

# מצביעים



קרן כליף

# ביחידה זו נלמד:

---

- מהו מצביע (פוינטר)
- מוטיבציה למצביעים
- אופרטור &
- אופרטור \*
- אתחול מצביע
- העברת פרמטר לפונקציה by pointer
- מצביע const

- החזרת יותר מערך אחד מפונקציה
- שפונקציה תוכל לשנות את הפרמטרים שהיא קיבלה (למשל כמו strcpy ו-strcat)
- העברת מערכים לפונקציות
- הקצאת מערכים בגודל לא ידוע בזמן קומפילציה –  
הקצאה דינאמית (לא נראה ביחידה זו)

# מהו מצביע (פוינטר)

□ עד כה ראינו טיפוסים שונים:

■ דוגמאות:

□ int – מכיל מספר שלם

□ double – מכיל מספר עשרוני

□ char – מכיל תו

□ מצביע הוא טיפוס המכיל כתובת של משתנה אחר

□ עבור כל טיפוס שלמדנו עד כה יש מצביע מהטיפוס המתאים (למשל מצביע לתא המכיל int, מצביע לתא המכיל double וכו')

□ גודלו של משתנה מטיפוס מצביע הוא 4 בתים

# הגדרת מצביע

□ כדי להגדיר מצביע:

```
<type>* <var_name>;
```

■ למשל:

□ הגדרת משתנה המצביע למשתנה אחר מטיפוס `int`:

```
int* ptr;
```

□ הגדרת משתנה המצביע למשתנה אחר מטיפוס `char`:

```
char* ptr;
```

□ כל משתנה נמצא בזיכרון בכתובת כלשהי

□ כדי לקבל את הכתובת של משתנה כלשהו  
נשתמש באופרטור &

□ את הכתובת שנקבל נוכל לשים במשתנה מטיפוס  
מצביע מהטיפוס המתאים

# אופרטור & - דוגמא

```
void main()
{
    int x = 3;
    int* pX = &x;
    printf("x = %d\n", x);
    printf("address of x = %p\n", &x);
    printf("pX = %p\n", pX);
    printf("address of pX = %p\n", &pX);
}
```

כדי להדפיס כתובת  
משתמשים ב- %p

int: x	3	1000
int*: pX	1000	1004
	0	1004
		1008
		1008

יודפס: x=3

יודפס: address of x = 1000

יודפס: pX = 1000

יודפס: address of pX = 1004

```
C:\WINDOWS\system32\cmd.exe
x = 3
address of x = 0012FF60
pX = 0012FF60
address of pX = 0012FF54
Press any key to continue .
```

אנו רואים שהכתובות במחשב הן לא  
מספרים דצימאליים, אלא הקסה-  
דצימאליים!

בדוגמאות שלנו נמשיך להשתמש  
בכתובות בבסיס דצימאלי

# נשים לב..

---

- הדרך היחידה לתת ערך למשתנה מטיפוס מצביע היא ע"י מתן כתובת של משתנה אחר ע"י האופרטור &, או השמה ממשתנה המכיל מצביע מאותו טיפוס
- לא ניתן לבצע השמה עם מספר
- לא נבצע השמה של כתובת למשתנה מצביע שאינו מאותו טיפוס (למשל מכתובת של double ל- \*int)



# דוגמאות

```
void main()
```

```
{
```

```
    int x = 3;
```

```
    double* pDouble;
```

```
    int* pInt1, *pInt2;
```

```
    pInt1 = &x;
```

```
    pInt1 = 1000;
```

// cannot convert from int to int\*

```
    pDouble = pInt1;
```

// cannot convert from int\* to double\*

```
    pInt2 = pInt1;
```

```
}
```

כאשר מגדירים כמה מצביעים בשורה אחת, צריך להגדיר \* לפני כל אחד

int: x	<b>3</b>	<b>1000</b>
double*: pDouble	<b>???</b>	<b>1004</b>
int*: pInt1	<b>1000</b>	<b>1008</b>
int*: pInt2	<b>1000</b>	<b>1012</b>

