

# כתובות ומערכים אריתמטיקה של כתובות



קרון כליף

# ביחידה זו נלמד:

---

- הקשר בין מערך לכתובת
- פעולות חיבור וחסור עם כתובות
- מצביע מטייל על מערך
- הפונקציות `strchr` ו-`strstr`
- העברת מערך לפונקציה
- הבעייתיות בהחזרת מערך מפונקציה
- מערך של כתובות
- איתחול מצביע למחרוזת קבועה
- העברת מצביע לפונקציה לצורך שינוי הצבעתו
- הפונקציה `strtok`
- הפונקציה `memcpy`

# הקשר בין מערך וכתובת

- כאשר פונים למשתנה רגיל מקבלים את הערך בתא
- כאשר פונים למערך, מקבלים את כתובת תחילת המערך

```
void main()
```

```
{
```

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

```
printf("In main1: The array starts at %p\n", arr);
```

```
printf("In main2: The array starts at %p\n", &arr);
```

```
}
```

פניה לשם המערך נותנת לנו את כתובת ההתחלה שלו!

ואפשר גם כך:

פניה לכתובת של arr

יודפס: In main: The array starts at 1000

```
C:\WINDOWS\system32\cmd.exe
In main1: The array starts at 0012FF58
In main2: The array starts at 0012FF58
Press any key to continue . . . _
```

|            |   |      |
|------------|---|------|
| int[]: arr | 4 | 1000 |
|            | 2 | 1004 |
|            | 8 | 1008 |

## הקשר בין מערך וכתובת (2)

---

- מאחר ושם המערך הוא למעשה כתובת תחילת המערך, ניתן להפעיל עליו את האופרטור \*
- ראינו כי כאשר מפעילים את האופרטור \* על משתנה המכיל כתובת של `int`, אנו מקבלים את הערך שבתוך התא של הכתובת, כלומר `int` כלשהו
- במערך, כתובת ההתחלה מכילה את האיבר הראשון במערך, לכן הפעלת האופרטור \* על שם המערך תחזיר את הערך של האיבר הראשון במערך

# הקשר בין מערך וכתובת (3)

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

```
void main()
```

```
{
```

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

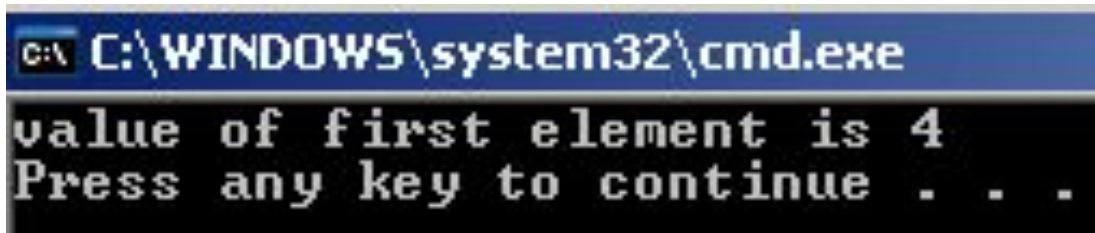
```
    printf("value of first element is %d\n", *arr);
```

```
}
```

כתובת 1000

התוכן שבכתובת 1000

כתובת ההתחלה  
מכילה את האיבר  
הראשון במערך



```
C:\WINDOWS\system32\cmd.exe
value of first element is 4
Press any key to continue . . .
```

|            |          |      |
|------------|----------|------|
| int[]: arr | <b>4</b> | 1000 |
|            | <b>2</b> | 1004 |
|            | <b>8</b> | 1008 |

# פעולות אריתמטיות על כתובות

□ מוגדרות 3 פעולות אריתמטיות לפעולות עם כתובות:

1. כתובת + מספר שלם □ כתובת

■ כאשר מחברים כתובת  $p$  לטיפוס  $\text{type}$  מספר שלם  $k$   
התוצאה:

$$p+k = p + k*\text{sizeof}(\text{type})$$

2. כתובת - מספר שלם □ כתובת

■ כאשר מחסרים מכתובת  $p$  לטיפוס  $\text{type}$  מספר שלם  $k$  התוצאה:

$$p-k = p - k*\text{sizeof}(\text{type})$$

3. כתובת- כתובת □ מספר שלם

# פעולות אריתמטיות על כתובות – דוגמא 1

```
void main()
```

```
{
```

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

```
    int* p = arr;
```

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

```
    p++;
```

```
    printf("&(arr+1)=%p, p=%p\n", arr+1, p);
```

```
    printf("*(arr+1)=%d, *p=%d\n", *(arr+1), *p);
```

```
}
```

מאחר וזוהי כתובת של int, קידום ב- 1 ייתן:

$$p+k = p + k*\text{sizeof}(\text{type})$$
$$1000 + 1*4 = 1004$$

יודפס: arr=1000, p=1000&

יודפס: 1004, p=1004=&(arr+1)

יודפס: 2, \*p=2=\*(arr+1)\*

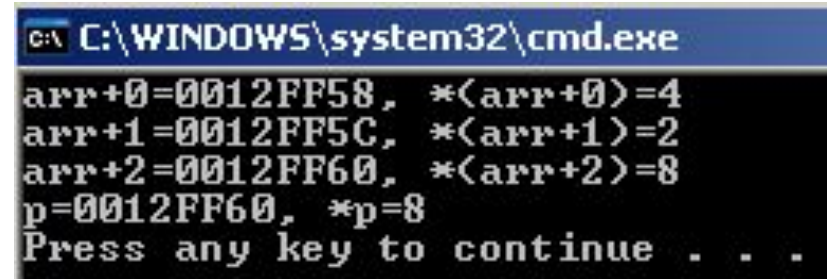
|            |     |      |
|------------|-----|------|
| int[]: arr | 4   | 1000 |
|            | 2   | 1004 |
|            | 8   | 1008 |
| int*: p    | 100 | 1012 |
| int*: p    | 4   | 1012 |

```
C:\WINDOWS\system32\cmd.exe
&arr=0012FF58, p=0012FF58
&(arr+1)=0012FF5C, p=0012FF5C
*(arr+1)=2, *p=2
Press any key to continue . . .
```

## פעולות אריתמטיות על כתובות – דוגמא 2

```
void main()
{
    int arr[] = {4,2,8};
    int* p = NULL;

    printf("arr+0=%p, *(arr+0)=%d\n", (arr+0), *(arr+0));
    printf("arr+1=%p, *(arr+1)=%d\n", (arr+1), *(arr+1));
    printf("arr+2=%p, *(arr+2)=%d\n", (arr+2), *(arr+2));
    p = arr + 2;
    printf("p=%p, *p=%d\n", p, *p);
}
```



```
C:\WINDOWS\system32\cmd.exe
arr+0=0012FF58, *(arr+0)=4
arr+1=0012FF5C, *(arr+1)=2
arr+2=0012FF60, *(arr+2)=8
p=0012FF60, *p=8
Press any key to continue . . .
```

|            |             |             |
|------------|-------------|-------------|
| int[]: arr | <b>4</b>    | <b>1000</b> |
|            | <b>2</b>    | <b>1004</b> |
|            | <b>8</b>    | <b>1008</b> |
| int*: p    | <b>1008</b> | <b>1012</b> |
| int*: p    | <b>8</b>    | <b>1012</b> |

arr+0=1000, \*(arr+0)=4 :יודפס  
arr+1=1004, \*(arr+1)=2 :יודפס  
arr+2=1008, \*(arr+2)=8 :יודפס  
p=1008, \*p=8 :יודפס



# פעולות אריתמטיות על כתובות – דוגמא 3

```
void main()
{
    int arr[] = {4,2,8}, i;
    int* p = NULL;
```

```
C:\WINDOWS\system32\cmd.exe
arr+0=0012FF58, *(arr+0)=4
arr+1=0012FF5C, *(arr+1)=2
arr+2=0012FF60, *(arr+2)=8
p=0012FF60, *p=8
Press any key to continue . . .
```

```
for (i=0 ; i < sizeof(arr)/sizeof(arr[0]) ; i++)
    printf("arr+%d=%p, *(arr+%d)=%d\n", i, (arr+i), i, *(arr+i));
```

```
p = arr + 2;
printf("p=%p, *p=%d\n", p, *p);
}
```

