

Uvod u programiranje

- predavanja -

prosinac 2019.

8. Pokazivači

Tip podatka *pokazivač*

Uvod

Radna memorija računala

- Radna memorija računala može se promatrati kao kontinuirani niz bajtova, od kojih svaki ima svoj "redni broj", odnosno *adresu*
 - slikom je ilustrirana memorija veličine 4GB

0	00110001
1	11010010
...	...
82560	11000001
82561	00001101
82562	11000001
82563	11101000
...	...
4294967294	00110001
4294967295	00000111

Objekti i vrijednosti u programskom jeziku C

- Objekt (*object*) je područje u memoriji čiji sadržaj reprezentira vrijednost
- Vrijednost (*value*) je interpretacija sadržaja objekta koja se temelji na *tipu* i *sadržaju* objekta

```
...  
char c = 'B';  
...  
int m = 7;  
...  
float x = -0.75f;  
...
```

...	...	
82560	01000010	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Adresa objekta

- Za objekt kažemo da se nalazi na adresi A (ili adresa objekta je A) ako je prvi bajt sadržaja objekta pohranjen na adresi A
 - Npr. varijabla m nalazi se na adresi 82642, odnosno adresa varijable m je 82642

- Radi ilustracije pretpostavljeno je da se varijable nalaze na prikazanim adresama. U stvarnosti, nemoguće je znati o kojim se točno adresama radi prije nego se program pokrene (a nije niti važno znati ih unaprijed).

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Kako se u programu dolazi do vrijednosti objekta

- Pristupanje objektu pomoću *identifikatora*
 - ime varijable (za skalarne tipove), ime varijable i indeks (za polja), ime varijable i ime člana (za strukture), ...
 - navođenjem identifikatora objekta dobiva se *lvalue* koji se može koristiti za čitanje ili postavljanje vrijednosti objekta
 - tip podatka poznat je iz definicije varijable
 - tip je važan: npr. ako tip podatka ne bi bio poznat, ne bi bilo moguće ispravno obavljati operacije

```
double y;  
y = m + x;
```

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Može li se objektu pristupiti pomoću adrese?

- Može li se do vrijednosti doći pomoću (samo) adrese objekta?
 - npr. ako je poznato da se neki objekt nalazi na adresi 82642?
 - ne, samo adresa nije dovoljna**
- Za ispravno pristupanje objektu potrebna je *i adresa i tip objekta* koji se nalazi na toj adresi
 - adresa i tip objekta predstavljaju jedan oblik *reference* na taj objekt
 - tip objekta kojem se pristupa pomoću *reference* naziva se **referencirani tip** (*referenced type*)

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Tip podatka *pokazivač* (*pointer type*)

- Tip podatka koji omogućuje pristupanje objektu pomoću *reference*
 - Ako je referencirani objekt tipa T , tada se za pristupanju objektu koristi tip podatka *pokazivač na T* . Npr. podatak tipa *pokazivač na int* omogućuje pristup objektu tipa int
 - Za tip podatka *pokazivač* ne postoje zasebne ključne riječi (kao za tipove podataka int , $float$, itd.). Tip podatka *pokazivač* opisuje se pomoću naziva referenciranog tipa i znaka $*$

```
int *p1, *p2;  
float *p3;
```

- Varijable $p1$ i $p2$ su tipa pokazivač na int
- Varijabla $p3$ je tipa pokazivač na $float$

Variable tipa pokazivač

- Za varijablu *tipa pokazivač* vrijedi sve što je do sada navedeno o varijablama ostalih skalarnih tipova, osim:
 - definira se na malo drugačiji način: navođenjem imena *referenciranog tipa* i znaka *** ispred imena varijable
 - pohranjuje podatke tipa pokazivač na referencirani tip

```
int m;
```

```
int *p1;
```



referencirani
tip

varijabla p1 nije tipa int, nego
tipa *pokazivač na int*

- dopušteno je u istoj naredbi definirati varijable referenciranog tipa i varijable tipa pokazivača na referencirani tip

```
int m, *p1, *p2, k;
```

Koju vrijednost upisati u varijablu tipa pokazivač

```
int m = 7, *p1;  
p1 = ?
```

- Općenito, ako je *x* varijabla (ili član polja, ili struktura ili član strukture, ...), tada je **&x** pokazivač na *x*
 - &** je tzv. *adresni operator*. Rezultat izraza **&m** je *pokazivač na int* jer je *m* objekt tipa *int*
 - adresa odgovara adresi varijable *m* (82642)
 - rezultat je tipa pokazivač na *int*
 - budući da je rezultat izraza **&m** *pokazivač na int*, smije se pridružiti varijabli *p1* (koja je tipa *pokazivač na int*)

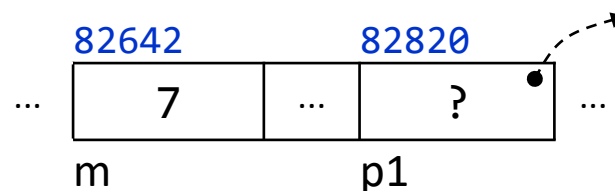
```
p1 = &m;
```

...	...	}	m
82642	00000000		
82643	00000000		
82644	00000000		
82645	00000111	}	p1
...	...		
82820	00000000		
82821	00000001		
82822	01000010		
82823	11010010		
...	...		

Primjer

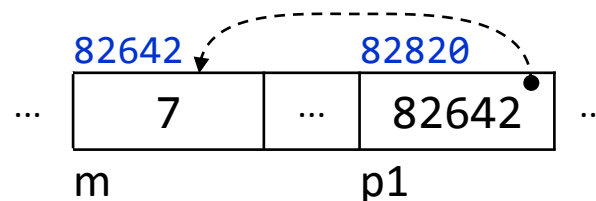
- U nastavku ćemo sadržaj memorije prikazivati na prikladniji način

```
int m = 7, *p1;
```



- varijabla `p1` još uvijek nije inicijalizirana: pokazivač pohranjen u varijabli `p1` "pokazuje u nepoznato"

```
p1 = &m;
```



- u varijablu `p1` sada je upisan podatak tipa *pokazivač na int* kojim se može pristupiti objektu tipa `int` na adresi 82642
 - radi pojednostavljenja, koristit će se kolokvijalni izrazi:
 - naredbom `int *p1;` definiran je pokazivač `p1`
 - naredbom `p1 = &m;` u `p1` je upisana adresa varijable `m`
 - `p1` pokazuje na objekt na adresi 82642
 - `p1` pokazuje na varijablu `m`, `p1` pokazuje na objekt `m`


Inicijalizacija varijable tipa pokazivača uz definiciju

- Jednako kao i varijable drugih tipova, varijable tipa pokazivač mogu se inicijalizirati u trenutku definicije

```
int m, *p1 = &m, *p2 = p1;  
float x, *p3 = &x, y, *p4 = &y;
```

- voditi računa o redoslijedu definicije i inicijalizacije. Objekt čija se adresa izračunava adresnim operatorom mora biti definiran


```
int *p1 = &m, m;
```



Neispravno, može se popraviti premještanjem

- voditi računa o tome da i varijabla tipa pokazivača može sadržavati "smeće" (*garbage value*)

```
int m, *p1;  
int *p2 = p1;  
p1 = &m;
```



U ovom trenutku p1 još uvijek sadrži "smeće"
Može se popraviti premještanjem naredbe

Paziti na razlike u tipovima pokazivača

- Tipovi pokazivača su međusobno različiti ako se razlikuju njihovi referencirani tipovi
 - u varijable jednog tipa pokazivača nije dopušteno upisivati pokazivače drugog tipa

```
int m;  
int *pInt;  
float x;  
float *pFloat;
```

```
pInt = &m;  
pFloat = &x;
```

```
pFloat = pInt;
```

```
pInt = &x;
```

```
pFloat = &m;
```

Neispravno

Neispravno

Neispravno

Adresa nije cijeli broj

- Iako *izgleda* kao cijeli broj, adresa u općem slučaju nije `int` (niti `short`, niti `long`, ...). Stoga nema smisla:
 - pokazivač pohranjivati u varijablu tipa `int`
 - cijeli broj pohranjivati u varijablu tipa pokazivača

```
int *p1;
```

```
int m;
```

```
m = 5;
```

```
p1 = &m;
```

```
p1 = m;
```

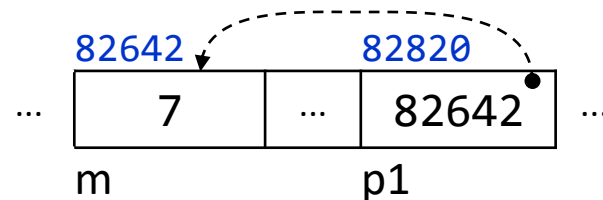
```
m = p1;
```

Neispravno

Neispravno

Pristupanje objektu pomoću pokazivača

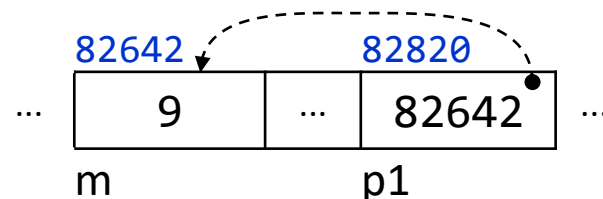
```
int m = 7, *p1 = &m;
```



- objektu (7, tip `int`) na adresi 82642 može se pristupiti:

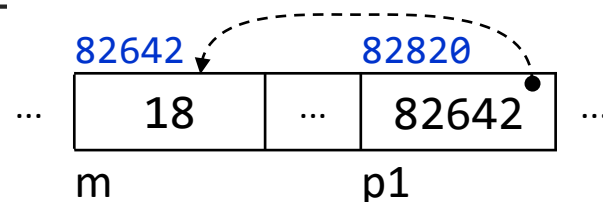
- (naravno) pomoću imena varijable `m`

```
m = m + 2;
```



- ali također i primjenom *operatora indirekcije* (unarni operator `*`) nad pokazivačem pohranjenim u varijabli `p1`

```
*p1 = 2 * *p1;
```



```
printf("%d %d", m, *p1);
```

```
18 18
```

pročitaj cijeli broj s mjesta na kojeg pokazuje `p1`, dobiveni rezultat (tipa `int`) pomnoži s 2 i rezultat upiši na mjesto kamo pokazuje `p1`

Operator indirekcije *

- Operator omogućuje da se objektu pristupi *indirektno* pomoću pokazivača (umjesto *direktno* preko imena varijable)
 - operator je također poznat pod imenom *operator dereferenciranja* jer operator "*dereferencira*" pokazivač (*referencu* na objekt) i tako dolazi do objekta
- Općenito, ako p pokazuje na objekt x, tada je rezultat operacije **p lvalue* koja predstavlja objekt na kojeg pokazuje p
 - to znači: ako je p varijabla koja sadrži pokazivač koji pokazuje na objekt u memoriji koji predstavlja varijablu m, tada se izraz *p može koristiti na svakom mjestu u programu gdje se može koristiti ime varijable m
 - za čitanje vrijednosti (npr. u nekom izrazu)
 - za postavljanje vrijednosti (kao lijeva strana izraza pridruživanja), uz uvjet da je sadržaj objekta izmjenljiv

Neke oznake su pomalo zbunjujuće?

```
int m = 7;  
int *p1 = &m;  
...  
p1 = &m;           Ispravno  
*p1 = &m;          Neispravno
```

- Kako to da je u naredbi za definiciju varijable `p1` ispravno napisati `*p1 = &m`, a naredba `*p1 = &m` je neispravna?

- u programskom jeziku C isti simboli u različitom kontekstu mogu imati različito značenje

```
int *p1 = &m;
```

`p1` definiraj kao varijablu
tipa pokazivač na `int`

varijablu koju si upravo definirao,
`p1`, inicijaliziraj na vrijednost `&m`

ovdje simbol `*` ne predstavlja operator indirekcije, nego označava da varijabla `p1` nije tipa `int`, nego tipa *pokazivač na int*

```
*p1 = &m;
```

neispravno jer je rezultat izraza `*p1` objekt tipa `int`, što znači da se u objekt tipa `int` pokušava upisati vrijednost tipa pokazivač na `int`

Neke oznake su pomalo zbunjujuće?

```
int m = 7;  
int *p1 = &m, *p2 = p1;  
...  
p2 = p1;           Ispravno  
*p2 = p1;          Neispravno
```

```
int *p1 = &m, *p2 = p1;
```

```
p2 = *p1;
```

- Kako to da je u naredbi za definiciju varijable p2 ispravno napisati `*p2 = p1`, a naredba `*p2 = p1;` je neispravna?

