

פונקציות



קרון כליף

ביחידה זו נלמד:

- מהי פונקציה, פרמטרים וערך מוחזר
- כיצד נראה הזיכרון
- ספריות של פונקציות
- מערכים כפרמטר לפונקציה
- יצירת מספרים אקראיים
- פונקציות מ- `math.h`
- תהליך הפיכת תוכנית ב- C לשפת מכונה
- סוגי משתנים וטווח הכרתם: לוקאליים, גלובליים, סטטיים

מהי פונקציה

- פונקציה היא אוסף הוראות לביצוע שמממשות רעיון משותף
- ראינו פונקציות בעבר: `printf`, `scanf`, `sizeof`, ועוד
 - גם ה- `main` שאנחנו כותבים הוא פונקציה
- עבורנו פונקציות אלו הן "קופסא שחורה" שקיבלנו מספריה מסוימת, ואנו רק יודעים מה הן עושות, אבל לא יודעים איך, כלומר מהו אוסף הפעולות המבוצעות

בשיעור זה נלמד לכתוב פונקציות
בעצמנו!

דוגמא לשימוש בפונקציה

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

```
void main()
```

```
{
```

```
    int num, size;
```

```
    size = sizeof(num);
```

```
    printf("the variable 'num' uses %d bytes\n", size);
```

```
}
```

הערך המוחזר מהפונקציה

שם הפונקציה

הנתונים המועברים. כל נתון נקרא ארגומנט

```
the variable 'num' uses 4 bytes
Press any key to continue . . .
```

דוגמא לשכפול קוד: חישוב חלוקת log

```
#define BASIS 10
void main()
{
    int num1=100, num2=1000, res;
    int log1, log2;

    res = 1;
    for (log1=0 ; res < num1 ; log1++)
        res *= BASIS;
    printf("log(%d) = %d\n", num1, log1);

    res = 1;
    for (log2=0 ; res < num2 ; log2++)
        res *= BASIS;
    printf("log(%d) = %d\n", num2, log2);

    printf("log(%d/%d) = %d\n", num1, num2, log1-log2);
}
```

$$\log_c \left(\frac{a}{b} \right) = \log_c(a) - \log_c(b)$$

חישוב $\log_{10} 100$

רצף הפעולות לחישוב לוג זהה, רק כל פעם ביצענו את הפעולות על ערכים שונים □ שכפול קוד! מי שקורא את ה- main צריך להבין מה תפקידה של כל לולאה...

חישוב $\log_{10} 1000$

```
log(100) = 2
log(1000) = 3
log(100/1000) = -1
```

הדוגמא ללא שכפול קוד, קריאות טובה ומודלריות

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

שם הפונקציה.
יעיד מה היא עושה

```
#define BASIS 10
```

טיפוס המידע
שהפונקציה מחזירה

```
int log10(int num)  
{
```

הנתון שהפונקציה צריכה לקבל

```
    int res = 1, i;  
    for (i=0 ; res < num ; i++)  
        res *= BASIS;  
    return i;
```

איך הפונקציה
מבצעת את העבודה

עם השימוש בפונקציה הקוד קצר
יותר וקריא יותר!
ה- **main** כתוב ב"ראשי פרקים"
וניתן להבין בקלות מה הוא מבצע

```
{
```

הערך שהפונקציה מחזירה

```
void main()  
{
```

שימוש בערך שהפונקציה מחזירה

```
    int num1=100, num2=1000, res;  
    res = log10(num1) - log10(num2);  
    printf("log(%d/%d) = %d\n", num1, num2, res);
```

```
{
```

קריאה לפונקציה שיודעת לחשב
,log10

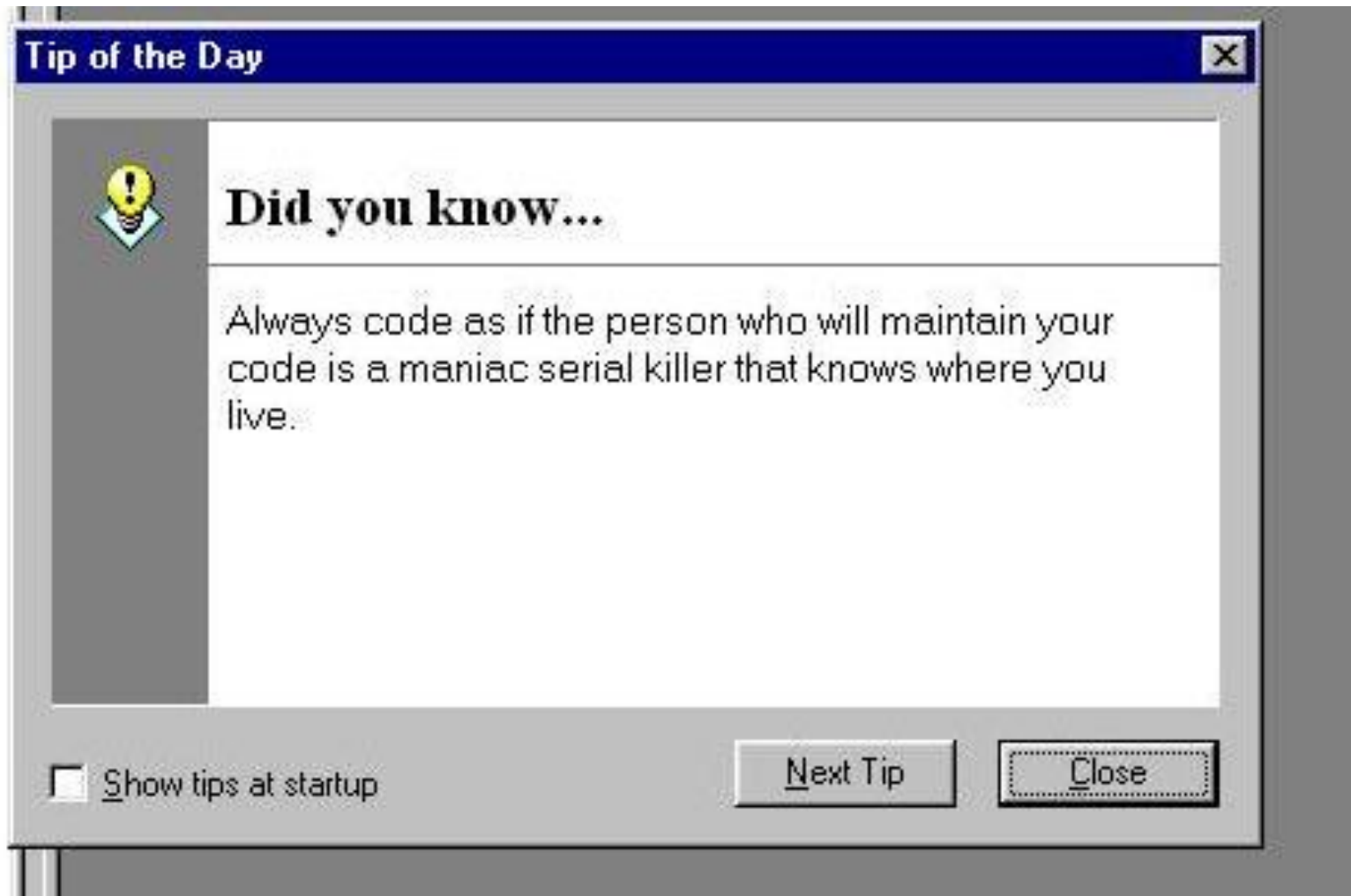
פונקציות - מוטיבציה

□ **מודולריות:** חלוקת התוכנית לקטעי קוד יותר קטנים, כאשר כל קטע עושה משהו נקודתי

□ **יתרונות:**

- קוד קריא יותר
- יותר קל לבדוק את התוכנית
- חלוקת עבודה
- שימוש חוזר בקוד
- חיסכון בבדיקות
- פחות שגיאות

תזכרו!



