Uvod u programiranje

- predavanja -

prosinac 2025.

15. Pokazivači

Vodič „12. prije 15. predavanja”

- Očekuje se da ste savladali sljedeće pojmove (na razini na kojoj je obrađeno u vodiču):
  - adresa varijable
  - definicija i pridjeljivanje vrijednosti pokazivaču
  - dereferenciranje pokazivača
  - generički (void *) pokazivač

- Pitanja?

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 referenciji (*pass by reference* ili *call by reference*)

- U nekim jezicima (npr. Pascal, Fortran) u parametre je moguće prenijeti referencije 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);          //a = 10, b = 5    ISPRAVAN REZULTAT!
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 referencije 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

U jeziku C ne postoji prijenos po referenciji

- Ali zato postoji zamjena za mehanizam prijenosa po referenciji
  - 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 referencijama (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

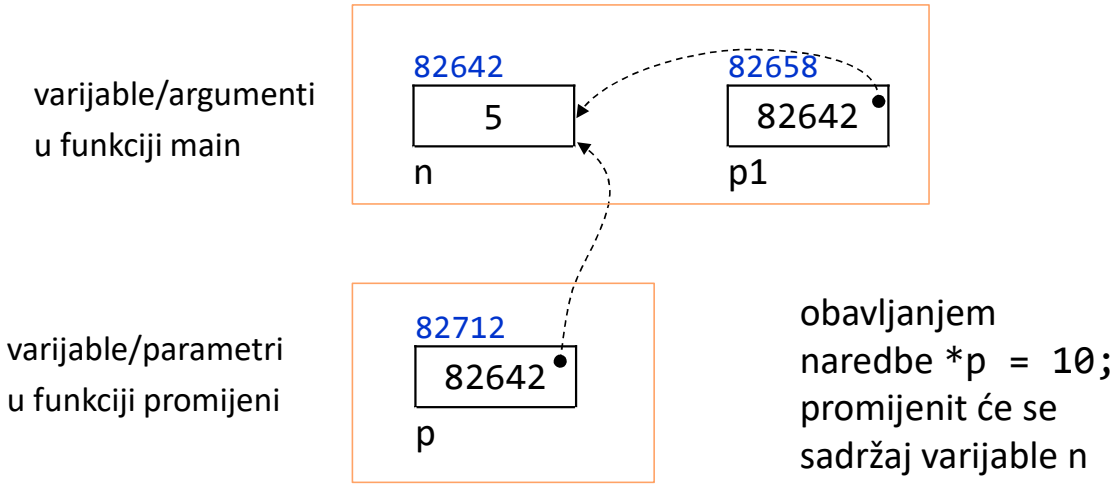
```
void promijeni(int *p) {
   *p = 10;
   return;
}
```

```
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;`



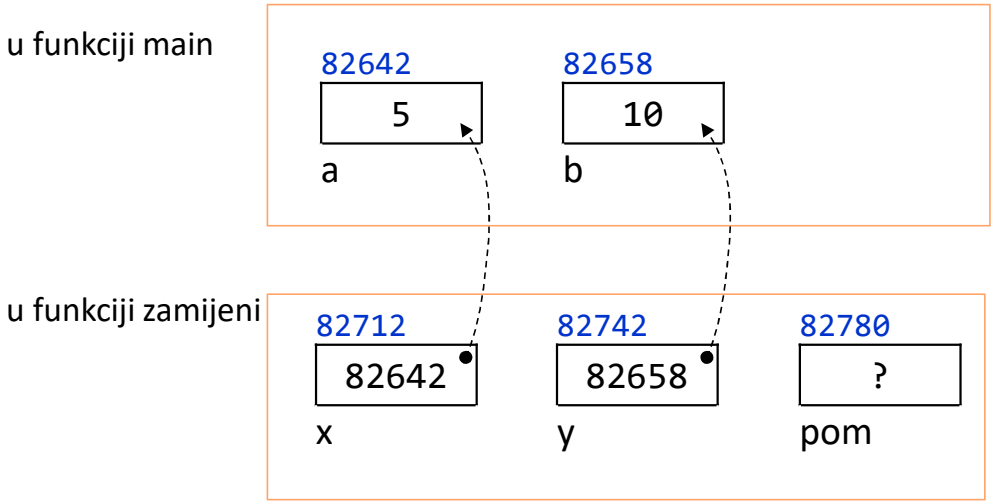
obavljanjem naredbe `*p = 10;` promijenit će se sadržaj varijable n

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;`

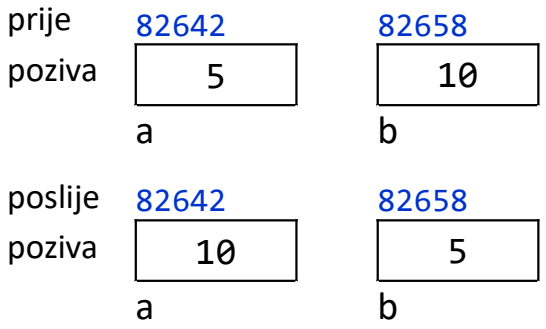


Primjer (dodatno objašnjenje)

