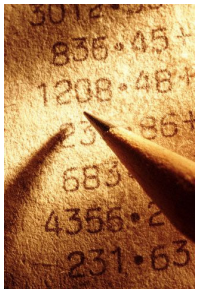
פרק 7
פונקציות

מטרת הפרק

בפרק זה נתכנת משחק מספרים נחמד לשני שחקנים. מהלך המשחק:

- בתחילת המשחק, המחשב מגריל מספר שלם כלשהו, ומציג אותו.
- כל שחקן בתורו צריך לבחור את אחת מספרותיו של המספר.
  - אם הספרה שבחר שונה מ-0, המחשב מאפשר לשחקן לבחור בכמה הוא רוצה להקטין אותה (הוא חייב להקטין אותה לפחות ב-1 ומותר לו להפחית אותה לאפס).
  - אם הספרה היא 0, המחשב מוחק את כל הספרות מימין לספרה שהשחקן בחר, כולל הספרה עצמה.
- השחקן שמביא את המספר לאפס, מפסיד.



משחק לדוגמה

המספר ההתחלתי הוא: **135945**

תור שחקן א'. הוא בוחר את הספרה השנייה (שערכה 4), ומפחית אותה ל-2. **135945** ← **135925**

תור שחקן ב'. הוא בוחר את הספרה החמישית (שערכה 3), ומפחית אותה ל-0. **135925** ← **105925**

תור שחקן א'. הוא בוחר גם כן את הספרה החמישית, שערכה כעת 0. **105925** ← **1**

תור שחקן ב'. הוא חייב לבחור בספרה היחידה שנותרה ולהוריד אותה ל-0. הוא הפסיד... **1** ← **0**

יצירת מספרים אקראיים במחשב

- בוחרים מספר אקראי התחלתי (random seed).
- משנבחר המספר ההתחלתי, ישנן טכניקות שונות ליצירת שרשרת של מספרים אקראיים נוספים לפי דרישה.
- בשפת C מזינים ל-**`srand()`** את המספר האקראי ההתחלתי (בד"כ השעה). זה נעשה <u>פעם אחת בלבד</u>, בתחילת התוכנית.
- בכל פעם שצריך מספר אקראי נוסף, קוראים לפונקציה **`rand()`** המחזירה **`int`** אקראי.
- הפונקציות **`rand()`** ו-**`srand()`** מוגדרות בספרייה **`<stdlib.h>`**.

```
srand(time(0));
x = rand();
…
x = rand();
```

סקיצה של התוכנית

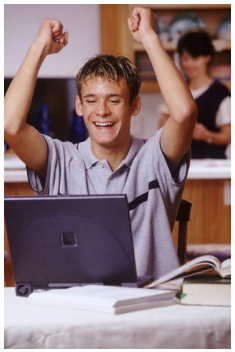
```
srand(time(0));
num = rand();

< display welcome message >

curr_player = 1;
while (num > 0)
{
  < print the game state >
  < get digit index from player >
  < update num according to player's choice >
  curr_player = 3 - curr_player;
}
printf("Player %d has won!\n", curr_player);
```

הפונקציה `time()` מחזירה את הזמן הנוכחי כמספר שלם (בד"כ מספר השניות מאז 00:00 1/1/1970). מוגדרת בספרייה `<time.h>`.

פונקציות

- פונקציה היא קטע קוד שניתן לו שם.
- גורמים לקטע הקוד לפעול על ידי קריאה לפונקציה.
- ניתן להעביר לפונקציה ערכים שבהם היא תעשה שימוש בריצתה. ערכים אלו נקראים פרמטרים של הפונקציה, והם מועברים לפונקציה בזמן הקריאה לה.
- ניתן לקבל מהפונקציה ערך החזרה, שמתקבל כאשר הפונקציה מסיימת לרוץ.

יתרונות השימוש בפונקציות



- **מקל על התכנות**: הבעיה מחולקת למשימות קטנות וקלות יותר לפתרון.
- **מייעל את ה-debugging**: ניתן לבדוק ולתקן כל פונקציה בנפרד, וכך לנפות שגיאות ביתר קלות.
- **שימוש חוזר בקוד**: משימות שיש צורך לבצע פעמים רבות נכתבות ונבדקות פעם אחת בלבד.
- מאפשר קוד קריא יותר ופחות עמוס.

חתימה של פונקציה (prototype)

```
return_type function_name(param1, param2,…);
```

- החתימה של פונקציה מפרטת את הממשק (interface) של הפונקציה – כיצד היא "מתקשרת עם העולם". היא כוללת את הפרטים הבאים:
  - שם הפונקציה (שצריך לקיים את כל הכללים הרגילים עבור מזהים, כפי שראינו בפרק 3).
  - המספר והטיפוס של הערכים שהפונקציה מקבלת כפרמטרים (ניתן, איך אין חובה, לתת שמות לפרמטרים אלו).
  - הטיפוס שהפונקציה מחזירה כשהיא מסיימת לרוץ (נקרא גם "הטיפוס של הפונקציה"). על מנת לציין פונקציה שאינה מחזירה אף ערך, כותבים שהיא מטיפוס `void`.

דוגמאות לחתימות של פונקציות

- פונקציה המחשבת את פעולת הסינוס המתמטית; שם הפונקציה **sin**, היא מקבלת ערך **double** ומחזירה ערך **double**:

```
double sin(double);
```

- פונקציה המקבלת שני משתנים שלמים ומחזירה את המקסימאלי ביניהם:

```
int max(int, int);
```

- פונקציה המקבלת מספר שלם, ומחזירה את ערכו של אחד הביטים בייצוג שלו; בחתימה זו בחרנו לציין שמות לפרמטרים בחתימה, על מנת שיהיה ברור מה המשמעות של כל אחד מהם:

```
int get_bit(int num, int bit_id);
```


אופן העבודה עם פונקציות

1. יש לכתוב את הקוד המלא של הפונקציה במקום כלשהו בקובץ, מחוץ לכל פונקציה אחרת (יש גם אפשרות לעבוד עם ריבוי קבצים, אך לא נדון בכך בקורס זה).