וכמובן שניתן גם עם לולאה...

|            |             |             |
|------------|-------------|-------------|
| int[]: arr | <b>4</b>    | <b>1000</b> |
|            | <b>2</b>    | <b>1004</b> |
|            | <b>8</b>    | <b>1008</b> |
| int*: p    | <b>1008</b> | <b>1012</b> |
| int*: p    | <b>8</b>    | <b>1012</b> |

יודפס: arr+0=1000, \*(arr+0)=4  
יודפס: arr+1=1004, \*(arr+1)=2  
יודפס: arr+2=1008, \*(arr+2)=8  
יודפס: p=1008, \*p=8

# חיסור בין כתובות

חיסור בין כתובות נותן את מספר התאים ביניהם (ולא את הפרש הכתובות!) □

```
void main()
{
    int arr[] = {4,2,8};
    int* p;
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("The values in the array: ");
    1000-1000 = 0
    for ( p=arr ;  $\overbrace{p-arr}^{1000-1000} < size$  ; p++ )
        printf("%d ", *p);
    printf("\n");
}
```

|            |             |      |
|------------|-------------|------|
| int[]: arr | <b>4</b>    | 1000 |
|            | <b>2</b>    | 1004 |
|            | <b>8</b>    | 1008 |
| int*: p    | <b>1012</b> | 1012 |
| int: size  | <b>3</b>    | 1016 |

# פעולות אריתמטיות על כתובות – דוגמא 4

```
void main()
```

```
{
```

```
    int* p;
```

```
    int size;
```

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

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

```
    p = arr + size;
```

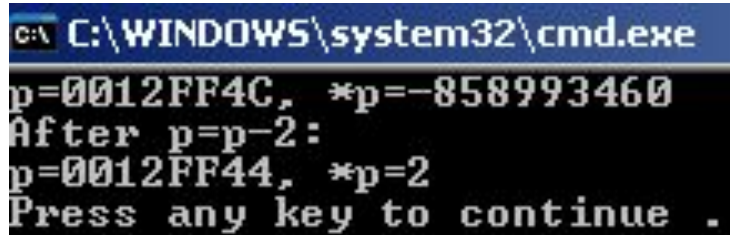
```
    printf("p=%p, *p=%d\n", p, *p);
```

```
    p = p - 2;
```

```
    printf("After p=p-2:\n");
```

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

```
}
```



```
C:\WINDOWS\system32\cmd.exe
p=0012FF4C, *p=-858993460
After p=p-2:
p=0012FF44, *p=2
Press any key to continue .
```

|            |      |      |
|------------|------|------|
| int*: p    | 1012 | 1000 |
| int: size  | 3    | 1004 |
| int[]: arr | 4    | 1008 |
|            | 2    | 1012 |
|            | 8    | 1016 |
|            | ???  | 1020 |

יודפס: \*p=1020, ???

יודפס: \*p=2, p=1012

# שקילויות

ניתן לראות כי אם `arr` הינו שם של מערך אזי ערך הביטוי `(arr+i)` הוא כתובת האיבר ה- `i` במערך, והביטוי `*(arr+i)` הינו תוכן האיבר ה- `i`:

- `&arr[i] ≡ (arr+i)`
- `arr[i] ≡ *(arr+i)`

| Watch 1                  |                         | Watch 1               |                |
|--------------------------|-------------------------|-----------------------|----------------|
| Name                     | Value                   | Name                  | Value          |
| <code>arr+0</code>       | <code>0x0012ff58</code> | <code>*(arr+0)</code> | <code>4</code> |
| <code>&amp;arr[0]</code> | <code>0x0012ff58</code> | <code>arr[0]</code>   | <code>4</code> |
| <code>arr+1</code>       | <code>0x0012ff5c</code> | <code>*(arr+1)</code> | <code>2</code> |
| <code>&amp;arr[1]</code> | <code>0x0012ff5c</code> | <code>arr[1]</code>   | <code>2</code> |
| <code>arr+2</code>       | <code>0x0012ff60</code> | <code>*(arr+2)</code> | <code>8</code> |
| <code>&amp;arr[2]</code> | <code>0x0012ff60</code> | <code>arr[2]</code>   | <code>8</code> |

```
void main()  
{
```

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

```
    printf("arr+1=%p, &arr[1]=%p\n", (arr+1), &arr[1]);  
    printf("*(arr+1)=%d, arr[1]=%d\n", *(arr+1), arr[1]);
```

```
}
```

```
C:\WINDOWS\system32\cmd.exe  
arr+1=0012FF5C, &arr[1]=0012FF5C  
*(arr+1)=2, arr[1]=2  
Press any key to continue . . .
```

# מעבר עולה על איברי מערך עם מצביע (ולא עם אינדקס) – פוינטר מטייל

```
C:\WINDOWS\system32\cmd.exe
Values in the array: 4 2 8
Press any key to continue . .
```

```
void main()
{
    int arr[] = {4,2,8};
    int* p;
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Values in the array: ");
    for ( p=arr; p < arr+size; p++)
        printf("%d ", *p);
    printf("\n");
}
```

|            |      |      |
|------------|------|------|
| int[]: arr | 4    | 1000 |
|            | 2    | 1004 |
|            | 8    | 1008 |
| int*: p    | 1012 | 1012 |
| int: size  | 3    | 1016 |

תזכורת:

$arr+size = \&arr + size * sizeof(int) = 1000 + 3 * 4 = 1012$

p הוא משתנה שמחזיק כל פעם את הכתובת של האיבר הבא במערך אותו רוצים להדפיס, ומאחר והוא מכיל כתובת ל- int הוא מטיפוס \*int. צריך לרוץ איתו עד אשר הוא יכיל את הכתובת של אחרי סיום המערך (התנאי לסיום הלולאה)

# מעבר יורד על איברי מערך עם מצביע (ולא עם אינדקס) – פוינטר מטייל

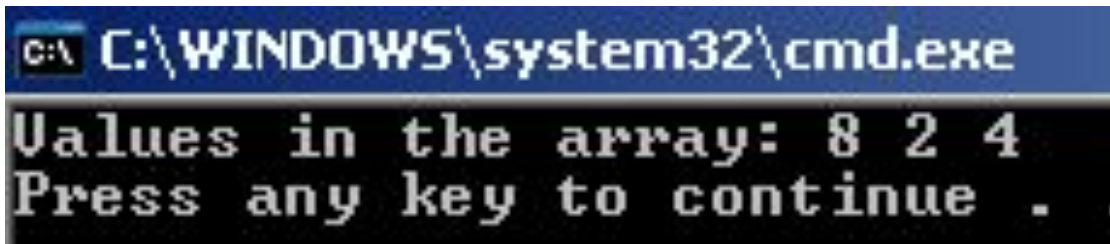
```
void main()
{
    int arr[] = {4,2,8};
    int* p;
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Values in the array: ");
    for ( p=arr+size-1; p >= arr ; p-- )
        printf("%d ", *p);
    printf("\n");
}
```

|            |            |             |
|------------|------------|-------------|
| int[]: arr | <b>4</b>   | <b>1000</b> |
|            | <b>2</b>   | <b>1004</b> |
|            | <b>8</b>   | <b>1008</b> |
| int*: p    | <b>996</b> | <b>1012</b> |
| int: size  | <b>3</b>   | <b>1016</b> |

תזכורת:

$$\begin{aligned} \text{arr} + \text{size} - 1 &= \&\text{arr} + (\text{size} - 1) * \text{sizeof}(\text{int}) = \\ &= 1000 + (3 - 1) * 4 = 1008 \end{aligned}$$



C:\WINDOWS\system32\cmd.exe

Values in the array: 8 2 4

Press any key to continue . .

# סדר פעולות

מה תהייה התוצאה של הפעולות הבאות:

```
int x=3;  
int* pX = &x;  
*pX++;
```

|          |             |      |
|----------|-------------|------|
| int: x   | <b>3</b>    | 1000 |
| int*: pX | <b>1000</b> | 1004 |

האם:

- קודם מקדמים את pX ל- 1004 ועל זה מפעילים \*?
- קודם מפעילים את \* על pX ועל התוכן מפעילים ++?