החלקים בשילוב פונקציה שלנו בקוד

1. הצהרה על הפונקציה

- כאשר כותבים פונקציה ראשית צריך להחליט **מה** הפונקציה עושה
- לאחר שהחלטנו על ה"מה", יש להחליט אילו נתונים הפונקציה צריכה **לקבל** כדי לבצע את העבודה
- לבסוף נגדיר מה הטיפוס שהפונקציה **מחזירה** לנו

2. שימוש בפונקציה

3. מימוש הפונקציה

איך כותבים הצהרה של פונקציה

דוגמא: פונקציה המחשבת חזקה

- מה הפונקציה עושה: הפונקציה מחשבת חזקה של שני מספרים
- מה הפונקציה מקבלת: 2 מספרים, הראשון בסיס והשני מעריך
- מה הפונקציה מחזירה: את תוצאת חישוב הבסיס במעריך
- לבחור שם משמעותי לפונקציה

base exponent
power
int ←
int base
int exponent

תחביר:

`int power(int base, int exponent)`
<returned value type> <function_name> (<parameters list>)

וראופו כללי

בחירת שם לפונקציה

□ שם הפונקציה צריך להעיד מה היא עושה
■ למשל: `power`, `printf`

□ מבחינת סגנון כתיבה, ההמלצה היא:

- שם הפונקציה יתחיל באות קטנה
- אם יש יותר ממילה אחת בשם הפונקציה כל מילה נוספת תתחיל באות גדולה
- למשל: `calcRectangleArea`
- ניתן לאמץ סגנון אחר, אבל הקפידו להיות עקביים בסגנון לאורך כל התוכנית. כנ"ל גם לגבי שמות משתנים

הצהרת פונקציה - הערך המוחזר (1)

□ הערך המוחזר יכול להיות כל אחד מהטיפוסים שאנחנו מכירים

■ למשל, פונקציה המקבלת אורך צלע ריבוע ומחזירה את שטחו
int calcSquareArea(int length);

■ למשל, פונקציה המקבלת מספר, ומחזירה את השורש שלו
double sqrt(int num);

הצהרת פונקציה - הערך המוחזר void

□ אם הפונקציה אינה מחזירה ערך, נרשום כערך המוחזר void

- החזרה = ניתן לאכסן ערך במשתנה כלשהו
- שימו לב: הדפסה למסך אינה החזרה של ערך!

■ למשל, פונקציה המדפיסה למסך
`void printRectangle(int base)`

הצהרת פונקציה – הפרמטרים המועברים

□ רשימת הנתונים אשר הפונקציה מקבלת מופרדת ע"י פסיקים

■ דוגמא: פונקציה המקבלת אורך ורוחב של מלבן ומחזירה את שטחו

```
int calcRectangleArea(int height, int width);
```

■ למשל, פונקציה המקבלת מספר n ומחזירה $n!$

```
int factorial(int num);
```

□ אם פונקציה אינה מקבלת נתונים, רשימת הפרמטרים תהייה ריקה

■ למשל, פונקציה הקוראת מהמשתמש את גילו ומחזירה אותו

```
int readAge();
```

מימוש פונקציה

□ דוגמא למימוש פונקציה המקבלת בסיס ומעריך ,
ומחזירה את החזקה

```
int power(int a, int b)
{
    int i, result=1;
    for (i=0 ; i < b ; i++)
        result *= a;
    return result;
}
```

הפקודה return מציינת הפסקת ביצוע
הפקודות בפונקציה והחזרת ערך המשתנה

פונקציה שהערך המוחזר שלה הוא void
אינה חייבת להכיל את הפקודה return.
באם מופיעה הפקודה return בפונקציה
המחזירה void, רצף הפעולות יפסק מיד.

אבל זהירות!

**How
people
run**



**How
programmer
run**

```
public void run()  
{  
    step++;  
}
```

http://funnypictures4u.files.wordpress.com/2013/06/people_vs_programmers.jpg

תוכנית שלמה עם פונקציה

```
int power(int a , int b )
{
    int i, result=1;
    for (i=0 ; i < b ; i++)
        result *= a;
    return result;
}
```

הצהרה+מימוש

הפונקציה מוכרת ממקום הגדרתה ומטוה, הצהרת הפונקציה תופיע לפני השימוש!

יש להעביר לפונקציה משתנים כמספר הפרמטרים שהיא דורשת, מאותו הסוג ובאותו הסדר!

```
void main()
{
    int base, exponent, result;
```

```
    printf("Please enter base and exponent: ");
    scanf("%d %d", &base, &exponent);
```

איחסון הערך המוחזר מהפונקציה

```
    result = power(base , exponent );
    printf("%d^%d=%d\n", base, exponent, result);
```

שימוש

```
}
```

C:\WINDOWS\system32\cmd.exe

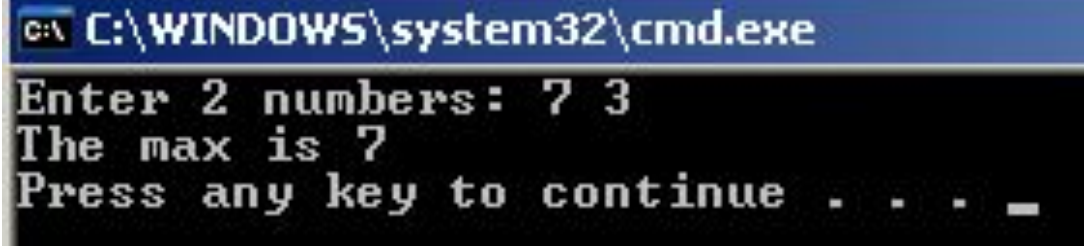
```
Please enter base and exponent: 2 3
2^3=8
Press any key to continue . . . _
```

דוגמא: תוכנית שלמה עם הפונקציה max

```
int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

```
void main()
{
    int num1, num2, maximum;

    printf("Enter 2 numbers: ");
    scanf("%d %d", &num1, &num2);
    maximum = max(num1, num2);
    printf("The max is %d\n", maximum);
}
```



C:\WINDOWS\system32\cmd.exe

Enter 2 numbers: 7 3

The max is 7

Press any key to continue . . . _

בשורה זו מוערך הביטוי מימין וערכו נכנס לתוך המשתנה **maximum**.
לא חייבים לאחסן את הערך המוחזר!

ואפשר גם כך..

```
int max (int x, int y)
{
    if (x > y)
        return x;
    return y; // without the 'else'...
}
```

} **return x > y ? x : y;**

```
void main()
{
    int num1, num2, maximum;

    printf("Enter 2 numbers: ");
    scanf("%d %d", &num1, &num2);
    maximum = max(num1, num2);
    printf("The max is %d\n", maximum);
}
```

מחסנית הקריאות

□ כאשר קוראים לפונקציה, למעשה "קופצים" לאוסף הפקודות שלה, ושומרים את מקום החזרה

□ כאשר נתקלים בפקודה return בתוך גוף הפונקציה, חוזרים לביצוע הפעולות מהמקום בו קראנו לה

■ בפונקציה שהטיפוס המוחזר שלה הוא void יתכן ולא תהייה הפקודה return

□ במקרה זה, לאחר סיום הפונקציה, חוזרים לביצוע הפעולות מהמקום בו קראנו לפונקציה

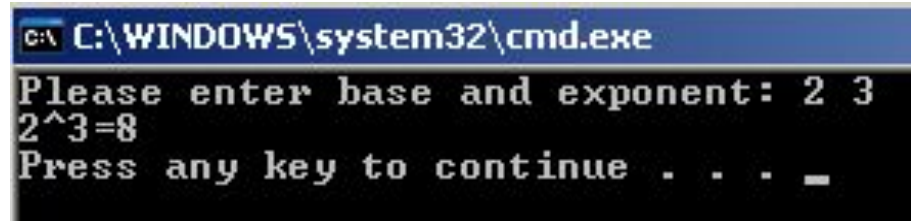
תוכנית שלמה עם פונקציה - הרצה

```
int power(int a, int b)
{
    int i, result=1;
    for (i=0 ; i < b ; i++)
        result *= a;
    return result;
}
```

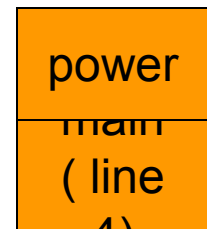
```
void main()
{
    int base, exponent, result;

    printf("Please enter base and exponent: ");
    scanf("%d %d", &base, &exponent);

    result = power(base, exponent);
    printf("%d^%d=%d\n", base, exponent, result);
}
```



C:\WINDOWS\system32\cmd.exe
Please enter base and exponent: 2 3
2^3=8
Press any key to continue . . . _



מחסנית הקריאות

סדר ביצוע פעולות

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

```
int power(int a, int b)
```

```
{
```

```
    printf("debuging.. in 'power'\n");
```

```
    int i, result=1;
```

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