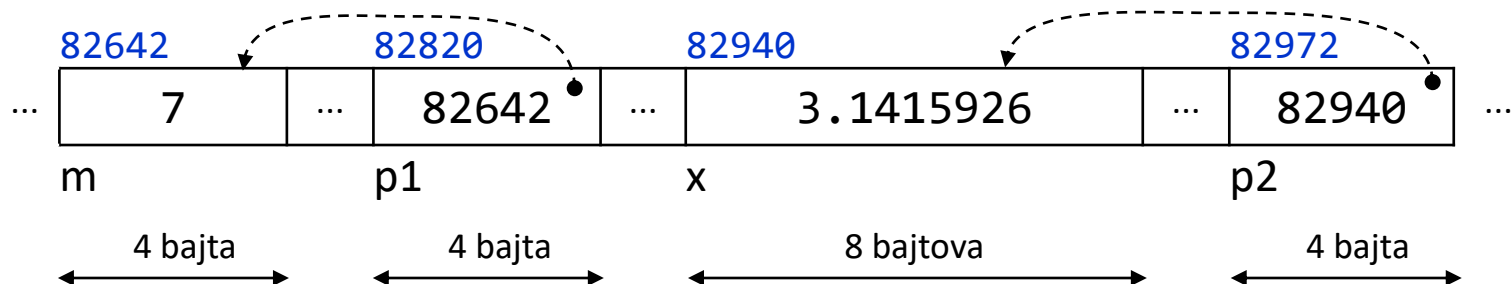
definirana je varijabla p1, inicijalizirana je na &m, zatim je definirana varijabla p2 koja se inicijalizira na vrijednost koja se nalazi u p1.

neispravno jer je rezultat izraza `*p1` objekt tipa `int`, što znači da se vrijednost tipa `int` pokušava upisati u varijablu tipa pokazivač na `int`

Koliko prostora zauzima pokazivač

- Adresa objekta je adresa na kojoj je pohranjen prvi bajt objekta
 - to znači da veličina referenciranog tipa ne bi trebala utjecati na veličinu prostora koju zauzima pokazivač na taj tip

```
int m = 7, *p1 = &m;  
double x = 3.1415926, *p2 = &x;
```



- jednaki prostor (4 bajta) zauzimaju pokazivač `p1` na objekt tipa `int` (koji je veličine 4 bajta) i pokazivač `p2` na objekt tipa `double` (koji je veličine 8 bajtova)

Koliko prostora zauzima pokazivač

- Pokazivači na jednoj platformi (isti operacijski sustav, arhitektura i prevodilac) u principu* zauzimaju jednaku količinu memorije bez obzira na koji tip podatka pokazuju

```
int m = 7, *p1 = &m;  
double x = 3.1415926, *p2 = &x;  
printf("%u %u %u\n", sizeof(p1), sizeof(m), sizeof(*p1));  
printf("%u %u %u", sizeof(p2), sizeof(x), sizeof(*p2));
```

x86_64, Windows, gcc

```
4 4 4  
4 8 8
```

x86_64, Linux, gcc

```
8 4 4  
8 8 8
```

* U praksi je to uglavnom tako, ali s obzirom da C standard takvo pravilo izrijeком ne propisuje, ne smije se u potpunosti isključiti mogućnost da će se na nekoj platformi veličine pokazivača međusobno razlikovati s obzirom na tip podatka na koji pokazuju.

Generički pokazivač (*pointer to void*)

- Referencirani tip pokazivača mora biti poznat kako bi se na temelju adrese (gdje je objekt) i tipa (kojeg tipa je objekt na toj adresi) sadržaj objekta mogao ispravno interpretirati
- Međutim, postoji specijalni tip pokazivača za kojeg to ne vrijedi
 - *generički pokazivač* (u literaturi također: *pokazivač na void*, *pointer to void*) je pokazivač koji može pokazivati na objekt bilo kojeg tipa
 - budući da referencirani tip generičkog pokazivača nije poznat, neće se moći koristiti za pristup objektu (kažemo: generički pokazivač se ne može *dereferencirati*)
 - ali zato je moguće napraviti **eksplicitnu konverziju (*cast*)** generičkog pokazivača na tip pokazivača za kojeg će referencirani tip biti T
 - rezultat sljedeće operacije nad generičkim pokazivačem je pokazivač na tip podatka T

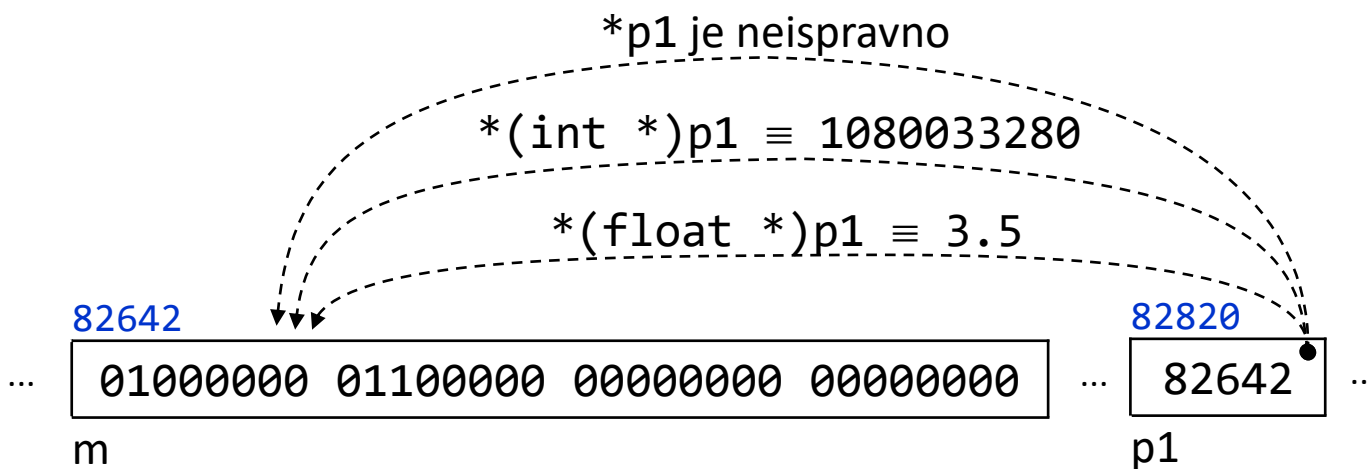
(T *) genericki_pokazivac

Primjer

```
int m = 1080033280;
void *p1;
p1 = &m;
// printf("%d", *p1);
printf("%d\n", *(int *)p1);
printf("%f\n", *(float *)p1);
```

Neispravno jer p1 nije moguće dereferencirati

1080033280
3.500000



Konverzijske specifikacije za printf i scanf

- konverzijska specifikacija %p koristi se za ispis i čitanje podatka tipa pokazivač
 - točan oblik ispisa nije propisan standardom (vrijednost će se ispisati kao broj u dekadskom ili heksadekadskom brojevnom sustavu ili u nekom drugom obliku)
 - argument (pokazivač) koji se ispisuje dobro je eksplicitno konvertirati u generički pokazivač, ali u većini slučajeva može se ispustiti

```
int m = 7, *p1 = &m;  
printf("m je na adresi %p", (void *)p1);  
// printf("m je na adresi %p", p1);
```

Može i bez (void *)

x86_64, Windows, gcc

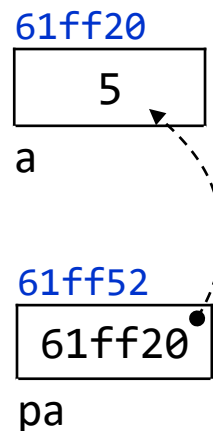
m je na adresi 0061ff28

x86_64, Linux, gcc

m je na adresi 0x7fffd6e8d324

Primjer

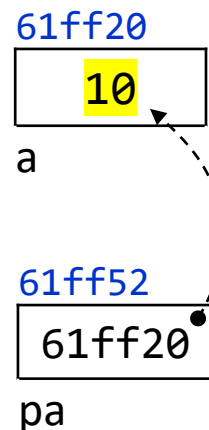
```
int a = 5, b = 10;  
int *pa, *pb;  
pa = &a;    // pretpostavka pa = 61ff20  
pb = &b;    // pretpostavka pb = 61ff24
```



- što će se ispisati sljedećim odsječkom?

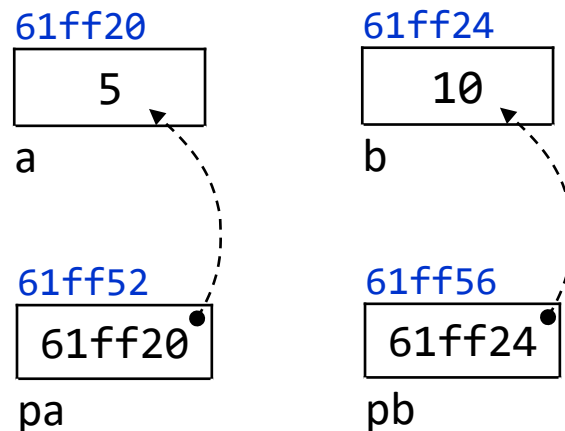
```
*pa = *pb;  
printf("%d %d\n", a, b);  
printf("%p %p\n", (void *)pa, (void *)pb);  
printf("%d %d\n", *pa, *pb);
```

```
10 10  
61ff20 61ff24  
10 10
```



Primjer

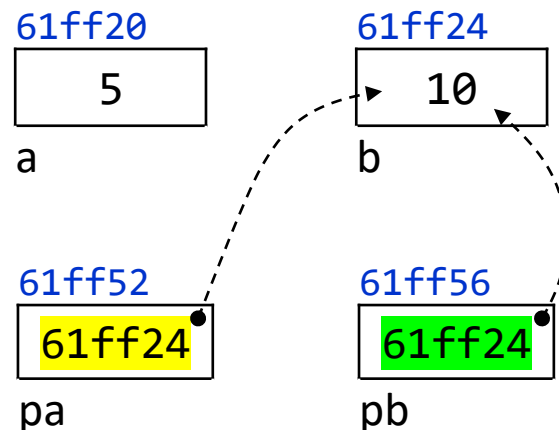
```
int a = 5, b = 10;  
int *pa, *pb;  
pa = &a;    // pretpostavka pa = 61ff20  
pb = &b;    // pretpostavka pb = 61ff24
```



- što će se ispisati sljedećim odsječkom?

```
pa = pb;  
printf("%d  %d\n", a, b);  
printf("%p  %p\n", (void *)pa, (void *)pb);  
printf("%d  %d\n", *pa, *pb);
```

```
5 10  
61ff24 61ff24  
10 10
```



Primjer

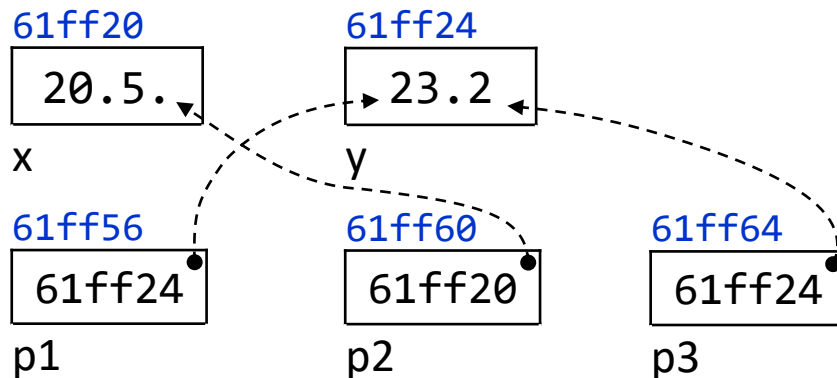
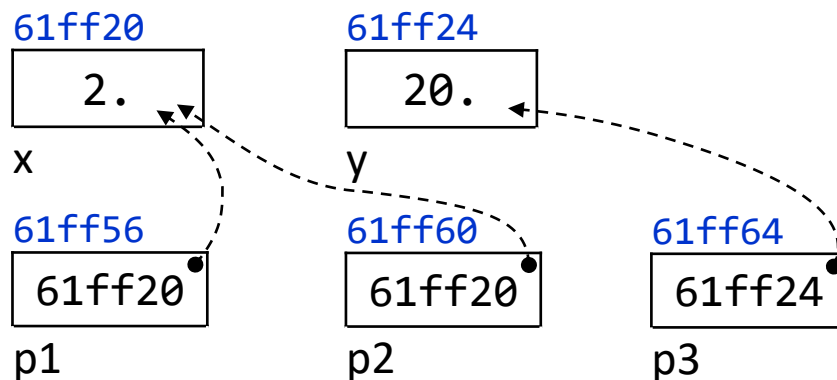
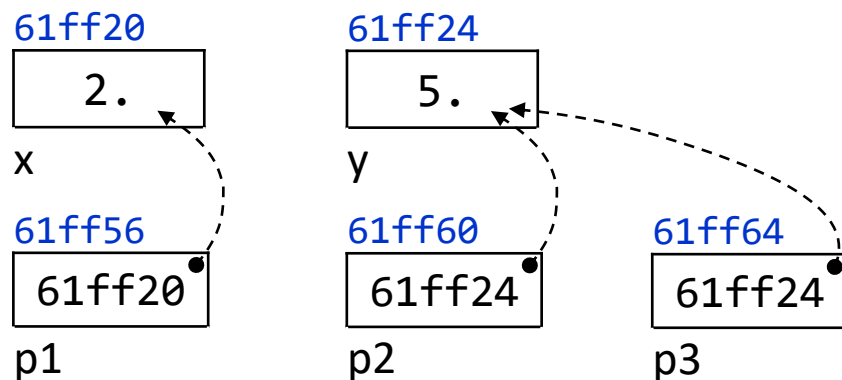
```
float x = 2.f, y = 5.f;  
float *p1, *p2, *p3;  
p1 = &x;  
p2 = p3 = &y;
```