כאשר מופעלים כמה אופרטורים אונריים על משתנה סדר הפעולות הוא מימין לשמאל

- לסוגריים יש עדיפות, לכן התיקון יהיה ++(pX\*)

## נשים לב...

ראינו שניתן לבצע את הפעולה ++ על משתנה מטיפוס כתובת

אבל אסור לקדם מערך (++arr) אפילו ש-arr הוא כתובת תחילת המערך

```
void main()
```

```
{
```

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

```
    int* p = arr;
```

```
    p++;
```

```
arr++; // same as: arr = arr+1 □ arr = 1004
```

```
}
```

|            |             |             |
|------------|-------------|-------------|
| int[]: arr | <b>4</b>    | <b>1004</b> |
|            | <b>2</b>    | 1004        |
|            | <b>8</b>    | 1008        |
| int*: p    | <b>1004</b> |             |
|            | <b>4</b>    | 1012        |

שורה זו לא תתקמפל, כי לא ניתן לשנות  
!!את מיקומם של משתנים בזיכרון



# הפונקציה strchr

---

פונקציה המקבלת מחרוזת ותו, ומחזירה את הכתובת של המופע הראשון של התו במחרוזת, NULL אם לא קיים

```
char* strchr(const char str[], char ch)
```

```
void main()
```

```
{
```

```
    char str[] = "abcdef";
```

```
    char* pos;
```

```
    char ch = 'c';
```

```
    printf("|%s| appears at address %p\n\n", str, str);
```

```
    pos = strchr(str, ch);
```

```
    if (pos != NULL)
```

```
        printf("'c' appears first at address %p (index=%d)\n", ch, pos, pos-str);
```

```
    ch = 'm';
```

```
    pos = strchr(str, ch);
```

```
    if (pos == NULL)
```

```
        printf("'m' is not in the string\n", ch);
```

```
}
```

## הפונקציה strchr-דוגמא

|             |      |      |
|-------------|------|------|
| char[]: str | 'a'  | 1000 |
|             | 'b'  | 1001 |
|             | 'c'  | 1002 |
|             | 'd'  | 1003 |
|             | 'e'  | 1004 |
|             | 'f'  | 1005 |
|             | 0    | 1006 |
| char*: pos  | NULL | 1007 |
| char: ch    | 'm'  | 1011 |

```
!abcdef! appears at address 0012FF58
```

```
'c' appears first at address 0012FF5A (index=2)
```

```
'm' is not in the string
```

# הפונקציה strstr

---

- פונקציה המקבלת 2 מחרוזות, ובודקת האם המחרוזת השניה היא תת-מחרוזת בראשונה
- אם כן, מחזירה את כתובת תחילת הרצף בראשונה, אחרת מחזירה NULL

```
char* strstr(const char str1[],  
             const char str2[])
```

```
void main()
{
```

```
    char str[] = "abcdef", *pos, subStr[] = "cde";
```

```
    printf("|%s| appears at address %p\n\n", str, str);
```

```
    pos = strstr(str, subStr);
```

```
    if (pos != NULL)
```

```
        printf("|%s| starts at address %p (index=%d)\n\n",
               subStr, pos, pos-str);
```

```
    strcpy(subStr, "cdd");
```

```
    pos = strstr(str, subStr);
```

```
    if (pos == NULL)
```

```
        printf("|%s| is not a sub-string\n", subStr);
```

```
}
```

# הפונקציה strstr-דוגמא

|             |      |      |
|-------------|------|------|
| char[]: str | 'a'  | 1000 |
|             | 'b'  | 1001 |
|             | 'c'  | 1002 |
|             | 'd'  | 1003 |
|             | 'e'  | 1004 |
|             | 'f'  | 1005 |
|             | 0    | 1006 |
| char*: pos  | NULL | 1007 |
| char[]: sub | 'c'  | 1011 |
|             | 'd'  | 1012 |
|             | 'd'  | 1013 |
|             | 0    | 1014 |

```
!abcdef! appears at address 0012FF58
```

```
!cde! starts at address 0012FF5A (index=2)
```

```
!cdd! is not a sub-string
```

# העברת מערך לפונקציה

---

- ראינו שבעזרת כתובת תחילת המערך (שם המערך) ניתן לגשת לכל איברי המערך
- ראינו כאשר מעבירים מערך לפונקציה, הפונקציה יכולה לשנות את המערך המקורי
- **הסיבה היא שלא מועבר עותק של המערך לפונקציה, אלא מועברת כתובת ההתחלה שלו (שם המערך)**
- כאשר מעבירים מערך לפונקציה יש להעביר כפרמטר גם את גודלו

# העברת מערך לפונקציה

```
void printArray(int* arr[], int size)
```

```
{
    printf("In func: The array starts at %p\n", arr);
}
```

```
void main()
{
```

```
    int arr[] = {4, 2, 8};
    printf("In main: The array starts at %p\n", arr);
    printArray(arr, sizeof(arr)/sizeof(arr[0]));
}
```

```
C:\WINDOWS\system32\cmd.exe
In main: The array starts at 0012FF58
In func: The array starts at 0012FF58
Press any key to continue . . .
```

|                 |      |      |
|-----------------|------|------|
| int*: arr       | 1000 | 2000 |
| int: printArray | 0    | 2000 |

כאשר מעבירים מערך כפרמטר לפונקציה, למעשה מעבירים את כתובת ההתחלה שלו. לכן ניתן לכתוב בהצהרה שהפרמטר הוא `int* arr` או `[]int arr` בכל צורת כתיבה, ההתייחסות למערך בתוך הפונקציה היא כאל מצביע

|            |   |      |
|------------|---|------|
| int[]: arr | 4 | 1000 |
|            | 2 | 1004 |
|            | 8 | 1008 |

הזיכרון של ה-main

# דוגמא: פונקציה הסוכמת איברי מערך (1)

```
int sumArray(int* arr, int size)
{
    int i;
    int sum = 0;
    for ( i=0 ; i < size ; i++ )
        sum += arr[i];           // same as: sum += *(arr+i)
    return sum;
}
```



```
C:\WINDOWS\system
The sum is 14
Press any key to
```

```
void main()
{
    int arr[] = {4,2,8};
    printf("The sum is %d\n", sumArray(arr, sizeof(arr)/sizeof(arr[0])) );
}
```

|             |             |             |
|-------------|-------------|-------------|
|             | <b>1000</b> |             |
| int*: arr   | <b>0</b>    | <b>2000</b> |
| int: size   | <b>3</b>    | <b>2004</b> |
| int: i      | <b>3</b>    | <b>2008</b> |
| int: sumArr | <b>14</b>   | <b>2012</b> |

|            |          |             |
|------------|----------|-------------|
| int[]: arr | <b>4</b> | <b>1000</b> |
|            | <b>2</b> | <b>1004</b> |
|            | <b>8</b> | <b>1008</b> |

הזיכרון של ה- main

## דוגמא: פונקציה הסוכמת איברי מערך (2)

```
int sumArray(int* arr, int size)
{
    int* p;
    int sum = 0;
    for ( p=arr ; p < 1012arr+size ; p++)
        sum += *p;
    return sum;
}
```



```
void main()
{
    int arr[] = {4,2,8};
    printf("The sum is %d\n", sumArray(arr, sizeof(arr)/sizeof(arr[0])) );
}
```

|           |             |             |
|-----------|-------------|-------------|
|           | <b>1000</b> |             |
| int*: arr | <b>0</b>    | <b>2000</b> |
| int: size | <b>3</b>    | <b>2004</b> |
| int*: p   | <b>1012</b> | <b>2008</b> |
| int: sum  | <b>14</b>   | <b>2012</b> |

|            |          |             |
|------------|----------|-------------|
| int[]: arr | <b>4</b> | <b>1000</b> |
|            | <b>2</b> | <b>1004</b> |
|            | <b>8</b> | <b>1008</b> |

