

# קורס תכנות

שיעור רביעי: פונקציות, מבוא  
לרקורסיה

# לולאות - תזכורת

- לולאה:
  - קטע קוד המתבצע שוב ושוב ברצף כל עוד תנאי מסוים מתקיים
- לולאות **for**:
  - כאשר רוצים לבצע את הלולאה בצורה סדרתית:
    - תבצע את הלולאה 10 פעמים ...
    - תבצע את הלולאה  $n$  פעמים ...
- לולאות **while**:
  - כאשר רוצים לבצע את הלולאה מספר שרירותי של פעמים:
    - כל עוד  $i$  זוגי...
    - כל עוד לא הייתה טעות בחישוב ...

# לולאות - תזכורת

- `break`:
  - מסיים (שובר) את ריצת הלולאה
- `continue`:
  - מסיים רק את הסיבוב (iteration) הנוכחי של הלולאה

# לולאות do-while

```
initialize ...  
  
do  
{  
    do something ...  
    increment ...  
}  
while ( condition );
```

ההבדל בין **do-while** ל-**while**:

- הפקודות מבוצעות לפחות פעם אחת (אפילו אם התנאי לא מתקיים לעולם).
- התנאי נבדק לאחר ביצוע הפקודות.

# לולאות do-while

```
#include <stdio.h>

int main()
{
    int i=1;

    do
    {
        printf("%d\n", i);
        i++;
    }
    while (i < 10);

    return 0;
}
```

```
#include <stdio.h>

int main()
{
    int i=1;

    while (i < 10)
    {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

# לולאות do-while

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

```
int main()
```

```
{
```

```
int i=11;
```

```
do
```

```
{
```

```
printf("%d\n", i);
```

```
i++;
```

```
}
```

```
while (i < 10);
```

```
return 0;
```

```
}
```

תנאי הלולאה לא מתקיים

פלט: "11"

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

```
int main()
```

```
{
```

```
int i=11;
```

```
while (i < 10)
```

```
{
```

```
printf("%d\n", i);
```

```
i++;
```

```
}
```

```
return 0;
```

```
}
```

אין פלט

# כיצד מחשבים?

```
#include <stdio.h>
int main()
{
    int i = 0;
    double sum = 0, result;

    for (result = 1, i = 1; i <= 20; ++i)
        result = result * 2;
    sum += result;

    for(result = 1, i = 1; i <= 15; ++i)
        result = result * 3;
    sum += result;

    for(result = 1, i = 1; i <= 17; ++i)
        result = result * 5;
    sum += result;

    printf("2^20 + 3^15 + 5^17= %g", sum);
    return 0;
}
```

כיצד מחשבים את:

$$2^{20} + 3^{15} + 5^{17}$$

חישוב  $2^{20}$

חישוב  $3^{15}$

חישוב  $5^{17}$

# מה היינו רוצים?

- בעיות:

- שכפול של קטע קוד כמעט זהה 3 פעמים
- אם היו יותר מחוברים היה צריך לשכפל פעמים נוספות
- התכנית מתארכת ונהיית מסורבלת
- מקור לבאגים וטעויות copy / paste
- קשה להבין "למה התכוון המשורר"

- מה היינו רוצים?

- לכתוב פעם אחת בלבד את הקוד
- להשתמש באותו קטע קוד מספר פעמים אבל עם פרמטרים שונים בכל פעם



# פתרון - פונקציות

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

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

```
{
```

```
    int i = 0;
```

```
    double result = 1;
```

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

```
        result = result * base;
```

```
    return result;
```

```
}
```

```
int main()
```

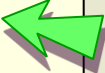
```
{
```

```
    double sum = power(2,20) + power(3,15) + power(5,17);
```

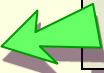
```
    printf("2^20 + 3^15 + 5^17= %g", sum);
```

