Основи програмирања 1

Лекција 5

др Зоран Бањац

zoran.banjac@viser.edu.rs

Висока школа електротехнике и рачунарства струковних студија
Београд

Садржај

Условни скокови

- break
- continue

Показивачи

- Дефинисање показивача
  - оператор **&**
  - оператор *
  - Додела почетне вредности показивачу
  - Генерички показивачи
- Приказивање вредности показивача

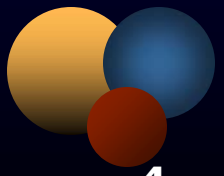
Условни скокови

- `break`
- `continue`
- ~~`goto`~~

`break`

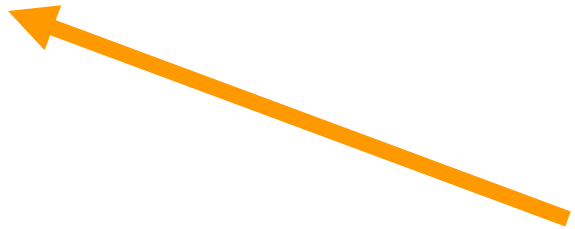
- **`break`** наредба се користи за превремено ПРЕКИДАЊЕ итеративних петљи (**`while`**, **`for`** и **`do-while`**), као и наредбе **`switch`**

- Уколико постоји више наредби итерације, које су уметнуте једна у другу, наредба **`break`** прекида извршење само једне од њих и то оне у којој се налази



break

```
for(i=0; ; ++i)
{
  ...
  x = i-3+12*i
  if( x >= 500)
      break;
  ...
}
...
```



break

```
#include <stdio.h>
main()
{
 int c;
 printf("Unesite karkter( x za izlaz):  ");
 while(1)
 {
   c = getc(stdin);
   if (c == 'x')
      break;
 }
 printf("Prekid beskonacne petlje!\n");
}
```

break - switch

```
main()
{
  int a = 1;
  int b = 2;
  switch( a+b )
  {
   case 12:
      printf("Zbir je 12\n");
      break;
   case 3:
      printf("Zbir je 3\n");
      break;
   case 41:
      printf("Zbir je 41\n");
      break;
   default:
      printf("Zbir je nepoznat\n");
      break;
  }
}
```

```
Zbir je 3
```

break - switch

```
main()
{
   int a = 1;
   int b = 2;
   switch( a+b )
   {
    case 1:
        printf("Zbir je 1\n"); break;
    case 2:
        printf("Zbir je 2\n"); break;
    case 3:
    case 4:
    case 5:
      printf("2 < Zbir < 6\n"); break;
    default:
        printf("Zbir je nepoznat\n"); break;
   }
}
```

2 < Zbir < 6

continue

- Наредба **continue**, за разлику од наредбе **break** НЕ прекида извршење целокупне наредбе итерације (**while**, **for** и **do-while**) већ прекида само једну њену (тренутну) итерацију.

`continue`

- Код **`while`** петље доводи до скока на тестирање
- Код **`do-while`** петље доводи до скока на тестирање
- Код **`for`** петље доводи до скока на **`izraz3`** (инкремент) и **`izraz2`** (тестирање)

continue

```
#include <stdio.h>
main()
{
   int x;
   printf("Ispis parnih brojeva do 10\n");

   for( x = 1; x <= 10; x++ )
   {
      if( x % 2 != 0 )/* Ako broj nije paran */
         continue; /* Idi na pocetak petlje */
      printf( "\n%2d", x );
   }
   printf( "\n");
}
```

```
2 4 6 8 10
```

continue

```
#include <stdio.h>
main()
{
  int h = 200, brojac = 0;

  printf("Brojevi od %d do 0\n", h);
  printf("koji nisu deljivi sa 2 i 3:\n");
  do
  {
   if ( h%2 == 0)  /* izostavi deljive sa 2 */
         continue;
   if (h % 3 == 0) /* ... sa 3 */
         continue;

   printf("%5d", h), brojac++;
   if (brojac % 10 == 0)  /* 10 u liniji */
         putchar('\n');
  }while (h--);
}
```

Показивачи

- Показивач (***pointer***) је градивни део програмског језика ***C*** који нуди већу контролу над меморијом рачунара.

- Показивач је променљива која може да садржи меморијску адресу друге променљиве.

Показивачи

- Тип податка:
  скуп вредности (**short int** од -32768 до 32767)
  скуп операција (сабирање, одузимање...).
- До сада поменути типови података су имали своје име (int, float, double...)
- Вредности које може да има податак типа показивач су **адресе меморијских локација.**
- Тип податка показивач **нема своје име**.

Показивачи

- Број бајтова које заузима Показивач је
  - 2 бајта
  - 4 бајта

- Зависи од величине меморије (опсега адреса)

За 64*кВ* меморије
минимална меморијска локација 1 бајт,
свака локација може да се опише са 16 бита,
показивач заузима 2 бајта

Дефинисање показивача

Општа синтакса је:

```
tip_podatka *ime_pokazivaca;
```

- **tip_podatka** представља тип податка чију адресу показивач (треба да) садржи.
- ***** обавезан знак
- **ime_pokazivaca** идентификатор.

Пример:

```
int *p;
float *ptr;
```

Дефинисање показивача

```
int *p; int* p;  int * p;
```

- Знак * може да се постави било где између типа податка и имена показивача.

```
int* p, q;
```

- Само `p` је показивач, а `q` није.