- nacrtati sliku nakon

```
*p3 = *p2 + 3.f * *p2;  
p2 = p1;
```

- i nakon

```
*p2 = *p3 + 0.5f;  
p1 = p3;  
*p1 += 3.2f;
```



Tip podatka *pokazivač*

Korištenje pokazivača pri pozivu funkcije

Podsjetnik

- Funkcija (u programskom jeziku C) **ne može** izmjenom vrijednosti parametara promijeniti vrijednosti argumenata jer
 - parametar sadrži *kopiju* vrijednosti argumenta. Zato kažemo da se u funkciju argumenti *prenose po vrijednosti* (*pass by value*)

```
#include <stdio.h>
void pokusajPromijenitiArgument(int n) {
    n = 10;
    printf("Funkcija je parametar promijenila u n = %d\n", n);
    return;
}
int main(void) {
    int n = 5;
    printf("Funkcija je pozvana s argumentom n = %d\n", n);
    pokusajPromijenitiArgument(n);
    printf("Ali argument je ostao n = %d", n);
    return 0;
}
```

Funkcija je pozvana s argumentom n = 5
Funkcija je parametar promijenila u n = 10
Ali argument je ostao n = 5

Prijenos po vrijednosti (*pass by value* ili *call by value*)

- Tako niti sljedeći program neće raditi ono što bismo htjeli:

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

```
void zamijeni(int x, int y) {
```

```
    int pom;
```

```
    pom = x;
```

```
    x = y;
```

```
    y = pom;
```

```
}
```

```
int main(void) {
```

```
    int a = 5, b = 10;
```

```
    zamijeni(a, b);
```

```
    printf("a = %d, b = %d", a, b); a = 5, b = 10 NEISPRAVAN REZULTAT!
```

```
    return 0;
```

```
}
```

- Na početku izvršavanja funkcije parametri x i y sadrže *kopije* vrijednosti argumenata
- operacije nad parametrima x i y ne utječu na vrijednosti argumenata s kojima je procedura pozvana - u ovom slučaju na vrijednosti varijabli a i b

- U C-u se funkcija *zamijeni* može ispravno implementirati samo uz pomoć *pokazivača*

Prijenos po vrijednosti - objašnjenje primjera

- Vrijednosti varijabli i parametara u svakom koraku izvršavanja:

```
void zamijeni(int x, int y) { (2)
    int pom; (3)
    pom = x; (4)
    x = y; (5)
    y = pom; (6)
    return; (7)
}

int main(void) {
    int a = 5, b = 10; (1)
    zamijeni(a, b);
}
```

Nakon	a	b	x	y	pom
(1)	5	10	-	-	-
(2)	5	10	5	10	-
(3)	5	10	5	10	?
(4)	5	10	5	10	5
(5)	5	10	10	10	5
(6)	5	10	10	5	5
(7)	5	10	-	-	-

- x, y ne postoje prije obavljanja koraka (2)
- pom ne postoji prije obavljanja koraka (3)
- pom sadrži *garbage value* nakon obavljanja koraka (3)
- nakon završetka funkcije, x, y i pom više ne postoje

Prijenos po vrijednosti

- Vrijednosti parametara smiju se mijenjati, bez posljedica na vrijednosti argumenata, čak i onda kada *naziv* parametra odgovara *nazivu* varijable koja se koristi kao argument

```
double eksp(float x, int n) {  
    int i;  
    double rez = 1.;  
    for (i = 0; i < n; ++i)  
        rez *= x;  
    return rez;  
}
```

```
double eksp(float x, int n) {  
    double rez = 1.;  
    for (; n > 0; --n)  
        rez *= x;  
    return rez;  
}
```

[...] parameters can be treated as conveniently initialized local variables in the called routine.

B. W. Kernighan, D. M. Ritchie (1988.), The C Programming Language, 2nd Edition, Englewood Cliffs, NJ: Prentice Hall

Prijenos po referenci (*pass by reference* ili *call by reference*)

- U nekim jezicima (npr. Pascal, Fortran) u parametre je moguće prenijeti *reference* na argumente:

```
program testVar;  
  var  
    a, b: integer;
```

```
  procedure zamijeni(var x, y: integer);  
    var  
      pom: integer;  
  begin  
    pom := x;  
    x := y;  
    y := pom;  
  end;
```

```
begin  početak "glavnog programa"  
  a := 5; b := 10;  
  zamijeni(a, b);  
  writeln('a = ', a, ', b = ', b);  
end.
```

- procedura i funkcija u Pascalu - slični su funkciji u C-u

- riječ 'var' navedena ispred definicije parametara znači da su parametri x i y reference na argumente s kojima je procedura pozvana
- svaka operacija u proceduri nad parametrima x i y zapravo je operacija nad argumentima - u ovom primjeru nad varijablama a i b

a = 10, b = 5 ISPRAVAN REZULTAT!

U jeziku C ne postoji prijenos po referenci

- Ali zato postoji zamjena za mehanizam prijenosa po referenci
 - u parametar se prenese kopija argumenta koji je pokazivač na objekt na pozivajućoj razini
 - u funkciji se parametar (sadrži pokazivač) može koristiti da bi se pristupilo vrijednosti objekta na pozivajućoj razini
 - funkcija sada, slično kao kod korištenja prijenosa po referencama (Pascal), može promijeniti vrijednost nekog objekta definiranog na pozivajućoj razini
 - razlika je samo u tome što će se parametar (koji je pokazivač), morati *dereferencirati* da bi se pristupilo tom objektu
 - i dalje se radi o *prijenosu po vrijednosti*, ali zato što su prenesene vrijednosti *pokazivači*, kolokvijalno kažemo da se radi o *prijenosu po pokazivaču* (*pass by pointer*)

Primjer

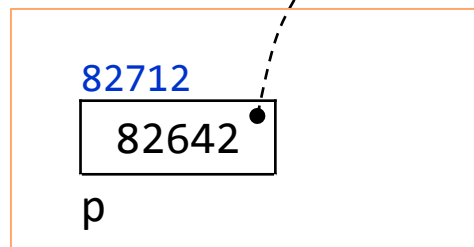
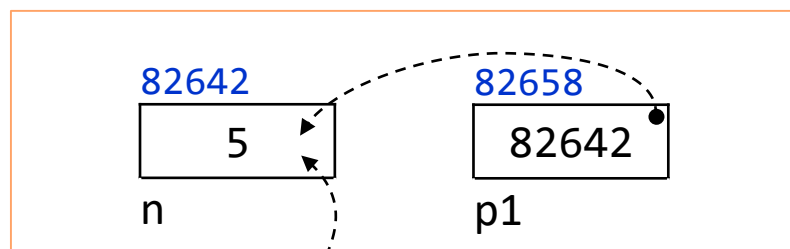
- U ovom primjeru se objektu koji je definiran u funkciji main, može pristupiti pomoću parametra koji sadrži kopiju argumenta, koji je pokazivač na objekt definiran u funkciji main

```
void promijeni(int *p) {  
    *p = 10;  
    return;  
}
```

```
int main(void) {  
    int n = 5, *p1 = &n;  
    promijeni(p1);  
alternativno:  
    int n = 5;  
    promijeni(&n);  
}
```

Slika prikazuje sadržaj memorije neposredno prije izvršavanja naredbe `*p = 10;`

varijable/argumenti u funkciji main



obavljanjem naredbe `*p = 10;` promijenit će se sadržaj varijable `n`

varijable/parametri u funkciji `promijeni`

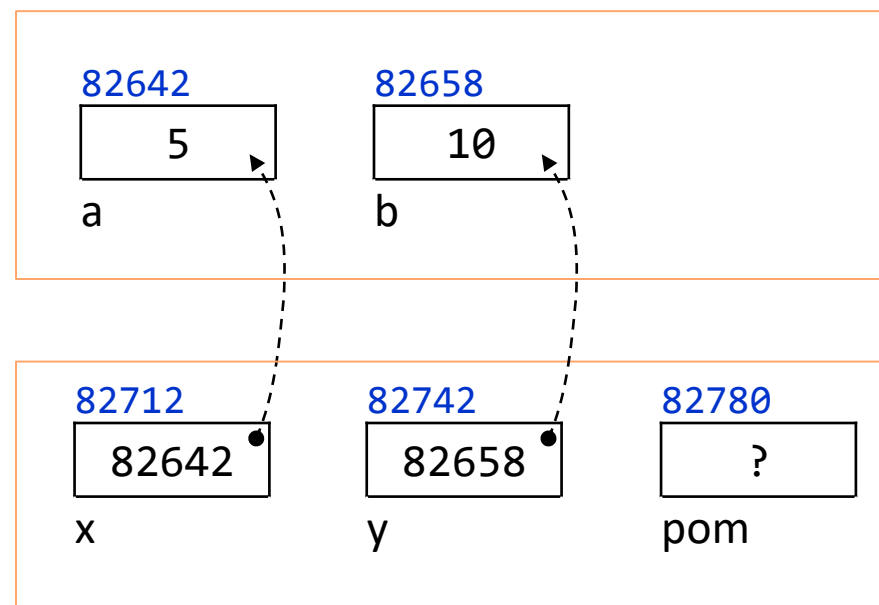
Primjer

```
void zamijeni(int *x, int *y) {  
    int pom;  
    pom = *x;  
    *x = *y;  
    *y = pom;  
    return;  
}
```

```
int main(void) {  
    int a = 5, b = 10;  
    zamijeni(&a, &b);  
    ...  
}
```

Slika prikazuje sadržaj memorije neposredno prije izvršavanja naredbe `pom = *x;`

u funkciji main



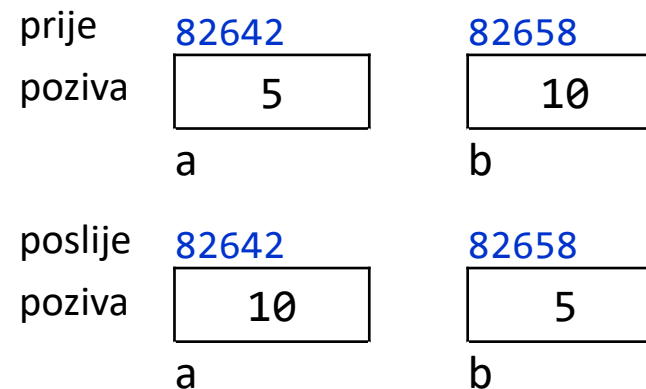
u funkciji zamijeni

Primjer (dodatno objašnjenje)

```
void zamijeni(int *x, int *y) { (2)
    int pom; (3)
    pom = *x; (4)
    *x = *y; (5)
    *y = pom; (6)
    return; (7)
}
```

```
int main(void) {
    int a = 5, b = 10; (1)
    zamijeni(&a, &b);
    ...
}
```

Nakon	a	b	x	y	pom
(1)	5	10	-	-	-
(2)	5	10	82642	82658	-
(3)	5	10	82642	82658	?
(4)	5	10	82642	82658	5
(5)	10	10	82642	82658	5
(6)	10	5	82642	82658	5
(7)	10	5	-	-	-



Primjer

- Programski zadatak
 - napisati funkciju koja će ispisati poruku `Upisite niz >` i zatim s tipkovnice učitati niz znakova (do 20 znakova uključujući znak `\n`). Funkcija treba *vratiti* broj velikih i broj malih slova u učitanoj nizu
 - napisati glavni program koji će pozvati funkciju i ispisati rezultate, u skladu s primjerima izvršavanja programa

```
Upisite niz > Kratica GPS↵  
Broj velikih slova: 4↵  
Broj malih slova: 6
```

```
Upisite niz > 12 3!456↵  
Broj velikih slova: 0↵  
Broj malih slova: 0
```

```
Upisite niz > ↵  
Broj velikih slova: 0↵  
Broj malih slova: 0
```

Funkcija može vratiti najviše jednu vrijednost!