```
    return 0;
```


```
}
```



הגדרת פונקציה בשם power  
המחשבת את  $base^{exponent}$



התוכנית מתחילה לרוץ מכאן



קריאה  
לפונקציה

# פונקציות

- פונקציה:

- קטע קוד בעל שם
- יכול לקבל ערכי קלט לתוך משתנים
- יכול להחזיר ערך פלט יחיד

- שימושים:

- אפשר לקרוא לפונקציה הרבה פעמים במהלך התוכנית, קריאה עם קלט שונה יכולה להניב תוצאה שונה
- מיחזור קוד:
- כותבים פעם אחת, בודקים פעם אחת ... משתמשים הרבה פעמים
- לדוגמא: printf, scanf
- בהירות קוד: פיתוח וקריאה (אבסטרקציה)
- עקרון ההכמסה (encapsulation)

# ושוב ... למה להשתמש בפונקציות?

- חלוקת התכנית לחלקים **לוגים שונים**
- כל חלק מבצע פעולה בסיסית אחרת
- קריאת נתונים, עיבוד נתונים, הדפסה ...
- **פירוק בעיה מסובכת לתת בעיות פשוטות יותר - אבסטרקציה**
  
- **שימוש חוזר** באותו קטע קוד מספר פעמים באותה התוכנית
  
- **שימוש חוזר** באותו קטע קוד בתכניות שונות
- ניתן להשתמש בפונקציות שכתבנו גם בתוכניות אחרות
- לדוגמא printf שמופיעה בספרייה `stdio.h`.

# הגדרת פונקציות

```
type function_name( type parameter1, type parameter2, ... )
{
    local variables declaration
    statement 1
    statement 2
    ...
    ...
    return value;
}
```

- לפני שניתן להשתמש בפונקציה צריך להגדיר אותה
- לפונקציה יש:
  - שם
  - טיפוס הערך המוחזר
  - משתני קלט
- אוסף פקודות המרכיב את גוף הפונקציה

# הגדרת פונקציות

```
double power(double base, int exponent)
{
    int i = 0;
    double result = 1;

    for (i = 1; i <= exponent; i++)
        result = result * base;

    return result;
}
```

בגוף הפונקציה ניתן להשתמש בכל  
אוסף הפקודות של C  
(כולל לקרוא לפונקציות אחרות)

• שם הפונקציה:

• `power`

• טיפוס ערך מוחזר:

• `double`

• משתני קלט:

• `double base, int exponent`

# שם הפונקציה

- שם הפונקציה:
  - אותם כללים כמו שם של משתנה
  - לא יכול להכיל סימנים כמו +-\$\*% ...
  - חייב להתחיל באות
  - אסור להשתמש במילים שמורות
- מומלץ ששם הפונקציה יהיה שם בעל משמעות:
  - max
  - printf
  - getPacManPos

# משתני קלט

- **משתני הקלט:**

- מוגדרים בהגדרת הפונקציה
- יש להם שם וטיפוס (כמו משתנים רגילים)
- מוכרים אך ורק על ידי הפונקציה

```
int max( int val1, int val2 )
```

```
double squareRoot( double number )
```

- **אם הפונקציה לא מקבלת ערכים:**

- אפשר להשאיר את הסוגריים ריקים
- או לכתוב בתוכם **void**

```
int isPacManAlive()
```

```
int getNumGhosts( void )
```

# קריאה לפונקציה

## קריאות לפונקציה

```
int main()
{
    double solution = power(2,20) + power(3,15) + power(5,17);

    printf("2^20 + 3^15 + 5^17 = %g", solution);

    return 0;
}
```

### קריאה לפונקציה:

- היא פקודה המתבצעת ברצף הפקודות של התוכנית
- ניתן לקרוא לפונקציה גם מתוך פונקציה אחרת
- **חייבים** להעביר לפונקציה ערכים בהתאם להגדרת הפונקציה
- בסיום ריצת הפונקציה, חוזרים למקום שלאחר הקריאה לפונקציה



# הערך המוחזר

```
void print_help()
```

- טיפוס הערך המוחזר:
  - אם הפונקציה לא מחזירה ערכים, אז הטיפוס הוא **void**.
  - כאשר הפונקציה מחזירה ערך שונה מ-**void**:
    - צריכה להתבצע פקודת **return** שבה יוחזר ערך מהטיפוס המתאים
    - **return** היא הפקודה **האחרונה** המבוצעת בפונקציה
    - גם אם מופיעות אחריה פקודות נוספות
  - ניתן להשתמש בפקודת **return** גם בפונקציה שלא מחזירה ערך. המשמעות תהיה – יציאה מיידית מהפונקציה.

# דוגמא להמחשה בלבד

```
int greater(int a, int b)
{
    if ( a > b )
        return 1;
    else
        return 0;

    printf("Will this be printed?\n");
}
```

יכולה להופיע יותר מפקודת return אחת בפונקציה  
אולם רק אחת תתבצע

הפקודה הזאת לא תתבצע

