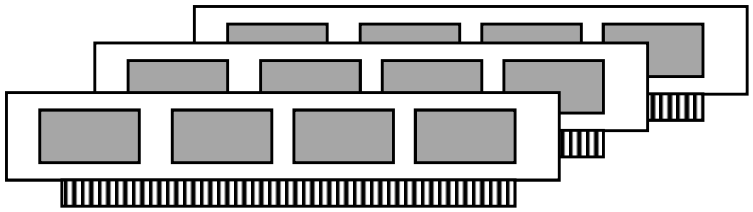
Programiranje i programsko inženjerstvo

Predavanja

2014. / 2015.

7. Pokazivači (*pointers*)

Organizacija radne memorije (*RAM*)

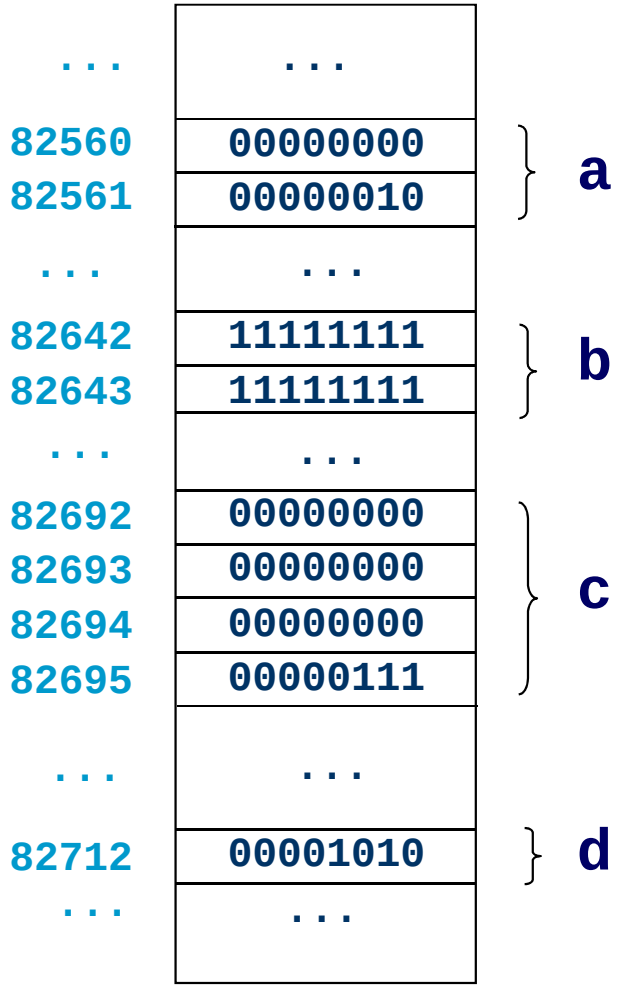


Memorija računala zapravo je kontinuirani niz bajtova: svaki bajt ima svoj "redni broj", tj. adresu. Memorija od 512 MB može se promatrati na sljedeći način:

0	00110001
1	11010010
...	...
82560	11000001
82561	00001101
82562	00111000
82563	11010101
82564	00000000
82565	10001000
82566	10101011
82567	11101000
82568	11111111
82569	01000010
...	...
536870910	00110001
536870911	00000111

Definicija "običnih" varijabli

Definicijom varijabli rezervira se prostor u memoriji na nekim adresama, npr.:



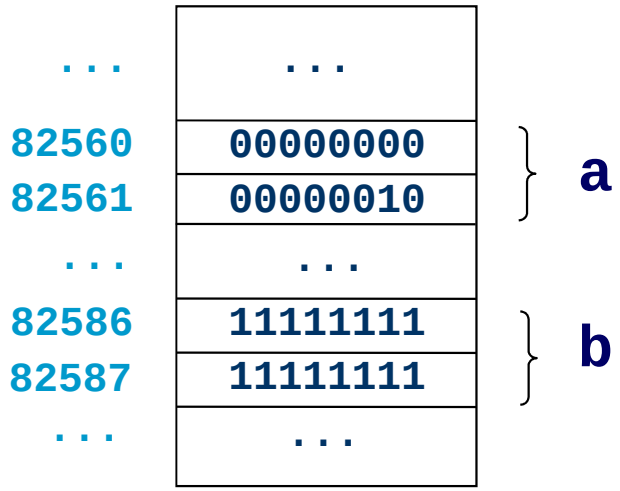

```
short a, b;
int c;
char d;
a = 2;
b = -1;
c = 7;
d = '\n';
```

Za varijablu kažemo da je "na adresi x" ako je prvi bajt sadržaja varijable pohranjen na adresi x.