- Пепорука је да се знак `*` пише непосредно поред имена променљиве.

```
int *p, *q;
```

Оператори за показиваче

- *C* програмски језик има два оператора за променљиве типа показивач:
  - оператор **&**
  - оператор *****

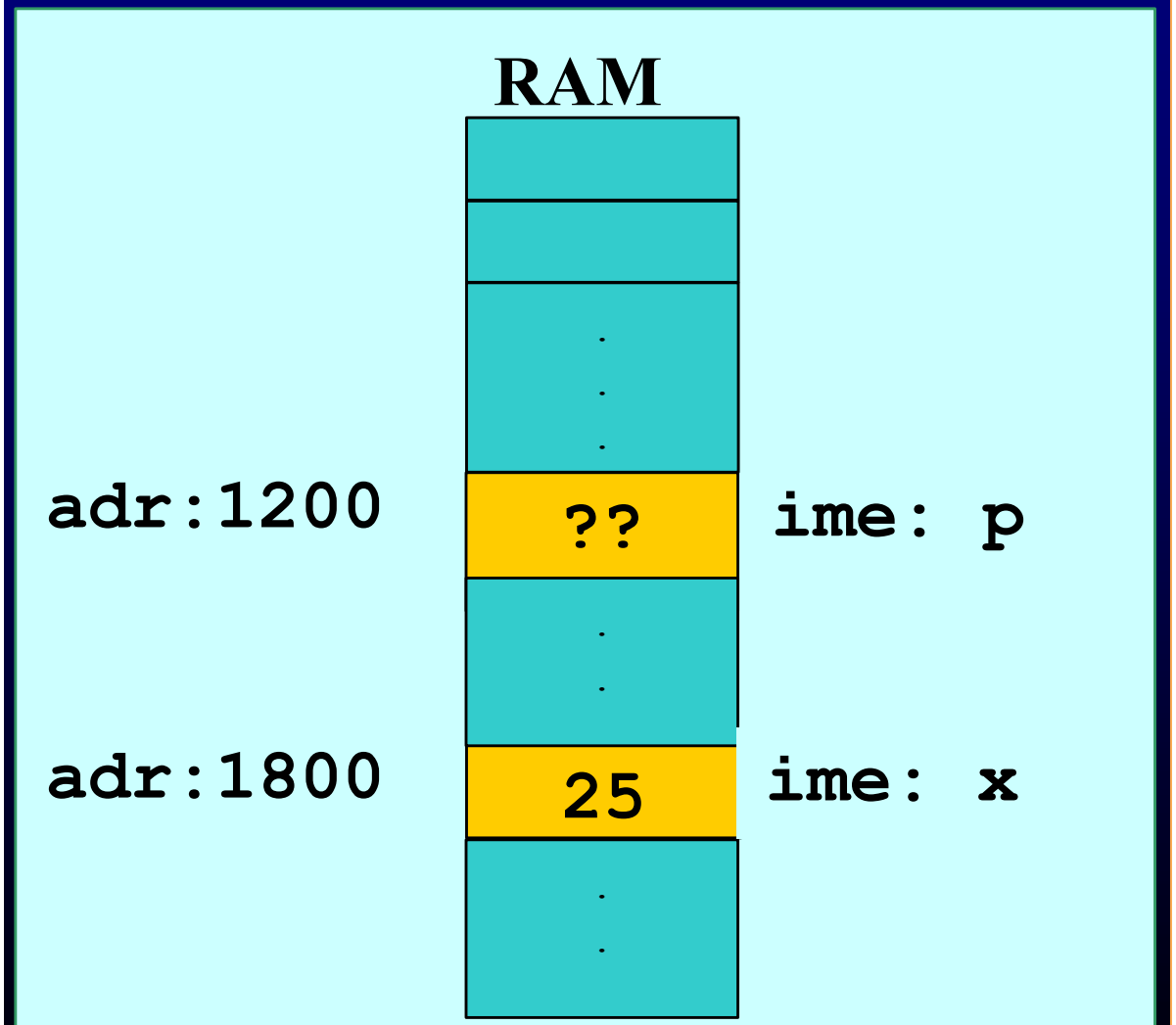
оператор &

- одређивање адресе променљиве
- додела адресе промељиве показивачу.

```
int main()
{
   int x = 25;
   int *p;

    p = &x;
   *p = 48;

    return 0;
}
```