- Ako se kao rezultat funkcije treba *dobiti* više vrijednosti, tada se kao argumenti/parametri moraju koristiti pokazivači
 - funkciji se kao argumenti predaju pokazivači na varijable u koje funkcija treba *upisati* rezultat. U takvom slučaju, kolokvijalno ćemo reći: funkcija *preko pokazivača* treba vratiti [*opis rezultata*]
 - primijenjeno na ovom primjeru: funkcija *preko pokazivača* treba vratiti broj velikih i broj malih slova u učitanoj nizu
 - kada funkcija vraća *jedan* rezultat naredbom `return`, kolokvijalno ćemo reći: funkcija *preko imena* treba vratiti [*opis rezultata*]
 - primijenjeno na primjeru funkcije `fact`: funkcija `fact` *preko imena* treba vratiti $n!$ za zadani broj n
 - ili samo: funkcija treba vratiti [*opis rezultata*]
 - funkcija `fact` treba vratiti $n!$ za zadani broj n

Rješenje

```
#include <stdio.h>
#define MAXNIZ 20

void ucitajPrebroji(int *pBrojVelikih, int *pBrojMalih) {
    char niz[MAXNIZ + 1];

    *pBrojVelikih = *pBrojMalih = 0;

    printf("Upisite niz > ");
    fgets(niz, MAXNIZ + 1, stdin);

    int i = 0;
    while (niz[i] != '\0') {
        if (niz[i] >= 'A' && niz[i] <= 'Z')
            ++*pBrojVelikih; // ili (*pBrojVelikih)++
        else if (niz[i] >= 'a' && niz[i] <= 'z')
            ++*pBrojMalih; // ili (*pBrojMalih)++
        ++i;
    }
    return;
}
```

Rješenje (nastavak)

```
int main(void) {  
    int velika, mala;  
    ucitajPrebroji(&velika, &mala);  
    printf("Broj velikih slova: %d\n", velika);  
    printf("Broj malih slova: %d", mala);  
    return 0;  
}
```


Alternativno rješenje (ali prilično loše)

```
...
int ucitajPrebroji(int *pBrojMalih) {
    char niz[MAXNIZ + 1];
    int brojVelikih = 0;
    *pbrojMalih = 0;

    ...
    while (niz[i] != '\0') {
        if (niz[i] >= 'A' && niz[i] <= 'Z')
            ++brojVelikih;
        else if (niz[i] >= 'a' && niz[i] <= 'z')
            ++*pBrojMalih;
        ++i;
    }
    return brojVelikih;
}

...
velika = ucitajPrebroji(&mala);
```

- ova funkcija *preko imena* vraća broj velikih slova, a *preko pokazivača* broj malih slova
- neprirodno, jer su ta dva rezultata po značenju slični

Primjer

■ Programski zadatak

- napisati funkciju koja će ispisati poruku `Upisite niz >` i zatim s tipkovnice učitati niz znakova (do 20 znakova uključujući znak `\n`). Funkcija *preko pokazivača* treba vratiti broj velikih i broj malih slova u učitanoj nizu, a *preko imena* logičku vrijednost istina ako je u niz učitao bar jedan znak osim `\n`, inače logičku vrijednost laž. Napisati glavni program koji će pozvati funkciju i ispisati rezultate, u skladu s primjerima izvršavanja programa

```
Upisite niz > Kratica GPS↵
```

```
Broj velikih slova: 4↵
```

```
Broj malih slova: 6
```

```
Upisite niz > 12 3!456↵
```

```
Broj velikih slova: 0↵
```

```
Broj malih slova: 0
```

```
Upisite niz > ↵
```

```
Učitao je prazan niz
```

Rješenje

```
...
_Bool ucitajPrebroji(int *pBrojVelikih, int *pBrojMalih) {
    char niz[MAXNIZ + 1];
    _Bool nizSadrziNesto;
    *pBrojVelikih = *pBrojMalih = 0;
    ... ispis poruke i učitavanje
    if (niz[0] == '\\0' || niz[0] == '\\n') {
        nizSadrziNesto = 0;
    } else {
        nizSadrziNesto = 1;

        ... brojanje velikih i malih slova

    }
    return nizSadrziNesto;
}
```

Rješenje (nastavak)

```
int main(void) {
    int velika, mala;
    _Bool nizNijePrazan;
    nizNijePrazan = ucitajPrebroji(&velika, &mala);
    if (nizNijePrazan) { // ili if (ucitajPrebroji(&velika, &mala))
        printf("Broj velikih slova: %d\n", velika);
        printf("Broj malih slova: %d", mala);
    } else {
        printf("Ucitan je prazan niz");
    }
    return 0;
}
```

- rezultati koji su po značenju slični, ovdje su prikladno grupirani
- to ne znači da bi rješenje u kojem bi se sva tri rezultata vraćala preko pokazivača bilo loše

Pokazivač NULL

- Varijabla tipa pokazivača koja se ne inicijalizira na vrijeme je mogući izvor velikih i teško uočljivih logičkih pogrešaka
 - sadrži *garbage value*, što znači da pokazuju "tko zna kamo"
 - pokušaj dereferenciranja će uzrokovati
 - prekid programa (ako smo imali sreće jer je "smeće" pokazivalo na područje memorije koje operacijski sustav štiti)
 - nedefinirano ponašanje programa, moguće drugačije svaki puta kada se pokrene (ako "smeće" pokazuje na područje memorije koje operacijski sustav dopušta čitati i mijenjati našem programu)
- Stoga, za izbjegavanje takvih logičkih pogrešaka, iznimno je važno:
 - svaku varijablu tipa pokazivača inicijalizirati tijekom definicije
 - ako ne znamo na koju vrijednost, inicijalizirajmo je na specijalnu vrijednost pokazivača: NULL
 - pokušaj dereferenciranja te vrijednosti pokazivača će sigurno uzrokovati prekid programa (to je bolje nego nedefinirano ponašanje!)

Pokazivač NULL

- Pokazivač NULL je simbolička konstanta definirana u <stdio.h>
- primjer definicije jedne varijable tipa pokazivač na siguran način

```
#include <stdio.h>
int main(void) {
    int *rez = NULL;
    ...
}
```

Pokazivač NULL

- Pokazivač NULL je također koristan u slučajevima kada u varijablu tipa pokazivač treba upisati neku vrijednost koja u kontekstu ima specijalno značenje
 - Primjer: s tipkovnice učitati cjelobrojne vrijednosti u dvije varijable. U varijablu p1 tipa pokazivača na int upisati pokazivač na onu varijablu koja sadrži veću vrijednost, ali tako da se može dojaviti jesu li vrijednosti jednake

```
int a, b, *p1 = NULL;
```

radi sigurnosti, odmah postaviti na NULL

```
...
```

```
if (a > b)
```

```
    p1 = &a;
```

```
else if (b > a)
```

```
    p1 = &b;
```

```
else
```

```
    p1 = NULL;
```

```
...
```

```
if (p1 == NULL) ...
```

a što ako su vrijednosti jednake?

sada znamo kako se pitati jesu li vrijednosti jednake

Primjer

- Programski zadatak
 - napisati funkciju koja kao parametre prima pokazivače na dva *objekta* tipa float. Funkcija vraća pokazivač na objekt koji sadrži veći broj. Ako su brojevi jednaki funkcija vraća pokazivač NULL
 - napisati glavni program koji će pozvati funkciju i ispisati rezultate, u skladu s primjerima izvršavanja programa

```
Upisite dva broja > 4.1 5.1↵  
Veci broj je 5.100000
```

```
Upisite dva broja > 4.1 4.1↵  
Brojevi su jednaki
```


Rješenje

```
#include <stdio.h>

float *vratiVeceg(float *px, float *py) {
    if (*px > *py)
        return px;
    else if (*py > *px)
        return py;
    else
        return NULL;
}

int main(void) {
    float a, b, *rez = NULL;
    printf("Upisite dva broja > ");
    scanf("%f %f", &a, &b);
    rez = vratiVeceg(&a, &b);
    if (rez == NULL)
        printf("Brojevi su jednaki");
    else
        printf("Veci broj je %f", *rez);
    return 0;
}
```

Viseći pokazivač (*dangling pointer*)

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

double *vratiKorijen(double x) {
    double rez;
    rez = sqrt(x);
    return &rez;
}

int main(void) {
    double *pokNaKorijen = NULL;
    pokNaKorijen = vratiKorijen(4.0);
    printf("Rezultat je %lf", *pokNaKorijen);
    return 0;
}
```

- u čemu je velika pogreška u ovom programu?

- vraća se pokazivač na objekt (varijablu) koja je definirana u funkciji
 - taj objekt više ne postoji kada funkcija završi, što znači da se pokušava ispisati vrijednost objekta koji u trenutku ispisa više ne postoji
 - pokazivač koji pokazuje na takav objekt se naziva *viseći* pokazivač

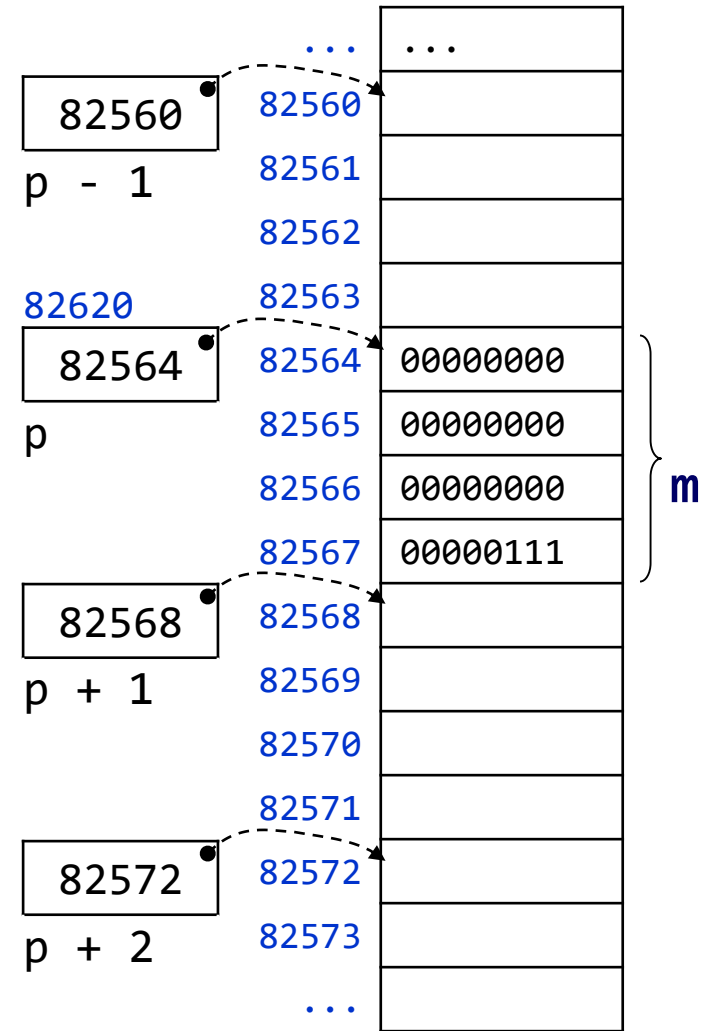
Pokazivači

Aritmetika s pokazivačima

Zbrajanje pokazivača i cijelog broja

```
int m = 7, *p = &m;
```

- uavećavanjem pokazivača za jedan dobije se pokazivač istog tipa, ali koji pokazuje na adresu veću za onoliko bajtova kolika je veličina referenciranog tipa. Npr.
 - $p+1$ je pokazivač na objekt tipa `int` na adresi koja je za 4 bajta veća od adrese upisane u p
- slično
 - $p+2$ je pokazivač na objekt tipa `int` na adresi 8 bajtova većoj od adrese na koju pokazuje p
 - $p-1$ je pokazivač na objekt tipa `int` na adresi 4 bajta manjoj od adrese na koju pokazuje p



Zbrajanje pokazivača i cijelog broja

- slično vrijedi i za ostale tipove podataka

```
char c, *cp = &c;  
short s, *sp = &s;  
int i, *ip = &i;  
double d, *dp = &d;  
long double ld, *ldp = &ld;  
  
printf(" cp = %p    cp + 1 = %p\n",  cp,  cp + 1);  
printf(" sp = %p    sp + 1 = %p\n",  sp,  sp + 1);  
printf(" ip = %p    ip + 1 = %p\n",  ip,  ip + 1);  
printf(" dp = %p    dp + 1 = %p\n",  dp,  dp + 1);  
printf("ldp = %p   ldp + 1 = %p   ", ldp, ldp + 1);
```

cp = 0061ff1b	cp + 1 = 0061ff1c	1 bajt "dalje"
sp = 0061ff18	sp + 1 = 0061ff1a	2 bajta "dalje"
ip = 0061ff14	ip + 1 = 0061ff18	4 bajta "dalje"
dp = 0061ff08	dp + 1 = 0061ff10	8 bajtova "dalje"
ldp = 0061fef0	ldp + 1 = 0061fefc	12 bajtova "dalje"

Smještaj članova polja u memoriji

- Članovi polja su uvijek smješteni u kontinuiranom području memorije, redom jedan član neposredno iza drugog

```
int a[5] = {2, 3, 5, 7, 11};
```

...	82644	82648	82652	82656	82660	...
	2	3	5	7	11	
	a[0]	a[1]	a[2]	a[3]	a[4]	

```
int b[3][4] = {{2, 3, 5, 7},  
               {11, 13, 17, 19},  
               {23, 29, 31, 37}  
               };
```

- članovi dvodimenzijskog polja u memoriji su pohranjeni redak po redak

...	82704	82708	82712	82716	82720	82724	82728	82732	82736	82740	82744	82748	...
	2	3	5	7	11	13	17	19	23	29	31	37	
	b[0][0]	b[0][1]	b[0][2]	b[0][3]	b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[2][0]	b[2][1]	b[2][2]	b[2][3]	

Smještaj članova polja u memoriji