```
int greater(int a, int b)
{
    return ( a > b );
}
```

הדרך הנכונה לממש את greater

# הערך המוחזר


- עם סיום הפונקציה, במידה והוחזר ערך:
  - ניתן לבצע בעזרתו פעולות
  - ניתן לשמור אותו במשתנה

שימוש בערך המוחזר לביצוע חישוב

```
int main()
{
    double sum = power(2,20) + power(3,15) + power(5,17);

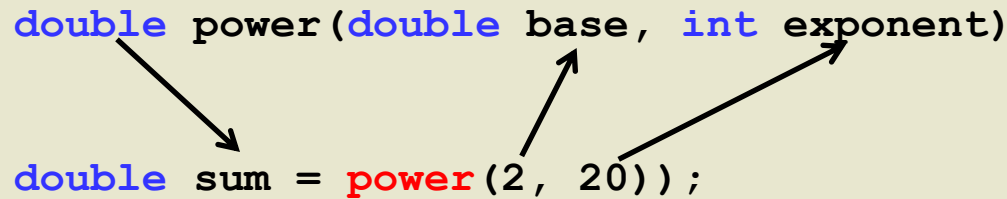
    printf("2^20 + 3^15 + 5^17 = %g", sum);

    return 0;
}
```



# קריאה לפונקציה

```
double power(double base, int exponent)
double sum = power(2, 20);
```



- קריאה לפונקציה:
  - היא פקודה המתבצעת ברצף הפקודות של התוכנית
  - ניתן לקרוא לפונקציה גם מתוך פונקציה אחרת
  - **חייבים** להעביר לפונקציה ערכים בהתאם להגדרת הפונקציה
  - בסיום ריצת הפונקציה, חוזרים למקום שלאחר הקריאה לפונקציה

# דוגמא

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

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

```
{
```

```
    int i = 0;
```

```
    double result = 1;
```

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

```
        result = result*base;
```

```
    return result;
```

```
}
```

```
int main()
```

```
{
```

```
    double sum = power(2,20) + power(3,15) + power(5,17);
```

```
    printf("2^20 + 3^15 + 5^17= %g", sum);
```

```
    return 0;
```

```
}
```

פונקציה המקבלת 2 משתנים,  
אחד מסוג double והשני מסוג  
int

main היא פונקציה

# ללא שימוש במשתנה עזר

```
double power(double base, int exponent)
{
    int i = 0;
    double result = 1;

    for (i = 1; i <= exponent; i++)
        result = result*base;

    return result;
}

int main()
{
    printf("2^20 + 3^15 + 5^17= %g",
        power(2,20) + power(3,15) + power(5,17));
    return 0;
}
```



נעביר את הביטוי ישירות  
כערך לפונקציה printf

# פונקציית main

- פונקציית main:
- כל תוכנית C מכילה פונקציה בשם main
- זו הפונקציה הראשית שבה התוכנית מתחילה
- הפונקציה (התכנית) מופעלת ע"י מערכת ההפעלה
- הערך המוחזר של main:
- מועבר למערכת ההפעלה
- טיפוסו צריך להיות int (או void)
- אם ערכו 0 המשמעות היא שהתוכנית הסתיימה באופן תקין

# פונקציית main

- פונקציית **main**:
  - יכולה לקבל משתנים ממערכת ההפעלה
  - יכולה לפעול גם בלי לקבל משתנים
- לא מומלץ לכתוב פונקציית main ארוכה שהכל נעשה בתוכה
- מקובל:
- לחלק את התוכנית לפונקציות בהתאם לחלקים הלוגיים שלה
- לכתוב פונקציית main קצרה שקוראת לפונקציות האחרות



# משתנים בפונקציה

- **משתנים בפונקציה:**
  - מוכרים אך ורק בפונקציה הזאת (ב-scope שלה)
  - לא מוכרים בפונקציות אחרות (גם לא ב-main)
  - כולל גם את משתני הקלט
- **בסיום ריצת פונקציה:**
  - המשתנים לא מוגדרים יותר
  - ערכי המשתנים לא נשמרים מקריאה אחת לשנייה

# מחסנית הקריאות