Primjer: adresa varijable **c** je **82692**.

Adresa ili pokazivač?

Za adresu (*address*) također se koristi pojam pokazivač (*pointer*) zato što adresa "pokazuje" na neki objekt u memoriji.

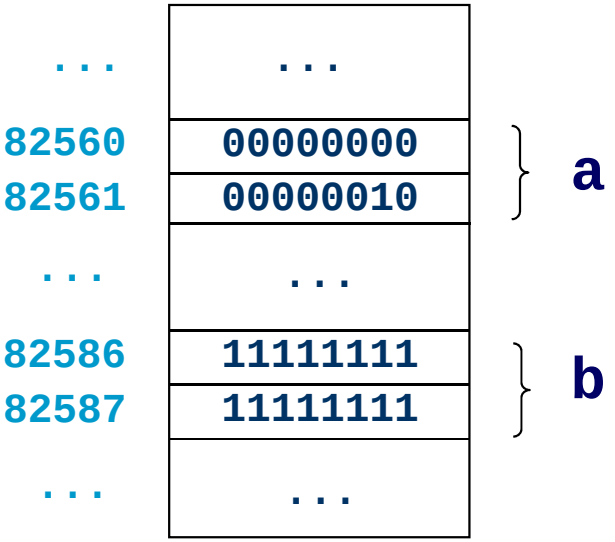



```
short a, b;
a = 2;
b = -1;
```

82586 "pokazuje" na jedan objekt tipa **short** (varijablu **b**)

Kako saznati adresu varijable? (Adresni operator)

Ako je **x** varijabla, tada je **&x** njena adresa u memoriji:




```
short a = 2, b = -1;
printf("%p   %p", &a, &b);
```

Ispis:

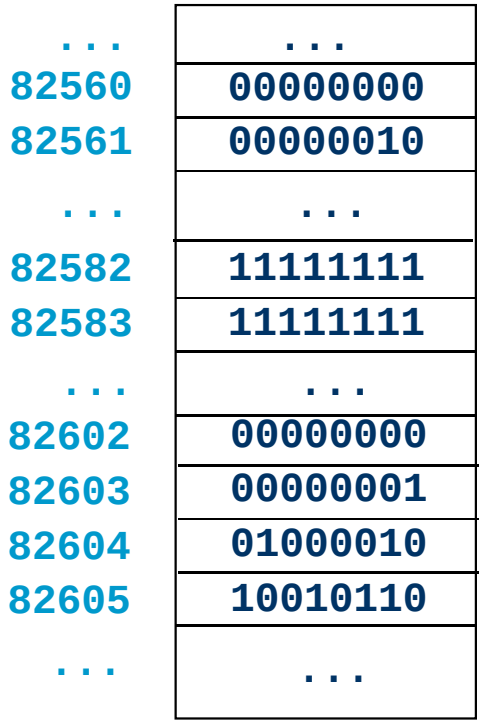
```
82560   82586
```

Kojeg su "tipa" ove dvije adrese? Iako "izgleda" kao **int**, adresa u općem slučaju nije **int** (niti **short**, niti **long**). Zato **nema smisla** adresu pohraniti u varijablu tipa **int**.

~~`int a, p;`~~
~~`p = &a;`~~

Kamo pohraniti adresu (pokazivač)?

Za pohranu adrese varijable koja je tipa **short**, mora se koristiti varijabla posebnog tipa (pokazivač na **short**):




```
short a = 2, b = -1;
short *p1;
p1 = &b;
printf("%p", p1);
```

Ispis:

82582

p1 je varijabla u koju je moguće smjestiti adresu objekta koji je tipa **short** (tj. pokazivač na objekt tipa **short**). Radi pojednostavljenja, kažemo "**p1** je pokazivač na **short**".