הזיכרון של ה- main



# דוגמא: סכימת רק חלק מאיברי המערך

```
int sumArray(int* arr, int size)
{
    int* p;
    int sum = 0;
    for ( p=arr ; p < 1012arr+size ; p++)
        sum += *p;
    return sum;
}
```

```
void main()
{
    int arr[] = {4,2,8};
    printf("The sum is %d\n",
        sumArray(arr+1, sizeof(arr)/sizeof(arr[0])-1)
    );
}
```

|           |             |             |
|-----------|-------------|-------------|
|           | <b>100</b>  |             |
| int*: arr | <b>4</b>    | <b>2000</b> |
| int: size | <b>2</b>    | <b>2004</b> |
| int*: p   | <b>1012</b> | <b>2008</b> |
| int: sum  | <b>10</b>   | <b>2012</b> |

|            |          |             |
|------------|----------|-------------|
| int[]: arr | <b>4</b> | <b>1000</b> |
|            | <b>2</b> | <b>1004</b> |
|            | <b>8</b> | <b>1008</b> |

הזיכרון של ה- main

# דוגמא: פונקציה המגדילה את כל איברי המערך

ב-1



```
void incArray(int* arr, int size)
{
    int* p;
    for ( p=arr ; p < arr+size ; p++)
        (*p)++;
}

void printArray(int* arr, int size)
{
    int* p;
    for ( p=arr ; p < arr+size ; p++ )
        printf("%d ", *p);
    printf("\n");
}

void main()
{
    int arr[] = {4,2,8};
    int size = sizeof(arr)/sizeof(arr[0]);
    incArray(arr, size);
    printf("The array after increment is: ");
    printArray(arr, size);
}
```

|           |             |             |
|-----------|-------------|-------------|
|           | <b>1000</b> |             |
| int*: arr | <b>0</b>    | <b>2000</b> |
| int: size | <b>3</b>    | <b>2004</b> |
| int*: p   | <b>1012</b> |             |

```
C:\WINDOWS\system32\cmd.exe
The array after increment is: 5 3 9
Press any key to continue . . .
```

|            |          |             |
|------------|----------|-------------|
| int[]: arr | <b>5</b> | <b>1000</b> |
|            | <b>3</b> | <b>1004</b> |
|            | <b>9</b> | <b>1008</b> |
| int: size  | <b>3</b> | <b>1012</b> |

הזיכרון של ה-main

# מדוע צריך להעביר לפונקציה את גודל המערך (ולא להסתמך על sizeof)

```
void printArraySize(int* arr)
{
```

```
    int size;
```

```
    printf("in function: sizeof(arr)= %d\n",sizeof(arr))
```

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

```
    printf("in function: size = %d\n", size);
```

```
}
```

```
void main()
```

```
{
```

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

```
    printf("in main: sizeof(arr)= %d\n",sizeof(arr));
```

```
    printArraySize(arr);
```

```
}
```

arr הוא מטיפוס כתובת, וגודלו של משתנה  
מטיפוס כתובת הוא תמיד 4 בתים...  
לכן כאשר מתייחסים לשם המערך בפונקציה  
לא ניתן לדעת את גודלו!  
רק בפונקציה שבה מוקצה שטח הזיכרון של  
המערך ניתן לדעת את גודלו ע"י sizeof!

```
in main: sizeof(arr)= 12
in function: sizeof(arr)= 4
in function: size = 1
Press any key to continue . . .
```

# דוגמא: מימוש הפונקציה strlen(1)

```
int myStrlen(char str[])
{
    int len = 0;
    for ( ; *str != '\0' ; str++)
        len++;
    return len;
}
```

```
void main()
{
    char str[4] = "Hi";
    printf("The len of |%s| is %d\n", str,
}
```

בפונקציה כן מותר לבצע ++ על שם המערך, שכן פה מתייחסים לתוכנו של משתנה, ולא משנים את כתובתו הפיזית של המערך

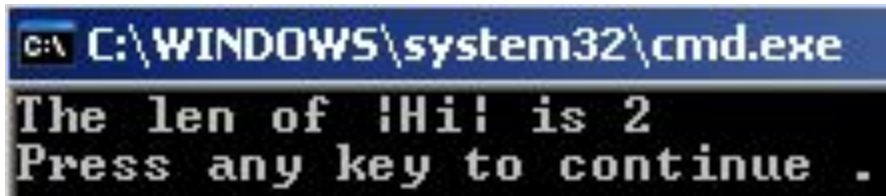
|            |             |             |
|------------|-------------|-------------|
| char*: str | <b>1002</b> | <b>2004</b> |
| Int: len   | <b>2</b>    | <b>2008</b> |

הזיכרון של ה- myStrlen

|             |            |             |
|-------------|------------|-------------|
| char[]: str | <b>'H'</b> | <b>1000</b> |
|             | <b>'i'</b> | <b>1001</b> |
|             | <b>0</b>   | <b>1002</b> |

הזיכרון של ה- main

myStrlen(str) );



```
C:\WINDOWS\system32\cmd.exe
The len of !Hi! is 2
Press any key to continue .
```

## דוגמא: מימוש הפונקציה strlen (2)

```
int myStrlen(char str[])
{
    char* p;
    for (p=str ; *p ; p++ );
    return p-str; // returns: 2
}
```

שקול ל: 'p != '\0\*

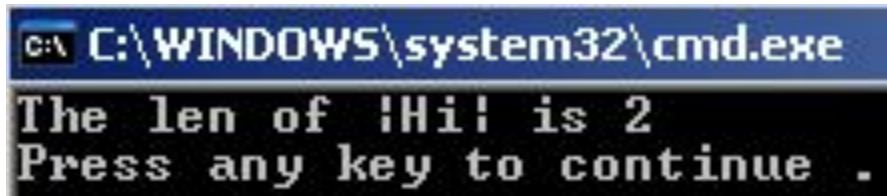
```
void main()
{
    char str[4] = "Hi";
    printf("The len of |%s| is %d\n", str, myStrlen(str) );
}
```

|            |      |      |
|------------|------|------|
| char*: str | 1000 | 2000 |
| char*: p   | 1002 | 2004 |

הזיכרון של ה- myStrlen

|             |     |      |
|-------------|-----|------|
| char[]: str | 'H' | 1000 |
|             | 'i' | 1001 |
|             | 0   | 1002 |

הזיכרון של ה- main



```
C:\WINDOWS\system32\cmd.exe
The len of !Hi! is 2
Press any key to continue .
```





# דוגמא: מימוש strcmp

```

int myStrcmp(char str1[], char str2[])
{
    while (*str1 == *str2)
    {
        if (*str1 == '\0')
            return 0;
        str1++;
        str2++;
    }

    if (*str1 < *str2)
        return -1;
    return 1;
}

void main()
{
    char str1[4] = "Hi", str2[4]="Hii";
    printf("Result of strcmp: %d\n", myStrcmp(str2, str1));
}

```

```

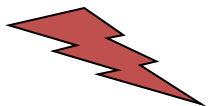
C:\WINDOWS\system32\cmd.exe
Result of strcmp: 1
Press any key to continue . . .

```

|             |            |             |
|-------------|------------|-------------|
|             | <b>100</b> |             |
| char*: str1 | <b>6</b>   | <b>2000</b> |
| myStrcmp    | <b>100</b> | הזי         |
| char*: str2 | <b>2</b>   | <b>2004</b> |

|              |            |             |
|--------------|------------|-------------|
| char[]: str1 | <b>'H'</b> | <b>1000</b> |
|              | <b>'i'</b> | <b>1001</b> |
|              | <b>0</b>   | <b>1002</b> |
|              | <b>0</b>   | <b>1003</b> |
| char[]: str2 | <b>'H'</b> | <b>1004</b> |
|              | <b>'i'</b> | <b>1005</b> |
|              | <b>'i'</b> | <b>1006</b> |
|              | <b>0</b>   | <b>1007</b> |

הזיכרון של ה- main



# דוגמא: מימוש strcat

```
Before: str1=|Hi|, str2=|You|
After: str1=|HiYou|, str2=|You|
Press any key to continue . . .
```

```
void myStrcat(char dest[], char src[])
{
    strcpy(dest+2strlen(dest), src);
}
```