оператор &

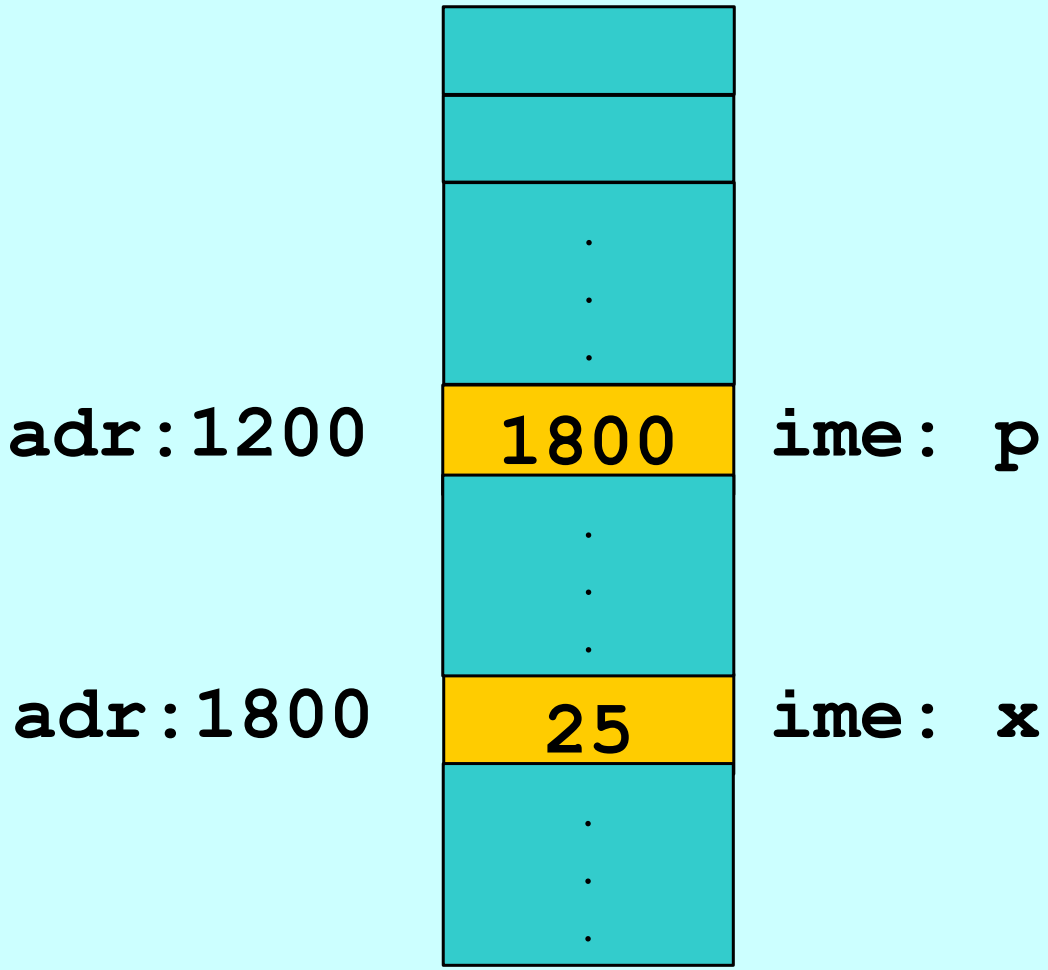
Вредности адреса у наредним примерима су изабране насумично!




```
int main()
{
   int x = 25;
   int *p;

   p = &x;

   *p = 48;


   return 0;
}
```

оператор &

p = &x чита се: у променљиву p упиши адресу (&) променљиве x



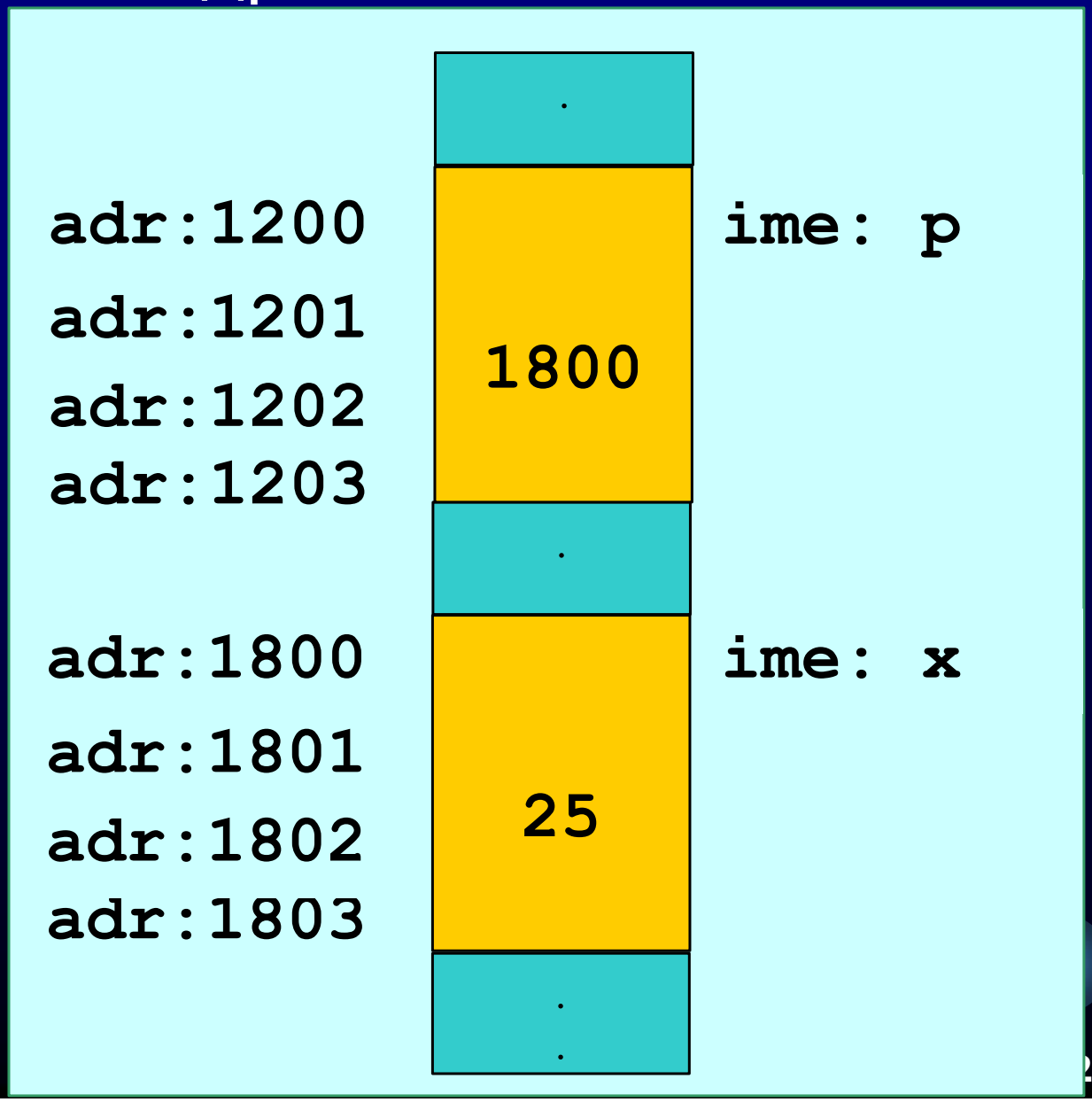

```
int main()
{
   int x = 25;
   int *p;

   p = &x;

   *p = 48;

   return 0;

}
```

оператор &

Чува се само почетна адреса!

Променљива **x** (int) заузима 4 бајта меморије

Показивач **p** садржи адресу првог бајта

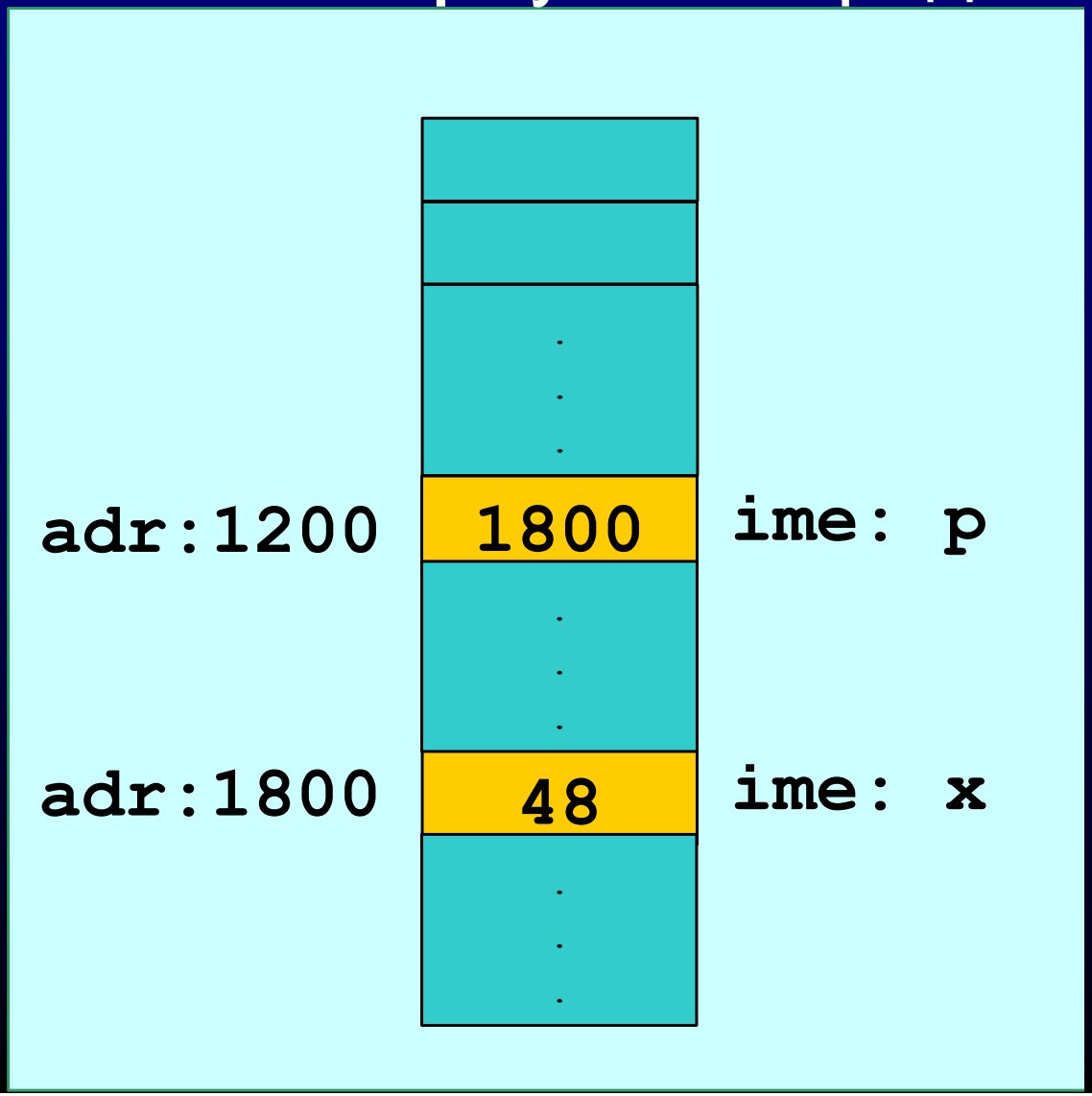


оператор *

- Податаку у меморији, може да се приступи применом оператора *****.
- Приступање је посредно, помоћу адресе, назива се и индиректно адресирање
- Оператор ***** се назива **оператор индиректног адресирања**.

оператор *

*p = 48 чита се: на адресу коју садржи показивач p упиши вредност 48




```
int main()
{
   int x = 25;
   int *p;

   p = &x;

   *p = 48;

   return 0;

}
```

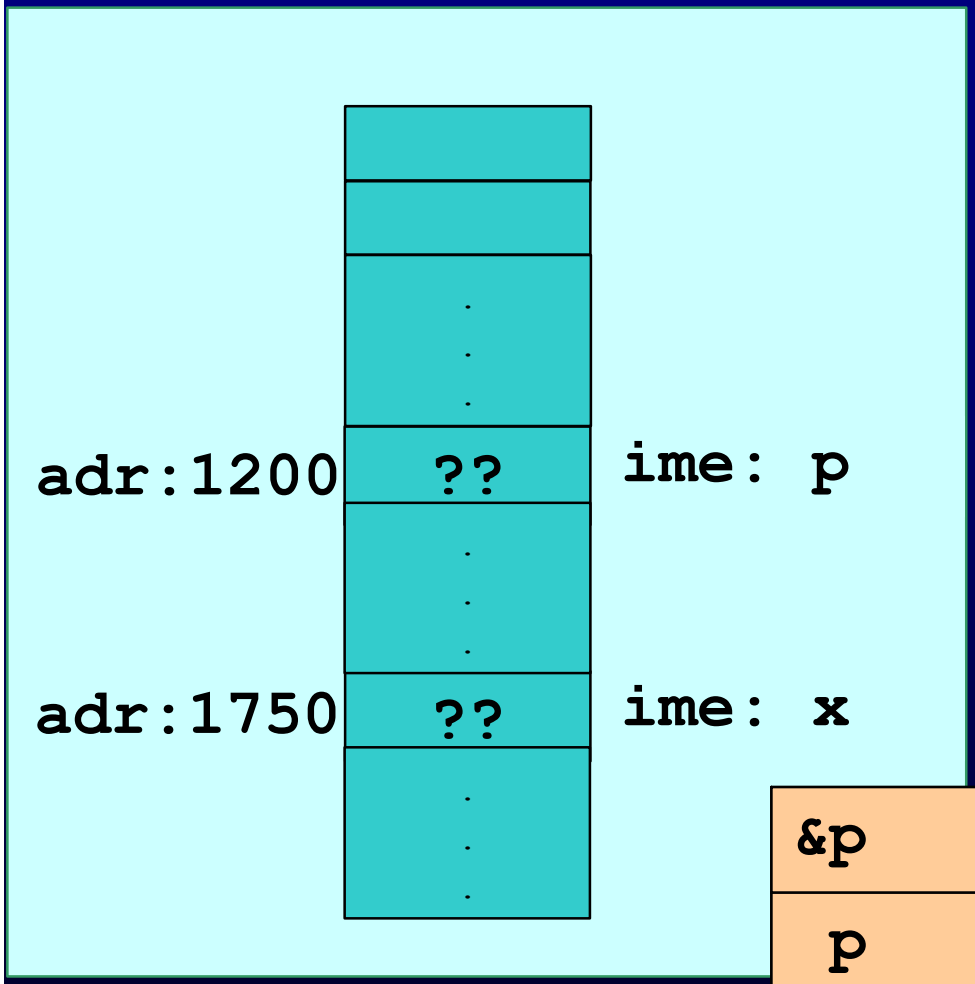
оператори

Ако је `p` променљива типа показивач

`p`, `&p`, и `*p` : имају различита значења.

- `p` Садржај променљиве `p` (адреса неке друге променљиве),
- `&p` Адреса променљиве `p`,
- `*p` Вредност меморијске локације чију адресу садржи показивач `p`

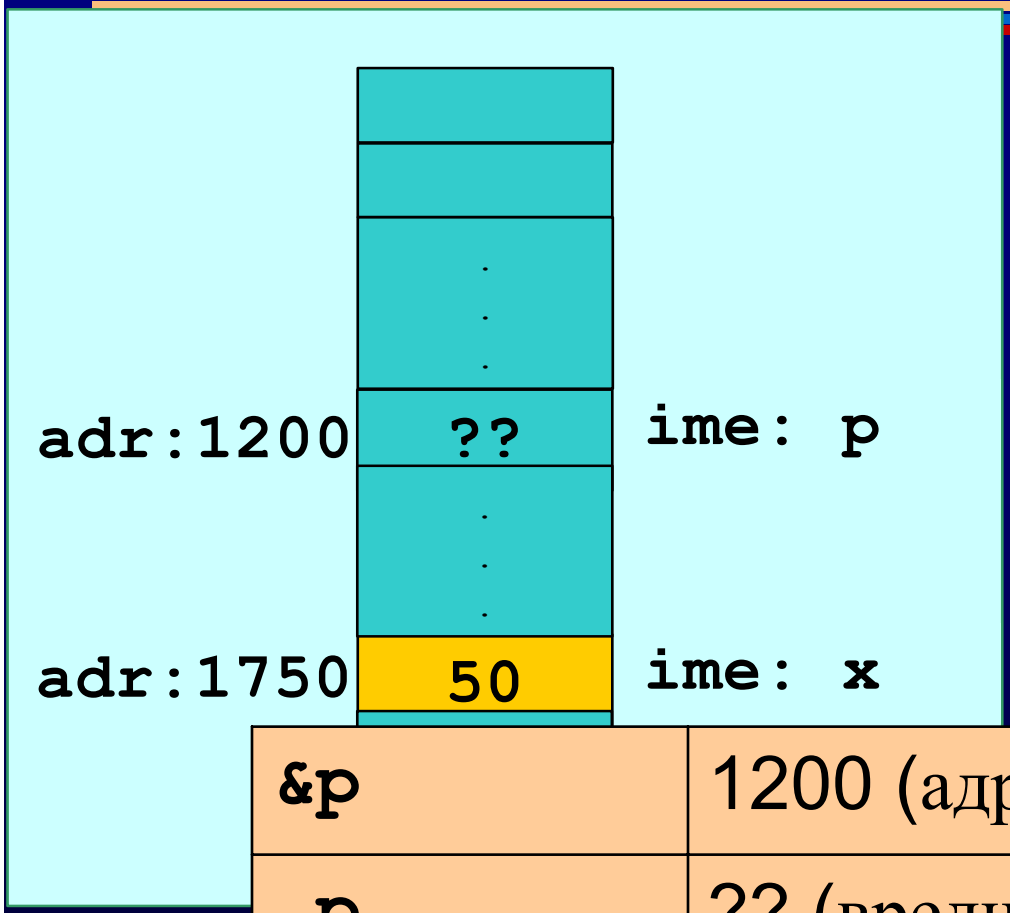
Пример 1




```
int main(){
   int x;
   int *p;
}
```