Definiranje pokazivača

Pokazivači se definiraju slično kao "obične" varijable.

* ispred imena varijable znači da se ta varijabla koristi za pohranu pokazivača na objekt odgovarajućeg tipa.

```
short a;    /* a sluzi za pohranu vrijednosti tipa short */
int b;      /* b sluzi za pohranu vrijednosti tipa int */
short *p1; /* p1 je pokazivac na short */
int *p2;    /* p2 je pokazivac na int */
```

Dopušteno je u istoj naredbi definirati i "obične" varijable i pokazivače:

```
short a, *p1;
int b = 5, *p2;
int *p3 = &b;
```

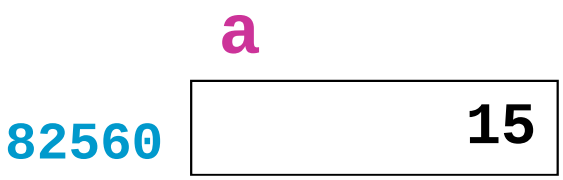
Tipovi pokazivača

Pokazivače na objekte jednog tipa nije dopušteno koristiti kao pokazivače na objekte nekog drugog tipa.

```
short a = 2, b = -1;
int c = 7;
short *p1;
int *p2;
p1 = &a;    /* O.K. */
p1 = &b;    /* O.K. */
p2 = &c;    /* O.K. */
p2 = &b;    /* ne valja! */
```

Pristupanje objektu na kojeg pokazuje pokazivač

```
int a = 15;
```



Koji je sadržaj **a**?

```
int *p;
```

Na što pokazuje **p**?

```
p = &a;
```

Na što pokazuje **p**?

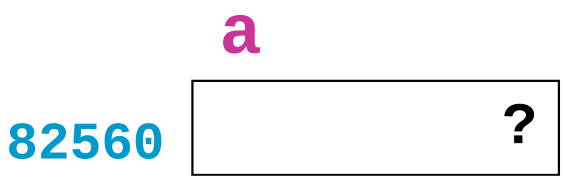
***p** → rezultat je **int** vrijednost na koju pokazuje p (tj. vrijednost koja se nalazi na adresi **82560**)

```
printf("%d", *p);
```

→ **15**

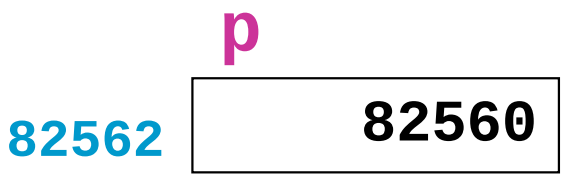
Izmjena vrijednosti objekta na kojeg pokazuje pokazivač

```
short a;
```



Koji je sadržaj **a**?

```
short *p = &a;
```



Na što pokazuje **p**?

```
*p = 16;
```

kao vrijednost objekta na kojeg pokazuje **p**, tj. na adresu **82560**, upiši vrijednost **16**.

Koji je sadržaj **a**?

Primjer:

pretpostavka o adresama varijabli - a: 8560, b: 8564, c: 8568

	a	b	c	ap	bp	cp
`int a=1, b=2, c=3;`	1	2	3			
`int *ap, *bp, *cp;`	1	2	3	?	?	?
`*ap = 100;` **ap** pokazuje na "tko zna što". Naredbom se zapisuje vrijednost 100 na "tko zna koju adresu". **Ne činiti to!**						
`ap = &a;`	1	2	3	8560	?	?
`bp = ap;`	1	2	3	8560	8560	?
`*bp = 200;`	200	2	3	8560	8560	?
`c = *ap;`	200	2	200	8560	8560	?
`cp = &c;`	200	2	200	8560	8560	8568
`a = 300;`	300	2	200	8560	8560	8568
`*cp = *ap + 10;`	300	2	310	8560	8560	8568

Primjer:

pretpostavka o adresama varijabli - x: 8560, y: 8564

```
int x = 5, y = 10;
int *px, *py;
px = &x;   /* px: 8560 */
py = &y;   /* py: 8564 */
```

Što će se ispisati?

a)
```
*px = *py;
printf("%d  %d  %p  %p", x, y, px, py);
```
→ 10 10 8560 8564

b)
```
px = py;
printf("%d  %d  %p  %p", x, y, px, py);
```
→ 5 10 8564 8564

Aritmetika s pokazivačima

Iako "izgleda" kao **`int`**, pokazivač nije **`int`** (niti **`short`**, niti **`long`**). Što je s aritmetičkim operacijama s pokazivačima?