1002

```
void main()
```

```
{
```

```
char str1[6] = "Hi", str2[4] = "You",
```

```
printf("Before: str1=|%s|, str2=|%s|\n", str1, str2);
```

```
myStrcat(str1, str2);
```

```
printf("After: str1=|%s|, str2=|%s|\n", str1, str2);
```

```
}
```

|             |          |             |
|-------------|----------|-------------|
| char*: dest | 100<br>0 | 2000        |
| myStrcat    | 100<br>6 | הזי<br>2004 |
| char*: src  |          |             |

|              |      |      |
|--------------|------|------|
| char[]: str1 | 'H'  | 1000 |
|              | 'i'  | 1001 |
|              | '\0' | 1002 |
|              | 'o'  | 1003 |
|              | 'u'  | 1004 |
|              | '\0' | 1005 |
| char[]: str2 | 'Y'  | 1006 |
|              | 'o'  | 1007 |
|              | 'u'  | 1008 |
|              | '\0' | 1009 |

הזיכרון של ה-main



# החזרת מערך מפונקציה

- פונקציה יכולה להחזיר כל טיפוס, פרט למערך (בינתיים)
- ראינו שכאשר פונים לשם המערך, פונים לכתובת ההתחלה שלו
- כאשר מעבירים מערך לפונקציה, מעבירים רק את כתובת ההתחלה שלו, ולא עותק של כל המערך
- ובאופן דומה, כאשר מחזירים מערך שהוגדר בפונקציה, חוזרת כתובת ההתחלה שלו, ולא עותק של כל המערך
- הבעייתיות: כאשר יוצאים מהפונקציה שטח הזיכרון שלה משתחרר ויש לנו מצביע לזיכרון שנמחק...
- הפתרון: כאשר נלמד על הקצאות דינאמיות

# החזרת מערך מפונקציה - דוגמא

```
#define SIZE 3
```

```
int* readArray()
```

```
{
    int arr[SIZE], i;
    printf("Please enter %d numbers: ", SIZE);
    for (i=0 ; i < SIZE ; i++)
        scanf("%d", &arr[i]);
    return arr;
}
```

```
void main()
```

```
{
    int* arr, i;
    arr = readArray();

    printf("The array is: \n");
    for (i=0 ; i < SIZE ; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

```
C:\WINDOWS\system32\cmd.exe
Please enter 3 numbers: 3 5 7
The array is:
270655952 -2 1245032
Press any key to continue . . .
```

|            |          |             |
|------------|----------|-------------|
| int[]: arr | <b>3</b> | <b>2000</b> |
|            | <b>5</b> | <b>2004</b> |
|            | <b>6</b> | <b>2008</b> |
| int: i     | <b>3</b> | <b>2012</b> |

הזיכרון של readArray

**הקומפיילר נותן warning:**  
**returning address of local variable or temporary**  
**שפירושה שאנחנו מחזירים כתובת למשתנה בזיכרון שישתחרר**

**לעולם לא נחזיר מפונקציה**  
**כתובת של משתנה**  
**שהוגדר מקומית בפונקציה!**

|           |             |             |
|-----------|-------------|-------------|
| int*: arr | <b>2000</b> | <b>1000</b> |
| int: i    | <b>???</b>  | <b>1004</b> |

הזיכרון של ה-main

# אריתמטיקה של מטריצות

□ כאשר מחברים לשם של מערך מספר  $i$ , מקבלים את כתובת האיבר  $i$

□ כאשר מחברים לשם של מטריצה מספר  $i$ , מקבלים את כתובת האיבר הראשון בשורה  $i$

```
void printArr(int* arr, int size)
{
    int i;
    for (i=0 ; i < size ; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void main()
{
```

```
1 2 3 4 5 6
1 2 3 4 5 6
4 5 6 -858993460 1245112 4266662
2 3 4 5 6 -858993460
```

נקבל warning כי הפונקציה מצפה לקבל כתובת התחלה של

עושים למטריצה casting ל- `*int` כדי לא לקבל את ה- warning: למעשה אומרים לקומפיילר להתייחס לכתובת כאל כתובת התחלה של מערך

שליחת כתובת השורה השנייה

שליחת כתובת האיבר השני

# העברת מטריצה לפונקציה המקבלת מערך

```
#define SIZE 3
```

```
void printMatrix(int mat[][SIZE], int rows)
```

```
{
    int i, j;
    for (i=0 ; i < rows ; i++)
    {
        for (j=0 ; j < SIZE; j++)
            printf("%4d", mat[i][j]);
        printf("\n");
    }
}
```

```
int getMax(int* arr, int size)
```

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

```
void printArr(int* arr, int size)
```

```
{
    int i;
    for (i=0 ; i < size ; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

# העברת מטריצה לפונקציה

## המקבלת מערך (2)

```
Matrix:
 1  2  3
 4  5  2
 8  2  3

Matrix as arr:
1 2 3 4 5 2 8 2 3

The max in line #1 is 3
The max in line #2 is 5
The max in line #3 is 8

The max in the matrix is 8
```

```
1. void main()
2. {
3.     int i, mat[SIZE][SIZE]=
4.     { {1,2,3}, {4,5,2}, {8,2,3} };
5.
6.     printf("Matrix:\n");
7.     printMatrix(mat, SIZE);
8.     printf("\nMatrix as arr:\n");
9.     printArr((int*)mat, SIZE*SIZE);
10.    printf("\n");
11.
12.    for ( i=0 ; i < SIZE; i++)
13.        printf("The max in line # %d is %d\n",
14.            i+1,      getMax(mat[i], SIZE));
15.    printf("\nThe max in the matrix is %d\n",
16.        getMax((int*)mat, SIZE*SIZE));
17. }
```

סימון לקומפיילר להתייחס לכתובת  
ככתובת התחלה של מערך

|               |          |             |
|---------------|----------|-------------|
| int[][3]: mat | <b>1</b> | <b>1000</b> |
|               | <b>2</b> | <b>1004</b> |
|               | <b>3</b> | <b>1008</b> |
|               | <b>4</b> | <b>1012</b> |
|               | <b>5</b> | <b>1016</b> |
|               | <b>2</b> | <b>1020</b> |
|               | <b>8</b> | <b>1024</b> |
|               | <b>2</b> | <b>1028</b> |
|               | <b>3</b> | <b>1032</b> |
| t: i          | <b>3</b> | <b>1036</b> |

# העברת מטריצה לפונקציה המקבלת מערך

□ מטריצה היא למעשה מערך דו-מימדי: שורות ועמודות

|       |       |       |       |
|-------|-------|-------|-------|
| [0,0] | [0,0] | [0,0] | [0,0] |
| [0,0] | [0,0] | [0,0] | [0,0] |
| [0,0] | [0,0] | [0,0] | [0,0] |

□ ניתן גם להסתכל עליה כמערך של מערכים, כאשר כל איבר הוא מערך חד-מימדי

□ לכן ניתן לשלוח כל איבר בה (שהוא מערך בפני עצמו) לפונקציה המקבלת מערך חד-מימדי

# מערך של כתובות

- ראינו כי מערך הוא אוסף של איברים מאותו טיפוס
- ראינו שמשתנה המכיל כתובת הוא גם טיפוס
- ניתן להגדיר מערך של כתובות:

```
int* matrix[]
```

- כל איבר במערך הוא כתובת
- כל כתובת כזו יכולה להיות כתובת התחלה של מערך חד-מימדי

- כאשר מעבירים מערך של כתובות לפונקציה נכתוב אותו בהצהרה כך: **int\*\* arr** או כך: **int\* arr[]**

# מערך של מצביעים

## דוגמא (1)

|           |      |      |
|-----------|------|------|
| int** mat | 1036 | 2000 |
| int rows  | 2    | 2004 |
| int cols  | 3    | 2008 |
| int i     | 1    | 2012 |
| int j     | 1    | 2016 |

int\* mat[]

void printMatrix(int\*\* mat, int rows, int cols)

```
{
    int i, j;
    for (i=0 ; i < rows ; i++)
    {
        for (j=0 ; j < cols ; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

כל איבר במערך הוא כתובת

```
void main()
{
```

```
    int arr1[]={1,2,3}, arr2[]={4,5,6}, arr3[]={7,8,9};
    int* mat1[] = {arr1, arr2};
```

