# קורס מקוון: *C*

***מרכז ההדרכה 2000***

אתר אינטרנט: www.mh2000.co.il

מבוסס על הספר "***הנדסת תוכנה בשפת C***"

**1. מבוא לתכנות** 3
**2. הכרת שפת C** 41
**3. אבני היסוד** 69
**4. קלט / פלט** 99
**5. אלגוריתמים ומבני בקרה** 121
**6. פונקציות** 143
**7. מערכים** 173
**8. מצביעים** 193
**9. מחרוזות** 215
**10. מבנים** 237
**11. ניהול קבצי התוכנה בפרוייקט** 257
**12. הקצאת זיכרון דינמית ורשימות מקושרות** 275
**13. קלט / פלט עם קבצים** 301
**14. טיפוסי נתונים מופשטים (ADT)** 315

# 1. מבוא לתכנות

- מערכת המחשב
- מבנה המעבד (CPU)
- מבוא לחשבון בינרי
- יצירת תכנית מחשב
- אלגוריתמים
- מבוא לשפת C


מרכז ההדרכה 2000 © כל הזכויות שמורות

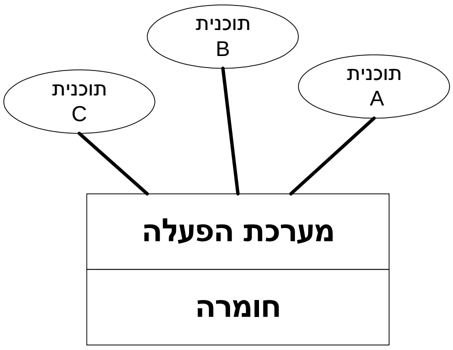
# מערכת המחשב

מחשב הוא מערכת אלקטרונית היכולה לקרוא נתונים ממקור קלט כלשהו, לעבד אותו ולהציג כפלט את תוצאת העיבוד.

העיבוד במחשב מבוצע ע"י תכניות הכתובות בשפה המובנת על ידו.

מערכת המחשב כוללת **חומרה** ו**תוכנה**: ה**חומרה** היא אוסף רכיבים אלקטרוניים ומכניים הפועלים במשולב, כשבמרכזם יחידת העיבוד המרכזית - **המעבד**.

התוכנה במחשב כוללת את ***מערכת ההפעלה*** ותוכניות משתמש. מערכת ההפעלה היא תוכנה המנהלת את החומרה ומספקת ממשק לתפעול המחשב בכללותו. תוכניות המשתמש על סוגיהן מורצות מעל מערכת ההפעלה:





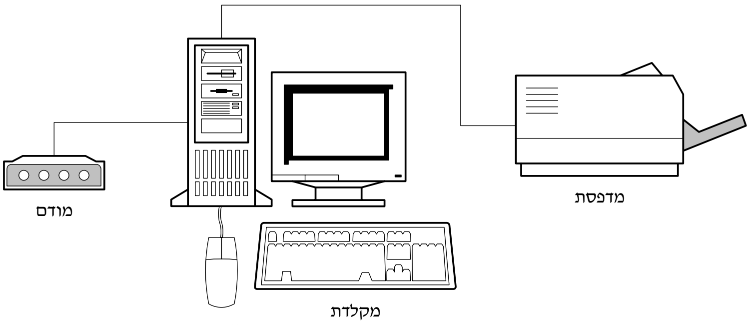
*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## חומרת המחשב

המחשב כולל מספר יחידות חומרה:

- מעבד
- זיכרון ראשי
- כונן קשיח (דיסק), כונן תקליטורים, כונני דיסקטים
- מסך
- מקלדת, עכבר
- מודם
- מדפסת

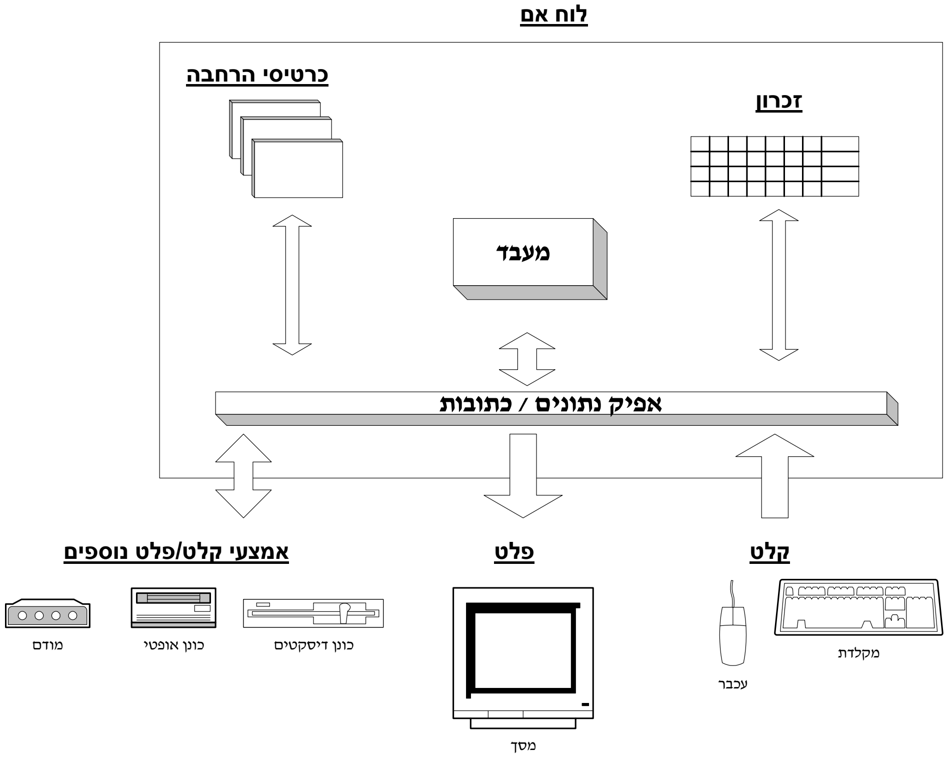
תרשים פיזי של דוגמת מערכת מחשב:





מרכז ההדרכה 2000 © כל הזכויות שמורות

תרשים לוגי:




כפי שניתן לראות בתרשים, המעבד מתקשר עם רכיבי החומרה באמצעות **אפיק הכתובות (Address Bus)** ו**אפיק הנתונים** (**Data Bus**).

**קווי בקרה** נוספים משמשים לתזמון ולבקרת הרכיבים לצורך פעולה משולבת.

על מנת להפעיל את היחידות הנ"ל ולארגן אותן לפעול במשולב מכיל המחשב בנוסף:

- לוח אם: תקשורת ותזמון בין המעבד והיחידות האחרות
- בקרי קלט / פלט: אחראים לפעולות הקלט/פלט בין המעבד וההתקנים הפריפריאליים
- כרטיסים שונים: כ. מסך, כ. מודם, כ. קול וכו'

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## מערכת ההפעלה

מערכת ההפעלה היא תוכנה המפעילה את ציוד המחשב ומספקת שירותים שונים למשתמש כגון: ניהול מערכת הקבצים, ניהול הזיכרון הראשי, הרצת וניהול תכניות ועוד.

התרשים הלוגי הבא מתאר את מיקומה ותפקידה של מערכת ההפעלה במחשב:

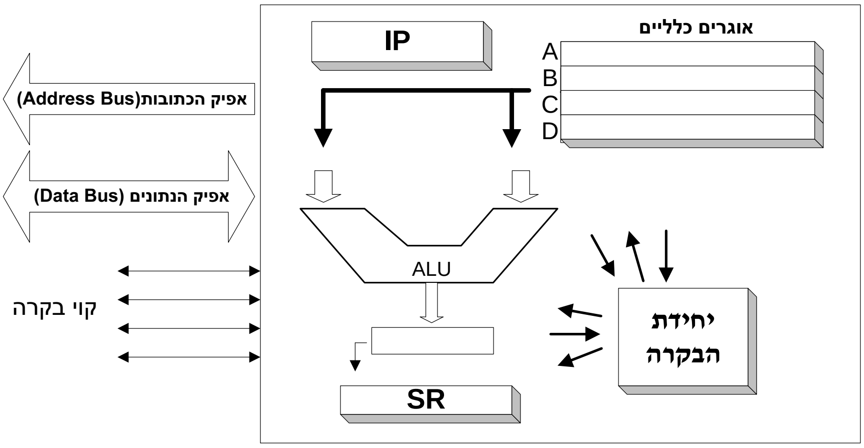
<u>מרכיבי מערכת ההפעלה</u>

- מערכת קבצים - מערכת המאפשרת שמירת מידע בכונן הקשיח כך שישמרו גם לאחר כיבוי המחשב.
- תמיכה בחומרה - הגדרת מנהלי התקנים לרכיבי חומרה.
- בקרת תכניות / תהליכים - ניהול ובקרת התוכנות הרצות במחשב, חלוקת משאבים ביניהן, הגנה על מרחבי הזיכרון שלהן.
- ניהול זיכרון - טיפול בניהול הזיכרון במחשב ובהקצאת זיכרון ליישומים.
- ממשק מערכת ההפעלה למשתמש - ממשק גרפי או טקסטואלי, תכניות שרות.
- ממשק מערכת ההפעלה למתכנת (API = Application Programmer Interface).
- מאפיינים נוספים: תמיכה ברשתות, תמיכה במערכות מרובות מעבדים.

מרכז ההדרכה 2000 © כל הזכויות שמורות 

# *מבנה המעבד (CPU)*

המעבד הוא שם מקוצר ל***יחידת העיבוד המרכזית*** (**CPU**, Central Processing Unit) המפעילה ומנהלת את מכלול החומרה שבמחשב:




המעבד מורכב ממספר יחידות בסיסיות:

- יחידה חשבונית-לוגית (ALU)
- אוגרים כלליים
- אוגר מצב (SR)
- מצביע הפקודה (IP)
- יחידת בקרה

כפי שצוין קודם, **אפיק הכתובות (Address Bus)** ו**אפיק הנתונים** (**Data Bus**) ביחד עם **קווי הבקרה** הם הקשר של המעבד לעולם החיצון, כלומר, לרכיבי החומרה שבמערכת.

נסקור כעת את מרכיבי המעבד, את תפקידיהם וכיצד הם פועלים במשולב.

## יחידה חשבונית-לוגית

יחידה זו היא לב המעבד: היא מבצעת את הפעולות החשבוניות (חיבור, חיסור, כפל וחילוק) ואת פעולות ההשוואה הלוגיות (קטן מ-, גדול מ-, שווה ל-) הנדרשות בביצוע תכנית מחשב.

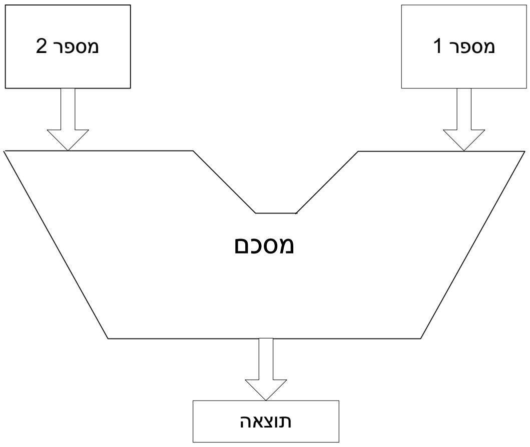
פעולות מורכבות כגון חזקה או חישוב תנאי מורכב נפרטות להוראות החשבוניות/לוגיות הבסיסיות יותר.

מעבדים מודרניים כוללים כיום תתי-יחידות המבצעות פעולות חשבוניות הן בשלמים והן בממשיים. לדוגמא, מסכם (Adder) היא תת-יחידה המבצעת חיבור של שני מספרים, ומכפל (Multiplier) היא תת-יחידה במעבד המבצעת כפל בין שני מספרים.

אנו נבחן כעת את יחידת המסכם שבמעבד מכיוון שהיא שימושית הן לביצוע פעולות חיבור/חיסור, הן לביצוע של פעולות הכפלה (ע"י חיבור חוזר כמספר הכופל) והן לביצוע פעולות לוגיות.

<u>מבנה המסכם (Adder)</u>

המסכם מקבל בשתי כניסותיו שני מספרים, מחשב את סכומם ומציב אותו באוגר תוצאה במוצא שלו:




יחידה זו מאפשרת למעבד לבצע פעולת חיבור בין 2 מספרים. באמצעותה ניתן גם לממש חיסור וכפל:

- <u>חיסור</u>: למימוש X - Y מבצעים X + (-Y).
- <u>כפל</u>: למימוש X · Y מבצעים Y + Y במסכם X פעמים.

## אוגרים כלליים (General Purpose Registers)

האוגרים הכלליים הם רכיבים אלקטרוניים המשמשים לאחסנת מספרים לשם ביצוע חישובים שונים.

כל אוגר מכיל מספר מסוים של **סיביות** שתלוי בסוג המחשב. לדוגמא, מערכת הכוללת 4 רגיסטרים - A,B,C,D:

ערכה של כל אחת מסיביותיו של כל רגיסטר יכול להיות "0" או "1".

## אוגר מצב (Status Register)

באוגר זה מוצב בכל רגע נתון מצב המעבד לאחר הפעולה החשבונית/לוגית האחרונה:

| SR | Z | N | C | ........ |
|---|---|---|---|---|

אוגר זה מתאר בכל אחת מהסיביות שבו את התכונות של התוצאה שהתקבלה במסכם. כל סיבית מתארת מצב פעולה מסוים:

- הסיבית **Z** (Zero) היא בעלת ערך 1 אם תוצאת המסכם בפעולה האחרונה שהתבצעה הייתה 0, אחרת ערך הסיבית 0.
- הסיבית **N** (Negative) היא בעלת ערך 1 אם התוצאה הייתה שלילית, 0 אחרת.
- הסיבית **C** (Carry) בעלת ערך 1 אם לתוצאת הפעולה האחרונה יש **נשא** (שארית).

דוגמא לתרגום קטע תוכנית במעבד:

*if X==Y*

...

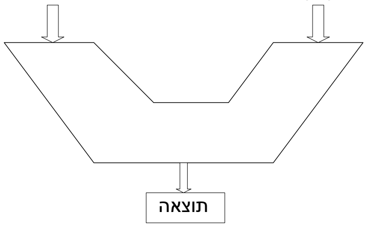
<u>חישוב ביטוי לוגי</u>

ביטוי לוגי הוא ביטוי שתוצאתו היא "אמת" (True) או "שקר" (False). דוגמאות:

| | |
|---|---|
| X < Y | X קטן מ- Y |
| X >= Y | X גדול או שווה ל- Y |
| X == Y | X שווה ל- Y |

**שאלה**: כיצד המעבד בודק האם X == Y?

1) המעבד מבצע חיסור של Y מ- X : (אפשר גם להפך)



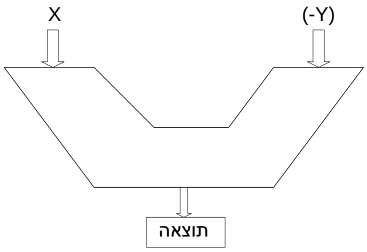

2) המעבד בודק את סיבית Z שבאוגר המצב:

– אם ערכה 1 אזי X שווה ל- Y וערך הביטוי "אמת".

– אחרת ערך הביטוי "שקר", כלומר X > Y או X < Y.

בדומה, הבדיקה האם X > Y מבוצעת כך:

1) המעבד מבצע חיסור של Y מ- X:




2) המעבד בודק את סיביות Z ו- N שבאוגר המצב:

– אם ערכי שתי הסיביות 0 אזי X גדול מ- Y וערך הביטוי "אמת".

– אחרת ערך הביטוי "שקר", כלומר X==Y או X < Y.

***תרגיל: כיצד יבדוק המעבד את ערכו של הביטוי הבוליאני X >= Y ?***

## מבנה הוראת מכונה

תכניות מתורגמות ע"י המהדר לשפת מכונה. שפת מכונה מורכבת מסדרת הוראות מכונה המבוצעות בזו אחר זו לפי הסדר המוכתב ע"י התוכנית.

בדוגמאות הבאות נניח שנתון מעבד עם שפת מכונה שבה הוראות בעלות שלושה מרכיבים:

1. קוד הפעולה (opcode) - מספר המציין למעבד מהי הפעולה. (במעבד ניתן קוד לכל פעולה).
2. נתון 1 - האופרנד השמאלי של הפעולה, שבו גם מוצבת תוצאת הפעולה. לכן נתון זה יכול להיות אוגר או תא זיכרון, אך לא מספר.
3. נתון 2 - האופרנד הימני של הפעולה. יכול להיות אוגר, תא זכרון או מספר.

<u>דוגמאות</u>

1. נניח שפעולת החיבור בעלת קוד 5, ונניח שאנו רוצים להוסיף לתוכן אוגר B את הערך 3: $B \leftarrow B+3$. ההוראה בשפת מכונה:

| | | |
|---|---|---|
| 5 | B | 3 |

2. נניח שקוד פעולת החיסור הוא 8. לביצוע 6 - A תבוצע הוראת המכונה:

| | | |
|---|---|---|
| 8 | A | 6 |

3. נניח שפעולת ההעתקה/הצבה בעלת קוד 6. אנו רוצים לבצע את הפעולה $C \leftarrow A$.

| | | |
|---|---|---|
| 6 | C | A |

4. נדרש לבצע 2 פקודות:

הוספת 21 לתוכן אוגר A: $A \leftarrow A + 21$.

העתקת תוכן אוגר A לאוגר C: $C \leftarrow A$.

ההוראות:

| | | | |
|---|---|---|---|
| 1. | 5 | A | 21 |
| 2. | 6 | C | A |

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 3-1 שבעמ 9.***


מרכז ההדרכה 2000 © כל הזכויות שמורות

## אוגר מצביע ההוראות (Instruction Pointer)

אוגר מצביע ההוראות מציין מהי ההוראה הבאה בתור לביצוע.

בדרך כלל ההוראות מבוצעות לפי סדר הופעתן, אולם ייתכנו דילוגים (ע"י הוראות דילוג) לכתובות כלשהן. מצביע ההוראות מכיל את כתובתם בזיכרון (ראה/י להלן).

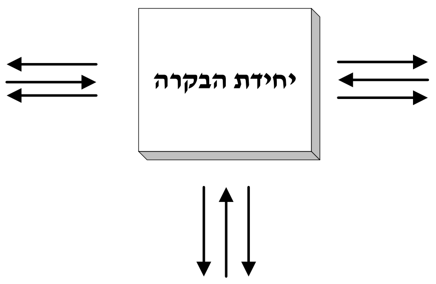
לדוגמא, נניח שגודל כל הוראה בזיכרון הוא 12 בתים (כלומר 4 בתים לאופקוד, ו- 4 בתים לכל אופרנד). הוראות שפת המכונה בזיכרון יכולות להיראות כך:

| **אופרנדים** | **opcode** | **ערך מצביע ההוראה (IP)** |
|---|---|---|
| A , 8 | 5 | 1000 |
| A , B | 6 | 1012 |
| D , 2 | 8 | 1024 |
| D , C | 5 | 1036 |
| | : | |
| | : | |

ה- IP הוא הערך שבעמודה השמאלית, והוא מכיל את כתובות ההוראות בזיכרון. באמצעות ה-IP מיישמים דילוג (jump) לכתובת לא עוקבת.

## יחידת הבקרה (Control Unit)

יחידת הבקרה מכוונת ומתזמנת את פעולת היחידות השונות במעבד. היא מבצעת זאת ע"י שליחת וקבלת אותות אל/מהיחידות:




ביחידת הבקרה נמצא המפענח - אחראי לתרגום פקודת המכונה לרצף של אותות בקרה ליחידות המתאימות במעבד, המחוללים את ביצוע הפקודה בפועל.

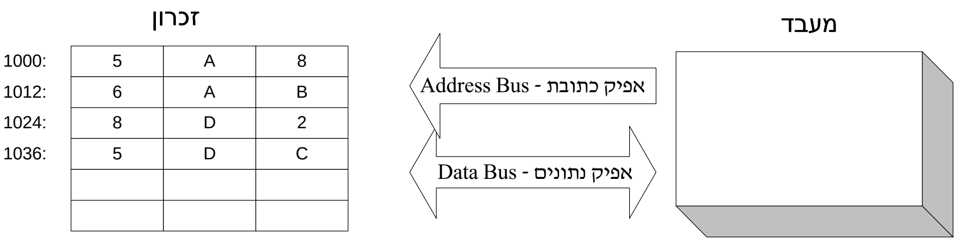
 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הזיכרון (Memory)

הזיכרון הוא אוסף של תאים שבהם נמצא קוד התכנית לביצוע, ערכי הנתונים ומידע נוסף.

לכל תא בזיכרון יש **כתובת** (Address) ודרכה טוענים אותו למעבד.

על מנת לטעון נתון מהזיכרון, המעבד כותב באפיק הכתובות (Address Bus) את כתובתו, וכתוצאה מכך הנתון נטען לאפיק הנתונים (Data Bus):




בדוגמא שבתרשים, ההנחה היא שכל מרכיב בהוראה - קוד ההוראה, הנתון הראשון והנתון השני - הם כל אחד בגודל 4 בתים. לכן כל הוראה תופסת 12 בתים בזיכרון.

כאשר יש צורך בביצוע הוראה מסוימת, יחידת הבקרה שבמעבד מעבירה על אפיק הכתובות את כתובת ההוראה לביצוע, ובתגובה, יחידת הזיכרון טוענת לאפיק הנתונים את ההוראה המתאימה.

המעבד אז קורא את ההוראה מאפיק הנתונים ומבצע אותה.

לדוגמא, נניח שהמעבד סיים לבצע את ההוראה שבכתובת 1012:

| | | |
|---|---|---|
| B | A | 6 |

בכדי לבצע את ההוראה הבאה, נדרשות 3 טעינות של נתונים מהזיכרון:

| | הכתובת שמעביר המעבד באפיק הכתובות | הנתון המוחזר מהזיכרון על אפיק הנתונים |
|---|---|---|
| 1. | 1024 | 8 |
| 2. | 1028 | D |
| 3. | 1032 | 2 |

ולאחר טעינת כל שלושת מרכיבי ההוראה, המעבד מוכן לבצעה:

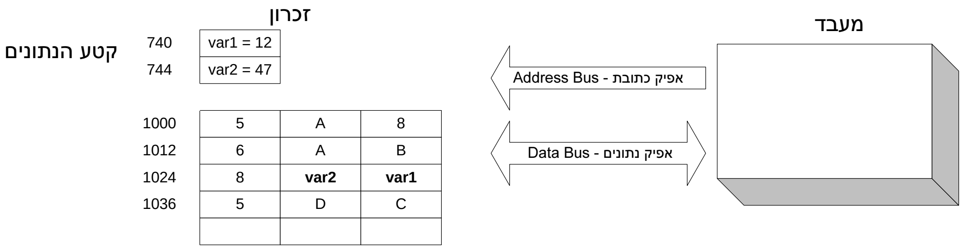
| | | |
|---|---|---|
| 2 | D | 8 |

## קטע הנתונים (Data Segment) וקטע קוד התוכנית (Code Segment)

תוכניות מחשב מחולקות לשני קטעים עיקריים: קטע הנתונים וקטע קוד התכנית.

כאשר תכנית נטענת לזיכרון לצורך ביצוע, מוקצים לה שני קטעי זיכרון עבור הנתונים ועבור הוראות התכנית.

דוגמא לתכנית:




התכנית כוללת שני משתנים בקטע הנתונים, var1 ו- var2 - אלו הם שני שמות שניתנו ע"י כותב/ת התכנית. הם מייצגים תאים בזיכרון שניתן **לכתוב** אליהם **ולקרוא** את ערכם.

> *הערה: המעבד יכול לקרוא או לכתוב לזיכרון. לכן אפיק הנתונים שבתרשים מסומן ע"י חץ דו-כווני.*

לדוגמא, בביצוע ההוראה המתחילה מכתובת 1024, המעבד מבצע פעולת החסרה של var1 מ- var2 ומציב את התוצאה ב- var2:

| | | |
|---|---|---|
| var1 | var2 | 8 |

לצורך ביצוע הוראה זו, על המעבד ראשית להביא את ערכם של המשתנים var1 ו- var2 מקטע הנתונים שבזיכרון, לבצע את החישוב ולאחר מכן לעדכן את ערכו של var2 ע"י כתיבתו לזיכרון.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

סדרת הפעולות הנדרשת היא לכן:

| | **הכתובת שמעביר המעבד באפיק הכתובות** | **הנתון המועבר על אפיק הנתונים** | **כתיבה או קריאה של הנתון?** |
|---|---|---|---|
| 1. | 1024 | 8 | קריאה |
| 2. | 1028 | var2 (744) | קריאה |
| 3. | 744 | 47 | קריאה |
| 4. | 1032 | var1 (740) | קריאה |
| 5. | 740 | 12 | קריאה |
| 6. | 744 | 35 | כתיבה |

הסבר: המעבד קורא את ראשית את קוד ההוראה, 8, המציין פעולת חיסור. לאחר מכן, הוא קורא את הנתון הראשון, var2, שהוא כתובת של תא בקטע הנתונים, כתובת 744.

מכיוון שכך, המעבד מעביר הוראת קריאה נוספת של הנתון עצמו היושב בכתובת 744, הערך 47.

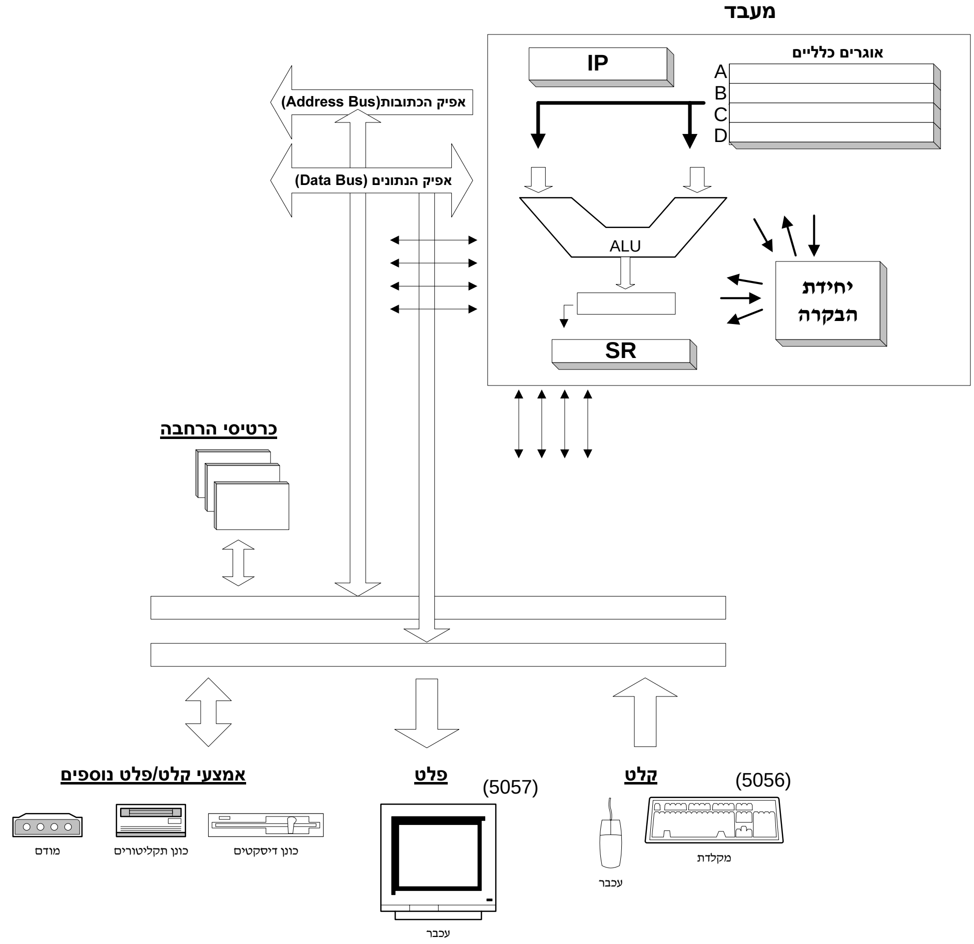
באופן דומה נקרא ערכו של המשתנה var1 מכתובת 740. לאחר השלמת קריאת ערכי האופרנדים של ההוראה, היא מבוצעת: var2← var2 - var1, כלומר, var2 מקבל את הערך 35.

לכן הפעולה האחרונה של המעבד היא אחסון התוצאה בכתובת המשתנה var2, כלומר בכתובת 744.

# רכיבי קלט פלט

בנוסף לגישה לזיכרון, המעבד יכול לפנות לרכיבי קלט פלט. לדוגמא, תכנית מחשב טיפוסית יכולה לבצע קריאה של ערך נתון בזיכרון ע"י <u>קלט מהמקלדת</u>, לבצע אתו חישוב מסוים ולהדפיסו <u>כפלט למסך</u>.

אלו רק שתי דוגמאות להתקני קלט/פלט: מרכיבים שכיחים נוספים הם כונן קשיח, כונן דיסקטים, כונן תקליטורים, מודם:




לשם ביצוע קלט/פלט , בדומה לגישה לזיכרון, המעבד מעביר על אפיק הכתובות את "כתובת" רכיב הקלט/פלט שאליו הוא פונה.

אם לדוגמא נדרש לקרוא נתון מהמקלדת, וכתובת המקלדת היא 5056, המעבד יציב כתובת זו על אפיק הכתובות, ובתגובה רכיב החומרה שבמקלדת יעביר את הנתון שהקליד המשתמש על אפיק הנתונים.

באופן דומה, אם כתובת המסך היא 5057, המעבד ידפיס נתון למסך ע"י הצבת הנתון על אפיק הנתונים והצבת הכתובת 5057 על אפיק הכתובות.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

תכנית הדוגמא הבאה קוראת מהמקלדת (כתובת 5056) ע"י הוראת קליטה (קוד 30) לרגיסטר A, לאחר מכן מכפילה אותו ב- 2 ע"י הוראת חיבור עצמי (קוד 5), ולבסוף מדפיסה אותו למסך ע"י הוראת הדפסה (קוד 31):

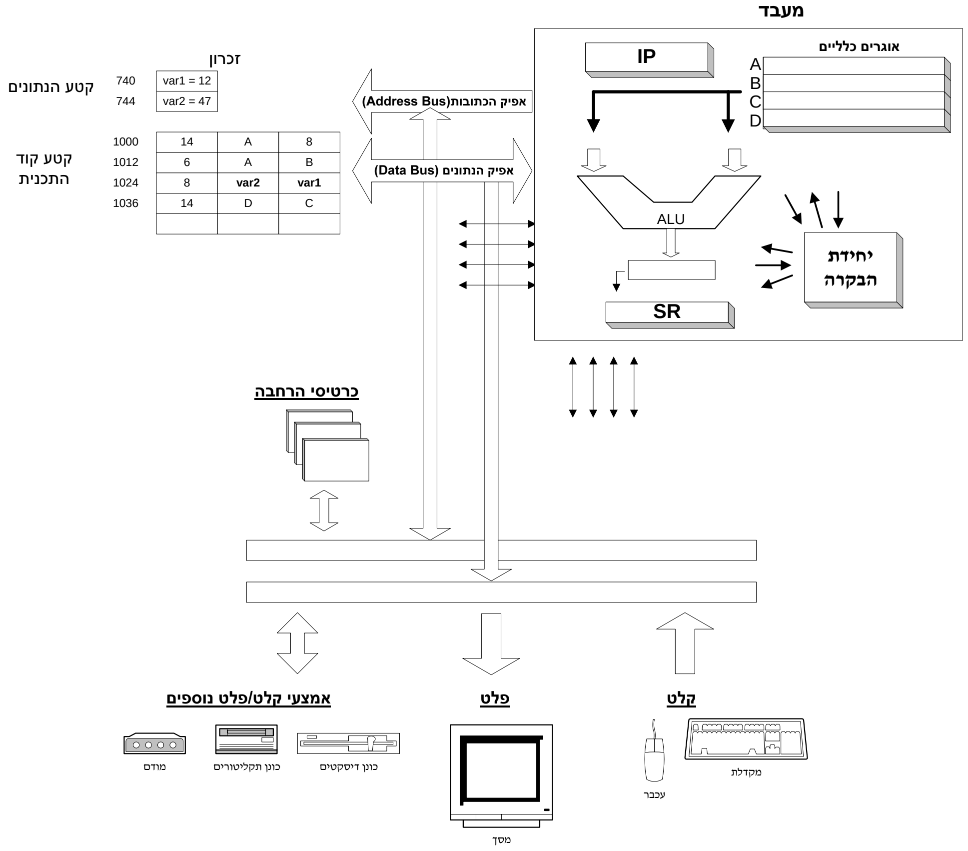
| **אופרנד 2** | **אופרנד 1** | **קוד הוראה** |
|---|---|---|
| A | keyboard (5056) | 30 |
| A | A | 5 |
| A | screen (5057) | 31 |

*הערה: לצורך הפרדה בין כתובות בזיכרון לבין כתובות התקני קלט/פלט קיים קו בקרה היוצא מהמעבד המציין אם הפניה באפיק הכתובות היא ל**מרחב הזיכרון** או ל**מרחב הקלט/פלט**.*


מרכז ההדרכה 2000 © כל הזכויות שמורות

## המבנה הכולל של המעבד, הזיכרון והתקני הקלט/פלט

לסיכום, התרשים הבא מציג את המבנה הכולל של כל מרכיבי מערכת המחשב שהכרנו: המעבד, הזיכרון והתקני הקלט/פלט:




<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 2-1 שבעמ' 14.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## שפת אסמבלר

מכיוון שקשה לבני אדם להבין את שפת המכונה, קיים ייצוג שמי של הוראות המכונה בשפה הנקראת **אסמבלר**: בשפה זו יש שמות לפקודות השונות, וכך הן הופכות למובנות לעין האנושית.

לדוגמא, שפת אסמבלר מתאימה לדוגמאות שלעיל, תספק שמות לפקודות:

| **שם הפקודה** | **קוד** | **משמעות** |
|---|---|---|
| **add** | 5 | חיבור |
| **sub** | 8 | חיסור |
| **move** | 6 | הצבה |
| **in** | 30 | קלט |
| **out** | 31 | פלט |

כמו כן, ניתנים שמות להתקני הקלט פלט: לדוגמא, התקן המקלדת ייקרא keyboard והתקן המסך screen. תכנית אסמבלר לדוגמא תראה כך:

```
_DATA
data1 = 13
data2 = 6

_CODE
add       data2, data1
move      A, data2
in        keyboard, B
sub       A, B
out             screen, A
```

<u>הסבר</u>: בתכנית האסמבלר מוגדרים שני קטעי התכנית - תחילת קטע הנתונים מצוינת ע"י DATA_ ותחילת קטע קוד התכנית ע"י CODE_.

בקטע הנתונים מוגדרים שני משתנים עם ערכים התחלתיים, ובקטע הקוד הוראות בשפת אסמבלר.

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 15.***

# מבוא לחשבון בינרי

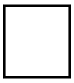
יחידת המידע הבסיסית במחשב היא **סיבית** (**Bit**). ערכה של הסיבית יכול להיות 0 או 1. יחידת המידע **בית** (**Byte**) היא סידרה של 8 סיביות מקובצות יחדיו. לדוגמא:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

יחידת המידע **מילה** (**Word**) היא בעלת גודל משתנה ממחשב למחשב. גדלים מקובלים הם 2 בתים (16 סיביות), 4 בתים (32 סיביות) ויש מחשבים בעלי מילה באורך 8 בתים (64 סיביות).

כמה ערכים ניתן לייצג ע"י בית?

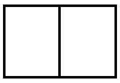
- 1 סיבית ==> 2 ערכים:



0

1

- 2 סיביות ==> 4 ערכים:



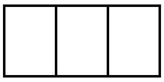
00

01

10

11

- 3 סיביות ==> 8 ערכים:



| | |
|---|---|
| 000 | 100 |
| 001 | 101 |
| 010 | 110 |
| 011 | 111 |

באופן כללי, ב- n סיביות ניתן לייצג $2^n$ ערכים. בהתאם לכך, ב- 8 סיביות ניתן לייצג $2^8$ ערכים, כלומר 256.

## יחידות בתים: KB, MB ו- GB

ערך החזקה $2^n$ כאשר $n=10, 20, 30, 40$ ... הוא קרוב למספר עשרוני בכפולות של 10, ונהוג לתת שמות מקוצרים לערכים אלו. דוגמאות:

- $2^{10}$ הוא 1024 (קרוב ל- 1000) ונקרא בקיצור KB (Kilo-Byte)
- $2^{20}$ הוא 1048576 (קרוב ל- 1000000) ונקרא בקיצור MB (Mega-Byte)
- $2^{30}$ הוא 1073741824 (קרוב ל- 1000000) ונקרא בקיצור GB (Giga-Byte)

יחידות אלו מקילות בהתייחסות לערכים גדולים של חזקות של 2. לדוגמא, החזקה $2^{14}$ ניתנת לביטוי פשוט כ- $2^{10} * 2^4 = 16KB$. באופן דומה, $2^{32}$ מבוטא כ- $2^{30} * 2^2 = 4GB$.

## ייצוג מספרים עשרוני

כיצד אנחנו מייצגים מספר עשרוני? נרשום את ערך המספר 1692 כסכום של ערכים בחזקות של 10:

$$1692 = 1*1000 + 6*100 + 9*10 + 2*1$$

$$= 1*10^3 + 6*10^2 + 9*10^1 + 2*10^0$$

בייצוג העשרוני קיימות 10 ספרות 9 ...,2, 1, 0 לכן אנו אומרים שהן מיוצגות בבסיס 10.