```
int c[2][2][3] = {{{2, 3, 5},  
                  {7, 11, 13}  
                  },  
                  {{17, 19, 23},  
                  {29, 31, 37}  
                  }  
                };
```

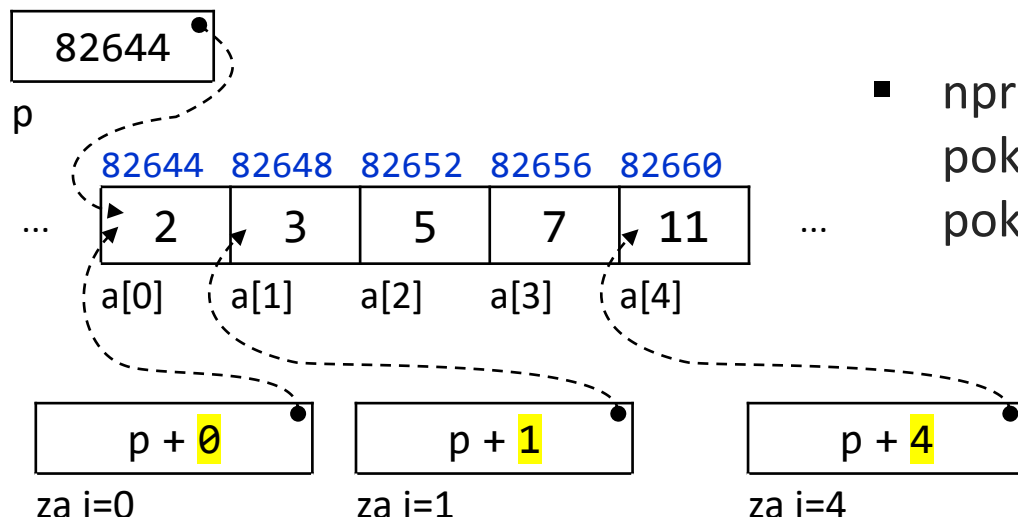
- članovi trodimenzijskog polja u memoriji su pohranjeni sloj po sloj, unutar svakog sloja redak po redak

...	82826	82830	82834	82838	82842	82846	82850	82854	82858	82862	82866	82870	...
	2	3	5	7	11	13	17	19	23	29	31	37	
	c[0][0][0]	c[0][0][1]	c[0][0][2]	c[0][1][0]	c[0][1][1]	c[0][1][2]	c[1][0][0]	c[1][0][1]	c[1][0][2]	c[1][1][0]	c[1][1][1]	c[1][1][2]	

Polja i pokazivači

- prikazana svojstva aritmetike s pokazivačima i način smještaja članova polja u memoriji omogućuju pristup bilo kojem članu polja na temelju pokazivača koji pokazuje na prvi član polja

```
int a[5] = {2, 3, 5, 7, 11};  
int *p = &a[0];
```



- kako pomoću pokazivača na prvi član polja pristupiti članu polja s indeksom `[i]`?
- npr. rezultat operacije `p + 3` je pokazivač tipa pokazivač na `int` koji pokazuje na objekt na adresi `82656`

ako je `p` pokazivač na prvi član polja `a`, tada se članu polja `a[i]` može pristupiti pomoću izraza `*(p + i)`

Primjer

- Na zaslon ispisati članove nekog jednodimenzijskog polja. Članovima polja pristupati pomoću pokazivača.

```
...  
int a[5] = {2, 3, 5, 7, 11};  
int *p = &a[0];  
int i;  
for (i = 0; i < 5; ++i) {  
    printf("%d\n", *(p + i));  
}
```

ili

```
for (i = 0; i < 5; ++i) {  
    printf("%d\n", *p);  
    p = p + 1; ili ++p; ili p++;  
}  
...
```

Ime polja kao pokazivač

- Ime **jednodimenzijskog** polja navedeno u nekom izrazu, kao rezultat će dati pokazivač na prvi član tog polja

```
int a[5] = {2, 3, 5, 7, 11};  
int *p = NULL;  
p = a; će dati isti rezultat kao p = &a[0];  
p = a + 2; će dati isti rezultat kao p = &a[2];
```

- za razliku od varijable *p* čija se vrijednost smije mijenjati (jer varijabla *p* jest *modifiable lvalue*), varijabla *a* ne smije se mijenjati jer predstavlja polje, a polje je *non-modifiable lvalue*

dopušteno

```
p = a;  
for (i = 0; i < 5; ++i) {  
    printf("%d\n", *p);  
    p = p + 1;  
}
```

nije dopušteno

```
for (i = 0; i < 5; ++i) {  
    printf("%d\n", *a);  
    a = a + 1;  
}
```

Primjer

- Programski zadatak
 - s tipkovnice učitati 10 članova cjelobrojnog polja. Na zaslon ispisati vrijednost najvećeg člana. Članovima polja pristupati isključivo preko pokazivača
 - primjer izvršavanja programa

```
Upisite clanove > 1 2 3 4 -1 9 -2 9 8 7↵  
Najveci clan je 9
```

Rješenje

```
#include <stdio.h>
#define DIMENZIJA 10

int main(void) {
    int polje[DIMENZIJA], *p = polje;
    int najveci, i;
    printf("Upisite clanove > ");

    for (i = 0; i < DIMENZIJA; ++i) {
        scanf("%d", p + i); može polje + i ili &polje[0] + i ili &polje[i]
        zbog short circuit evaluation smije se napisati
        if (i == 0 || *(p + i) > najveci) {
            najveci = *(p + i);
        }
    }
    printf("Najveci clan je %d", najveci);
    return 0;
}
```

Rješenje (alternativno)

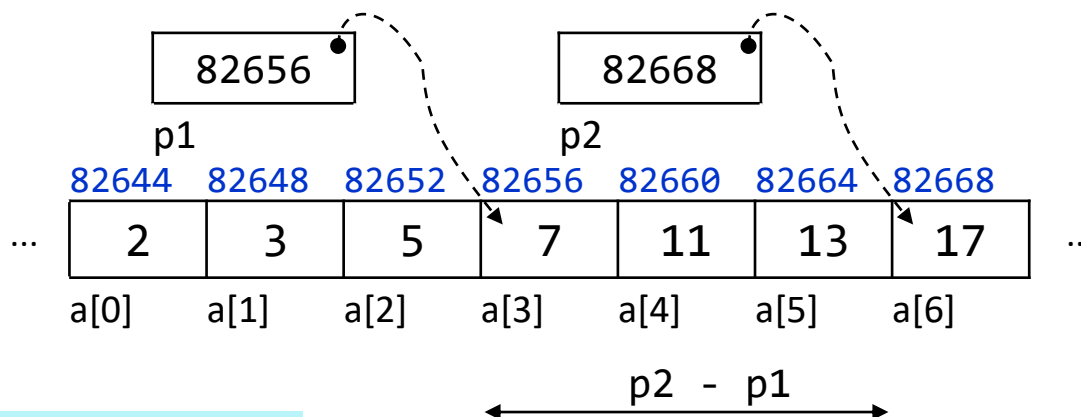
```
#include <stdio.h>
#define DIMENZIJA 10

int main(void) {
    int polje[DIMENZIJA], *p = polje;
    int najveci, i;
    printf("Upisite clanove > ");
    for (i = 0; i < DIMENZIJA; ++i) {
        scanf("%d", p);
        zbog short circuit evaluation smije se napisati
        if (i == 0 || *p > najveci) {
            najveci = *p;
        }
        ++p;
    }
    printf("Najveci clan je %d", najveci);
    return 0;
}
```

Aritmetika s pokazivačima - ostale operacije

- rezultat operacije oduzimanja dva pokazivača, p1 i p2 (koji moraju biti istog tipa) je cijeli broj koji predstavlja "udaljenost adresa" na koju pokazivači pokazuju, ali izraženu u *broju objekata referenciranog tipa* (a ne broju bajtova).

```
int a[7] = {2, 3, 5, 7, 11, 13, 17};  
int *p1 = &a[3];  
int *p2 = &a[6];  
printf("%d %d %d %d", p2 - p1, p1 - p2, p2 - a, *p2 - *p1);
```

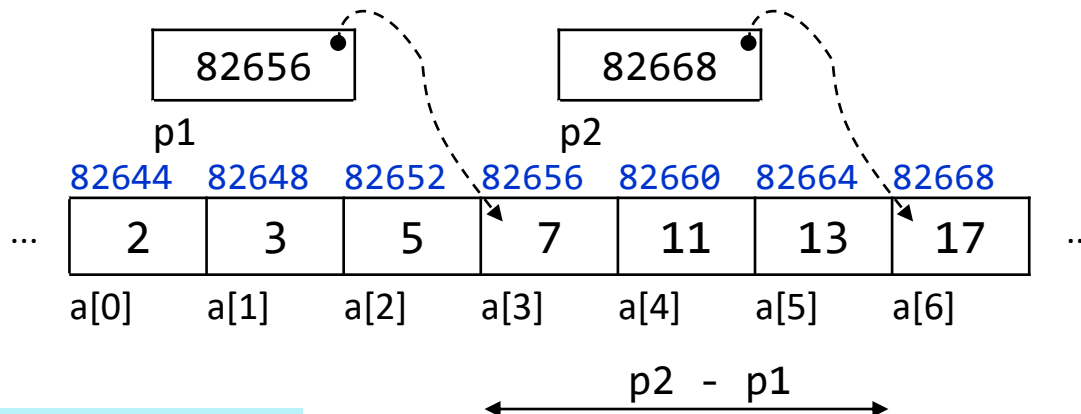


3 -3 6 10

Aritmetika s pokazivačima - ostale operacije

- ako se iz nekog razloga želi izračunati "udaljenost adresa" izražena u bajtovima, treba primijeniti operator pretvorbe (*cast*) u `char *`

```
int a[7] = {2, 3, 5, 7, 11, 13, 17};  
int *p1 = &a[3];  
int *p2 = &a[6];  
printf("%d %d %d %d", (char *)p2 - (char *)p1,  
                    (char *)p1 - (char *)p2,  
                    (char *)p2 - (char *)a,  
                    *p2 - *p1);
```



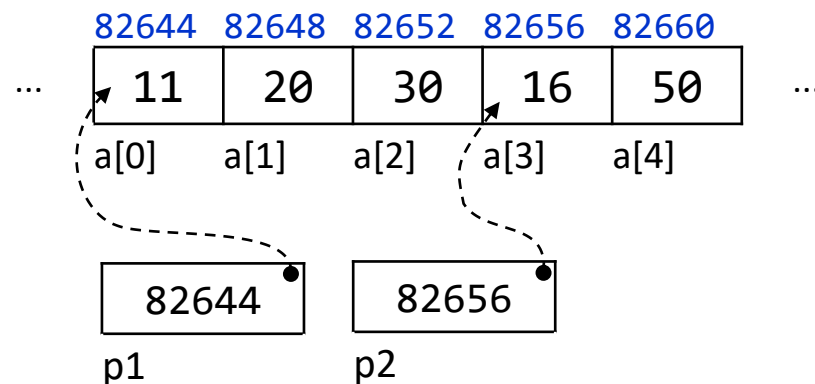
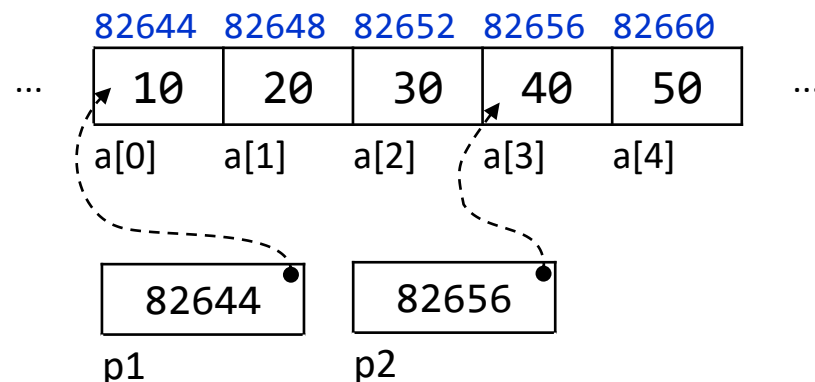
12 -12 24 10

Primjer

```
int a[] = {10, 20, 30, 40, 50};  
int *p1 = &a[0], *p2 = p1 + 3;
```

```
*p2 = ++*p1 + 5;
```

- desna strana:
 - ++*p1 : vrijednost objekta na kojeg pokazuje p1 uvećaj za jedan, rezultat je 11
 - a[0] postaje 11
 - + 5 : ukupni rezultat na desnoj strani: 16
- lijeva strana
 - na mjesto kamo pokazuje p2 upiši 16

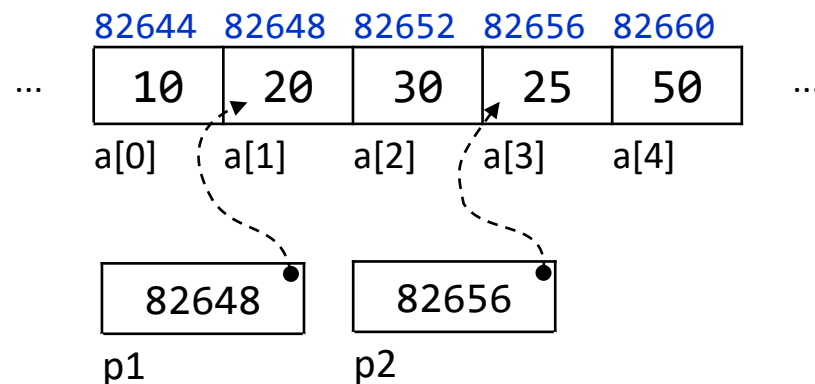
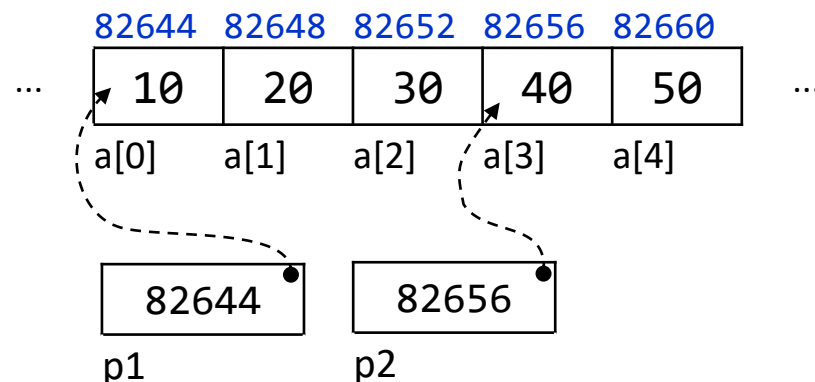


Primjer