```
    printf("matrix 1:\n");
    printMatrix(mat1, 2, 3);
}
```

$mat[i][j] = (*(mat+i)+j)$

דוגמא עבור  $i=j=0$

$*(1036+0)=(*1036) = 1000$

|              |     |         |
|--------------|-----|---------|
| int[]: arr1  | 1   | 1000    |
|              | 2   | 1004    |
|              | 3   | 1008    |
| int[]: arr2  | 4   | 1012    |
|              | 5   | 1016    |
|              | 6   | 1020    |
| int[]: arr3  | 7   | 1024    |
|              | 8   | 1028    |
|              | 9   | 1032    |
| int*: mat1[] | 100 | 1036    |
| int*: mat1[] | 0   | 1040    |
| main -       | 101 | הזיכרון |



# מערך של מצביעים

## דוגמא (2)

|              |                        |             |
|--------------|------------------------|-------------|
| int**: mat   | <b>103</b><br><b>6</b> | <b>2000</b> |
| int: rows    | <b>2</b>               | <b>2004</b> |
| int: cols    | <b>3</b>               | <b>2008</b> |
| int**: pRows | <b>104</b>             | <b>2012</b> |
| int**: pCols | <b>4</b>               | <b>2016</b> |
| int**: pCols | <b>102</b>             | <b>2016</b> |
| int*: pCols  | <b>4</b>               | <b>2016</b> |

```
void printMatrix(int** mat, int rows, int cols)
{
```

```
    int** pRows, *pCols;
```

```
    for ( pRows=mat ; pRows < mat+rows ; pRows++ )
```

```
    {
```

```
        for ( pCols=*pRows ; pCols < *pRows+cols ; pCols++)
```

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

```
            printf("\n");
```

```
        }
```

```
    }
```

```
void main()
```

```
{
```

```
    int arr1[]={1,2,3}, arr2[]={4,5,6}, arr3[]={7,8,9};
```

```
    int* mat1[] = {arr1, arr2};
```

```
    printf("matrix 1:\n");
```

```
    printMatrix(mat1, 2, 3);
```

```
}
```

את הכתובת של האיבר במערך אליו אנו רוצים  
לפנות.  
מאחר והמערך מכיל כתובות ל- int הוא מטיפוס

|              |            |                |
|--------------|------------|----------------|
| int[]: arr1  | <b>1</b>   | <b>1000</b>    |
|              | <b>2</b>   | <b>1004</b>    |
|              | <b>3</b>   | <b>1008</b>    |
| int[]: arr2  | <b>4</b>   | <b>1012</b>    |
|              | <b>5</b>   | <b>1016</b>    |
|              | <b>6</b>   | <b>1020</b>    |
| int[]: arr3  | <b>7</b>   | <b>1024</b>    |
|              | <b>8</b>   | <b>1028</b>    |
|              | <b>9</b>   | <b>1032</b>    |
| int*: mat1[] | <b>100</b> | <b>1036</b>    |
| int*: mat1[] | <b>0</b>   | <b>1036</b>    |
| main -       | <b>101</b> | <b>הזיכרון</b> |

# איתחול מצביע למחרוזת קבועה

ראינו איתחול מחרוזת:

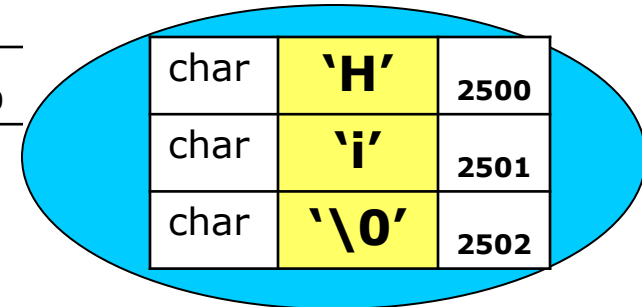
```
char str[] = "Hi";
```

|             |      |      |
|-------------|------|------|
| char[]: str | 'H'  | 1000 |
|             | 'i'  | 1001 |
|             | '\0' | 1002 |

ניתן לאתחל מצביע למחרוזת כך:

```
char* str = "Hi";
```

|            |             |      |
|------------|-------------|------|
| char*: str | <b>2500</b> | 1000 |
|------------|-------------|------|



זיכרון ה- static storage

- str יכול את כתובת ההתחלה של המערך
- כאשר מאתחלים מצביע למחרוזת בצורה זו המערך נשמר בזיכרון הנקרא static storage
- כל פעולה שניתן לבצע על מערך ניתן לבצע על מערך תווים זה (פרט לשינוי ערכי המערך)

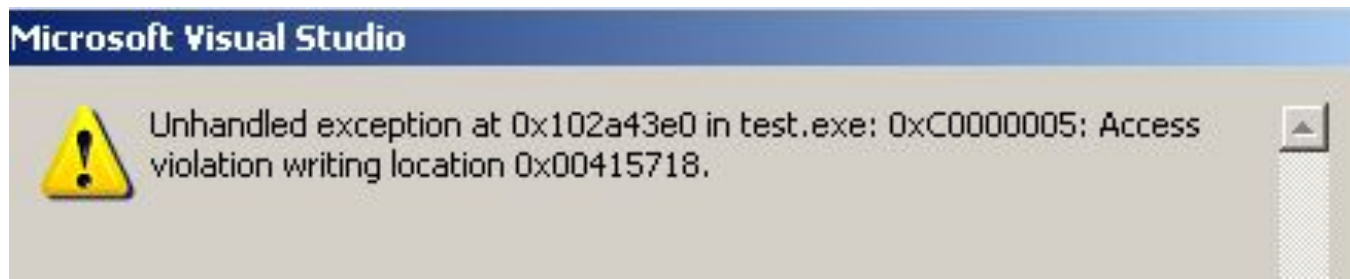
## ה- static storage

□ זהו שטח זיכרון המכיל מחרוזות סטטיות שהוגדרו בזמן קומפילציה, ושלא הוקצה להן שטח זיכרון על ה-heap, כמו בדוגמאת איתחול מצביע למחרוזת:

```
char* str = "Hi";
```

□ זיכרון זה הינו סטטי ולא ניתן לשנות את תוכנו:

```
int main()
{
    char* str = "hhhhh";
    scanf("%s", str);
}
```



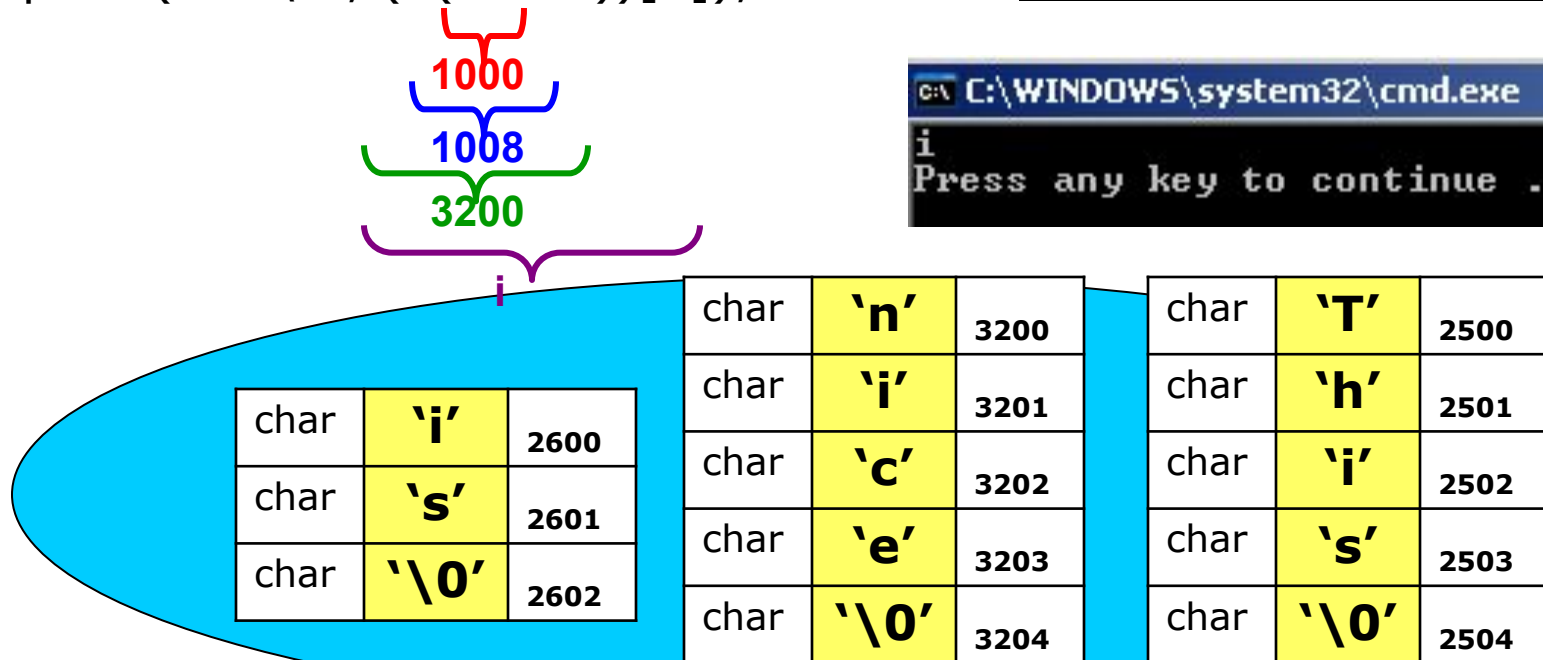
# מערך של מצביעים למחרוזות - דוגמא