מיקומה של ספרה במספר קובע את הערך שהיא מייצגת בו.

חיבור מספרים עשרוניים מבוצע כך:

| | | | | |
|---|---|---|---|---|
| נשא: | | 1 | | |
| | 3 | 5 | 0 | 1 |
| | 3 | 7 | 4 | + |
| | 6 | 2 | 5 | 1 |

## ייצוג מספרים בינרי

בייצוג בינרי קיימות רק 2 ספרות לכן נאמר שהן מיוצגות בבסיס 2. לדוגמא, המספרים הבאים הם בינריים:

1, 101, 100100, 1111001001

כיצד מתורגם מספר בינרי למספר עשרוני? ניקח לדוגמא את המספר הבינרי 0110:

$$0110 = 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$$

$$= 0 + 4 + 2 + 0$$

$$= 6$$

כלומר $0110_2 = 6_{10}$

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-6 שבעמ' 18.***

## חיבור מספרים בינריים

נרשום את הערך העשרוני של כל המספרים המיוצגים ע"י מספר בינרי בעל 4 ספרות:

| | | | |
|---|---|---|---|
| 1100 = 12 | 1000 = 8 | 0100 = 4 | 0000 = 0 |
| 1101 = 13 | 1001 = 9 | 0101 = 5 | 0001 = 1 |
| 1110 = 14 | 1010 = 10 | 0110 = 6 | 0010 = 2 |
| 1111 = 15 | 1011 = 11 | 0111 = 7 | 0011 = 3 |

כללי החיבור בבסיס 2:

0 + 0 = 0

1 + 0 = 1

1 + 1 = 1 0

1 + 1 + 1 = 1 1

חיבור מספרים בינריים מבוצע בדומה לחיבור מספרים עשרוניים:

נשא: 1 1 1 1

```
   0 1 1 0 1 0 1 1
 +         1 1 0 1
 -----------------
   0 1 1 1 1 0 0 0
```

## הפיכה מייצוג עשרוני לייצוג בינרי

כדי להפוך מספר המיוצג בשיטה העשרונית למספר בייצוג בינרי צריך לבצע את התהליך הבא:

1. נתון המספר העשרוני X.
2. חלק את X ב- 2 ורשום את השארית.
3. חלק את המנה שהתקבלה ב- 2 ורשום את השארית החדשה.
4. חזור על פעולה 3 עד לקבלת מנה 0.
5. המספר בייצוג בינרי הוא סידרת השאריות בסדר כתיבה הפוך.

לדוגמא, נמיר את המספר העשרוני 137 למספר בינרי:

| שארית | מנה | | |
|---|---|---|---|
| 1 | 68 | = | 137 / 2 |
| 0 | 34 | = | 68 / 2 |
| 0 | 17 | = | 34 / 2 |
| 1 | 8 | = | 17 / 2 |
| 0 | 4 | = | 8 / 2 |
| 0 | 2 | = | 4 / 2 |
| 0 | 1 | = | 2 / 2 |
| 1 | 0 | = | 1 / 2 |

המספר הבינרי שהתקבל: 10001001

תרגול

***קרא/י סעיף זה בספר ובצע/י את תר' 1-4 שבעמ' 19.***

## ייצוג מספרים שליליים בשיטת משלים ל- 2
## (Two's Complement)

בייצוג עשרוני מסמנים מספר שלילי ע"י סימן "-". בייצוג בינרי מסמנים מספר בינרי ע"י הספרה השמאלית ביותר של הנתון: אם היא 0 - המספר חיובי, 1 - שלילי.

<u>דוגמא:</u>

מספר חיובי - 0 1 1 0 1 1 0 **0**

מספר שלילי - 0 1 1 1 0 1 1 **1**

מובן שטווח המספרים הניתנים לייצוג כאשר סיבית אחת מוקדשת לסימן קטן פי 2.

לדוגמא, כפי שראינו, מספר הערכים הניתנים לייצוג ע"י בית (8 סיביות) הוא $2^8=256$ עבור מספרים חיוביים, אולם אם נרצה לשריין מקום לסיבית הסימן, יקטן מספר הערכים פי 2 ל- $2^7=128$.

קיימות מספר שיטות לייצוג מספרים שליליים: השיטה המקובלת נקראת "**משלים ל- 2**" (**Two's complement**). בשיטה זו, נניח שאנו רוצים לכתוב בייצוג בינרי את המספר השלילי 33- (ייצוג עשרוני):

1. ראשית נרשום את המספר החיובי 33: 00100001 = 33

2. נמיר כל סיבית בסיבית ההפוכה לה (0 מומר ל- 1 ו- 1 מומר ל- 0): 11011110

+

3. נוסיף 1 למספר שהתקבל: 1

- 33 = 11011111

התקבל המספר השלילי המבוקש בשיטת "משלים ל- 2".

כלומר, בכדי לקבל ייצוג של מספר שלילי יש לרשום אותו תחילה בבסיס בינרי כחיובי, להפוך כל סיבית, ולבסוף להוסיף 1.

## תכונות של מספרים בינריים

הגדרות:

- **MSB** (Most Significant Bit) - הסיבית המשמעותית ביותר במספר בינרי. זוהי הסיבית בקצה השמאלי של המספר.
- **LSB** (Least Significant Bit) - הסיבית הכי פחות המשמעותית במספר בינרי. זוהי הסיבית בקצה הימני של המספר.

**MSB** **LSB**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

תכונות:

- ה- **MSB** במספר בינרי קובעת אם הוא חיובי (0) או שלילי (1).
- ה - **LSB** במספר בינרי קובעת אם הוא זוגי (0) או אי זוגי (1).

אם נזיז את כל הסיביות של מספר בינרי שמאלה מקום אחד ונוסיף סיבית 0 מימין, יהיה ערך המספר הבינרי המתקבל גדול פי- 2 מערכו המקורי.

לעומת זאת, הזזה ימינה במקום אחד מהווה חילוק ב- 2 (חילוק שלם, ללא שארית). פעולה זו על סיביות נקראת **הזזה** (SHIFT). דוגמאות:

הזזת המספר 0010 = 2 מקום אחד שמאלה תיתן את הערך: 0100 = 4

הזזת המספר 0101 = 5 מקום אחד ימינה תיתן את הערך: 0010 = 2

באופן כללי, הזזה של סיביות מספר בינרי n מקומות שמאלה שקולה לכפלה ב- $2^n$, ואילו הזזה ימינה n סיביות שקולה לחילוק ב- $2^n$. דוגמאות:

הזזת המספר 00000100 = 4 שלושה מקומות שמאלה תיתן את הערך: 00100000 = 32

הזזת המספר 00001101 = D שלושה מקומות ימינה תיתן את הערך: 00000001 = 1

## ייצוג מספרים הקסה-דצימלי (בסיס 16)

השימוש במספרים בייצוג בינרי מסורבל וקשה להבנה. נעזרים בייצוג הקסה-דצימלי (בסיס 16) המתאים לבסיס הבינרי. הספרות בבסיס 16 :

9 - 0 - כמו בייצוג עשרוני.

A - בייצוג עשרוני ערך 10.

B - בייצוג עשרוני ערך 11.

C - בייצוג עשרוני ערך 12.

D - בייצוג עשרוני ערך 13.

E - בייצוג עשרוני ערך 14.

F - בייצוג עשרוני ערך 15.

כל ספרה הקסה-דצימלית מתאימה ל- 4 ספרות בינריות:

| הקסה-דצימלי | | בינרי | הקסה-דצימלי | | בינרי |
|---|---|---|---|---|---|
| 8 | - | 1000 | 0 | - | 0000 |
| 9 | - | 1001 | 1 | - | 0001 |
| A | - | 1010 | 2 | - | 0010 |
| B | - | 1011 | 3 | - | 0011 |
| C | - | 1100 | 4 | - | 0100 |
| D | - | 1101 | 5 | - | 0101 |
| E | - | 1110 | 6 | - | 0110 |
| F | - | 1111 | 7 | - | 0111 |

בהתאם לכך, בית מיוצג ע"י 2 ספרות הקסה-דצימליות. לדוגמא : נתון בית בייצוג הקסה-דצימלי: **B9**. מהו המספר בייצוג בינרי?

תשובה: 9 הוא בעל ערך 1001 , B הוא בעל ערך 1011, ולכן ערך הבית **10111001**. כפי שניתן לראות, ההמרה בין הבסיסים הבינרי וההקסה-דצימלי היא ישירה.

## המרה בין בסיסים דצימלי והקסה-דצימלי

ההמרה בין ייצוג הקסה-דצימלי לייצוג עשרוני זהה להמרה בין הייצוג הבינרי לעשרוני.

- המרה מהקסה-דצימלי לעשרוני: נתון המספר 1F34 בבסיס הקסה-דצימלי. נמיר אותו ע"י רישומו כסכום מכפלות של ספרותיו בחזקות של 16:

$$1F34_{16} = 1*16^3 + 15*16^2 + 3*16^1 + 4*16^0$$

$$= 1*4096 + 15*256 + 3*16 + 4*1$$

$$= 7988_{10}$$

- המרה הפוכה מייצוג עשרוני לייצוג הקסה-דצימלי: נתון המספר העשרוני 7,988. נמיר אותו ע"י חלוקה ורישום השארית:

| | | <u>מנה</u> | <u>שארית</u> |
|---|---|---|---|
| 7988 / 16 | = | 499 | 4 |
| 499 / 16 | = | 31 | 3 |
| 31 / 16 | = | 1 | F |
| 1 / 16 | = | 0 | 1 |


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## חיבור מספרים הקסה דצימליים

חיבור מספרים הקסה דצימליים מבוצע בדומה לבסיסים אחרים (דצימלי ובינרי), כאשר בבסיס 16 יש לקחת בחשבון את הספרות 0-F. דוגמאות לחיבור מספרים חד ספרתיים:

F + 1 = 10

8 + 7 = F

F + F = 1E

דוגמאות לחיבור מספרים רב ספרתיים:

.1

| | | | | | |
|---|---|---|---|---|---|
| | 1 | | 1 | | נשא: |
| | 3 | F | 1 | D | |
| | + | 2 | 0 | 3 | |
| | 4 | 1 | 2 | 0 | |

.2

| | | | | | |
|---|---|---|---|---|---|
| | 1 | 1 | | | נשא: |
| | 2 | E | F | 3 | |
| + | 3 | F | 6 | A | |
| | 6 | E | 5 | D | |

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-4 שבעמ' 23.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## ייצוג מספרים אוקטלי (בסיס 8)

בייצוג אוקטלי המספרים הם בבסיס 8, והם כוללים את הספרות 0..7.

בדומה למספרים בבסיס הקסה-דצימלי, ספרות בסיס 8 מתאימות למספרים בינריים בני 3 סיביות. דוגמאות:

$7_8 = 111_2 = 7_{10}$

$10_8 = 001\ 000_2 = 8_{10}$

$12_8 = 001\ 010_2 = 10_{10}$

ההמרות בין הבסיס האוקטלי והבסיסים הקודמים הן בדומה לבסיסים האחרים.

# יצירת תכנית מחשב

תכניות הרצות על מחשב מסוים כתובות בשפת מכונה, כלומר בשפה המובנת ע"י המעבד המסוים שבלב המחשב.

שפת המכונה מורכבת מאוסף הוראות הכתובות כקודים בינריים - כלומר בבסיס 2 - והיא כוללת אפשרויות לגישה לנתונים בזיכרון, ביצוע פעולות חישוב חשבוניות ולוגיות וכן גישה להתקני הקלט/פלט השונים במחשב:

**תכנית בשפת מכונה**

```
_DATA
#1
#2
#3
#4

_CODE
30        5056, #1
30        5056, #2
30        5056, #3
5         #4, #1
5         #4, #2
5         #4, #3
31        5057,  #4
```

מכיוון ששפת המכונה קשה להבנה ע"י בני אדם, הוגדרו שפות תכנות **עיליות** הדומות לשפת אנוש: אלה כתובות באופן טקסטואלי, וכוללות הוראות נוחות לקריאת נתונים מהקלט, הגדרת משתנים בזיכרון, ביצוע פעולות חשבוניות ולוגיות, הדפסת נתונים לפלט וכו'.

לדוגמא, התכנית לעיל תיכתב כך בשפה עילית:

**תכנית בשפה עילית**

```
integer data1
integer data2
integer data3
integer sum

read data1
read data2
read data3
sum = data1 + data2 + data3
print sum
```

*הערה: שפת אסמבלר אינה נחשבת כשפה עילית - היא ייצוג טקסטואלי של שפת המכונה. שפה עילית לעומתה, מלבד ייצוגה הטקסטואלי, כוללת הוראות מורכבות, משפטי תנאי, לולאות ומבני בקרה נוספים המפשטים את התכנית.*

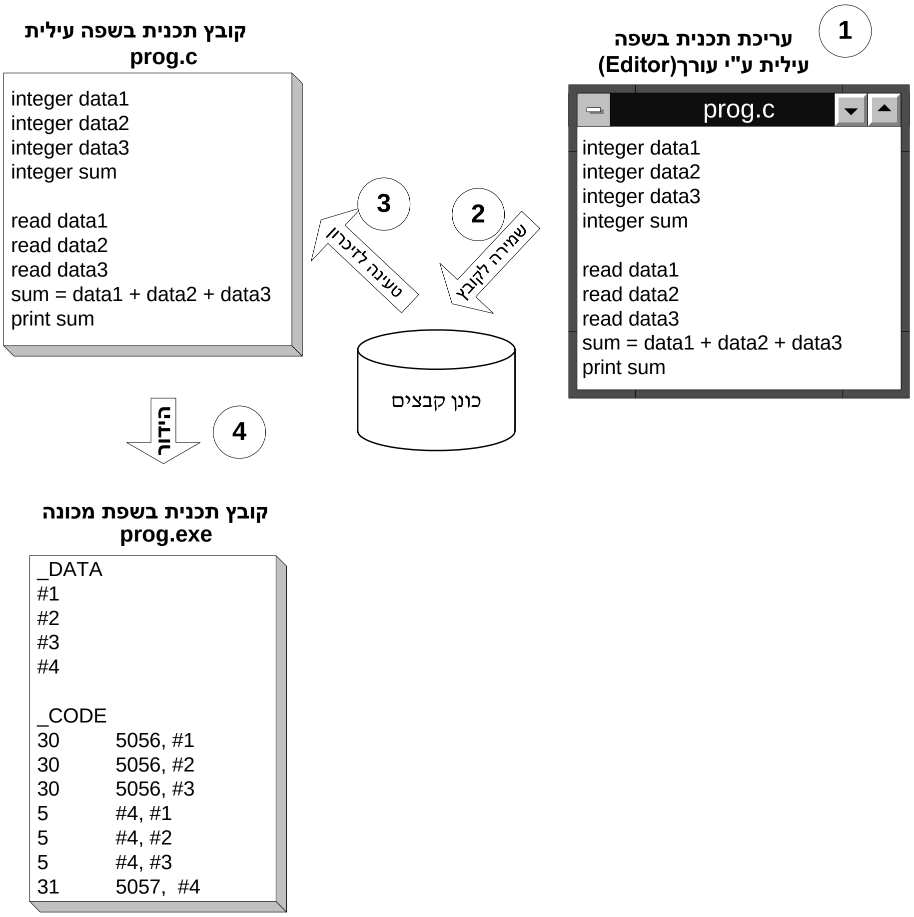

מרכז ההדרכה 2000 © כל הזכויות שמורות

תכנית בשפה עילית נכתבת ע״י **עורך** (Editor) לקובץ. בדרך כלל מותאמת תכנית העורך במיוחד לשפה ומסייעת בכתיבת תכניות ע״י מתן עזרה מקוונת לשפת התכנות, סימון מרכיבים שונים בשפה בצבעים מיוחדים וכו׳.

תכניות הכתובות בשפות התכנות העיליות מתורגמות לשפת המכונה ע״י **מהדר** (Compiler): מהדר הוא בעצמו תכנית מחשב הקוראת תכנית שפה עלית מהקובץ בו היא כתובה - **קובץ המקור** - ומתרגמת אותה לשפת מכונה.

התכנית המתורגמת נשמרת לקובץ הנקרא **קובץ ביצוע**.

התרשים הבא מתאר מהלך יצירה של תכנית - עריכה, שמירה לקובץ, טעינה לזיכרון והידור:



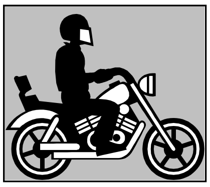

בדרך כלל העורך והמהדר - ביחד עם כלי פיתוח נוספים - כלולים ב**סביבת פיתוח משולבת** (**IDE,** Integrated Development Environment) המאפשרת הפעלת הוראות כמו הידור, הרצה וניפוי ע״י ממשק משתמש גרפי ידידותי.

סביבת הפיתוח מאפשרת גם ארגון קבצי תכניות בפרוייקטים, קביעת מאפייני עריכה והידור, שמירת תכניות לקבצים ועוד.

# אלגוריתמים

אלגוריתם הוא סדרה של הוראות מדויקות לביצוע משימה נתונה. הוראות האלגוריתם חייבות להיות **מובנות** ו**חד-משמעיות** למי שאמור לבצע אותו.

לדוגמא, נניח שבבעלותנו פיצרייה, וקיבלנו הזמנה טלפונית למשלוח פיצה ללקוח בכתובת מסוימת. ההוראות לשליח עם קטנוע יהיו:



***סע לכתובת הלקוח***

***אתר את הדירה עפ"י שם הלקוח או עפ"י מספר הדירה***

***מסור ללקוח את הפיצה וגבה ממנו תשלום***

***חזור לפיצרייה***

מכיוון שהשליח **מבין** כל אחת מההוראות והן **חד-משמעיות** הוא יבצען ללא קושי.

לעומתו, מורה לנהיגה על קטנוע לא ייתן לתלמיד בשיעורו הראשון את ההוראה *"סע לכתובת זו"* מכיוון שהתלמיד לא יודע כיצד לבצע זאת.

הוא ייתן לו סדרת הוראות לביצוע תת-משימה זו:

***התנע את הקטנוע***

***שחרר את הבלם***

***הכנס להילוך***

***החל לנסוע***

***בצומת הקרובה פנה ימינה***

***...***

באופן דומה, בהגדרת אלגוריתם ליצירת תכנית מחשב יש לקחת בחשבון אילו הוראות בסיסיות קיימות בשפת התכנות המיועדת.

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## אלגוריתמים כתכנון תכנית מחשב

אלגוריתמים משמשים להגדרה מקדימה, לפני כתיבת תכנית המחשב, למשימה הנדרשת לביצוע.

את הוראות האלגוריתם ניתן לרשום באופן טקסטואלי, בדומה להכנת מתכון לבישול, ע"י תרשים זרימה, או בכל צורה אחרת.

לדוגמא, נתונה המשימה הבאה:

***כתוב/י תכנית שתקרא מהקלט 3 מספרים, תחשב ותדפיס את הממוצע שלהם.***

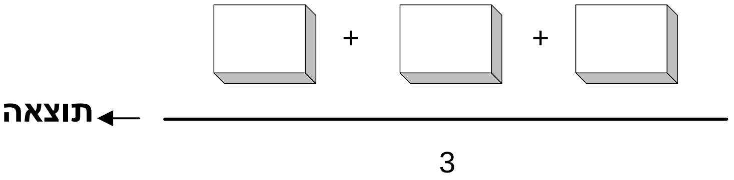
ננסה להגדיר את האלגוריתם לביצוע המשימה:

***קרא 3 מספרים מהקלט***

***חשב את סכום שלושת המספרים***

***חלק את הסכום ב- 3***

***הדפס את התוצאה***




האם האלגוריתם מוביל לביצוע המשימה? ראשית, נבדוק אם הוראותיו **מובנות** בשפת התכנות המיועדת:

- ההוראה *"קרא 3 מספרים מהקלט"* היא אינה הוראה בסיסית ויש להגדירה ביתר פירוט - יש לציין את שמות תאי הזיכרון שלתוכם ייקלטו המספרים.

  תאי זיכרון אלו נקראים **משתנים** מכיוון שערכם יכול להשתנות במהלך התכנית. את המשתנים נגדיר בתחילת האלגוריתם:

  ***משתנים: num1, num2, num3 - משתני הקלט***

  ***קרא מספר ראשון מהקלט לתוך num1***

  ***קרא מספר שני מהקלט לתוך num2***

  ***קרא מספר שלישי מהקלט לתוך num3***

- ההוראה *"חשב את סכום שלושת המספרים"* היא הוראה חשבונית בסיסית מובנת, אולם יש לציין היכן לשמור את הסכום שחושב, כלומר באיזה תא זיכרון או באיזה משתנה.

  לשם כך, נגדיר משתנה נוסף

  ***avg - משתנה התוצאה***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

ונגדיר את ההוראה במפורט:

***חשב את הסכום של num3, num2, num1 והצב אותו ב- avg***

- ההוראה *"חלק את הסכום ב- 3"* גם היא הוראה הדורשת פירוט בהתייחס למשתנים:

***חלק את avg ב- 3 והצב את התוצאה ב- avg***

- גם את ההוראה *"הדפס את התוצאה"* נרשום באופן מפורש:

***הדפס את avg***

נכתוב שוב את האלגוריתם במלואו:

***משתנים: num1, num2, num3 - משתני הקלט***

***avg - משתנה התוצאה***

***קרא מספר ראשון מהקלט לתוך num1***

***קרא מספר שני מהקלט לתוך num2***

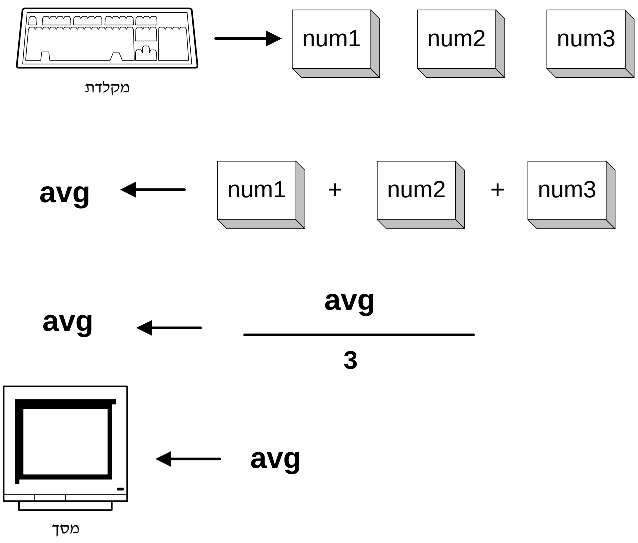
***קרא מספר שלישי מהקלט לתוך num3***

***חשב את הסכום של num1, num2, num3 והצב אותו ב- avg***

***חלק את avg ב- 3 והצב את התוצאה ב- avg***

***הדפס את avg***

תרשים האלגוריתם:




האם הוראות האלגוריתם הן **חד-משמעיות**? התשובה לכך שוב תלויה בשפת התכנות: בהגדרת המשתנים לא צוין סוג המספר - ממשי או שלם - ולכן היא איננה **חד-משמעית** בשפת תכנות המפרידה בין טיפוס מספרים **שלם** לבין **ממשי**.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

נכתוב שוב את האלגוריתם, תוך ציון הטיפוס המספרי של המשתנים: משתני הקלט יהיו ממשפחת השלמים.

משתנה התוצאה, avg, צריך להיות ממשי מכיוון שהוא מכיל ממוצע, וזה בדרך כלל אינו ערך שלם.

***משתנים: num1, num2, num3 - משתני הקלט, שלמים***

***avg - משתנה התוצאה, ממשי***

***קרא מספר ראשון מהקלט לתוך num1***

***קרא מספר שני מהקלט לתוך num2***

***קרא מספר שלישי מהקלט לתוך num3***

***חשב את הסכום של num1, num2, num3 והצב אותו ב- avg***

***חלק את avg ב- 3 והצב את התוצאה ב- avg***

***הדפס את avg***

כעת האלגוריתם מוגדר היטב עבור שפות תכנות עיליות. בפרק הבא נממש את האלגוריתם בשפת C.

# מבוא לשפת C

שפת C היא אחת השפות הנפוצות ביותר בעולם התוכנה מזה כשלושה עשורים. חברות הייטק ואנשי תוכנה רבים בכל העולם עדיין משתמשים בה כשפה עיקרית לפיתוח.

מבין כל שפות התכנות הקיימות בעולם, לשפת C מספר המהדרים הרב ביותר למגוון מערכות, החל ממחשבי Unix, PC, Mainframe וכלה במערכות זמן אמת.

אחד מענפי שוק ההייטק הנשלט עדיין ברובו ע"י שפת C הוא שוק המערכות משובצות המחשב. בין המערכות הרבות השייכות לקטגוריה זו:

- מערכות תקשורת - מודמים, נתבים, מערכות סלולריות.
- מערכות רפואיות - מערכות צילום ואבחון, אמצעים ממוחשבים לניתוחים ולטיפולים.
- מערכות צילום דיגיטליות - מצלמות תמונה, מצלמות וידאו, סורקים.

בעשורים האחרונים פותחו גישות תוכנה שונות מהגישה הפרוצדורלית, כאשר הבולטת מביניהן היא הגישה מונחית העצמים.

חלוצת הגישה היא שפת ++C שהינה הרחבה של שפת C, ובעיקבותיה התפתחה גם Java. מכיוון ש- ++C היא הרחבה של C, שפת C היא קדם ללימוד שפת ++C.

כיום, מרבית המהדרים של שפת C הם גם מהדרים של שפת ++C. משום כך, ניתן ורצוי לנצל שיפורים שהוכנסו לשפה ב- ++C ולכתוב תכניות בשפת C משופרת.

בהמשך נציין בכל מקום את השיפורים המתאימים ב- ++C. בכדי לנצל את השיפורים, יש להגדיר למהדר שקובץ המקור הוא קובץ ++C - זה מבוצע בדרך כלל ע"י מתן סיומת מתאימה לקובץ (**.cpp** ב- Windows ו- **.cc** ב- Unix/Linux), או ע"י קביעת האופציה המתאימה בסביבת הפיתוח.

במידת הצורך, ניתן לעיין להרחבה בקורס המקוון "**++C - מדריך מקצועי**" שבאתר האינטרנט, בכתובת **http://www.mh2000.co.il/cpp**.

## רקע היסטורי

שפת C פותחה בראשית שנות ה- 70 ע״י דניס ריצ׳י. היא מבוססת על השפות הקודמות B (קן תומפסון) ו- BCPL (מרטין ריצ׳רדס) ומכאן קיבלה את שמה.

שפת C נוצרה כאמצעי לפיתוח מערכת ההפעלה Unix.

לשפת C הוגדר תקן ע״י מכון התקנים האמריקאי ANSI (American National Standard Institute) בסוף שנות ה- 80.

מספר כללים בתקן שונים מההגדרה הראשונית של השפה. רוב המהדרים כיום מותאמים לתקן זה (ANSI-C) מה שמאפשר העברת תכניות הכתובות בשפת C ממערכת אחת לאחרת במינימום שינויים.

# 2. הכרת שפת C

- תכנית ראשונה ב– C
- הגדרת משתנים, קלט ופלט
- משפטי תנאי ולולאות
- הגדרת קבועים
- קלט / פלט של תווים
- מחרוזות
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

# תכנית ראשונה ב- C

נכתוב תכנית בסיסית בשפת C המדפיסה למסך שורה בודדת. קוד התכנית:

*file: hello.c*

```
#include <stdio.h>

void main ( )
{
	printf("hello, Israel!\n");
}
```

פלט התכנית:

```
hello, Israel!
```

### הסבר התכנית

התכנית כתובה בקובץ בשם **hello.c**. בהמשך נראה כיצד ניתן להדר את התכנית ולהריץ אותה.
החלק הראשון:

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

מודיע למהדר שאנו עומדים להשתמש בספריית הקלט/פלט התקנית stdio.h (Standard I/O).
החלק השני:

```
void main ()
{

}
```

הוא הגדרת הפונקציה הראשית של התכנית:

- הפונקציה **main** חייבת להופיע בכל תכנית C.
- הסוגריים המסולסלות מציינות את התחלת וסיום הפונקציה.
- כפי שנראה בהמשך, התכנית יכולה להכיל פונקציות נוספות.

החלק הנמצא בין הסוגריים המסולסלות

```
{
	printf("hello, Israel!\n");
}
```

הוא גוף הפונקציה הראשית, והוא כולל הוראת הדפסה ע"י קריאה לפונקצית הספרייה **printf**:
הפונקציה מדפיסה את המחרוזת הנתונה לה בין הסוגריים, "hello, Israel!\n", כאשר סיומת המחרוזת "\n" היא צירוף תווים מיוחד המורה על הדפסת תו שורה חדשה (new line).

הפונקציה **printf** לא מדפיסה תו שורה חדשה אלא אם כן הוכנס צירוף התווים המיוחד במחרוזת להדפסה. לכן ניתן לכתוב את התוכנית גם כך:

```
#include <stdio.h>
void main( )
{
        printf("hello,");
        printf("Israel!");
        printf("\n");
}
```

***שיפור ב C++ :*** בשפת C++ קיים אופרטור פשוט, "<<", לביצוע פלט. התכנית הנ"ל תיכתב כך ב- C++:

```
#include <iostream.h>
void main()
{
   cout << "hello C++!" << endl;
}
```

העצם **cout** מוגדר בספרייה **iostream.h** החליפית ל- stdio.h, והפלט נשלח אליו ע"י האופרטור "<<" ולאחריו הנתונים להדפסה. endl מציין סוף שורה בפלט.

## כתיבה, הידור והרצת התכנית

לצורך יצירת והרצת התכנית יש צורך בסביבת פיתוח. ניתן להוריד סביבת פיתוח בסיסית מאתר האינטרנט של "**מרכז ההדרכה 2000**", **www.mh2000.co.il**, ולהתקין אותה במחשב.

ליצירת התכנית ולהרצתה דרושים מספר שלבים:

- **פתיחת קובץ חדש**: פותחים קובץ חדש ע"י בחירה בתפריט File / New:
- **כתיבת התכנית**: כותבים את התכנית הנ"ל בקובץ החדש שיצרנו.
- **שמירת הקובץ**: לצורך שמירת התכנית, יש לבחור ב- File / save ולבחור שם קובץ לשמירה hello.c.
- **הידור והרצת התכנית**: להידור התכנית בוחרים בתפריט Compile. אם ההידור הצליח, ניתן להריץ את התכנית ע"י בחירה בתפריט Run.

  אם יש שגיאות, הן מופיעות בחלון התחתי, חלון השגיאות. סימון הודעת השגיאה יציג בחלון העריכה את שורת השגיאה.

קיימות סביבות פיתוח נוספות ליצירת תכניות C/C++, לדוגמא סביבת הפיתוח Visual C++ של חברת Microsoft הפועלת מעל מערכת ההפעלה Windows.

בסביבה זו יש ליצור פרוייקט מסוג "**Win32 Console Application**" לצורך יצירת והרצת תכניות.

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 33.***

# הגדרת משתנים, קלט ופלט

בתכנית הדוגמא הבאה נכיר מרכיבים בסיסיים בשפת C:

- הגדרת משתנים והתייחסות אליהם בתכנית
- קליטת ערכים מהקלט לתוך משתנים
- הדפסת תוכן המשתנים לפלט

## חישוב והדפסת הממוצע של 3 מספרים

התכנית הבאה מממשת את האלגוריתם שראינו בפרק 1, "**מבוא לתכנות**", לחישוב והדפסת הממוצע של 3 מספרים שלמים:

***משתנים: num1, num2, num3 - משתני הקלט, שלמים***

***avg - משתנה התוצאה, ממשי***

***קרא מספר ראשון מהקלט לתוך num1***

***קרא מספר שני מהקלט לתוך num2***

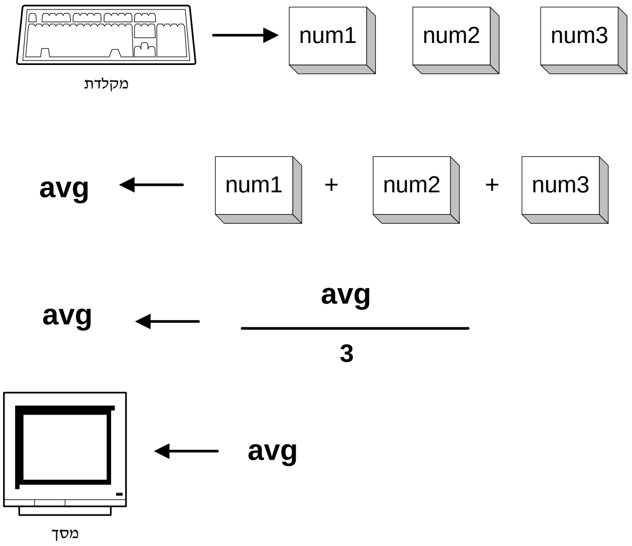
***קרא מספר שלישי מהקלט לתוך num3***

***חשב את הסכום של num1, num2, num3 והצב אותו ב- avg***

***חלק את avg ב- 3 והצב את התוצאה ב- avg***

***הדפס את avg***

תרשים האלגוריתם :




<u>קוד התכנית</u>

קוד התכנית בשפת C נכתב בקובץ **average.c** ונראה כך :

```
/* file: average.c */
#include <stdio.h>

/* calculate the average of 3 numbers */
void main()
{
	int num1, num2, num3;
	float avg;

	printf("Enter 3 integer numbers: ");
	scanf("%d %d %d", &num1, &num2, &num3);

	avg = num1 + num2 + num3;

	avg = avg / 3;

	printf("The average is: %f", avg);
}
```

<u>הרצת התכנית</u>

בהרצת התכנית הוכנסו 3 מספרים

```
Enter 3 numbers: 23 45 11
```

והתקבל הפלט :

```
The average is: 26.333334
```

## הסבר התכנית

<u>הקטע ההצהרתי</u>

- כמו בתכנית הקודמת, גם כאן נצטרך להשתמש בספריית הקלט/פלט התקנית ולכן נבצע את ההכללה המתאימה:

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

- **סימן הערה**: המהדר מתעלם מהתווים הנמצאים בין /*...*/ ולכן הם משמשים לרישום הערות הסבר לתכנית:

```
/* calculate the average of 3 numbers */
void main ( )
{
...
}
```

כמו בתכנית הקודמת, קוד התכנית נכתב בתוך פונקציה בשם **main**. על משמעות המילה **void** נלמד בהמשך. כמו כן נראה שניתן להגדיר את הפונקציה בדרכים שונות.

- **הצהרה על משתנים וטיפוסיהם** - בהצהרה על משתנה , ראשית מופיע הטיפוס (type) ואח"כ שם המשתנה.

  – הטיפוס **int** (integer) משמש להצהרה על משתנה מטיפוס <u>שלם</u>:

```
int num1, num2, num3;
```

num1, num2, num3 הם משתנים שלמים שלתוכם ייקלטו הערכים מהקלט. כפי שניתן לראות, ניתן להכריז על מספר משתנים באותה הוראה.

  – הטיפוס **float** (floating point) מציין טיפוס ממשי בהצהרת המשתנה avg:

```
float avg;
```

avg הוא משתנה ממשי שבו תחושב תוצאת הסכום ולאחר מכן הממוצע.

טיפוסים נוספים הקיימים בשפה:

char - תו, בית בודד

short - שלם קצר

long - שלם ארוך

double - ממשי בעל דיוק כפול

long double- ממשי בעל טווח ערכים גדול

נרחיב על טיפוסי משתנים בפרק הבא.

<u>הקטע הביצועי</u>

הקטע הביצועי של התכנית מתחיל עם ההוראה הראשונה שאינה הגדרת משתנה. לאחר תחילת הקטע הביצועי לא ניתן להגדיר משתנים נוספים. ההוראה הביצועית הראשונה היא הוראת הפלט

*printf("Enter 3 integer numbers: ");*

בהוראה זו מתבקש המשתמש להקליד 3 מספרים שלמים. בשלב הבא נקראים המספרים ע"י ההוראה **scanf** :

*scanf("%d %d %d", &num1, &num2, &num3);*

**scanf** היא פונקצית קלט הקוראת מהקלט לתוך רשימת פרמטרים עפ"י פורמט הנתון לה במחרוזת בקרה. לצורך ביצוע הקלט אנו מספקים ל- **scanf** את <u>מחרוזת הבקרה</u>

*"%d %d %d"*

ולאחריה את רשימת **כתובות** המשתנים שמבקשים לקלוט, מופרדים ע"י פסיק:

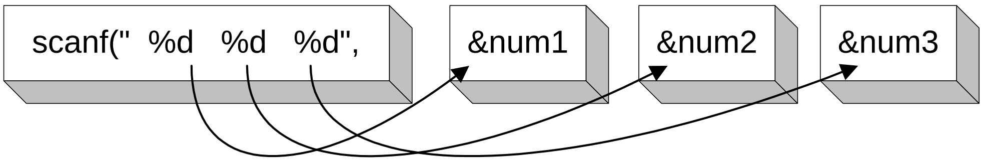
*&num1, &num2, &num3*

הפונקציה scanf קוראת נתונים מהקלט לתוך <u>כתובות</u> שניתנות לה כפרמטרים.