```
        result *= a;
```

```
    return result;
```

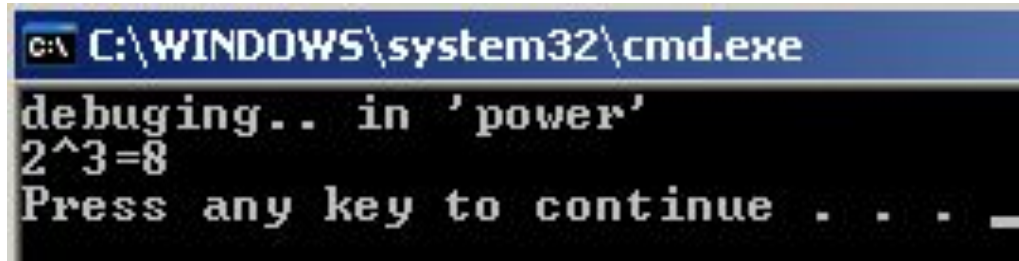
```
}
```

```
void main()
```

```
{
```

```
    printf("2^3=%d\n", power(2, 3));
```

```
}
```



קודם תבוצע הפונקציה

סדר ביצוע פעולות (2)

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

```
int power(int a, int b)
```

```
{
```

```
    printf("debuging.. in 'power': a=%d b=%d\n", a, b);
```

```
    int i, result=1;
```

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

```
        result *= a;
```

```
    return result;
```

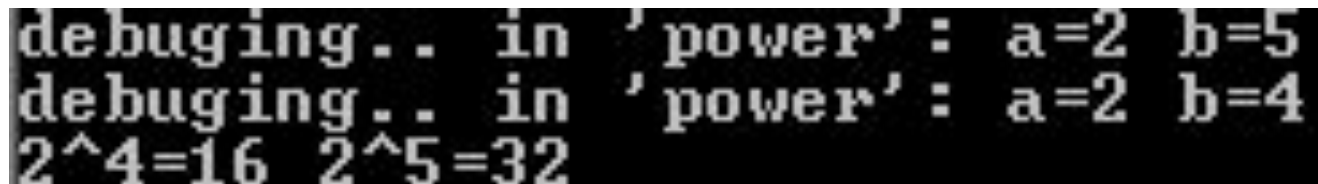
```
}
```

```
void main()
```

```
{
```

```
    printf("2^4=%d, 2^5=%d\n", power(2, 4), power(2, 5));
```

```
}
```



```
debuging.. in 'power': a=2 b=5
debuging.. in 'power': a=2 b=4
2^4=16 2^5=32
```

ביצוע הפונקציות מתבצע מימין לשמאל

תוכנית שלמה עם פונקציה - הזיכרון

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

```
int power(int a, int b)
{
    int i, result=1;
    for (i=0 ; i < b ; i++)
        result *= a;
    return result;
}
```

```
void main()
{
    int base, exponent, result;

    printf("Please enter base and exponent: ");
    scanf("%d %d", &base, &exponent);

    result = power(base, exponent);
    printf("%d^%d=%d\n", base, exponent, result);
}
```

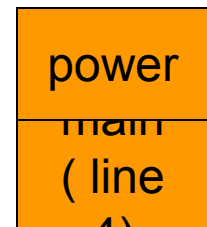
int:base	2	1000
int: exponent	3	1004
int: result	8	1008

הזיכרון של ה- main

המשתנה result שב-main וזה
שבפונקציה אינם אותו משתנה,
ואין הכרח שהם יהיו עם שם זהה!

	...	2008
int: result	8	2012

הזיכרון של power



מחסנית הקריאות

כיצד נראה הזיכרון - סיכום

- ראינו את הזיכרון של ה- `main`
 - תזכורת: ה- `main` הוא פונקציה
- לכל פונקציה יש שטח נפרד בזיכרון ה- `stack` בו מוגדרים המשתנים
- פונקציה יכולה לגשת לשטח הזיכרון שלה בלבד, ולכן אינה יכולה לגשת למשתנים שהוגדרו בפונקציות אחרות
 - הפרמטרים המועברים לפונקציה הם גם משתנים של הפונקציה
 - הפרמטרים המועברים לפונקציה הם **העתקים** של המשתנים שנשלחו, ולכן העברת פרמטרים כזו נקראת **by value**
- כאשר יוצאים מפונקציה, התאים במחסנית שבהם נשמרו ערכיה נמחקים, ולכן קריאה נוספת לפונקציה תייצר אותם מחדש

הפרדה בין הצהרות למימוש

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

```
// prototypes
```

```
int power(int base, int exponent);
```

```
void main()
```

```
{  
    int base, exponent, result;  
  
    printf("Please enter base and exponent: ");  
    scanf("%d %d", &base, &exponent);  
  
    result = power(base, exponent);  
    printf("%d^%d=%d\n", base, exponent, result);  
}
```

```
int power(int base, int exponent)
```

```
{  
    int i, result=1;  
    for (i=0 ; i < exponent ; i++)  
        result *= base;  
    return result;  
}
```

בשורת ההצהרה לא חייבים
לציין את שמות המשתנים

• את ההצהרות נרשום לפני ה- `main`, ובסיום כל הצהרה ;

• חלק זה נקרא `prototype` (אב-טיפוס)

• את המימושים נכתוב מתחת ל- `main`

• ההפרדה היא מאחר ומעניין אותנו איזה פונקציות יש בתוכנית, ופחות איך הן מבצעות את העבודה

• חתימת הפונקציה בהגדרה בראש הקובץ ובמימוש חייבות

תכנון TOP-DOWN

- כאשר ניגשים לכתוב תוכנית, יש לחשוב מהי הבעיה הגדולה שעלינו לפתור ונחלק אותה לחלקים
- לכל חלק "קשה" נתייחס כאל "קופסא שחורה" (שתמומש בהמשך)
- כאשר נממש כל חלק "קשה" אנו מתמודדים עם בעיה יותר קטנה שננסה גם אותה לפרק לבעיות יותר קטנות, עד אשר נגדיר בעיות מספיק קטנות

תכנות TOP-DOWN

דוגמא: ציור משולש

```
C:\WINDOWS\system32\cmd.exe
Enter base and char: 3 $
$
$$
$$$
Press any key to continue .
```

#include <stdio.h>

// prototypes

void printLine(int, char);
void printTriangle(int, char);

void main()
{

int base;
char ch;

printf("Enter base and char: ");
scanf("%d %c", &base, &ch);
printTriangle(base, ch);

}

int: base	3	3000
char: ch	'\$'	3004

הזיכרון של ה- main

int: length	3	2000
char: ch	'\$'	2004
int: i	4	2005

הזיכרון של printLine

int: base	3	1000
char: ch	'\$'	1004
int: i	4	1005

הזיכרון של printTriangle

```
void printLine(int length, char c)
{
    int i;
    for ( i=1; i<=length; i++ )
        printf("%c", c);
    printf("\n");
}
```

```
void printTriangle(int base, char ch)
{
```

```
    int i;
    for( i=1; i<=base ; i++)
        printLine(i, ch);
```

print Line
print Triangle (line 3)
main (line 5)

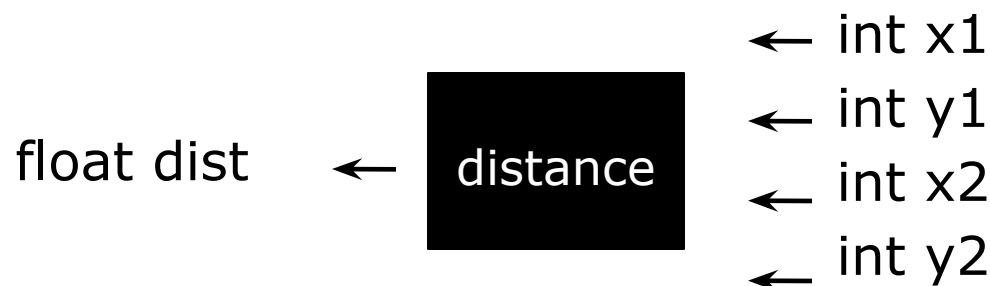
מחסנית הקריאות

דוגמאת סיכום: חישוב מרחק בין 2 נקודות

הנוסחא לחישוב מרחק בין 2 הנקודות (x_1, y_1) ו- (x_2, y_2) :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

נרצה לכתוב פונקציה המקבלת 4 קואורדינטות המייצגות 2 נקודות, ומחזירה את המרחק בניהן:



לכן ההצהרה של הפונקציה תראה כך:

```
float distance(int x1, int y1, int x2, int y2);
```

דוגמא: חישוב מרחק בין 2 נקודות (2)

□ הנוסחא לחישוב מרחק בין 2 הנקודות (x_1, y_1) ו- (x_2, y_2) :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

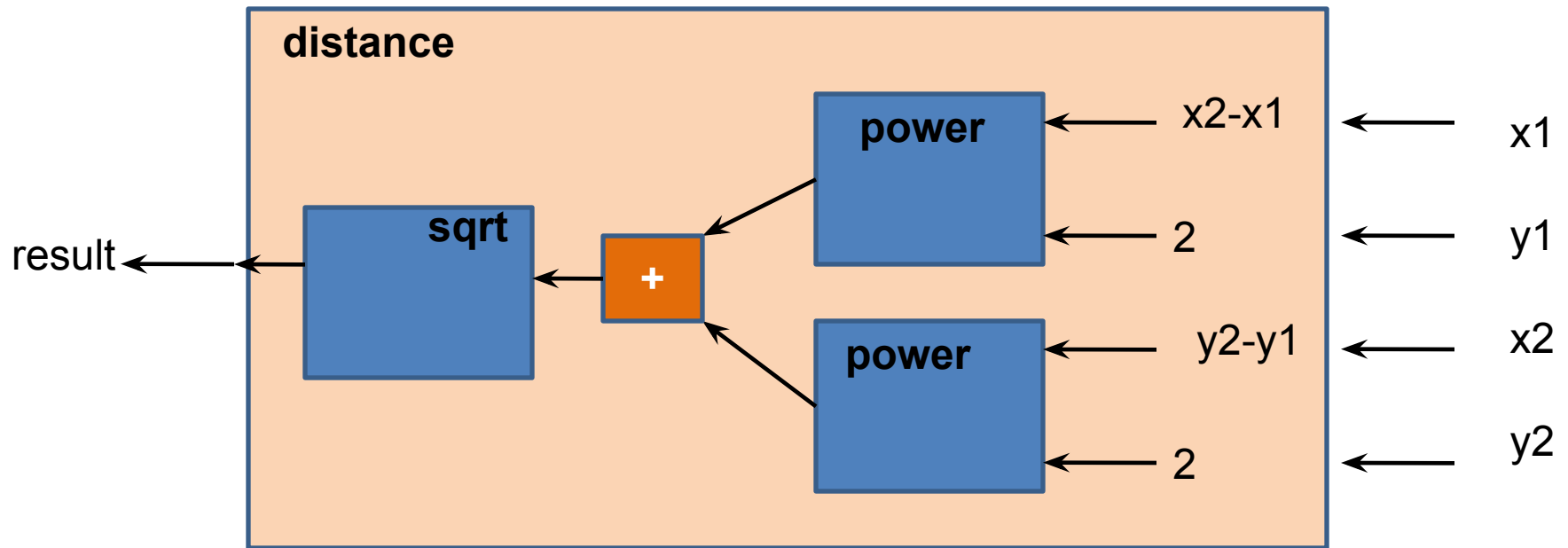
□ ניתן לראות שכדי לחשב מרחק בין שתי נקודות צריך לדעת לחשב חזקה ושורש, לכן נגדיר ונשתמש ב- 2 פונקציות עזר:

- `sqrt` המקבלת מספר `float` ומחזירה את השורש שלו. פונקציה זו כבר קיימת בספריה `math.h`
- `power` שאותה נכתוב בעצמנו לצורך התרגול (למרות שהיא גם קיימת ב- `math.h`)

```
C:\WINDOWS\system32\cmd.exe
Please 2 points (x1, y1, x2, y2): 0 0 4 4
The distance is 5.656854
Press any key to continue . . .
```

דוגמא: חישוב מרחק בין 2 נקודות – תרשים הפונקציה

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



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

הצהרות על
פונקציות שנמש

```
int power(int base, int exponent);
float distance(int x1, int y1, int x2, int y2);
```

```
void main()
```

```
{
```

```
    int x1, y1, x2, y2;
    float dist;
```

```
    printf("Please enter 2 points (x1, y1, x2, y2): ");
    scanf("%d %d %d %d",
          &x1, &y1, &x2, &y2);
```

```
    dist = distance(x1, y1, x2, y2);
    printf("The distance is %f\n", dist);
```

```
}
```

```
int power(int base, int exponent)
```

```
{
```

```
    int i, result=1;
    for (i=0 ; i < exponent ; i++)
        result *= base;
    return result;
```

```
}
```

גם ל- power יש מרחב
זיכרון, אך לא נראה אותו
פה מפאת חוסר מקום

sqrt
distance
e
(line 5)
(line 5)

מחסנית הקריאות

```
float distance(int x1, int y1, int x2, int y2)
{
```

```
    float xDiffPower, yDiffPower;
    float result;
```

```
    xDiffPower = power(x2-x1, 2);
```

```
    yDiffPower = power(y2-y1, 2);
```

```
    result = sqrt(xDiffPower+yDiffPower);
    return result;
```

int: x1	0	1000
int: y1	0	1004
int: x2	4	1008
int: y2	4	1012
float: dist	5.6	1016

הזיכרון של ה- main

int: x1	0	2000
int: y1	0	2004
int: x2	4	2008
int: y2	4	2012
float: xDiffPower	16	2016
float: yDiffPower	16	2024
float: result	5.6	2032

הזיכרון של distance

דוגמא איך לא כותבים פונקציה

```
#include <stdio.h>

void power()
{
    int base, exponent, i, result=1;

    printf("Please enter base and exponent: ");
    scanf("%d %d", &base, &exponent);

    for (i=0 ; i < exponent ; i++)
        result *= base;