2. על מנת להשתמש בפונקציה, יש לקרוא לה ולהעביר לה את כל הערכים הדרושים לפעולתה.
3. ניתן להשתמש בערך המוחזר של הפונקציה בדיוק כמו בכל משתנה או ביטוי אחר מהטיפוס שהפונקציה מחזירה.
4. לפני הקריאה הראשונה לפונקציה, יש לספק לקומפיילר את החתימה שלה, על מנת שיוכל לוודא את תקינות הקוד שכתבנו.

קריאה לפונקציה

```
function_name(param1, param2,…)
```

- קריאה לפונקציה נעשית על ידי ציון שמה עם סוגריים. אופרטור הסוגריים מציין לקומפיילר שהשם שכתבנו מייצג פונקציה.
- בתוך הסוגריים כותבים את כל הפרמטרים שצריך להעביר לפונקציה (אם יש כאלה). כל פרמטר צריך להיות ביטוי עם ערך כלשהו.
- הפרמטרים הנשלחים לפונקציה לא משתנים כתוצאה מריצת הפונקציה.
- עם סיום ריצתה הפונקציה מחזירה ערך מטיפוס כלשהו (אלא אם הפונקציה היא מטיפוס **void**). ניתן להשתמש בערך החזרה זה, אך ניתן גם להתעלם ממנו. אנו עושים זאת לעיתים קרובות, למשל, כשאנו קוראים ל-**printf()** מבלי להשתמש בערך ההחזרה שלה.

Top-Down Design

החלפנו את הסכימה הכללית שראינו קודם **בקוד C ממשי**, העושה שימוש בקריאות לפונקציות. כל שנותר לנו לעשות הוא לממש את הפונקציות המסומנות – וסיימנו!

```
srand(time(0));
num = rand();

clear_screen();
print_welcome_message();

curr_player = 1;
while (num > 0) {
  print_game_state(num, curr_player);
  dig_id = get_dig_id(num);
  num = update_num(num, dig_id);
  curr_player = 3 - curr_player;
}
printf("Player %d has won!\n", curr_player);
```

דוגמאות לקריאות לפונקציות

```
clear_screen();
print_welcome_message();
```

- פונקציות אלה אינן מקבלות אף פרמטר ואינן מחזירות אף ערך, כלומר ערך ההחזרה שלהן הוא **void**. הפונקציה הראשונה מנקה את המסך, והשנייה מדפיסה הודעת פתיחה קבועה לפני תחילת המשחק.

```
print_game_state(num, curr_player);
```

- פונקציה זו מדפיסה את המצב הנוכחי של המשחק – של מי התור, ומהו המספר כעת. לשם כך עליה לדעת את ערכו של המספר הנוכחי, וכן את מספר השחקן שתורו כעת, ולכן אנו מעבירים לה ערכים אלה כפרמטרים. היא אינה צריכה להחזיר דבר.

דוגמאות לקריאות לפונקציות

```
dig_id = get_dig_id(num);
```

- פונקציה זו קוראת מהמשתמש את האינדקס של הספרה אותה הוא מעוניין לשנות. אנו רוצים שהפונקציה עצמה תדאג לביצוע כל הבדיקות הדרושות, ולכן עליה לדעת מהו המספר הנוכחי. בסופו של דבר היא מחזירה את האינדקס (החוקי) אותו היא קראה, כלומר אין לנו צורך בבדיקות נוספות.

```
num = update_num(num, dig_id);
```

- פונקציה זו מחשבת את הערך החדש של **num** בהתאם לבחירת השחקן. היא מחזירה ערך זה, ואז אנו שמים אותו ב-**num** במקום הערך הישן.

מימוש פונקציה

- מתחילים מכתיבת חתימת הפונקציה, כאשר כאן יש לציין שם מפורש לכל פרמטר, על מנת שיהיה אפשר להתייחס אליו מתוך קוד הפונקציה.
- לאחר מכן בא בלוק הקוד עצמו, בתוך סוגריים מסולסלים **{}**.

```
void print_welcome_message()
{
  printf("Welcome to NumberTournanment 2006!\n\n"
      "Please enter the arena. In a moment you "
      "will be fighting head-to-head\nat the NNDF "
      "(National Number & Digit Federation) "
      "world cup championships.\nThe Number Master "
      "will now choose the golden number;\nyou must "
      "keep this number above zero at all cost.\n"
      "The player who survives the longest wins.\n\n"
      "Ready?\nGo!\n" );
}
```


שימוש במשתנים בתוך פונקציה

- ניתן להשתמש במשתנים בתוך קוד הפונקציה. יש להצהיר עליהם בתוך הפונקציה, לפני תחילת הקוד עצמו.
- בזמן הקריאה לפונקציה, התוכנית מקצה זיכרון זמני למשתנים אלו. המשתנים הללו קיימים רק במהלך ריצת הפונקציה, וכאשר היא מסיימת, הזיכרון שהוקצה להם משתחרר.
- לא ניתן מתוך פונקציה לגשת למשתנים שהוגדרו בפונקציות אחרות או ב-**main()**.

שימוש במשתנים בתוך פונקציה

- הדוגמה הבאה היא פונקציה ל"ניקוי המסך". שימו לב שלמעשה היא אינה באמת מנקה את המסך – היא פשוט מניחה שמספר השורות שנכנסות במסך אחד הוא לכל היותר **SCR_HEIGHT**, והיא "מנקה" אותו על ידי הדפסת תווי **'\n'** בכמות מספקת.

```
#define SCR_HEIGHT 25

void clear_screen()
{
  int i;
  for (i = 0 ; i < SCR_HEIGHT; ++i)
    putchar('\n');
}
```


פונקציות עם פרמטרים

- בכל פעם שמתבצעת קריאה לפונקציה, התוכנית מקצה מקומות בזיכרון עבור הפרמטרים שלה, ואז היא שמה בהם **עותק** של הערכים שהעברנו אליה כפרמטרים. מנגנון פעולה זה נקרא העברת פרמטרים **by value**.
- בתוך הפונקציה עצמה, ניתן להשתמש בפרמטרים כמשתנים לכל דבר, כולל שינוי ערכם. עם זאת, כל שינוי כזה מתבצע רק **על העותק** של הפרמטר (זה שהוקצה בתחילת הקריאה לפונקציה).
- כשריצת הפונקציה מסתיימת, הזיכרון שהוקצה לפרמטרים שלה משתחרר.
- אם הערכים שסיפקנו בקריאה לפונקציה לא תואמים את טיפוסי הפרמטרים שהפונקציה אמורה לקבל, מתבצעת **המרה אוטומטית** שלהם בזמן שהם מועתקים לפרמטרים של הפונקציה – לפי אותם הכללים שלמדנו לגבי השמות רגילות.

דוגמה לפונקציה עם פרמטרים

- נממש את הפונקציה שמדפיסה את מצב המשחק. שימו לב כי אנו משתמשים בפרמטרים של הפונקציה כמשתנים לכל דבר:

```
void print_game_state(unsigned int num, int player)
{
   printf("\n** Player no. %d **\n\n", player);
   printf("The number is: %u\n", num);
}
```

החזרת ערך והפקודה return

- הפקודה **return** מפסיקה את ריצה הפונקציה מיידית, ומחזירה את התוכנית לנקודה שבה הפונקציה נקראה.
- כל פונקציה שאינה מטיפוס **void** חייבת להחזיר ערך כלשהו. בפונקציה כזו, מציינים בפקודת ה-**return** את הערך שאותו הפונקציה תחזיר, באופן הבא:

```
return <value> ;
```

- בפונקציה מטיפוס **void** אין מחזירים ערך, ולכן השימוש בפקודה **return** נראה כך:

```
return ;
```

החזרת ערך והפקודה return

- שאלה: מה אם הפונקציה אינה נתקלת באף פקודת **return** במהלך ריצתה?
- במקרה של פונקציה מטיפוס **void**, אם הפונקציה איננה נתקלת באף **return**, היא מגיעה לסוגר המסולסל **}** שמסיים את הפונקציה, והשליטה מוחזרת לשורה שבה הפונקציה נקראה.
- במקרה של פונקציה שאיננה מטיפוס **void**, הפונקציה חייבת להחזיר ערך כלשהו, ולכן זו **שגיאה** להגיע לסוגר המסיים **}** ללא פקודת **return** כלשהי. כלומר, בסוף כל פונקציה שכזו חייבת לבוא פקודת **return**.

דוגמאות ל-return

נממש את הפונקציה שקוראת מהמשתמש את אינדקס הספרה אותה הוא רוצה להפחית. שימו לב שאנו עושים כאן שימוש בפונקציות עזר שאותן נממש בהמשך:

```
int get_dig_id(unsigned int num)
{
  int num_len, dig_id;

  num_len = num_length(num);
  printf("Please choose a digit index (1..%d): ",
         num_len);
  dig_id = read_value(1, num_len);

  return dig_id;
}
```


Top-Down Design

הנה פירוט הפונקציות שעלינו לממש בהמשך:

- פונקציה המחשבת את מספר הספרות של שלם כלשהו:

```
int num_length(unsigned int num);
```

- פונקציה הקוראת מהמשתמש מספר שלם, ומוודאת שהוא בתחום בין **min** ל-**max**. אם לא, היא ממשיכה לבקש ממנו, עד שהמספר בטווח הנכון:

```
int read_value(int min, int max);
```


פונקציות נוספות עם return

הנה מימוש הפונקציה **num_length()**. את הקוד עצמו כבר ראינו בעבר:

```
int num_length(unsigned int num)
{
  int digit_num = 1;

  while(num > 9) {
    num /= 10;
    digit_num++;
  }
  return digit_num;
}
```

פונקציות נוספות עם return

פונקציה שמקבלת זוג ערכים שלמים **a** ו-**b**, ומחזירה את ערכו של הגדול מביניהם:

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

פונקציה נוספת שנזדקק לה בקרוב מקבלת שני פרמטרים – מספר שלם ואינדקס של ספרה, ומחזירה את ערך הספרה שזה האינדקס שלה:

```
int get_digit(unsigned int num, int digID)
{
  return (num/(int)pow(10,digID-1)) % 10;
}
```


שימוש חוזר בקוד

- הפעולה של קריאת מספר שלם מהמשתמש, ווידוא שהוא בטווח חוקי כלשהו, הינה פעולה שאנו עושים מספר פעמים במהלך התוכנית.
- לפיכך, נכתוב פונקציה שתבצע עבורנו פעולה זו. בכל מקום בתוכנית בו עלינו לקרוא מספר שלם, נשתמש בפונקציה זו במקום בפקודת **scanf()** ישירה.
- הטכניקה המודגמת כאן נפוצה מאוד. הרעיון הוא לבודד את פעולות הקלט (כולל ווידוא החוקיות שלו) מיתר חלקי התוכנית, באמצעות פונקציה ייעודית העוסקת בכך. פונקציה זו תטפל בעצמה בכל הבעיות שעלולות להתעורר במהלך קבלת הקלט, כך שכאשר היא מסיימת לבסוף ומחזירה ערך – מובטח שפעולת הקלט הסתיימה בהצלחה ואף שהערך שנקרא הינו חוקי.

פונקצית הקלט

```
int read_value(int min, int max) {
  int value;

  do {
    if (scanf("%d", &value) != 1) {
      printf("Error reading input! Exiting.\n");
      exit(1);
    }

    if (value >= min && value <= max)
      break;

    printf("invalid value! please try again: ");
  } while (1);

  return value;
}
```

על פונקציה זו בהמשך

שימוש חוזר בפונקצית הקלט

- אנו מוכנים כעת להשלים את הפונקציה **update_num()**.
- זכרו שאם הספרה שנבחרה על ידי השחקן אינה אפס, יש לשאול אותו בכמה להקטין את הספרה שבחר:

```
unsigned int update_num(unsigned int num, int digID)
{
  int dig_val;
  dig_val = get_digit(num, digID);
  if (dig_val == 0) {
    num /= (int)pow(10,digID);
  }
  else {
    printf("Choose amount: (1..%d) ", dig_val);
    num -= read_value(1,dig_val) * pow(10,digID-1);
  }
  return num;
}
```

התוכנית עד כה (1)

סיימנו לממש את **`main()`** ואת כל פונקציות העזר! כך נראה קובץ הקוד כעת:

```
int main() {
  …
}

void print_welcome_message() {
  …
}

void clear_screen() {
  …
}

void print_game_state(unsigned int num, int player) {
  …
}
```


התוכנית עד כה (2)

```
int get_dig_id(unsigned int num) {
  …
}

int num_length(unsigned int num) {
  …
}

int get_digit(unsigned int num, int digID) {
  …
}

int read_value(int min, int max) {
  …
}
```


הצהרה על פונקציות

- קומפילציה של קובץ מתבצעת מלמעלה למטה!
- לכן, כאשר הקומפיילר נתקל בקריאה לפונקציה כלשהי במהלך הקוד, עליו להיות מודע לקיומה של פונקציה זו, ולהכיר את חתימתה, עוד לפני כן (שימו לב שללא הכרת חתימת הפונקציה, הקומפיילר לא יוכל לוודא את תקינות הקריאה לפונקציה).
- בשפת C, פונקציה מוכרת על ידי הקומפיילר החל מהמקום בו אנו **מצהירים** עליה, ועד לסוף אותו קובץ.
- ישנן שתי אפשרויות להצהיר על פונקציה:

1. לכתוב את כל מימוש הפונקציה בקובץ לפני הקריאה הראשונה אליה.
2. להשתמש במנגנון ה- forward declaration.

Forward Declaration

- על פי רוב, האופציה הראשונה (כתיבת מימוש הפונקציה לפני שמשתמשים בה) אינו נוח. כשיש פונקציות רבות, וכל אחת מהן קוראת לפונקציות נוספות, מציאת הסדר הנכון נעשה מסובך. במקרים מסוימים כלל **אין** אף סדר חוקי!
- לפיכך, יש להשתמש במקום זאת במנגנון ה- forward declaration, שפותר בעיות אלו:

1. לפני הקריאה הראשונה לפונקציה, ומחוץ לכל פונקציה אחרת, נכתוב את חתימת הפונקציה, כולל **`;`** (נקודה-פסיק) בסוף. מאותה שורה ואילך, הקומפיילר ידע על קיום הפונקציה, וניתן לקרוא לה.
2. את קוד הפונקציה עצמו ניתן לכתוב כעת בכל מקום שנוח לנו בקובץ.

הצהרות מקדימות בתוכנית שלנו

נוסיף בראש קובץ הקוד שורות המצהירות על כל הפונקציות שלנו.
את המימוש עצמו נכתוב לאחר הפונקציה **main()**.

```
void print_welcome_message();
int read_value(int min, int max);
int get_digit(unsigned int num, int digID);
int num_length(unsigned int);
…

int main() {…}

int get_digit(unsigned int num, int digID) {
  return (num/(int)pow(10,digID-1)) % 10;
}
…
```

Run!

הפונקציה main() והפקודה exit

- כל תוכנית שאנו כותבים חייבת להכיל פונקציה בשם **main()**.
- כאשר אנו מריצים את התוכנית, אנו למעשה מבצעים קריאה לפונקצית **main()** הזו.
- חתימת הפונקציה **main()** היא: `int main();`

כלומר, פונקציה שאינה מקבלת כל פרמטרים, ומחזירה טיפוס **int**.

- כזכור, הפעלת הפקודה **return** מתוך פונקציה כלשהי מפסיקה את פעולת אותה הפונקציה. דבר זה נכון גם עבור הפונקציה **main()** – אולם, כיוון ש-**main()** היא הפונקציה הראשית בתוכנית, הפסקתה **מסיימת את ריצת התוכנית כולה.**

הפונקציה main() והפקודה exit

- הפונקציה **main()** מחזירה ערך מטיפוס **int**. לכן כשמשתמשים ב-**return** בפונקציה **main()**, יש לציין ערך החזרה שלם.
- מבחינה מעשית, אין זה משנה לתוכנית שלנו איזה ערך יוחזר. עם זאת, הערך המוחזר של הפונקציה **main()** מועבר **למערכת ההפעלה**, והיא יכולה להחליט כיצד לנהוג בהתאם לערך זה.
- מקובל שהפונקציה **main()** מחזירה 0 אם התוכנית סיימה את ריצתה בהצלחה, וערך שאיננו 0 (בד"כ 1) אם היא עצרה בגלל תקלה.
- אם ברצוננו להפסיק את ריצת התוכנית מיידית, אבל מתוך פונקציה שאיננה **main()** (מאוד לא מומלץ בתכנות טוב!), יש להשתמש בפונקציה **exit()**. פונקציה זו מפסיקה מייד את ריצת התוכנית, והיא מקבלת כפרמטר את ערך ההחזרה הרצוי של הפונקציה **main()**.

דוגמה לריצת התוכנית (1)

```
Welcome to Number Tournanment 2005!

Please enter the arena. In a moment you will be
fighting head-to-head at the National Number & Digit
Federation (NNDF) world cup championships.
The Number Master will now choose the golden number
for this evening, you must keep this number above
zero at all cost. The player who survives the
longest wins.

Ready?
Go!

*** Player number 1 ***
```

דוגמה לריצת התוכנית (2)

```
The number is: 1213748338
Please choose a digit (1..10): 11
invalid value! please try again: 0
invalid value! please try again: 9
Choose amount to reduce: (1..2) 2

*** Player number 2 ***

The number is: 1013748338
Please choose a digit (1..10): 9

*** Player number 1 ***

The number is: 1
```


דוגמה לריצת התוכנית (3)

```
Please choose a digit (1..1): 1
Choose amount to reduce: (1..1) 5
invalid value! please try again: 0
invalid value! please try again: 1
Player 2 has won!


Thank you for participating in the NNDF
championships. Hope to see you again in 2007!
```


מחלקות אחסון

- עד כה עסקנו רק בתכונה אחת של משתנים: הטיפוס. הטיפוס של המשתנה קבע את גודלו בזיכרון, את אופן הייצוג שלו, ואת הפעולות שניתן להפעיל עליו.
- פרט לטיפוס, למשתנים שאנו מגדירים ב-C קיימת תכונה נוספת, שעד כה לא עסקנו בה: **מחלקת האחסון** שלהם. מחלקת האחסון של משתנה משפיעה על שני פרמטרים:
  - **אורך החיים של המשתנה** – מתי מוקצה הזיכרון בשבילו, ומתי זיכרון זה מפונה.
  - **טווח ההכרה של המשתנה** – היכן בקוד עצמו ניתן לגשת למשתנה.
- כפי שנראה, שתי תכונות אלו אינן מתלכדות ב-C! אנו נראה כי ייתכנו משתנים שקיימים בזיכרון, אך לא ניתן לגשת אליהם.

חלוקת קוד עם בלוקים

- בלוק הוא קטע קוד שתחום על ידי זוג סוגריים מסולסלים **{}**.
- בלוקים מחלקים את התוכנית לחלקים מבחינת שני הפרמטרים שציינו. כלומר, המשתנים שנמצאים בזיכרון, וכן המשתנים שניתן לגשת אליהם, עשויים להשתנות מבלוק לבלוק.
- בלוקים יכולים להיות זרים (כלומר אין ביניהם קשר) או מקוננים (כלומר אחד מהם נמצא בתוך השני, כמו במקרה של לולאה בתוך לולאה).
- דוגמאות לבלוקים:
  - כל פונקציה נכתבת בתוך בלוק.
  - ניתן להשתמש בבלוקים בלולאות ובפקודות התניה.
  - ניתן גם לשים בלוק סתם כך באמצע קוד רציף!

משתנים לוקאליים אוטומטיים

- בתחילת כל בלוק ניתן להצהיר על משתנים חדשים. משתנים אלה (אם לא נצהיר במפורש אחרת) יהיו מסוג **לוקאלי אוטומטי**.
- משתנה לוקאלי אוטומטי מוקצה בזיכרון בכל פעם שנכנסים לבלוק, ומפונה מהזיכרון מייד כאשר יוצאים מהבלוק.
- ניתן לגשת למשתנה כזה רק מתוך הבלוק שבו הוא הוקצה, או בבלוקים מקוננים לו.
- כל המשתנים שהגדרנו עד כה בתוך **`main()`** וביתר הפונקציות היו מסוג זה.
- פירוש השם: **אוטומטי** – כיוון שהמשתנה מוקצה ומפונה אוטומטית במהלך ריצת התוכנית. **לוקאלי** – כיוון שהמשתנה לוקאלי לבלוק שבו הוא הוקצה.

המחסנית (stack)

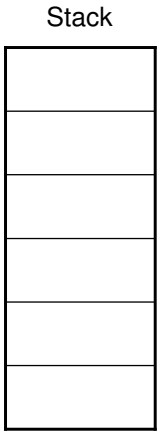
- כל המשתנים האוטומטיים בתוכנית מאוחסנים באזור מסוים בזיכרון שנקרא המחסנית. באזור זה מנוהל תהליך ההקצאה והפינוי של המשתנים בבלוקים השונים.
- הזיכרון מנוהל בשיטה שנקראת Last In First Out – LIFO.
- בתחילת התוכנית, המחסנית ריקה.
- בכל פעם שנכנסים לתוך בלוק, התוכנית מקצה את המשתנים הלוקאליים של הבלוק בראש המחסנית וכך היא גדלה.
- ברגע שיוצאים מבלוק, התוכנית מפנה את כל המשתנים שהוקצו עבור בלוק זה, והמחסנית קטנה בחזרה.

משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```


משתנים אוטומטיים: דוגמה

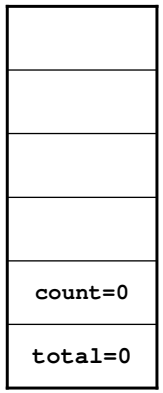
```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

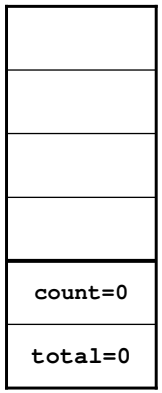
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

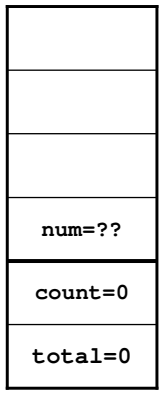
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

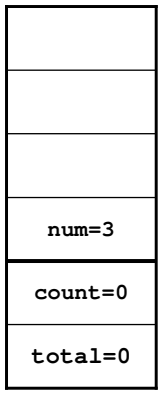
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

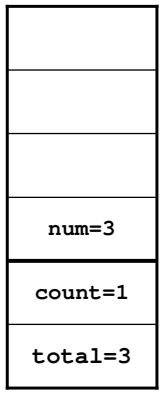
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

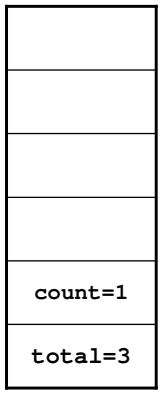
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

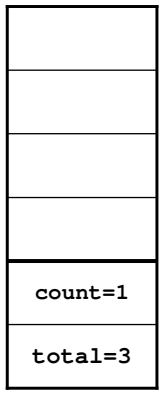
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

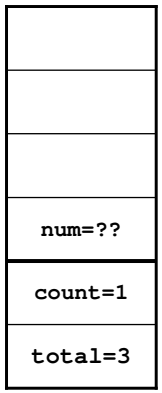
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

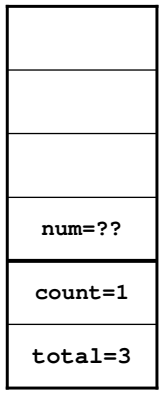
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

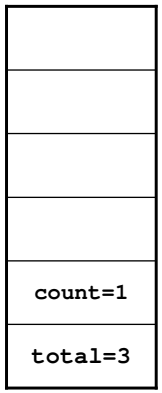
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

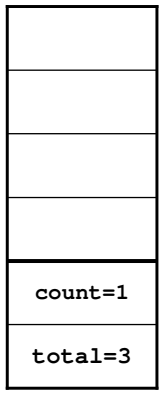
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

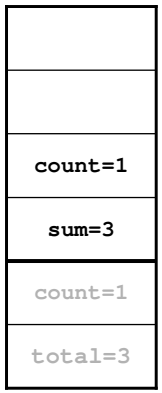
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

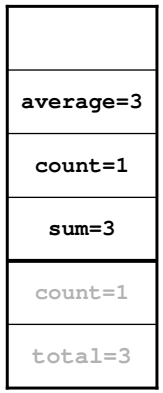
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

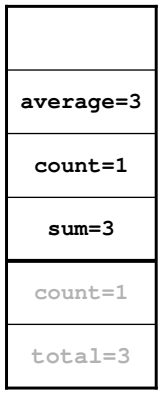
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

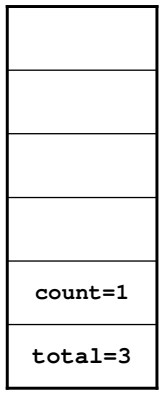
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

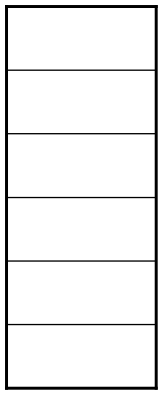
Stack



משתנים אוטומטיים: דוגמה

```
void print_average(int sum, int count) {
  double average = sum/(double)count;
  printf("average = %lf\n", average);
}

int main()
{
  int total=0, count=0;
  while(1) {
    int num;
    if (scanf("%d", &num)<1)
      break;
    total += num;
    count++;
  }
  print_average(total, count);
}
```

Stack



משתנים לוקאליים סטטיים

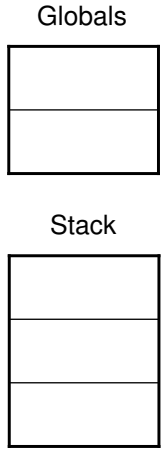
- משתנים אלו מוגדרים אף הם בתחילת בלוק.
- על מנת להגדיר משתנה לוקאלי סטטי בקוד, יש לכתוב בשורת ההצהרה עליו, לפני שם הטיפוס, את המילה השמורה **`static`**.
- משתנה סטטי מוקצה בזיכרון פעם אחת בלבד, מייד בתחילת ריצת התוכנית – ללא קשר לבלוק שבו הוא מוגדר בפועל. זיכרון זה מפונה אך ורק כאשר התוכנית מסתיימת.
- ערכו של משתנה סטטי לוקאלי **נשמר** במהלך כל ריצת התוכנית, גם כשיוצאים מהבלוק או מהפונקציה שלו. אם וכאשר נחזור לאותה פונקציה או בלוק, ערכו הקודם של משתנה זה יישמר.
- למרות שהמשתנה קיים לאורך כל ריצת התוכנית, מבחינת טווח ההכרה שלו הוא מתנהג כמשתנה **לוקאלי** – כלומר המשתנה נגיש רק בתוך הבלוק שבו הוא הוגדר, וכל בלוק מקונן.

אתחול משתנה סטטי

- כל המשתנים הסטטיים מוקצים בתחילת ריצת התוכנית, כולל אלו שההצהרה עליהם נמצאת בבלוקים פנימיים, ואפילו אם הם מוגדרים בפונקציה שלעולם אינה נקראת (!).
- כל משתנה סטטי שמוקצה, חייב להיות מאותחל לערך כלשהו (בניגוד למשתנה אוטומטי שאם הוא אינו מאותחל עשוי להכיל "זבל").
- לפיכך, אם לא ציינו למשתנה סטטי כלשהו ערך אתחול, הוא יאותחל באופן אוטומטי לערך 0 בתחילת התוכנית.
- במידה וציינו ערך אתחול, ערך זה יושם למשתנה מייד בתחילת ריצת התוכנית.כלומר, הכרחי לדעת כבר בתחילת הריצה את ערך האתחול, ולכן מותר לציין ערכי אתחול קבועים בלבד למשתנים סטטיים (בפרט לא ניתן לאתחל לערך של משתנה אחר, או לתוצאה של פונקציה).

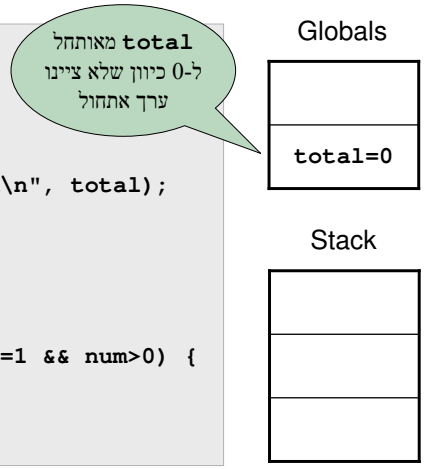
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```



משתנים סטטיים: דוגמה

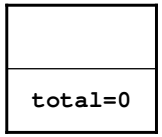
```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```



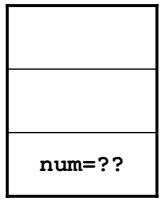
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



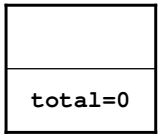
Stack



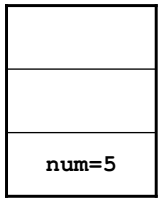
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



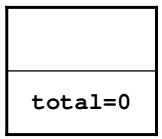
Stack



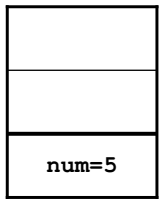
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



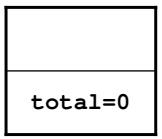
Stack



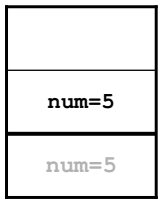
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



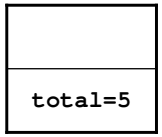
Stack



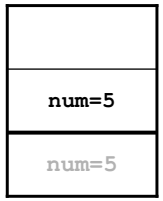
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



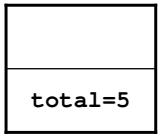
Stack



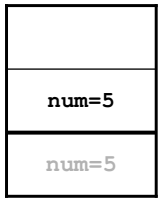
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



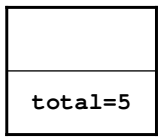
Stack



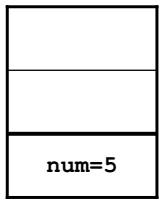
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



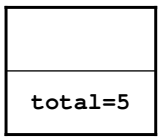
Stack



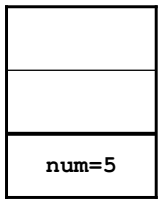
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



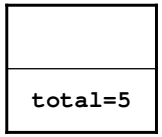
Stack



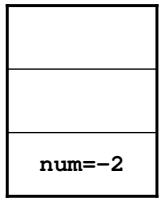
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



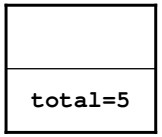
Stack



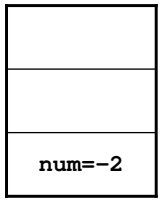
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



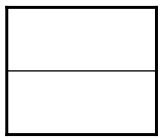
Stack



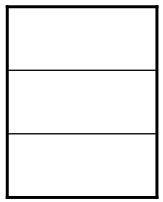
משתנים סטטיים: דוגמה

```
void add_this(int num)
{
  static int total;
  total += num;
  printf("total is now: %d\n", total);
}

int main()
{
  int num;
  while (scanf("%d",&num)==1 && num>0) {
    add_this(num);
  }
}
```

Globals



Stack



שימושים של משתנים לוקאליים סטטיים

- ספירת מספר הפעמים שנכנסנו לבלוק או לפונקציה:

```
void f() {
  static int count;
  ...
  printf("f was used %d times!\n", ++count);
}
```

- הפונקציה **`rand()`** משתמשת במשתנה לוקאלי סטטי על מנת לזכור את הערך האחרון שהיא החזירה.

משתנים גלובאליים

- משתנה גלובאלי מוגדר מחוץ לכל בלוק. כלומר, בפרט גם מחוץ לפונקציה **main()**.
- משתנה גלובאלי מוקצה עם תחילת התוכנית ומפונה עם סיומה, כמו משתנה סטטי. אבל מבחינת טווח ההכרה, הוא מוכר בכל מקום ובכל פונקציה בתוכנית, החל מהשורה שבה הוא מוגדר.

המשתנה **x** איננו מוכר כאן

המשתנה **x** מוכר בכל הפונקציות שבחלק הזה

```
void g() { … }

int x = 3;

int f() { … }

int main() { … }
```

משתנה גלובאלי

משתנים גלובאליים

- אתחול משתנים גלובאליים דומה לזה של משתנים סטטיים – כלומר הם מאותחלים ל-0 אם לא צוין להם ערך אתחול אחר. כמו כן ניתן לציין להם ערכי אתחול קבועים בלבד, כיוון שהם מאותחלים מייד עם תחילת ריצת התוכנית.
- משתמשים במשתנים גלובאליים על מנת להחזיק אינפורמציה שיש צורך בגישה אליה ממקומות רבים בתוכנית, וברצוננו להימנע מהסרבול שבהעברתם כפרמטרים כל הזמן. עם זאת, כאשר נלמד על מצביעים, נראה טכניקה שבדרך כלל טובה יותר לשם כך.
- למעשה, שימוש במשתנים גלובאליים נחשב ברוב המקרים תכנות **רע**, וזאת כיוון שמשתנים גלובאליים מפרים את הרעיון של חלוקת התוכנית לפונקציות ולבעיות נפרדות.

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

Stack

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

num=??
status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=0

Stack

num=8
status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

num=8
status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=??

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

num=??
status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

Stack

num=??
status=0

משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals

sum=8

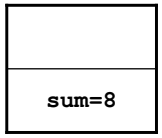
Stack

status=-1

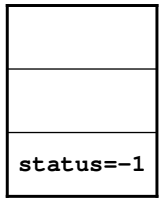
משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals



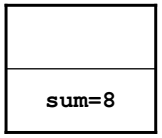
Stack



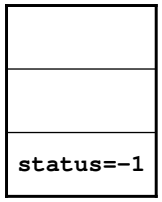
משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals



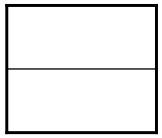
Stack



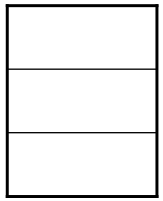
משתנים גלובאליים: דוגמה

```
int sum;

int readnum() {
  int num;
  if (scanf("%d",&num) < 1)
    return -1;
  sum += num;
  return 0;
}

int main() {
  int status;
  do {
    status = readnum();
  } while(status==0);
  printf("sum = %d\n", sum);
}
```

Globals



Stack

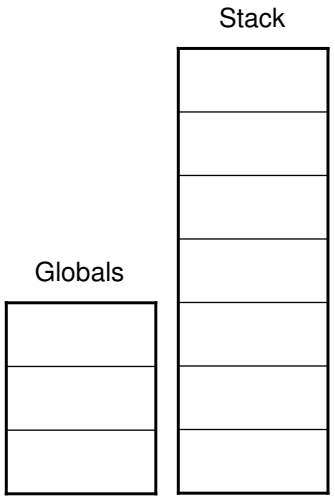


התנגשויות בין שמות של משתנים

- כאשר נכנסים לבלוק כלשהו, ייתכן ויהיו משתנים שכבר מוכרים בבלוק, אפילו אם לא הגדרנו מפורשות אף משתנה! זה קורה כאשר ישנם משתנים גלובאליים, או כאשר הבלוק הוא מקונן ויש משתנים לוקאליים שהוגדרו בבלוק שמעליו.
- במקרים אלו ניתן (אם כי מאוד לא מומלץ) לתת למשתנה לוקאלי שם שכבר מוכר כשמו של משתנה אחר.
- באופן זה אנו פשוט **נחסום** את הגישה למשתנה הקודם מתוך הבלוק הנוכחי, ונוכל לגשת רק למשתנה החדש שהגדרנו. המשתנה הקודם עדיין יתקיים בזיכרון, אך לא יהיה נגיש מתוך בלוק זה. כשנצא מהבלוק, המשתנה הקודם יהיה מוכר שוב.

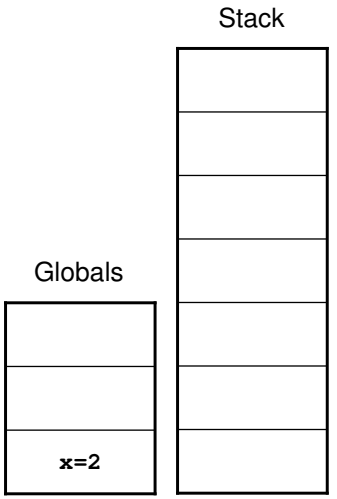
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



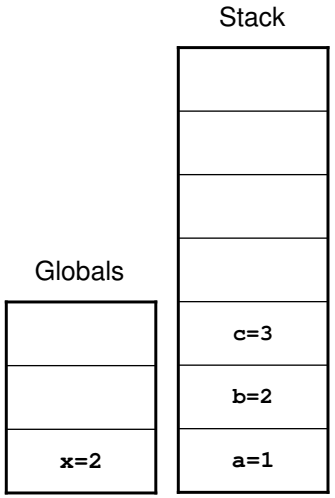
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



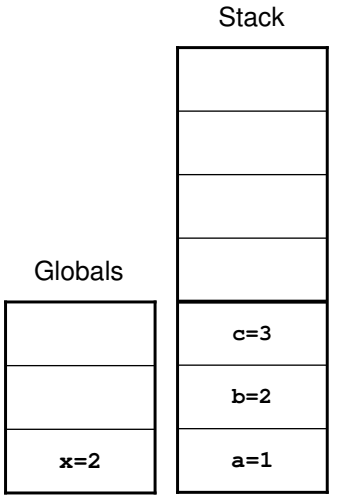
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



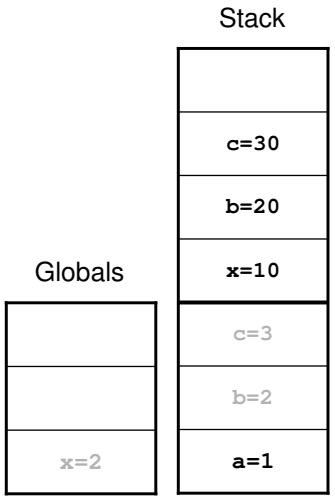
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



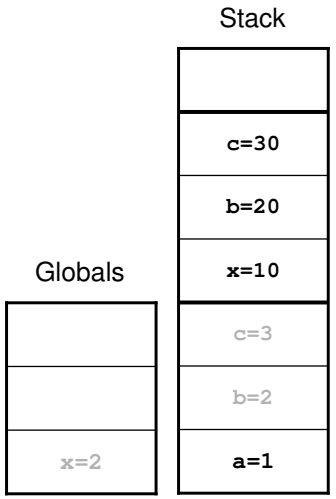
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



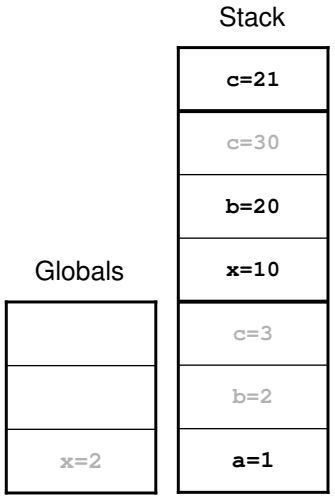
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



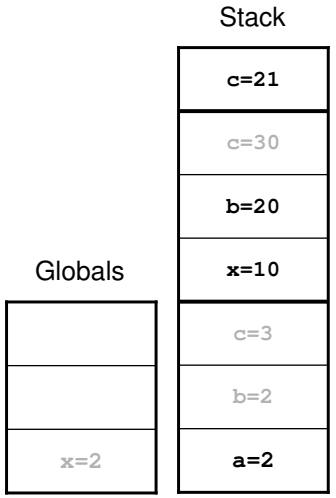
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



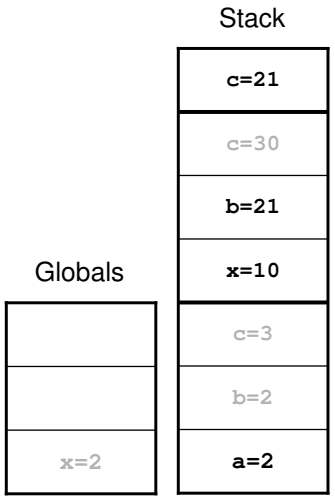
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



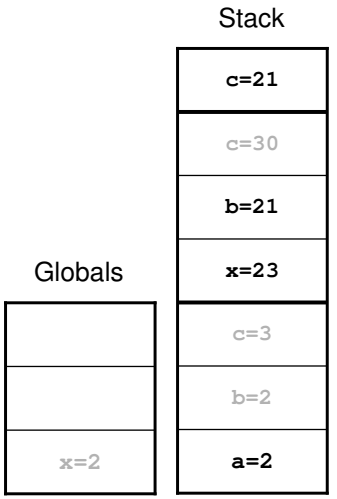
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



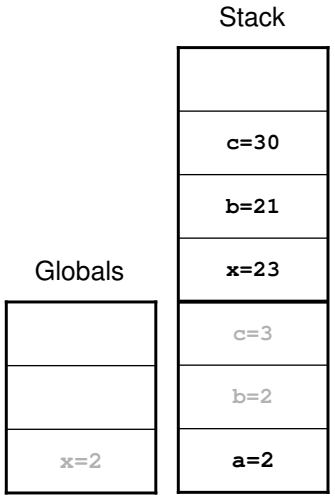
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



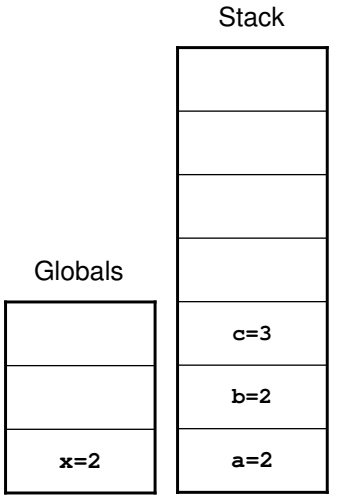
דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```



דוגמה: כמה אתם מרוכזים?

```
int x=2;

int main()
{
  int a=1, b = 2, c = 3;
  {
    int x=10, b=20, c = 30;
    {
      int c = a+b;
      a = 2*a;
      b = c;
      x = a+b;
    }
  }
}
```