כל משתנה מטיפוס מצביע תופס 4 בתים בזיכרון, בלי קשר לגודל הטיפוס אליו הוא מצביע

# דוגמא להדפסת משתנים בפורמט שונה

```
void main()
```

```
{
```

```
    int x = 97;
```

```
    printf("as int:           %d\n",    x);
```

```
    printf("as char:          %c\n",    x);
```

```
    printf("as address(hexa): %p\n\n", x);
```

```
    printf("&x as address: %p\n", &x);
```

```
    printf("&x as int:         %d\n", &x);
```

```
}
```

$$97_{10} == 61_{16}$$

$$12FF60_{16} == 1245024_{10}$$

☐ as int: 97

☐ as char: a

☐ as hexa: 00000061

☐ &x as address: 0012FF60

☐ &x as int: 1245024

```
as int:           97
as char:          a
as address(hexa): 00000061

&x as address: 0012FF60
&x as int:       1245024
```

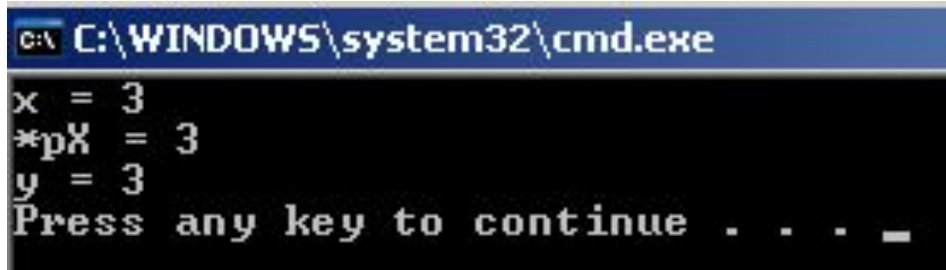
# \* אופרטור

□ כדי לפנות לתוכן שבכתובת אליה אנו מצביעים נשתמש  
באופרטור \*

```
void main()
{
    int x = 3, y;
    int* pX = &x;
    y = *pX; // same as y = x;
    printf("x = %d\n", x);
    printf("*pX = %d\n", *pX);
    printf("y = %d\n", y);
}
```

תפנה לתוכן שבכתובת 1000

int: x	<b>3</b>	<b>1000</b>
int: y	<b>3</b>	<b>1004</b>
int*: pX	<b>1000</b>	<b>1008</b>



```
C:\WINDOWS\system32\cmd.exe
x = 3
*pX = 3
y = 3
Press any key to continue . . . _
```

```
void main()
{
    int x = 7;
    int* px = &x;
    int** ppx = &px;
    int*** pppx = &ppx;
```

1000	<b>7</b>	int: x
1004	<b>1000</b>	int*: px
1008	<b>1004</b>	int**: ppx
1012	<b>1008</b>	int***: pppx

```
printf("x      = %d \t &x=%p\n", x, &x);
printf("px     = %p   &px=%p\n", px, &px);
printf("ppx    = %p   &ppx=%p\n", ppx, &ppx);
printf("pppx   = %p   &pppx=%p\n", pppx, &pppx);
printf("pppx   = %p\n", pppx);
printf("*pppx   = %p\n", *pppx);
printf("***pppx = %p\n", **pppx);
printf("****pppx = %d\n", ***pppx);
```

```
}
```