```
void g()  
{  
    printf("G\n");  
}  
  
void f()  
{  
    g();  
    printf("F\n");  
}  
  
int main()  
{  
    f();  
    printf("MAIN\n");  
    return 0;  
}
```

Main()  
-> f()

f()  
->g()

g()

מה יודפס על המסך?

# משתנים בפונקציה - דוגמא

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

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

```
{
```

```
    int i = 0;
```

```
    double result = 1;
```

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

```
        result = result * base;
```

```
    return result;
```

```
}
```

```
int main()
```

```
{
```

```
    double sum = power(2,20) + power(3,15) + power(5,17);
```

```
    printf("2^20 + 3^15 + 5^17= %g", sum);
```

```
    return 0;
```

```
}
```

המשתנים:

`i`, `result`, `base`, `exponent`  
מוגדרים אך ורק בפונקציה `power`

המשתנה `sum`

מוגדר אך ורק ב-`main`

# משתנים בפונקציה - דוגמא

```
#include <stdio.h>

double power(double base, int exponent)
{
    int i = 0;
    double result = 1;

    for (i = 1; i <= exponent; ++i)
        result = result * base;

    return result;
}

int main()
{
    double result = power(2,20) + power(3,15) + power(5,17);

    printf("2^20 + 3^15 + 5^17= %g", result);
    return 0;
}
```

אפשר להגדיר בפונקציות שונות משתנים  
שונים באותו השם  
(למשל result)

אין כל קשר בין שני המשתנים!

# פונקציות – העברת ערכים

```
#include <stdio.h>

int square(int num)
{
    num = num * num;
    return num;
}

int main()
{
    int num = 16;

    printf("The square of %d is %d", num, square(num));
    return 0;
}
```

אל פונקציה מועברים ערכים (השמה)  
ולא המיקום בזיכרון, למעשה יש כאן  
העתקה של הערך במשתנה

שינוי ערך המשתנה בפונקציה לא משפיע על  
המשתנה המקורי

**הפלט : The square of 16 is 256**

# בזמן ריצה

## source code

```
double power(double base, int exponent)
{
    int i = 0;
    double result = 1;

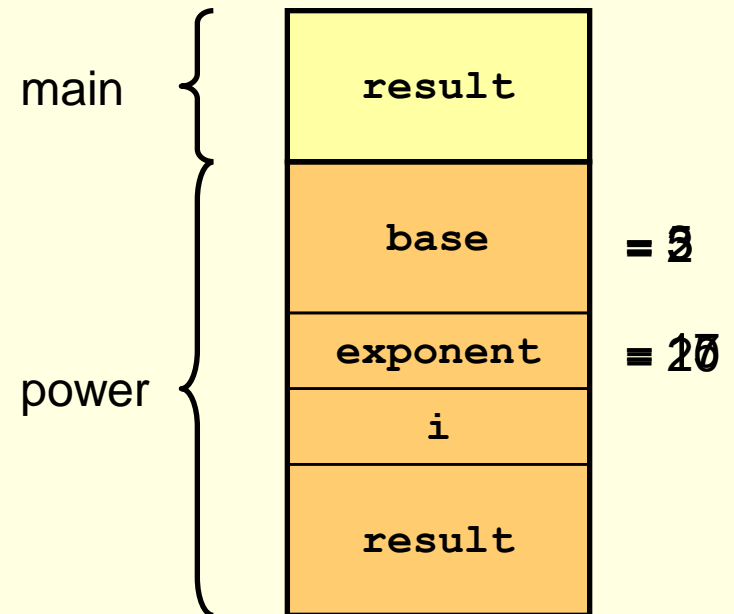
    for (i = 1; i <= exponent; i++)
        result = result * base;

    return result;
}

int main()
{
    double result = power(2,20) +
                    power(3,15) +
                    power(5,17);

    printf("2^20 + 3^15 + 5^17= %g",
           result);
    return 0;
}
```

## call stack



# הכרזה על פונקציות

- על מנת להשתמש בפונקציה, יש להגדירה
- לכן הפונקציה מומשה לפני ה-main
- אופציה נוספת היא:
- להכריז על הפונקציה בתחילת הקובץ (ללא מימוש)  
Function Declaration
- לממש את הפונקציה בהמשך הקובץ, או בקובץ נפרד  
Implementation
- ההכרזה על הפונקציה (prototype) והמימוש חייבים להיות זהים!

# דוגמה להכרזה על פונקציות

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

```
...
```

```
...
```

```
implement something
```

```
...
```

```
...
```

```
implement something else
```

```
...
```

```
...
```

הכרזה על הפונקציה  
(function declaration)

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

```
{
```

```
    int i;
```

```
    double result = 1;
```

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

```
        result = result * base;
```

```
    return result;
```

```
}
```

מימוש הפונקציה  
(function implementation)



# בשביל מה צריך הכרזה על פונקציות?

- לפעמים נוח לראות בתחילת הקובץ אילו פונקציות ממומשות בקובץ
- אם התכנית מורכבת ממספר קבצים, **ההגדרה** של הפונקציות צריכה להופיע רק **באחד מהקבצים**
- בכל **שאר** הקבצים תופיע רק **ההכרזה**
- למשל, שימוש בספריות של פונקציות, כמו `stdio.h`

# הכרזה על פונקציות – ספריות

- `#include <filename>`
- הקומפיילר מצרף לקובץ שלנו את הקובץ `filename`
- כאשר `filename` הוא אחד מקבצי ההכרזות של C
- הקובץ מכיל הכרזות של פונקציות שימושיות
- לדוגמא `stdio.h` מכיל פונקציות קלט / פלט (כמו `printf,scanf`)
  
- כך אנחנו יכולים להשתמש בפונקציות מבלי להכיר את המימוש שלהן
- בזמן הקומפילציה ה-`linker` מצרף את המימוש שלהן (מתוך הספריות הקיימות של C) לתוכנית שלנו

# ספריות נוספות

- ב- C קיימות ספריות נוספות לשימושינו
- ב- **math.h** נוכל למצוא פונקציות כמו:
  - sin
  - cos
  - abs
  - log
  - sqrt
  - ...
- ב- **ctype.h** נוכל למצוא פונקציות שעוסקות בתווים כמו:
  - toupper
  - islower
  - ...

# סיכום ביניים

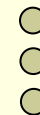
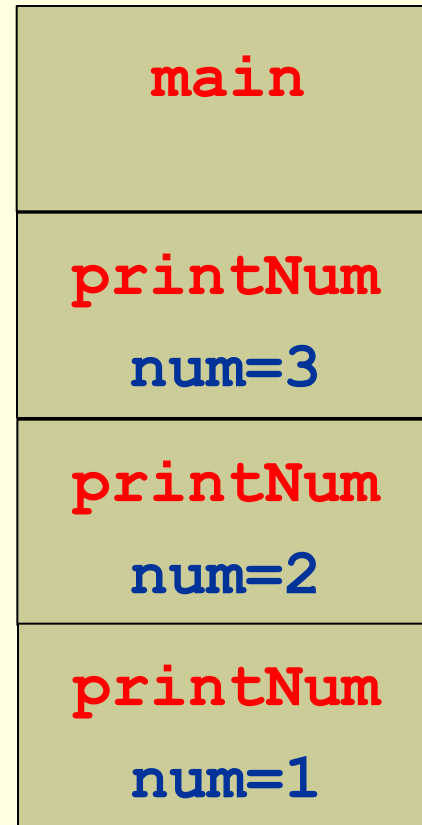
- מהן פונקציות ולמה הן שימושיות
- קריאה לפונקציה והערך שמוחזר ממנה
- משתנים בפונקציות והעברת ערכים אליהן
- הכרזה על פונקציות וספריות של פונקציות

# פונקציה יכולה לקרוא גם לעצמה

• מה תעשה התוכנית הבאה?

