



Универзитет “Св. Кирил и Методиј” во Скопје  
Факултет за електротехника и информациски технологии



## ПРОГРАМИРАЊЕ И АЛГОРИТМИ

# Покажувачи

- Програмски јазик C -  
Учебна 2018/19 година



# Потсетување

## Меморија-организација

- Чува **инструкции и податоци** за програма (променливи...)

- ▶ Секоја локација има

**‘адреса’**

- ▶ Секоја локација чува информација како

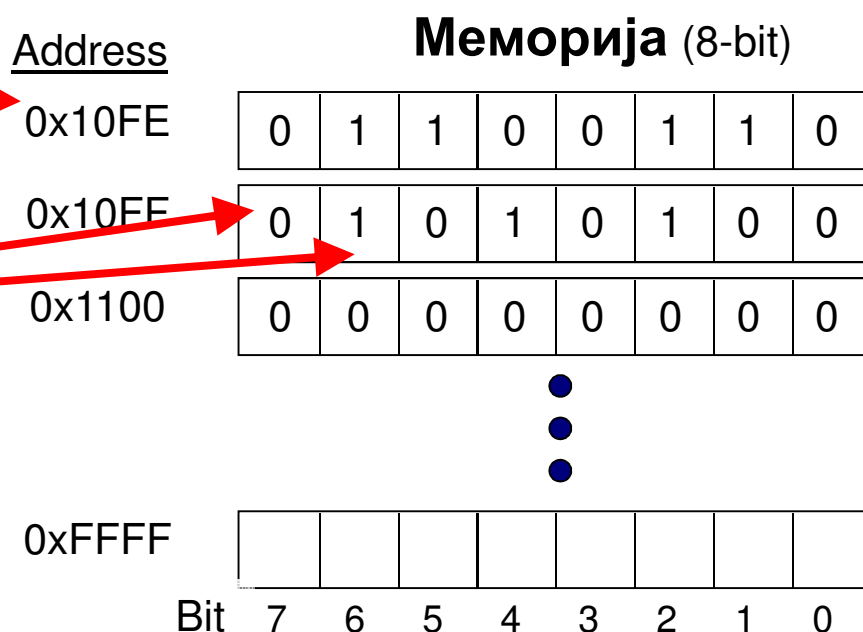
**‘битови’**

- ☐ 0 или 1

- ☐ 8 бита = 1 бајт

- ▶ Меморијата се

**‘запишува’** или **‘чита’**





# Потсетување

## Променливи и меморија

### ■ Променливи- декларација

1. Име на променлива

2. Тип на променливата

### ■ Акции:

- Соодветен број бајти се алоцираат во меморија
- **Симбол табела:** име, тип, адреса, вредност

```
int var1 = 0;
```

Address Меморија (8-bit)

0x10FE	0	0	0	0	0	0	0	
0x10FF	0	0	0	0	0	0	0	
0x1100	0	0	0	0	0	0	0	
0x1101	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0

Симбол табела

Var name	Var type	Var address	Var value
var1	int	10FE	0

★ Симболички - променлива  
има две “вредности”

- Вредност што е сместено на мемориската локација
- Вредност на мемориската локација (нејзина адреса)



# Потсетување

## Пристап и промена на променливи

- Доделување вредност ('**запишување**') (`i = 3;`)

```
short int i, j;
```

```
i = 3;
```

```
j = i;
```

- Вредност се копира/запишува на адреса наведена во символ табела

- Користење променлива ('**читање**') (`j = i;`)

- Пристап до **вредност** на она што е зачувано на мемориската локација

Address

0x10FE

0x10FF

Bit 7 6 5 4 3 2 1 0

Меморија (8-bit)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1



Big-endian

Символ табела

Name	Type	Address	Value
<b>i</b>	short int	10FE	
<b>j</b>	short int	1100	



# Променливи што чуваат адреси

- Може да се дефинираат променливи што како вредност чуваат/сместуваат мемориска адреса
- Такви променливи се викаат “покажувачи” (“pointer”)