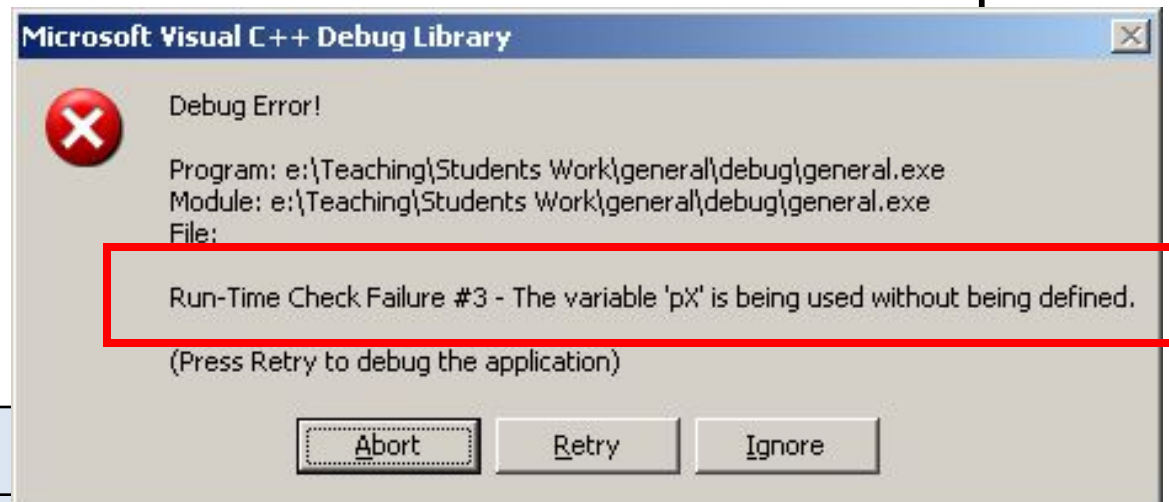
- ☐ **x = 7    &x= 1000**
- ☐ **px = 1000   &px=1004**
- ☐ **ppx = 1004   &ppx=1008**
- ☐ **pppx = 1008   &pppx=1012**
- ☐ **pppx = 1008**
- ☐ **\*pppx = 1004**
- ☐ **\*\*pppx = 1000**
- ☐ **\*\*\*pppx = 7**

# אתחול מצביעים

- כמו כל משתנה אחר, גם משתנה מטיפוס מצביע מכיל זבל עד אשר הוא מאותחל
- כאשר ננסה לפנות לתוכן של מצביע שהתוכן שלו הוא זבל – התוכנית תעוקף!! כי למעשה אנחנו מנסים לגשת לתא זיכרון שלא קיים

```
void main()
{
    int x = 3, y;
    int* pX;
    y = *pX;
}
```

int: x	3	
int: y	???	1004
int*: pX	???	1008



# איך נראה מצביע מזובל בקומפיילר

- ראינו שהכתובות שאיתן אנו מדגימים אינן הכתובות שהקומפיילר משתמש
- כתובת מזובלת בקומפיילר אינה ??? אלא כתובת מיוחדת בהקסה-דצימלי:

```
void main()
```

```
{
```

```
    int x;
```

```
    int* pX = &x, *p;
```

```
    printf("pX=%p, p=%p\n", pX, p);
```

```
}
```

פה התוכנית תעוף כי מנסים לגשת לכתובת מזובלת

Watch 1	
Name	Value
+ pX	0x0012ff60
+ p	0xffffffff
x	-858993460

# מצביעים – מה יקרה בתוכנית? (1) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    printf("&num1 = %p\n", &num1);
    printf("&p1 = %p\n", &p1);
}
```

יודפס: &num1 = 1000  
יודפס: &p1 = 1008

int: num1	<b>10</b>	<b>1000</b>
int: num2	<b>20</b>	<b>1004</b>
int*: p1	<b>???</b>	<b>1008</b>
int*: p2	<b>???</b>	<b>1012</b>

# מצביעים – מה יקרה בתוכנית? (2) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    p1 = &num1;
    num2 = *p1;
    printf("p1=%p\n", p1);
    printf("num2=%d\n", num2);
    printf("&num2=%p\n", &num2);
}
```

יודפס: p1 = 1000

יודפס: num2 = 10

יודפס: num2 = 1004&

int: num1	<b>10</b>	<b>1000</b>
int: num2	<b>10</b>	<b>1004</b>
int*: p1		<b>1008</b>
int*: p2	<b>1000</b>	<b>1008</b>
int*: p2	<b>???</b>	<b>1012</b>



# מצביעים – מה יקרה בתוכנית? (3) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    *p1 = num1;
    num2 = *p1;
    printf("num2=%d\n", num2);
}
```

התוכנית תעוף כי מנסים לגשת לתוכן של זבל..

int: num1	<b>10</b>	1000
int: num2	<b>20</b>	1004
int*: p1	<b>???</b>	1008
int*: p2	<b>???</b>	1012

# מצביעים – מה יקרה בתוכנית? (4) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    p1 = &num1;
    *p1 = num2;
    printf("num1=%d\n", num1);
}
```