Operacije koje "imaju smisla" su:

- pokazivaču pribrojiti cijeli broj
- od pokazivača oduzeti cijeli broj

```
char c, *cp = &c;   /* neka je cp: 38000 */
short s, *sp = &s;  /* neka je sp: 48000 */
int i, *ip = &i;    /* neka je ip: 58000 */
double d, *dp = &d; /* neka je dp: 68000 */
```

`cp + 1` → `38001`	`cp - 1000` → `37000`
`sp + 1` → `48002`	`sp - 1000` → `46000`
`ip + 1` → `58004`	`ip - 1000` → `54000`
`dp + 1` → `68008`	`dp - 1000` → `60000`

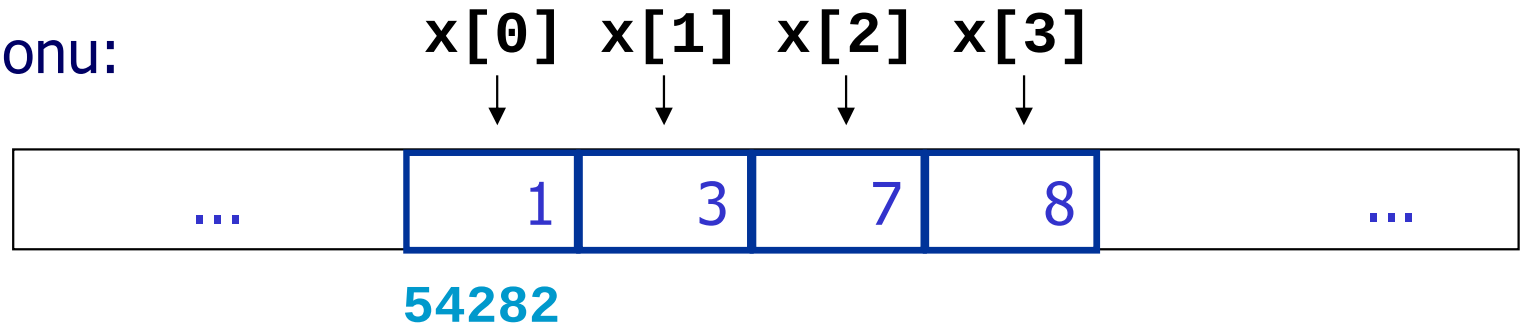
Polja i pokazivači

```
#include <stdio.h>
int main (void) {
  int x[4] = {1, 3, 7, 8};
  int *p = &x[0]; /* 54282 */
  printf("%d %d %d %d", *p, *(p+1), *(p+2), *(p+3));
  return 0;
}
```

54286 54290 54294

Ispis na zaslonu:

1 3 7 8



Što bi se dogodilo da se npr. prije **`return 0;`** obave naredbe:

```
*(p+4) = 1000;
*(p-1) = 1000;
```

Primjer

- U jednodimenzijsko realno polje (niz) učitati 10 vrijednosti. Ispisati vrijednosti članova polja. Članovima polja treba pristupati preko pokazivača.

```
Upisite elemente polja: 5 2 -1 2.5 0 0 1 1 9.1 -2
polje[0] =  5.0
polje[1] =  2.0
polje[2] = -1.0
polje[3] =  2.5
polje[4] =  0.0
polje[5] =  0.0
polje[6] =  1.0
polje[7] =  1.0
polje[8] =  9.1
polje[9] = -2.0
```