```
int a[] = {10, 20, 30, 40, 50};  
int *p1 = &a[0], *p2 = p1 + 3;
```

```
*p2 = *++p1 + 5;
```

- izračunaj desnu stranu:
 - ++p1 : p1 uvećaj za jedan, p1 pokazuje na 82648
 - vrijednost na adresi 82648 je 20, na to dodaj 5, ukupno rezultat na desnoj strani 25
- lijeva strana:
 - na mjesto kamo pokazuje p2 upiši 25

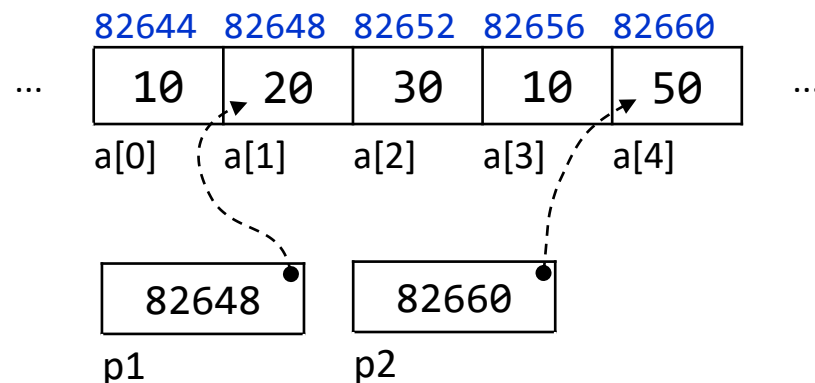
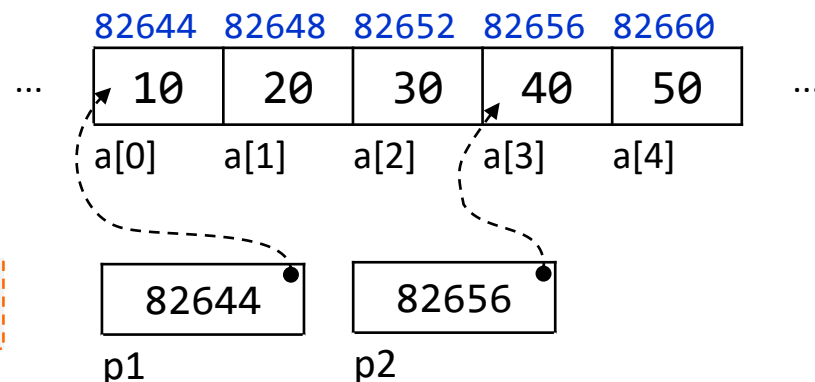


Primjer

```
int a[] = {10, 20, 30, 40, 50};  
int *p1 = &a[0], *p2 = p1 + 3;
```

```
*p2++ = *p1++; tj. *(p2++) = *(p1++);
```

- desna strana:
 - $p1++$: rezultat je pokazivač na 82644 ($p1$ će se kasnije povećati)
 - $*p1++$: rezultat je vrijednost na koju pokazuje 82644, tj. 10
- lijeva strana:
 - $p2++$: rezultat je pokazivač na 82656 ($p2$ će se kasnije povećati)
 - na mjesto kamo pokazuje 82656 upiši 10
- uvećaj $p1$ i $p2$ za 1



Pokazivači

Polja, funkcije i pokazivači

Kako *funkciji* omogućiti pristup članovima polja?

- Polje se ne može koristiti kao argument/parametar funkcije
 - kako onda funkciji omogućiti pristup članovima polja koje je definirano u funkciji na pozivajućoj razini?
- Iskoristiti mogućnost pristupa članovima polja pomoću pokazivača
 - kao argument/parametar navesti **pokazivač** na prvi član polja
 - parametar će biti kopija argumenta, ali to ne smeta: kopija pokazivača će također pokazivati na prvi član tog polja
 - dodati i argument/parametar koji opisuje koliko članova polja ima
 - u funkciji koristiti pokazivač da bi se pristupilo članovima polja koje je definirano u funkciji na pozivajućoj razini
 - ako je p pokazivač na prvi član polja polje, tada se članu polje[i] može pristupiti pomoću pokazivača p, korištenjem izraza $*(p + i)$

Primjer

- Programski zadatak

- napisati funkciju `najveciClan1D` koja kao rezultat vraća najveću vrijednost u zadanom jednodimenzijskom cjelobrojnom polju
- napisati glavni program koji će s tipkovnice učitati 10 članova cjelobrojnog polja, pomoću funkcije odrediti najveći član, te ga ispisati na zaslonu
- primjer izvršavanja programa

```
Upisite clanove > 1 2 3 4 -1 9 -2 9 8 7↵  
Najveci clan je 9
```

- česte pogreške u rješenjima

- funkcija radi samo s poljima od 10 članova, a treba raditi s poljem koje ima bilo koji broj članova
- funkcija na zaslon ispisuje rezultat, a rezultat je trebala *vratiti* u funkciju na pozivajućoj razini

Rješenje

```
#include <stdio.h>
#define DIMENZIJA 10

int najveciClan1D(int *p, int n) {
    int najveci, i;
    for (i = 0; i < n; ++i)
        if (i == 0 || *(p + i) > najveci)
            najveci = *(p + i);
    return najveci;
}

int main(void) {
    int polje[DIMENZIJA];
    /* izostavljen je uobicajeni kod za učitavanje članova polja */
    printf("Najveci clan je %d", najveciClan1D(polje, DIMENZIJA));
    return 0;
}
```

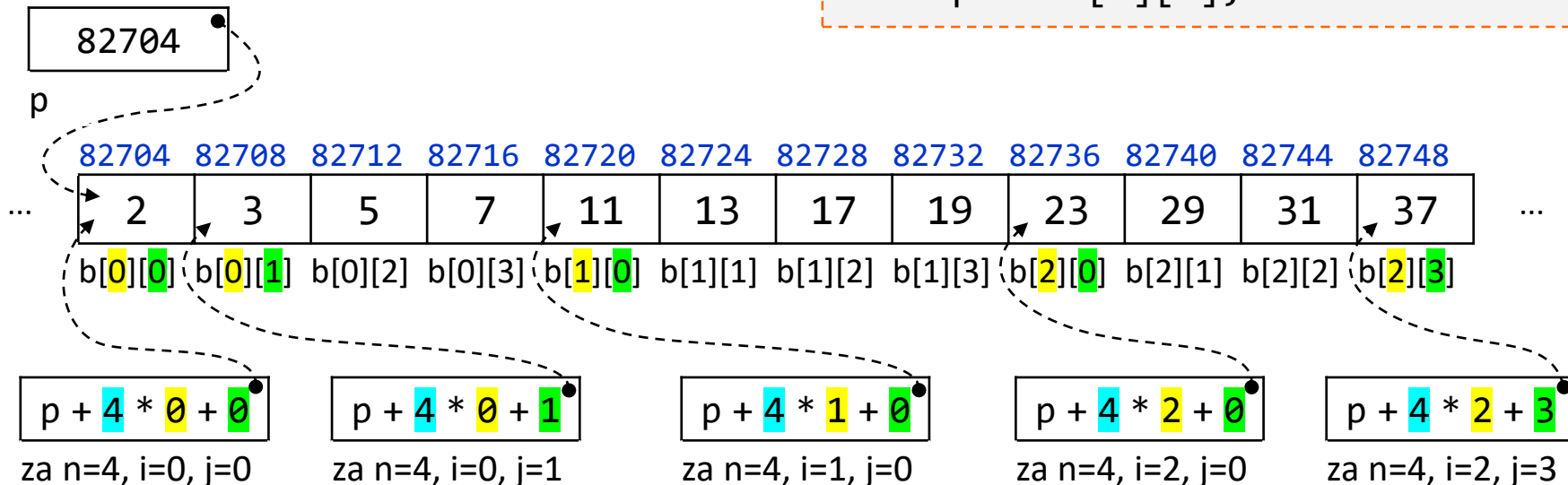
Ne DIMENZIJA!
Ne 10!
Zašto?

ili &polje[0]

Dvodimenzijska polja i pokazivači

- kako pomoću pokazivača na prvi član polja pristupiti članu polja s indeksom $[i][j]$ u polju koje ima m redaka i n stupaca?

```
int b[3][4] = {{2, 3, 5, 7},  
               {11, 13, 17, 19},  
               {23, 29, 31, 37}};  
int *p = &b[0][0];
```



ako je `p` pokazivač na prvi član polja `b` koje ima `n` stupaca, tada se članu polja `b[i][j]` može pristupiti pomoću izraza `*(p + n * i + j)`

Primjer

- Na zaslon ispisati članove nekog dvodimenzijskog polja. Članovima polja pristupati pomoću pokazivača.

```
...
int b[3][4] = {{2, 3, 5, 7},
               {11, 13, 17, 19},
               {23, 29, 31, 37}
               };
int *p = &b[0][0];
int i, j;
for (i = 0; i < 3; ++i) {
    for (j = 0; j < 4; ++j) {
        printf("%5d", *(p + 4 * i + j));
    }
    printf("\n");
}
...
```


Ime dvodimenzijskog polja kao pokazivač?

- Samo ime **dvodimenzijskog** polja navedeno u nekom izrazu ne može se koristiti kao pokazivač na prvi član tog polja

```
int b[3][4] = {{2, 3, 5, 7},  
              {11, 13, 17, 19},  
              {23, 29, 31, 37}  
            };
```

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

```
p = &b[0][0];  ispravno
```

```
p = b[0];     ispravno!
```

```
p = b;        neispravno!
```

Primjer

■ Programski zadatak

- Napisati funkciju `sumeRedaka` koja u zadano jednodimenzijsko polje (članovi tipa `int`) upisuje sume redaka zadanog dvodimenzijskog polja od `m` redaka i `n` stupaca (članovi tipa `int`).
- U glavnom programu definirati matricu dimenzije 3 x 4, s tipkovnice učitati članove, pozvati funkciju te na zaslon ispisati dobiveni rezultat.

		n						
mat	m	1	2	3	4	rez	m	
		5	6	7	8			
		9	10	11	12			
						10		
						26		
						42		

```
Upisite clanove >↵
1 2 3 4↵
5 6 7 8↵
9 10 11 12↵
Sume redaka su:↵
10↵
26↵
42↵
```

Rješenje

```
#include <stdio.h>
#define BR_RED 3
#define BR_STUP 4

void sumeRedaka(int *mat, int m, int n, int *rez) {
    int i, j;
    for (i = 0; i < m; ++i) {
        *(rez + i) = 0;
        for (j = 0; j < n; ++j)
            *(rez + i) += *(mat + n * i + j);
    }
    return;
}

int main(void) {
    int mat[BR_RED][BR_STUP], rez[BR_RED];
    /* izostavljen je uobicajeni kod za učitavanje clanova mat */
    sumeRedaka(&mat[0][0], BR_RED, BR_STUP, &rez[0]);
    /* izostavljen je uobicajeni kod za ispis clanova rez */
    return 0;
}
```

Zašto ovo rješenje nije ispravno?