    printf("The result is %d\n", result);
}

void main()
{
    power();
}
```

מדוע לא לכתוב קוד כמו הדוגמא הקודמת?

ניתן לעקוף חלקית את הרעיון של העברת ארגומנטים והחזרת ערך מפונקציות ע"י כך שפונקציה תבקש מהמשתמש את הנתונים ותדפיס בסוף את התוצאה

■ למשל, כמו בדוגמא הקודמת, הפונקציה המחשבת חזקה לא קיבלה פרמטרים ולא החזירה את התוצאה

לא נעשה זאת!

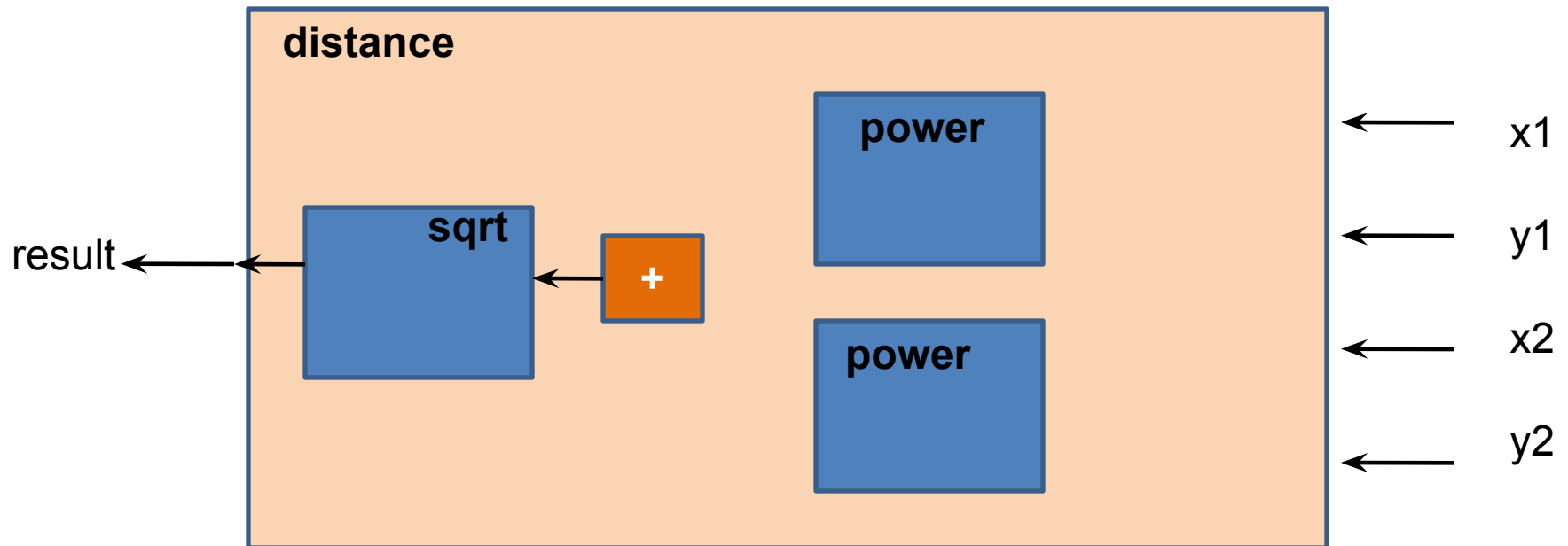
נמנע מלקלוט קלט בתוך פונקציה, אלא אם זה יעוד הפונקציה הסיבה: יתכן ופונקציה אחרת צריכה להשתמש בנתון המתקבל או להעביר נתון, ולא ניתן לצפות מי ישתמש בפונקציה שלנו

■ דוגמא: כמו בפונקציה distance הקוראת ל- power

כדי לבדוק פונקציה, נקלוט את הנתונים ב- main ונעבירם לפונקציה. את התוצאה נדפיס ב- main.

מדוע לא לכתוב קוד כמו הדוגמא הקודמת?

אם הפונקציה `power` אינה מקבלת ערכים, אלא קולטת מהמשתמש, לא ניתן להעביר אליה תוצאת חישוב כמו שנדרש בשאלה זו...



פונקציות ספריה

- כל תוכנית בשפת תכנות עילית, ובפרט שפת C, מורכבת מאוסף של פונקציות
 - קיימות פונקציות ספריה בשפה אותן ניתן להכליל בתוכנית ולקרוא להן בכל מקום בו נרצה ולהשתמש בהן כמה פעמים שנרצה
 - למשל כאשר עושים:
- ```
#include <stdio.h>
```
- ניתן להשתמש בפונקציות המוגדרות בספריה זו בכל מיני מקומות בקוד (למשל printf, scanf)
- גם אנו יכולים לכתוב פונקציות או ספריות של פונקציות, שאותן ניתן להכליל ולהשתמש בהן כרצוננו

# יצירת קובץ ספריה

- עד כה השתמשנו בפונקציות שקיבלנו משפת C
- כל פונקציה כזו נכתבה בספריה שאליה ביצענו `include` למשל כדי להשתמש בפונקציות של מחרוזות, עשינו
  - `#include <string.h>`
- כדי לייצר ספריה משלנו נייצר 2 קבצים חדשים:
  - `file_name>.h` הוא קובץ אשר יכלול את ה-  
prototypes של הפונקציות, ואליו אח"כ נעשה `include` מהקובץ שירצה להשתמש בפונקציות המוגדרות בו
  - `file_name>.c` הוא קובץ שיכיל `include` לקובץ ה-  
header התואם ויממש את הפונקציות המוגדרות בו
- `include` לספריה שכתבנו יהיו בתוך "" (גרשיים) ולא בתוך `<>`

# דוגמא – ספריה המטפלת בתווים (1)

---

הקובץ character.h

// prototypes

```
int isSmallLetter(char);
int isCapitalLetter(char);
int isAlpha(char);
int isDigit(char);
```

## דוגמא – ספריה המטפלת בתווים (2)

---

**#include "character.h"**

הקובץ character.c

```
int isSmallLetter(char ch)
{
 return (ch >= 'a' && ch <= 'z');
}

int isCapitalLetter(char ch)
{
 return (ch >= 'A' && ch <= 'Z');
}

int isAlpha(char ch)
{
 return (isSmallLetter(ch) || isCapitalLetter(ch));
}

int isDigit(char ch)
{
 return (ch >= '0' && ch <= '9');
}
```

## דוגמא – ספריה המטפלת בתווים (3)

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

הקובץ main.c

```
#include "character.h"
```

```
void main()
```

```
{
```

```
 char ch=0;
```

```
 printf("Please enter a character: ");
```

```
 ch = getchar();
```

```
 printf("Is '%c' a digit? %d\n", ch, isDigit(ch));
```

```
 printf("Is '%c' a small letter? %d\n", ch, isSmallLetter(ch));
```

```
 printf("Is '%c' a capital letter? %d\n", ch, isCapitalLetter(ch));
```

```
 printf("Is '%c' a letter? %d\n", ch, isAlpha(ch));
```

```
}
```



# ספריות שלנו - סיכום

- ניתן לכתוב את כל הקוד בקובץ אחד, אבל אם אנחנו כותבים קוד כללי, שיתכן ויהיה שימושי גם במקומות אחרים, נעדיף לחלק את הקוד לקובץ ספריה נפרד
- מי שירצה להשתמש בספריה שלנו, יתעניין מה יש לה להציע, ולא איך היא מבצעת את הפעולות, וכך הוא יוכל להסתכל בקובץ header בלבד בו יש ריכוז של כל הפונקציות
- מכירת הספריה ללקוח תסתכם בקובץ ה- h ובקובץ בינארי שהוא תוצר של הקומפילציה, וכך לא נחשוף את ה"איך"

# תזכורת למשמעות של העברה by value

```
void incNumber(int x)
{
```

```
 printf("In function: number before: %d\n", x);
```

```
 x++;
```

```
 printf("In function: number after: %d\n", x);
```

```
}
```

```
void main()
```

```
{
```

```
 int num = 3;
```

```
 printf("In main: number before function: %d\n", num);
```

```
 incNumber(num);
```

```
 printf("In main: number after function: %d\n", num);
```

```
}
```

```
In main: number before function: 3
In function: number before: 3
In function: number after: 4
In main: number after function: 3
Press any key to continue . . .
```

|        |          |      |
|--------|----------|------|
| int: x | <b>4</b> | 2000 |
|--------|----------|------|

הזיכרון של incNumber

|          |          |      |
|----------|----------|------|
| int: num | <b>3</b> | 1000 |
|----------|----------|------|

הזיכרון של main

# מערכים כפרמטר לפונקציה - דוגמא

```
void incArray(int arr[], int size)
{
 int i;
 for (i=0 ; i < size ; i++)
 arr[i]++;
}
```

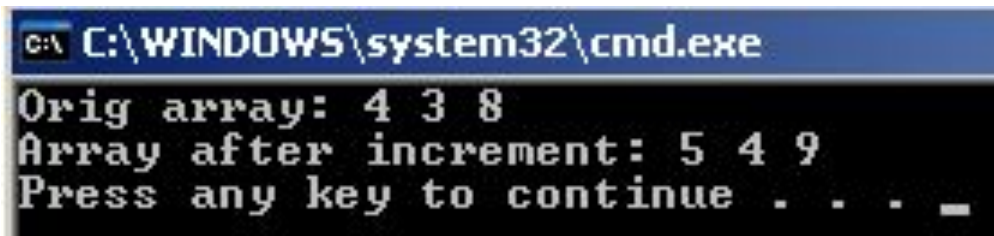
```
void printArray(int arr[], int size)
{
 int i;
 for (i=0 ; i < size ; i++)
 printf("%d ", arr[i]);
 printf("\n");
}
```

```
void main()
{
 int arr[] = {4,3,8};
 int size = sizeof(arr)/sizeof(arr[0]);

 printf("Orig array: ");
 printArray(arr, size);

 incArray(arr, size);

 printf("Array after increment: ");
 printArray(arr, size);
}
```



```
C:\WINDOWS\system32\cmd.exe
Orig array: 4 3 8
Array after increment: 5 4 9
Press any key to continue . . . _
```

# מערכים כפרמטר לפונקציה

---

- ראינו כי מערך מתנהג באופן שונה מאשר משתנה מטיפוס בסיסי כאשר מעבירים אותו לפונקציה
- כאשר מעבירים מערך לפונקציה, לא מועבר עותק של המערך (by value), אלא מועברת רק כתובת ההתחלה של המערך
- לכן כאשר מעבירים מערך לפונקציה ומשנים אותו, השינוי משפיע על המערך המקורי, ולא על עותק – בניגוד לכל משתנה אחר שאנחנו מכירים!
- נסביר לעומק כאשר נלמד את השיעור על מצביעים

# העברת מספר האיברים במערך כפרמטר לפונקציה

---

□ ראינו כי כאשר שלחנו מערך לפונקציה העברנו גם את מספר האיברים שבו, ולא התבססנו על ערך קבוע

□ זאת כדי שהפונקציה תהיה מספיק כללית על-מנת שתבצע את העבודה על מערכים בגדלים שונים

# דוגמא איך פונקציה המקבלת מערך לא צריכה להיות