```
int main()
{
    char* arr[3] = {"This", "is", "nice"};
    printf("%c\n", (*(arr+2))[1]);
}
```

$(*(arr+2))[1] \equiv (*(arr+2)+1)$

|              |             |      |
|--------------|-------------|------|
| char*[]: arr | <b>2500</b> | 1000 |
|              | <b>2600</b> | 1004 |
|              | <b>3200</b> | 1008 |

```
C:\WINDOWS\system32\cmd.exe
i
Press any key to continue . . .
```



זיכרון ה- static storage

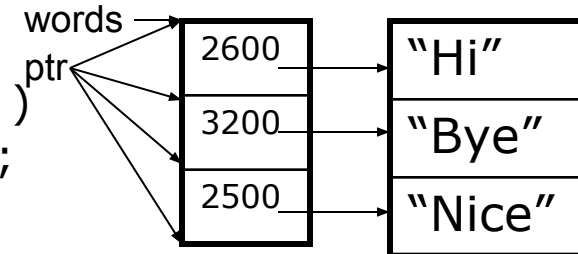
# דוגמא: הדפסת איברי מערך של מחרוזות

```
int main()
{
    char* words[] = {"Hi", "Bye", "Nice"};
    char** ptr;
    int numOfWork = sizeof(words)/sizeof(words[0]);
```

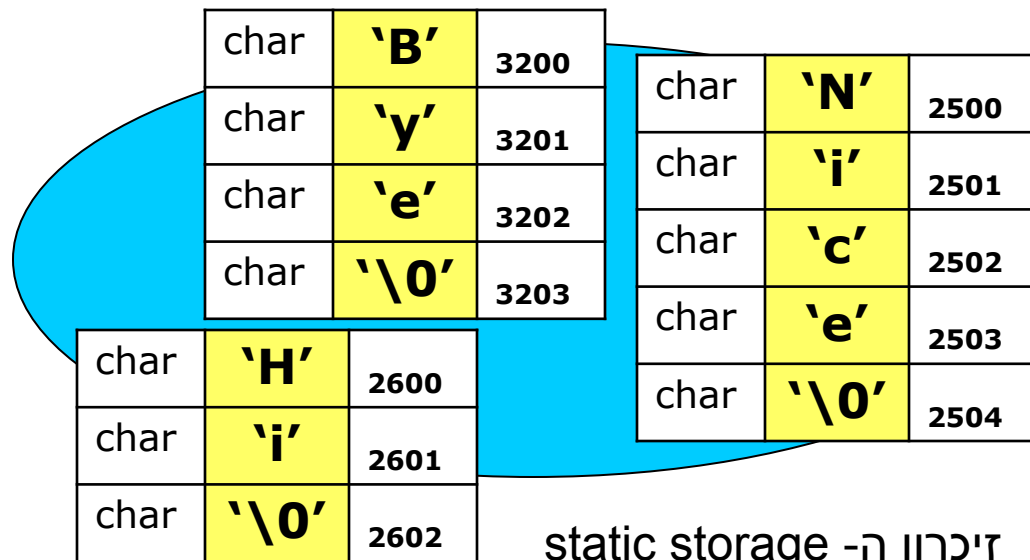
```
    for ( ptr=words ; ptr < words+numOfWork ; ptr++ )
        printf("Day: %s \t2nd letter: %c\n", *ptr, *(*ptr+1));
```

```
}
```

```
C:\WINDOWS\system32\cmd.exe
Day: Hi          2nd letter: i
Day: Bye         2nd letter: y
Day: Nice        2nd letter: i
Press any key to continue . . .
```



|                |             |             |
|----------------|-------------|-------------|
| char*[]: words | <b>2600</b> | <b>1000</b> |
|                | <b>3200</b> | <b>1004</b> |
|                | <b>2500</b> | <b>1008</b> |
| char**: ptr    | <b>1012</b> | <b>1012</b> |
| int: numOfWork | <b>3</b>    | <b>1016</b> |



# פונקציה המשנה מצביע

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

```
    int i;
    min = max = &arr[0];
    for (i=1 ; i < size ; i++)
    {
```

```
        if (arr[i] < *min)
            min = &arr[i];
        if (arr[i] > *max)
            max = &arr[i];
    }
```

```
    printf("Min at %p, Max at %p\n", min, max);
}
```

```
{
```

```
void main()
{
```

```
    int arr[] = {5,6,3};
    int* min=NULL, *max=NULL;
```

```
    getMinMaxAddress(arr, sizeof(arr)/sizeof(arr[0]), min, max);
    printf("Min at %p, Max at %p\n", min, max);
}
```

|           |             |      |
|-----------|-------------|------|
| int*: arr | <b>1000</b> | 2004 |
| int: size | <b>3</b>    | 2008 |
| int*: min | <b>1008</b> | 2012 |
| int*: max | <b>1004</b> | 2016 |

הזיכרון של ה- getMinMaxAddress

|            |             |      |
|------------|-------------|------|
| int[]: arr | <b>5</b>    | 1000 |
|            | <b>6</b>    | 1004 |
|            | <b>3</b>    | 1008 |
| int*: min  | <b>NULL</b> | 1012 |
| int*: max  | <b>NULL</b> | 1016 |

הזיכרון של ה- main

הפונקציה שינתה את ההעתקים של  
הכתובות, לכן צריך להעביר כתובת של

## פונקציה המשנה מצביע (2)

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

```
{
```

```
    int i;
```

```
    *min = *max = &arr[0];
```

```
    for ( i=1 ; i < size ; i++ )
```

```
    {
```

```
        if (arr[i] < **min)
```

```
            *min = &arr[i];
```

```
        if (arr[i] > **max)
```

```
            *max = &arr[i];
```

```
    }
```

```
    printf("Min at %p, Max at %p\n", *min, *max);
```

```
}
```

```
void main()
```

```
{
```

```
    int arr[] = {5,6,3};
```

```
    int* min=NULL, *max=NULL;
```

```
    getMinMaxAddress(arr, sizeof(arr)/sizeof(arr[0]), &min, &max);
```

```
    printf("Min at %p, Max at %p\n", min, max);
```

```
{
```

|            |             |      |
|------------|-------------|------|
| int*: arr  | <b>1000</b> | 2004 |
| int: size  | <b>3</b>    | 2008 |
| int**: min | <b>1012</b> | 2012 |
| int**: max | <b>1016</b> | 2016 |
| int: i     | <b>3</b>    | 2020 |

הזיכרון של ה- getMinMaxAddress

|            |             |      |
|------------|-------------|------|
| int[]: arr | <b>5</b>    | 1000 |
|            | <b>6</b>    | 1004 |
|            | <b>3</b>    | 1008 |
| int*: min  | <b>1008</b> | 1012 |
| int*: max  | <b>1004</b> | 1016 |

הזיכרון של ה- main

# הפונקציה strtok

---

`char* strtok(char*, const char*)`

□ הפונקציה מפרקת מחרוזת ל- token'ים עפ"י תווי הפרדה

■ למשל: המחרוזת "AKA: as know as" ותווי ההפרדה ' ' ו- ':' יוחזרו ה- token'ים הבאים: AKA as known as



# הפונקציה strtok (2)

`char* strtok(char*, const char*)`

הפונקציה מקבלת מחרוזת `text` ומחרוזת נוספת המכילה תווי הפרדה, `delimiters`, ומבצעת את הדברים הבאים:

- מחליפה את המופע הראשון ב- `text` המכיל את אחד מהתווים שב- `delimiters` ב- `'\0'`
  - המחרוזת `text` אינה יכולה להיות `const`
- מחזירה את הכתובת של תחילת ה- `token`
- כדי לקרוא ל- `strtok` מהמקום בו הפסיקה בפעם הקודמת יש לקרוא לה עם `NULL`
- מחזירה `NULL` כאשר מגיעה לסוף המחרוזת