`&p`	1200 (адреса промењиве p)
`p`	?? (вредност промењиве p)
`*p`	не постоји
`&x`	1750 (адреса промењиве x)
`x`	?? (вредност промењиве x)

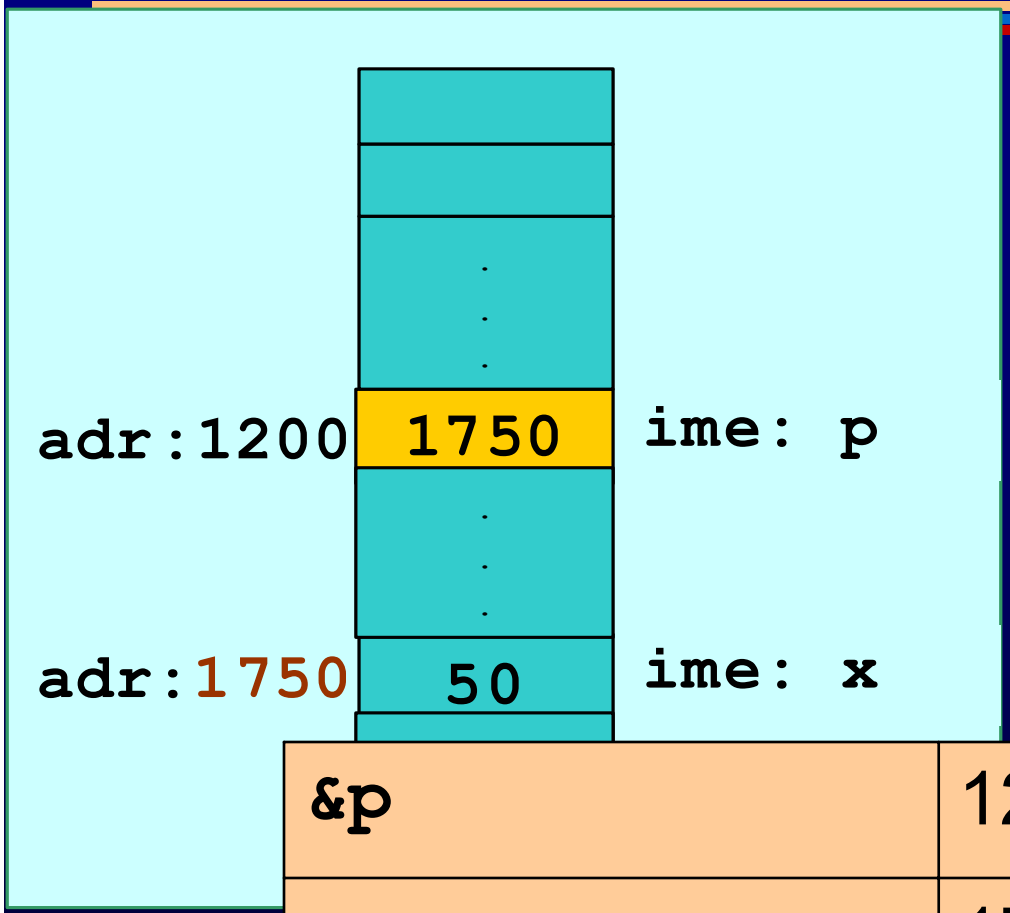
Пример 1 *наставак*




```
int main(){
   int x;
   int *p;
   x = 50;
}
```

&p	1200 (адреса промењиве p)
p	?? (вредност промењиве p)
***p**	не постоји
&x	1750 (адреса промењиве x)
x	50

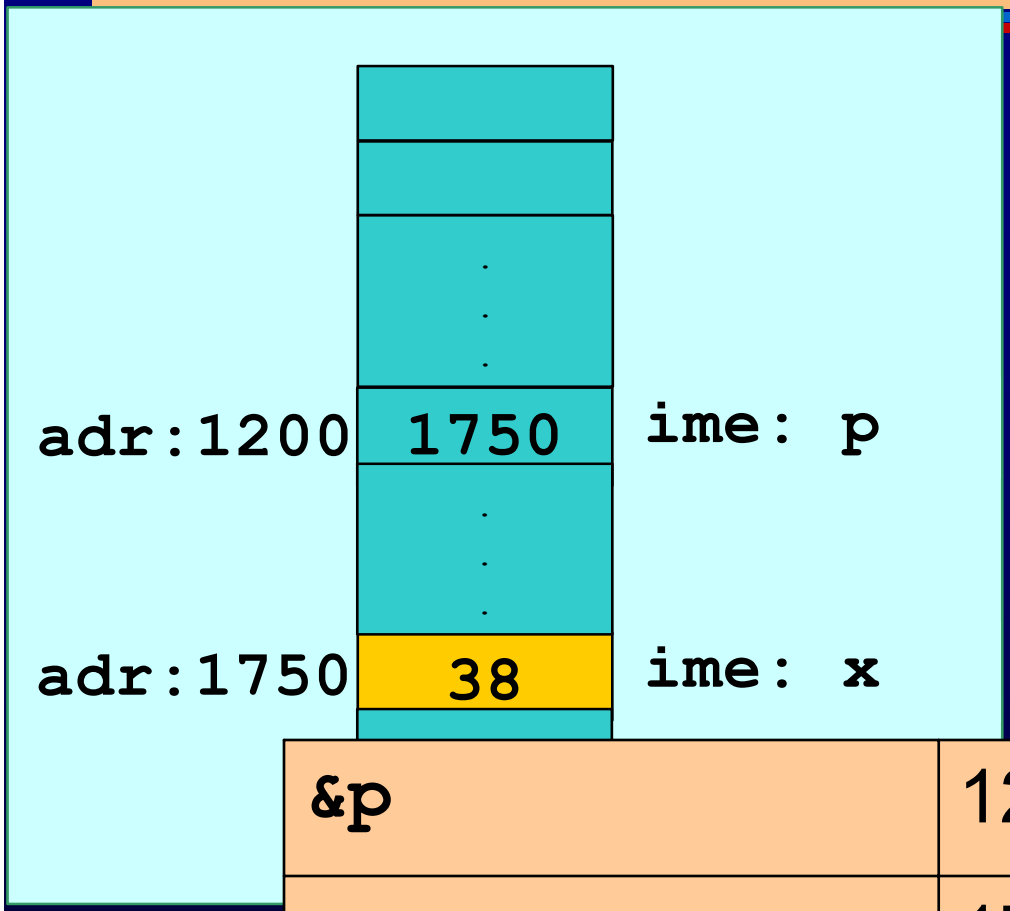
Пример 1 *наставак*




```
int main(){
   int x;
   int *p;
   x = 50;
   p = &x;
}
```

&p	1200 (адреса промењиве p)
p	1750
***p**	50
&x	1750 (адреса промењиве x)
x	50

Пример 1 *наставак*




```
int main(){
   int x;
   int *p;
   x = 50;
   p = &x;
   *p = 38;
}
```

&p	1200 (адреса промењиве p)
p	1750
***p**	38
&x	1750 (адреса промењиве x)
x	38

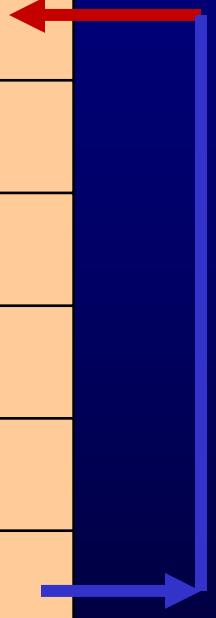
Пример 2

&ptr1	2350
ptr1	??
*ptr1	??
&ptr2	2600
ptr2	??
*ptr2	??
&x	1320
x	??
&y	1950
y	??

```
int main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

Пример 2 *наставак*

&ptr1	2350
ptr1	1320
*ptr1	??
&ptr2	2600
ptr2	??
*ptr2	??
&x	1320
x	??
&y	1950
y	??



```
int main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

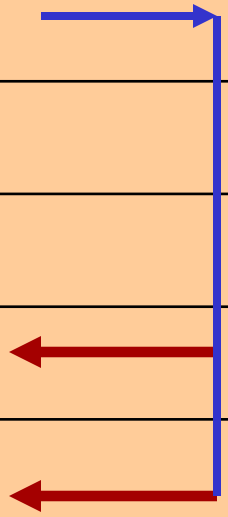
Пример 2 *наставак*

&ptr1	2350
ptr1	1320
*ptr1	42
&ptr2	2600
ptr2	??
*ptr2	??
&x	1320
x	42
&y	1950
y	??

```
int main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

Пример 2 *наставак*

&ptr1	2350
ptr1	1320
*ptr1	42
&ptr2	2600
ptr2	1320
*ptr2	42
&x	1320
x	42
&y	1950
y	??



```
int main()
{
 int *ptr1, *ptr2;
 int x,y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

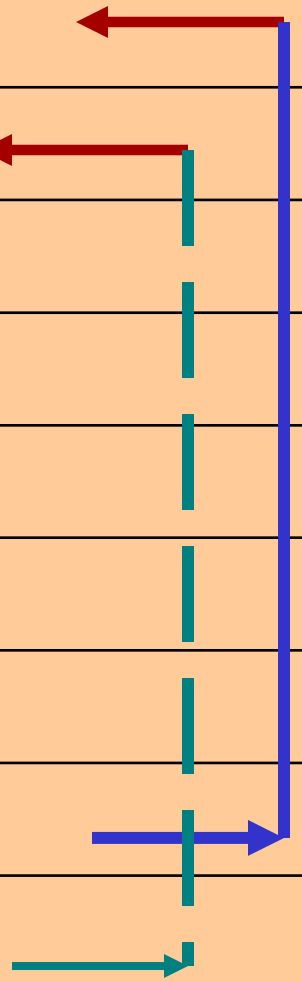
Пример 2 *наставак*

&ptr1	2350
ptr1	1320
*ptr1	53
&ptr2	2600
ptr2	1320
*ptr2	53
&x	1320
x	53
&y	1950
y	??

```
int main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

Пример 2 *наставак*

&ptr1	2350
ptr1	1950
*ptr1	??
&ptr2	2600
ptr2	1320
*ptr2	53
&x	1320
x	53
&y	1950
y	??



```
int main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
 return 0;
}
```

Пример 2 *наставак*

&ptr1	2350
ptr1	1950
*ptr1	88 ←
&ptr2	2600
ptr2	1320
*ptr2	53
&x	1320
x	53
&y	1950
y	88 ←

```
main()
{
 int *ptr1, *ptr2;
 int x, y;
 ptr1 = &x;
 *ptr1 = 42;
 ptr2 = ptr1;
 *ptr2 = 53;
 ptr1 = &y;
 *ptr1 = 88;
}
```

Додела почетне вредности показивачу

- Почетна вредност показивача треба да буде:
  - адреса неког претходно дефинисаног податка;
  - 0 (нула).
  - NULL

```
int main( )
{
   int x;
   int *pokazivac = &x;
   int *p2 = NULL
   /* moze i int *p2 = 0 */
   return 0;
 }
```

Генерички показивачи

- Показивачи код којих није одређен тип.
- Уместо типа податка (`int, float, double...`) ставља се резервисана реч **void**

```
viod *point;
```

- Помоћу **генеричких** показивача није могуће да се приступи подацима!

Генерички показивачи

```
int main()
{
 int x;
 void *p = &x; /* Greska!  */
 void *p2;

 p2 = &x; /* Greska! Tip pokazivaca..*/

 ...

}
```

Генерички показивачи

```
int main()
{
 int x;
 void *p2;
 ...
 /* p2 sadrzi adresu promenljive x */
 (int *)p2=&x;

 /* na adresu promenljive x upisati 3 */
 *((int *)p2) = 3; /* isto sto i x=3 */
}
```

Приказивање вредности показивача

- printf и формат %p

```
#include<stdio.h>
int main()
{
 int x;
 int *px;
 px = &x;
 printf("Vrednost pokazivaca: %p\n", px);
 return 0;
}
```

Приказивање вредности показивача

```
#include<stdio.h>
int main()
{
 int x = 5;
 int *px;

 px = &x;
 printf("Vrednost x: %d\n", x);
 printf("Adresa od x: %p\n", &x);
 printf("Vrednost pokazivaca: %p\n", px);
 printf("Sadrzaj mem. ciju adr. sadrzi \
                    pokazivac: %d\n", *px);
}
```

```
Vrednost x: 5
Adresa od x: 0012FF7C
Vrednost pokazivaca: 0012FF7C
Sadrzaj mem. ciju adr. sadrzi
pokazivac: 5
```

Рачунске операције над показивачима

- Операције над показивачима:
  - додела вредности једног показивача другом
  - додавање целобројног податка на вредност показивача и одузимање целобројног податка од вредности показивача
  - упређивање вредности два показивача
  - упоређивање вредности показивача са нулом

Додела вредности једног показивача другом

```
main()
{
 double x = 1.23;
 double *px = &x;  /* px sadrzi adresu
                      promenljive x */
 double *p1;
 int *p2;

 p1 = px;         /* p1 sadrzi adresu
                     promenljive x */
 p2 = px;         /* Greska! tip */
 p2=(int *)px; /* Ovako moze, ali na
                  odgovornost programera */
}
```

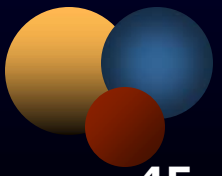
Додавање и одузимање

- Операторима:
  - **+**
  - **++**
  - **+=**

  рачуна се збир вредности показивача и целог броја,

- Операторима:
  - **–**
  - **– –**
  - **– =**

  рачуна се разлика вредности показивача и целог броја.



Додавање и одузимање

```
int main()
{
 int *p;
 double *q;
 char *chPtr;
 ...
 p++; /*  p=p+1 povecava p za 4 bajta  */
 q++; /*  q=q+1 povecava q za 8 bajtova */
 chPtr++;/* povecava chPtr za 1 bajt  */
 p=p+2; /* povecava vrednost p za 2x4 bajtova*/
 ...
}
```

Упоређивање вредности два показивача

- Оператори
  - = =
  - !=
- два показивача истог типа могу се упоређивати како би се добио одговор да ли показују на исти податак или не.

Упоређивање вредности показивача са нулом

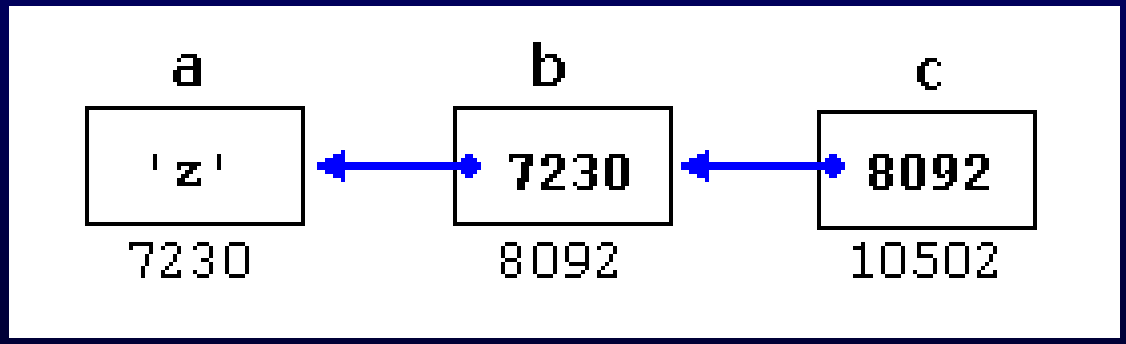
- Да ли показивач садржи валидну адресу?

- Уколико показивач има вредност нула (или **NULL**) онда он не показује ни на један податак.
- Примена симболичке константе: **NULL** захтева навођење стандардне библиотеке **<stdio.h>**.

Показивач на показивач

- може да садржи адресу другог показивача

```
tip **ime_pokazivaca
```



Пример 2

&b	8092
b	??
*b	??
&c	10502
c	??
*c	??
**c	??
&a	7320
a	z

```
int main()
{
  char a='z';
  char * b;
  char ** c;

  b = &a;
  c = &b;
  return 0;
}
```

Пример 2 *наставак*

&b	8092
b	7320
*b	z
&c	10502
c	??
*c	??
**c	??
&a	7320
a	z

```
int main()
{
   char a='z';
   char *b;
   char **c;

   b = &a;
   c = &b;
   return 0;
}
```

Пример 2 *наставак*

&b	8092
b	7320
*b	z
&c	10502
c	8092
*c	7320
**c	z
&a	7320
a	z

```
int main()
{
   char a='z';
   char *b;
   char **c;

   b = &a;
   c = &b;
   return 0;
}
```

Хвала на пажњи

Питања?