```
#define SIZE 3
```

שימו לב: הקבוע מוגדר באותיות גדולות. דגש

```
void printArray(int arr[])
```

```
{
```

```
 int i;
```

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

```
 printf("%d ", arr[i]);
```

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

```
}
```

```
void main()
```

```
{
```

```
 int arr1[SIZE] = {1,2,3}, arr2[5]={10,20,30,40,50};
```

```
 printf("arr1: ");
```

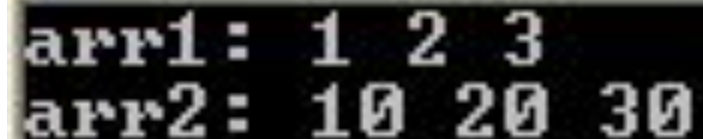
```
 printArray(arr1);
```

```
 printf("arr2: ");
```

```
 printArray(arr2);
```

```
{
```

בהמשך...  
הפונקציה יודעת לטפל אך ורק במערכים בגודל הקבוע **SIZE**, ולא תעשה את המתבקש עבור מערכים בגודל



```
arr1: 1 2 3
arr2: 10 20 30
```

# דוגמא איך פונקציה המקבלת מערך כ צריכה להיות

```
#define SIZE 3
```

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

שימו לב: הפרמטר **size** והקבוע **SIZE**  
שונים (הקומפיילר הוא **case sensitive**).  
דגש בהמשך...

הפונקציה מקבלת כפרמטר נוסף את כמות האיברים  
שעליה להדפיס, ולכן יודעת לטפל במערך בכל גודל

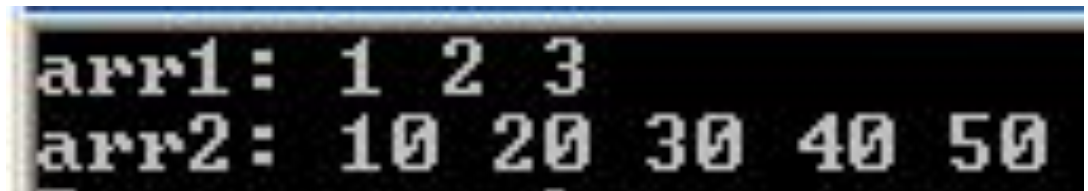
```
void main()
{
```

```
 int arr1[SIZE] = {1,2,3}, arr2[5]={10,20,30,40,50};
```

```
 printf("arr1: ");
 printArray(arr1, SIZE);
```

```
 printf("arr2: ");
 printArray(arr2, 5);
```

```
{
```



```
arr1: 1 2 3
arr2: 10 20 30 40 50
```

# שימו לב: שגיאה נפוצה!

□ מי שלא מקפיד על הגדרת קבועים באותיות גדולות עלול להיתקל בשגיאת הקומפילציה הבאה:

```
error C2143: syntax error : missing '}' before 'constant'
error C2143: syntax error : missing '{' before 'constant'
error C2059: syntax error : '<Unknown>'
error C2059: syntax error : '}'
```

#define **size** 3

בעקבות פקודת ה- **define**: בכל מקום שהקומפיילר רואה **size** הוא מחליף אותו בערך 3

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

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

```
{
void main() }...{
```

ולכן מה שהקומפיילר רואה בתור הפרמטר השני: **int 3**  
ו- 3 אינו שם תקף למשתנה



# העברת מטריצה לפונקציה –

## דוגמא

```
#define COLS 5
```

```
void setMatrix(int mat[][COLS], int rows)
{
```

הפונקציה מספיק כללית כדי  
לבצע את העבודה עבור  
מטריצה עם כל מספר שורות

```
 int i, j;
 for (i=0 ; i < rows ; i++)
 {
 for (j=0 ; j < COLS ; j++)
 mat[i][j] = i*10+j;
```

```
 }
}

void printMatrix(int mat[][COLS], int rows)
{
```

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

```
 }
}
```

```
void main()
{
```

```
 int mat1[3][COLS];
 int mat2[4][COLS];
```

```
 setMatrix(mat1, 3);
 setMatrix(mat2, 4);
```

```
 printf("Matrix 1:\n");
 printMatrix(mat1, 3);
```

```
 printf("Matrix 2:\n");
 printMatrix(mat2, 4);
```

```
}
```

# העברת מטריצה לפונקציה – דוגמא (פלט)

---

```
Matrix 1:
 0 1 2 3 4
 10 11 12 13 14
 20 21 22 23 24
Matrix 2:
 0 1 2 3 4
 10 11 12 13 14
 20 21 22 23 24
 30 31 32 33 34
Press any key to continue . . .
```

# העברת מטריצה לפונקציה

- גם כאשר מעבירים מטריצה לפונקציה, עוברת כתובת ההתחלה בלבד, ולכן שינוי המטריצה בפונקציה משנה את המטריצה המקורית
- כאשר מעבירים מטריצה לפונקציה, ניתן לציין רק את כמות העמודות ולהשאיר את הסוגריים של השורות ריקים (במקום להעביר כפרמטר את מספר השורות)
- נרצה להעביר את כמות השורות כדי שהפונקציה תהיה מספיק כללית לכל מטריצה עם אותו מספר עמודות
- בהמשך נראה שאפשר גם להעביר מטריצה שמספר העמודות בה שונה

# יצירת מספרים אקראיים

- בשפת C קיימת הפונקציה rand() אשר מגרילה מספר
- טווח הערכים שיוגרו הינו בין 0 לקבוע MAX\_RANDOM
- קבוע זה מוגדר בספריה stdlib.h

```
#include <stdio.h>
#include <stdlib.h> // for 'RAND_MAX'
```

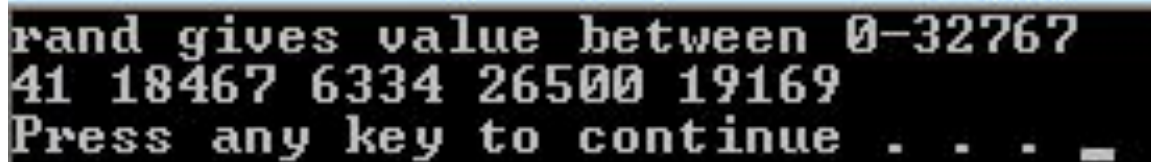
```
void main()
{
```

```
 int i;
```

```
 printf("rand gives value between 0-%d\n", RAND_MAX);
 for (i=0 ; i < 5 ; i++)
 printf("%d ", rand());
```

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

```
}
```



```
rand gives value between 0-32767
41 18467 6334 26500 19169
Press any key to continue . . . _
```

## יצירת מספרים אקראיים (2)

---

- rand פועלת עפ"י אלגוריתם קבוע המתבסס על מספר התחלתי כלשהו (נקרא seed number), ולכן תמיד בכל הרצה תחזיר ערכים זהים
- כדי ש- rand באמת תחזיר מספרים אקראיים, כלומר תתבסס על מספר התחלתי שונה בכל פעם, יש להפעיל את הפונקציה srand עם ערך התחלתי המייצג את הזמן הנוכחי (והוא אכן יחודי בכל הרצה)

## יצירת מספרים אקראיים (3)

```
#include <stdio.h>
#include <stdlib.h> // for 'RAND_MAX'
#include <time.h> // for 'time'
```

```
void main()
{
```

```
 int i;
```

*time(NULL)* מחזירה מספר המייצג את הזמן הנוכחי (מספר השניות שעברו מאז ה- 1.1.1970)

```
 srand (time(NULL)); // initialize random seed
```

```
 printf("rand gives value between 0-%%d\\n", RAND_MAX);
```

```
 for (i=0 ; i < 5 ; i++)
 printf("%d ", rand());
```

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

```
{
```

```
rand gives value between 0-32767
27814 29769 16801 24198 9211
Press any key to continue . . .
```

```
rand gives value between 0-32767
27236 27845 570 24832 18539
Press any key to continue . . .
```

# יצירת מספרים אקראיים בטווח מסוים

```
#include <stdio.h>
#include <time.h> // for 'time'
```

```
void main()
{
```

```
 int i;
```

```
 srand (time(NULL)); // initialize random seed
```

```
 printf("rand gives value between 1-6\n");
```

```
 for (i=0 ; i < 5 ; i++)
```

```
 printf("%d ", rand()%6+1);
```

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