```
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 učitanom 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 učitanom 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 učitanom nizu, a *preko imena* logičku vrijednost istina ako je u niz učitan 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 > ↵
Ucitan je prazan niz
```

Rješenje

```
...
_Bool ucitajPrebroji(int *pBrojVelikih, int *pBrojMalih) {
   char niz[MAXNIZ + 1];
   _Bool nizSadrziNesto;

   *pBrojVelikih = *pBrojMalih = 0;

   ... ispis poruke i ucitavanje

   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 pokazuje "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                           a što ako su vrijednosti jednake?
   p1 = NULL;
...
if (p1 == NULL) ...            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

Zadatak

Znanstveni zapis (notacija) realnog broja je zapis koji se sastoji od realnog broja kojem je decimalna točka postavljena nakon prve znamenke različite od 0 i potencije broja 10.
Primjerice, znanstvena notacija broja 341,5 je $3{,}415 \cdot 10^2$.

a) Napišite funkciju ***znanstveniZapis*** koja prima pozitivni realni broj standardne preciznosti i *preko pokazivača* vraća realni dio (realni broj standardne preciznosti) i eksponent (cijeli broj) znanstvenog zapisa toga broja.

b) Napišite glavni program koji će učitati pozitivni realni broj standardne preciznosti, pozvati funkciju ***znanstveniZapis***, koja će vratiti realni dio i eksponent znanstvenog zapisa broja, nakon čega će glavni program ispisati broj u znanstvenoj notaciji.

```
Upisite broj > 22656.125↵
Znanstveni zapis: 2.26561236382 10^4
```

```
Upisite broj > 0.00008865734↵
Znanstveni zapis: 8.86573314667 10^-5
```