```
...
int *sumeRedaka(int *mat, int m, int n) {
    int rez[m], i, j;
    for (i = 0; i < m; ++i) {
        rez[i] = 0;
        for (j = 0; j < n; ++j)
            rez[i] += *(mat + n * i + j);
    }
    return &rez[0];
}
...
int mat[BR_RED][BR_STUP], int *rez, i;
/* izostavljen je uobicajeni kod za ucitavanje clanova mat */
rez = sumeRedaka(&mat[0][0], BR_RED, BR_STUP);
for (i = 0; i < m; ++i)
    printf("%d\n", *(rez + i));
...
```

Primjer

- Programski zadatak

- Napisati funkciju `najveciPoRetcima` koja u zadanom dvodimenzijskom polju od `m` redaka i `n` stupaca (članovi tipa `int`) pronalazi najveće vrijednosti članova po retcima. Najveće članove po retcima treba vratiti u jednodimenzijskom polju (članovi tipa `int`)
- Za traženje najveće vrijednosti u pojedinom retku matrice koristiti već viđenu funkciju `najveciClan1D`

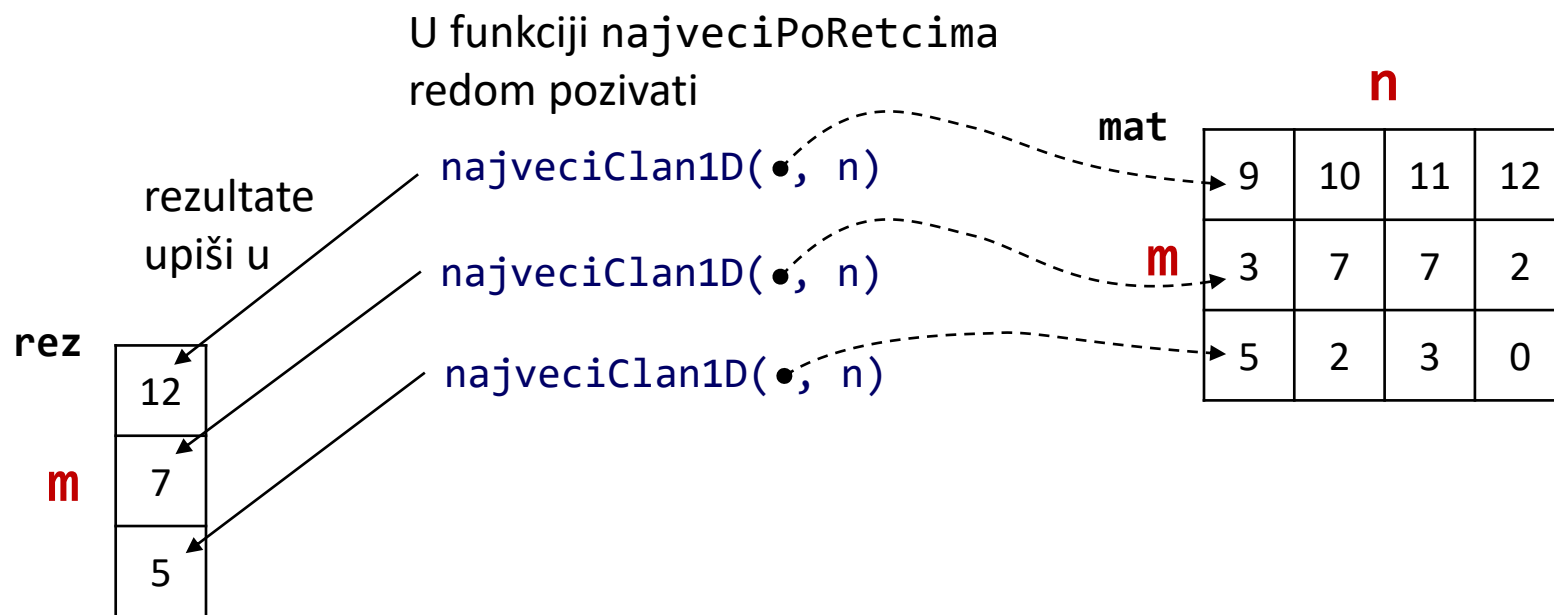
```
int najveciClan1D(int *p, int n) {  
    int najveci, i;  
    for (i = 0; i < n; ++i)  
        if (i == 0 || *(p + i) > najveci)  
            najveci = *(p + i);  
    return najveci;  
}
```

- U glavnom programu učitati dimenzije matrice `m` i `n`, učitati članove matrice, pozvati funkciju te na zaslon ispisati dobiveni rezultat.

Rješenje

■ Ideja

- u funkciji najveciPoRetcima, za svaki redak i matrice mat, funkciju najveciClan1D pozvati s argumentima: pokazivač na prvi član matrice u i-tom retku, duljina retka n. Rezultat funkcije upisati na odgovarajuće mjesto (i-ti član) u polju rez.



Rješenje (nastavak)

```
#include <stdio.h>

int najveciClan1D(int *p, int n) {
    int najveci, i;
    for (i = 0; i < n; ++i)
        if (i == 0 || *(p + i) > najveci)
            najveci = *(p + i);
    return najveci;
}

void najveciPoRetcima(int *mat, int m, int n, int *rez) {
    int i;
    for (i = 0; i < m; ++i) {
        *(rez + i) = najveciClan1D(mat + n * i + 0, n);
    }
    return;
}
```

Rješenje (nastavak)

```
...
int main(void) {
    int m, n; // broj redaka i stupaca matrice mat
    printf("Upisite broj redaka > ");
    scanf("%d", &m);

    printf("Upisite broj stupaca > ");
    scanf("%d", &n);
    int mat[m][n]; // VLA polje!

    /* izostavljen je uobicajeni kod za učitavanje članova mat */
    int rez[m];
    najvećiPoRetcima(&mat[0][0], m, n, &rez[0]);

    /* izostavljen je uobicajeni kod za ispis rezultata (rez) */
    ...
}
```


Pokazivači i operator *sizeof*

- Voditi računa o tipu objekta nad kojim se primjenjuje operator `sizeof`

```
void fun(short *p, double broj) {  
    printf("%d\n", sizeof(p));           4 (ili 8 na Linuxu)  
    printf("%d\n", sizeof(*p));          2  
    printf("%d\n", sizeof(broj));        8  
    ...  
  
int main(void) {  
    short polje[5][10], *p1 = &polje[0][0];  
    double x, *p2 = &x;  
  
    printf("%d\n", sizeof(polje));        100  
    printf("%d\n", sizeof(p1));           4 (ili 8 na Linuxu)  
    printf("%d\n", sizeof(*p1));          2  
    printf("%d\n", sizeof(x));            8  
    printf("%d\n", sizeof(p2));           4 (ili 8 na Linuxu)  
    fun(p1, x);  
    ...  
}
```

Pokazivači i nizovi znakova

- za pohranu niza znakova koristi se jednodimenzijsko polje čiji su članovi tipa char, pri čemu se kraj niza obavezno označava članom polja koji sadrži nul-znak '\0'

```
char niz[6] = "Ivan";  
char *p1 = &niz[0];
```

Sadržaj *niza znakova* može se mijenjati

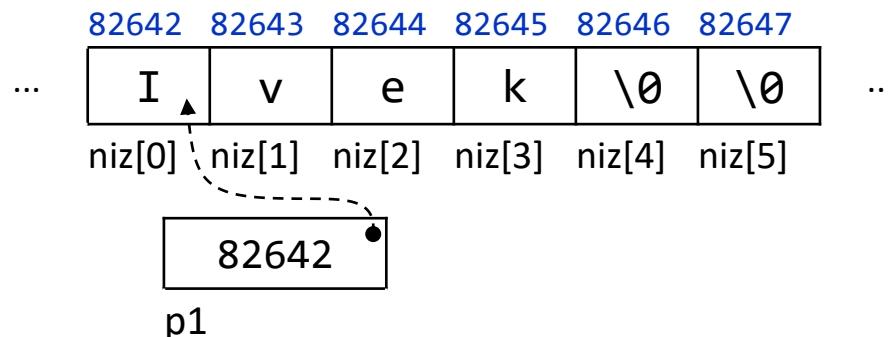
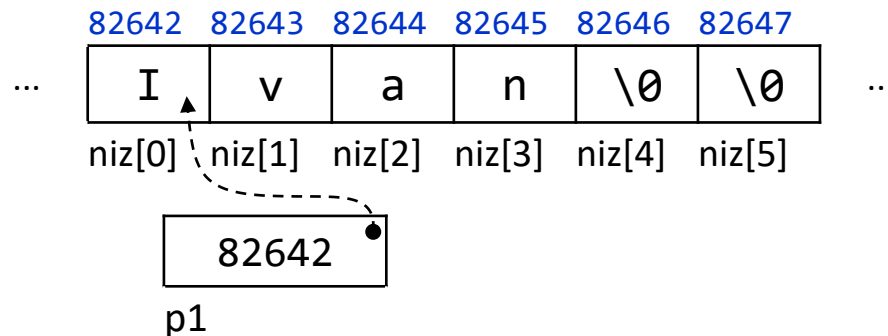
```
*(p1 + 2) = 'e';  
*(p1 + 3) = 'k';
```

ili

```
*(niz + 2) = 'e';  
*(niz + 3) = 'k';
```

ili

```
niz[2] = 'e';  
niz[3] = 'k';
```

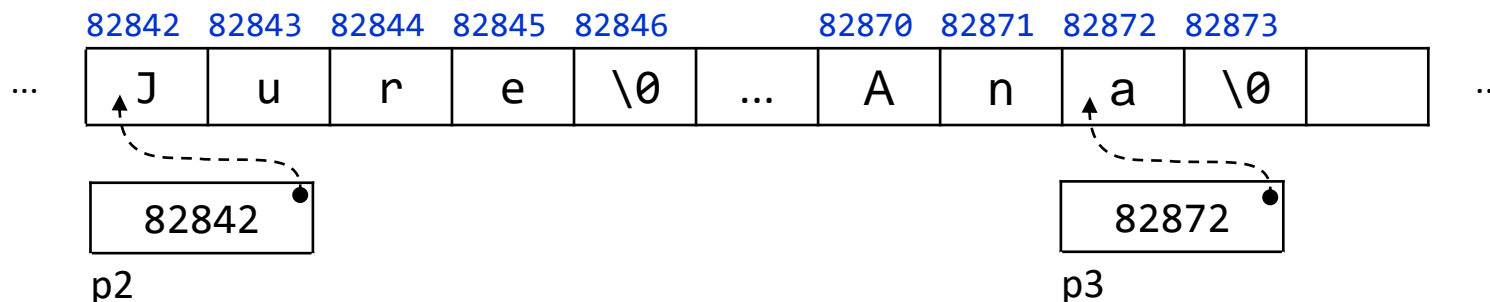


Pokazivači i konstantni znakovni nizovi

- konstantni znakovni niz se u memoriju pohranjuje na jednaki način

```
char *p2 = NULL, *p3 = NULL;  
p2 = "Jure";           "Jure" je konstantni znakovni niz  
p3 = "Ana" + 2;        "Ana" je konstantni znakovni niz
```

- konstantni znakovni niz se u izrazu evaluira kao pokazivač tipa *pokazivač na char* koji pokazuje na prvi znak u tom konstantnom znakovnom nizu



Sadržaj *konstantnog znakovnog niza* ne može se mijenjati

```
*(p2 + 3) = 'a';           nije dopušteno  
*p3 = 'e';                 nije dopušteno  
printf("%c", *p3);        dopušteno
```

Primjer

- Funkciji `printf` kao argument uz konverzijsku specifikaciju `%s` predaje se pokazivač na prvi znak u nizu kojeg treba ispisati. Što će se ispisati navedenim pozivima funkcije `printf`?

```
char niz[] = "Blaise Pascal";
char *p1 = niz;
printf("%s", niz);           Blaise Pascal
printf("%s", p1);           Blaise Pascal
printf("%s", niz + 7);      Pascal
printf("%s", &niz[0] + 7);  Pascal
printf("%s", &niz[7]);      Pascal
printf("%s", p1 + 7);       Pascal

printf("%s", "Isaac Newton"); Isaac Newton
printf("%s", "Isaac Newton" + 4); c Newton

char *p2 = "Benjamin Franklin";
printf("%s", p2);           Benjamin Franklin
printf("%s", p2 + 5);       min Franklin
```

Primjer

- Programski zadatak
 - Napisati funkciju `uVelikaSlova` koja kao parametar prima niz znakova i u njemu svako malo slovo zamijeni velikim
 - U glavnom programu jedan niz znakova inicijalizirati na sadržaj "Ivana 123", pozivom funkcije promijeniti niz i promijenjeni niz ispisati na zaslon

Uputa:

Nizovi znakova su jednodimenzijska polja terminirana znakom `'\0'`.

To znači da se u funkcijama mogu tretirati kao sva ostala jednodimenzijska polja, ali s jednom važnom razlikom: duljinu niza nije potrebno navoditi kao argument jer se duljina niza (ili gdje se nalazi kraj niza) u funkciji može pouzdano utvrditi prema poziciji znaka `'\0'`.

Rješenje

```
#include <stdio.h>

void uVelikaSlova(char *niz) {
    int i = 0;
    while (*(niz + i) != '\0') {
        if (*(niz + i) >= 'a' && *(niz + i) <= 'z') {
            *(niz + i) = *(niz + i) - ('a' - 'A');
        }
        ++i;
    }
}

int main(void) {
    char ime[] = "Ivana 123";
    uVelikaSlova(ime);
    printf("%s", ime);
    return 0;
}
```

Ova se funkcija ne smije pozvati s argumentom koji je konstantni znakovni niz:

`uVelikaSlova("Ivana 123");`

Zašto?

Alternativna rješenja

```
void uVelikaSlova(char *niz) {  
    while (*niz != '\0') {  
        if (*niz >= 'a' && *niz <= 'z') {  
            *niz = *niz - ('a' - 'A');  
        }  
        ++niz;  
    }  
}
```

```
void uVelikaSlova(char *niz) {  
    for (; *niz != '\0'; ++niz) {  
        if (*niz >= 'a' && *niz <= 'z') {  
            *niz = *niz - ('a' - 'A');  
        }  
    }  
}
```

Primjer

■ Programski zadatak

- Napisati funkciju `nadjiPrviZnak` koja kao parametre prima niz znakova `niz` i znak `z`. Ako znak `z` postoji u nizu, funkcija vraća pokazivač na prvi pronađeni znak `z` u nizu, inače, funkcija treba vratiti *prikladan rezultat* na temelju kojeg će se moći prepoznati da tog znaka u nizu nema
- U glavnom programu s tipkovnice pročitati niz znakova ne dulji od 50 znakova, pročitati znak `z`, a zatim pomoću funkcije utvrditi gdje se taj znak nalazi te ispisati niz od pronađenog znaka do kraja niza

```
Upisite niz   > Nigdar ni tak bilo da ni nekak bilo↵
Upisite znak > n↵
ni tak bilo da ni nekak bilo
```

```
Upisite niz   > pak ni vezda ne bu da nam nekak ne bu↵
Upisite znak > B↵
Znak B ne postoji u nizu
```


Rješenje

```
#include <stdio.h>

char *nadjPrviZnak(char *niz, char z) {
    while (*niz != '\0') {
        if (*niz == z)
            return niz;
        ++niz;
    }
    return NULL;
}

int main(void) {
    char niz[50 + 1], z, *rez = NULL;
    printf("Upisite niz > "); fgets(niz, 50 + 1, stdin);
    printf("Upisite znak > "); scanf("%c", &z);
    rez = nadjPrviZnak(niz, z);
    if (rez != NULL)
        printf("%s", rez);
    else
        printf("Znak %c ne postoji u nizu", z);
    return 0;
}
```

Primjer

■ Programski zadatak

- Napisati funkciju `nadopuniNiz` koja kao parametre prima nizove znakova `niz1` i `niz2`. Funkcija treba promijeniti `niz1` tako da na njegov kraj dopiše sadržaj niza `niz2`.
- U glavnom programu prvi niz inicijalizirati na "Sadrzaj prvog niza" i ispisati ga na zaslon, drugi niz inicijalizirati na "Sadrzaj drugog niza" i ispisati ga na zaslon, pozvati funkciju i na zaslon ispisati novi sadržaj prvog niza

```
Sadrzaj prvog niza↵  
Sadrzaj drugog niza↵  
Sadrzaj prvog nizaSadrzaj drugog niza
```

Rješenje

```
#include <stdio.h>

void nadopuniNiz(char *niz1, char *niz2) {
    while (*niz1 != '\0')
        ++niz1;
    while (*niz2 != '\0') {
        *niz1 = *niz2;
        ++niz1;
        ++niz2;
    }
    *niz1 = '\0';
}

int main(void) {
    char niz1[37 + 1] = "Sadrzaj prvog niza";
    char niz2[] = "Sadrzaj drugog niza";
    printf("%s\n", niz1);
    printf("%s\n", niz2);
    nadopuniNiz(niz1, niz2);
    printf("%s", niz1);
    return 0;
}
```

Osigurati dovoljno memorije za dopunjavanje niza niz1



Pokazivači

Strukture, funkcije i pokazivači

Struktura jest *modifiable lvalue*

- To znači da se struktura može koristiti kao argument/parametar funkcije, a također se može koristiti i kao rezultat funkcije. Npr.

```
struct tocka_s {double x; double y;};  
  
struct tocka_s  
translatiraj(struct tocka_s tocka, double dx, double dy) {  
    struct tocka_s novaTocka;  
    novaTocka.x = tocka.x + dx;  
    novaTocka.y = tocka.y + dy;  
    return novaTocka;  
}  
  
int main(void) {  
    struct tocka_s t1 = {3.0, 4.0}, t2;  
    t2 = translatiraj(t1, 2.0, -1.0);  
    printf("%lf, %lf => %lf, %lf", t1.x, t1.y, t2.x, t2.y);  
}
```

3.000000, 4.000000 => 5.000000, 3.000000

Alternativno (pomoću definicije tipa strukture)

```
typedef struct {double x;  
                double y;  
            } tocka_t ;  
  
tocka_t translatiraj(tocka_t tocka, double dx, double dy) {  
    tocka_t novaTocka;  
    novaTocka.x = tocka.x + dx;  
    novaTocka.y = tocka.y + dy;  
    return novaTocka;  
}  
  
int main(void) {  
    tocka_t t1 = {3.0, 4.0}, t2;  
    t2 = translatiraj(t1, 2.0, -1.0);  
    printf("%lf, %lf => %lf, %lf", t1.x, t1.y, t2.x, t2.y);  
}
```

Alternativno (iskoristiti parametar)

- Rješenje u kojem funkcija vraća promijenjeni parametar
 - uštedio se prostor na stogu koji se u prethodnom rješenju trošio na lokalnu varijablu novaTocka

```
struct tocka_s {double x; double y;};  
struct tocka_s  
translatiraj(struct tocka_s tocka, double dx, double dy) {  
    tocka.x += dx;  
    tocka.y += dy;  
    return tocka;  
}
```

- U oba rješenja argument (varijabla t1) se nije promijenio
 - međutim, što ako položaj točke kakav je bio prije translacije nije potrebno pamtiti? Npr.

```
t1 = translatiraj(t1, 2.0, -1.0);
```

Pokazivač na strukturu

- Funkcija kao parametar može koristiti pokazivač na strukturu

```
struct tocka_s {double x; double y};  
void translaticiraj(struct tocka_s *pTocka, double dx, double dy) {  
    (*pTocka).x += dx;  
    (*pTocka).y += dy;  
    return;  
}  
  
int main(void) {  
    struct tocka_s t1 = {3.0, 4.0};  
    printf("%lf, %lf", t1.x, t1.y);  
    translaticiraj(&t1, 2.0, -1.0);  
    printf(" => %lf, %lf", t1.x, t1.y);  
}
```

- umjesto cijele strukture, na stog se u ovom slučaju kopira samo pokazivač na strukturu, a funkcija pomoću dobivenog pokazivača mijenja sadržaj varijable t1

Alternativno (pomoću definicije tipa strukture)

```
typedef struct {double x;  
                double y;  
            } tocka_t;  
  
void translaticaj(tocka_t *pTocka, double dx, double dy) {  
    (*pTocka).x += dx;  
    (*pTocka).y += dy;  
    return;  
}  
  
int main(void) {  
    tocka_t t1 = {3.0, 4.0};  
    printf("%lf, %lf", t1.x, t1.y);  
    translaticaj(&t1, 2.0, -1.0);  
    printf(" => %lf, %lf", t1.x, t1.y);  
}
```

Operator pristupa članu strukture preko pokazivača

- Binarni operator **->**. U literaturi također: strelica, *arrow operator*
 - lijevi operand je pokazivač na strukturu, desni operand je ime člana strukture, rezultat operacije je član strukture

```
struct osoba_s {  
    char prezime[40+1];  
    char ime[40+1];  
    int visina;  
};  
  
struct osoba_s osoba, *pOsoba = &osoba;  
scanf("%s", (*pOsoba).prezime);      ili pOsoba->prezime  
scanf("%s", (*pOsoba).ime);           ili pOsoba->ime  
scanf("%d", &(*pOsoba).visina);       ili &pOsoba->visina  
...  
printf("%s", (*pOsoba).prezime);      ili pOsoba->prezime  
printf("%s", (*pOsoba).ime);          ili pOsoba->ime  
printf("%d", (*pOsoba).visina);       ili pOsoba->visina
```

Član strukture može biti polje

- I struktura koja sadrži polje smije se koristiti kao parametar
 - Primjer: podaci o studentu i bodovima stečenim na predmetu UPRO pohranjeni su u sljedećoj strukturi

```
struct bodovi_s {  
    char jmbag[13 + 1];  
    char ime[50 + 1];  
    char prezime[50 + 1];  
    float mi;  
    float zi;  
    float lab[8];  
};
```

Ukupno 156 bajtova

- napisati funkciju koja kao parametar prihvata prikazanu strukturu, a kao rezultat vraća cijeli broj koji predstavlja odgovarajuću ocjenu
 - korištenje strukture koja sadrži polje kao parametar funkcije nije zabranjeno, ali treba voditi računa o utrošku memorije (stog)

Rješenje

```
#include <stdio.h>
struct bodovi_s {
    char jmbag[13 + 1]; ...
};

int izracunajOcjenu(struct bodovi_s bodovi) {
    int suma = bodovi.mi + bodovi.zi + bodovi.lab[...] ...
    if (suma >= 50.0f && suma < 62.5f) ocjena = 2; else ...
    return ocjena;
}

int main(void) {
    struct bodovi_s bodoviHorvat;
    ...
    printf("Ocjena je %d", izracunajOcjenu(bodoviHorvat));
    return 0;
}
```

Alternativno rješenje koje *štedi* stog

```
#include <stdio.h>
struct bodovi_s {
    char jmbag[13 + 1]; ...
};

int izracunajOcjenu(struct bodovi_s *bodovi) {
    int suma = bodovi->mi + bodovi->zi + bodovi->lab[...] ...
    if (suma >= 50.0f && suma < 62.5f) ocjena = 2; else ...
    return ocjena;
}

int main(void) {
    struct bodovi_s bodoviHorvat;
    ...
    printf("Ocjena je %d", izracunajOcjenu(&bodoviHorvat));
    return 0;
}
```

Ne zloupotrebljavati to svojstvo struktura!

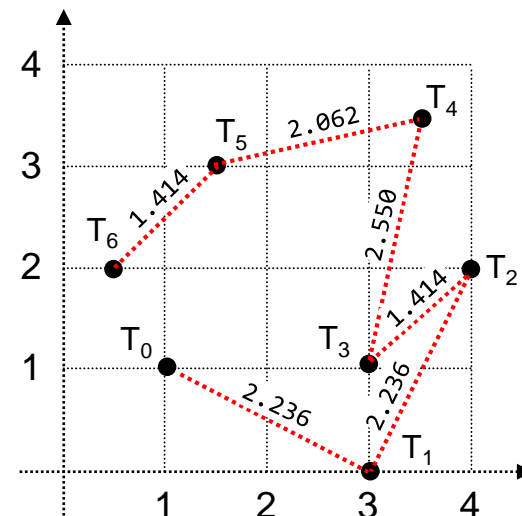
- U vrlo lošim programima ta se mogućnost zloupotrebljava tako što se "polje omota strukturom" s jedinim ciljem da ga se "prenese u funkciju". To je **vrlo loša** ideja i tako se **ne smije** raditi.

```
struct omotanoPolje_s {  
    int polje[1000];  
};  
  
int najveciClan1D(struct omotanoPolje_s omotanoPolje, int n) {  
    ...  
    return najveci;  
}  
  
int main(void) {  
    struct omotanoPolje_s omotanoPolje;  
    ...  
    printf("Najveci je %d", najveciClan1D(omotanoPolje, n));  
}
```

Za koliko bajtova se poveća sadržaj
stoga kada se pozove funkcija?

Primjer

- Programski zadatak
 - Linija je u Kartezijevom koordinatnom sustavu opisana nizom točaka. Za pohranu podataka o jednoj točki (koordinate x i y, tipa double) koristi se struktura `tocka_s`. Podaci o točkama koje predstavljaju liniju pohranjeni su u jednodimenzijskom polju čiji su članovi strukture `tocka_s`.



```
struct tocka_s {double x;  
                double y;  
};
```

```
struct tocka_s linija[n];
```

x	1.0	x	3.0	x	4.0	x	3.0	x	3.5	x	1.5	x	0.5
y	1.0	y	0.0	y	2.0	y	1.0	y	3.5	y	3.0	y	2.0

Primjer

- Programski zadatak (nastavak)
 - Napisati funkciju koja izračunava ukupnu duljinu linije koja je predstavljena jednodimenzijskim poljem čiji su članovi strukture tocka_s. U glavnom programu učitati broj i koordinate točaka, pozivom funkcije izračunati, a zatim na zaslon ispisati duljinu linije
- Primjer izvršavanja programa

```
Upisite broj tocaka linije > 7↵
Upisite koordinate tocke T0 > 1.0 1.0↵
Upisite koordinate tocke T1 > 3.0 0.0↵
Upisite koordinate tocke T2 > 4.0 2.0↵
Upisite koordinate tocke T3 > 3.0 1.0↵
Upisite koordinate tocke T4 > 3.5 3.5↵
Upisite koordinate tocke T5 > 1.5 3.0↵
Upisite koordinate tocke T6 > 0.5 2.0↵
Ukupna duljina linije je      11.912 (jed.mj.)
```


Rješenje (1. dio)

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

struct tocka_s {double x;
                double y;
                };

/* izracunava udaljenost medju tockama t1 i t2 */
double udaljenost(struct tocka_s *t1, struct tocka_s *t2) {
    return sqrt(pow(t2->x - t1->x, 2.) + pow(t2->y - t1->y, 2.));
}

/* izracunava duljinu linije odredjene tockama u polju linija */
double duljinaLinije(struct tocka_s *linija, int n) {
    double duljina = 0.;
    for (int i = 1; i < n; ++i) {
        duljina += udaljenost(linija + i, linija + i - 1);
    }
    return duljina;
}
```

Rješenje (2. dio)

```
int main(void) {
    int n;
    printf("Upisite broj tocaka linije > ");
    scanf("%d", &n);
    struct tocka_s linija[n];
    /* ucitaj koordinate za n tocaka */
    for (int i = 0; i < n; ++i) {
        printf("Upisite koordinate tocke T%d > ", i + 1);
        scanf("%lf %lf", &linija[i].x, &linija[i].y);
    }
    printf("Ukupna duljina linije je %9.3f (jed.mj.)"
        , duljinaLinije(linija, n));
    return 0;
}
```