```
{
```

```
rand gives value between 1-6
5 6 5 2 3
Press any key to continue . . .
```

```
rand gives value between 1-6
5 1 5 1 2
Press any key to continue . . .
```

פעולת %6 תחזיר לנו ערכים בין 0-5,  
ומאחר ואנחנו רוצים בטווח בין 1-6 הוספנו

## יצירת מספרים אקראיים בטווח מסוים (2)

```
#include <stdio.h>
#include <time.h> // for 'time'
```

```
void main()
{
 int i;
```

```
 srand (time(NULL)); // initialize random seed
```

```
 printf("rand gives value between -10 to 10\n");
 for (i=0 ; i < 5 ; i++)
 printf("%d ", rand()%21-10);
```

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

```
{
```

```
rand gives value between -10 to 10
-3 7 -6 5 3
Press any key to continue . . .
```

```
rand gives value between -10 to 10
3 10 9 -4 2
Press any key to continue . . .
```

פעולת `%21` תחזיר לנו ערכים בין 0-20, ומאחר ואנחנו רוצים בטווח בין -10 ל-10 החסרנו 10



אגב, זו לא פונקציית rand טובה (-);

---

```
int getRandomNumber()
{
 return 4; // chosen by fair dice roll.
 // guaranteed to be random.
}
```

[https://sslimgs.xkcd.com/comics/random\\_number.png](https://sslimgs.xkcd.com/comics/random_number.png)

# פונקציות נפוצות שיש בספרייה `math.h`

---

**`cos, sin`**: מחשבות סינוס וקוסינוס בהתאמה (לשים לב שמקבלות זווית ברדיאנים, ולא במעלות)

**`ceil`**: מעגלת מספר כלפי מעלה, מחזירה `float`

**`floor`**: מעגלת מספר כלפי מטה, מחזירה `float`

**`abs`**: נותנת ערך מוחלט של מספר ועושה `floor` לשלם

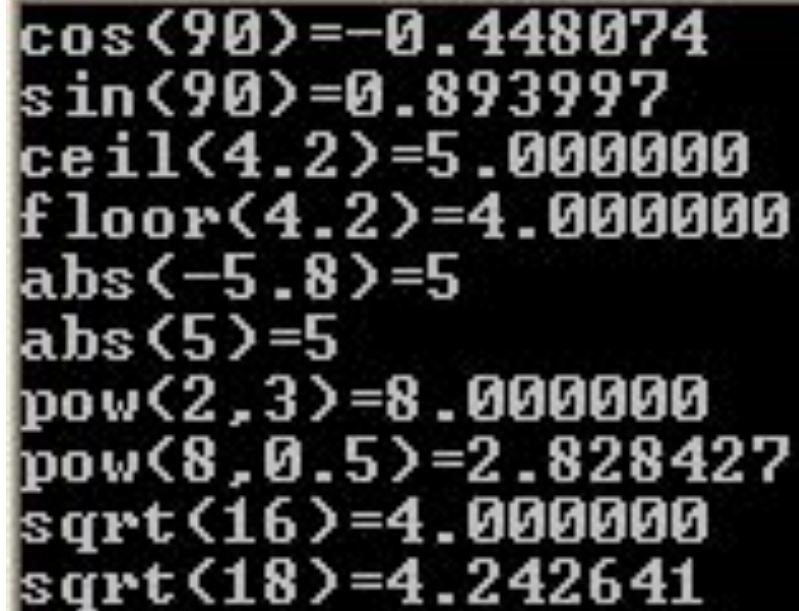
**`pow`**: מחשבת חזקה

**`sqrt`**: מחשבת שורש

# דוגמאות חישוב

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

void main()
{
 printf("cos(90)=%lf\n", cos(90));
 printf("sin(90)=%lf\n", sin(90));
 printf("ceil(4.2)=%f\n", ceil(4.2));
 printf("floor(4.2)=%f\n", floor(4.2));
 printf("abs(-5.8)=%d\n", abs(-5.8));
 printf("abs(5)=%d\n", abs(5));
 printf("pow(2,3)=%lf\n", pow(2, 3));
 printf("pow(8,0.5)=%lf\n", pow(8, 0.5));
 printf("sqrt(16)=%lf\n", sqrt(16));
 printf("sqrt(18)=%lf\n", sqrt(18));
}
```



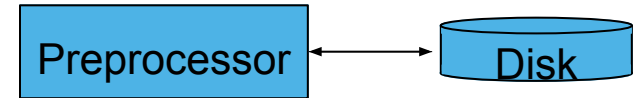
```
cos<90>=-0.448074
sin<90>=0.893997
ceil<4.2>=5.000000
floor<4.2>=4.000000
abs<-5.8>=5
abs<5>=5
pow<2,3>=8.000000
pow<8,0.5>=2.828427
sqrt<16>=4.000000
sqrt<18>=4.242641
```

# תהליך הפיכת תוכנית C לשפת מכונה

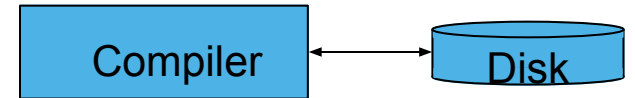
התוכנית נכתבת בעורך טקסטואלי ונכתבת לדיסק



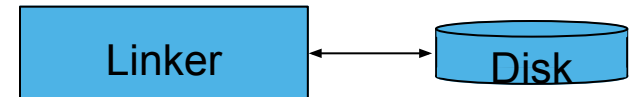
קדם המעבד עובר על הקוד ומבצע החלפות נחוצות (כל הפקודות המתחילות ב-#, כמו define ו-include



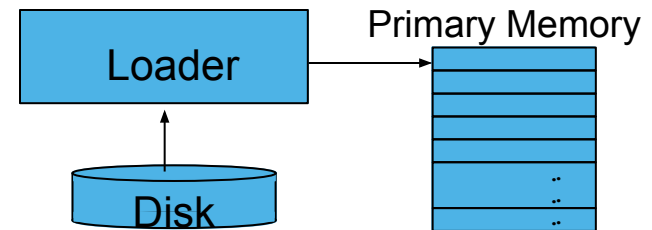
עבור כל קובץ c, הקומפילר יוצר קובץ obj בשפת מכונה ושומר אותו על הדיסק. בשלב זה רק נבדקת תקינות הסינטקס בפונקציות.



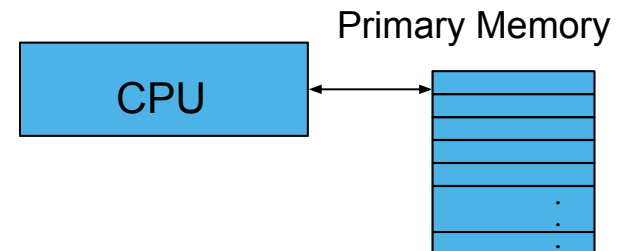
ה-linker קושר את קובץ ה-obj עם הספריות, ויוצר קובץ exe המוכן להרצה ונשמר בדיסק. כלומר, מקשר בין קריאה לפונקציה, למימוש שלה.



בעת ההרצה, טוענים את התוכנית מהדיסק לזיכרון הראשי



עם הרצת התוכנית, ה-cpu עובר על כל פקודה ומריץ אותה



# דוגמא לשגיאת קומפילציה

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

```
void foo(int x)
{
 printf("the number is %d\n");
}
```

```
void main()
{
 for (i=0 ; i < 5 ; i++)
 {
 printf("%d ", i);
 goo(i);
 }
}
```

נקבל את שגיאת הקומפילציה הבאה:  
error C2065: 'i' : undeclared identifier

הקומפיילר רק נותן אזהרה שהוא לא מוצא את goo:  
warning C4013: 'goo' undefined; assuming  
extern returning int

במידה ויש שגיאות קומפילציה, הקומפיילר אינו ממשיך לתהליך לינקר

# דוגמא לשגיאת לינקר

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

```
void foo(int x)
{
 printf("the number is %d\n");
}
```

```
void main()
{
 int i;

 for (i=0 ; i < 5 ; i++)
 {
 printf("%d ", i);
 goo(i);
 }
}
```

תוכנית זו מתקמפלת (אין שגיאות סינטקטיות בתוך הפונקציות) ולכן הקומפיילר עובר לתהליך הלינקר, ומוציא את השגיאה הבאה:

**error LNK2019: unresolved external symbol \_goo referenced in function \_main**

# נכון שזה נכון?? אז די!

---

**A software engineer was smoking...**

**A lady standing nearby asked him,  
"can't you see the Warning?? Smoking  
is injurious to health..!"**

**He replied, "We are bothered only about  
Errors, not Warnings !!"**

# סוגי משתנים

---

משתנים מקומיים □

משתנים סטטיים □

משתנים גלובליים □



# משתנים מקומיים

- עד כה ראינו שכל הקוד שלנו מורכב מפונקציות
- המשתנים שבתוך כל פונקציה (גם אלו שהועברו כפרמטרים) נקראים **משתנים מקומיים (לוקאליים)** וטווח ההכרה שלהם הוא רק בפונקציה בה הם מוגדרים
- ערכו של משתנה לוקאלי נשמר כל עוד אנחנו בפונקציה, והוא נמחק ביציאה ממנה
- משתנה לוקאלי מאוחסן במחסנית של הפונקציה
- בכל קריאה חדשה לפונקציה המשתנים מאותחלים מחדש ונמחקים בסיום ביצוע הפונקציה
- ערכו זבל במידה ולא אותחל

# משתנים סטטיים

משתנה סטטי הוא משתנה שערכו נשמר בין הקריאות השונות לפונקציה

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

```
int counter()
{
```

```
 static int count = 0;
 count++;
 return count;
}
```

כדי להגדיר משתנה סטטי נשתמש במילת המפתח static לפני טיפוס המשתנה

משתנה סטטי מאוחל רק בקריאה הראשונה לפונקציה

counter::count = 0

זיכרון ה- data segment

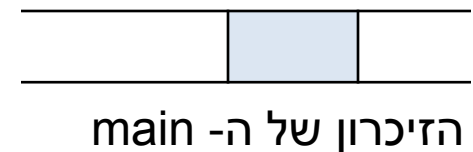
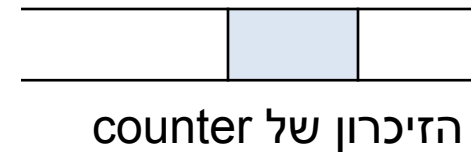
```
void main()
{
```

```
 printf("'counter' was called %d times\n",
 "'counter' was called %d times\n",
 "'counter' was called %d times\n",
 1, 2, 3);
}
```

```
counter();
counter();
counter();
```

counte  
r  
main  
( line  
2)

מחסנית הקריאות



```
C:\WINDOWS\system32\cmd.exe
'counter' was called 1 times
'counter' was called 2 times
'counter' was called 3 times
Press any key to continue . . .
```

# משתנים סטטיים

---

- בדומה למשתנים מקומיים, המשתנים הסטטיים מוכרים אך ורק בתוך הפונקציה אשר בה הם הוגדרו
- משך חייהם מרגע תחילת הריצה של התוכנית ועד סיום ריצת התוכנית.
- מאחר ושטח הזיכרון של כל פונקציה נמחק עם היציאה ממנה, משתנים סטטיים נשמרים באזור זיכרון אחר הנקרא data-segment, הקיים לאורך כל חיי התוכנית, והם משתחררים רק עם היציאה מהתוכנית

משתנה גלובלי מוגדר בראש התוכנית  
(אינו משויך לאף פונקציה)

כל הפונקציות שכתובות  
בקובץ בו הוגדר יכולות  
לגשת אליו ולשנות את ערכו

## משתנים גלובליים

```
int global = 3;
```

```
void incGlobal()
```

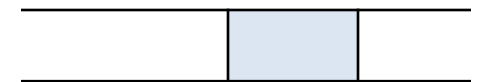
```
{
 global++;
 printf("In function: global=%d\n", global);
}
```

```
void main()
```

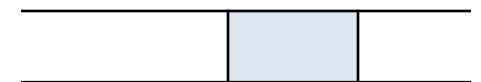
```
{
 printf("At first, global=%d\n", global);
 incGlobal();
 printf("After function (1), global=%d\n", global);
 global = 10;
 printf("In main after change global, global=%d\n", global);
 incGlobal();
 printf("After function (2), global=%d\n", global);
}
```

```
At first, global=3
In function: global=4
After function (1), global=4
In main after change global, global=10
In function: global=11
After function (2), global=11
Press any key to continue . . . _
```

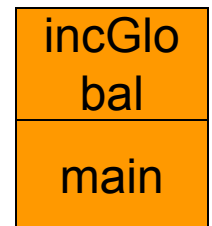
זיכרון ה- data  
segment  
global = 10



הזיכרון של incGlobal



הזיכרון של ה- main



מחסנית הקריאות

# משתנים גלובליים

□ במידה ולא אותחל ערכו 0, ולא זבל

□ משתנה גלובלי קיים לאורך כל חיי התוכנית

□ השימוש בו הוא בעייתי ולכן נשמר למצבים מיוחדים בלבד

■ כל אחד יכול לשנות אותו, מה שפוגע ברעיון של הסתרת המידע ושכל פונקציה מסתמכת רק על נתונים שהם פנימיים לה

□ מאחר ואינו משויך לאף פונקציה, הוא אינו נמצא על המחסנית, אלא על חלקת הזיכרון המשותפת לכל הפונקציות ה- data-segment

# השוואה בין סוגי המשתנים השונים

| היכן מוגדר        | משתנה מקומי<br>(רגיל)     | משתנה גלובלי                    | משתנה סטטי                   |
|-------------------|---------------------------|---------------------------------|------------------------------|
| היכן מוגדר        | בתוך הפונקציה             | מחוץ לפונקציות                  | בתוך הפונקציה                |
| טווח ההכרה        | בפונקציה בה הוא מוגדר     | מנקודת הגדרתו ומטה              | בפונקציה בה הוא מוגדר        |
| מי יכול לגשת אליו | רק הפונקציה בה הוא מוגדר  | כל מקום בקוד הנמצא מתחת להגדרתו | רק הפונקציה בה הוא מוגדר     |
| מתי מאותחל        | בכל פעם כשנכנסים לפונקציה | בתחילת ריצת התוכנית             | רק בפעם הראשונה שמגיעים אליו |

# ביחידה זו למדנו:

---

□ מהי פונקציה, פרמטרים וערך מוחזר

□ כיצד נראה הזיכרון

□ ספריות של פונקציות

□ מערכים כפרמטר לפונקציה

□ יצירת מספרים אקראיים

□ פונקציות מ- `math.h`

□ תהליך הפיכת תוכנית ב- C לשפת מכונה

□ סוגי משתנים וטווח הכרתם: לוקאליים, גלובליים, סטטיים

# תרגיל 1:

□ כתוב פונקציה המקבלת 2 מספרים

□ במידה ושני המספרים בעלי אורך זהה, היא תחזיר בכמה מיקומים זהים במספרים יש ספרה זהה

□ דוגמאות:

■ עבור 1594 ו- 7599 יוחזר 2 מאחר וספרת העשרות וספרת המאות זהה בשני המספרים

■ עבור 9287 ו- 9487 יוחזר 3, כי גם ספרות האחדות, עשרות ואלפים זהות בשני המספרים

□ במידה והמספרים שונים באורכם הפונקציה תחזיר 1-



## תרגיל 2:

□ כתוב פונקציה המקבלת 2 מחרוזות. הפונקציה תחזיר 1 האם המחרוזת השניה היא סופה של המחרוזת הראשונה ו- 0 אחרת

□ דוגמאות:

- עבור **abcde** ו- cde יוחזר 1
- עבור abcde ו- cfe יוחזר 0
- עבור abcde ו- zabcde יוחזר 0

## תרגיל 3:

- כתוב פונקציה המקבלת מערך של מספרים, גודלו וספרה
- הפונקציה תחזיר כמה ערכים במערך מכילים את הספרה

### □ דוגמאות:

- עבור המערך 123, 456, 817, 991 והספרה 1 יוחזר 3, מאחר והספרה 1 מופיעה ב- 3 איברים במערך
- עבור המערך 9988, 458, 751, 654, 888 והספרה 8 יוחזר 3 מאחר והספרה 8 מופיעה ב- 3 מספרים. שימו לב שאין לספור את הספרה פעמיים עבור מספר מסויים!

## תרגיל 4:

□ כתוב פונקציה המקבלת מטריצה של מספרים ואת כמות השורות במטריצה. הפונקציה תחזיר 1 אם ערכי מסגרת המטריצה שווים ל-1 ו-0 אחרת. אין חשיבות לערכם של האיברים שאינם על המסגרת

□ דוגמא:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 3 | 4 | 5 | 4 | 3 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |