

שגיאות עיגול וקיצוץ

```
# include <math.h>
```

```
a = (x != 0) ? (exp(x)-1)/ x : 1;
```

```
a = (f abs(x) < 1.e-5) ? (exp(x)-1)/ x : 1;
```

```
a = (f abs(x) < 1.e-5) ? (exp(x)-1)/ x : 1+x/ 2;
```

$\exp(1.e-5)=1.0000101$

$a=1.01$

$\sim 10^{-7}, + / -$ עיגול:

$x^2/6$ קיצוץ:

שגיאות עיגול וקיצוץ

```
a = (f abs(x) < 1.e-2) ? (exp(x)-1)/ x : 1+x/ 2;
```

float

$$10^{-7}/10^{-2} = 10^{-5} \quad \text{עיגול:}$$

$$(10^{-2})^2/6 \approx 10^{-5} \quad \text{קיצוץ:}$$

```
a = (f abs(x) < 1.e-5) ? (exp(x)-1)/ x : 1+x/ 2;
```

double

$$10^{-16}/10^{-5} = 10^{-11} \quad \text{עיגול:}$$

$$(10^{-5})^2/6 \approx 10^{-11} \quad \text{קיצוץ:}$$

Array - מערך (שום דבר פוליטי...)

5

int x

▪ משתנה בודד

▪ ולעומתו מערך של משתנים

array of
integers

12	5	66	4
----	---	----	---

0

1

2

3

בניית מערך

- דברים שיש לתכנן מראש:
 - סוג המשתנים: char, double, float, int... רק סוג אחד לכל מערך
 - גודל המערך – מספר האלמנטים חייב להיות מוגדר מראש, בזמן הקומפילציה
 - יודעים מראש (למשל מערך בן 31 אלמנטים לימים (בחודש)
 - מגזימים (מערך של סטודנטים שנה א' – בן 500 (איברים)
 - איננו יודעים מה מספר המשתנים אבל רוצים גמישות ודינאמיות... מערך הוא לא הכלי לכך...

הגדרת מערך

```
elementType arrayName [size];
```

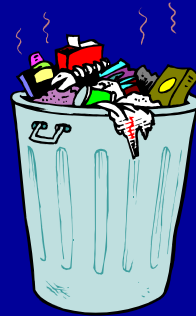
- elementType : סוג המשתנים
- arrayName : שם המערך
- size : מספר האיברים

```
int number[10];
```

```
double oilprice[31];
```

איתחול

- כמה אפשרויות
- אבל לפני כן זיכרו: משתנים רגילים שערכם לא מוגדר יכילו



...

- מערך לא מוגדר



איתחול בזמן ההגדרה

- אפשרות ראשונה:

```
int temp[5] = {45, 47, 44, 56, 49};
```

- אפשרות שנייה:

```
int temp[] = {45, 47, 44, 56};
```

- כאן יבנה מערך בן 4 איברים

- אפשרות שלישית:

```
int temp[5] = {0};
```

- כל האיברים במערך יקבלו את הערך הראשוני אפס.

שימוש באיברי המערך

- לקרוא את ערכו של משתנה מסוים:
`var_name[index];`
- זכרו כי מערך בן N איברים מתחיל באיבר 0 ומסתיים באיבר $N-1$.

דוגמא:

```
int temp[5] = {45, 47, 44, 56, 49};
```

- להדפסת האיבר האפס ("הראשון") נשתמש ב:
`printf ("%d", temp[0]);`
- להדפסת האיבר האחרון
`printf ("%d", temp[4]);`

מערכים בזכרון

```
int temp[30];
```

2 bytes/int * 30 integers = 60 bytes

- C לא בודק את גבולות המערכים.

- **דוגמא:**

```
int temp[5] = {45, 47, 44, 56, 49};  
printf ("%d", temp[5]);
```

- זה עובר קומפילציה אבל מכניס באג.

אמא

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

```
main() {
```

```
    int numbers[100],i;
```

```
    for (i=0; i<100; i++)
```

```
        numbers[i] = 10;
```

```
    --numbers[2];
```

```
    printf("The 3rd element is %d\n", numbers[2]);
```

```
}
```

```
--(numbers[2]);
```

```
numbers[2] = numbers[2]-1;
```

9

מערכים רב ממדיים

```
int mult_arr[4][7] =  
{1,2,5,6,9,3,4},  
{5,6,7,8,9,-10,5},  
{1,2,5,6,9,3,4},  
{5,6,7,8,9,-10,5}  
};
```

1	2	5	6	9	3	4
5	6	7	8	9	-10	5
1	2	5	6	9	3	4
5	6	7	8	9	-10	5

למעשה מערך בן 28 אברים שורה שורה.

לולאה פנימית על האינדקס הימני:

```
f or (i=0; i<4; i++)  
  f or (j=0; j<7; j++)  
    printf ("%i\n",3*mult_arr[i][j]);
```

פונקציות

```
type name(argument s)
{
    declarations;
    statements;
}
```

```
int sum(int a, int b)
{
    int result;
    result = a + b;
    return result;
}
```

מקומי
local



type: int (default)
void (procedure)

```
void positive(int a)
{
    if (a > 100)
    {
        printf ("large\n");
        return;
    }
    if (a > 0)
        printf ("positive\n");
}
```

פונקציות

```
type name(argument s)
{
    declar at ions;
    st at ement s;
}
```

```
int sum(int a, int b)
{
    int result;
    result = a + b;
    return result;
}
```

מקומי
local

type: int (default)
void (procedure)

```
void positive(int a)
{
    if (a > 100)
    {
        printf ("large\n");
        return;
    }
    if (a > 0)
        printf ("positive\n");
}
```

שימוש בפונקציות

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

```
int sum(int a, int b);
```

```
int sum(int, int);
```



```
main()
```

```
{
```

```
    printf ("1 + 1 = %d\n", sum(1,1));
```

```
}
```

```
int sum(int a, int b)
```

```
{
```

```
    int result;
```

```
    result = a + b;
```

```
    return result;
```

```
}
```