האופרטור "&" מציין "הכתובת של": הצבתו לפני שם המשתנה מציינת התייחסות לכתובתו - נעסוק במצביעים בהרחבה בפרק 8, "**מצביעים**".

במקרה זה ציינו שאנו מעוניינים לקרוא 3 מספרים שלמים: %d במחרוזת הבקרה הוא **מציין טיפוס** המציין עבור המהדר טיפוס שלם של פרמטר מתאים ברשימת הפרמטרים.

לכל פרמטר מותאם מציין טיפוס, עפ"י הסדר:




קיימים מצייני טיפוס גם לטיפוסים האחרים, לדוגמא:

%f - מציין טיפוס ממשי

%c - מציין טיפוס תו

%s - מציין טיפוס מחרוזת

scanf מתעלמת מהתווים ה"לבנים" - רווח, טאב ותו שורה חדשה - אלא אם כן סוג הנתון הנקרא הוא תווי.

לכן ניתן להקליד את המספרים עם רווחים ושורות ריקות ביניהם, והם עדיין ייקראו נכונה.

לאחר קליטת המספרים, מחושב סכומם ומוצב ל- avg :

*avg = num1 + num2 + num3;*

הסימן "=" ב- C מציין **פעולת הצבה** (Assignment). תוצאת הביטוי שמימין לסימן מוצבת לתא בזיכרון שמשמאל לו. פעולה זו נקראת גם "**השמה**".

<u>שאלה</u>: איך מסמנים שוויון בביטוי לוגי?

<u>תשובה</u>: ע"י שני תווים רצופים כנ"ל: "==".

חישוב הממוצע מתבצע ע"י חילוק avg ב- 3:

*avg = avg / 3;*

avg / 3 מבצע פעולת חילוק וזו מוצבת ל- avg. אין כל בעיה בהימצאותו של avg משני צידי הוראת ההצבה - הצד הימני מחושב קודם, ולאחר מכן מוצבת התוצאה למשתנה שבצד השמאלי.

ניתן לקצר את התכנית ע"י ביצוע פעולות החיבור והחלוקה בהוראה יחידה:

*avg = (num1 + num2 + num3) / 3;*

כאן ביצענו חיבור בתוך סוגריים מכיוון שלאופרטור החילוק עדיפות על אופרטור החיבור. הסוגריים מציינים את סדר הפעולות הנדרש.

בסיום מודפס ערכו של avg ע"י הפונקציה printf:

*printf("The average is: %f", avg);*

בדומה לפונקציה scanf, גם printf כוללת מחרוזת בקרה ולאחריה רשימת פרמטרים.

בשונה ממנה, הפרמטרים ל- printf אינם כתובות המשתנים אלא המשתנים עצמם. במקרה זה ציינו שאנו מעוניינים להדפיס את המחרוזת

```
The average is:
```

ולאחריה את ערכו של avg. כפי שראינו, פלט התכנית שהורצה לעיל הוא

```
The average is: 26.333334
```

***שיפור ב C++ :*** בשפת C++ קיים אופרטור פשוט, "<<", לביצוע קלט.
התכנית הנ"ל תיכתב כך ב- C++:

```
/* file: average.cpp */
#include <iostream.h>

/* calculate the average of 3 numbers */
void main()
{
    int num1, num2, num3;
    float avg;

    cout << "Enter 3 integer numbers: ";
```

```
    cin >> num1 >> num2 >> num3;

    avg = num1 + num2 + num3;

    avg = avg / 3;

    cout << "The average is: " << avg;
}
```

העצם cin מוגדר בספרייה iostream.h החליפית ל- stdio.h, והקלט מובא ממנו ע"י האופרטור "<<" למשתנים שלאחריו.

יש לשים לב שבהוראות הקלט/פלט ב- C++ אין צורך לציין את טיפוס האיברים במחרוזת בקרה. שיפור נוסף הוא שבהוראת הקלט אין צורך להעביר את כתובות המשתנים.

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

# משפטי תנאי ולולאות

לעיתים תכופות מהלך התכנית נקבע באופן דינמי עפ"י הקלטים או חישובים מספריים. בנקודות מסוימות בתכנית נרצה לבצע קטע קוד כתלות בתנאי מסוים. בנקודות אחרות נרצה לבצע קטע קוד מספר פעמים.

שפת C כוללת הוראות בקרה לקביעת מהלך התכנית:

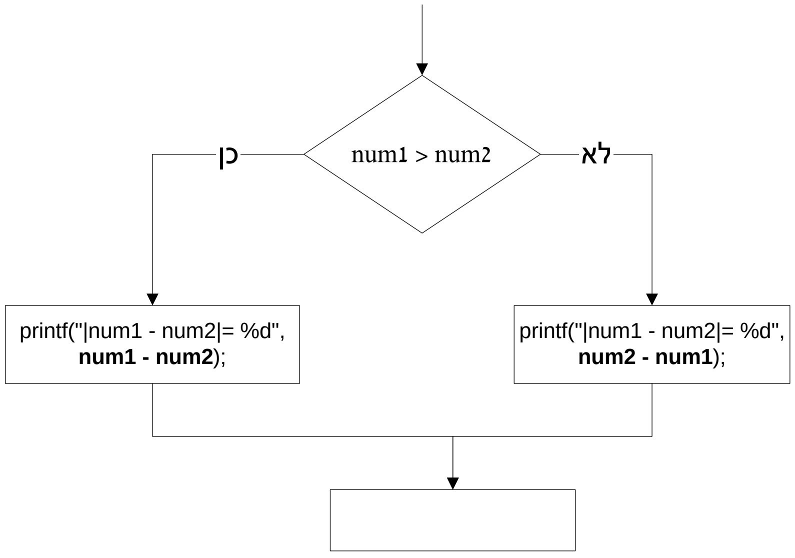
- משפט **if-else** לביצוע מותנה של הוראות
- לולאות מסוגים שונים משמשות לביצוע חוזר של קטע קוד

## משפט if-else

משפט if-else מבצע הוראה או מספר הוראות כתלות בתנאי מסוים. לדוגמא, אם נרצה להדפיס את ההפרש שבין 2 מספרים, num1 ו- num2, בערכו המוחלט נבצע:

```
if( num1 > num2 )
        printf("|num1 - num2|= %d", num1 - num2);
else
        printf("|num1 - num2|= %d", num2 - num1);
```

ניתן לתאר את הוראת if-else ע"י תרשים זרימה:




ההוראה if כוללת ביטוי לוגי בסוגריים, ובמידה וערכו **אמת** - כלומר num1 גדול מ- num2 - ההוראה העוקבת מבוצעת:

```
if( num1 > num2 )
        printf("|num1 - num2|= %d", num1 - num2);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

ההוראה else לאחר ההוראה if מציינת פעולה אלטרנטיבית לביצוע אם תוצאת הביטוי הלוגי הייתה **שקר**:

```
else
        printf("|num1 - num2|= %d", num2 - num1);
```

ההוראה else היא אופציונלית - ייתכן משפט if ללא הוראת else.

***עיין/י בתכנית הדוגמא המובאת בעמ' 40-41.***

<u>ביצוע מספר הוראות - בלוק</u>

אם רוצים לבצע מספר הוראות בהוראת if או else, יש להקיפם בסוגריים מסולסלות, לדוגמא:

```
if( num1 > num2 )
{
        max = num1;
        printf("The maximum is: %d", max);
}
```

סדרת הוראות מוקפת בסוגריים { } נקראת **בלוק**.

***שיפור ב C++ :*** גירסת C++ של התכנית תראה כך:

```
/* file: if-else.cpp */
#include <iostream.h>

void main()
{
   int num1, num2;

   cout << "Enter 2 integers: ";
   cin >> num1 >> num2;
   if( num1 > num2 )
           cout << "|num1 - num2|= " << num1 - num2;
   else
           cout << "|num1 - num2|= " << num2 - num1;

}
```

<u>אופרטורים לוגיים</u>

האופרטור '>' מציין את היחס הלוגי "גדול מ-". הטבלה הבאה כוללת את האופרטורים הלוגיים ב- C ואת משמעותם:

| | |
|---|---|
| x == y | x שווה ל- y |
| x != y | x שונה מ- y |
| x > y | x גדול מ- y |
| x >= y | x גדול מ- y או שווה לו |
| x < y | x קטן מ- y |
| x <= y | x קטן מ- y או שווה לו |


מרכז ההדרכה 2000 © כל הזכויות שמורות

## לולאת while

לולאת while משמשת לביצוע חוזר של הוראות כתלות בתנאי מסוים: תחביר הלולאה הוא

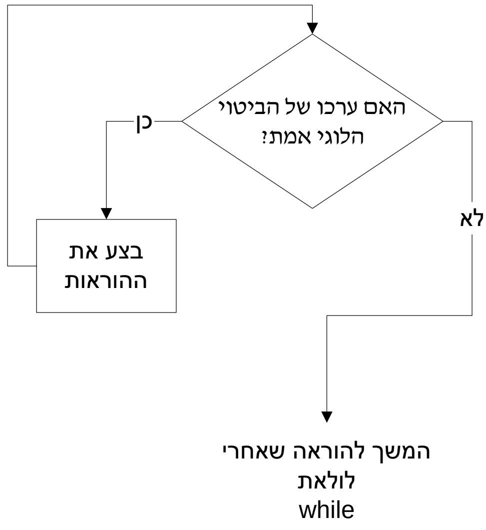
**while(*ביטוי לוגי*)**

{

*הוראות*

}

כל עוד ערכו של הביטוי הלוגי אמת ההוראות שבגוף הלולאה מתבצעות. תרשים זרימה עבור ביצוע לולאת while:




נכתוב תוכנית להדפסת טבלת המרה מאינצ׳ים לסנטימטרים. נוסחת ההמרה:

1 inch = 2.54 cm

קוד התכנית:

```
/* file: convert.c */
#include <stdio.h>

/* print an Inch - Centimeter conversion table */
void main ( )
{
        int     Xinch;
        float  Xcm;
        int     lower, upper, step;

        lower=0;   /* lower limit of table */
        upper=10; /* upper limit */
        step=1;      /* step size */
        Xinch=lower;
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

```
    while(Xinch<=upper)
    {
            Xcm = Xinch * 2.54;
            printf("%d\t%f\n",Xinch,Xcm);
            Xinch=Xinch+step;
    }
}
```

פלט התכנית:

```
0       0.000000
1       2.540000
2       5.080000
3       7.620000
4       10.160000
5       12.700000
6       15.240000
7       17.780000
8       20.320000
9       22.860000
10      25.400000
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## הסבר התכנית

בחלקה הראשון של התכנית מצהירים על משתנים:

```
int     Xinch;
float   Xcm;
int     lower, upper, step;
```

כפי שניתן לראות, ניתן להכריז על משתנים מסוג שלם, אחר כך על משתנים מסוג ממשי ושוב על משתנים מסוג שלם. לאחר מכן מאתחלים אותם:

```
lower=0;   /* lower limit of table */
upper=10;  /* upper limit */
step=1;    /* step size */
Xinch=lower;
```

**לולאת while** מתבצעת כל עוד התנאי הנמצא בסוגריים מתקיים:

```
while(Xinch <= upper)
```

הסימן "=>" מציין "קטן או שווה". משמעות התנאי היא "Xinch קטן או שווה ל- upper". הלולאה בכללותה:

```
while(Xinch <= upper)
{
        Xcm = Xinch * 2.54;
        printf("%d\t%f\n",Xinch,Xcm);
        Xinch=Xinch+step;
}
```

גוף הלולאה כולל 3 הוראות:

1) תרגום הערך מאינצ'ים לסנטימטרים:

```
Xcm = Xinch * 2.54;
```

2) הדפסת התוצאות: בהדפסה אנו מספקים לפונקציה printf את <u>פורמט</u> המשתנים להדפסה:

```
printf("%d\t%f\n",Xinch,Xcm);
```

%d הוא מציין טיפוס שלם המתייחס לשלם Xinch , ו- %f מתייחס לממשי Xcm.

התווים '\t' ו- '\n' הם תווים מיוחדים להדפסה:

\t - מציין הדפסת טאב

\n - מציין הדפסת תו שורה חדשה

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 45.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## סידור פלט התכנית

בעיה: פלט התכנית לא ערוך מספיק יפה:

- המספרים השלמים אינם מוצמדים ימינה
- מיקום הנקודה העשרונית בממשיים אינו אחיד
- די בדיוק של שני מקומות אחרי הנקודה העשרונית בהדפסת הממשיים

פתרון: נשתמש ב**שדות רווח** ו**שדות הצמדה** בפלט:

*printf("%**2**d\t%**5.2**f\n", Xinch, Xcm);*

עבור הדפסת השלם:

- רוחב השדה (2) מצוין בין הסימן % לבין סימן הטיפוס.

עבור הדפסת הממשי:

- רוחב השדה (הכולל) מצוין בין הסימן % לבין סימן הטיפוס.
- רוחב שדה השבר מצוין לאחר הנקודה העשרונית.

כמו כן, נוסיף לטבלה הדפסת כותרות ע"י:

```
printf("Inch\tCm\n");
printf("----\t-----\n");
```

***התכנית המתוקנת מובאת בעמ' 46.***

והפלט:

```
Inch   Cm
----   -----
 0      0.00
 1      2.54
 2      5.08
 3      7.62
 4     10.16
 5     12.70
 6     15.24
 7     17.78
 8     20.32
 9     22.86
10     25.40
```

סיכום שדות ההדפסה ב- printf שהכרנו עד כה:

| | |
|---|---|
| הדפס כשלם עשרוני | %d |
| הדפס כשלם עשרוני ברוחב 6 לפחות | %6d |
| הדפס כממשי (floating point) | %f |
| הדפס כממשי ברוחב 2 אחרי הנקודה העשרונית | %.2f |
| הדפס כממשי ברוחב 6 לפחות, 2 מקומות אחרי הנקודה העשרונית | %6.2f |

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 2-1 שבעמ' 47.***

# הגדרת קבועים

שימוש במספרים בגוף התכנית הוא בדרך כלל רעיון לא "בריא" ממספר בחינות:

- **עריכה** - עלולים לשגות בהקלדת מספרים בגוף התכנית (בעיקר כאשר הם מופיעים מספר פעמים).
- **תיעוד** - קשה להבין מקריאת התכנית את משמעות המספרים.
- **תחזוקה** - אם מספר מופיע פעמים רבות בתכנית, שינויו צריך להתבצע בכל המקומות.

**קבועים** הם שמות המייצגים ערכים קבועים שאינם משתנים לאורך כל התכנית. הם מוגדרים בצירוף המילה השמורה **const**.

שימוש בהגדרת **קבועים** בתכנית מקל על פעולות העריכה, התיעוד והתחזוקה של ערכים קבועים בתכנית.

נבצע מספר שינויים בתכנית:

– נגדיר את המשתנים lower ו- upper כקבועים (מכיוון שאינם אמורים להשתנות במהלך התכנית)

– נגדיר את הקבוע FACTOR שיציין את היחס שבין אינץ' לסנטימטר (2.54)

– נשמיט את המשתנה step המציין את גודל הצעד, ובמקומו נפעיל אופרטור לקידום עצמי.

התכנית החדשה:

```
/* file: convert3.c */
#include <stdio.h>

/* print a Inch - Centimeter conversion table, define constants */
void main ( )
{
	const int   LOWER=0, UPPER=10;
	const float FACTOR=2.54f;
	int     Xinch;
	float  Xcm;

	Xinch=LOWER;
	printf("Inch\tCm\n");  /* print table header */
	printf("----\t--\n");
	while(Xinch<=UPPER)
	{
		Xcm = Xinch * FACTOR;
		printf("%2d\t%5.2f\n",Xinch,Xcm);
		Xinch++;
	}
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

הגדרת הקבועים מתבצעת תוך שימוש במילה השמורה const :

```
const int    LOWER=0, UPPER=10;
const float  FACTOR=2.54f;
```

> *הערה: לקבוע הממשי מצורפת האות 'f' המציינת שהקבוע הוא מטיפוס float. טיפוס ברירת המחדל עבור ערכים ממשיים בתכנית הוא double, כפי שנראה בפרק הבא. הסיומת f מונעת את הודעת האזהרה של המהדר שהייתה בגרסה הקודמת של התכנית.*

במקומות המתאימים בתכנית שמות הקבועים מוחלפים בשמות הערכים:

```
Xinch=LOWER;
printf("Inch\tCm\n");  /* print table header */
printf("----\t--\n");
while(Xinch<=UPPER)
{
        Xcm = Xinch * FACTOR;
        printf("%2d\t%5.2f\n",Xinch,Xcm);
        Xinch++;
}
```

כעת לא ניתן לשנות את הקבועים בתכנית - ניסיון לשנותם ייתן הודעת שגיאה בהידור.

האופרטור "++" מבצע קידום ב- 1 של המשתנה. לכן, הביטוי

**Xinch++**

שקול לביטוי

**Xinch = Xinch +1;**

באופן דומה משמש האופרטור "--" להחסרה של 1.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הגדרת קבועים ע"י הקדם-מעבד (Pre-processor)

קיימת דרך נוספת להגדרת קבועים - ע"י שימוש ב**קדם-מעבד**:

```
/* file: convert4.c */
#include <stdio.h>

#define LOWER       0      /* lower limit of table*/
#define UPPER       10     /* upper limit*/
#define FACTOR      2.54f  /* conversion factor */

/* print a Inch - Centimeter conversion table, define constants */
void main ( )
{
        int    Xinch;
        float  Xcm;

        Xinch=LOWER;
        printf("Inch\tCm\n");  /* print table header */
        printf("----\t--\n");
        while(Xinch<=UPPER)
        {
                Xcm = Xinch * FACTOR;
                printf("%2d\t%5.2f\n",Xinch,Xcm);
                Xinch++;
        }
}
```

<u>הסבר</u>

הגדרות הקבועים

```
#define LOWER       0      /* lower limit of table*/
#define UPPER       10     /* upper limit*/
#define FACTOR      2.54f  /* conversion factor */
```

הן הוראות ל**קדם-מעבד** (pre-processor): זוהי תת-תכנית במהדר העוברת על הקוד שבקובץ ומבצעת את כל ההוראות המתחילות בסימן # (נרחיב בנושא בפרק 11, בסעיף "**הקדם-מעבד**").

שמות הקבועים מוחלפים על ידו למספרים המצויינים, לפני מעבר המהדר על התכנית.

מה ההבדל בין שתי הצורות? בהגדרת קבוע מספרי ע"י const מוגדר תא בזיכרון עבור הקבוע, ואילו בהגדרה ע"י הקדם מעבד אין הגדרה של תא בזיכרון.

הבדל זה יכול להיות משמעותי במערכות משובצות מחשב, כאשר משאבי הזיכרון מוגבלים, שאז כדאי לבחור בהגדרת קבועים מספריים ע"י הקדם-מעבד.

הבדל נוסף הקיים בין 2 הצורות הוא במרחב השם (namespace) של הקבוע: קבוע המוגדר ע"י הקדם מעבד מוכר בכל קובץ התכנית, החל מהמקום בו הוגדר. קבוע המוגדר ע"י const לעומת זאת הוא בעל מרחב שם מוגבל יותר, עפ"י מקום הגדרתו, כפי שנראה בפרק 6, "**פונקציות**".


מרכז ההדרכה 2000 © כל הזכויות שמורות

# קלט / פלט של תווים

לצורך ביצוע קלט/פלט של תווים נשתמש בפונקציות הספרייה התקנית stdio.h:

c=getchar( ) - קוראת את התו הבא מהקלט לתוך המשתנה c.

putchar(c) - מדפיסה את המשתנה c למקום הבא בפלט.

נכתוב תכנית להעתקת תווי הקלט ישירות לפלט. התכנית תקרא תווים בזה אחר זה ותדפיס אותם. אלגוריתם התכנית:

***קרא תו***

***כל עוד לא הגענו לסוף קובץ הקלט:***

***- הדפס את התו שנקרא לקובץ הפלט***

***- קרא תו***

התכנית:

```
/* file: copy.c */
#include <stdio.h>

/* copy input to output: 1st version */
void main()
{
	int c;
	c=getchar();
	while (c!=EOF)
	{
		putchar(c);
		c=getchar();
	}
}
```

<u>הסבר</u>: התכנית מגדירה משתנה מסוג int וקוראת תו ראשון לתוכו. הסימן "=!" מציין "שונה" - כלומר משמעות הביטוי (c!=EOF) היא "התו c שונה מתו סוף קובץ".

מדוע הגדרנו את c כ "int" ולא כ"char"? מכיוון ש- c אמור גם לקבל את ערך הקבוע EOF (שהוא 1-) בהגעה לסוף הקובץ.

יש לשים לב לכך שבהגעה לסוף שורה בקלט נקרא תו סוף השורה '\n' (תו יחיד) ע"י הפונקציה getchar() ומודפס ע"י הפונקציה putchar(). כמו כן ניתן להדפיסו ישירות ע"י

*putchar('\n');*

בפרק הבא נעסוק בהרחבה בטיפוסי התווים ובסוגי תווים מיוחדים.

## שימוש בביטוי הצבה כמחזיר ערך

בשפת C ביטוי הצבה מחזיר ערך. לדוגמא:

*c = getchar();*

הוא ביטוי שערכו כערך שהושם ל- c. כלומר, ערך ביטוי ההצבה הוא הערך שהושם. למשל, ניתן לכתוב

*x = ( c = getchar() );*

ואז x יקבל את הערך שהושם ב- c.

נשנה את התכנית כך שההצבה ובדיקת ההגעה לסוף הקלט יבוצעו באותה הוראה. אלגוריתם התכנית:

***כל עוד התו הנקרא אינו תו סוף הקובץ***

***- הדפס את התו***

קוד התכנית:

```
/* file: copy2.c */
#include <stdio.h>

/* copy input to output: 2nd version */
void main()
{
        int c;
        while ((c=getchar())!=EOF)
                putchar(c);
}
```

קיבלנו תכנית יותר קטנה מצד אחד אך פחות קריאה מצד שני. בדרך כלל לא מומלץ להכניס ביטויים מורכבים מדי בהוראה יחידה, אך לעיתים זה מפשט את התכנית.

<u>שאלה</u>: מדוע נחוצים סוגריים סביב ההצבה c=getchar()?

<u>תשובה</u>: מכיוון שלאופרטור != יש עדיפות גבוהה מלאופרטור =. לו כתבנו

*c = getchar() != EOF*

הביטוי היה מחושב כ-

*c = ( getchar() != EOF )*

כלומר c היה מקבל את הערך הבוליאני של אי-השוויון! בפרק הבא נעסוק בטיפוס בוליאני ובייצוגו בשפת C.

***שיפור ב C++*** : משתנים תוויים, כמו משתנים אחרים, ניתנים לקריאה ולכתיבה ע"י אופרטורי הקלט/פלט:

char c;


מרכז ההדרכה 2000 © כל הזכויות שמורות

```
cin >> c;
cout << c;
```

במקרה זה, התו הנקרא מהקלט ע"י cin הוא התו הראשון שאינו תו לבן (רווח, טאב, תו שורה חדשה) שיופיע. לכן לא ניתן להשתמש באופן זה של קלט תווים בתכנית להעתקת קלט לפלט.

בכדי לקרוא תו כלשהו מהקלט -גם אם הוא לבן - משתמשים בפונקציה get שבעצם cin. באופן דומה, לכתיבת תו לקובץ ניתן להשתמש בפונקציה put של העצם cout. לדוגמא:

```
char c;
cin.get(c);
cout.put(c);
```

התכנית להעתקת קלט לפלט תראה כך:

```
/* file: copy.cpp */
#include <iostream.h>

/* copy input to output*/
void main()
{
    char c;
    while (cin.get(c))
            cout.put(c);
}
```

יש לשים לב שטיפוס התו בביצוע קלט/פלט ב- C++ הוא char ולא int. זאת מכיוון שבסוף קובץ לא מוחזר EOF - הלולאה while "יודעת" עפ"י מצב העצם cin על הגעה לסוף הקובץ.

<u>הצבה כפולה</u>

הואיל וביטוי הצבה מחזיר ערך, ניתן לבצע הצבה כפולה, לדוגמא:

*x = y = 5;*

ביטוי זה שקול לביטוי

*x = (y = 5);*


מרכז ההדרכה 2000 © כל הזכויות שמורות

# מחרוזות

בשפת C לא קיים טיפוס מחרוזת בסיסי. מחרוזת מיוצגת כמערך תווים המסתיים בתו מסיים מחרוזת.

על מערכים נלמד בהרחבה בפרק 7, "**מערכים**", לעת עתה נראה רק כיצד להגדיר מחרוזות באמצעותם.

מציינים ערך מחרוזת ע"י גרשיים משני צידיה, לדוגמא:

```
char str[17] = "www.mh2000.co.il";          /* home page of this book */
```

ההגדרה char str[17] מציינת ש- str הוא מערך תווים בעל 17 מקומות. לאחר ביצוע ההשמה של המחרוזת "www.mh2000.co.il" ל- str היא מחרוזת בת 16 תווים + תו מסיים מחרוזת (בלתי נראה):

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| w | w | w | . | m | h | 2 | 0 | 0 | 0 | . | c | o | . | i | l | '\0' |

התו '\0' (קו נטוי הפוך ואפס) הוא תו סוף מחרוזת. באופן דומה, ניתן להגדיר את str גם ללא ציון גודל המערך:

```
char str[] = "www.mh2000.co.il";
```

המהדר יקצה אוטומטית ל- str את גדלו (17 תווים), עפ"י המחרוזת המוצבת לו. הדפסת מחרוזת מבוצעת ע"י הוראת **printf** באמצעות מזהה הטיפוס %s , לדוגמא:

```
char str[] = "www.mh2000.co.il";
printf("Welcome to %s !",str);
```

יודפס:

```
Welcome to www.mh2000.co.il !
```

## הגדרת טיפוס מחרוזת פשוט

לשם טיפול נוח במחרוזות נגדיר טיפוס מחרוזת, String, באופן הבא:

```
typedef char String[256];
```

**typedef** משמש להגדרת טיפוס חדש: במקרה זה הגדרנו את הטיפוס **String** כמערך של 256 תווים.

בפרק הבא נכיר בפירוט את ההוראה typedef.

כעת ניתן להגדיר משתנים מסוג String :

```
String s1 = "hello";
String  s2 = "world";
```

תכנית דוגמא:

```
#include <stdio.h>

typedef char String[256];

void main ( )
{
        String first_name;
        String second_name;

        printf("Enter your first name: ");
        scanf("%s", first_name);
        printf("Enter your second name: ");
        scanf("%s", second_name);

        printf("Your full name is %s %s\n", first_name, second_name);

}
```

דוגמא להרצת התכנית:

```
Enter your first name: Louis
Enter your second name: Armstrong
Your full name is Louis Armstrong
```

התכנית מגדירה את הטיפוס String כמו קודם. בתחילת הפונקציה main מוגדרים שני משתנים מסוג String:

```
String first_name;
String second_name;
```

המשתנים נקראים מהקלט ע"י הוראות scanf עם מציין הטיפוס %s, אולם בניגוד למשתנים רגילים, משתני המחרוזת עצמם מועברים כפרמטרים ולא כתובותיהם:

```
printf("Enter your first name: ");
scanf("%s", first_name);
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

```
printf("Enter your second name: ");
scanf("%s", second_name);
```

הסיבה לכך היא שמחרוזת, המיוצגת כמערך, היא בעצמה כתובת.

נעסוק בכך בהרחבה בפרק 9, "**מחרוזות**". לאחר ביצוע הקלט, המחרוזות מודפסות ע"י הוראת printf, שוב עם מציין הטיפוס %s:

```
printf("Your full name is %s %s\n", first_name, second_name);
```

## פעולות על מחרוזות

בשפת C לא ניתן להציב מחרוזת אחת לשנייה ע"י s1=s2 או לבדוק שוויון ביניהן ע"י s1==s2.

לביצוע פעולות אלו קיימות פונקציות הספרייה string.h שנכיר אותן בפרק 9, "**מחרוזות**".

# סיכום

- בפרק זה סקרנו את המרכיבים הבסיסיים בשפה:

**טיפוסים ומשתנים**

לפני שימוש במשתנה יש להכריז על שמו וטיפוסו בתחילת התכנית. הטיפוסים העיקריים: שלם (int), תו (char), ממשי (float) .

**ביטויים**

ניתן לבצע פעולות שונות בין משתנים. אוסף משתנים ופעולות עליהם נקרא **ביטוי**. ראינו מספר סוגי ביטויים: ביטויים חשבוניים (x*y) , ביטויים לוגיים (Xinch<=10), ביטויי הצבה (Xcm=0).

**קבועים**

קבועים הם שמות של ערכים שאינם משתנים לכל אורך התכנית. ניתן להגדיר קבועים ע"י שימוש במילה const או ע"י הגדרתם ב**קדם-מעבד** (pre-processor) בהוראת #define.

**מילים שמורות**

המילים השמורות בשפה משמשות לציון טיפוסים והוראות בשפה ואינן ניתנות לשימוש אחר ע"י המתכנת. דוגמאות למילים שמורות: ,int, float, while.

- **קלט / פלט** של משתנים בתכנית מבוצע ע"י פונקציות הספרייה stdio.h. הכרנו את הפונקציות:

  **scanf()** לביצוע קלט של משתנים מטיפוסים שונים (מספרים ומחרוזות).

  **printf()** להדפסה מורכבת של טקסט משולב עם משתנים מטיפוסים שונים.

  **getchar()** ו- **putchar()** לקריאת ולהדפסת תו בודד.

- **לולאות** משמשות לביצוע פעולות חוזרות מספר מסוים של פעמים או כתלות בקיום תנאי כלשהו. לולאת **while** היא לולאה המתבצעת כל עוד תנאי הלולאה מתקיים.

- **מחרוזות** הן מערכי תווים בעלי תו סוף מחרוזת בסופם. הן ניתנות לקריאה ולהדפסה ע"י scanf() ו- printf() עם מציין הטיפוס %s.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום 1-5 שבעמ' 55-56.***

 *© כל הזכויות שמורות*

# 3. אבני היסוד

- ◄ מזהים (Identifiers)
- ◄ טיפוסי נתונים
- ◄ קבועים
- ◄ ליטרלים
- ◄ אופרטורים
- ◄ המרת טיפוסים
- ◄ סיכום
- ◄ תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

# מזהים (Identifiers)

מזהים הם שמות בתכנית המייצגים משתנים, קבועים, טיפוסים, פונקציות וכו'.

כללים לקביעת שמות מזהים:

1. מזהה הוא רצף של אותיות, ספרות והתו '_' (underscore) . רצף זה חייב להתחיל באות בשפה האנגלית או בתו '_'.
2. אורך מזהה אינן מוגבל
3. אין להשתמש במילים שמורות כמזהי טיפוסים, משתנים וכו'.
4. קיימת ההבחנה בין אותיות גדולות וקטנות. לדוגמא, המזהים top, Top, TOP הם מזהים שונים.
5. על שמות המזהים להיות משמעותיים. לדוגמא אם נרצה לתת שם למונה בתוכנית ניקרא לו counter.

- דוגמאות למזהים חוקיים:
  - x
  - y20
  - a_very_long_variable
  - _counter
- דוגמאות למזהים לא חוקיים:

| | |
|---|---|
| 7_Eleven | - מתחיל בספרה |
| hello! | - התו "!" לא חוקי |
| my-var | - התו "-" לא חוקי |

## הערות (Comments)

אם נרצה להוסיף לתכנית מלל אשר נרצה שהמהדר יתעלם ממנו (לדוגמא, הסברים על התוכנית או על פקודות מסוימות) נסמנו כ**הערה**. **הערה** מתחילה בצמד התווים */ ומסתיימת בצמד התווים /*. לדוגמא :

***/* this is a comment. */***
*int x = 5 * 2;*

לא ניתן לבצע קינון הערות - לדוגמא, במבנה ההערה הבא

```
/* .......... */ ......... */
```

ההערה מסתיימת כאן.

כל הקוד אשר מופיע לאחר סיום ההערה המסומן יחשב כחלק מהתוכנית עבור המהדר. כאשר סימן תחילת או סיום הערה נמצא בתוך מחרוזת, הוא לא יזוהה והמהדר לא יתייחס אליו.

> ***שיפור ב ++C :*** בשפת ++C קיים סוג נוסף של הערה, "הערת שורה": הערה זו מתחילה בצמד התווים // ונמשכת עד לסוף השורה. לדוגמא :
>
> int x = 5 * 2; **// a C++ comment**

## מילים שמורות (keywords)

מילים שמורות הן מילים בשימושה של השפה ואסורים לשימוש כשמות מזהים. רשימת המילים השמורות בשפת C :

| | | | |
|---|---|---|---|
| **auto** | **do** | **if** | **struct** |
| **break** | **double** | **int** | **switch** |
| **case** | **else** | **long** | **typedef** |
| **char** | **enum** | **register** | **union** |
| **const** | **extern** | **return** | **unsigned** |
| **continue** | **float** | **short** | **void** |
| **default** | **for** | **signed** | **volatile** |
| **goto** | **sizeof** | **static** | **while** |

מרכז ההדרכה 2000 © כל הזכויות שמורות www.mh2000.co.il

# טיפוסי נתונים

הטבלה הבאה מציגה את הטיפוסים הבסיסיים בשפת C :

| **טיפוס** | **תאור** |
|---|---|
| char | תו בודד |
| short | שלם קצר |
| int | שלם |
| long | שלם ארוך |
| float | ממשי |
| double | ממשי כפול |
| long double | ממשי ארוך |

### טיפוסים שלמים

**int** הוא הטיפוס הבסיסי לייצוג מספר שלם. אופן הגדרת משתנה מסוג שלם:

```
int     i;
```

i יכול כעת לקבל ערכים שלמים, חיוביים ושליליים לדוגמא:

```
i = 34;
i = -2456;
```

קיימים תתי-טיפוסים של שלם:

short - שלם קצר

long - שלם ארוך

כמו כן ניתן לציין עבור טיפוס שלם כלשהו שהוא מקבל ערכים חיוביים בלבד ע"י המציין **unsigned**. דוגמאות:

```
short       i1 =  - 23;
unsigned    i2 =  3000;
unsigned    short  i3 = 23;
```

## טיפוסים ממשיים

**float** הוא הטיפוס הבסיסי לייצוג מספר ממשי. אופן הגדרת משתנה מסוג ממשי:

```
float  f;
```

f יכול כעת לקבל ערכים ממשיים (ראה/י טבלה להלן):

```
f = 34.56;
f = -123.458997;
f = 23E+12;
```

קיימים טיפוסים ממשיים בעלי טווח מספרים וערך דיוק גדול יותר:

**double** - ממשי בעל דיוק כפול וטווח ערכים גדול

**long double** - תוספת דיוק וטווח ל- double

## טיפוסים תוויים

תווים מיוצגים במחשב ע"י טבלה הנקראת **טבלת ASCII**. הטבלה מכילה 256 תווים (0 עד 255) הכוללים ספרות, אותיות (גדולות וקטנות) וסימנים שונים. הטבלה במלואה מופיעה בנספח הספר.

דוגמאות לתווים בטבלת ASCII:

A 2 % * ! a . ~ )

הטיפוס הקיים בשפת C לייצוג תו בודד הוא **char**. אופן הגדרת משתנה תווי:

```
char  c;
```

וכעת c יכול לקבל ערכים מסוג תווי - ערך תווי מסומן ע"י שני גרשים משני צידיו:

```
c = 'a';
c = 'A';
c = '!';
```

<u>האם char הוא signed או unsigned?</u>

התשובה לשאלה תלויה במערכת עליה עובדים. כלומר הגדרת משתנה מסוג char כגון:

```
char ch;
```

אינה חד משמעית בשפת C אלא תלויית מערכת וסביבת פיתוח. דבר זה יכול ליצור הבדלים חדים בהידור והרצת תכניות על מחשבים שונים. לדוגמא, אם נגדיר את המשתנה כך

```
char ch = 255;
```

ערכו של המשתנה השלם x לאחר ההצבה

```
int x = ch;
```

יהיה בעל ערך 255 במערכת שבה char הוא unsigned ובעל ערך 1- במערכת שבה הוא signed!!!

## גדלי הטיפוסים

בשפת C אין הגדרה מדויקת לגודלי הטיפוסים הבסיסיים בשפה, כלומר למספר הבתים המוקצים למשתנה מסוג אותו הטיפוס.

גדלי הטיפוסים תלויים במחשב ובמערכת ההפעלה עליה עובדים.

לדוגמא, במחשב מבוסס Intel - Pentium, עם מערכת ההפעלה Windows 95 / NT גדלי הטיפוסים הבסיסיים הם:

| **טיפוס** | **מספר בתים** | **טווח ערכים** |
|---|---|---|
| char | 1 | -128..127 |
| short | 2 | -32KB .. 32KB |
| int | 4 | - 2GB .. 2GB |
| long | 4 | - 2GB .. 2GB |
| float | 4 | -3.4E +/- 38 .. 3.4E +/ - 38 |
| double | 8 | -1.7E +/- 308 .. 1.7E +/- 308 |
| long double | 8 | -1.7E +/- 308 .. 1.7E +/- 308 |

## enum

הוראת **enum** משמשת להגדרת משתנים היכולים לקבל ערכים מתוך תת-תחום מסוים. לדוגמא, אם משתנה מסוים מייצג את היום בשבוע, נוכל להגדיר תת - תחום של ימי השבוע:

```
enum Day { SUN=1, MON=2, TUE=3, WED=4, THU=5, FRI=6, SAT=7};
```

והגדרת משתנה מסוג ה- enum הנ"ל:

```
enum Day  today;
```

בדוגמא זו Day הוא שם תג (tag name) . הגדרת המשתנה מצוינת ע"י המילה enum ושם התג (Day) מיד אחריה. תכנית דוגמא:

```
#include <stdio.h>
void main()
{
	enum Day { SUN=1, MON=2, TUE=3, WED=4, THU=5, FRI=6, SAT=7};
	enum Day day1, day2;

	day1 = SUN;
	day2 = THU;
}
```

אם לא מצוין ערך של קבוע בתוך הגדרת ה- enum ערכו שווה לערך האיבר הקודם + 1. אם ערכו של האיבר הראשון לא מצוין ערכו הוא 0. לכן ניתן להגדיר את ה- enum הנ"ל בדרך מקוצרת:

```
enum Day { SUN=1, MON, TUE, WED, THU, FRI, SAT};
```

*הערה: אם משתנה מסוים מוגדר מסוג enum בעל תת-תחום נתון ובתכנית נותנים לו ערך שאינו מתת-התחום המהדר **אינו** מודיע על שגיאה, וכמו כן זו אינה שגיאה בזמן ריצה. כלומר מנגנון ה- enum אינו נאכף על המתכנת בשפת C, אלא משמש כאמצעי עזר תכנוני ותיעודי.*

מרכז ההדרכה 2000 © כל הזכויות שמורות 

## הגדרת טיפוס חדש ע"י typedef

בנוסף לטיפוסים הקיימים בשפת C, קיימת אפשרות להגדיר טיפוסים חדשים ע"י ההוראה **typedef**.

לדוגמא, ניתן להגדיר את ה- enum מהסעיף הקודם כטיפוס ובכך למנוע את הצורך בציון המילה enum בצירוף שם התג:

***typedef*** *enum { SUN=1, MON, TUE, WED, THU, FRI, SAT}* ***Day;***

***Day*** *today, yesterday, tomorrow;*

כעת Day הוא טיפוס עצמאי ולא רק שם תג.

דוגמא שימושית נוספת - הגדרת טיפוס בוליאני בעל 2 ערכים TRUE ו- FALSE:

```
typedef enum { FALSE=0, TRUE=1 } Boolean;
Boolean flag = FALSE;
```

שימוש שכיח בהגדרת טיפוסים חדשים בשפת C הוא לצורך כתיבת תוכנה עבור מחשבים שונים.

אחת הבעיות הקשות בהעברת תוכנה ממערכת אחת למערכת שנייה היא בהתייחסות לגדלי הטיפוסים.

הואיל ובשפת C אין הגדרה תקנית לגודלי הטיפוסים בשפה, מהדרים הרצים על מערכות שונות יתרגמו אחרת את התכנית.

אחת הדרכים להקל על מלאכת הסבת התוכנה היא להגדיר את הטיפוסים הבסיסיים מחדש ע"י typedef . דוגמא להגדרות כאלו:

```
typedef    char       CHAR;
typedef    int        INT;
typedef    unsigned   UNS;
```

ובתכנית עצמה משתמשים אך ורק בהגדרות אלו:

```
CHAR    c1;
INT     i;
```

כעת, אם נרצה להסב את התוכנה למחשב ומערכת הפעלה בהם הטיפוסים הבסיסיים שונים - למשל אם במערכת המקורית השלם הוא 32 סיביות ובמערכת החדשה הוא 16 סיביות, נבצע שינוי בהגדרת INT כך שגדלו לא ישתנה:

```
typedef    long            INT;
typedef    unsigned long   UNS;
```

וכעת אם long במערכת החדשה הוא 32 סיביות גדלו של INT יישאר זהה ביחס למערכת הישנה.

# קבועים

## קבועים

קבועים הם שמות של ערכים שאינם משתנים לכל אורך התכנית. כפי שכבר ראינו, ניתן להגדיר קבועים ע"י שימוש במילה const:

```
const int   MAX = 28;
```

או ע"י הגדרתם בקדם-מעבד בהוראת #define:

```
#define     MAX   28
```

מהו ההבדל בין השניים? במקרה הראשון אנחנו מגדירים תא בזיכרון בשם MAX שערכו הוא 28 ושלא ניתן לשנותו. במקרה השני לא מוגדר תא בזיכרון: זוהי הוראה לקדם-מעבד העובר על התכנית ומחליף כל מופע של MAX במספר 28.

הבדל נוסף הקיים בין 2 הצורות הוא במרחב השם (namespace) של הקבוע: קבוע המוגדר ע"י הקדם מעבד מוכר בכל קובץ התכנית, החל מהמקום בו הוגדר. קבוע המוגדר ע"י **const** לעומת זאת הוא בעל מרחב שם מוגבל יותר, עפ"י מקום הגדרתו, כפי שנראה בפרק 6, "**פונקציות**".

ניתן להגדיר בשתי הצורות קבועים מטיפוסים שונים. דוגמאות:

```
#define     MIN         -34.55
#define     STREET      "Ben Yehuda"

const char  STR[] = "hello";
const char  C = 'c';
const float NUMBER = 34.88;
```

> *הערה: כמוסכמה, נהוג לציין שמות קבועים באותיות גדולות (capitals).*


מרכז ההדרכה 2000 © כל הזכויות שמורות

# ליטרלים

**ליטרלים** הם ערכים המופיעים ישירות בקוד התכנית. הם יכולים להיות מטיפוסים שונים - שלם, ממשי, תווי או מחרוזת. טיפוס הליטרל נקבע ע"י המהדר עפ"י הערך.

לדוגמא, המספר 390 יובן ע"י המהדר כליטרל מטיפוס שלם, והערך 'f' יובן כליטרל מטיפוס תווי.

## ליטרלים שלמים

ליטרלים שלמים הם מספרים שלמים הנכתבים ישירות בקוד התכנית. לדוגמא, בהוראות

```
int x;
x = 34;
```

המספר 34 הוא קבוע מספרי ממשפחת השלמים. מהו טיפוסו המדויק? הטיפוס, אם לא מצוין אחרת הוא int . במידה ורוצים לציין שהוא מסוג long יש להוסיף סיומת "l" או "L" למספר, לדוגמא:

```
long x;
x = 34L;
```

ניתן לציין בסיס שונה מהבסיס העשרוני עבור שלמים: קידומת של 0 (אפס) בראש המספר מציינת שהמספר נתון בבסיס **אוקטלי** (בסיס 8), קידומת 0x מציינת שהוא בבסיס **הקסהדצימלי** (בסיס 16).

לדוגמא, ההוראות הבאות שקולות - בכולן מוצב הערך 34 (דצימלי) ל- x:

```
int x;
x = 34;    /* decimal */
x = 042;   /* octal */
x = 0x22;  /* hexa */
```

## ליטרלים ממשיים

ליטרלים ממשיים, בדומה לליטרלים שלמים, נכתבים ישירות בקוד התכנית. לדוגמא, בהוראות

```
float y;
y = 34.55;
```

המספר 34.55 הוא ליטרל מספרי ממשפחת הממשיים. מהו טיפוסו? אם לא צוין אחרת הטיפוס הוא double.

מכיוון שהליטרל מוצב למשתנה מסוג float מתבצע קיצוץ, ומהדרים מסוימים יציגו הודעות אזהרה.

ניתן לציין שטיפוס הליטרל הממשי הוא float ע"י הוספת סיומת f או F למספר, לדוגמא:

```
y = 34.55f;
```

במקרה זה לא תוצג הודעת אזהרה.

ליטרלים ממשיים ניתנים לכתיבה בצורה מעריכית ע"י ציון המעריך בתוספת האות e או E.

לדוגמא, המספר 34.55 ניתן לרישום כ- 3.455E1 או 3455E-2 או 0.3455E2 .

צורת רישום זו מתייחסת לחזקה של בסיס 10:

| **ייצוג הליטרל ב- C** | **ערך** |
|---|---|
| 3.455E1 | $3.455 * 10^{1}$ |
| 3455E-2 | $3455 * 10^{-2}$ |
| 0.3455E2 | $0.3455 * 10^{2}$ |

## ליטרלים תוויים

ליטרלים תוויים מצוינים ע"י התו המיוצג ושני גרשים משני צידיו. לדוגמא, 'x' מציין את ערך ה- ASCII של התו x (טבלת ה- ASCII נתונה בנספח הספר).

ליטרלים תוויים שייכים למעשה למשפחת השלמים: הם משמשים בפעולות חשבוניות ולוגיות כמספרים שלמים לכל דבר כשתחום הערכים שלהם מוגבל (בית בודד).

קיימים ליטרלים תוויים מיוחדים המשמשים בעיקר בהדפסה. שניים מהתווים המיוחדים הכרנו בפרקים הקודמים: '\n' כתו שורה חדשה, ו- '\t' כטאב בפעולות הדפסה.

התו '\' מציין משמעות מיוחדת עבור תווים מסוימים. חלק מתווים אלו נתון בטבלה הבאה (הרשימה המלאה נמצאת בנספח):

| סימון | ערך ה- | שם | שם התו |
|---|---|---|---|
| \n | 10 | NL (LF) | Newline |
| \t | 9 | HT | Tab |
| \\ | 92 | \ | Backslash |
| \' | 39 | ' | Single quotation mark |
| \" | 34 | " | Double quotation mark |
| \0 | 0 | NUL | Null character |

לדוגמא, כדי להדפיס את השורות הבאות:

```
First   line
Second  line
Third   line
```

נכתוב:

*printf("First**t**line**n**Second**t**line**n**Third**t**line");*

ואם נרצה להדפיס את התו " (גרשיים) שערך ה- ASCII שלו הוא 34 נוכל לעשות זאת במספר דרכים:

```
putchar('\"');     /* as character with backslash*/
putchar(34);       /* direct decimal ASCII value */
putchar('\x22');   /* direct hexadecimal ASCII value */
putchar('\42');    /* direct octal ASCII value */
printf("\"");             /* as part of a string, with backslash*/
```

תרגיל: כיצד יודפס התו '\' עצמו? הצע/י 5 דרכים.

# אופרטורים

**אופרטורים** הם סימנים המוצבים ליד ובין נתונים, ומורים למהדר על ביצוע פעולה מסוימת.
האופרטורים נחלקים למספר קבוצות:

- אופרטורים חשבוניים
- אופרטורים לוגיים
- אופרטורי סיביות
- אופרטורי הצבה

## אופרטורים חשבוניים

האופרטורים החשבוניים פועלים על טיפוסים מספריים שלמים או ממשיים:

| אופרטור | משמעות |
|---|---|
| + | חיבור |
| - | חיסור |
| * | כפל |
| / | חילוק |
| ++ | קידום משתנה ב- 1 |
| -- | חיסור משתנה ב- 1 |
| % | שארית החלוקה (מודולו) - בשלמים בלבד |

האופרטורים מופעלים עפ"י טיפוסי האופרנדים - שלמים או ממשיים. לדוגמא:

```
int    i;
float  f;

i = 4 / 5;          /* i=0 */
f = 4 / 5;          /* f=0.0 */
f = 4.0 / 5.0;      /* f=0.8 */
```

כלומר, תוצאת פעולת החילוק שונה בין שלמים לממשיים: 4/5 הוא חלוקת שלמים שתוצאתה השלם 0. לעומת זאת 4.0/5.0 היא חלוקת ממשיים שתוצאתה ממשית - 0.8.

<u>אופרטורי קידום וחיסור: "++" ו- "--"</u>

האופרטור ++ מבצע קידום ב- 1 והאופרטור – מבצע חיסור ב- 1 של משתנה שלם. שני האופרטורים יכולים להופיע משני צדי המשתנה, וקיים הבדל בין פירוש 2 הצורות:

– אם האופרטור נמצא מימין למשתנה (postfix) , לדוגמא:

```
int  i = 9;
int j =  i ++;
          /* ➔ j=9, i=10 */
```

אז הדבר שמתבצע הוא שימוש בערך של המשתנה בביטוי ולאחר מכן קידום ב- 1 .

– לעומת זאת, אם האופרטור נמצא משמאל למשתנה (prefix) , לדוגמא:

```
int  i = 9;
int j =  ++ i ;
          /* ➔ j=10, i=10 */
```

מתבצעת הוספה של 1 ל - i, ולאחר מכן הצבה ל- j.

> *הערה: כאשר הביטוי בו נמצא אופרטור קידום/חיסור הוא משפט עצמאי ולא כחלק מביטוי אחר אין הבדל בין שני האופנים.*

תכנית דוגמא:

```
#include <stdio.h>
void main ( )
{
	int	i = 5;
	int	j = 0;

	printf("%d", i++);	/* output: 5,	i = 6	*/
	printf("%d", --j);	/* output: -1,	j = -1	*/
	printf("%d", j = i++);  /* output: 6,	j = 6	i=7 */
}
```

<u>אופרטור שארית החלוקה %</u>

האופרטור % פועל רק על טיפוסים שלמים, ומבצע פעולת מודולו, כלומר, הוא מחזיר את השארית של תוצאת החלוקה. לדוגמא:

```
int	s;

s  =  10 % 3;	/* s = 1 */
s  =  8 % 8;	/* s = 0 */
s  =  8 % 0;	/* Error! */
s  =  8 % 9;	/* s = 8 */
s  = -8 % 9;	/* s = -8 */
s  =  8 % -9;	/* s = 8 */
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## אופרטורים וביטויים לוגיים

אופרטורי היחס מרכיבים ביטויים לוגיים שתוצאתם היא אמת או שקר:

| **אופרטור** | **משמעות** |
|---|---|
| x == y | x שווה ל- y |
| x != y | x שונה מ- y |
| x > y | x גדול מ- y |
| x >= y | x גדול מ- y או שווה לו |
| x < y | x קטן מ- y |
| x <= y | x קטן מ- y או שווה לו |

ניתן להרכיב ביטויים ממספר ביטויים בסיסיים ע"י האופרטורים הלוגיים הבאים:

| **אופרטור** | **משמעות** |
|---|---|
| && | וגם |
| \|\| | או |
| ! | היפוך |

דוגמאות:

x >= y || w == z - x גדול או שווה ל- y **או** w שווה ל-z

x == y && y == z - x שווה ל- y **וגם** y שווה ל- z

x == y && !(y == z) - x שווה ל- y **וגם** y שונה מ- z

הביטויים הלוגיים מופיעים בדרך כלל כחלק ממשפטי תנאי, לדוגמא:

```
if(x > y)
        printf("x is bigger than y");
else
        if(y > x)
                printf("y is bigger than x");
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

```
else
	printf("x and y are equals");
```

<u>טיפוס שלם כתחליף לטיפוס בוליאני</u>

תוצאת ביטוי לוגי יכולה להיות **אמת** (true) או **שקר** (false). בשפות מסוימות קיים טיפוס בוליאני שאלו שני הערכים היחידים שהוא מקבל. בשפת C לא קיים טיפוס כזה - טיפוס השלם משמש לצורך כך.

בשפת C נקבע שערך 0 של משתנה שלם מציין ערך "**שקר**" בביטוי לוגי, ו- 1 או כל ערך שונה מ- 0 מציין ערך "**אמת**". לדוגמא:

```
int		i;
int		x=5, y=5;

i = (x > y); /* i = 0 */
i = (x == y);	/* i = 1 */
```

וכן ניתן להשתמש בתוצאת הביטוי במשפט תנאי:

```
i = (x == y);
if(i)
	printf("x is equal to y");
```

מכיוון ששלם מייצג גם טיפוס בוליאני ניתן גם להשתמש במשתנים שלמים בתנאי הלולאה. לדוגמא:

```
#include <stdio.h>
void main ( )
{
	int	i = 4;
	while(i)
	{
		i--;
		printf("i = %d\t",i);
	}
}
```

הפלט:

```
i = 3   i = 2   i = 1   i = 0
```

<u>משפט תנאי מקוצר</u>

האופרטור "?:" הוא אופרטור טרינרי - כלומר בעל שלושה אופרנדים - המשמש במקרים מסויימים כקיצור למשפט if-else. לדוגמא, המשפט

```
if(x > y)
	max = x;
else
	max = y;
```

יכול להיכתב בקיצור ע"י

```
max = x > y ? x : y;
```

תחביר האופרטור "?:" :

<ביטוי-לוגי> **?** <ביטוי 1> **:** <ביטוי 2>

הביטוי הלוגי שמופיע לפני הסימן "?" הוא ביטוי שתוצאתו "אמת" או "שקר". אם התוצאה היא "אמת", תוצאת המשפט היא <ביטוי 1>, אחרת תוצאתו היא <ביטוי 2>.

ניתן להשתמש בביטוי זה באופן מקוצר כחלק מהוראות אחרות. לדוגמא, אם נרצה להדפיס את המקסימום מבין 2 המשתנים שלעיל מבלי להשתמש במשתנה הנוסף max נוכל לכתוב:

```
printf("The maximum is = %d", x > y ? x : y);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## אופרטורים הפועלים על סיביות (bitwise operators)

בשפת C קיימים אופרטורים הפועלים על סיביות של טיפוסים שלמים (שלם, שלם קצר/ארוך, תו). קבוצה זו כוללת 6 אופרטורים:

| **<u>אופרטור</u>** | **<u>משמעות</u>** |
|---|---|
| & | AND |
| \| | OR |
| ^ | XOR |
| ~ | NOT, משלים ל- 1 (one's complement) |
| << | הזזה שמאלה |
| >> | הזזה ימינה |

ארבעת האופרטורים הראשונים מבצעים פעולות לוגיות בינריות על סיביות. שני האופרטורים האחרונים מבצעים הזזה של הסיביות במשתנים.

<u>פעולות לוגיות על סיביות</u>

בפעולות לוגיות בין סיביות מתקיימים הכללים הבאים:

פעולת AND:

*1 & 1 = 1*
*1 & 0 = 0*
*0 & 1 = 0*
*0 & 0 = 0*

פעולת OR:

*1 | 1 = 1*
*1 | 0 = 1*
*0 | 1 = 1*
*0 | 0 = 0*

פעולת XOR:

*1 ^ 1 = 0*
*1 ^ 0 = 1*
*0 ^ 1 = 1*
*0 ^ 0 = 0*

פעולת NOT:

*~1 = 0*
*~0 = 1*

נניח שנתון השלם x בבסיס 16 :

```
int x = 0x52;
```

במערכת שבה השלם הוא מגודל 4 בתים, כלומר 32 סיביות, המשתנה ייראה כך בזיכרון :

| x = | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 | 0010 |

השורה התחתונה מייצגת את הסיביות של המספר, ובשורה העליונה מוצגים הערכים בבתים בייצוג הקסה. נגדיר משתנה נוסף :

```
int y = 0x8472;
```

y ייראה כך בזיכרון :

| y = | 0 | 0 | 0 | 0 | 8 | 4 | 7 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 1000 | 0100 | 0111 | 0010 |

**פעולת AND** בין שני המשתנים מסומנת כ- x & y . תוצאתה היא משתנה שלם שבו כל סיבית היא תוצאת הפעולה AND הבינרית בין שתי הסיביות המתאימות ב- x וב- y. לדוגמא, אם נבצע

```
int z = x & y;
```

מה יהיה ערכו של z ?


מרכז ההדרכה 2000 © כל הזכויות שמורות

| x = | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 00000 | 0000 | 0000 | 0101 | 0010 |

&

| y = | 0 | 0 | 0 | 0 | 8 | 4 | 7 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 1000 | 0100 | 0111 | 0010 |

=

| z = | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 | 0010 |

כלומר z = 0x52.

באופן דומה:

| מבצע פעולת **OR** בינרי,

^ מבצע **XOR** בינרי,

~ מבצע פעולת NOT בינרית בשיטת המשלים ל- 1.

***דוגמאות נוספות מובאות בעמ' 74.***

<u>פעולות הזזה</u>

פעולות ההזזה גורמות להזזת כל סיביות הנתון מספר מקומות ימינה או שמאלה תוך מילוי המקומות החדשים ב- 0 או ב- 1.

האופרטור << מבצע הזזה שמאלה, והאופרטור >> מבצע הזזה ימינה. צורת הסימון:

x >> 2     x מוזז שני מקומות ימינה

x << 2     x מוזז שני מקומות שמאלה

לדוגמא, אם x הוא כמו קודם:

| x = | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 | 0010 |

אז תוצאת הפעולה x << 2 היא

| x << 2 = | 0 | 0 | 0 | 0 | 0 | 1 | 4 | 8 |
|---|---|---|---|---|---|---|---|---|
| | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 0100 | 1000 |

כלומר התוצאה היא 0x148. פעולה זו זהה להכפלת x ב- 4.

כאשר הערך מוזז ימינה, המקומות משמאל ממולאים ב- 0 עבור מספרים חיוביים וב- 1 עבור מספרים שליליים.

לדוגמא, אם ערכו של x הוא 4- , הוא מיוצג במחשב בשיטת המשלים ל 2- ע"י :

| x = | F | F | F | F | F | F | F | C |
|---|---|---|---|---|---|---|---|---|
| | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1100 |

תוצאת הפעולה x >> 2 היא

| x>>2 | F | F | F | F | F | F | F | F |
|---|---|---|---|---|---|---|---|---|
| | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 |

כלומר, ערכו יהיה 1- . פעולה זו זהה לחילוק x ב- 4.

***דוגמאות נוספות מובאות בעמ' 75.***


מרכז ההדרכה 2000 © כל הזכויות שמורות

# אופרטורי הצבה

האופרטור "=" הוא אופרטור ההצבה בשפת C. לדוגמא:

```
int x;
x = 5;
```

הערך 5 מוצב למשתנה x. ניתן גם להציב ערך למשתנה תוך כדי הגדרתו, לדוגמא:

```
int  x=5;
double  y=6.2,  z=1.45E10;
```

אופרטור ההצבה מחזיר את הערך המושם, לכן ניתן להדפיסו או להציבו בשרשרת למשתנה נוסף. דוגמאות:

```
1. printf("%d", x = y);

2. z = x = y;

3. w = z = x = y;
```

יש להבחין בין אופרטור ההצבה "=" לבין אופרטור השוויון "==". ההוראה

```
x = y;
```

היא הוראת הצבה של הערך של y למשתנה x. לעומת זאת, ההוראה

```
x == y
```

היא ביטוי לוגי שתוצאתו "אמת" או "שקר", עפ"י ערכי המשתנים, שמשמשת בדרך כלל במשפטי תנאי (if) או בלולאות.

בעיקר יש לשים לב לכך שמכיוון שביטוי הצבה מחזיר ערך, המהדר עלול שלא להתריע בפני בלבול אפשרי. לדוגמא, במקרה הבא

```
int x=5, y=2;
if(x=y)
        printf("x and y are equal");
```

לא תתקבל מהמהדר הודעת שגיאה! (מהדרים מסוימים יציגו הודעת אזהרה) זאת מכיוון שביטוי ההצבה מחזיר את ערך ההצבה, וערך זה חוקי כערך בוליאני. כלומר בפועל יבוצע

```
if(2)
        printf("x and y are equal");
```

ותודפס הודעת השוויון!

<u>ביטויי הצבה מקוצרים</u>

ניתן לכתוב ביטויי הצבה באופן מקוצר: למשל את הביטוי

```
x  =  x + 5;
```

ניתן לכתוב כך:

```
x += 5;
```

קיצור זה אפשרי עבור כל האופרטורים הבאים:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| + | - | * | / | % | << | >> | & | ^ | \| |

חמשת האופרטורים משמאל הם האופרטורים החשבוניים, וחמשת האופרטורים מימין הם אופרטורים הפועלים על סיביות. דוגמאות:

```
int x=2, y=8, z=0;

x += 3;     /* x = 5 */
z -= x+y;   /* z = -13 */
z /= x;     /* z = -13/5 = -2 */
z %= 8;     /* z = -2 % 8 = -2 */
z <<= 2;    /* z = -8 */
z &= 0;     /* z = 0 */
```

שים/י לב: ההוראה

```
x = - 3;
```

אינה הצבה מקוצרת אלא הצבה של הערך 3- למשתנה x. בהצבה מקוצרת מופיע האופרטור משמאל לפעולת ההצבה:

```
x -= 3;
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## האופרטורים ב- C עפ"י קדימויות

***הטבלה שבעמ' 78 מציגה את האופרטורים ב- C עפ"י קדימויות.***

אופרטורים בשורה נתונה בטבלה הם בעלי קדימות גבוהה מאופרטורים בשורה נמוכה יותר. לדוגמא, הביטוי

*3 + 4 * 5*

יחושב ע"י הכפלת 4 ב- 5 ואח"כ חיבור ל- 3. הביטוי שקול ל-

*3 + (4 * 5)*

הסוגריים ( ) הם בעלי הקדימות הגבוהה ביותר בטבלה. לכן, אם רוצים לתת קדימות לביטוי מסויים, ניתן להקיפו בסוגריים. לדוגמא:

*(3 + 4) * 5*

העמודה השמאלית קובעת את כוון הפעלת האופרטור. לדוגמא, הביטוי

*x1 / x2  / x3*

יחושב ע"י חילוק x1 ב- x2, ואחר כך חילוק התוצאה ב- x3. כלומר, אם נשתמש בסוגריים לציון הקדימות, הביטוי השקול הוא

*(x1 / x2) / x3*

ולא

*x1 / (x2 / x3)*

זאת מכיוון שלאופרטור / סדר ביצוע **משמאל לימין**. לעומת זאת, הביטוי

*x1 = x2 = x3 + 1;*

יחושב ע"י הצבת (x3+1) ל- x2, ואחר כך הצבת התוצאה ל- x1. זאת מכיוון שלאופרטור ההצבה "=" סדר ביצוע **מימין לשמאל**.

דוגמאות:

```
#include <stdio.h>
void main ( )
{
	int	i = 2,  j=6, k=10;
	int	s;

	s = i + j * k;			/* s = 62 */

	s = (i + j)* k;		/* s = 80 */

	s = k / i * j;		/* s = 30 */

	s = j / i++ ;			/* s = 3  */
}
```

בביטוי האחרון, לאופרטור ++ יש עדיפות על פני אופרטור החילוק /, אולם מכיוון שהוא בצורת postfix הוא מבוצע רק לאחר חישוב הביטוי כולו.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# המרת טיפוסים

## המרה מרומזת

ניתן לכתוב ביטויים בהם מעורבים משתנים מטיפוסים שונים - במקרה זה מבוצעת המרה אוטומטית לטיפוס המתאים. בדרך כלל נקבל אזהרה מהמהדר על ביצוע המרה כזו.

לדוגמא, המרה מרומזת של מספר ממשי לשלם גורמת לקיצוץ חלק השבר ולאיבוד מידע, ולכן גוררת הודעת אזהרה מהמהדר:

```
int    i;
float  f=14.5;

i = f;        /* i = 14 */
```

בהמרת שלם לממשי אין איבוד מידע:

```
int    i=14;
float  f;

f = i;        /* f = 14.0 */
```

בנוסף לפעולת הצבה, המרה מרומזת יכולה להתרחש במהלך ביטויים חשבוניים:

```
float f;

f = 4 / 5;    /* f=0.0 */
f = 4.0 / 5;  /* f=0.8 */
```

הסבר: בביטוי הראשון מבוצעת פעולת חילוק שלמים שתוצאתה היא 0. רק לאחר מכן התוצאה מומרת לממשי 0.0. כלומר ההמרה מבוצעת באופרטור ההצבה.

בביטוי השני, אופרטור החילוק / נדרש לבצע חילוק של ממשי בשלם - לצורך כך השלם מומר תחילה לממשי ורק אז מתבצעת פעולת החילוק.


מרכז ההדרכה 2000 © כל הזכויות שמורות

ככלל, בביטוי מעורב משתנה מטיפוס "נמוך" יותר מומר לטיפוס ה"גבוה". טבלת הקדימויות לצורך המרה:

| קדימות | טיפוס |
|---|---|
| 1 | long double |
| 2 | double |
| 3 | float |
| 4 | long |
| 5 | int |
| 6 | short |
| 7 | char |

כיצד מומר משתנה מטיפוס נמוך לטיפוס גבוה? הטבלה הבאה מתארת מה מתבצע בפועל בהמרה:

| מ- | ל- | אופן ההמרה |
|---|---|---|
| int | char | מודולו 256 (התייחסות לבית הנמוך) |
| float / double | char | קיצוץ השבר, מודולו 256 |
| float / dubl e | int | קיצוץ השבר, מודולו גודל int |
| double | float | חצי של הספרות המשמעותיות |


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## המרה מפורשת (casting)

לפעמים נדרשת המרה מפורשת בתכנית כדי לאלץ פרשנות מסוימת של הביטוי החשבוני. לדוגמא, בתכנית הבאה קיימת בעיה:

```
float f;
int i = 5, j=12;
f =  j / i;                /* f = 2.0 */
```

כדי לאלץ פעולת חילוק בממשיים נכתוב:

```
float f;
int i = 5, j=12;
f =    j / (float) i;         /* f = 2.4 */
```

פעולת המרה יזומה בין טיפוסים מוגדרת כך:

*(**טיפוס**) ביטוי*

תוצאת הביטוי תומר לטיפוס המצוין.


מרכז ההדרכה 2000 © כל הזכויות שמורות

## האופרטור sizeof

האופרטור sizeof הוא אופרטור אונרי המחזיר את הגודל בבתים (bytes) של ביטוי, טיפוס או משתנה. אופן השימוש:

1) **sizeof(***ביטוי***)**

2) **sizeof(***טיפוס***)**

3) **sizeof(***משתנה***)**

אופרטור זה תמיד מחזיר שלם חיובי. לדוגמא, תכנית להדפסת הגדלים של מספר טיפוסים בסיסיים ב- C:

```
#include <stdio.h>
void main ( )
{
	printf("the size of integer is %d\n", sizeof(int));
	printf("the size of short is %d\n", sizeof(short));
	printf("the size of long is %d\n", sizeof(long));
	printf("the size of double is %d\n", sizeof(double));
}
```

פלט התכנית, כפי שהתקבל במערכת הפעלה Windows על מחשב Intel-Pentium :

```
the size of integer is 4
the size of short is 2
the size of long is 4
the size of double is 8
```

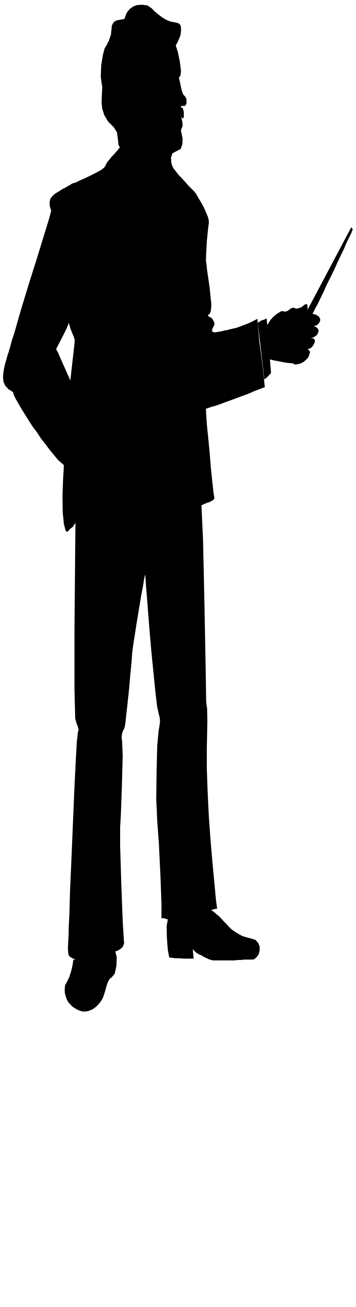
# סיכום

- **מזהים** (Identifiers ) הם שמות בתכנית המייצגים משתנים, קבועים, טיפוסים, פונקציות ותוויות בתוכנית.
- **הערות** (Comments) הן מלל שהמהדר מתעלם ממנו ובדרך כלל משמשות להסברים על התוכנית או על פקודות מסוימות. הערה מתחילה בצמד התווים /* ומסתיימת בצמד התווים */.
- **מילים שמורות** (keywords) הן מילים בשימושה של השפה. אסור לתת שמות מזהים הזהים למלים אלה.
- **הטיפוסים** (types) מציינים את סוג הנתונים בתכנית. הטיפוסים נחלקים ל- 3 קבוצות עיקריות: שלמים, תוויים וממשיים. ניתן בנוסף להגדיר טיפוסים חדשים ע"י שימוש בהוראה **typedef**. **enum** משמש להגדרת תת-תחום של מספרים שלמים.
- **קבועים** הם שמות המייצגים ערכים המופיעים בקוד התכנית. ניתן להגדיר שמות של קבועים ע"י המהדר תוך שימוש בהוראת const , או ע"י הקדם-מעבד בהוראת #define.
- **ליטרלים** הם ערכים הנכתבים ישירות בקוד ויכולים להיות טיפוסים שונים: שלמים, ממשיים, תוויים ותתי-סוגים שלהם.
  - ליטרלים שלמים יכולים להיות בבסיס דצימלי, אוקטלי או הקסהדצימלי
  - ליטרלים ממשיים יכולים להיכתב בצורה עשרונית או מעריכית.
  - ליטרלים תוויים מצוינים ע"י התו ושני גרשים משני צידיו. תווים מיוחדים מצוינים בצירוף התו \ , לדוגמא, '\n', '\b', '\\'.
- **אופרטורים** הם סימנים המוצבים ליד ובין נתונים ומורים למהדר על ביצוע פעולה מסוימת. האופרטורים נחלקים למספר קבוצות: חשבוניים, לוגיים, הצבה, הצבה+חשבוניים, אופרטורי סיביות (bitwise operators) ואחרים (המרה, sizeof). לאופרטורים יש עדיפויות שונות - סדר הפעלתם בביטוי תלוי בעדיפות.
- **המרת טיפוסים** של משתנים מתרחשת בשני מקרים: באופן מרומז (implicit) בכל ביטוי בו משולבים נתונים מטיפוסים שונים, או במפורש (explicit) ע"י אופרטור ההמרה (type). המרה מפורשת נקראת גם casting.

# תרגילי סיכום

***בצע/י את תר' 1-3 שבעמ' 83-84.***

# 4. קלט / פלט



- קלט / פלט של תו בודד
- קריאת תווי קלט וניתוחם
- פלט לפי תבנית ע"י printf
- קלט לפי תבנית ע"י scanf
- קלט / פלט של מחרוזות
- ניתוב קלט / פלט (IO Redirection)
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

# קלט / פלט של תו בודד

כפי שראינו בפרק 2, "**הכרת שפת C**", קריאת וכתיבת תו בודד מבוצעת ע"י הפונקציות getchar() ו- putchar().

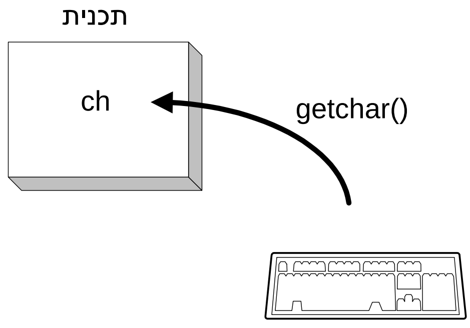
שתי אלה מוגדרות בספרייה התקנית של C כפונקציות מקרו (באמצעות הקדם מעבד) בקובץ stdio.h.

<u>getchar()</u>

זוהי פקודת הקלט הפשוטה ביותר אשר בעזרתה ניתן לקרוא תו אחד מהקלט הסטנדרטי (בדרך כלל המקלדת). פונקציה זו מחזירה את תו הקלט הבא בכל פעם שהיא נקראת.

לדוגמא, בכדי לקרוא תו מהקלט נבצע:

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




**שאלה**: מדוע התו מוגדר כ- int ולא כ- char?

**תשובה**: סוף הקלט מצוין ע"י הקבוע EOF המוגדר בספרייה stdio.h כך:

```
#define EOF      (-1)
```

בהגעה לסוף הקלט, הפונקציה getchar() מחזירה את EOF. משום כך על הערך המוחזר של הפונקציה להיות מטיפוס שלם, הכולל גם את תחום הערכים השליליים.

לדוגמא, הלולאה הבאה מונה את כלל התווים שנקראו מהקלט:

```
while (getchar()!=EOF)
        ++chars_no;
```

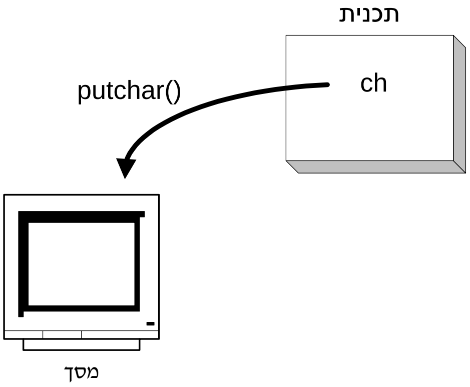
**שאלה**: כיצד מסמנים סוף קובץ קלט ע"י המקלדת?

**תשובה**: במערכת ההפעלה Windows ע"י **Ctrl-Z**, וב- Unix ע"י **Ctrl-D**.

## putchar()

זוהי פקודת הפלט הפשוטה ביותר אשר כותבת לפלט הסטנדרטי ( בדרך כלל המסך ) את התו הנתון לה כפרמטר:

*putchar(ch);*




לדוגמא, ההוראה

*putchar('A');*

תדפיס:

```
A
```

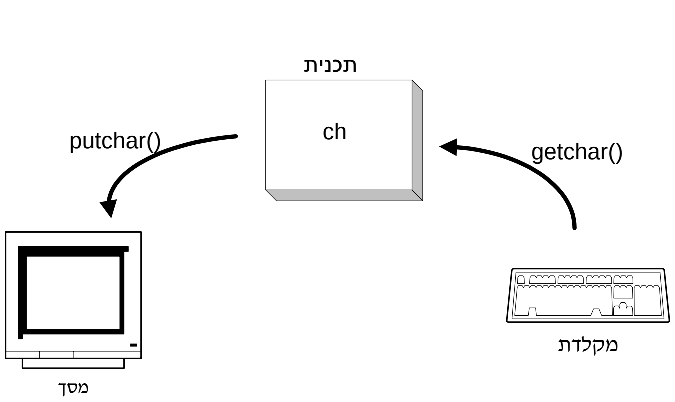
'A' הוא ערך ASCII של התו A, כלומר 65. לכן ניתן להדפיס את התו A גם ע"י ההוראה:

*putchar(65);*

בדומה ל- getchar(), גם putchar() מטפלת בטיפוס int ולא char בכדי לאפשר הדפסת תו סוף קובץ, EOF.

תכנית הדוגמא הבאה קולטת תו מהמשתמש (באמצעות המקלדת) ומדפיסה אותו למסך:

```
#include <stdio.h>
void main ( )
{
	int ch;

	ch = getchar();
	putchar(ch);
}
```




 מרכז ההדרכה 2000 © כל הזכויות שמורות

# קריאת תווי קלט וניתוחם

נכתוב מספר תכניות למניית תווים, שורות ומילים בקבצי טקסט. בתכניות אלו נעשה שימוש בפונקציות קלט פלט, ובמרכיבים בסיסיים בשפת C שהכרנו עד כה.

## תכנית למניית תווים

התכנית הבאה מונה את מספר תווי הקלט שנקראו ומדפיסה מספר זה:

```
/* file: count_chars.c */
#include <stdio.h>
void main ()
{
	long chars_no=0;

	while (getchar()!=EOF)
		++chars_no;
	printf("Read %ld chars\n", chars_no);
}
```

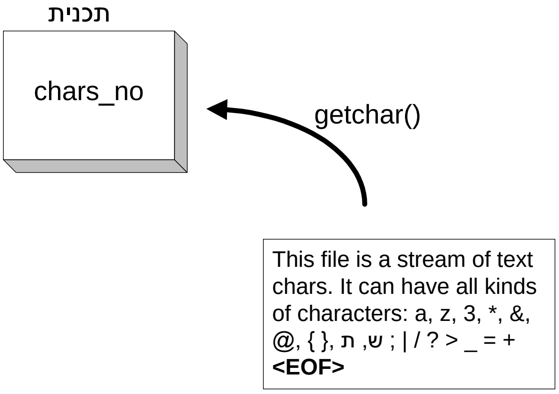
אם נריץ את התכנית ונקליד את הקלט הבא:

```
This file is a stream of text chars. It can have all
kinds of characters: a, z, 3, *, &, @, { }, ת ,ש ; |
/ ? > _ = +
```

יתקבל הפלט:

```
Read 118 chars
```

<u>הסבר</u>: התכנית קוראת תווים מהקלט - שהוא זרם תוים - עד להגעה לתו מסיים קובץ (EOF):




- התכנית קוראת את התווים וסופרת אותם ע"י המשתנה chars_no. בהגעה לסוף הקובץ מודפס מספר התווים שנקראו - כלומר גודל קובץ הקלט.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

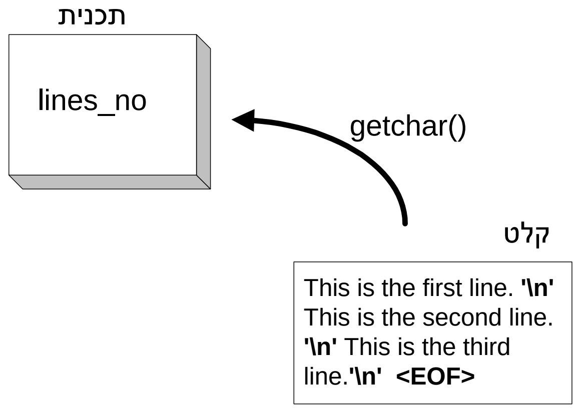
- המשתנה המונה את התווים chars_no הוא מטיפוס long כדי לאפשר טיפול במספר גדול של תווים.
- האות l במחרוזת הבקרה "%ld" מציינת ל- printf שהפרמטר הוא מטיפוס long.

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-2 שבעמ' 89.***

## מניית שורות

הקלט הוא זרם תווים המסתיים בתו מסיים קובץ (EOF). בתוך זרם התווים, נמצא תו מיוחד ('\n') המציין מעבר שורה חדשה :




נכתוב תכנית למניית מספר השורות בקלט.

לספור את השורות פירושו לספור תווי "שורה חדשה" בקלט, כלומר כמה פעמים הופיע התו מסיים השורה ('\n') בזרם הטקסט שבקלט.

אלגוריתם התכנית :

***משתנים : c - התו הנקרא הבא, lines_no - מונה השורות***

***אתחול: lines_no מאותחל ל- 0***

***כל עוד התו הנקרא, c , אינו תו סוף קובץ***

***אם c תו סוף שורה, הוסף 1 ל- lines_no***

***הדפס את lines_no***

קוד התכנית :

```
/* file: count_lines.c */
#include <stdio.h>
void main ()
{
	int c;
	int lines_no=0;
	while ((c=getchar())!=EOF)
	{
		if (c=='\n')
			++lines_no;
	}
	printf("Read %d lines\n",lines_no);
}
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

אם נריץ את התכנית ונקליד את הקלט הבא:

```
This is the first line.
This is the second line.
This is the third line.
```

יתקבל הפלט:

```
Read 3 lines
```

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 2-1 שבעמ' 91.***

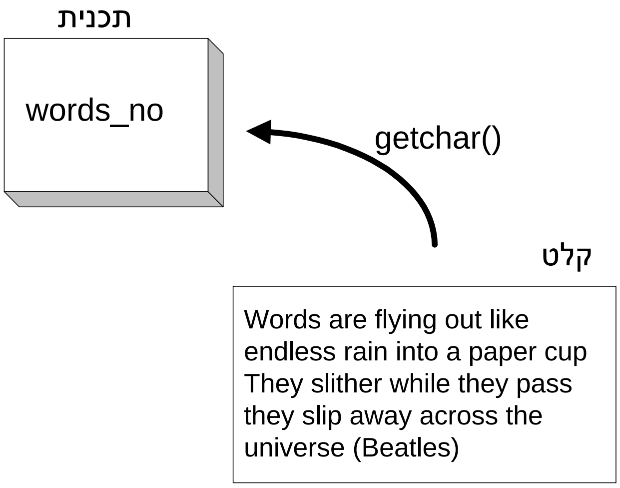

*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## מניית מילים

כדי לספור מילים צריך להגדיר מילה. אנו נגדיר מילה בצורה פשוטה:

<u>**מילה** היא סדרת תווים רצופה שלא מכילה בתוכה תו "לבן" מכל סוג שהוא: תו רווח, טאב או תו שורה חדשה.</u>

בפועל גם תווי פיסוק מפרידים בין מילים - אנו נתעלם מהם בשלב זה. למנות מילים פירושו למנות כמה פעמים הופסק זרם התוים ע"י תו לבן - כלומר תו רווח, טאב או תו שורה חדשה:




<u>הרעיון</u>: נגדיר משתנה מצב (state) שיציין במהלך קריאת התווים אם אנו בתוך מילה (INSIDE) או מחוץ לה (OUTSIDE).

בכל מעבר ממצב OUTSIDE למצב INSIDE נקדם את מונה המילים.

האלגוריתם:

***משתנים: state - משתנה המצב, c - התו הנקרא מהקלט, words_no - מונה המלים***

***אתחול: state <-- OUTSIDE, words_no <-- 0***

***כל עוד התו הנקרא, c, אינו תו סוף קובץ***

***אם c הוא תו "לבן"***

***state <-- OUTSIDE***

***אחרת, אם state הוא OUTSIDE***

***state <-- INSIDE***

***הוסף 1 ל- words_no***

***הדפס את words_no***

***קוד התכנית והסבר מובאים בעמ' 93. קרא/י סעיף זה בספר ובצע/י את תר' 1-3 שבעמ' 94.***

# *פלט לפי תבנית ע"י printf*

פונקצית הספריה **printf** משמשת להדפסת פלט מורכב: היא כוללת אפשרויות לשילוב טקסט עם ערכי משתנים, רווח, הצמדה ואפשרויות נוספות.

הטכניקה שבה היא פועלת כוללת מחרוזת בקרה ופרמטרים להדפסה:

**int printf ("מחרוזת בקרה", <פרמטר1>, <פרמטר2>, ...);**

מחרוזת הבקרה מורה ל- **printf** כיצד והיכן להציב את הפרמטרים בשורת הפלט.

היא כוללת את המלל שברצוננו להציג, ביחד עם מצייני הפורמט (format specifiers) המתארים את הטיפוס והמיקום של הפרמטרים. לדוגמא:

***printf**("The **%d** exam grades are **%f** and **%f**", 2, 87.5, 93.4);*

הפלט:

```
The 2 exam grades are 87.500000 and 93.400000
```

כל מציין פורמט מתחיל בתו האחוזים (%) ואחריו אות המסמנת את טיפוס הפרמטר:

- מציין הטיפוס הראשון, %d , מציין טיפוס שלם ומתייחס לפרמטר 2
- מציין הטיפוס השני, %f, מציין טיפוס ממשי ומתייחס לפרמטר 87.5
- מציין הטיפוס השלישי, %f, מציין טיפוס ממשי ומתייחס לפרמטר 93.4

הטבלה שבעמ' 95 מפרטת את מצייני הטיפוס עפ"י קטגוריות הטיפוסים.

דוגמאות פשוטות:

| **פלט** | **הוראה** |
|---|---|
| A | *printf("%c",65);* |
| I am 12 years old | *printf("I am %d years old",12);* |
| An apple costs 14.440000 per Kilo | *printf("An apple costs %f per Kilo",14.44);* |
| 17 is 11 in hexa and 21 in octal | *printf("%d is %x in hexa and %o in octal", 17,17,17);* |
| a b c | *printf("a\tb\tc");* |

דוגמאות מורכבות יותר:

| **הוראה** | **פלט** |
|---|---|
| *printf("%s world", "hello");* | `h e l l o   w o r l d` |
| *printf( "Real = %f %e %E", 23.452, 23.452, 23.452);* | `Real = 23.452000`<br>`2.345200e+001`<br>`2.345200E+001` |
| *int num;*<br>*printf( "Address=%p", &num);* | `A d d r e s s = 0 0 6 F D D 8` |
| *printf("1234567890%n", &chars_no);*<br>*printf("\nChars writen=%d",chars_no);* | `1 2 3 4 5 6 7 8 9 0`<br>`C h a r s   w r i t e n = 1 0` |


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## קביעת תצורת הפלט

### ריווח

בפונקציה printf ניתן לשלוט בריווח השורה ובמספר התווים המוצגים לכל שדה. לדוגמא, הוראת ההדפסה

*printf("%f %d %s\n", 45.583f, 5, "hello");*

תציג למסך:

```
45.583000 5 hello
```

ניתן לקבוע את רוחב השדה בהדפסת פרמטר מסוים ע"י הצבת מספר בין הסימן % למציין הטיפוס. לדוגמא, את הנתונים מההדפסה הקודמת נדפיס בשדה ברוחב 8 תווים:

*printf("%**8**f%**8**d%**8**s\n", 45.583f, 5, "hello");*

יודפס:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 5 | . | 5 | 8 | 3 | 0 | 0 | 0 | | | | | | | | 5 | | | | h | e | l | l | o |

כפי שניתן לראות, רוחב השדה הכולל הוא **מינימום** - אם הנתון כולל מספר תווים גדול מרוחב השדה הוא יודפס במלואו, לכן הפרמטר הראשון מודפס על פני 9 מקומות ולא 8.

הפרמטר השני, 5, מודפס על פני 8 מקומות ומוצמד לחלק הימני שלהם. גם הפרמטר השלישי, "hello" , מודפס על פני 8 מקומות ומוצמד ימינה.

### דיוק

ניתן גם לקבוע את הדיוק שמשמעותו היא בהתאם לטיפוס:

עבור ממשיים - מספר התווים המודפסים לאחר הנקודה העשרונית.

עבור שלמים - מספר הספרות המינימלי להדפסה.

עבור מחרוזת - מספר התווים להדפסה.

הדיוק מצויין מימין לנקודה במחרוזת הבקרה:

*printf("%8**.2**f%8**.4**d%8**.3**s\n", 45.583f, 5, "hello");*

יודפס:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 4 | 5 | . | 5 | 8 | | | | | 0 | 0 | 0 | 5 | | | | | h | e | l | l |

### הצמדה

בברירת מחדל, המספרים המודפסים מוצמדים לימין בפלט. כדי לבצע הצמדה לשמאל של

המספרים, מציבים סימן "-" לפני ערך רוחב השדה. לדוגמא:

*printf("%-8.2f%-8d%-8s\n", 45.583f, 5, "hello");*

יודפס:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 5 | . | 5 | 8 | | | | 5 | | | | | | | | h | e | l | l | o | | | |


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## מצייני פורמט נוספים

מצייני הפורמט הבאים מוצבים בין הסימן % למציין הטיפוס, או לרוחב השדה - אם קיים.

קביעת short (h) ו- long (l) לשלמים:

התו h המופיע בין הסימן % למציין טיפוס שלם מציין שהפרמטר הוא מסוג short, והתו l מציין שהפרמטר מסוג long:

```
short s1 = 1234;
long l1 = 123456789L;
printf("%ld  %hd", l1, s1);
```

קביעת long (L) לממשיים:

התו L המופיע בין הסימן % למציין טיפוס ממשי מציין שהפרמטר הוא מסוג long double:

```
long double ld = 34E+33;
printf("%Lg", ld);
```

הצגת אפסים מקדימים (0):

התו 0 (אפס) בין הסימן % לרוחב השדה מציין מילוי באפסים:

```
printf("%08.2f %d\n", 45.58f, 5);
```

יודפס:

```
00045.58 5
```

כפי שראינו קודם, עבור שלמים מציין ה**דיוק** גם הוא משמש למילוי אפסים לצורך הדפסת מספר ספרות מינימלי.

קביעת הרווח ע"י פרמטרים (*)

התו * המופיע בין הסימן % למציין הטיפוס מציין שרוחב השדה נמצא ברשימת הפרמטרים במקום המתאים. לדוגמא:

```
printf("%*.*f %d\n", 8, 2, 45.583f, 5);
```

יודפס:

```
   45.58 5
```

קביעת תצורה ע"י #

התו # המופיע בין הסימן % למציין הטיפוס מציין תצורה שונה להדפסות הבאות:

%x, %X - השלם יודפס עם הקידומת 0X או 0x בהתאמה

ממשיים - הממשי יודפס עם הנקודה העשרונית ללא תלות בדיוק הנדרש

לדוגמא, פלט ההוראה הבאה

*printf("%g %x\n", 45.0f, 0xF5);*

הוא

```
45 f5
```

כעת, אם נוסיף את התו # במחרוזת הבקרה:

*printf("%#g %#x\n", 45.0f, 0xF5);*

יודפס:

```
45.0000 0xf5
```

<u>הדפסת +</u>

התו + המופיע בין הסימן % למציין הטיפוס מציין הדפסת מספר חיובי עם הסימן +. (אם המספר הוא שלילי מודפס הסימן "-" ללא תלות במציין זה).

דוגמא:

*printf("%+d %+7.2f %+d", 22, 23.44, -15);*

יודפס:

```
+22  +23.44 -15
```

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-2 שבעמ' 98-99.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# קלט לפי תבנית ע"י scanf

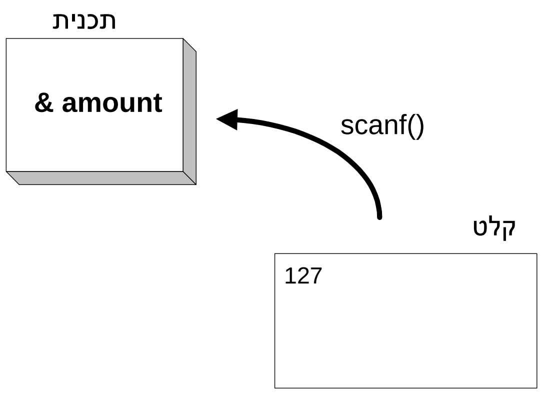
הפונקציה **scanf** מבצעת קריאת קלט עפ"י פורמט. השם scanf הוא קיצור של scan + format, כלומר סריקה אחר נתונים בקלט עפ"י פורמט נתון.

scanf היא פונקציה מקבילה ל- printf הקוראת מהקלט לתוך רשימת פרמטרים עפ"י פורמט נתון, אך עם הבדל אחד משמעותי: במקום הפרמטרים עצמם מועברות **הכתובות** שלהם.

לדוגמא, בפעולת הקלט למשתנה ממשי

*float amount;*
*scanf("%f", **&amount**);*

מועברת כתובת המשתנה, **&amount**, לפונקציה scanf:




הפונקציה scanf קוראת נתונים מהקלט לתוך <u>כתובות</u> שניתנות לה כפרמטרים.

האופרטור & מציין "הכתובת של": הצבתו לפני שם המשתנה מציינת התייחסות לכתובתו - נעסוק במצביעים בהרחבה בפרק 8, "**מצביעים**".

scanf מתעלמת מהתווים ה"לבנים" - רווח, טאב ותו שורה חדשה - אלא אם כן סוג הנתון הוא תווי. לדוגמא:

```
int num;
scanf("%d", &num);
printf("The number is %d", num);
```

אם בקלט יהיה כתוב "123abc", הפונקציה תקרא את המספר 123 ותתעלם מהאותיות abc, כיוון שאינן ספרות.

**שאלה**: מה היה קורה לו במחרוזת הבקרה היה כתוב %x במקום %d?

**תשובה**: abc ספרות חוקיות במספר הקסה-דצימלי ולכן הן היו נקראות. המספר הנקרא היה 0x123ABC.

## מצייני הטיפוס

הטבלה שבעמ' 100 כוללת את מצייני הטיפוסים עבור scanf.

scanf מנסה להתאים כל תו הנמצא במחרוזת הבקרה לקלט, כולל רווחים. לדוגמא, אם נרצה לקלוט 2 מספרים - שלם וממשי - מופרדים ע"י הסימן "^" נכתוב:

```
int     inum;
float  fnum;
printf("Enter an integer and a float, separated by ^\n");
scanf("%d ^ %f", &inum, &fnum);
printf("%d %f", inum, fnum);
```

עבור הקלט

```
Enter an integer and a float, separated by ^
23  ^  23.45
```

יודפס

```
23 23.450001
```

<u>דוגמאות</u>

נניח שמוגדרים המשתנים

```
int        inum;
float      fnum;
char       ch;
```

וכמו כן נניח שמוגדר טיפוס מחרוזת ומשתנה מחרוזת:

```
typedef    char   String[256];
String     str;
```

הטבלה הבאה כוללת דוגמאות להוראות קלט ומבנה קלט מתאים:

| הערות | קלט מתאים | הוראה |
|---|---|---|
| לאחר המספר הממשי מותאם רווח בקלט ואחריו התו r | 23 23.45 r hello | *scanf("%d %f %c %s", &inum, &fnum, &ch, str);* |
| inum מקבל את הערך 5 | hello | *scanf("%s%n", str, &inum);* |
| מספר הקסה-דצימלי בן 8 ספרות מקסימום | 34EF25A1 | *scanf("%p", &inum);* |

## בדיקת הצלחת פעולת הקלט

כאשר מבצעים פעולת קלט, יש לוודא שהקלט נקרא באופן תקין. לדוגמא, אם המשתמש מתבקש להקליד מספר ובמקום זאת הוא מקליד אות, הקלט אינו תקין.

**שאלה**: כיצד נדע אם הקלט הצליח?

**תשובה**: הפונקציה scanf מחזירה את מספר הפרמטרים שנקראו באופן תקין מהקלט. ניתן להשתמש בערך זה כדי להחליט אם הקלט היה תקין. לדוגמא:

```
#include <stdio.h>
void  main()
{
        int     num1;
        float  num2;
        int     n;
        printf("Please enter 2 numbers - integer and real: ");
        n = scanf("%d %f", &num1, &num2);
        if(n!=2)
                printf("Incorrect input!");
        else
                printf("The numbers are: %d %.2f", num1, num2);
}
```

בתכנית מבקשים מהמשתמש להקליד שני מספרים, שלם וממשי. המשתנה n משמש לקריאת הערך המוחזר מהפונקציה scanf

```
n = scanf("%d %f", &num1, &num2);
```

במידה וערך זה שונה מהערך הצפוי - 2 - מדפיסים הודעת שגיאה למשתמש. דוגמא להרצת התכנית עם קלט שגוי:

```
Please enter 2 numbers - integer and real:
12 word
Incorrect input!
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

# קלט / פלט של מחרוזות

כפי שראינו, מחרוזת היא מערך תווים המסתיים בתו מסיים מחרוזת. הדפסת מחרוזת אפשרית ע"י הפונקציה printf בתוספת מזהה הטיפוס %s:

```
typedef char String[256];

String  str = "hello";
printf("%s",str);
```

או בקיצור:

```
printf(str);
```

קריאת מחרוזות מתבצעת ע"י מזהה הטיפוס %s , לדוגמא:

```
scanf("%s",str);
```

מחרוזת היא מקרה מיוחד - אין צורך באופרטור הכתובת "&" מכיון ששמה הוא כתובתה. בפרק 8, "**מצביעים**", נרחיב בנושא כתובות המשתנים והפרמטרים לפונקציות.

## קלט / פלט שורת טקסט ע"י gets ו- puts

קיימות פונקציות קלט / פלט ייעודיות לקריאה וכתיבה של מחרוזת הפרושה על שורה שלמה:

gets(str) - פונקציה לקריאת שורת טקסט מהקלט לתוך המחרוזת הנתונה.

puts(str) - פונקציה להדפסת המחרוזת הנתונה לפלט כשורת טקסט.

שתי הפונקציות מוצהרות בקובץ הספרייה stdio.h ומקבלות כפרמטר מחרוזת:

- gets קוראת מהקלט את התווים לתוך מחרוזת הנתונה לה כפרמטר ומחליפה את תו מעבר השורה '\n' בתו '\0' המסמן סוף מחרוזת.
- puts מדפיסה לפלט את התווים מתוך המחרוזת הנתונה תוך החלפת תו סיום המחרוזת '\0' בתו שורה חדשה '\n'.

***עיין/י בתכנית הדוגמא שבעמ' 103.***

# ניתוב קלט / פלט
# (I/O Redirection)

**הפעלת התכנית ממערכת ההפעלה**

לאחר הידור התכנית נוצר קובץ ביצוע הניתן להרצה ממערכת ההפעלה. אופן ההפעלה תלוי במערכת ההפעלה בה עובדים.

לדוגמא, ב- Windows ניתן להפעיל את התכנית מתוך מנהל הקבצים ע"י הקשה כפולה על קובץ הביצוע של התכנית (exe) .

כמו כן ניתן להפעילה מתוך חלון DOS ע"י הקלדת שמה והקשה על Enter. אם שם התכנית הוא prog.exe והיא נמצאת בספרייה c:\c_course נבצע:

c:\c_course> **prog**

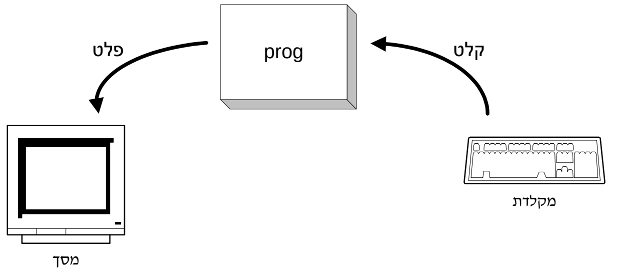
קלט התכנית יינתן ע"י המקלדת ופלט התכנית יוצג למסך.

## קלט / פלט תקני (Standard I/O)

כאשר התכנית מבצעת פעולות קלט / פלט ע"י printf או scanf למשל, היא אינה "יודעת" מהיכן מגיע הקלט ולאן מופנה הפלט.

**קלט/פלט תקני** (Standard I/O) הוא התייחסות כללית למקור הקלט וליעד הפלט של התכנית. אלה תלויים באופן ההפעלה של התכנית ע"י המשתמש.

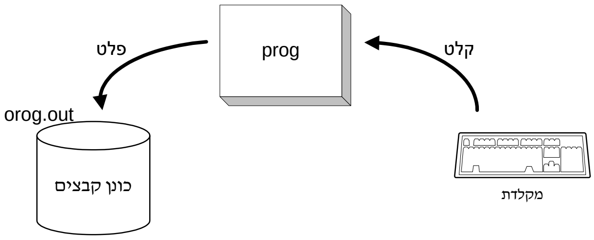
בברירת מחדל, מקור קלט התכנית הוא המקלדת ויעד הפלט הוא המסך:




<u>ניתוב הקלט והפלט</u>

המשתמש יכול לקבוע את מקור הקלט ואת יעד הפלט בהפעלת התכנית. לדוגמא, ניתן להפנות את הפלט לקובץ במקום למסך:

c:\c_course>  **prog > prog.out**



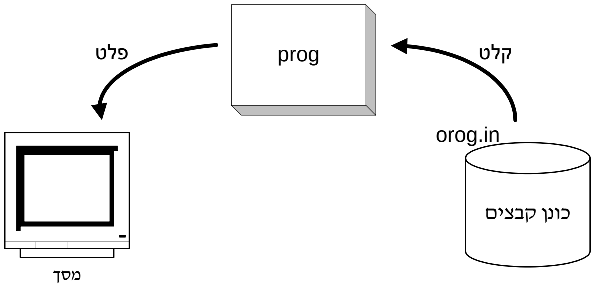

התכנית prog.exe תופעל והפלט שלה המבוצע ע"י ההוראות putchar, printf, puts ינותב לקובץ prog.out. למסך לא יודפס דבר.

כמו כן ניתן לקרוא מקובץ את קלט התכנית במקום מהמקלדת.

לדוגמא, אם נרצה לקרוא קלט לתכנית prog.exe מהקובץ prog.in נבצע:

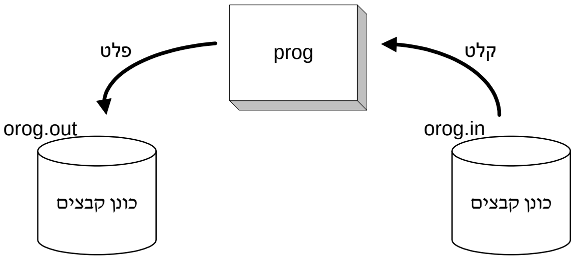
c:\c_course>  **prog < prog.in**

כעת התכנית לא תמתין לקלט מהמקלדת ע"י המשתמש, אלא תקרא את הנתונים מהקובץ באופן רציף ותמשיך בביצוע:




ניתן לשלב את השניים - כלומר לקרוא מקובץ ולכתוב לקובץ:

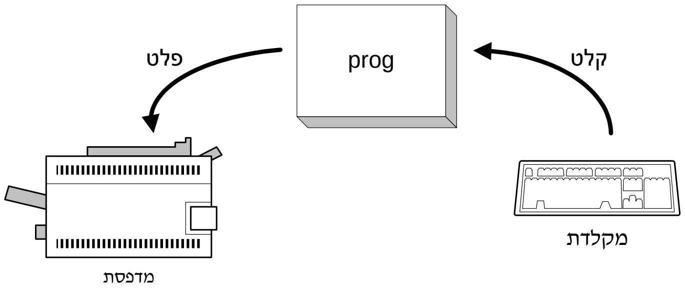
c:\c_course> **prog < prog.in > prog.out**




במערכות הפעלה רבות קבצים מתארים התקני מערכת, ולכן ניתן להשתמש בהפניית קלט/פלט לקבצים כדי לפשט פעולות מורכבות.

לדוגמא, תחת מערכות ההפעלה DOS / Windows הקובץ prn מתאר את המדפסת - לכן ניתן להדפיס את פלט התכנית prog.exe ע"י:

c:\c_course> **prog > prn**




 מרכז ההדרכה 2000 © כל הזכויות שמורות

# סיכום

- ספריית הקלט / פלט התקני כוללת פונקציות שונות לביצוע קריאה וכתיבה להתקני הקלט והפלט התקניים (מקלדת ומסך).
- קיימות פונקציות לתמיכה בקריאה וכתיבה של תווים בודדים (getchar(), putchar()) וכן פונקציות לקריאה וכתיבת מגוון עשיר של טיפוסים: שלם, ממשי, תו ומחרוזת (printf(), scanf()).
- scanf ו- printf כוללות מחרוזת בקרה המכילה מצייני פורמט עבור הפרמטרים המודפסים. ניתן גם לקבוע הצמדות ורווחים בהדפסה ע"י שדות מיוחדים הנלווים למצייני הפורמט. בפונקציה scanf יש להעביר את כתובות המשתנים כפרמטרים.
- קלט / פלט של מחרוזת בת מילה אחת מבוצע ע"י הפונקציות printf ו- scanf עם מציין הטיפוס s%. אם המחרוזת מכילה יותר ממילה אחת, ניתן לבצע קלט / פלט של כל השורה ע"י הפונקציות puts ו- gets.
- ניתן לבצע ניתוב לקלט / פלט התקני כך שמקור הקלט ו/או יעד הפלט יהיו שונים מברירת המחדל (מקלדת ומסך) כגון: פלט לקובץ, קלט מקובץ, פלט למדפסת וכו'. הניתוב מבוצע משורת הפקודה (Command Line) של מערכת ההפעלה.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבעמ' 106.***

# 5. אלגוריתמים ומבני בקרה

- הגדרת אלגוריתם ופירוקו למשימות
- סוגי מבני בקרה ב– C
- משפטי תנאי if-else
- לולאות
- משפטי continue ו– break
- משפט switch-case
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

# הגדרת אלגוריתם ופירוקו למשימות

אלגוריתם הוא סדרה של הוראות מדויקות לביצוע משימה נתונה. את הוראות האלגוריתם ניתן לרשום באופן טקסטואלי, בדומה להכנת מתכון לבישול, ע"י תרשים זרימה, ע"י קוד דמוי שפת תכנות (פסאודו-קוד) או בכל צורה אחרת.

לדוגמא, נדרש לחשב את המקסימום מבין 3 מספרים שלמים הנתונים מהקלט ולהדפיסו.

1) אלגוריתם טקסטואלי:

***משתנים: x, y, z - משתני קלט, מטיפוס שלם***

***max - משתנה התוצאה, מטיפוס שלם***

***קלוט את שלושת המספרים לתוך המשתנים x,y,z***

***אם x גדול מ- y:***

***אם x גדול מ- z הצב ל- max את x***

***אחרת, הצב ל- max את z***

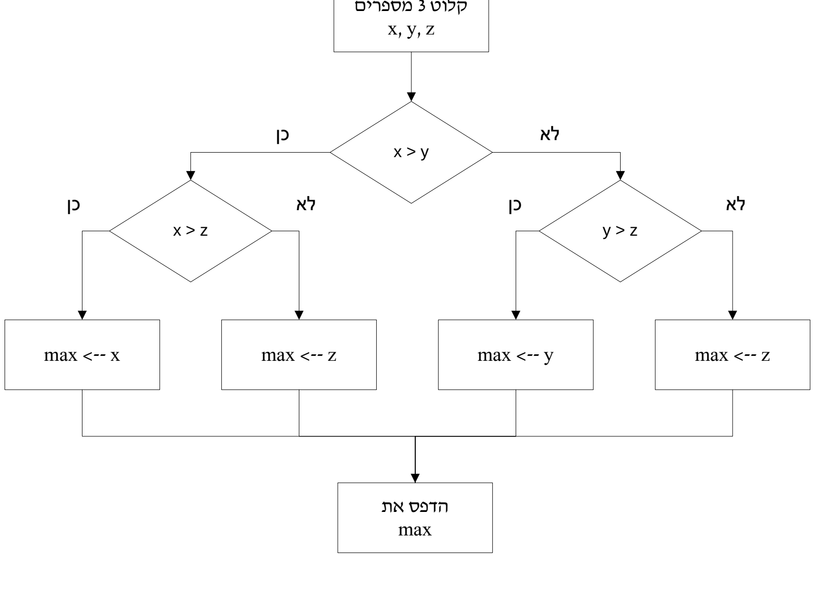
***אחרת:***

***אם y גדול מ- z הצב ל- max את y***

***אחרת, הצב ל- max את z***

***הדפס את max***

2) אלגוריתם ע"י תרשים זרימה:



3) אלגוריתם ע"י פסאודו-קוד:

```
Integer x,y, z, max

read x, y, z
if x > y
        if x > z
                max <-- x
        else
                max <-- z
else
        if y > z
                max <-- y
        else
                max <-- z
print max
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## פירוק למשימות משנה

תהליך הגדרת האלגוריתם מתחיל בהגדרת המשימה הכללית לביצוע **כמשפט בשפת אנוש** המתאר את המטרה הכללית של התכנית.

משפט זה בדרך כלל מופשט וכללי מדי להבנה ע"י המחשב. בהגדרת הוראות האלגוריתם, יש לקחת בחשבון אילו הוראות בסיסיות ניתנות לביצוע ע"י המחשב, ולאילו נדרשת הגדרה מפורטת יותר.

כל הוראה מורכבת שאינה "מובנת" דיה ע"י המחשב ניתנת לפירוק באחד משלושה אופנים:

1. פירוק לסדרת הוראות משנה
2. פירוק כמשפט תנאי
3. פירוק כלולאה

תהליך הפירוק ממשיך באופן רקורסיבי עד להגעה להוראות ברמת שפת התכנות שאיתה מממשים את האלגוריתם.

כמו כן, במהלך הפירוק צצות בעיות ונקודות הדורשות הגדרות נוספות, כגון:

– הגדרות משתנים, קבועים וטיפוסים
– אתחול המשתנים

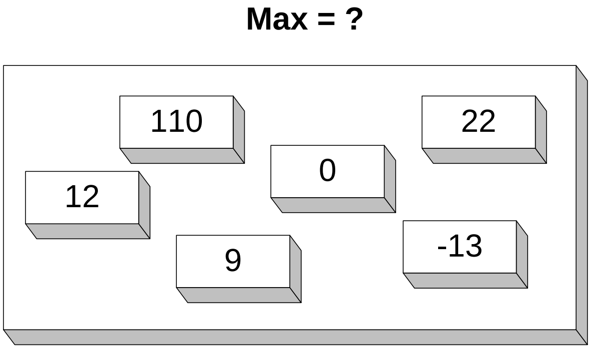
משפטי תנאי ולולאות הם מרכיבים בסיסיים בכל שפת תכנות עילית, ולעתים תכופות נכתבים באופן שמזכיר שפת אנוש.

השם הכולל למשפטי התנאי והלולאות הוא "מבני בקרה", ובהמשך נכיר את מבני הבקרה בשפת C.

## דוגמא: אלגוריתם למציאת ערך מקסימום בקלט

כדוגמא להגדרת אלגוריתם ולפירוקו נגדיר את המשימה הבאה:

***כתוב תכנית שתקרא מהקלט סדרת מספרים שלמים, ותדפיס את המספר בעל הערך המקסימלי מביניהם.***




1) נבצע פירוק של המשימה הכללית ל**לולאה**:

***משתנים: x - המספר הנוכחי שנקרא מהקלט***

**כל עוד נקרא מספר מהקלט לתוך x בהצלחה**

***אם x גדול מהמספר המקסימלי שנקרא הצב אותו למספר המקסימלי***

***הדפס את המספר המקסימלי***

כפי שניתן לראות, המשימה הכללית תורגמה למבנה לולאה הכוללת תנאי לביצוע הלולאה ושתי הוראות לביצוע בגוף הלולאה, ולאחריה הוראת הדפסה של המספר המקסימלי.

תנאי הלולאה הוא הצלחה בקריאת מספר לתוך המשתנה x: במצב זה אם המשתמש יקליד תו לא חוקי, או תו מסיים קובץ במקום מספר, הלולאה תסתיים.

2) נמשיך ונפרק את משימות המשנה:

– תנאי הלולאה ***"כל עוד נקרא מספר מהקלט לתוך x בהצלחה"*** מתורגם להצלחה בפעולת הקלט של מספר שלם לתוך x.

– הוראת הבדיקה ***"אם x גדול מהמספר המקסימלי שנקרא עד כה הצב אותו למספר המקסימלי"*** מפורקת כ**משפט תנאי**.

בנוסף, נדרשת הגדרת משתנה max שישמור את הערך המקסימלי בכל שלב. max ייבדק לעומת המספר הנקרא, x, ואם x גדול יותר - יקבל את ערכו.

– ההוראה ***"הדפס את המספר המקסימלי"*** היא הוראת פלט בסיסית ולכן אין צורך להמשיך ולפרקה.