Rješenje

```
#include <stdio.h>
int main (void) {
   float x[10], *p;
   int i;
   p = &x[0];

   printf ("Upisite elemente polja: ");
   for (i = 0; i < 10; ++i)
      scanf("%f", p+i);

   for (i = 0; i < 10; ++i)
      printf ("polje[%d] = %4.1f\n", i, *(p+i));
   return 0;
}
```

zašto se u ovom slučaju u **scanf** ne koristi adresni operator **&**

- koja je bitna razlika u odnosu na sljedeće rješenje:

```
      printf ("polje[%d] = %4.1f\n", i, *(p++));
```

Korištenje varijable tipa polje umjesto pokazivača na prvi element polja

Ako je **x** varijabla definirana kao **jednodimenzijsko** polje, tada se adresa prvog člana polja može dobiti na dva načina:

```
&x[0]
x
```

U prethodnom primjeru, umjesto `p = &x[0];` može `p = x;`

Napomena: ako je **x** varijabla tipa polje s dvije ili više dimenzija, tada se **x** ne može koristiti kao adresa prvog člana polja **x**

Ako je **x** varijabla definirana kao **dvodimenzijsko** polje, tada se adresa prvog člana polja **x** može dobiti na dva načina:

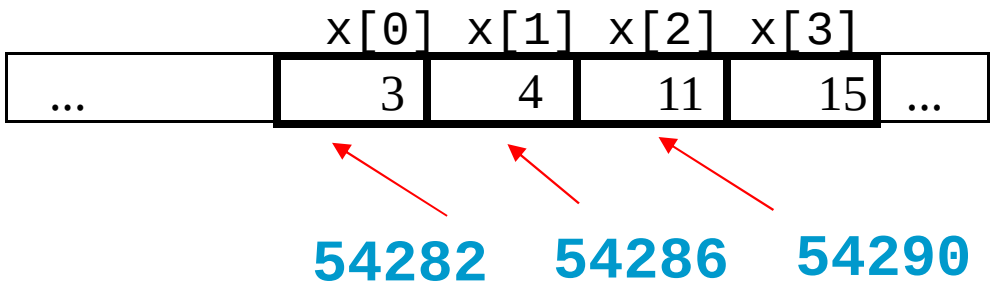
```
&x[0][0]
x[0]
```

Ako je **x** varijabla definirana kao **trodimenzijsko** polje, tada se adresa prvog člana polja **x** može dobiti na dva načina:

```
&x[0][0][0]
x[0][0]
```

"Oduzimanje" pokazivača

```
int x[4] = {3, 4, 11, 15};
```




```
int *p1 = &x[0]; /* 54282 */
int *p2 = &x[2]; /* 54290 */
printf("%d", p2 - p1);
```