יודפס: num1 = 20

int: num1	<b>20</b>	<b>1000</b>
int: num2	<b>20</b>	<b>1004</b>
int*: p1	<b>1000</b>	<b>1008</b>
<del>int*: p1</del> int*: p2		<del>1008</del> <b>1012</b>
int*: p2	<b>???</b>	<b>1012</b>

# מצביעים – מה יקרה בתוכנית? (5) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    p1 = &num1;
    *p1 = &num2;
    *p1 = *p2;
    printf("num1=%d\n", num1);
}
```

לא יתקמפל כי מנסים לשים כתובת בתוך משתנה המכיל int:  
cannot convert from int\* to int

int: num1	<b>10</b>	1000
int: num2	<b>20</b>	1004
int*: p1	<b>1000</b>	1008
int*: p2		1012
int*: p2		1012
int*: p2	<b>???</b>	1012

# מצביעים – מה יקרה בתוכנית? (6) (הנחה: הזיכרון מתחיל בכתובת 1000)

```
void main()
{
    int num1=10, num2=20;
    int *p1, *p2;

    p1 = &num1;
    &num2= p1;
    printf("num2=%d\n", num2);
}
```

לא יתקמפל כי לא ניתן לשנות את הכתובת של משתנה, אלא רק את ערכו!!

int: num1	<b>10</b>	1000
int: num2	<b>20</b>	1004
int*: p1		1008
int*: p2	<b>1000</b>	1012
int*: p2	<b>???</b>	1016

# העברה by value – דוגמא: swap

המטרה: פונקציה המקבלת 2 מספרים ומחליפה ביניהם

```
void swap(int a, int b)
{
    printf("In function, before swap: a=%d, b=%d\n", a, b);
    int temp = b;
    b = a;
    a = temp;
    printf("In function, after swap: a=%d, b=%d\n", a, b);
}
```

int: a	<b>3</b>	<b>2000</b>
int: b	<b>2</b>	<b>2004</b>
int: temp	<b>3</b>	<b>2008</b>

הזיכרון של swap

```
void main()
{
```

הפונקציה לא באמת החליפה בין  
הערכים

int: num1	<b>2</b>	<b>1000</b>
int: num2	<b>3</b>	<b>1004</b>

הזיכרון של main

```
    int num1=2, num2=3;

    printf("In main, before swap: num1=%d, num2=%d\n", num1, num2);
    swap(num1, num2);
    printf("In main, after swap: num1=%d, num2=%d\n", num1, num2);
}
```

```
C:\WINDOWS\system32\cmd.exe
In main, before swap: num1=2, num2=3
In function, before swap: a=2, b=3
In function, after swap: a=3, b=2
In main, after swap: num1=2, num2=3
Press any key to continue . . .
```

# העברה by pointer – דוגמא: swap

המטרה: פונקציה המקבלת 2 מספרים ומחליפה ביניהם

```
void swap(int* a, int* b)
{
    printf("In function, before swap: *a=%d, *b=%d\n", *a, *b);
    int temp = *b;
    *b = *a;
    *a = temp;
    printf("In function, after swap: *a=%d,* b=%d\n",* a, *b);
}
```

```
void main()
{
    int num1=2, num2=3;

    printf("In main, before swap: num1=%d, num2=%d\n", num1, num2);
    swap(&num1, &num2);
    printf("In main, after swap: num1=%d, num2=%d\n", num1, num2);
}
```

int*: a	<b>1000</b>	2000
int*: b	<b>1004</b>	2004
int temp	<b>3</b>	הזיכרון 2008