האלגוריתם:

***משתנים: max - שלם המייצג את המספר המקסימלי שנקרא עד כה,***

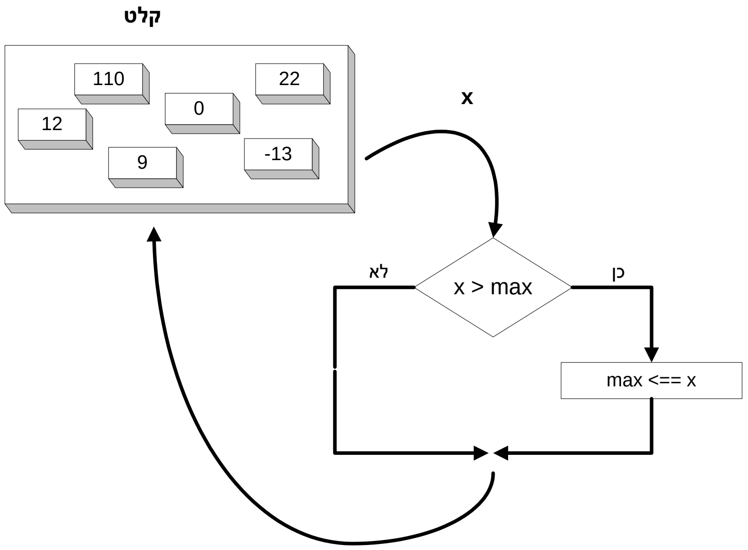
***x - המספר הנוכחי שנקרא מהקלט***

***כל עוד נקרא מספר מהקלט לתוך x בהצלחה***

***אם x גדול מ- max***

***הצב ל- max את x***

***הדפס את max***




מהתבוננות שנייה באלגוריתם, ניתן לראות שישנה בעיה: max אינו מאותחל. זה עלול לגרום להדפסת תוצאה שגויה, ולכן התכנית אינה תקינה. לאיזה ערך נאתחל את max ? קיימות שתי אפשרויות:

1. למספר השלם המינימלי האפשרי
2. למספר הראשון הנקרא מהקלט

לשם הפשטות נבחר בשיטה השנייה.

***3) האלגוריתם בתוספת הוראת האתחול ל- max מובא בעמ' 112.***

**בעיה**: כיצד תגיב התכנית במקרה של קלט ריק, כלומר 0 מספרים בקלט? כרגע תגובת התכנית לא מוגדרת, מה שאומר שיודפס ערך לא הגיוני. יש לתקן אותה כך שתודפס הודעה מתאימה במקרה זה.

**פתרון**: נוסיף בדיקה באתחול המשתנה max מהקלט: רק אם פעולת הקלט הצליחה, נבצע את הלולאה ונדפיס לאחריה את ערכו של max. אחרת, נדפיס הודעה למשתמש על קלט ריק.

***4) האלגוריתם בצירוף טיפול בקלט ריק מובא בעמ' 112.***

# סוגי מבני בקרה ב- C

מבני הבקרה כוללים את הוראות התנאי (if-else), לולאות ומשפטי בקרה נוספים. הכרנו עד עתה את הוראת if-else, ואת לולאת while.

בפרק זה נכיר את שאר מבני הבקרה ב- C. הרשימה הכוללת של מבני הבקרה ב- C:

- משפטי תנאי if-else
- לולאת while
- לולאת do-while
- לולאת for
- משפטי continue ו- break
- משפט switch-case

מבנה בקרה יכול לכלול הוראה אחת או יותר. במידה ויש מספר הוראות יש להקיף אותן ע"י סוגריים מסולסלות { }.

# *משפטי תנאי if-else*

משפטי התנאי מורכבים מההוראות **if** ו- **if-else**:

תחביר הוראת if:

if( *תנאי* )

*הוראה/ות*

תחביר הוראת if-else:

if( *תנאי* )

*הוראה/ות*

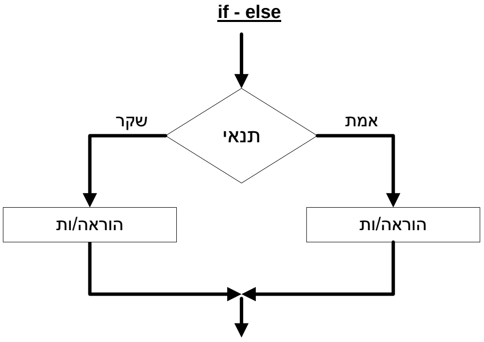
else

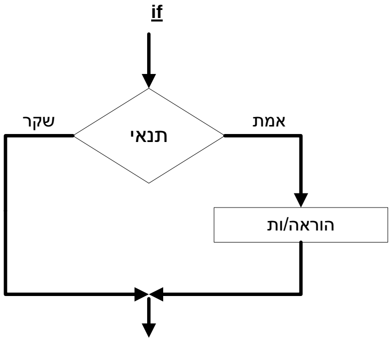
*הוראה/ות*

בתוך גוף המשפט יכולה להופיע הוראה בודדת או מספר הוראות, מוקפות ע"י הסוגריים {}.

- במשפט if אם התנאי מתקיים, מבוצעת/ות ההוראה/ות שבגוף משפט התנאי, אחרת ממשיכים להוראה העוקבת למשפט התנאי.
- במשפט if - else אם התנאי מתקיים, מבוצעת/ות ההוראה/ות שבגוף הוראת if, אחרת מבוצעת/ות ההוראה/ות שבגוף הוראת else. לאחר מכן ממשיכים להוראה העוקבת למשפט התנאי.

ניתן לתאר את משפטי התנאי if ו- if-else ע"י תרשים זרימה באופן הבא:




***דוגמאות מובאות בעמ' 114-115.***

# לולאות

**לולאה** היא אמצעי לביצוע חוזר של פעולות מסוימות מספר פעמים, או כתלות בתנאי כלשהו.

הלולאה מאפשרת חסכון בכתיבת קוד הכולל חזרות - במקום לכתוב קוד זה מספר פעמים כמספר החזרות, הוא נכתב פעם אחת בגוף הלולאה:

כל ביצוע של גוף הלולאה נקרא **חזרה** (איטרציה).

הלולאות ב- C כוללות אפשרויות בקרה מגוונות כגון: הפסקת הלולאה, הפסקת החזרה הנוכחית ומעבר לחזרה הבאה.

סוגי הלולאות ב- C:

- **while**
- **do-while**
- **for**

## לולאות while ו- do-while

2 סוגי לולאות אלו מבוצעות על פי קיומו של תנאי הלולאה. מבנה תחבירי:

- לולאת **while**:

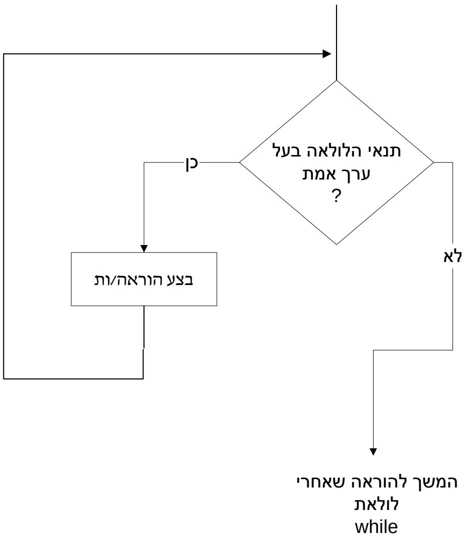
**while** *(תנאי)*

*{*

*הוראות*

*}*

כל עוד התנאי מתקיים, ההוראות שבגוף הלולאה מתבצעות. תרשים זרימה של לולאת while:




- לולאת **do-while**:

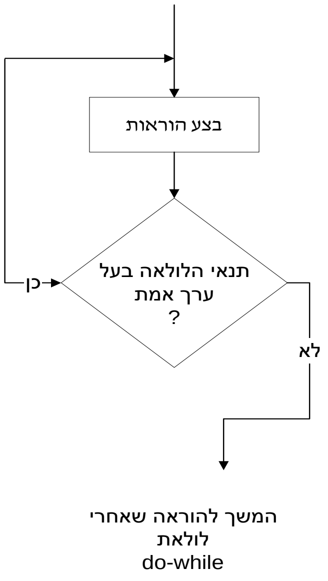
**do**

{

*הוראות*

} **while (*תנאי*);**

ההוראות שבגוף הלולאה מתבצעות כל עוד התנאי מתקיים. תרשים זרימה של הלולאה:




מה ההבדל בין שני סוגי הלולאה? בלולאת **while** התנאי נבדק לפני ביצוע הלולאה, ולכן אם הוא אינו מתקיים הלולאה לא מבוצעת.

בלולאת **do-while** גוף הלולאה ראשית מתבצע ואח"כ נבדק התנאי, ולכן מובטח לפחות ביצוע של חזרה אחת.

תכנית דוגמא: חישוב העצרת של מספר נתון n. העצרת של מספר היא תוצאת מכפלת המספרים 1,2,3..n. לדוגמא, העצרת של 5 היא 120 = 1*2*3*4*5.

את משתנה התוצאה נגדיר כ- long double בכדי שיוכל להכיל ערכים גדולים:

```
/* file: azeret1.c */
#include <stdio.h>
void main()
{
	long double azeret;
	int i,n;

	/* get input */
	printf("Enter a positive number:");
	scanf("%d",&n);

	/* initialize variables */
	azeret=1.0;
	i=1;

	/* loop over numbers 1..n */
	while (i<=n)
	{
		azeret=azeret*i;
		i++;
	}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

```
        /* output result */
        printf("Azeret of %d is %Lg",n,azeret);
}
```

דוגמא להרצת התכנית:

```
Enter a positive number:12
Azeret of 12 is 4.79002e+008
```

בדומה, ניתן לבצע את הלולאה בתכנית כ- **do-while**:

```
do
{
        azeret=azeret*i;
        i++;
} while (i<=n);
```

מהו ההבדל בין שתי הגרסאות? נניח שבקלט הוכנס המספר 0. העצרת של 0 מוגדרת כ- 1. בלולאת while התנאי ייבדק ומכיוון ש i=1 ו- n=0 תוצאתו תהיה שקר

```
while (i<=n)
{
        azeret=azeret*i;
        i++;
}
```

ולכן אף חזרה לא תתבצע. ערכו של azeret הוא 1 וזו התוצאה המודפסת. לעומת זאת בגרסת do-while החזרה הראשונה מבוצעת

```
do
{
        azeret=azeret*i;
        i++;
} while (i<=n);
```

ולאחריה נבדק התנאי, הנמצא כשקר. עדיין, מכיוון שערכו של i באתחול הוא 1, azeret=1 והתכנית מדפיסה תוצאה נכונה.

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-2 בעמ' 119.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## לולאת for

לולאת for היא לולאה כללית נוחה מאוד לביצוע של הוראות מספר פעמים או כתלות בתנאי כלשהו. המבנה שלה מסתמך על הרעיון שכל לולאה מכילה בדרך כלל שלושה חלקים: 1) אתחול 2)בדיקת תנאי הלולאה 3) קידום צעד.

תחביר לולאת for:

```
for ( <ביטוי 1> ; <ביטוי 2> ;  <ביטוי 3>)
{
        הוראות
}
```

בתוך הסוגריים שלאחר המלה for שלושה חלקים, מופרדים ע"י התו ";":

ביטוי 1 הוא **אתחול** המתבצע לפני תחילת הלולאה.

ביטוי 2 הוא **תנאי הלולאה** - ביטוי לוגי שכל עוד ערכו אמת הלולאה מתבצעת.

ביטוי 3 הוא **קידום הצעד** בלולאה.

ניתן לתאר את הוראת for ע"י אלגוריתם טקסטואלי:

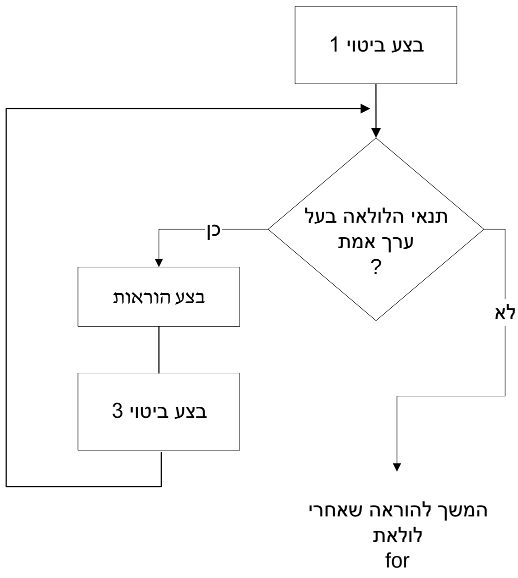
***בצע את ביטוי 1***

***כל עוד ערכו של ביטוי 2 אמת***

***בצע את גוף הלולאה***

***בצע את ביטוי 3***

כמו כן ניתן לתאר את לולאת for ע"י תרשים זרימה:



נממש את התכנית הנ"ל לחישוב העצרת באמצעות לולאת for:

```
/* file: azeret2.c */
#include <stdio.h>
void main()
{
	long double azeret;
	int i,n;

	/* get input */
	printf("Enter a number:");
	scanf("%d",&n);

	/* compute azeret using for loop */
	for(azeret=1.0, i=1; i<=n; i++)
		azeret=azeret*i;

	/* output result */
	printf("Azeret of %d is %Lg",n,azeret);
}
```

קיבלנו תכנית קצרה יותר ומובנית יותר. בלולאת for שבתכנית ביצענו 2 הוראות בביטוי 1 (אתחול הלולאה) ע"י שימוש באופרטור הפסיק "," להפרדה ביניהן:

```
for(azeret=1.0, i=1; i<=n; i++)
```

ניתן כמו כן להשתמש בטכניקה זו גם בביטוי 3, קידום הצעד.

<u>**מסקנה**</u>: הלולאה הנוחה ביותר לכתיבה היא for. לולאת for מכילה בכותרתה את שלושת השלבים הבסיסיים בטיפול בלולאה: אתחול, בדיקת תנאי להמשך הביצוע וקידום/שינוי משתנה.

<u>השמטת חלקים בלולאת for</u>

בלולאת for אין חובה להגדיר את כל חלקיה: ניתן להשמיט כל אחד משלושת חלקיה או את כולם ביחד. עם זאת, חובה עדיין להפריד בין החלקים ע"י התו ";".

- השמטת ביטוי האתחול. במקרה זה האתחול פשוט אינו מבוצע. לדוגמא:

```
scanf("%d", &num);
for(; num< 5; num++)
	...
```

- השמטת תנאי הלולאה. במקרה זה הלולאה היא אינסופית, לדוגמא:

```
for(i=0; ; i++)  /* forever */
{
	...
}
```

- השמטת קידום הצעד. לדוגמא:

```
for(i=0; i< 1000; )
{
        scanf("%d", &num);
        i = i + num;
        printf("current iteration = %d", i);
}
```

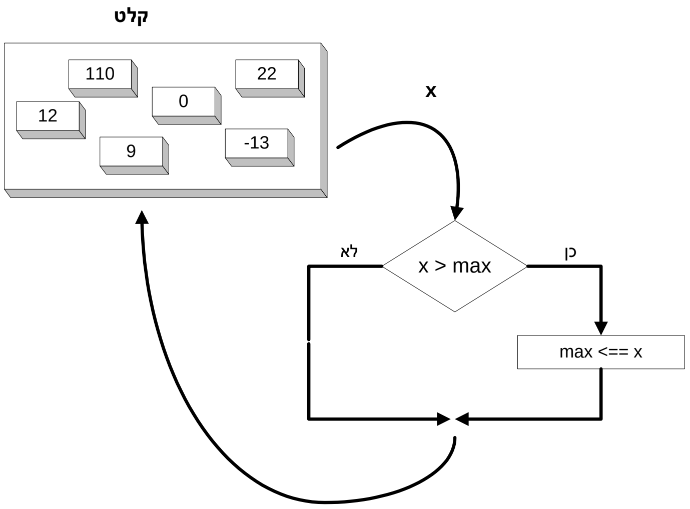
- השמטת כל שלושת החלקים. הלולאה היא אינסופית:

```
for(;;)                /* forever */
{
        ...
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## מימוש האלגוריתם לחישוב המספר המקסימלי

כעת, לאחר שהכרנו את מבני הבקרה הבסיסיים ב- C, נממש את האלגוריתם שהגדרנו בתחילת פרק זה למציאת המספר המקסימלי בקלט:




***משתנים: max - שלם המייצג את המספר המקסימלי שנקרא עד כה,***

***x - המספר הנוכחי שנקרא מהקלט***

***קרא מספר מהקלט לתוך max***

***אם הפעולה הצליחה***

***כל עוד נקרא מספר מהקלט לתוך x בהצלחה***

***אם x גדול מ- max הצב ל- max את x***

***הדפס את max***

***אחרת***

***הדפס "לא נמצאו מספרים בקלט"***

***מימוש האלגוריתם ע"י לולאת while מובא בעמ' 122-123.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## לולאות כפולות

ניתן לבצע לולאה בתוך לולאה אחרת. דוגמא ללולאה כפולה - הדפסת לוח הכפל של המספרים 1 עד 5 :

```
/* file: luach5x5.c */
#include <stdio.h>
void main()
{

	int i, j;
	for(i=1; i <= 5; i++)
	{
		for(j=1; j <= 5; j++)
			printf("%4d", j*i);
		putchar('\n');
	}
}
```

פלט התכנית:

```
1   2   3   4   5
2   4   6   8  10
3   6   9  12  15
4   8  12  16  20
5  10  15  20  25
```

הלולאה החיצונית עוברת על המספרים 1..5 עם האינדקס i. בכל חזרה מבוצעת לולאה פנימית העוברת על המספרים 1..5 עם האינדקס j, ומדפיסה את מכפלת שני האינדקסים.

> *הערה : ההוראה החיצונית מכילה 2 הוראות לכן יש צורך בסוגריים, בעוד שבלולאה הפנימית אין בכך צורך.*


מרכז ההדרכה 2000 © כל הזכויות שמורות

# ***משפטי continue ו- break***

כאשר נמצאים באמצע ביצוע של חזרה בלולאה מסוימת ניתן לסיימה בשני אופנים:

1. הוראת continue - הפסקת החזרה הנוכחית ומעבר לחזרה הבאה בלולאה. דוגמא:

```
for (i=1, azeret=1.0; i<=n; i++)
{
        if (i==3)
                continue;
        azeret=azeret*i;
}
```

אם n=5 אזי יתבצע: 1 * 2 * 4 * 5 = 40

2. הוראת break - הפסקה מוחלטת של הלולאה ומעבר לביצוע ההוראה העוקבת ללולאה. דוגמא:

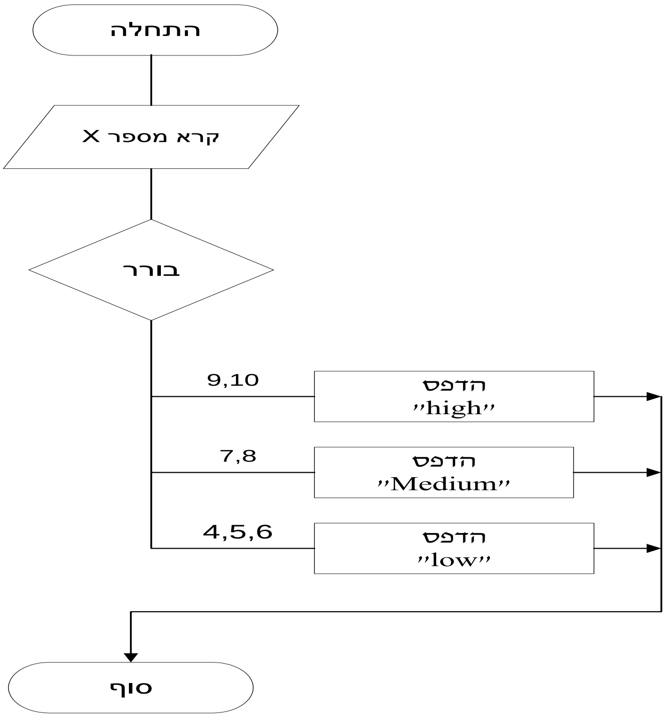
```
for (i=1, azeret=1.0; i<=n; i++)
{
        if (i==3)
                break;
        azeret=azeret*i;
}
```

כעת יתבצע: 1 * 2 = 2


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# משפט switch-case

ההוראה **switch** משמשת לברירה בין מספר אפשרויות, כלומר להסתעפות רב כוונית. לדוגמא, נתון אלגוריתם לקריאת ערך מספרי בתחום שבין 1..10 ולהדפסתו בצורה טקסטואלית:




מימוש האלגוריתם ע"י משפטי תנאי if-else הוא ארוך ומסורבל:

```
#include <stdio.h>
void main()
{
	int x;
	printf("Enter a number between 4..10:");
	scanf("%d",&x);

	if(x==10 || x==9)
		printf("High");
	else if(x==8 || x==7)
		printf("Medium");
	else if(x==6 || x==5 || x==4)
		printf("Low");
	else
		printf("Incorrect number!");
}
```

משפט **switch-case** הוא תחליף טבעי וקריא יותר למימוש הסתעפות רב כוונית. תחביר המשפט:

 מרכז ההדרכה 2000 © כל הזכויות שמורות

switch( *ביטוי* )

{

case *קבוע-שלם* : הוראות

break;

case *קבוע-שלם* : הוראות

break;

case *קבוע-שלם* : הוראות

break;

default: הוראות

}

במשפט **switch** הערך הנבדק מושווה לכל אחד מהערכים שבכניסות ה- case, החל מהראשון. הכניסה הראשונה שנמצאת נכונה גורמת לביצוע סדרת ההוראות שבאותה כניסה.

טיפוס הנתון הנבדק בהוראת **case** חייב להיות ממשפחת השלמים, כלומר שלם או תו. כאשר מזוהה כניסה נכונה בהוראת **switch** הביצוע נמשך גם לכניסות הבאות עד אשר מזוהה ההוראה **break**. תכונה זאת נקראת **falling-through**.

נממש את התכנית ע"י משפט **switch-case** :

```
/* file: switch.c */
#include <stdio.h>
void main()
{
	int x;
	printf("Enter a number between 4..10:");
	scanf("%d",&x);
	switch(x)
	{
		case 10	:
		case  9 :	printf("High");
				break;

		case 8  :
		case 7	:	printf("Medium");
				break;

		case 6	:
		case 5	:
		case 4	:	printf("Low");
				break;
		default:	printf("Incorrect number!");
	}
}
```

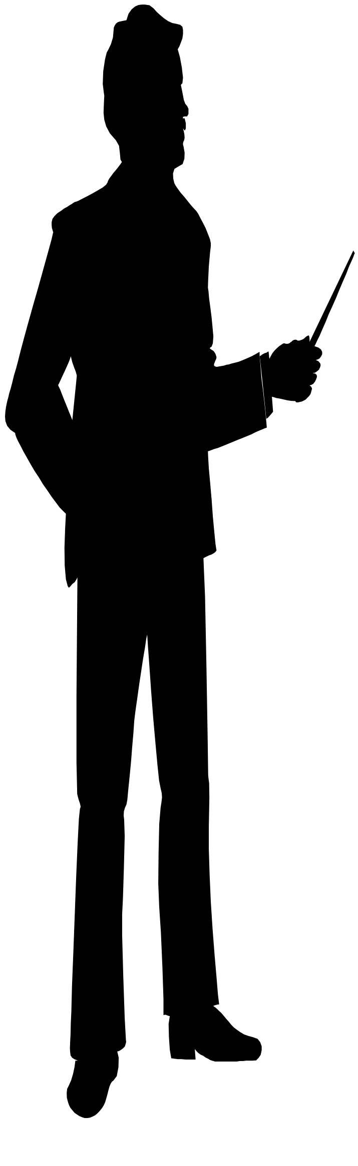
# סיכום

- מבני הבקרה בשפת C כוללים משפטי תנאי, לולאות והוראות בקרה נוספות.
- משפטי התנאי הם מהצורה if או if-else.
- הלולאות הקיימות ב- C:
  - לולאות **while** ו- **do-while**
  - לולאות **for**
- הוראות ומבני בקרה נוספים:
  - הוראת **continue** גורמת להפסקת החזרה הנוכחית בלולאה ולמעבר לחזרה הבאה.
  - הוראת **break** גורמת ליציאה מבלוק הבקרה הנוכחי, בין אם הוא תנאי, לולאה או משפט switch.
  - משפט **switch-case** הוא הסתעפות רב כוונית עפ"י מספר ערכים אפשריים של שלם.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום 1-2 שבעמ' 128-129.***

# 6. פונקציות



- פירוק אלגוריתם למשימות עצמאיות
- מנגנון הפונקציות ב- C
- הפונקציה main
- הצהרת הפרמטרים
- ערך מוחזר
- משתנים ופונקציות
- מבנה זיכרון התכנית
- רקורסיה
- סיכום
- תרגילי סיכום


מרכז ההדרכה 2000 © כל הזכויות שמורות

# פירוק אלגוריתם למשימות עצמאיות

בפרק הקודם ראינו כיצד ניתן לפרק אלגוריתם למשימות משנה ע"י שימוש במבני בקרה - משפטי תנאי ולולאות. לפעמים סוג זה של פירוק אינו מספיק.

לדוגמא, נתונה המשימה הבא:

***כתוב/י תכנית שתדפיס את כל החזקות החד ספרתיות (המספרים 0..9) של 2 ושל 3 בטבלה.***

מטרת המשימה היא להדפיס את הטבלה הבאה:

```
i     power(2,i)     power(3,i)
-     ----------     ----------
0     1              1
1     2              3
2     4              9
3     8              27
4     16             81
5     32             243
6     64             729
7     128            2187
8     256            6561
9     512            19683
```

במבט ראשון, נראה שכדאי לפרק את המשימה ע"י לולאה:

***הדפס כותרות לטבלה***

***בלולאה עם אינדקס i מ- 0 עד 9 בצע:***

***חשב את החזקה של 2 ב- i (res1)***

***חשב את החזקה של 3 ב- i (res2)***

***הדפס שורה בטבלה - i, res1 ו- res2***

כעת יש לפרק את הוראות הלולאה הדורשות את חישוב החזקה: נבצע זאת ע"י לולאה פנימית העוברת על המספרים 1..i ומבצעת הכפלה של 2 או 3 כמספר החזרות.

האלגוריתם המלא:

***משתנים: i,j - אינדקסים בלולאות, res1, res2 - משתנים לחישוב החזקות***

***אתחול: res1 <-- 1, res2 <-- 1***

***הדפס כותרות לטבלה***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

***בלולאה עם אינדקס i מ- 0 עד 9 בצע:***

***בלולאה עם אינדקס j מ- 1 עד i בצע:***

***res1 <-- res1*2***

***בלולאה עם אינדקס j מ- 1 עד i בצע:***

***res2 <-- res2*3***

***הדפס שורה בטבלה - i, res1 ו- res2***

כפי שניתן לראות, res1 ו- res2 מטופלים באופן דומה לחישוב החזקה ה- i-ית של 2 ו- 3 בהתאמה.

כל אחד מהם מוכפל j פעמים בבסיס החזקה (2 או 3). בכדי לבצע את חישוב החזקה נכון יש לאתחל את res1 ו- res2 ל- 1.

מהם חסרונות פירוק זה?

– ביצענו פעמיים פעולה זהה של מציאת חזקה של מספר אחד במספר אחר, ולצורך כך שכפלנו קוד דומה מאוד לחישוב עם בסיס ו/או מעריך שונים.

– במידה ובמקום אחר בתכנית נצטרך לבצע חישוב נוסף של חזקה, עם בסיס ומעריך זהים או שונים, נצטרך שוב לכתוב לולאה דומה.

## הגדרת משימה עצמאית ע"י פונקציה

בפרק זה נראה כיצד ניתן לפרק משימה כללית למשימות משנה עצמאיות הנקראות **פונקציות**. **פונקציות** ניתנות לביצוע בכל שלב של התכנית ופעולתן יכולה להיקבע ע"י **פרמטרים**.

בסיום החישוב הן **מחזירות** את התוצאה.

נחזור לדוגמא הקודמת: את המשימה של חישוב חזקה ניתן להגדיר כפונקציה המקבלת 2 פרמטרים - את הבסיס ואת המעריך - ומחשבת את החזקה:

*פונקציה **power** לחישוב חזקה:*

***פרמטרים: base - בסיס החזקה, n - מעריך החזקה***

***אתחול: result <-- 1***

***בלולאה עם אינדקס i מ- 1 עד n בצע:***

***result * base --> result***

***החזר את result***

כעת ניתן להשתמש בפונקציה שהגדרנו **power** ע"י ציון שמה והעברת פרמטרים מתאימים.

לדוגמא, בכדי לחשב את החזקה של 2 ב- 5 נכתוב power(2,5). אם נרצה לקבל את התוצאה במשתנה res למשל, נכתוב:

***power(2,5) --> res1***

נגדיר את אלגוריתם המשימה הכללית תוך שימוש בפונקציה שהגדרנו power:

***משתנים: i - אינדקס הלולאה, res1, res2 - משתני תוצאות החזקות***

***הדפס כותרות לטבלה***

***בלולאה עם אינדקס i מ- 0 עד 9 בצע:***

***power(2,i) --> res1***

***power(3,i) --> res2***

***הדפס שורה בטבלה - i, res1 ו- res2***

# מנגנון הפונקציות ב- C

פונקציות ב-C הן מנגנון המאפשר חלוקה של משימה מורכבת למשימות קטנות יותר ועצמאיות.

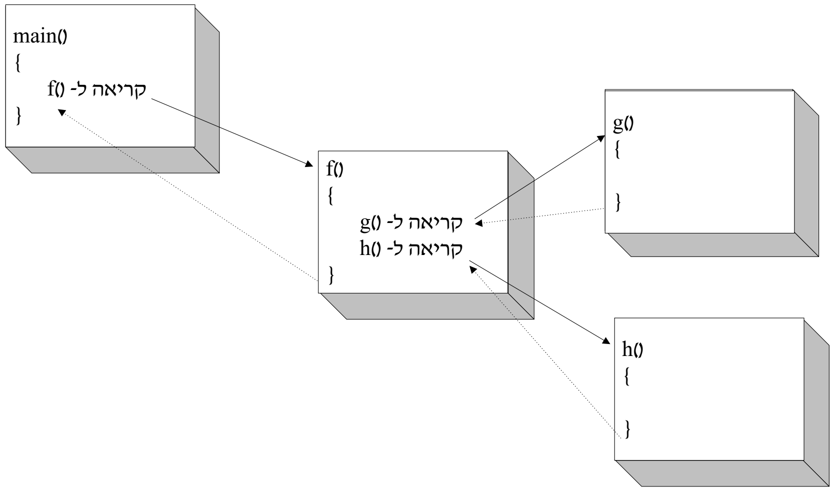
חלוקה של התכנית לפונקציות מאפשרת פיתוח מודולרי והדרגתי של התכנית, וכן יכולת טובה יותר לנפות (debug) את השגיאות בתכנית.

תכנית בשפת C היא אוסף שטוח (ללא קינון) של פונקציות. אחת הפונקציות חייבת להיות main() - זו הפונקציה שממנה מתחיל ביצוע התכנית.

כאשר מפונקציה מסוימת קוראים לפונקציה אחרת מתבצעת הפסקה זמנית בביצוע הפונקציה הקוראת ועוברים לביצוע הפונקציה הנקראת, עד אשר זו מסתיימת.

פונקציה נקראת יכולה להמשיך ולקרוא לפונקציה נוספת ובכך ליצור שרשרת קריאות.

לדוגמא, נניח שיש לנו תכנית שבה קיימות הפונקציות main(), f(), g(), h(). main מבצעת קריאה ל- f ו- f קוראת לפונקציות g ו- h :




לאחר שהפונקציה הנקרא מסתיימת, חוזר הביצוע לפונקציה הקוראת, אל ההוראה העוקבת להוראת הקריאה לפונקציה.

ניתן להעביר מידע בין הפונקציה הקוראת לנקראת: הפונקציה הקוראת מעבירה **רשימת פרמטרים** (או **ארגומנטים**) לפונקציה הנקראת לצורך העיבוד.

כאשר הפונקציה הנקראת מסתיימת, היא מחזירה את תוצאת העיבוד לפונקציה הקוראת. תוצאה זו היא **הערך המוחזר** של הפונקציה.

פונקציות לדוגמא שהכרנו (פונקציות סטנדרטיות ב- C): printf(), scanf(), main().

## הגדרת פונקציה

המבנה התחבירי של הגדרת פונקציה :

***<טיפוס-ערך-המוחזר> <שם-הפונקציה> (<הצהרת פרמטרים>)***

{

***<הגדרות טיפוסים, קבועים ומשתנים>***

***<הוראות>***

}

הגדרות פונקציות יכולות להיות בסדר כלשהו, בקובץ אחד או יותר (בפרק 11 נראה תוכניות מרובות קבצים).

כתכנית דוגמא, נממש את האלגוריתם שהגדרנו בתחילת הפרק. ראשית נגדיר פונקציה בשם power להעלאה בחזקה של מספר אחד בשני :

```
/* file: powers.c */
#include <stdio.h>
int power (int base, int n)
{
	int result=1;
	int i;

	for (i=1; i<=n; ++i)
		result = result * base;
	return result;
}
```

כעת נכתוב את הפונקציה הראשית בתכנית, **main()**, תוך שימוש בפונקציה **power()** :

```
void main()
{
	int i;
	int res1, res2;
	printf("%-4s %-13s %-13s\n", "i", "power(2,i)", "power(3,i)");
	printf("%-4s %-13s %-13s\n", "-", "----------", "----------");
	for (i=0; i<10; ++i)
	{
		res1 = power(2,i);
		res2 = power(3,i);
		printf("%-4d %-13d %-13d\n", i, res1, res2);
	}
}
```

פלט התכנית:

```
i     power(2,i)     power(3,i)
-     ----------     ----------
0     1              1
1     2              3
2     4              9
3     8              27
4     16             81
5     32             243
6     64             729
7     128            2187
8     256            6561
9     512            19683
```

<u>הסבר</u>

- בהגדרת הפונקציה **power**

```
int power (int base, int n)
{
	int result=1;
	int i;
	...
}
```

ציינו ש- :

– הפונקציה מקבלת שני פרמטרים מטיפוס שלם: base ו- n.

– הפונקציה מחזירה ערך מטיפוס שלם.

– בתוך הפונקציה ניתן להגדיר משתנים מקומיים (i, result) לצורך חישובי ביניים.

– גוף הפונקציה כולל את חישוב החזקה ע"י לולאת for:

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

– התוצאה המוצבת במשתנה result מוחזרת בסוף הפונקציה ע"י ההוראה:

```
return result ;
```

- השימוש בפונקציה **power** בתוך **main** :

– הפונקציה **power** מוגדרת ראשונה בכדי ש- **main** תוכל לקרוא לה. ככלל, פונקציה יכולה לקרוא רק לפונקציות שהוגדרו או הוכרזו לפניה (ראה/י להלן).

– הקריאה לפונקציה **power** מתבצעת פעמיים:

```
res1 = power(2,i);
res2 = power(3,i);
```

בכל קריאה מועברים 2 פרמטרים לפונקציה ומוחזר ערך התוצאה, שמוצב בפונקציה הקוראת למשתנה המתאים.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

<u>קריאה ישירה לפונקציה בהוראת ההדפסה</u>

נכתוב את הפונקציה הראשית **main** שוב באופן מעט שונה:

```
void main()
{
	int i;
	printf("%-4s %-13s %-13s\n", "i", "power(2,i)", "power(3,i)");
	printf("%-4s %-13s %-13s\n", "-", "----------",      "----------");
	for (i=0; i<10; ++i)
		printf("%-4d %-13d %-13d\n", i, power(2,i), power(3,i));
}
```

הפעם נקראת הפונקציה power בתוך הוראת ההדפסה, ותוצאתה מועברת ישירות כפרמטר לפונקציה printf לצורך הדפסה:

```
printf("%-4d %-13d %-13d\n", i, power(2,i), power(3,i));
```

בגרסה זו של הפונקציה main אין צורך בהגדרת המשתנים res1, res2.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הכרזה על אבטיפוס פונקציה

הכרזה על **אבטיפוס פונקציה** (**function prototype**) נדרשת כאשר היא נקראת ע״י פונקציה אחרת המופיעה לפניה בקובץ התכנית. נכתוב את התכנית שוב בסדר פונקציות שונה:

```
#include <stdio.h>
int power(int base, int n);  /* prototype for the function power() */

void main()
{
	int i;
	printf("%-4s %-13s %-13s\n", "i", "power(2,i)", "power(3,i)");
	printf("%-4s %-13s %-13s\n", "-", "----------", "----------");
	for (i=0; i<10; ++i)
		printf("%-4d %-13d %-13d\n", i, power(2,i), power(3,i));
}

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

	for (i=1; i<=n; ++i)
		result = result * base;
	return result;
}
```

השורה

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

היא הכרזה מוקדמת על הפונקציה power הדרושה כדי ש- main תוכל לקרוא לה, הואיל ו- power מופיעה מאוחר יותר מ- main בתכנית.

ההכרזה אומרת ש- power מקבלת שני פרמטרים מטיפוס שלם base ו- n ומחזירה ערך מטיפוס שלם. ניתן גם לכתוב את אבטיפוס הפונקציה כך:

```
int power(int, int) ;
```

כלומר לשמות base ו- n אין משמעות בהכרזה על האבטיפוס.

יש להבחין בין **הכרזה** (או **הצהרה**) **על אבטיפוס הפונקציה** לבין **הגדרת הפונקציה**:

– בהכרזה מספקים רק את הממשק של הפונקציה, בהגדרת הפונקציה צריך בנוסף לכתוב את גוף הפונקציה.

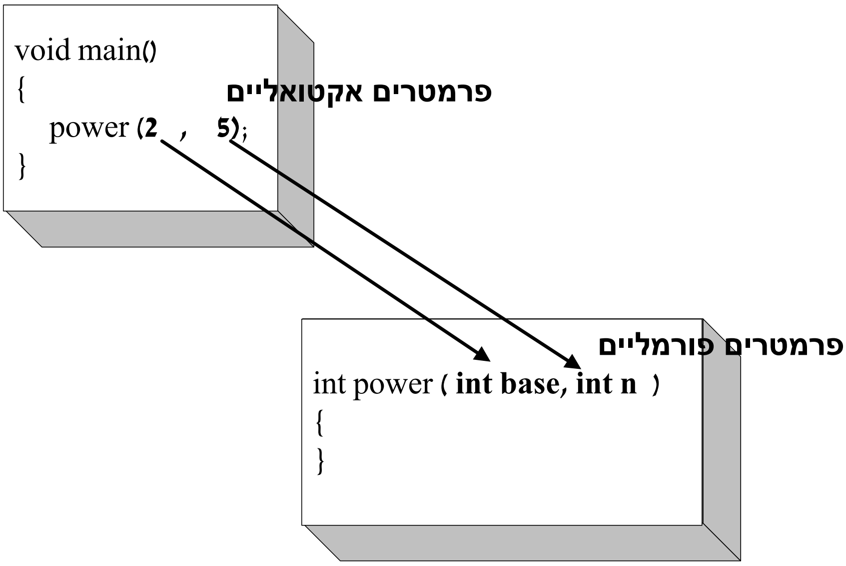
 מרכז ההדרכה 2000 © כל הזכויות שמורות

# הצהרת הפרמטרים

בקטע ההצהרה על הפרמטרים אנו קובעים אילו פרמטרים יועברו לפונקציה, כלומר מאיזה טיפוס וכמה פרמטרים. פרמטרים אלה נקראים פרמטרים פורמליים.

הפונקציה הקוראת מעבירה לפונקציה הנקראת פרמטרים הנקראים פרמטרים אקטואליים. הפרמטרים הפורמליים הם **העתק** של הפרמטרים האקטואליים.

הם מקבלים את ערכם בתחילת ביצוע הפונקציה:




המהדר בודק התאמה בין הפרמטרים האקטואליים לפרמטרים הפורמליים בקריאה לפונקציה.

אם קיימת אי-התאמה הוא ינסה לבצע המרה מרומזת (תוך הודעת אזהרה במידת הצורך) - אחרת הוא יודיע על שגיאה.

דוגמא:

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

void main()
{
	int i = 3;
	float f = 2.5;

	int res = power(i,f);
}
```

בקריאה לפונקציה power המהדר יגלה אי-התאמה בפרמטר השני בין הטיפוס הפורמלי לטיפוס האקטואלי.

מכיוון שקיימת המרה מרומזת ממשי לשלם הוא יבצע אותה - כלומר יקצוץ את השבר של f - ותוצאת הפונקציה תהיה power(3,2) = 9. כמו כן תתקבל אזהרה מהמהדר על ביצוע ההמרה.

לעומת זאת, אם ננסה לבצע

```
int res = power(i,"hello");
```

המהדר יודיע על שגיאה מכיוון שאין שיטת המרה ממחרוזת למספר.

## הצהרה על פרמטר כקבוע ע"י const

בדומה להגדרת משתנה כקבוע, ניתן להכריז על פרמטר שפונקציה מקבלת כקבוע.

במקרה זה הפונקציה לא תוכל לשנותו - ניסיון לשנותו יגרור הודעת שגיאה בהידור. מכריזים על פרמטר כקבוע ע"י המילה **const** . לדוגמא:

```
int func(const int c1, const float c2);
```

כעת לא ניתן לשנות את c1,c2 בתוך הפונקציה func.

<u>שאלה</u>: מדוע ומתי להגדיר פרמטר כ- const ?

<u>תשובה</u>: כאשר ברור עפ"י תפקידו של הפרמטר שהפונקציה לא אמורה לשנות אותו. לדוגמא, בפונקציה power לעיל, הפרמטרים של הפונקציה הם בסיס החזקה והמעריך.

ברור שבמקרה זה הפונקציה לא אמורה לשנות את הפרמטרים. הגדרתם כ- const יכולה לאתר שגיאה אפשרית של המתכנת:

```
int power (const int base, const int n);
```

באופן עקרוני, נהוג להפריד בין מספר סוגי פרמטרים לפונקציה:

- **פרמטרי קלט** - פרמטרים המהווים קלט לעיבוד בפונקציה
- **פרמטרי פלט** - פרמטרים שבהם מוצבת תוצאת העיבוד (ראה/י פרק 8, "**מצביעים**")
- **פרמטרי קלט/פלט** - פרמטרים בשימוש כפול: גם כקלט לעיבוד וגם כפלט התוצאה

בשפות מסוימות מצהירים על הפרמטר (in, out, inout) בהתאם לסוגו.

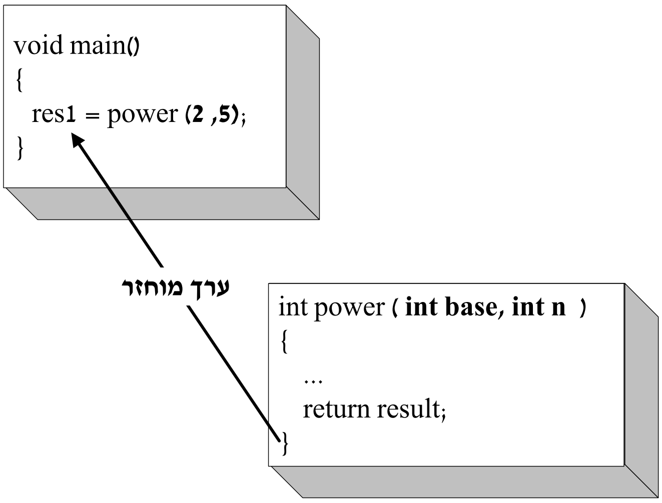
זה עוזר מאוד הן בתיעוד התוכנה והן ביכולת של המהדר להתריע בפני שגיאות תכנות.

בשפת C אופן זה של הכרזה על פרמטרים לא קיים - כחלופה בסיסית פרמטרי הקלט יכולים להיות מצוינים ע"י **const**.

# ערך מוחזר

הערך המוחזר הוא תוצאת העיבוד בפונקציה. הטיפוס שלו נקבע בכותרת הגדרת הפונקציה.

בדוגמא הקודמת טיפוס הערך המוחזר של power() הוא שלם (int), והיא מחזירה את תוצאת החזקה, המוצבת בפונקציה הקוראת למשתנה מקומי שלה:




הערך המוחזר מוחזר ע"י ההוראה **return**. הוראה זו גם גורמת לסיום הביצוע של הפונקציה וחזרה לפונקציה הקוראת.

ברירת המחדל עבור טיפוס הערך המוחזר היא שלם (int).

כלומר אם לא מציינים את טיפוס הערך המוחזר הוא מוגדר כ- int. לדוגמא, הגדרת main() ללא ציון הטיפוס מציינת שהיא מחזירה int

```
main()
{
	...
	return 0;
}
```

במידה ורוצים להגדיר פונקציה שלא מחזירה כלל ערך, מגדירים את טיפוס הערך המוחזר כ- **void**.

לדוגמא בתכנית power הפונקציה **main** היא מטיפוס **void**, כלומר לא מחזירה ערך כלשהו.

## המרת הערך המוחזר

הערך המוחזר מהפונקציה יכול לעבור המרה - מרומזת או מפורשת - כדי להתאים את טיפוסו לטיפוס שאליו מציבים את התוצאה. לדוגמא:

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

void main()
{
	float  a=9.4, b=34.8;
	int i;
	i = max(a,b);  /* convert float to integer implicitly*/
}
```

במקרה זה המהדר יציג הודעת אזהרה. ניתן לבצע המרה מפורשת ע"י

```
	i = (int) max(a,b);  /* convert float to integer explicitly*/
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

# ***הפונקציה main***

**main** היא פונקציה כמו כל פונקציה אחרת הנקראת ע"י מערכת ההפעלה, לכן גם היא יכולה להחזיר ערך. עד עתה, ציינו ש- main אינה מחזירה ערך ע"י ציון void כטיפוס הערך המוחזר.

נהוג לציין ש- main מחזירה ערך מטיפוס int המציין את תוצאת הביצוע של התכנית: הצלחה מצוינת ע"י החזרת 0, וכשלון ע"י החזרת מספר שונה מ- 0 (למשל 1). תכנית דוגמא:

```
#include <stdio.h>
int main()
{
	int	num1;
	float	num2;
	int	n;
	printf("Please enter 2 numbers - integer and real: ");
	n = scanf("%d %f", &num1, &num2);
	if(n!=2)
	{
		printf("Incorrect input!");
		return 1;
	}
	else
	{
		printf("The numbers are: %d %.2f", num1, num2);
		return 0;
	}
}
```

את הערך המוחזר של התכנית ניתן לבדוק מתוך מערכת ההפעלה. צורת הבדיקה תלויה במערכת ההפעלה עליה עובדים, והיא אינה חלק מהגדרת שפת C.

***לדוגמא, בעמ' 142 מובא קובץ batch המפעיל את התכנית prog.exe. עיין/י בקובץ זה.***

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 2-1 בעמ' 143.***

# משתנים ופונקציות

המשתנים ב- C נחלקים ל- 2 סוגים עיקריים:

- **משתנה גלובלי** - משתנה המוגדר מחוץ לכל פונקציה.
- **משתנה מקומי** - משתנה המוגדר בתוך פונקציה מסוימת.

בהקשר לסוגי המשתנים, קיימים שני מושגים חשובים:

– **תחום פעולה (scope)**: קטע התכנית שבו פונקציות "מכירות" את המשתנה, כלומר מסוגלות לקרוא את ערכו או לשנות אותו.

– **משך קיום** - משך ה"חיים" של המשתנה, כלומר קטע הזמן בריצת התכנית בו מוקצה למשתנה מקום בזיכרון.

## משתנים גלובליים

משתנה גלובלי נוצר בתחילת התכנית ומתקיים לכל אורכה. תחום הפעולה שלו הוא בכל הקובץ החל מהמקום בו הוא מוגדר (מבחינה זו הוא זהה לפונקציה!).

דוגמא:

```
#include <stdio.h>
int Xcm;

void convert();
int main()
{
	printf("Input a distance:");
	scanf("%d",&Xcm);
	convert();
	return 0;
}
void convert()
{
	float Xinch;
	Xinch= Xcm / 2.54;
	printf("%d centimeters are %6.2f inches\n",Xcm,Xinch);
}
```

**Xcm** הוא משתנה גלובלי ה"מוכר" לשתי הפונקציות, main ו- convert, ע"י כך שהוא מוגדר מחוץ להן

```
int Xcm;
```

הפונקציה main קוראת את ערכו מהקלט ע"י

```
	scanf("%d",&Xcm);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

ולאחר מכן קוראת לפונקציה ()convert. זו האחרונה ממירה את ערכו לסנטימטרים ומדפיסה את הערך החדש.

רצוי להמעיט בהגדרת משתנים גלובליים בתכנית משתי סיבות עיקריות:

– **מודולריות**: שימוש במשתנה גלובלי מקשה על שימוש חוזר בקוד.

– **ניפוי**: כאשר מגלים באג הקשור במשתנה גלובלי, קשה למצוא את הפונקציה ה"אשמה" מכיוון שמספר האפשרויות רב.

## משתנים מקומיים

משתנה מקומי נוצר בזמן הקריאה לפונקציה: למשתנה מוקצית עמדת זיכרון על **מחסנית הקריאות** (ראה/י להלן). עם גמר ביצוע הפונקציה היא משתחררת והמשתנה חדל מלהתקיים.

משתנה מקומי נקרא לפעמים גם משתנה אוטומטי מכיוון שהוא נוצר אוטומטית בזמן הקריאה לפונקציה והוא נעלם ברגע שהיא מסתיימת.

דוגמא:

```
#include <stdio.h>

void f();
int main()
{
	int i;
	for(i=0; i<4; i++)
		f();
}

void f()
{
	int x=0;
	int y=0;
	printf("x is %d, y is %d\n", x, y);
	++x;
	++y;
}
```

פלט התכנית:

```
x is 0, y is 0
x is 0, y is 0
x is 0, y is 0
x is 0, y is 0
```

כפי שרואים, x ו- y מאותחלים בכל קריאה לפונקציה f() מחדש, וקידומם בסוף הפונקציה אינו תורם דבר.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## משתנה מקומי - סטטי

ניתן להגדיר משתנה מקומי כך שלא יהיה אוטומטי, אלא ייווצר פעם אחת ויתקיים לכל אורך התכנית. משתנה כזה נקרא <u>משתנה סטטי</u>.

אופן ההגדרה:

```
static int x;
static float f;
```

ערכו של משתנה סטטי נשמר בין הקריאות לפונקציה בה הוא מוגדר. משתנה סטטי מאוחסן באזור הנתונים (data segment) של התכנית - האזור בו מוגדרים המשתנים הגלובליים - ולכן משך חייו כמשך התכנית.

למשל, אם בדוגמא הקודמת נשנה את הגדרת המשתנה x:

```
#include <stdio.h>

void f();
int main()
{
	int i;
	for(i=0; i<4; i++)
		f();
}

void f()
{
	static int x=0;
	int y=0;
	printf("x is %d, y is %d\n", x, y);
	++x;
	++y;
}
```

פלט התכנית יהיה:

```
x is 0, y is 0
x is 1, y is 0
x is 2, y is 0
x is 3, y is 0
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## משתנה מקומי בבלוק

ניתן להגדיר משתנה מקומי בתוך בלוק. בלוק מצוין ע"י סוגריים מסולסלות בתחילתו ובסופו. במקרה זה המשתנה יוכר רק בתוך הבלוק המגדיר.

דוגמא:

```
#include <stdio.h>
void main()
{
	int x=2;

	{
		int x;
		x = 3;
	}
	printf("x=%d",x);
}
```

יודפס:

```
x = 2
```

המשתנה x המוגדר בתוך הבלוק הפנימי שונה מהמשתנה x המוגדר בבלוק הפונקציה ()main ולכן ערכו של x החיצוני אינו משתנה.

משתנה מקומי בבלוק אינו שימושי בדרך כלל ורצוי להימנע מלהגדירו כך. ככלל, רצוי להגדיר את כל המשתנים המקומיים בתחילת הפונקציה.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## איתחול משתנים בברירת מחדל

למשתנה שלא אותחל קיים ערך ברירת מחדל, בהתאם לסוגו:

- **משתנה גלובלי** מאותחל בברירת מחדל ל- 0. שלמים מאותחלים ל- 0, ממשיים ל- 0.0 ותווים מקבלים את ערך ה- ASCII 0.
- **משתנה מקומי** אינו מאותחל לערך ידוע. כלומר, משתנה מקומי שהוגדר ללא איתחול הוא בעל ערך אקראי (ערך "זבל").
- **משתנה מקומי סטטי** מאותחל כמו משתנה גלובלי.

## סוגי משתנים - סיכום

| **סוג משתנה** | **תחום פעולה ( scope )** | **משך הקיום** |
|---|---|---|
| חיצוני (גלובלי) | ממקום ההגדרה ועד סוף הקובץ (ניתן להרחבה ע"י extern, כפי שנראה בפרק 11). | כמשך קיום התכנית |
| מקומי | בפונקציה המגדירה בלבד (או בבלוק המגדיר) | כמשך קיום הפונקציה (או הבלוק המגדיר) |
| מקומי - סטטי | בפונקציה המגדירה בלבד | כמשך קיום התכנית |

# מבנה זיכרון התכנית

## קטע הקוד וקטע הנתונים

כאשר התכנית עוברת הידור נוצר קובץ ביצוע (exe.) הכתוב בשפת מכונה, כלומר בשפת המחשב עליו בוצע ההידור. קובץ זה מכיל 2 קטעי זיכרון עיקריים:

- **קטע הקוד** (Code segment)
- **קטע הנתונים** (Data Segment)

**קטע הקוד** מורכב מפונקציות התכנית הכוללות את הוראות הביצוע בשפת המכונה, כפי שתורגמו משפת C. הן כוללות בדרך כלל הוראות העתקת ערך של משתנה אחד לשני, הדפסת ערך משתנה לפלט, פעולות חישוב מתמטיות, פעולות כתיבה לקבצים וכו׳.

**קטע הנתונים** הוא אזור האחסון של המשתנים הגלובליים של התכנית. מכיוון שעל משתנים אלו להתקיים במשך כל התכנית, הוגדר עבורם אזור אחסון זה באופן קשיח.

לדוגמא, נתונה התכנית

*prog.c*

```
#include <stdio.h>
int  g1=8,  g2=20;

void main()
{
        int x;
        if(g1>g2)
                x = g1;
        else
                x = g2;
        printf("x=%d",x);
}
```

קובץ הביצוע הנוצר לאחר ההידור כולל את 2 מרכיבי הזיכרון:

*prog.exe*

| | |
|---|---|
| *Data Segment* | *g1=8*<br>*g2=20* |
| *Code Segment* | *main:*<br>*int x;*<br>*if(g1>g2)*<br>*x = g1;*<br>*else*<br>*x = g2;*<br>*printf("x=%d",x);*<br>*...* |


מרכז ההדרכה 2000 © כל הזכויות שמורות

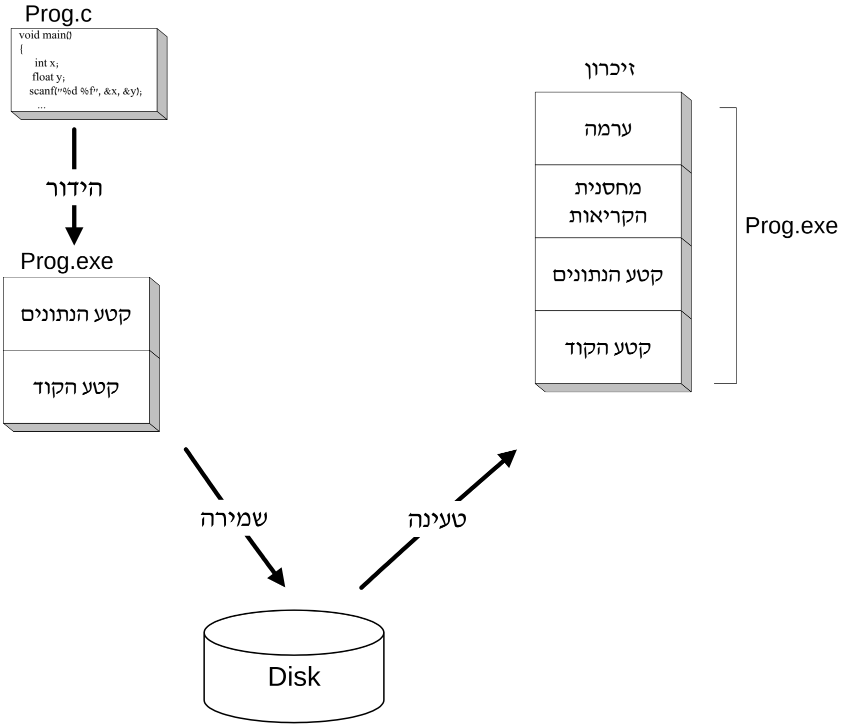
> ***הערה:*** *קטע הקוד מכיל קוד בשפת מכונה - מטעמי פשטות הקוד בתרשים הוא בשפת המקור C.*

לאחר סיום ההידור קובץ הביצוע נשמר על הדיסק. כאשר המשתמש מפעיל אותו מערכת ההפעלה טוענת אותו לזיכרון הראשי של המחשב והוא מתחיל להתבצע החל מההוראה הראשונה שבקטע הקוד.

לפני תחילת ביצוע התכנית מערכת ההפעלה דואגת להקצות עבורה 4 אזורי אחסון:

- קטע הנתונים - עבור אחסון הנתונים הנטענים מקובץ ה- exe.
- קטע קוד - עבור אחסון קוד התכנית הנטען מקובץ ה- exe.
- מחסנית הקריאות לפונקציות - עבור נתוני הפונקציות והתקשורת ביניהן (להלן).
- ערמה - עבור הקצאות זיכרון מפורשות מהתכנית. על נושא זה נלמד בפרק 12.

מהלך יצירת התכנית וטעינתה לביצוע נראה כך:




**שאלה**: היכן מאוחסנים שאר נתוני התכנית - כלומר המשתנים המקומיים שבפונקציות?

**תשובה**: משתנים אלה נוצרים על **מחסנית הקריאות**.

## מחסנית הקריאות

בזמן ריצת התכנית קיים מנגנון המתאר את מצב הקריאות בין פונקציות בתכנית, הנקרא **מחסנית הקריאות**. הוא נקרא כך מכיוון שמבנה הנתונים שבו המהדר משתמש במנגנון זה הוא מסוג **מחסנית** (על מבנה נתונים זה נלמד בפרק 14, "**טיפוסי נתונים מופשטים**").

בכל רגע בזמן ריצת התכנית, משקפת מחסנית הקריאות את מצב הקריאות לפונקציות. לכל פונקציה במחסנית נשמרת **מסגרת הפונקציה** הכוללת את:

- הפרמטרים שמועברים לפונקציה
- המשתנים המקומיים המוגדרים בפונקציה
- הערך המוחזר מהפונקציה
- כתובת החזרה של הפונקציה הקודמת (הפונקציה הקוראת)

לדוגמא, בתכנית הבאה קיימות 2 פונקציות:

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

void main()
{
        int  res;
        int a=8, b=29;
        res = max(a,b);
}
```

בתחילת ריצת התכנית, הפונקציה ()main מתבצעת. לפני הקריאה ל- ()max מחסנית הקריאות כוללת רק את מסגרת הפונקציה ()main. מצב המחסנית הוא:

| main() | res<br>a=8<br>b=29 |
|---|---|

> *הערה: מטעמי פשטות נתעלם מכתובת החזרה הנשמרת כחלק ממסגרת הפונקציה.*

 מרכז ההדרכה 2000 © כל הזכויות שמורות

לאחר הקריאה לפונקציה ()max ובמשך ביצועה מצב המחסנית הוא :

| | |
|---|---|
| *m*<br>*x=8*<br>*y=29* | max() |
| *res*<br>*a=8*<br>*b=29* | main() |

בחזרה מהפונקציה ()max, מוחזר הערך של m - שהוא המקסימום מבין x,y - על המחסנית :

| | |
|---|---|
| ***29*** | |
| *res*<br>*a=8*<br>*b=29* | *main()* |

והערך המוחזר מוצב למשתנה res שבפונקציה main

| | |
|---|---|
| ***res**=29*<br>*a=8*<br>*b=29* | *main()* |

## תמונת הזיכרון הכוללת

***בעמ' 155-151 מובאת תוכנית דוגמא תוך הדגמת מבנה הזיכרון בכל שלב. עיין/י בעמודים אלו.***

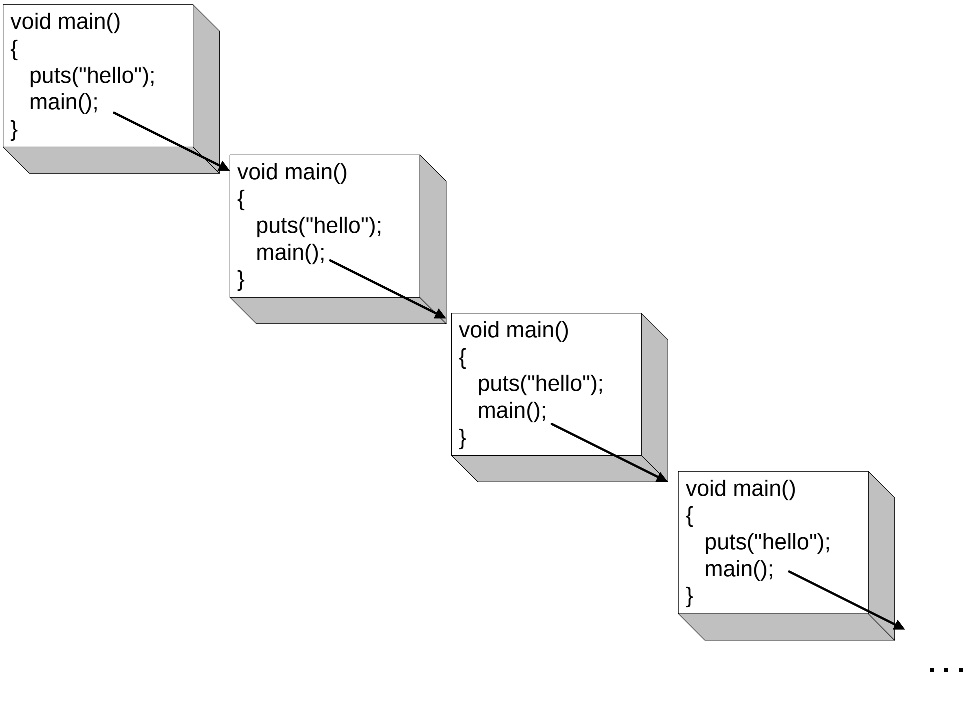
# רקורסיה

רקורסיה היא תהליך שבו פונקציה קוראת לעצמה. במילים אחרות, פונקציה הכוללת הוראת קריאה לעצמה נקראת פונקציה רקורסיבית.

דוגמא פשוטה לפונקציה רקורסיבית:

```
#include <stdio.h>
void main()
{
        puts("hello");
        main();
}
```

הסבר: הפונקציה ()main מבצעת הדפסה של המחרוזת hello ולאחר מכן קוראת לעצמה. בקריאה הבאה, ()main שוב מבצעת הדפסה ושוב קוראת לעצמה:




תהליך זה הוא אינסופי, ולכן על המסך תודפס בלולאה אינסופית המחרוזת hello:

```
hello
hello
hello
hello
hello
...
```

כפי שניתן לראות, פונקציה זו אינה שימושית במיוחד, וכמו כן היא עלולה לתקוע את המחשב. אנו נכתוב כעת דוגמא שימושית יותר לרקורסיה - פונקציה לחישוב העצרת של מספר נתון:

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

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

```
int azeret(int n)
{
	if(n<=1)
		return 1;
	else
		return n*azeret(n-1);
}

void main()
{
	printf("Azeret of 4 is : %d", azeret(4));
}
```

פלט התכנית:

```
Azeret of 4 is: 24
```

<u>הסבר</u>: הפונקציה azeret מחשבת את העצרת של מספר נתון ע"י קריאה רקורסיבית לעצמה. לפני הקריאה לעצמה היא מבצעת בדיקה של **תנאי עצירה**:

```
	if(n<=1)
		return 1;
```

במידה והפרמטר לפונקציה הוא 1 או פחות מוחזר 1. אחרת, הפונקציה מחזירה את העצרת של n-1 ע"י קריאה רקורסיבית:

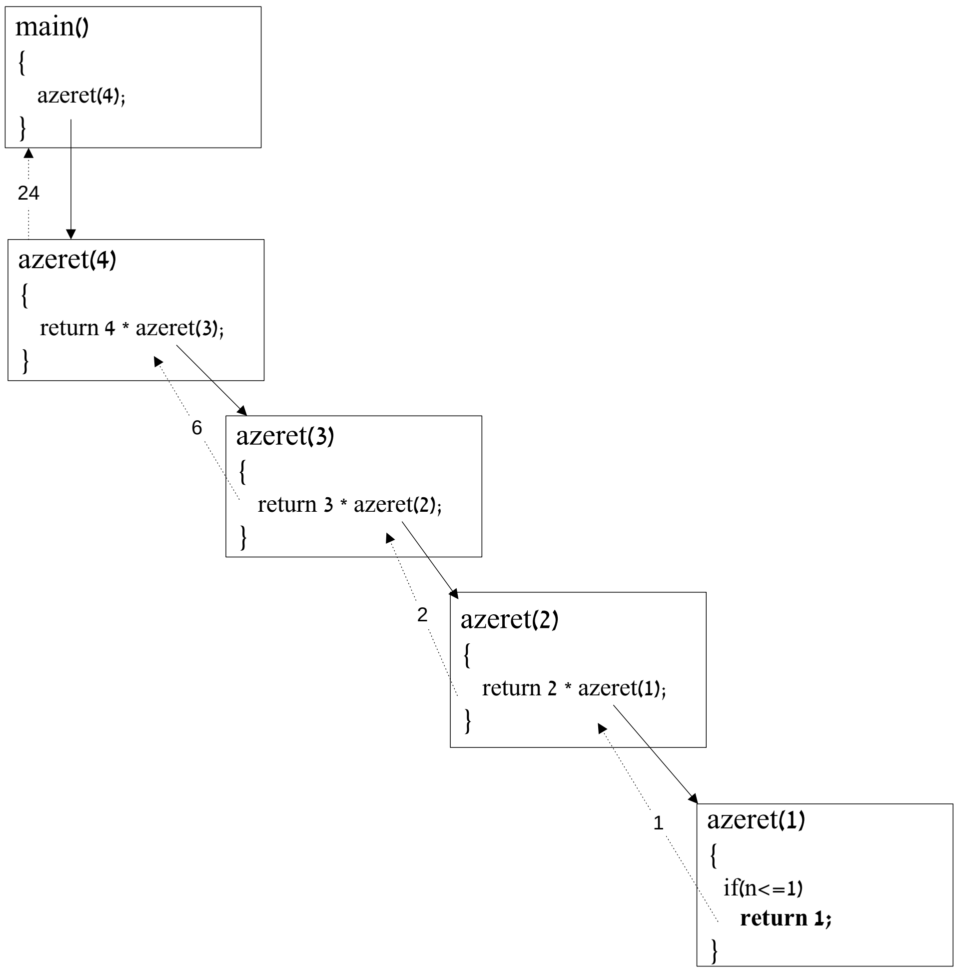
```
	else
		return n*azeret(n-1);
```

התכנית קוראת ומדפיסה את ערך העצרת של 4 ע"י הקריאה

```
	printf("Azeret of 4 is : %d", azeret(4));
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

תרשים הקריאות:

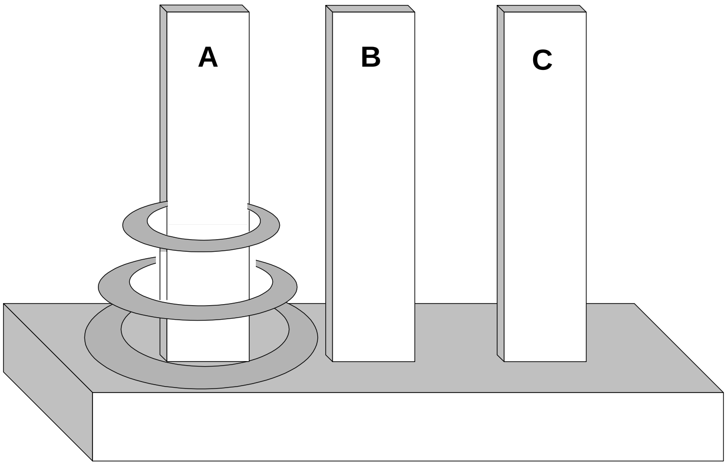




מרכז ההדרכה 2000 © כל הזכויות שמורות

## משחק מגדלי הנוי

מגדלי הנוי הוא משחק עתיק המציב בפני השחקן את האתגר הבא: נתונים 3 מגדלים, A,B,C, כאשר על מגדל A מושחלות n טבעות:




השחקן נדרש להעביר את הטבעות ממגדל A למגדל C, כך שהן יהיו מסודרות עפ"י סדר, כפי שהן במגדל A - טבעת קטנה נמצאת מעל טבעת גדולה יותר. יש לשמור על הכללים הבאים:

- בכל שלב מועברת טבעת יחידה.
- ניתן להשתמש במגדל B להעברות ביניים. כמו כן ניתן בכל שלב להעביר טבעת מכל מגדל לכל מגדל.
- בכל שלב, בכל מגדל חייב להישמר סדר יורד של טבעות, כלומר טבעת קטנה תהיה מעל טבעת גדולה יותר.

לדוגמא, עבור n=2, הפתרון יהיה:

- העבר את הטבעת הקטנה מ- A ל- B
- העבר את הטבעת הגדולה מ- A ל- C
- העבר את הטבעת הקטנה מ- B ל- A

עבור n=3 הפתרון יהיה:

- העבר את הטבעת הקטנה מ- A ל- C
- העבר את הטבעת הבינונית מ- A ל- B
- העבר את הטבעת הקטנה מ- C ל- B
- העבר את הטבעת הגדולה מ- A ל- C
- העבר את הטבעת הקטנה מ- B ל- A
- העבר את הטבעת הבינונית מ- B ל- C

– העבר את הטבעת הקטנה מ- A ל- C

**שאלה**: עבור n כלשהו מהו האלגוריתם לפתרון ?

**תשובה**: הדרך הפשוטה ביותר להגדרת האלגוריתם היא באופן רקורסיבי. נגדיר אלגוריתם עבור פונקציה רקורסיבית, move, באופן הבא:

***פונקציה move:***

***פרמטרים: n - מספר הטבעות על המגדל המקורי***

***t1 - המגדל שממנו מעבירים את הטבעות***

***t2 - המגדל שדרכו מועברות הטבעות***

***t3 - המגדל שאליו מועברות הטבעות***

***אם n=1***

***העבר את הטבעת (היחידה) מ- t1 ל- t3***

***אחרת***

***קרא ל- move להעברת n-1 טבעות מ- t1 ל- t2 דרך t3***

***העבר את הטבעת ה- n-ית (הגדולה ביותר) מ- t1 ל- t3***

***קרא ל- move להעברת n-1 הטבעות מ- t2 ל- t3 דרך t1***

כלומר, הרעיון הוא לפתור בכל שלב את הבעיה עבור מספר קטן ב- 1 של מספר הטבעות, באופן רקורסיבי. קוד התכנית המממשת את האלגוריתם מובא בעמ' 160-161.

תרגול

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 162.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# סיכום

- פונקציות הן מנגנון המאפשר חלוקה של משימה מורכבת למשימות קטנות יותר ועצמאיות. חלוקה של התכנית לפונקציות מאפשרת פיתוח מודולרי והדרגתי של התכנית.
- תכנית בשפת C היא אוסף שטוח (ללא קינון) של פונקציות. אחת הפונקציות חייבת להיות **main** - זו הפונקציה שממנה מתחיל ביצוע התכנית.
- הפונקציה הקוראת מעבירה **רשימת פרמטרים (ארגומנטים)** לפונקציה הנקראת. הפרמטרים המועברים (פרמטרים אקטואליים) משוכפלים בפונקציה הנקראת (פרמטרים פורמליים), תוך ביצוע המרה מרומזת או מפורשת במידת הצורך.
- כאשר הפונקציה הנקראת מסתיימת, היא מעבירה את **הערך המוחזר** לפונקציה הקוראת. גם בקבלת הערך המוחזר תתכן המרת טיפוס - מפורשת או מרומזת.
- קיימים מספר סוגי משתנים בהקשר לפונקציות: **גלובליים**, **מקומיים** או **מקומיים סטטיים**. המשתנים הגלובליים מוגדרים מחוץ לכל פונקציה. המשתנים המקומיים והסטטיים מוגדרים בתוך הפונקציה: בתחילתה או בתוך בלוק כלשהו.
- זיכרון התכנית מורכב ממספר קטעים:

  **קטע הנתונים** - מכיל את הקצאות המקום עבור המשתנים הגלובליים.

  **קטע הקוד** - מכיל את קוד הפונקציות שבתכנית (מתורגמות לשפת מכונה).

  **מחסנית הקריאות** - משקפת בזמן ריצה את מצב הקריאות בין הפונקציות (עץ הקריאות).

  **הערימה** - אזור זיכרון נוסף להקצאות מפורשות במהלך התכנית.
- רקורסיה היא תהליך שבו פונקציה קוראת לעצמה. הפונקציה כוללת הוראת קריאה לעצמה ותנאי לסיום הרקורסיה. פונקציות רקורסיביות יעילות לפתרון בעיות בעלות אופי רקורסיבי.

# תרגילי סיכום

***בצע/י את תר' 1-4 שבעמ' 163-164.***

# 7. מערכים

- הגדרת מערך
- אתחול מערך
- פעולות על מערכים
- העברת מערך כפרמטר לפונקציה ושיקולי מודולריות
- מערכים רב–ממדיים
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות www.mh2000.co.il

# הגדרת מערך

**מערך** משמש לאחסון מספר משתנים מאותו טיפוס בסדרה רצופה בזיכרון.

כל איבר ב**מערך** הוא משתנה ללא שם - ההתייחסות אליו היא באמצעות ה<u>אינדקס</u> שלו, כלומר מיקומו ביחס לתחילת ה**מערך**.

ניתן לקרוא את הערך של כל איבר במערך, לשנות את ערכו, להדפיס אותו, לקלוט לתוכו ערך מהקלט - כלומר ניתן לבצע עליו על פעולות כאילו היה משתנה פשוט.

איברי המערך יכולים להיות מטיפוס פשוט, כגון: שלם, ממשי, תו. כמו כן הם יכולים להיות מטיפוס מורכב יותר (מצביע, רשומה, מערך) כפי שנראה בהמשך.

בנוסף, טיפוס האיבר יכול להיות מוגדר משתמש (ע"י typedef) - לדוגמא, טיפוס המחרוזת String.

**תחביר**: הגדרת מערך דומה להגדרת משתנה, בתוספת סוגריים מרובעות וציון מספר האיברים, כלומר גודל המערך, בתוך הסוגריים:

**<טיפוס-איבר>** **<שם-המערך>**[**<גודל-המערך>**];

לדוגמא, הגדרת מערך שלמים בגודל 5:

```
int   integers[5];
```

במערך מגודל n, הערכים מאוחסנים במערך החל מאינדקס 0 ועד ל- n-1. לכן המערך **integers** ייראה כך:

**integers**[] =

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

הגישה לאיבר באינדקס i במערך מבוצעת ע"י שם המערך, בתוספת האינדקס בסוגריים:

**<שם-המערך>**[**i**]

לדוגמא, נציב לאיבר הראשון (אינדקס 0) ערך 23, ולאיבר האחרון (אינדקס 4) ערך 11:

```
integers[0] = 23;
integers[4] = 11;
```

באופן דומה, נדפיס את האיבר שבאינדקס 3 (האיבר הרביעי) כך:

```
printf("integers[3] = %d", integers[3]);
```

נקדם הערך שבתא האחרון ב- 1:

```
integers[4]++;   /* now integers[4] is 12 */
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## תכנית למניית מספר הספרות בקלט

אם בתכנית מסויימת נרצה למנות את מספר המופעים של כל ספרה בקלט, נצטרך לשם כך 10 משתנים. יהיה נוח ופשוט יותר להשתמש **במערך**. נגדיר מערך שלמים בן 10 מקומות

*int digits[10];*

ובכל פעם שנזהה ספרה i בקלט (0<= i <=9) נקדם את האיבר המתאים במערך ע"י:

*digits[i]++;*

קוד התכנית:

```
/* file: digits.c */
#include <stdio.h>
void main ()
{
	int c,i;
	int digits[10];

	/* initialize array */
	for (i=0; i<10; i++)
		digits[i]=0;
	while ((c=getchar())!=EOF)
	{
		if (c >= '0'  &&  c <= '9')
			digits[c-'0']++;
	}
	printf("digits=");
	for (i=0; i<10; ++i)
		printf("%d",digits[i]);
}
```

התכנית הורצה עם הקלט הבא:

```
The 2nd wolrd war started in 1939
```

והפלט:

```
digits=0111000002
```

<u>הסבר</u>: ההגדרה int digits[10] מכריזה על digits כמערך שלמים בן 10 מקומות:

*int digits[10];*

| **digits** [] = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |


מרכז ההדרכה 2000 © כל הזכויות שמורות

זו הסיבה שלולאת ה- for המאפסת את המערך מתחילה מ- 0 ומסתיימת מקום אחד לפני הערך המציין את גודל המערך:

```
for (i=0; i<10; i++)
        digits[i]=0;
```

גוף הלולאה יתבצע רק עבור ערכי i בתחום 0..9. בהגעה ל- 10 יוצאים מהלולאה ללא ביצועה.
בשלב הבא נקרא תו מהקלט, וכל עוד הוא אינו סוף קובץ

```
while ((c=getchar())!=EOF)
```

בודקים אם הוא ספרה ע"י הביטוי

```
if (c>='0' && c<='9')
```

הבדיקה מסתמכת על כך שבטבלת התווים במחשב מיוצגות הספרות החל ממקום מסוים ('0') ברציפות לאורך 10 מקומות (עד '9'). ראה/י טבלת תווי ASCII בנספח. לחילופין, ניתן להחליף ביטוי זה בביטוי:

```
if (isdigit(c))
```

תוך הוספת הכללה לקובץ ctype.h. בביטוי

```
digits[c-'0']++;
```

החיסור c-'0' נותן את אינדקס הספרה המתאימה ומקדם את הערך במערך. לדוגמא, עבור הערך c='3' תוצאת ההפרש

```
digits['3'-'0']++
```

היא

```
digits[3]++
```

שמשמעותה קידום הערך שבתא 3 במערך.