# Показувачи

- Показувач содржи адреса на променлива (мемориска локација)  
или
- Показувачот содржи целобројна вредност, што се интерпретира како адреса на друга променлива
- Променливите содржат вредности за податокот (директно референцирање)
- Показувачите содржат адреси на променливи (индиректно референцирање)

Се сеќавате ли на  
`scanf ("%d", &i);`



# Зошто да се користат покажувачи?

- Една функција може да ги менува променливите директно во повикувачката функција (без глобални променливи)
- Да се врати повеќе од една вредност при повик на функција
- Да се прати покажувач кон голема податочна структура, наместо да се копира целата структура
  - На пр. Полиња



# Адреса и Показувач

- Променлива показувач е променлива што содржи адреса
- Пристапување до адресата на секоја променлива се прави со користење на “&” (**адресен оператор**):

“&num1” : адресата  
за променливата num1

□ `int num1 = 5;`

□ `printf("address of num1: %p", &num1);`

address of num1: 0x7ffee0a99328

Формат за печатење на  
показувач/адреса  
(Хексадецимален)

Адресата се менува (зависи од  
компјутерот, од извршување,...)

- Декларација на показувач:

□ `type * varname`

- `int* ptr1;`
- `float *ptr2;`
- `char * ptr3;`

□ Се чита:

- “ptr1 е пок. кон цел број”
- “ptr2 е пок. кон float”
- “ptr3 е пок. кон char”



# Што е покажувач?

```
#include <stdio.h>
int main()
{
    int num1;
    int *ptr1 = &num1, *ptr2;
    num1 = 7;
    printf("size of num1: %d\n", sizeof(num1));
    printf("value of num1: %d\n", num1);
    printf("address of num1: %p\n", &num1);
    printf("address in ptr1: %p\n", ptr1);
    return 0;
}
```

&num1 значи: адресата  
за променливата num1

- Што е големина на num1?
- Дали &num1 == ptr1?
- Која адреса е сместена во ptr2?

Секогаш иницијализирајте ги  
декларираните покажувачи!

## Output:

size of num1: 4  
value of num1: 7  
address of num1: 0x7ffee0a99328  
address in ptr1: 0x7ffee0a99328



# Декларација на покажувачи

## ■ Формат:

```
tip *pokIme;
```

## ■ Пример:

```
int *pok; (se deklarira pokazuvac kon celobrojna promenliva)
```

```
double *myPtr (pokazuvac kon realna promenliva)
```

- При декларацијата за секој покажувач мора да се декларира неговиот податочен тип

- ☐ покажувач кон цел број, покажувач кон реален број, итн.
- ☐ може да се декларираат покажувачи од кој и да е податочен тип



# Иницијализација на покажувачи



- Иницијализација се врши со поставување на вредноста на покажувачот: 0, NULL, или на мемориска адреса

- 0 или NULL – невалидна вредност

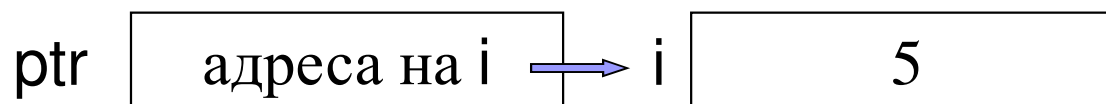
`int i = 5;` – декларирање и иницијализација на целобројна променлива

`int *ptr;` – декларирање на променлива покажувач

`ptr = &i;` – иницијализација на покажувачот `ptr`

□ `ptr` е променлива во која се сместува адреса

□ во `ptr` не се сместува вредноста на променливата `i`



Што ќе биде отпечатено?

```
printf("i = %d\n", i);  
printf("*ptr = %d\n", *ptr);  
printf("ptr = %p\n", ptr);
```

Излез:  
`i = 5`  
`*ptr = 5`  
`ptr = effff5e0`

вредност на `ptr` =  
адреса на `i` во  
меморијата



# Пристап до она каде покажува покажувачот



## ■ Оператор за индирекција “\*”

- Ја дава вредноста сместена на адресата зачувана во покажувачот

```
#include <stdio.h>
int main()
{
    int num1;
    int *ptr1 = &num1;
    num1 = 7;
    printf("value of num1: %d\n", num1);
    printf("value of num1: %d\n", *ptr1);

    return 0;
}
```

Излез:  
value of num1: 7  
value of num1: 7

Вредноста на тоа каде  
што покажува покажувачот



# Операции со покажувачи

- “&” (адресен оператор) – ја враќа адресата на операндот

```
int y = 5;
```

```
int *yPtr;
```

```
yPtr = &y;
```

во `yPtr` се сместува адресата на `y`  
`yPtr` “покажува кон” `y`

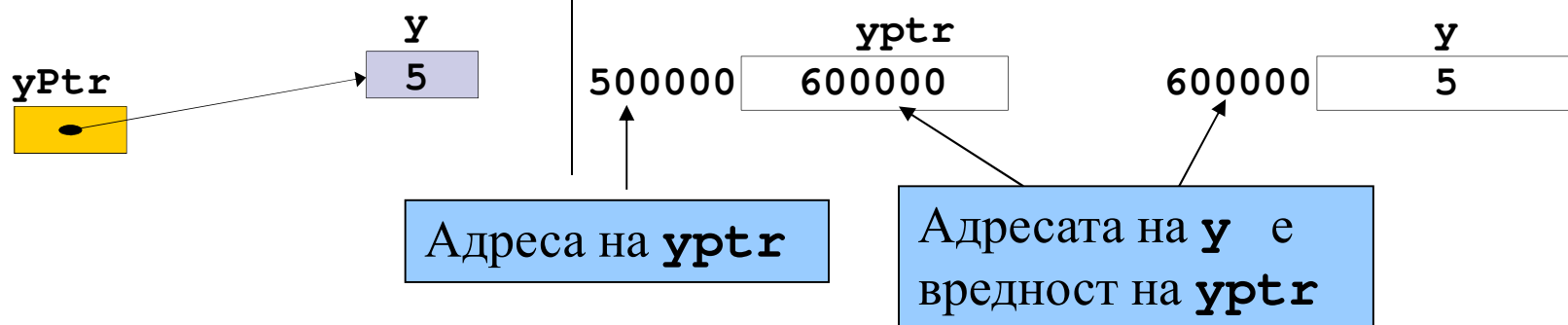
- “\*” (индирекција) – ја враќа содржината на променливата кон која покажува покажувачот

`*yPtr = 7;` — ја менува содржината на `y` во 7

- “\*” и “&” се инверзни и се поништуваат

`*&yPtr`  $\Leftrightarrow$  `*(&yPtr)`  $\Leftrightarrow$  `*(address of yPtr)`  $\Leftrightarrow$  `yPtr`

`&*yPtr`  $\Leftrightarrow$  `&(*yPtr)`  $\Leftrightarrow$  `&(y)`  $\Leftrightarrow$  `yPtr`

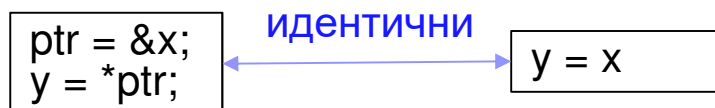




# Користење покажувачи

## Пример 1

`int *ptr;` - ptr е покажувач кон целобројна променлива  
`int x, y;` - декларирани се две целобројни променливи  
`ptr = &x;` - ptr покажува на x. &x означува адреса на x  
`y = *ptr;` - \*ptr означува вредност во променливата кон која покажува ptr

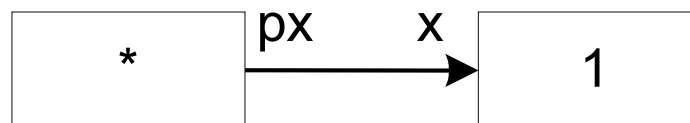


## Пример 2

`int x = 1, *px;`



`px = &x;`



`*px = 2;`



`x = *px + 1;`





# Пример: Кориштење показувачи

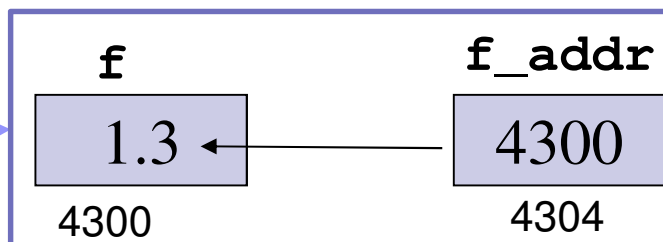
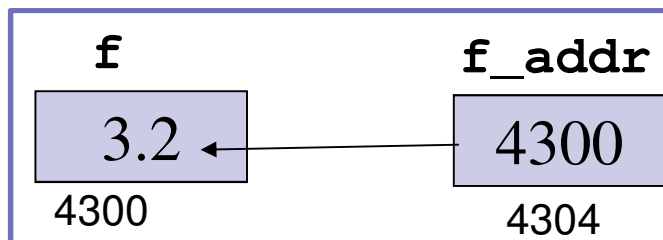
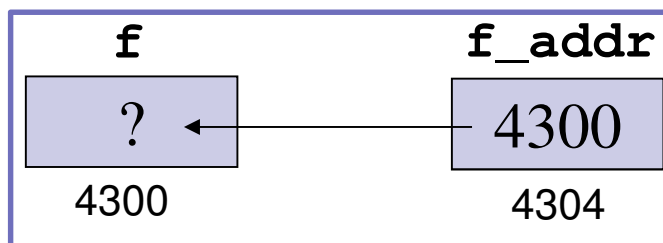
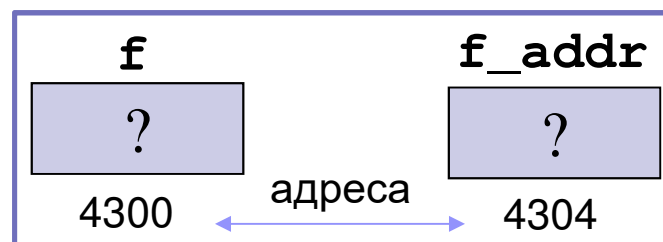
```
float f; /*realna promenлива*/  
float *f_addr; /* pokazuvac */
```

```
f_addr = &f;  
/* & - operator adresa na */
```

```
*f_addr = 3.2;  
/* * - operator za indirekcija */
```

```
float g = *f_addr;  
/* indirekcija: g stanuva 3.2*/
```

```
f = 1.3;  
/* g se uste e 3.2 */
```





# Изрази со покажувачи

- Следните аритметички операции може да се извршуваат со покажувачи
  - инкрементирање/декрементирање на покажувачи (++ или --)
    - вредноста на покажувачот се зголемува односно намалува за **големината на мемориската локација на која покажува** покажувачот
  - додавање на целобројна вредност на покажувач  
( + или += , - или -= )
  - **Одземање на покажувачи** – го враќа бројот на елементи помеѓу двете адреси



# Изрази со покажувачи

- **p, q** и **n** се покажувачи

- **p = p+1;** или **p++;**

двата израза извршуваат иста операција, и овозможуваат **p** да покажува кон следниот мемориски елемент што следи по елементот на кој почетно покажувал покажувачот **p**.

- **q = p+i;**

**q** покажува кон податочниот елемент што се наоѓа **i** позиции по елементот на кој покажува **p**.

- **n = q - p;**

**n** е број на елементи меѓу **p** и **q**, и претставува целобројна вредност.



# Аритметички операции со показувачи

- Показувачи од ист тип може да се употребат во наредби за доделување на вредност
  - ако не се од ист тип потребно е користење на **cast** оператор
  - исклучок: показувач од типот **void: void \***
    - генерички показувач, покажува кон кој и да е тип
    - не е неопходен **cast** оператор за да се конвертира вредноста на показувачот во **void** показувач
- Споредба на показувачи (  $<$ ,  $==$ ,  $>$  )
  - $q == p$  - дали  $q$  и  $p$  покажуваат кон иста мемориска адреса,
  - $q < p$  - дали елементот кон кој покажува  $q$  претходи на елементот на кој покажува  $p$ .
- КУСА ПРОВЕРКА
  - Што значи  $ptr + 1$ ? ✓
  - Што означува  $ptr - 1$ ? ✓
  - Што означуваат  $ptr * 2$  и  $ptr / 2$ ? ✗



# Пренесување на променливи во функции

## ■ Пренесување на **променливи** се прави со покажувачи

- ☐ се пренесува адресата на аргументот со операторот **&**
- ☐ овозможува да се смени содржината на адресата
- ☐ **полињата во функциите се пренесуваат како покажувачи**

## ■ **\*** оператор

- ☐ се користи како алијас/прекар за променливата во функцијата

```
void double(int *number) {  
    *number = 2 * (*number);  
}
```



## Пример: Пренесување вредност

```
#include <stdio.h>
void swap(int, int);

int main() {
    int x,y;
    x = 5;
    y = 6;
    swap(x,y);
    printf("%d %d\n",x,y);
}
```

```
void swap(int a,int b){
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Излез:  
5 6



## Пример: Пренесување променлива

```
#include <stdio.h>
void swap (int*, int*);

int main() {
    int x,y;
    x = 5;
    y = 6;
    swap(&x,&y);
    printf("%d %d\n",x,y);
}
```

```
void swap(int *a,int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Излез:  
6 5



# Полиња и покажувачи

- Тесно се поврзани, името на полето е покажувач кон првиот елемент на истото, односно ако важи `int a[10]`, тогаш '`a`' значи `&a[0]`
- `a[i]` и `*(a+i)` се еквивалентни и овозможуваат пристап до елементот од полето на позиција `i`
- Пример: нека се декларирани вектор `a[5]` и покажувач `aPtr`. Следните наредби се точни:

```
aPtr = a;
```

```
aPtr == &a[0];
```

```
a[n] == *( aPtr + n )
```

```
a[3] == *(a + 3)
```

```
a+i == &a[i]
```

```
*(a+i) == a[i] == i[a]
```

Пристапот до елемент на поле преку неговото име и индекс `a[i]` преведувачот секогаш интерно го интерпретира како `*(a+i)`, така што на пример наместо `a[5]` сосема правилно ќе работи и `5[a]` !!!

- Индексирање може да се употреби и на покажувачи за да се пристапи до елемент на вектор, така следните два изрази се идентични:  
`aPtr[3]` и `a[3]`



# Полиња и покажувачи (1)

- **Пример:** Вектор со 5 целобројни променливи.

```
int v[5];
```

```
int *vPtr, *vPtr2;
```

```
vPtr2 = &v[2];
```

```
vPtr = &v[0];
```

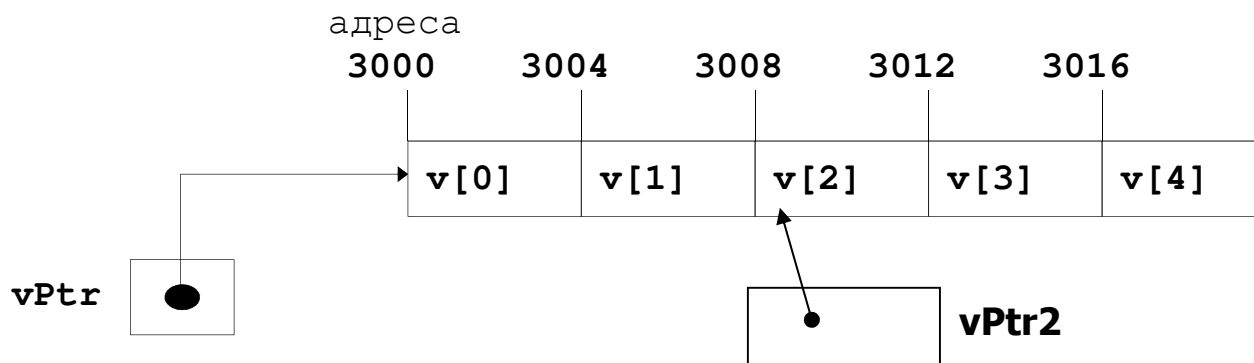
→ **vPtr** покажува кон првиот елемент **v[0]** на локација 3000. (**vPtr** = 3000)

```
vPtr2 - vPtr == 2;
```

→ Точно

```
vPtr += 2;
```

→ го поставува **vPtr** на 3008, **vPtr** покажува на **v[2]** (зголемен е за две мемориски локации)





## Полиња и покажувачи (2)



- **Пример:** Нека важат следните декларации

```
char a[50], x, y, *pa, *pa1, *pai;  
pa = &a[0]; - адресата на a[0] смести ја во pa  
pa = a; - исто како и претходното  
x = *pa; - вредноста на a[0] смести ја во x  
pa1 = pa+1; - определи ја адресата на a[1]  
pai = pa+i; - определи ја адресата на a[i]  
y = *(pa+i); - вредноста на a[i] во y
```



# Илустрација на користење на покажувачи



```
float a[4];  
float *ptr;
```

```
ptr = &(a[2]);  
*ptr = 3.14;  
ptr++;  
*ptr = 9.0;  
ptr = ptr - 3;  
*ptr = 6.0;  
ptr += 2;  
*ptr = 7.0;
```

Напомени:

```
a[2] == *(a + 2)  
ptr == &(a[2])  
    == &*(a + 2)  
    == a + 2
```

Податочна табела			
Име	Тип	Опис	Вредност
 a[0]	float	елемент на вектор (променлива)	<del>6.0</del>
a[1]	float	елемент на вектор (променлива)	?
 a[2]	float	елемент на вектор (променлива)	<del>3.14</del>
 a[3]	float	елемент на вектор (променлива)	<del>9.0</del>
ptr	float *	покажувач кон реална променлива	?
*ptr	float	променлива кон која покажува покажувачот	?



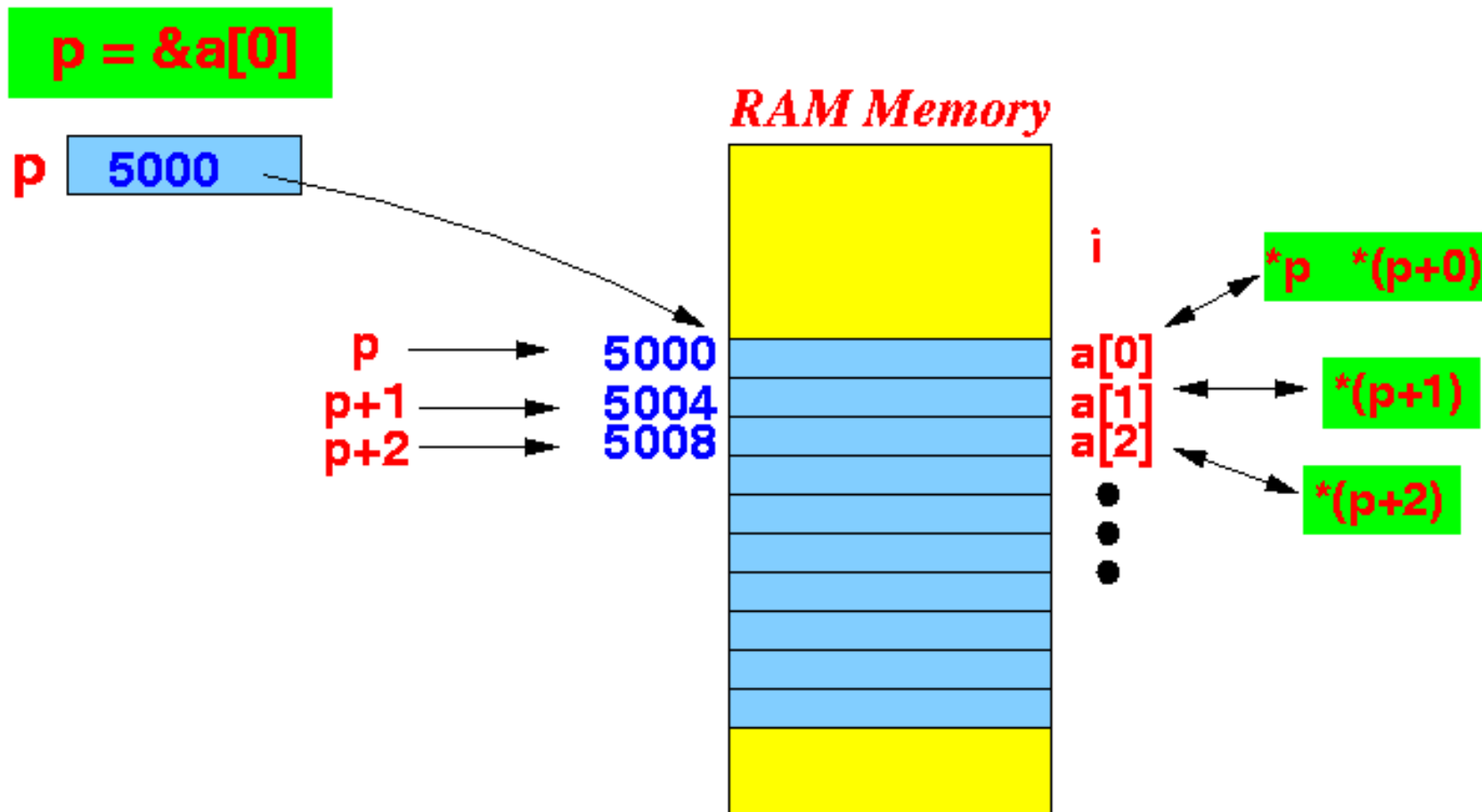
# Операции со покажувачи и полиња

- Нека **a** е вектор од **double** променливи
  - ако важи **double \*ptr = a;**
  - што означува **ptr + 1**? – поместување на покажувачот на следната мемориска локација во полето
- Аритметичките оператори **+** и **-** се однесуваат на една мемориска локација потребна да се смести променлива од даден тип
- Ако важат претходните дефиниции и имајќи ја предвид следната табела:
  - **1000 + sizeof(double) = 1000 + 4 = 1004**

	Addr	Content	Addr	Content	Addr	Content	Addr	Content
a[0]: 37.9	1000	...	1001	...	1002	...	1003	...
a[1]: 1.23	1004	...	1005	...	1006	...	1007	...
a[2]: 3.14	1008	...	1009	...	1010	...	1011	...



# Операции со покажувачи и полиња





# Пренесување на полиња во функции - дефинирање

- Декларирање на формален параметар поле во заглавје на функција

`tipFunkcija Imefunkcija (tipElement ImePole[ ], int n){...}`

вообичаено бројот на елементи во полето исто така се пренесува како аргумент. Пример:

```
void funkcija(int tpole[], int n) {  
    tpole[n-1] = 0;  
}
```

број на елементи во полето

- Функциски прототипови

```
void modifyArray( int b[], int arraySize );
```

бидејќи името на формалните параметри не е важно:

- `int b[]` може да биде заменето со `int []` и
- `int arraySize` може да биде заменето со `int`,

со што претходната наредба гласи:

```
void modifyArray ( int [], int ) ;
```



# Пренесување на полиња во функции



- **Повикување на функција со аргумент поле:** името на полето без заградите

```
int myArray[24];
```

```
myFunction(myArray, 24);
```

- Преносот на полето се реализира со пренесување на адресата на првиот елемент (&myarray[0]), поради што **сите промени на вредностите на полето во функцијата, се видливи и по завршувањето на функцијата**

- **Пренесување на елементи на полето**

- ☐ може да се пренесат како вредности
- ☐ во листата на аргументи се наведува името на полето и индексот на елементот ( **myArray[3]** )

```
myFunction(myArray[3], 24);
```

- ☐ ако елементот на полето е пренесен како вредност сите промени на вредноста на формалниот аргумент во функцијата **нема да се рефлектираат на аргументот со кој е повикана функцијата**



# Пример за пренесување на вектори



Следните функции извршуваат иста работа:

```
int clear1(int array[], int size){
    int i;
    for(i = 0; i < size; i++) array[i] = 0;
    return 0;
}
```

```
int clear2(int *array, int size){
    int *p;
    for(p=&array[0]; p<&array[size];p++) *p=0;
    return 0;
}
```

```
int clear3(int *array, int size){
    int i;
    for(i = 0; i < size; i++) array[i] = 0;
    return 0;
}
```

```
int clear4(int array[], int size) {
    int *p;
    for(p=&array[0]; p<&array[size];p++) *p=0;
    return 0;
}
```

При преносот како аргумент на ф-ја,  
`int a[]` и `int *a` се однесуваат исто.

Во главната програма (каде се  
декларирани) разликата е:

```
int a[10], *pa=a;
```

☺ `pa[5]=...` е исто со `a[5]=...`

☺ `pa=...` или `pa++` е ОК,

Но `a` е **константа** и може само да се чита,

☹ `a = ...` или `a++` **не може!**

Ако направиме `pa++`, тогаш `pa` покажува  
на `a[1]`

`pa[0]` е исто со `a[1]`,

`pa[-1]` е исто со `a[0]` !!!



# Пример: Пренесување на вектори со покажувачи

```
#define N 5
int find_largest(int *a, int n){
    int i, max;
    max = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}
int main(){
    int b[N] = {8,9,1,4,7}, largest;
    largest = find_largest(&b[2], 3);
}
```

7

```
#define N 5
int find_largest(int *p, int n){
    int i, max;
    max = *p; // i.e. max = p[0]
    for (i = 1; i < n; i++)
        if (*(p+i) > max)
            max = *(p+i);
    return max;
}
int main(){
    int b[N] = {8,9,1,4,7}, largest;
    largest = find_largest(b, N);
}
```

9

```
#define N 5
int find_largest(int *p, int n){
    int i, max;
    max = *p;
    for (i = 1; i < n; i++){
        if (*p > max)
            max = *p;
        p++;
    }
    return max;
}
int main(){
    int b[N] = {8,9,1,4,7}, largest;
    largest = find_largest(b, N);
}
```

9

Сите три програми решаваат ист проблем. Но излезот на една од нив е различен. Зошто? Функцијата за наоѓање максимум во една од програмите нема да работи коректно во сите случаи. Која и зошто?



# Квиз прашања:

- Ако важи следната декларација:
  - `int a[100], *pa, *qa;`
- Кои од следниве наредби се валидни, нема да вратат грешка при компајлирање?

`pa = a;` ✓

`a = pa;` ✗

`pa++;` ✓

`a++;` ✗

`qa = pa;` ✓

`*qa = *pa;` ✓

`pa = &a;` ✗



# Квиз прашања:

Нека  $x=3$ ,  $y=4$ , колкава ќе биде вредноста на  $x$ =?  $y$ =?  
по следниот функциски повик `swap_1(x,y)`;

```
void swap_1(int a, int b) {  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}
```

$x=3$   
 $y=4$

Нека  $x=3$ ,  $y=4$ , Колкава ќе биде вредноста на  $x$ =?  $y$ =?  
по следниот функциски повик `swap_2(&x,&y)`;

```
void swap_2(int *a, int *b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

$x=4$   
 $y=3$



# Крај



# ASCII Code

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>



# Extended ASCII Code

128	Ç	144	É	160	á	176	░	192	Ł	208	⌌	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	ł	209	⌍	225	β	241	±
130	é	146	Æ	162	ó	178	▓	194	Ł	210	⌎	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	ł	211	⌏	227	π	243	≤
132	ä	148	ö	164	ñ	180	┆	196	—	212	⌐	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	┆	197	+	213	⌑	229	σ	245	∫
134	â	150	û	166	▪	182	┆	198	┆	214	⌒	230	μ	246	÷
135	ç	151	ù	167	°	183	π	199	┆	215	┆	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	┆	200	⌔	216	┆	232	Φ	248	°
137	ë	153	Ö	169	┆	185	┆	201	┆	217	┆	233	Ⓢ	249	.
138	è	154	Ü	170	┆	186	┆	202	⌕	218	┆	234	Ω	250	.
139	í	155	◊	171	½	187	┆	203	┆	219	■	235	δ	251	√
140	î	156	£	172	¼	188	┆	204	┆	220	■	236	∞	252	π
141	ï	157	¥	173	¡	189	┆	205	=	221	■	237	φ	253	²
142	Ä	158	£	174	«	190	┆	206	┆	222	■	238	ε	254	■
143	Å	159	ƒ	175	»	191	┆	207	┆	223	■	239	∩	255	

Source : [www.LookupTables.com](http://www.LookupTables.com)