```
C:\Teaching\Students Work\general\debug\general.exe
In main, before swap: num1=2, num2=3
In function, before swap: *a=2, *b=3
In function, after swap: *a=3,* b=2
In main, after swap: num1=3, num2=2
```

int: num1	<b>3</b>	1000
int: num2	<b>2</b>	1004

הזיכרון של ה-main

# העברת פרמטר לפונקציה – by pointer

---

ראינו: □

- כאשר מעבירים משתנה לפונקציה עותק שלו מועבר למחסנית של הפונקציה (העברה by value)
- אם בפונקציה משנים את הפרמטר זה לא משפיע על המשתנה המקורי

□ גם כאשר מעבירים משתנה מטיפוס מצביע לפונקציה מעבירים עותק (של הכתובת), אבל כאשר מבצעים שינוי **בתוכן המצביע** בפונקציה אז השינוי משפיע גם על המשתנה המקורי

□ העברת פרמטר מטיפוס מצביע לפונקציה המשנה את תוכן המצביע נקראת העברה by pointer

# החזרת יותר מערך יחיד מפונקציה

---

- למשל נרצה לכתוב פונקציה המקבלת מערך, וצריכה להחזיר מהו המספר המקסימאלי ומה המינימאלי
- מאחר וניתן ע"י הפקודה return להחזיר ערך אחד בלבד אנחנו בבעיה...
- הפתרון הוא להעביר כפרמטר by pointer משתנה שיכיל לבסוף את התוצאה



# מציאת מינימום ומקסימום

```
void minMax(int arr[], int size, int* min, int* max)
```

```
{
    int i;
    *min = *max = arr[0];
    for (i=1; i < size; i++)
    {
        if (*min > arr[i])
            *min = arr[i];
        if (*max < arr[i])
            *max = arr[i];
    }
}
```

```
void main()
```

```
{
    int arr[] = {5,2,8};
    int minimum, maximum;
```

```
    minMax(arr, sizeof(arr)/sizeof(arr[0]), &minimum, &maximum);
    printf("max is %d and min is %d\n", maximum, minimum);
}
```

int: size	3	2000
	101	

```
C:\WINDOWS\system32\cmd.exe
max is 8 and min is 2
Press any key to continue . . .
```

int:i	5	2012
int[: arr	5	1000
	2	1004
	8	1008
int:minimum	2	1012
int:maximum	8	1016

הזיכרון של ה- main

תזכורת: כאשר מעבירים  
מערך לפונקציה מתייחסים  
למערך המקורי, ולא לעותק  
שלו!

ניתן היה להעביר לפונקציה רק את `min` או רק את `max` ואת הערך השני להחזיר ע"י `return`

■ אבל: כאשר הפונקציה מחזירה יותר מערך אחד והם כולם בעלי אותה תפקיד, נעדיף שכולם יוחזרו `by pointer` (אחידות בסגנון)

■ אם מעבירים לפונקציה פרמטר `by pointer` שהפונקציה מתבססת על ערכו, חובה לאתחלו בפונקציה, ולא להתבסס על אתחול (שאולי) בוצע בפונקציה שקראה

# אתחול פרמטר המועבר by pointer

```
void countPositive(int arr[], int size, int* count)
```

```
{
```

```
    int i;
```

```
    *count = 0; // it is our responsibility to initialize the value!
```

```
    for ( i=0 ; i < size ; i++ )
```

```
    {
```

```
        if (arr[i] > 0)
```

```
            (*count)++;
```

```
    }
```

```
}
```

```
void main()
```

```
{
```

```
    int arr[] = {-4,2,-8};
```

```
    int numOfPositive; // we can't assume that who wrote  

// the main initialized that variable!!
```

```
    countPositive(arr, sizeof(arr)/sizeof(arr[0]), &numOfPositive);
```

```
    printf("There are %d positive numbers in the array\n", numOfPositive);
```

```
}
```

int: size	<b>3</b>	2000
int*: count	<b>101</b> <b>2</b>	2004
int: i	<b>3</b>	2008

int[: arr	<b>-4</b>	1000
	<b>2</b>	1004
	<b>-8</b>	1008
int: numOfPositive	<b>1</b>	1012

הזיכרון של ה- main

# פונקציה המחזירה מצביע: כתובת האיבר המקסימלי

```
int* getAddressOfMaxElem(int arr[], int size)
{
```

```
    int i, maxIndex = 0;
```

```
    for ( i=1 ; i < size ; i++ )
        if (arr[i] > arr[maxIndex])
            maxIndex = i;
    return &arr[maxIndex];
```

```
{
void main()
```

```
{
    int arr[] = {3,7,2};
    int size = sizeof(arr) / sizeof(arr[0]);
    int* pMax =  getAddressOfMaxElem(arr, size);
```