```
void printNum(int num)
{
    printf("%d ", num);
    printNum(num-1);
}
```

```
int main()
{
    printNum(3);
    return 0;
}
```



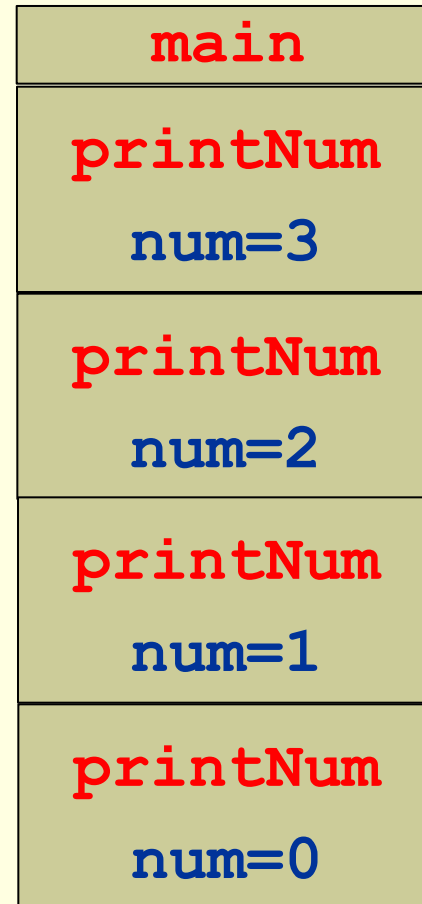
התוכנית "תעוף" כשיגמר הזכרון...

# תנאי עצירה

• מה תעשה התוכנית הבאה?

```
void printNum(int num)
{
    if (num == 0)
        return;
    printf("%d ", num);
    printNum(num-1);
}

int main()
{
    printNum(3);
    return 0;
}
```



# חשוב לקדם משתנה מקריאה לקריאה

• מה תעשה התוכנית הבאה?

```
void printNum(int num)
{
    if (num == 0)
        return;
    printf("%d ", num);
    printNum(num);
}
```

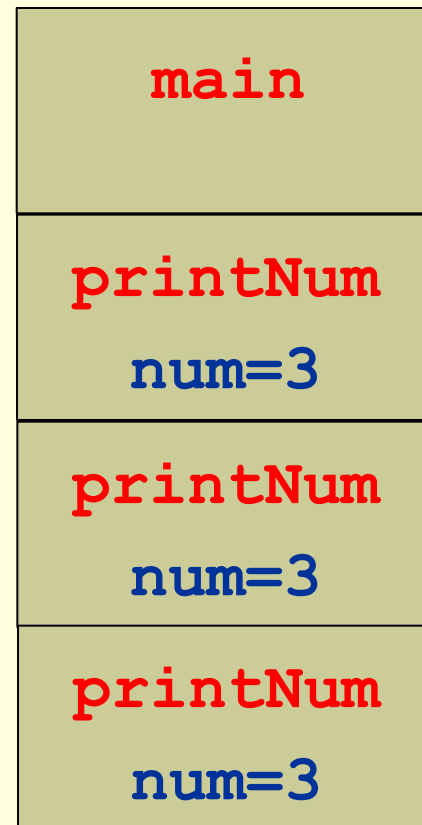
```
int main()
{
```

```
    printNum(3);
```

```
    return 0;
```

```
}
```

התוכנית "תעוף" כשיגמר הזכרון...

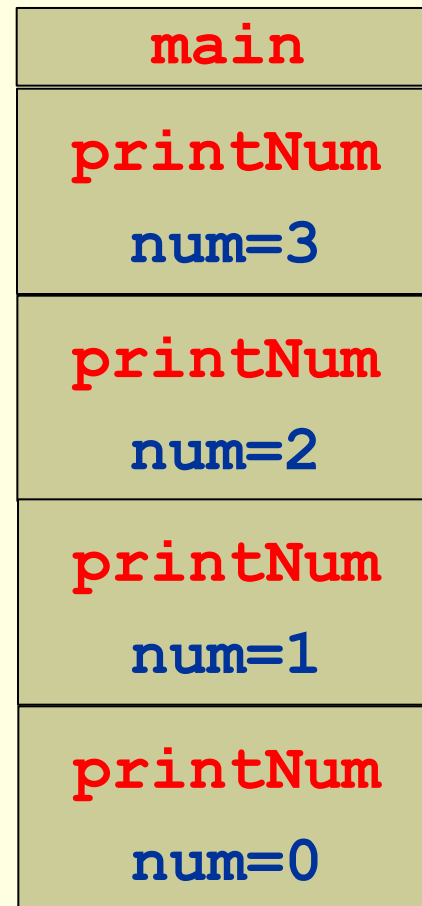


# דוגמא נוספת

• מה תעשה התוכנית הבאה?

```
void printNum(int num)
{
    if (num == 0)
        return;
    printf("%d ", num);
    printNum(num-1);
    printf("%d ", num);
}

int main() {
    printNum(3);
    return 0;
}
```





# רקורסיה

- פונקציה שקוראת לעצמה נקראת פונקציה רקורסיבית.
- פרטים – בשבוע הבא...