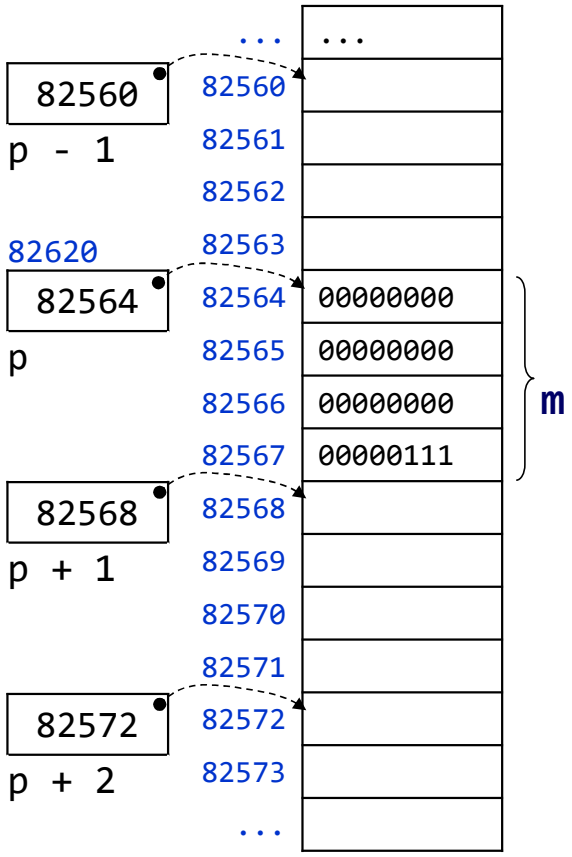
Pokazivači

Aritmetika s pokazivačima

Zbrajanje pokazivača i cijelog broja

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

- uvećavanjem 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

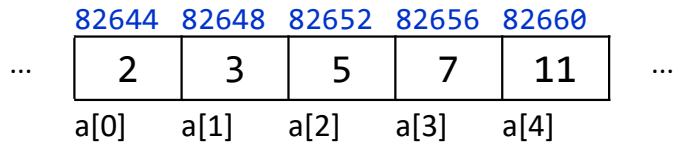
```
char c, *cp = &c;
short s, *sp = &s;
int i, *ip = &i;
double d, *dp = &d;
long double ld, *ldp = &ld;

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

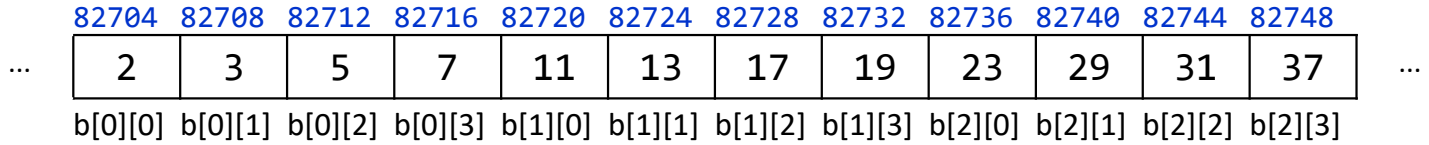
- Članovi polja su uvijek smješteni u kontinuiranom području memorije, redom jedan član neposredno iza drugog

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




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

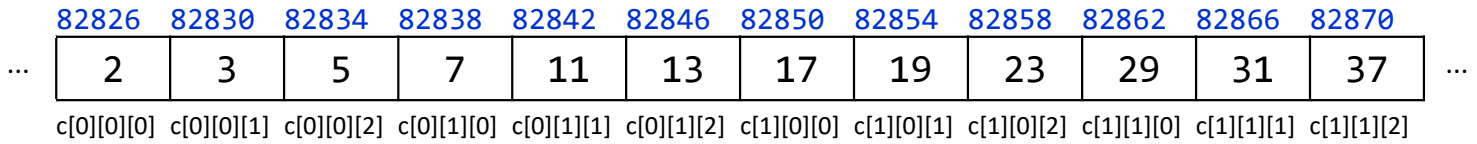
- članovi dvodimenzijskog polja u memoriji su pohranjeni redak po redak



Smještaj članova polja u memoriji

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

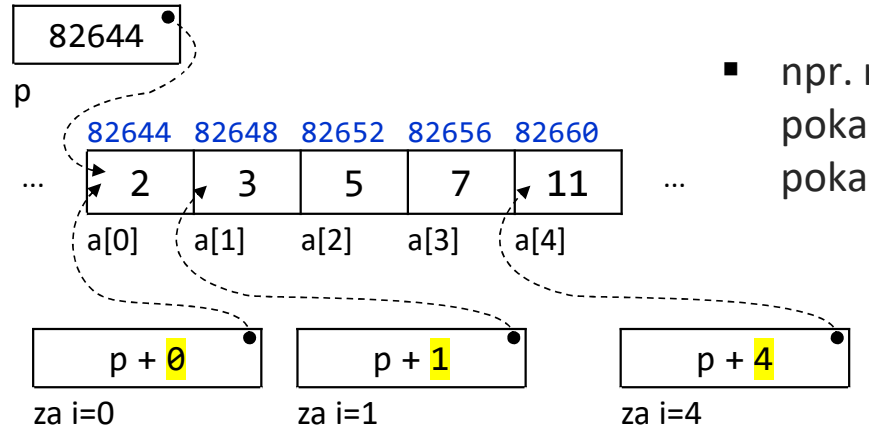
- članovi trodimenzijskog polja u memoriji su pohranjeni sloj po sloj, unutar svakog sloja redak po redak



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;
}
```

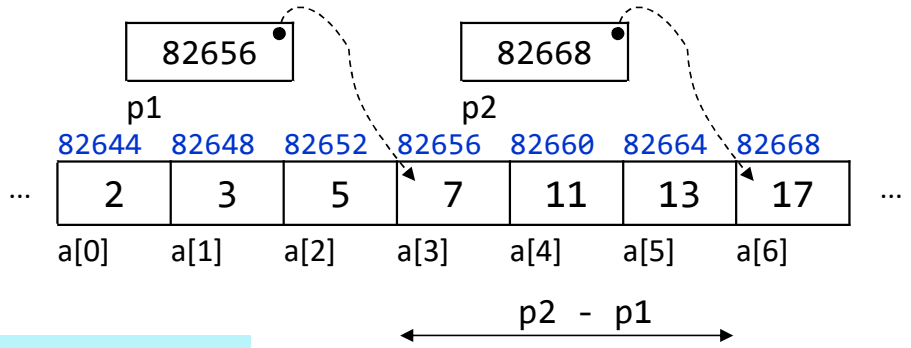
Rješenje (alternativno)

```
#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);
```

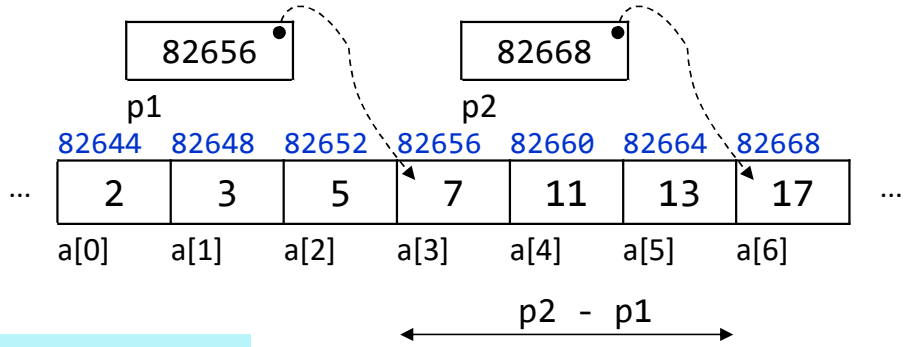



```
3  -3  6  10
```

Aritmetika s pokazivačima - ostale operacije

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

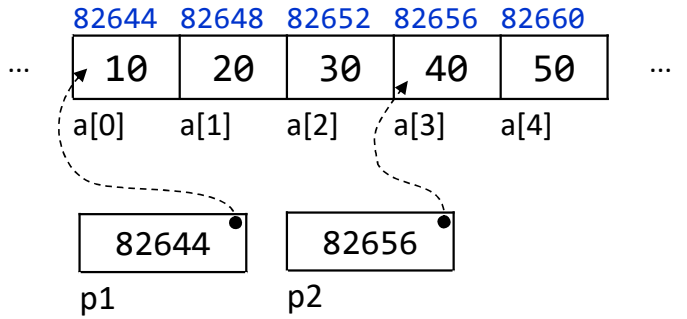
```
int a[7] = {2, 3, 5, 7, 11, 13, 17};
int *p1 = &a[3];
int *p2 = &a[6];
printf("%d  %d  %d  %d", (char *)p2 - (char *)p1,
                         (char *)p1 - (char *)p2,
                         (char *)p2 - (char *)a,
                         *p2 - *p1);
```




```
12  -12  24  10
```

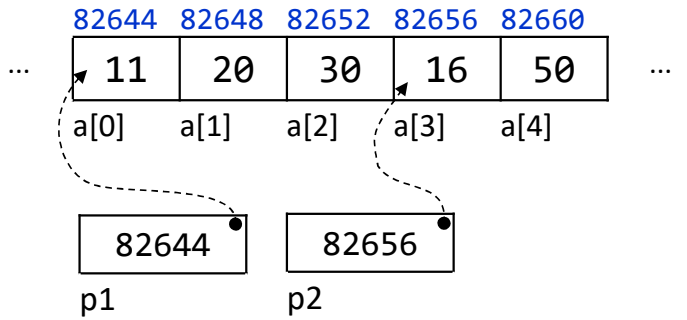
Primjer

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



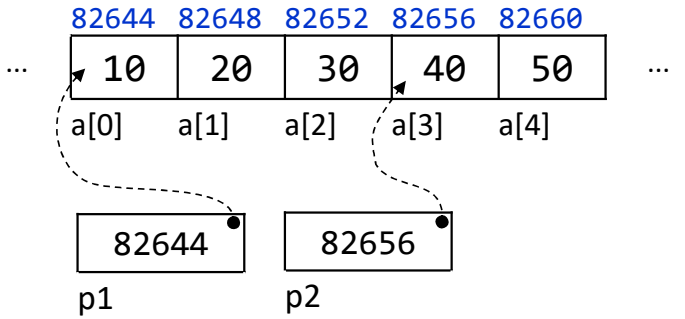

```
*p2 = ++*p1 + 5;
```

- 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



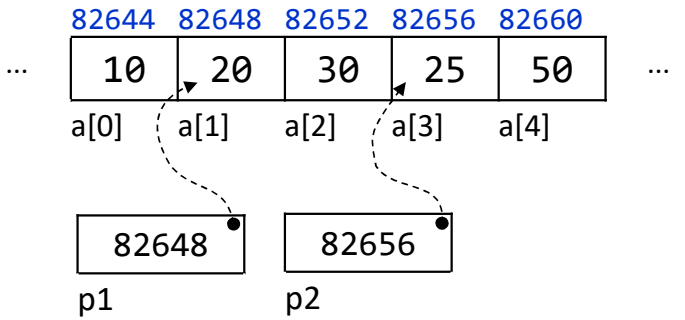
Primjer

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



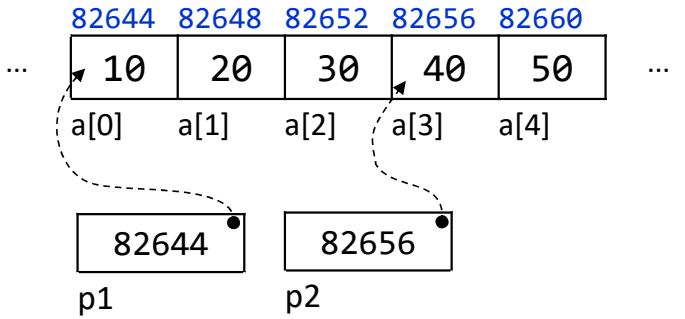

```
*p2 = *++p1 + 5;
```

- 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



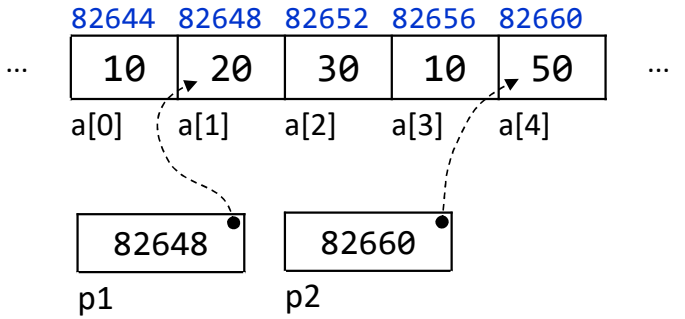
Primjer

```
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



Zadatak

S tipkovnice učitati niz znakova (do 20 znakova uključujući znak \n).) koji predstavlja lozinku za koju treba izračunati koliko je sigurna za korištenje („jaka“) na sljedeći način:

	Broj znakova	težina		Primjer: 3ra-laa#l@	Broj znakova	težina
Ukupno znakova	n	n*4			n = 10	40
Znamenki	z	z*4	+		z = 1	+ 4
Specijalnih znakova !#$%&()*+`-./:;<=>?@	sz	sz*6	+		sz = 3	+ 18
Znamenki i specijalnih znakova koji nisu na početku/kraju	zsz	zsz*2	+		zsz = 2	+ 4
Zadovoljeno pravila	p	p*2	+		p = 3	+ 6
		Rezultat =			Rezultat =	72

Na zaslon ispisati izračunati brojčani rezultat.

Članovima polja pristupati pomoću pokazivača.

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **nema**
  - **15. vježbe uz predavanja**