```
    printf("Max value is at address %p and is %d\n", pMax, *pMax);
```

```
{
```

int: size	<b>3</b>	<b>2000</b>
int: i	<b>3</b>	<b>2004</b>
int: maxIndex	<b>1</b>	<b>2008</b>

הזיכרון של getAddressOfMaxElem

int[]: arr	<b>3</b>	<b>1000</b>
	<b>7</b>	<b>1004</b>
	<b>2</b>	<b>1008</b>
int: size	<b>3</b>	<b>1012</b>
int*: pMax	<b>1004</b>	<b>1016</b>

הזיכרון של ה- main

# אתחול מצביע

- ראינו שניתן לאתחל מצביע עם כתובת של משתנה מהטיפוס המתאים
- ראינו שכאשר לא מאתחלים מצביע, אז כמו כל משתנה לא מאותחל, הוא מכיל לזבל
- ניתן לאתחל מצביע שלא מצביע לשום-מקום בערך מיוחד הנקרא NULL, שזוהי למעשה הכתובת 0
- פניה למצביע שהוא NULL לא תגרום לתעופת התוכנית
- פניה לתוכן של מצביע שהוא NULL כן תגרום לתעופת התוכנית, כי אין בו כלום..

# פניה למצביע NULL לעומת מצביע זבל

```
void main()
```

```
{
```

```
int* p1 = NULL, *p2;
```

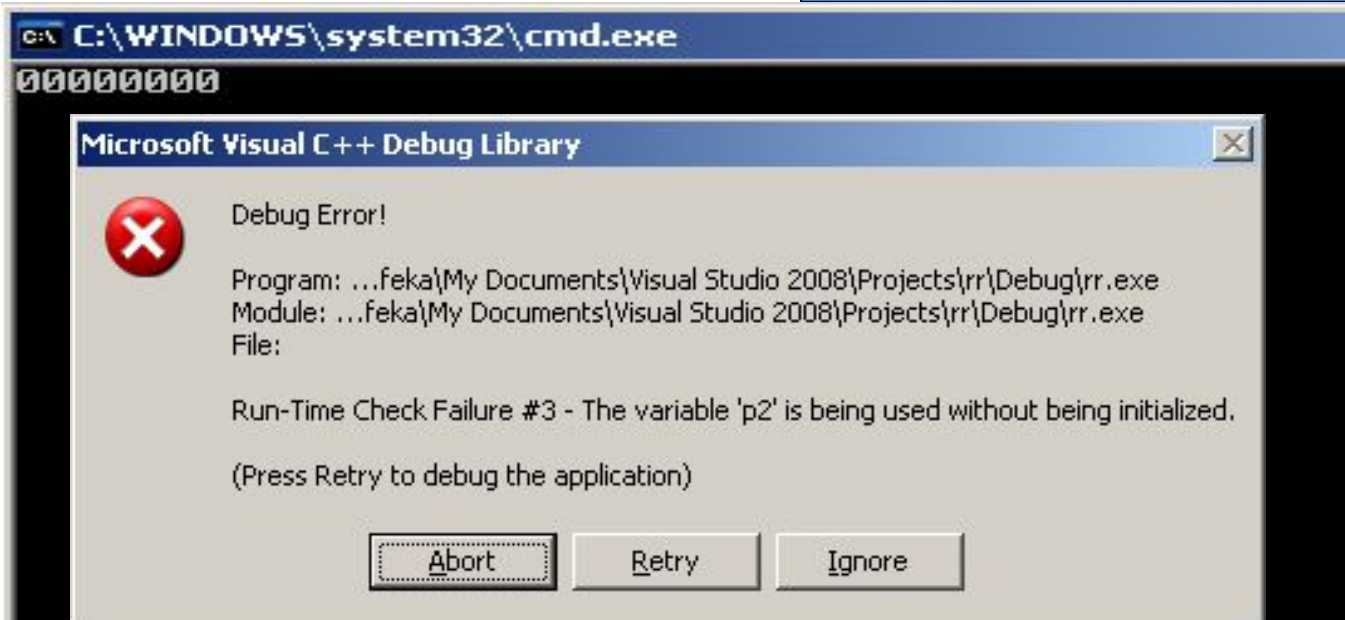
```
printf("%p", p1);
```

