✓ Zbog toga su najčešće sledeće greške:

Primer:

```
i = &3;
j = &(k+5);
k = &(a--b);
&a = &b;
&a = 150;
```

VII – Operacije sa pokazivačima

1. Pokazivaču može biti pridružena adresa (npr. **px=&x**);
2. Pokazivaču može biti pridružen pokazivač istog tipa (npr. **px=py**);
3. Pokazivaču može biti pridružena nula (npr. **px=0** ili **px=NULL**);
4. Pokazivaču može biti dodata ili oduzeta celobrojna promenljiva (npr. **px+3**, **++px**, **--px** itd.);
5. Dva pokazivača mogu se sabirati ili oduzimati ukoliko pokazuju na isto polje;
6. Dva pokazivača mogu biti povezana relacijskim operatorom ako pokazuju na isto polje.

VII – Pokazivači kao argumenti f-je

- Kako su funkcije memorijski objekti onda takođe možemo deklarisati i inicijalizirati pokazivače na funkcije.
- Važi da se za f-ju imena **F**, koja je deklarisana/definisana u obliku:

  oznaka_tipa F (list_ parametara);

  pokazivač na tu funkciju imena **pF**, deklariše se u obliku:

  oznaka_tipa (*pF) (list_ parametara);
- Ime funkcije, napisano bez zagrada predstavlja adresu funkcije, pa se dodelom vrednosti: **pF = F;** inicira pokazivač **pF** na adresu funkcije **F**.
- Kada je pokazivač iniciran, indirekcijom pokazivača može se izvršiti poziv funkcije u obliku: **(*pF)(lista_argumenata);** ili još prostije, sa **pF(lista_argumenata);**male zagrade predstavljaju operator poziva f-je kompajler sam vrši indirekciju pokazivača, ako se napišu male zagrade
- Videli smo da C predaje argumente funkcijama pomoću vrednosti.
- To znači da nema načina da se direktno promene promenljive u funkciji iz koje je druga funkcija pozvana.
- Pogledajmo sada potprogram **izmena** koji treba da izvrši zamenu vrednosti kod dve promenljive a i b, tj. da vrednosti zamene mesta.

VII - Pokazivači kao argumenti f-je

Primer: Realizovati funkciju za zamenu vrednosti dve promenljive

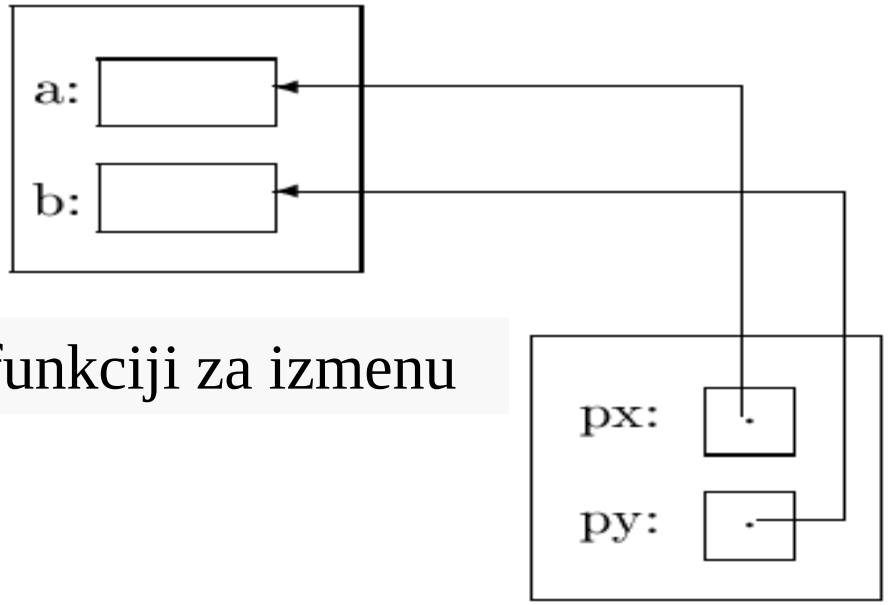
```
izmena(x,y)
int x, y;
{
        int pomocna;
        pomocna=x;
        x=y;
        y=pomocna;
}
main ()
{
   int a=8,b=3;
   izmena(a, b);
   printf("Brojevi posle poziva funkcije %d, %d", a,b);
}
```

Brojevi posle poziva funkcije 8, 3 ???

VII - Pokazivači kao argumenti f-je

- Funkcija nije uradila ništa, jer je ona razmenjivala samo sopstvene kopije promenljivih **a** i **b** zbog prenosa parametara po vrednosti.
- Zbog toga se funkcija mora realizovati korišćenjem pokazivača.
- Naime, kao argumenti funkcije se prenose pokazivači na promenljive **a** i **b**, a ne njihove vrednosti.

U pozivnoj funkciji:



u funkciji za izmenu

VII - Pokazivači kao argumenti f-je

Ispravno rešenje:

```
izmena(int *px,int *py)
{
        int pomocna;
        pomocna=*px;
        *px=*py;
        *py=pomocna;
}
main ()
{
          int a=8,b=3;
          izmena(&a,&b);
          printf("Brojevi posle poziva funkcije %d, %d", a,b);
}
```

Brojevi posle poziva funkcije 3, 8

VII – Pokazivači kao argumenti f-je

- Pokazivači se često koriste kao argumenti funkcije, jer se na taj način mogu prenositi promenljive u funkciju.
- To se postiže tako što se kao parametar f-je deklariše pokazivač na neki objekat: **void Increment(int *pVar);**
- Prema pravilu C jezika, pri pozivu f-je prenosi se vrednost stvarnog argumenta, a u funkciji se parametar funkcije tretira kao lokalna promenljiva koja ima vrednost stvarnog argumenta.
- U slučaju kada je parametar funkcije pokazivač, stvarni argument f-je je adresa objekta koji ima isti tip kao pokazivač.
- Korišćenjem indirekcije pokazivača može se pristupiti tom objektu i menjati njegov sadržaj(objekat se može tretirati kao promenljiva u f-ji)
- Na sledećem slajdu prikazan je program gde se pomoću funkcije

void Increment(int *pVar){(*pVAr)++;}

može inkrementirati vrednost bilo koje celobrojne promenljive imena **var**, pozivom funkcije **Increment(&var)**

VII - Pokazivači kao argumenti f-je

Primer:

```
#include <stdio.h>
void Increment(int *pVar)
{ /* Funkcija inkrementira vrednost promenljive, čija se adresa
   * prenosi u funkciju kao vrednost pokazivača pVar
   * Promenljivoj se pristupa pomoću indirekcije pokazivača pVar */
(*pVar)++;
}
int main()
{
int var = 7;
Increment(&var);          /* argument je adresa varijable */
printf ("var = %d\n", var);
return 0;
}
```

Program ispisuje:

var = 8

- Uočite da se u definiciji f-je koristi pokazivački parametar,a pri pozivu f-je se kao argument koristi adresa promenljive na koju ova f-ja deluje.

VII – Pokazivači kao argumenti f-je

- Kod funkcija imamo ograničenje u tome što f-je ne mogu da vrate više od jedne vrednosti a i ne mogu da menjaju vrednosti promenljivih koje su deklarisane unutar drugih funkcija jer ih „ne vide“
- Ovo se može promeniti korišćenjem pokazivača za parametre funkcija.

Primer:

```
int test( int *a, int *b )
{
*a = 789;
*b = 456;
}
void main(void)
{
int e = 23,r = 10;
test( &e, &r);
//prikazi vrednost
printf(“E je %d, R je %d \n“, e,r);
}
```

Program će ispisati:

„E je 789, R je 456 “

VII – Generički pokazivači

Parametri funkcije tipa void pokazivača

- Ako se neki pokazivač deklariše pomoću reči void, **void *p;**, tada nije određeno na koji tip podataka on pokazuje.
- Pokazivači koji nemaju određen tip podataka na koji ukazuju nazivaju se **generičkim pokazivačima**
- Takvim pokazivačima može se dodeliti adresa bilo kog memorijskog objekta, ali se ne može vršiti pristup memorijskim objektima pomoću operatora indirekcije, jer on pokazuja na ,,**ništa**,,.
- Većina današnjih kompajlera ne dozvoljava aritmetičke operacije sa **void** pokazivačima.
- Očito je da nema smisla koristiti **void** pokazivače kao regularne promenljive ali oni ipak mogu biti jako korisni kao parametri funkcija.
- Kada se **void** pokazivač koristi kao parametar funkcije tada se pri pozivu funkcije tom pokazivaču može dodeliti adresa bilo kog memorijskog objekta, a unutar same funkcije se može sa prefiksom (**tip ***) vršiti forsirano pretvaranje pokazivačkog tipa.

VII - Pokazivači kao argumenti f-je

Primer: korišćenje void pokazivača kao parametra funkcije

```
#include <stdio.h>
#define CHAR 0
#define INT 1
#define FLOAT 2
#define DOUBLE 3
void UnesiVrednost(void *p, int tip)
{
switch (tip) {
   case CHAR:
   printf( "Unesite jedan znak: \n");
   scanf("%c", (char *) p);
   break;
case INT:
   printf( "Unesite celi broj:\n");
   scanf("%d", (int *) p);
   break;
case FLOAT:
   printf( "Unesite realni broj:\n");
   scanf("%g", (float *) p);
   break;
case DOUBLE:
   printf( "Unesite realni broj:\n");
   scanf("%lg", (double *) p);
   break;
}
fflush(stdin);        /* odstrani višak znakova s ulaza*/
}

int main()
{
double dval;
int ival;
UnesiVrednost(&ival, INT);
printf("Vrednost je %d\n" , ival);
UnesiVrednost(&dval, DOUBLE);
printf("Vrednost je %lg\n" , dval);
return 0;
}
```

- Uočite kako je korišćena funkcija **scanf()**.
- Ispred imena argumenta nije korišćen adresni operator jer je vrednost pokazivača **p** adresa već je ispred argumenta eksplicitno označen tip.
- Ovakvo korišćenje **void** pokazivača je opasno, jer ako se pri pozivu funkcije ne pozovu kompatibilni argumenti, može doći do blokade

VII – Primer korišćenja pokazivača

Primer: *Analizirati efekat sledećeg programa i objasniti programske naredbe u kojima se pojavljuju pokazivači.*

```
main()
{
int broj_a, *point_br;
char slovo1, slovo2, *point_sl;
broj_a=4;
slovo_1='c';
point_br=&broj_a;
broj_a+=*point_br+1;
printf (" %d n", broj_a);
point_sl=&slovo_1;
slovo_2=*point_sl;
printf ("%c n", slovo_2);
}
```

Program će ispisati:

9

C

VII – Primer korišćenja pokazivača

- Da bismo shvatili program, potrebno je objasniti četiri programske naredbe, u kojima se pojavljuju pokazivači.

1. Posle izvršenja naredbe: **point_br=&broj_a;** pokazivač **point_br** pokazuje na adresu, gde je smeštena promenljiva broj_a.
2. Naredbom: **broj_a+=dS*point_br+1;** promenljiva **broj_a** dobija vrednost 4+4+1=9. Naime, vrenosti promenljive **broj_a** (a to je 4) se dodaje jedinica i promenljiva tipa **int** na koju pokazuje pokazivač **point_br**. Kako **point_br** pokazuje na istu promenljivu **broj_a**, rezultat naredbe će biti 9.
3. Posle izvršenja naredbe: **point_sl=&slovo_1;** pokazivač **point_sl** pokazuje na adresu, gde je smeštena promenljiva **slovo_1**, koja je tipa **char**.
4. Naredbom: **slovo_2=*point_sl;** promenljivoj **slovo_2** se dodeljuje vrednost promenljive tipa **char**, na koju pokazuje pokazivač **point_sl**, a to je vrednost **slovo_1** (a to je slovo **C**).

Hvala na pažnji !!!

Pitanja

? ? ?