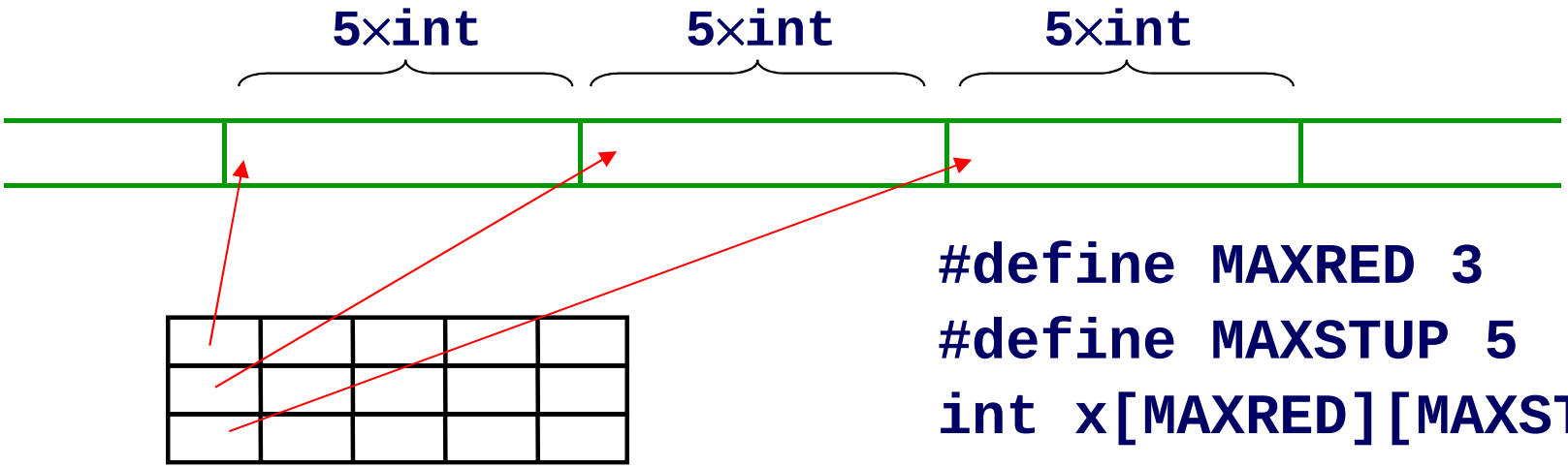
Znamo da je p1+2 = p2

Koliko je onda p2 - p1 ?

```
/* sto ce se ispisati sljedecom naredbom? */
printf("%d", *p2 - *p1);
```

Dvodimenzijska polja i pokazivači

- Dvodimenzijsko polje: pohranjuje se redak za retkom




```
#define MAXRED 3
#define MAXSTUP 5
int x[MAXRED][MAXSTUP]
```

3 susjedna jednodimenzijska polja veličine 5 × `int`

```
int *p = &x[0][0];
*(p + 0*MAXSTUP + 0)    → vrijednost člana x[0][0]
*(p + 0*MAXSTUP + 1)    → vrijednost člana x[0][1]
*(p + 1*MAXSTUP + 0)    → vrijednost člana x[1][0]
*(p + 2*MAXSTUP + 0)    → vrijednost člana x[2][0]
*(p + 2*MAXSTUP + 3)    → vrijednost člana x[2][3]
```

Primjer

- S tipkovnice učitati vrijednosti za broj redaka mr ≤ 10 i broj stupaca ms ≤ 5. Učitati vrijednosti članova dvodimenzijskog realnog polja od mr redaka i ms stupaca. Učitano polje ispisati u obliku tablice i zatim ispisati vrijednost najvećeg člana u svakom retku. Članovima polja treba pristupati preko pokazivača.

```
Upisite mr i ms: 4 3
1 2 3
6 5 4
9 10 8
3 1 2
 1.00  2.00  3.00
 6.00  5.00  4.00
 9.00 10.00  8.00
 3.00  1.00  2.00
```

```
Najveci clanovi po retcima:
U 0. retku najveci je  3.00
U 1. retku najveci je  6.00
U 2. retku najveci je 10.00
U 3. retku najveci je  3.00
```

Rješenje
- 1. dio -

```
#include <stdio.h>
#define MAXRED 10
#define MAXSTUP 5

int main(void) {
  int mr, ms, i, j;
  float najveci, polje[MAXRED][MAXSTUP];
  float *p = &polje[0][0];

  /* ucitavati mr i ms dok ne budu ispravni */
  do {
      printf("Upisite mr i ms: ");
      scanf("%d %d", &mr, &ms);
  } while (mr < 1 || mr > MAXRED ||
           ms < 1 || ms > MAXSTUP);
```

Rješenje
- 2. dio -

```
/* ucitaj polje od mr redaka i ms stupaca */
for (i = 0; i < mr; ++i) {
  for (j = 0; j < ms; ++j) {
    scanf("%f", p + i*MAXSTUP + j);
  }
}
/* Ispisi polje */
for (i = 0; i < mr; ++i) {
  for (j = 0; j < ms; ++j) {
    printf("%5.2f ", *(p + i*MAXSTUP + j));
  }
  printf ("\n");       /* skok u novi red */
}
```

Rješenje
- 3. dio -

```
  /* Ispis najvecih clanova u retcima */
  printf("Najveci clanovi po retcima:\n");

  for (i = 0; i < mr; ++i) {
    najveci = *(p + i*MAXSTUP); /* prvi clan retka i */
    for (j = 1; j < ms; ++j)
      if (*(p + i*MAXSTUP + j) > najveci)
        /* clan retka i stupca j */
        najveci = *(p + i*MAXSTUP + j);
    /* Ispis vrijednosti nadjenog najveceg clana */
    printf("U %d. retku najveci je %5.2f\n", i, najveci);
  }
  return 0;
}
```