```
printf("%p", p2);
```

שורה זו עוברת בהצלחה  
ומדפיסה את הכתובת NULL

ניסיון הדפסה לכתובת זבל  
מעיף את התוכנית

```
{
```



# אתחול מצביע ל- NULL - דוגמא

```
void main()
```

```
{
```

```
    int x;
```

```
    int* pX = &x, *p = NULL;
```

```
    printf("pX=%p, p=%p\n", pX, p);
```

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

```
}
```

פה התוכנית לא תעוף כי ניגשים לכתובת מאופסת

C:\e:\Teaching\Students Work\ge

pX=0012FF60, p=00000000

Watch 1

Name	Value
pX	0x0012ff60
	-858993460
p	0x00000000
	CXX0030: Error: expression cannot be evaluated
x	-858993460
&x	0x0012ff60
	-858993460

פה התוכנית כן תעוף כי ניגשים לתוכן של מקום שאין בו כלום

# פונקציה המחזירה NULL: כתובת איבר לחיפוש

```
int* findNumber(int arr[], int size, int lookFor)
```

```
{
```

```
    int i;
```

```
    for ( i=0 ; i < size ; i++ )
```

```
        if (arr[i] == lookFor)
```

```
            return &arr[i];
```

```
    return NULL;
```

```
{
```

```
void main()
```

```
{
```

```
    int arr[] = {3,7,2};
```

```
    int size = sizeof(arr) / sizeof(arr[0]);
```

```
    int* p = findNumber(arr, size, 4);
```

```
    if (p != NULL)
```

```
        printf("The number is found at address %p\n", p);
```

```
    else    printf("The number is not in the array\n");
```

```
{
```

int: size	<b>3</b>	<b>2000</b>
int: i	<b>3</b>	<b>2004</b>
int: lookFor	<b>4</b>	<b>2008</b>

הזיכרון של findNumber

int[: arr	<b>3</b>	<b>1000</b>
	<b>7</b>	<b>1004</b>
	<b>2</b>	<b>1008</b>
int: size	<b>3</b>	<b>1012</b>
int*: p	<b>NULL</b>	<b>1016</b>

הזיכרון של ה- main



# משתנה const - תזכורת

---

ראינו כי ניתן להגדיר משתנה כ-const, ואז לא ניתן לשנות את ערכו במהלך ריצת התוכנית

```
void main()  
{  
    const double PI = 3.14;  
    PI = 3.1417; // l-value specifies const object  
}
```

# מצביע const על תוכן ההצבעה

```
void main()
```

```
{
```

```
int x = 2, y;
```

```
const int* pX = &x;
```

עדיין ניתן לפנות ל-x ישירות ולשנות את ערכו

```
x = 5;
```

```
*pX = 4; // l-value specifies const object
```

```
pX = &y;
```

```
{
```

לא ניתן לפנות ל- pX\* ולשנות את ערכו

מצביע const למשתנה אינו הופך את המשתנה ל- const, אלא אך ורק בעיני המצביע עצמו!

int: x	5	1000
int: y	???	1004
const int*: pX	100	1008
const int*: pX	4	1008

# מצביע const על ההצבעה

```
void main()
```

```
{
```

```
int x = 2, y;
```

```
int* const pX = &x;
```

```
x = 5;
```

```
*pX = 4;
```

```
pX = &y;
```

// l-value specifies const object

```
{
```

התוכן של pX קבוע ולא ניתן לשנות את ערכו לאחר האתחול

int: x	4	1000
int: y	???	1004
int* const: pX	100	1008
int* const: pX	0	1008

הזיכרון של ה-main

# סיכום: מצביע const

---

- ניתן גם להגדיר מצביע כ- `const`
- ישנם 2 אופנים להגדיר מצביע כ- `const`:

1. כך שלא ניתן לשנות את התוכן בכתובת שהמשתנה מצביע מכיל:

```
const <type>* var;
```

2. כך שלא ניתן לשנות את הכתובת אותה המשתנה מצביע מכיל:

```
<type>* const var;
```

# שימוש במצביע `const` בהעברת פרמטר לפונקציה

---

- כאשר מעבירים נתונים לפונקציה, למעשה מעבירים העתק שלהם (by value), אלא אם מעבירים אותם by pointer
- ראינו שכאשר מעבירים מערך לפונקציה, למעשה מעבירים את המערך המקורי, ולא העתק (by pointer)
- הפונקציה יכולה "בטעות" לשנות אותו
- לכן פונקציות המקבלות מערך ללא כוונה לשנות אותו, יצהירו על הפרמטר שהוא `const`

# דוגמא להעברת פרמטר כ- const

---

הפונקציה מצהירה שלא תשנה את ערכי המערך

```
void foo(const int arr[], int size)
```

```
{  
  arr[0] = 10; // l-value specifies const object  
}
```

ניסיון לשנות את תוכן המערך, בניגוד להצהרה!

# ביחידה זו למדנו:

---

- מהו מצביע (פוינטר)
- מוטיבציה למצביעים
- אופרטור &
- אופרטור \*
- אתחול מצביע
- העברת פרמטר לפונקציה by pointer
- מצביע const

# תרגיל 1: אם מתקמפל מה הפלט, אחרת מהי השגיאה

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

```
void func(int** ptr)
{
    **ptr = 99;
}
```

```
void main()
{
    int x = 30;
    int* pX = &x;
    func(&pX);
    printf("%d\n", *pX); 99
}
```

int**: ptr	<b>1004</b>	2000
------------	-------------	------

הזיכרון של func

int: x	<b>99</b>	1000
int*: pX	<b>1000</b>	1004

הזיכרון של ה-main



## תרגיל 2: אם מתקמפל מה הפלט, אחרת מהי השגיאה

```
void myFunc(int** x, int* y, int z)
{
    y = &z;
    x = &y;
}
```

```
void main()
{
    int x=3, *y, **z;
    myFunc(x, y, z);
    printf("x=%d *y=%d **z=%d\n", x, *y, **z);
}
```

int**: x	<b>3</b>	<b>2000</b>
int*: y	<b>???</b>	<b>2004</b>
int: z	<b>???</b>	<b>2008</b>

הזיכרון של myFunc

int: x	<b>3</b>	<b>1000</b>
int*: y	<b>???</b>	<b>1004</b>
int**: z	<b>???</b>	<b>1008</b>

הזיכרון של ה- main

# תרגיל 3: אם מתקמפל מה הפלט, אחרת מהי השגיאה

```
void foo(int *ptr1, int *ptr2)
{
    if (*ptr1 > *ptr2)
        *ptr1 = *ptr2;
}
```

int*: ptr1	<b>1000</b>	2000
int*: ptr2	<b>1004</b>	2004

הזיכרון של foo

```
void main()
{
    int a=8, b[]={1,2,3};
    char ch='a';
    int *p1, *p2;
    p1 = &a;
    p2 = b;
```

```
printf("%d %d\n", *p1+*p2, *p2-b[1]);
```

```
foo(&a,b);
a = ch;
printf("%d\n", a);
```

```
}
```

int: a	<b>97</b>	1000
int[]: b	<b>1</b>	1004
	<b>2</b>	1008
	<b>3</b>	1012
char: ch	<b>'a'</b>	1016
int*: p1	<b>1000</b>	1017
int*: p2	<b>1004</b>	1021

הזיכרון של ה-main

**9 -1**

**97**

## תרגיל 4:

□ כתוב פונקציה המקבלת לפחות את הפרמטרים הבאים:

■ מערך של מספרים וגודלו, ו- 2 מספרים נוספים

□ הפונקציה תחזיר את כמות האיברים שערכם מתחלק

ללא שארית במספר הראשון וכן את כמות האיברים

שערכם מתחלק ללא שארית במספר השני

□ דוגמא: עבור המערך

$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ , גודלו 11 והמספרים

2 ו- 4, הפונקציה תחזיר:

■ 5 (מאחר ו- 5 ערכים מתחלקים ב- 2 ללא שארית)

■ 2 (מאחר ו- 2 ערכים מתחלקים ב- 4 ללא שארית)