```
The words in the sentence:
Hello
World!
The orig sentence: Hello
Press any key to continue . . .
```

# אגמא – strtok

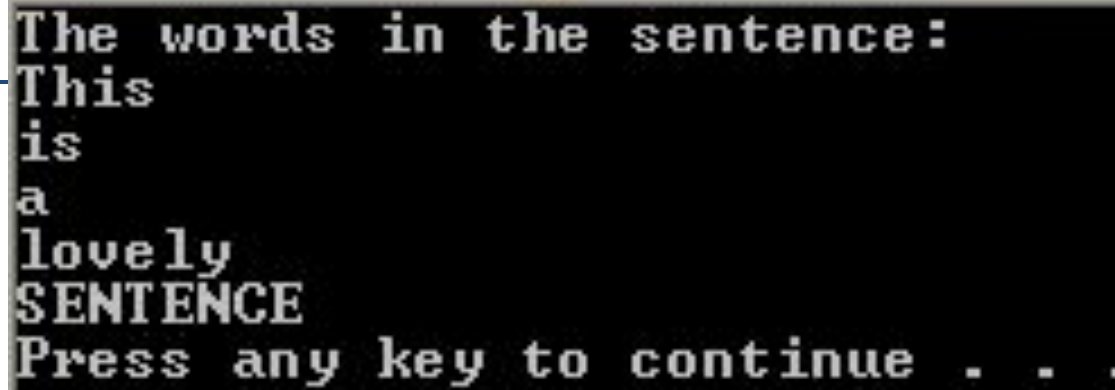
```
1. void main()
2. {
3.     char str[] = "Hello World!";
4.     char* delimiters = " :-,";
5.     char* words;

6.     printf("The words in the sentence:\n");
7.     words = strtok(str, delimiters);
8.     while (words != NULL)
9.     {
10.        printf("%s\n", words);
11.        words = strtok(NULL, delimiters);
12.    }
13.    printf("The orig sentence: %s\n", str);
14. {
```

| 1012 | 1011 | 1010 | 1009 | 1008 | 1007 | 1006 | 1005 | 1004 | 1003 | 1002 | 1001 | 1000 |
|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0\   | !    | d    | l    | r    | o    | W    | 0\   | o    | l    | l    | e    | H    |

str words

# דוגמא נוספת – strtok



```
The words in the sentence:
This
is
a
lovely
SENTENCE
Press any key to continue . . .
```

```
1. void main()
2. {
3.     char str[] = "This is: a lovely - SENTENCE";
4.     char* delimiters = " :-,";
5.     char* words;
6.
7.     printf("The words in the sentence:\n");
8.     words = strtok(str, delimiters);
9.     while (words != NULL)
10.    {
11.        printf("%s\n", words);
12.        words = strtok(NULL, delimiters);
13.    }
```

# הפונקציה memcpy

---

```
void* memcpy(void* destination,  
             const void* source,  
             int num );
```

□ מקבלת כתובת כלשהי destination ומעתיקה לתוכה את התוכן שנמצא החל מכתובת source באורך num בתים

□ מחזירה את destination

□ source מועברת כ- const כי לא משנים אותה

□ שימוש אפשרי: העתקת מערכים

# הפונקציה memcpy - דוגמא

```
void printArray(int arr[], int size)
{
```

```
    int i;
    for (i=0 ; i < size ; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

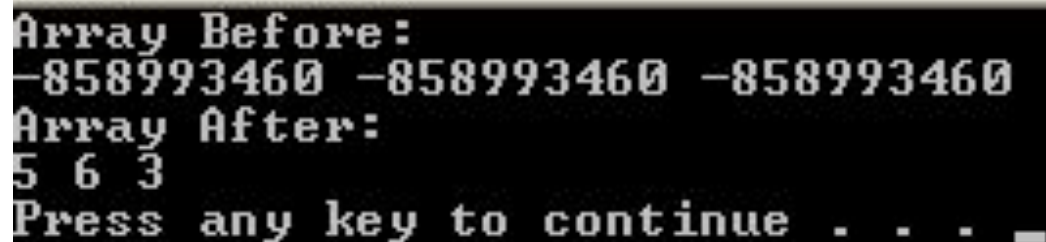
```
void main()
{
```

```
    int arr1[] = {5,6,3}, arr2[3];
```

```
    printf("Array Before:\n");
    printArray(arr2, 3);
```

```
    memcpy(arr2, arr1, 3*sizeof(int));
    printf("Array After:\n");
    printArray(arr2, 3);
```

```
{
```



```
Array Before:
-858993460 -858993460 -858993460
Array After:
5 6 3
Press any key to continue . . . _
```

# ביחידה זו למדנו:

---

- הקשר בין מערך למצביע
- פעולות חיבור וחיסור עם מצביעים
- מצביע מטייל על מערך
- הפונקציות `strchr` ו-`strstr`
- העברת מערך לפונקציה
- הבעייתיות בהחזרת מערך מפונקציה
- מערך של מצביעים
- איתחול מצביע למחרוזת קבועה
- העברת מצביע לפונקציה לצורך שינוי הצבעתו
- הפונקציה `strtok`
- הפונקציה `memcpy`

# תרגיל 1:

## □ כתוב תוכנית:

- הגדר מערך של מחרוזות
- עבור כל מחרוזת יש לדאוג שהאות הראשונה תהיה גדולה
- הדפס את מערך המחרוזות המעודכן

## □ דוגמא:

- עבור המערך "Kuku", "abc", "good" הפונקציה תשנה אותו להיות "Kuku", "Abc", "Good"
- הגבלה: אין להשתמש ב- [], אלא רק במצביעים

## תרגיל 2:

---

1. כתוב פונקציה המקבלת מערך של מספרים עשרוניים וגודלו. הפונקציה תקלוט ערכים לתוך המערך מהמשתמש

■ הגבלה: הפונקציה תשתמש אך ורק במצביעים, ולא ב- []

2. כתוב פונקציה המקבלת מערך של מספרים עשרוניים וגודלו ומדפיסה את איבריו מהסוף להתחלה

■ הגבלה: הפונקציה תשתמש אך ורק במצביעים, ולא ב- []



## תרגיל 2 (המשך):

□ כתוב main:

- הגדר מערך של 7 מספרים עשרוניים
- קלוט לתוכו ערכים באמצעות הפונקציה מסעיף 1
- קרא לפונקציה מסעיף 2 עבור 3 האיברים האחרונים המערך
- קרא לפונקציה מסעיף 2 עבור כל האיברים פרט לראשון ולאחרון

□ שימו לב: הקוד ב- main צריך להיות כללי כך שאם נשנה את גודל המערך, לא נצטרך לעשות שינויים נוספים ב- main

## תרגיל 3:

---

□ בתרגיל זה יש להשתמש ב-2 הפונקציות מתרגיל 2

□ כתוב main:

- הגדר מטריצה של מספרים בגודל 5X4
- שלח כל שורה במטריצה לפונקציה הקוראת נתונים
- שלח את חצי האיברים הראשונים לפונקציה השניה, ואח"כ את חצי האיברים השניים

## תרגיל 3 (המשך):

למשל: עבור המטריצה הבאה בגודל  $5 \times 4$  עם ה-20 הערכים הבאים (שנקראו ע"י הפונקציה ראשונה)

4 3 2 1

8 7 6 5

2 1 0 9

6 5 4 3

0 9 8 7

עליך להדפיס את חצי מכמות האיברים הראשונים (בצהוב) מהסוף להתחלה, ואח"כ תדפיס את חצי מכמות האיברים האחרונים (בירוק) מהסוף להתחלה:

7,8,9,0,3,4,5,6,9,0 , 1,2,5,6,7,8,1,2,3,4