# אתחול מערך

ניתן לאתחל את ערכי המערך ב- 2 אופנים:

1. **אתחול בהגדרה**. לדוגמא, מערך ממשיים מסוג double:

```
double a[4] = {45.33, 12.21, 98.89, 2.5E10};
```

כאן מבוצעת ההצבה לאיברי המערך בזמן הגדרתו. במקרה זה ניתן גם להגדיר את המערך ללא ציון המימד, לדוגמא:

```
double a[] = {45.33, 12.21, 98.89, 2.5E9};
```

המהדר ״מבין״ עפ״י רשימת איברי האתחול מהו גודל המערך המבוקש.

2. אתחול ע״י **הצבה מפורשת**:

```
int main()
{
	double a[4];

	a[0] = 45.33;
	a[1] = 12.21;
	a[2] = 98.89;
	a[3] = 2.5E9;
}
```

כאן אנחנו מציבים ערכים במפורש לתאי המערך **a**:

| **a**[] = | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| | 45.33 | 12.21 | 98.89 | 2.5E9 |

בדרך כלל פעולות על מערכים מבוצעות בלולאות. לדוגמא, התכנית הבאה מגדירה מערך גלובלי:

```
#include <stdio.h>

#define ARRAY_SIZE   7
int array[ARRAY_SIZE];      /* define a  global array */
```

המערך מוגדר ע״י קבוע קדם-מעבד. ניתן כעת לכתוב פונקציה לאיתחול המערך מהקלט:

```
void input()
{
	int i;
	for (i=0; i<ARRAY_SIZE; i++)
	{
		scanf("%d",&array[i]);
	}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

}

<u>הסבר</u>: הפונקציה עוברת בלולאה על איברי המערך הגלובלי וקוראת ערכים לתוכו ע"י

***scanf("%d",&array[i]);***

array[i] הוא האיבר ה- i במערך, והקידומת & מציינת את כתובתו עבור scanf. הרצה לדוגמא של הפונקציה הנ"ל עם הקלט:

```
2 7 -1 0 19 666 -29
```

תרשים המערך לאחר ביצוע הפונקציה:

**array** [] =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 2 | 7 | -1 | 0 | 19 | 666 | -29 |

## כללים

- ערכי המחדל של איברי מערך באיתחול הם עפ"י הכללים של משתנים פשוטים: אם המערך מוגדר כמקומי בפונקציה, ערכי איבריו לא מוגדרים ("זבל"). אם הוא מוגדר כגלובלי, האיברים הם בעלי ערך 0.
- גם מבחינת משך הקיום וטווח ההכרה (scope) הכללים לגבי המערך הם בדומה למשתנים פשוטים. ראה/י פרק 6, סעיף "משתנים ופונקציות".

# פעולות על מערכים

### מציאת ערך מקסימום

הפונקציה find_max המובאת בעמ׳ 170 מוצאת ומחזירה את המספר הגדול ביותר במערך.

### בדיקת קיום של ערך מסוים במערך

הפונקציה exist המובאת בעמ׳ 170 בודקת קיום של ערך הניתן לה כפרמטר במערך.

### מיון איברי המערך - שיטת מיון ״בועות״

כדי למיין את איברי המערך נגדיר פונקציה בשם sort שתבצע את המיון ע״י שיטת **מיון בועות (Bubble sort)**.

בשיטה זו משווים בין כל 2 איברים סמוכים ואם יש צורך מחליפים בין ערכיהם.

לדוגמא, נניח שנתון מערך של 7 שלמים עם הערכים הבאים:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 8 | 5 | 21 | 17 | 100 | -5 | 12 |

אנו מעוניינים למיין את המערך בסדר עולה. נבצע את המיון בלולאה עם מספר חזרות. חזרה ראשונה:

– משווים בין איבר 0 ואיבר 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 8 | 5 | 21 | 17 | 100 | -5 | 12 |

– מכיוון ש- 8 גדול מ- 5 מחליפים בין ערכי האיברים:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **5** | **8** | 21 | 17 | 100 | -5 | 12 |

מרכז ההדרכה 2000 © כל הזכויות שמורות

– משווים בין איבר 1 ואיבר 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 21 | 17 | 100 | -5 | 12 |

מכיוון ש- 8 קטן מ- 21 לא מבצעים החלפה.

– משווים בין איבר 2 ואיבר 3

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 21 | 17 | 100 | -5 | 12 |

– מכיוון ש- 21 גדול מ- 17 מחליפים בין ערכי האיברים:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | **17** | **21** | 100 | -5 | 12 |

– השוואה בין איברים 3,4 - אין החלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | 100 | -5 | 12 |

– השוואה בין איברים 4,5 והחלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | **-5** | **100** | 12 |

– השוואה בין איברים 5,6 והחלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | -5 | **12** | **100** |

כפי שניתן לראות, בסוף החזרה הראשונה נמצא האיבר המקסימלי (100) בסוף המערך.

בחזרה השנייה יש לבצע כנ"ל על 5 האיברים הראשונים בלבד:

– השוואה בין איברים 0,1 - אין החלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | -5 | 12 | 100 |

– השוואה בין איברים 1,2 - אין החלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | -5 | 12 | 100 |

– השוואה בין איברים 2,3 - אין החלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | 21 | -5 | 12 | 100 |

– השוואה בין איברים 3,4 והחלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | **-5** | **21** | 12 | 100 |

– השוואה בין איברים 4,5 והחלפה:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5 | 8 | 17 | -5 | **12** | **21** | 100 |

בסוף החזרה השנייה נמצא האיבר השני בגדלו (21) במקומו. באופן זהה ממשיכים בביצוע חזרות עד למיון המלא של המערך.

שמו של אלגוריתם מיון זה בא לו מתכונה זו: בכל שלב **מוצף** האיבר המקסימלי לסוף המערך בדומה ל**בועה**.

**שאלה**: כמה חזרות נדרשות למיון מערך בגודל n ?

**תשובה**: מכיוון שבכל חזרה מוצף איבר אחד למקומו, אזי במקרה הגרוע ביותר - האיבר המינימלי נמצא בסוף המערך - נצטרך n-1 חזרות להזזתו מהסוף להתחלה.

נראה כעת מימוש של מיון בועות ע"י הפונקציה sort:

```
/* sort the array in ascending order */
void sort()
{
	int i,j;
	int temp;

	for(i=0; i<ARRAY_SIZE - 1; i++)
	{
		for(j=0; j<ARRAY_SIZE - 1 - i; j++)
		{
			if(array[j] > array[j+1])
			{
				temp = array[j];
				array[j] = array[j+1];
				array[j+1] = temp;
			}
		}
	}
}
```

הפונקציה מבצעת מעבר בלולאה כפולה על איברי המערך:

- בלולאה החיצונית מבצעים n-1 חזרות.
- בלולאה הפנימית עוברים עפ"י הסדר על כל אחד מאיברי המערך (עד לאיבר שלפני האחרון), משווים אותו לאיבר העוקב לו ומחליפים ביניהם אם יש צורך.

באופן כללי, אחרי החזרה ה- i -ית האיבר ה- i בגדלו נמצא במקומו. לכן, בלולאה הפנימית אין צורך לעבור ולהשוות את כל האלמנטים - די לעבור על מספר האלמנטים פחות i.

התכנית במלואה מובאת בעמ' 174-176 וכוללת את הפונקציות:

- input() - פונקציה לאתחול איברי המערך מהקלט
- output() - פונקציה להדפסת איברי המערך לפלט
- find_max() - פונקציה המוצאת ומחזירה את ערך האיבר המקסימלי במערך
- exist() - פונקציה המחזירה ערך **אמת** אם ערך נתון קיים במערך, **שקר** אחרת
- sort() - פונקציה המבצעת מיון של המערך בשיטת "בועות"
- main() - הפונקציה הראשית בתכנית. פונקציה זו קוראת לשאר הפונקציות ו"מנהלת" את מהלך התכנית.

# העברת מערך כפרמטר לפונקציה ושיקולי מודולריות

בתכנית האחרונה הגדרנו את המערך כמשתנה גלובלי כדי לאפשר שיתופו בין מספר פונקציות. לו היינו מגדירים אותו כמקומי בפונקציה main למשל, הוא לא היה מוכר בפונקציות האחרות.

כאשר מועבר נתון פשוט כפרמטר לפונקציה נוצר שיכפול שלו בפונקציה הנקראת. שינוי ערכו בפונקציה זו לא משפיע על ערכו אצל הפונקציה הקוראת.

לעומת זאת מערכים המועברים כפרמטרים לא משוכפלים בפונקציה הנקראת. הפונקציה מתייחסת לאותו מקום בזיכרון בו הוגדר המערך בפונקציה הקוראת.

לדוגמא, הפונקציה הראשית בתכנית הבאה מעבירה את המערך כפרמטר לפונקציה המקדמת את ערכי כל איברי המערך ב- 1 :

```
#include <stdio.h>
void increment_array(int arr[], int size)
{
	int i;
	for(i=0; i<size; i++)
		arr[i]++;
}

void main()
{
	enum { ARRAY_SIZE=5 };
	int array[ARRAY_SIZE] = {6,9,2,5,18};
	int i;

	increment_array(array, ARRAY_SIZE);
	for( i=0; i<ARRAY_SIZE; i++)
		printf("%d ", array[i]);
}
```

פלט התכנית :

```
7 10 3 6 19
```

הסבר : הפונקציה increment_array() מבצעת קידום של כל איברי מערך השלמים ב- 1. לצורך כך, היא מקבלת את המערך ואת גדלו כפרמטרים :

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

הפונקציה ()increment_array פועלת על המשתנה הפורמלי arr, המצביע למערך האקטואלי שמועבר בפונקציה הקוראת ()main. הפונקציה עוברת על כל איבריו ומקדמת אותם ב- 1.

הפונקציה הקוראת, ()main, מגדירה את המערך array כמקומי. לצורך ציון גדלו, מוגדר הקבוע ARRAY_SIZE כ- enum :

```
enum { ARRAY_SIZE=5 };
int array[ARRAY_SIZE] = {6,9,2,5,18};
```

> *הערה: כאשר מציינים גודל מערך בשפת C, חובה לציין קבוע מספרי (ליטרל), קבוע שהוגדר ע"י קדם-המעבד (pre-processor) או קבוע שהוגדר ע"י enum. לא ניתן לציין כגודלו של המערך קבוע שהוגדר ע"י const.*

בהמשך, מבצעים קריאה לפונקציה increment_array תוך העברת המערך וגדלו כפרמטרים:

```
increment_array(array, ARRAY_SIZE);
```

לבסוף מודפסים איברי המערך:

```
for( i=0; i<ARRAY_SIZE; i++)
	printf("%d ", array[i]);
```

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## פונקציות מודולריות

העברת המערך כפרמטר לפונקציה היא שיטת תכנות מודולרית יותר ועדיפה על פני הגדרתו כמשתנה גלובלי ממספר סיבות:

- ניתן לבצע שימוש חוזר בפונקציה המקבלת את המערך כפרמטר עבור מספר מערכים.
- המעטה במספר המשתנים הגלובליים מקילה על ניפוי התכנית, כלומר מציאת השגיאות בזמן הריצה.

נחזור לתכנית המטפלת במערכים: נגדיר את המערך כמקומי ב- main, שתעביר אותו לשאר הפונקציות כפרמטר -

```
/* file: array2.c */
#include <stdio.h>
```

– הגדרת הפונקציות:

```
/* initialize the array by input from the user */
void input(int arr[], int size)
{
	int i;

	printf("\nEnter %d numbers:", size);
	for (i=0; i<size; i++)
	{
		scanf("%d",&arr[i]);
	}
}

/* output array elements  */
void output(int arr[], int size)
{
	int i;

	printf("\nArray elements:");
	for (i=0; i<size; i++)
	{
		printf("%d\t",arr[i]);
	}
}
...
```

– הפונקציה main() :

```
int main()
{
	enum {ARRAY_SIZE=7};
	int num;
	int array[ARRAY_SIZE];

	/* initialize the array */
	input(array, ARRAY_SIZE);
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

```
	/* find max number in the array */
	printf("\nThe maximum number is %d", find_max(array, ARRAY_SIZE));

	/* is a given number exist in the array */
	printf("\nEnter a number to find in the array:");
	scanf("%d",&num);
	if(exist(array, ARRAY_SIZE, num))
		printf("\nThe number %d exist in the array", num);
	else
		printf("\nThe number does not exist in the array");

	/* sort the array */
	printf("\nSorting the array...");
	sort(array, ARRAY_SIZE);
	output(array, ARRAY_SIZE);
	return 0;
}
```

יש לשים לב לנקודות החשובות בגירסה זו של התכנית:

- המערך מוגדר כמקומי בפונקציה main ומועבר כפרמטר לפונקציות השונות.
- גודל המערך - הקבוע ARRAY_SIZE - מוגדר באופן מקומי בפונקציה main ע"י enum והוא אינו מוכר מחוץ לה. באופן זה **מרחב השמות** (**namespace**) הגלובלי אינו מזוהם.

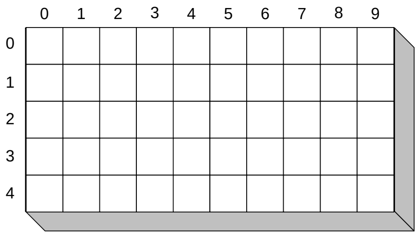

*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# מערכים רב-ממדיים

ניתן להגדיר מערך ממספר ממדים. לדוגמא, הגדרת מערך דו-ממדי בגודל 5 על 10 :

```
int     matrix[5][10];
```

המימד הראשון המצוין, 5, הוא מימד השורות והמימד השני, 10, הוא מימד העמודות :




לדוגמא, תכנית המדפיסה מטריצה שבאלכסונה הראשי '1' -ים ובכל שאר איבריה אפסים :

```
/* file: matrix.c */
#include <stdio.h>
#define LINES_NO             5
#define COLUMNS_NO         10

void main()
{
        char  matrix[LINES_NO][COLUMNS_NO];
        int    i,j;

        for(i=0; i<LINES_NO; i++)
                for(j=0; j<COLUMNS_NO; j++)
                        if(i==j)
                                matrix[i][j] = '1';
                        else
                                matrix[i][j] = '0';

        for(i=0; i<LINES_NO; i++)
        {
                for(j=0; j<COLUMNS_NO; j++)
                        putchar(matrix[i][j]);
                putchar('\n');
        }
}
```

פלט התכנית :

```
1000000000
0100000000
0010000000
0001000000
0000100000
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## אתחול מערך רב-ממדי

כמו מערך חד-ממדי, ניתן לאתחל מערך רב ממדי ע"י לולאה כנ"ל או ע"י אתחול בהגדרה. דוגמאות:

1. מטריצת ממשיים בגודל 2x3 :

*float matrix1[][3] = { {1.0, 1.5, 2.0}, {3.0,3.5,4.0}};*

ציון המימד הראשון אופציונלי, את השני חובה לציין.

2. מערך תווים תלת-ממדי בגודל 2x2x3 :

*char array3D[][2][3] = { {{'1', '2', '3'}, {'a', 'b', 'c'}}, {{'4','5','6'},{'e','f','g'}} };*

המימד הראשון הוא אופציונלי, את השני והשלישי חובה לציין. ככלל, אתחול בזמן הגדרה מחייב לציין את n-1 הממדים האחרונים של מערך ממימד n.

## העברת מערך רב-ממדי כפרמטר לפונקציה

אם מערך דו-ממדי מוגדר כך :

*float array[3][4];*

פונקציה המקבלת אותו כפרמטר תוגדר כך :

*void func(float array[][4]);*

או כך :

*void func(float array[3][4]);*

ציון המימד הראשון הוא אופציונלי, אולם את המימד השני של המערך חובה לציין כדי שהמהדר ידע כיצד לתרגם בפונקציה func את האינדקסים. לדוגמא :

```
void func(float array[][4])
{
	array[1][3] = 4.56f;
}
```

מערך רב-ממדי מיוצג בשפת C ע״י פרישתו למערך חד-ממדי :

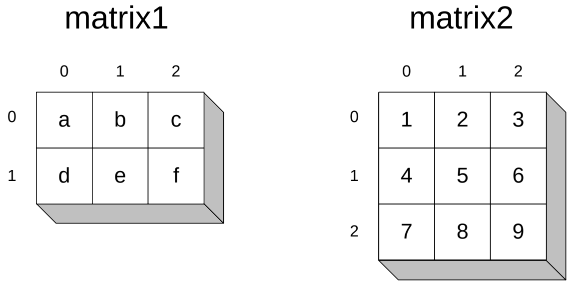
**array**[] =

| 0-0 | 0-1 | 0-2 | 0-3 | 1-0 | 1-1 | 1-2 | 1-3 | 2-0 | 2-1 | 2-2 | 2-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | **4.56** | | | | |

המהדר חייב לדעת שמימד העמודות של המערך הוא 4 כדי לדעת את מיקום האיבר array[1][3].

כלומר, מיקום האיבר array[1][3] בפונקציה שמגדירה את המימד השני של המערך כ- 4 מחושב ע״י array[1*4 + 3].

בתכנית הדוגמא המובאת בעמ׳ 181 כוללת שני מערכים דו-מימדיים (מטריצות) של תווים :




***עיין/י בקוד התכנית.***

## מיון שורות ועמודות מערך רב-ממדי

לשורות של מטריצה ניתן להתייחס כאל מערך רגיל מכיוון שהמטריצה נפרשת בזיכרון למערך חד ממדי שורה-שורה. לדוגמא, נניח שמוגדרת המטריצה הבאה:

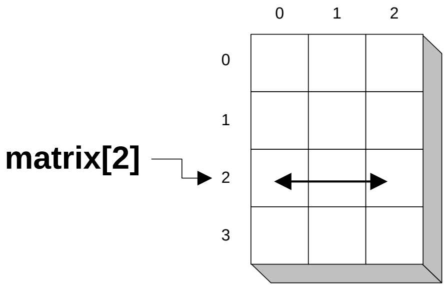
```
#define COLUMNS_NO 3

char matrix[4][COLUMNS_NO] =
        {{'f', 'a', 'h'} ,{'w', 'c', 'k'}, {'g', 'o', 'l'}, {'d', 'j', 'i'}};
```

פונקציה למיון שורה אחת מהמטריצה ניתן לכתוב כך:

```
void sort_line(char line[], int columns_no)
{
        int     i,j;
        char temp;

        for(i=0; i<columns_no - 1; i++)
                for(j=0; j<columns_no-1-i; j++)
                        if(line[j] > line[j+1])
                        {
                                temp = line[j];
                                line[j] = line[j+1];
                                line[j+1] = temp;
                        }
}
```

מתוך התכנית המשתמשת הפונקציה תיקרא כך, למשל:

```
sort_line(matrix[2], COLUMNS_NO);
```



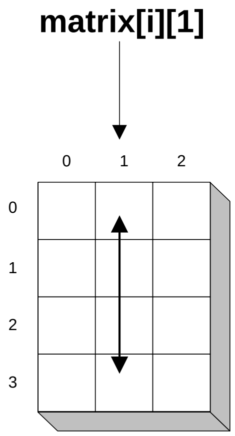

קריאה זו תגרום למיון שורה 2 במטריצה.

<u>תרגיל</u>: מה יבוצע אם במקום הפרמטר COLUMNS_NO יועבר לפונקציה sort_line הערך 6:

```
sort_line(matrix[2], 6);
```

מיון עמודות

בפונקצית המיון של העמודות, הואיל והם לא נפרשים באופן רציף בזיכרון - יש לקבל כפרמטר את המטריצה, את מספר השורות הכולל ואת אינדקס העמודה למיון:




פונקצית המיון של עמודה במטריצה:

```
void sort_column(char matrix[][COLUMNS_NO], int lines_no, int col)
{
	int	i,j;
	char	temp;

	for(i=0; i<lines_no - 1; i++)
	{
		for(j=0; j<lines_no-1-i; j++)
		{
			if(matrix[j][col] > matrix[j+1][col])
			{
				temp = matrix[j][col];
				matrix[j][col] = matrix[j+1][col];
				matrix[j+1][col] = temp;
			}
		}
	}
}
```

והפונקציה תיקרא כך, למשל:

```
sort_column(matrix, 4,  1);
```

הוראה זו תגרום למיון עמודה 1 במטריצה. יש לשים לב שגודל העמודות חייב להיות ידוע לפונקציה, בכדי שהיא תתורגם נכון, לכן מספר העמודות מוגדר כקבוע ומצוין כמימד שני. לעומת זאת, מספר השורות מועבר כפרמטר.

תכנית המיון הכוללת

***קוד תכנית המיון הכוללת מובאת בעמ' 185-187.***


מרכז ההדרכה 2000 © כל הזכויות שמורות

# סיכום

- **מערך** בגודל n הוא אוסף איברים מטיפוס זהה המסודרים ברציפות בזיכרון מאינדקס 0 ועד אינדקס n-1. ניתן להגדיר מערך ממספר ממדים.
- איברי המערך ניתנים לגישה ע"י ציון שם המערך בצירוף האינדקס בסוגריים מרובעות [].
- ניתן לאתחל מערך ע"י מעבר על האיברים בלולאה ובהצבת ערכים מתאימים או ע"י אתחול בהגדרה.
- פעולות על איברי המערך מבוצעות בדרך כלל בלולאות:
  - מציאת ערך מקסימום
  - בדיקה אם ערך מסוים קיים במערך
  - מיון
- **מערך רב ממדי** הוא מערך בעל מספר ממדים. ניתן להגדיר מערך ב- C ממספר לא מוגבל של מימדים, והוא מיוצג בזיכרון ע"י פרישתו כמערך חד-ממדי. חובה לציין את n-1 הממדים האחרונים של מערך בעל n ממדים באתחולו ובהעברתו כפרמטר לפונקציה.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבעמ' 188.***

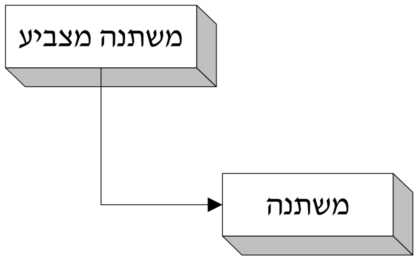
# 8. מצביעים

- הגדרת מצביע
- גישה למשתנה דרך המצביע אליו
- העברת פרמטר לפונקציה ע"י מצביע
- מצביעים ומערכים
- פעולות חשבוניות על מצביעים
- מצביעים לפונקציות
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

# הגדרת מצביע

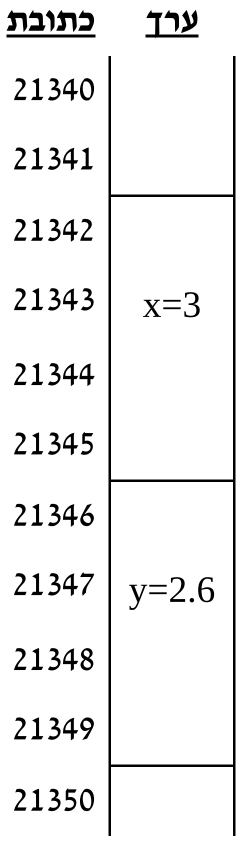
מצביע הוא משתנה שערכו הוא כתובת של משתנה אחר. כאשר משתנה אחד מכיל כתובת של משתנה אחר נאמר שהמשתנה הראשון **מצביע** למשתנה השני:




לדוגמא, נתון קטע התכנית הבא:

```
void main()
{
	int x;
	float y;

	x=3;
	y=2.6;
}
```

נניח שתמונת הזיכרון של המשתנים בתכנית נראית כך:



בעמודה השמאלית מוצגות הכתובות בזיכרון באופן רציף, כל כתובת מתייחסת לבית (Byte) יחיד, כלומר לנתון בן 8 סיביות.

בעמודה הימנית מוצגים ערכי המשתנים שהוגדרו בתכנית, כאשר במערכת המסוימת שעליה הופעלה הדוגמא משתנה שלם הוא בן 4 בתים, וכך גם משתנה ממשי.

מהן כתובות המשתנים בתכנית?

21342 - x

21346 - y

נניח שאנו רוצים ששתי כתובות המשתנים y,x יוצבו במצביעים מתאימים xptr ו - yptr בהתאמה. ראשית צריך להגדיר אותם:

*int *xptr;*
*float *yptr;*

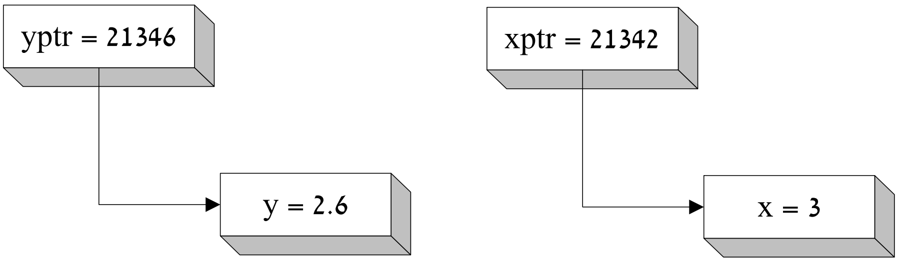
על ידי האופרטור "*" הגדרנו את xptr ו- yptr מטיפוס <u>מצביעים</u>: xptr **מצביע לשלם** ו- yptr **מצביע לממשי**. כעת נרצה להציב להם את כתובות x ו- y . כתובת המשתנה ניתנת ע"י אופרטור הכתובת "&":

*xptr = &x;*
*yptr = &p;*

פירושו של האופרטור & הוא "כתובתו של". ערכי המצביעים כעת:

21342 - xptr

21346 - yptr




***התכנית במלואה ותמונת הזיכרון מובאים בעמ' 192-191.***

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 192.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# גישה למשתנה דרך המצביע אליו

ניתן לשנות את ערך המשתנה ע״י המצביע אליו, לדוגמא:

```
void main()
{
	int x=2;
	int *px;

	px=&x;
	*px=36;
}
```

ערכו של x לאחר השורה האחרונה בתכנית שווה ל- 36. כמו כן ניתן לבצע דרך מצביע למשתנה כל פעולה שניתן לבצע על המשתנה עצמו. דוגמאות:

```
*px = *px + 5;   /*     equivalent to:  x = x + 5       */
(*px)++;         /*     equivalent to:  x++             */
*px++ = 5;       /*     equivalent to: *px = 5;  px++   */
```

יש לשים לב בדוגמא האחרונה שלאופרטור ++ אמנם יש קדימות על פני האופרטור * אך מכיוון שהוא בצורת postfix הוא מבוצע רק לאחר סיום המשפט כולו.

האופרטור "*" מורה למהדר לגשת לערך המשתנה המוצבע ע״י המצביע xptr שהוא x. כלומר משמעות האופרטור "*" היא ״הנתון המוצבע על-ידי״.

לדוגמא, בהגדרת המצביע

*int *px;*

אנו בעצם מכריזים ״הנתון המוצבע על-ידי px הוא מסוג int״.

ואילו בפעולת הצבה

**px=36;*

אנו מורים למהדר ״הצב 36 לנתון המוצבע ע״י px״.

*הערה: האופרטורים & ו- * הם בעצם הפכיים. האופרטור & מחזיר את כתובת המשתנה ואילו האופרטור * מחזיר את הנתון המוצבע ע״י הכתובת הנתונה.*

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

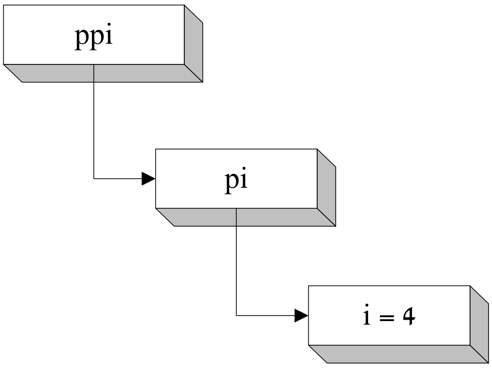
## מצביע למצביע

ניתן להגדיר מצביע למצביע ובכך להגדיל את רמת העקיפות בתכנית. מגדירים מצביע למצביע בצורה דומה להגדרת מצביע רגיל, לדוגמא:

```
int  ** ppi;
```

באופן זה ניתן לקבל את כתובתו של משתנה מסוג מצביע, לדוגמא:

```
int  i;
int *pi = &i;
int  ** ppi = &pi;

**ppi  = 4; /* i = 4 */
```




השימוש במצביעים למצביעים (או ברמה עקיפה יותר) אינו שכיח בשפת C כמו השימוש במצביעים מדרגה ראשונה.

מצביעים למצביעים שימושיים במערכי מצביעים (כפי שנראה בפרק 9, "**מחרוזות**") וכן בפונקציות הנדרשות לשנות את כתובת המצביע, כגון: פונקציות המבצעות הקצאת זיכרון דינמית.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# העברת פרמטר לפונקציה ע"י מצביע

### העברת פרמטרים ע"י ערך והעברה ע"י התייחסות

כפי שכבר ראינו, העברת הפרמטרים בשפת C היא עפ"י ערך (by value).

כלומר, שינוי **הפרמטרים הפורמליים** בתוך הפונקציה הנקראת אינו משפיע על **הפרמטרים האקטואליים** מכיוון שהם העתק שלהם.

לדוגמא, נניח שמוגדרת פונקציה בשם swap להחלפה בין ערכיהם של שני משתנים שלמים:

```
/* swap between 2 integers */
void swap(int a1, int a2)
{
	int a3 = a1;

	a1 = a2;
	a2 = a3;
}
```

והפונקציה הקוראת לה:

```
void main()
{
	int x=5, y=3;

	swap(x,y);
	printf("x=%d\n", x);
	printf("y=%d\n", y);
}
```

<u>שאלה</u>: מה יהיה פלט התכנית?

<u>תשובה</u>:

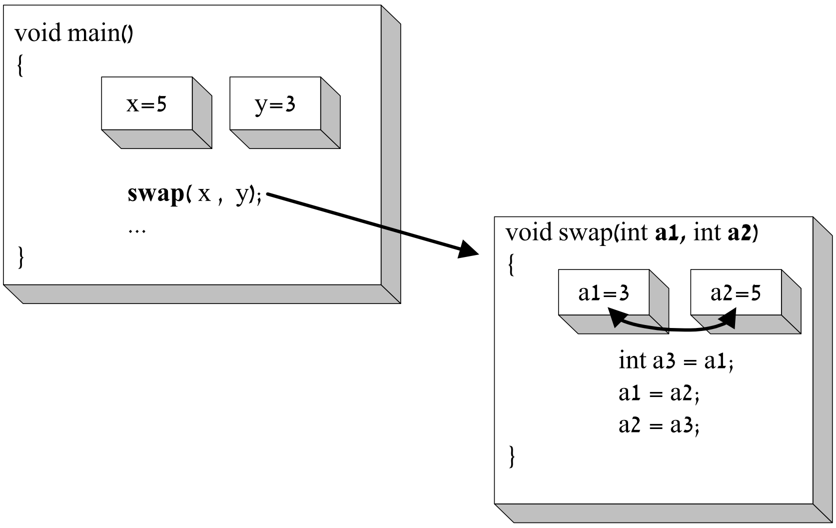
```
x = 5
y = 3
```

כלומר ערכי המשתנים לא שונו!!!

הפונקציה swap מגדירה שני פרמטרים פורמליים a1,a2. כאשר main קוראת ל- swap מועתקים ערכי x,y למשתנים a1,a2 בהתאמה.

בסופה של swap ערכיהם של a1 ו- a2 מוחלפים, אך עם חזרת הביצוע ל- main לא מושפעים המשתנים x,y והם מודפסים ללא שינוי:

 מרכז ההדרכה 2000 © כל הזכויות שמורות




צורה זו של העברת פרמטרים נקראת **העברת פרמטרים ע"י ערך** (by value), זאת מכיוון שרק ערך הפרמטרים מועבר מהפונקציה הקוראת לפונקציה הנקראת.

בשפת C זוהי השיטה היחידה המקובלת. בשפות מסוימות קיימת דרך אחרת להעברת פרמטרים - **העברה ע"י התייחסות** (by reference/var), לדוגמא, בפסקל וב- C++.

***שיפור ב C++ :*** ניתן להגדיר לפונקציה העברת פרמטר ע"י **התייחסות** (By Reference) תוך שימוש בסימן & בהגדרת הפרמטרים לפונקציה:

```
void swap(int & a1, int & a2)
```

לדוגמא, את הפונקציה swap() ניתן לכתוב כך ב- C משופרת:

```
/* file: swap.cpp */
#include <stdio.h>
/* swap between 2 integers */
void swap(int & a1, int & a2)
{
   int a3 = a1;

   a1 = a2;
   a2 = a3;
}

void main()
{
   int x=5, y=3;

   swap(x,y);
   printf("x=%d\n", x);
   printf("y=%d\n", y);
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

```
}
```

וכעת הפלט:

**x=3**
**y=5**


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## העברת פרמטרים ע"י התייחסות - שימוש במצביעים

אחד השימושים במצביעים הוא בהעברת פרמטר לפונקציה, כאשר היא אמורה לשנות את ערכו.

**הרעיון**: הפונקציה הקוראת תעביר כפרמטרים את **כתובות** המשתנים, והפונקציה הנקראת תיגש לנתונים המוצבעים על ידם ותשנה אותם.

נחזור לדוגמא הקודמת ונגדיר את הפונקציה swap אחרת:

```
#include <stdio.h>

/* swap between 2 integers */
void swap(int *pa1, int *pa2)
{
	int a3 = *pa1;

	*pa1 = *pa2;
	*pa2 = a3;
}

void main()
{
	int x=5, y=3;

	swap(&x,&y);
	printf("x=%d\n", x);
	printf("y=%d\n", y);
}
```

והפעם הפלט הוא כנדרש:

```
x = 3
y = 5
```

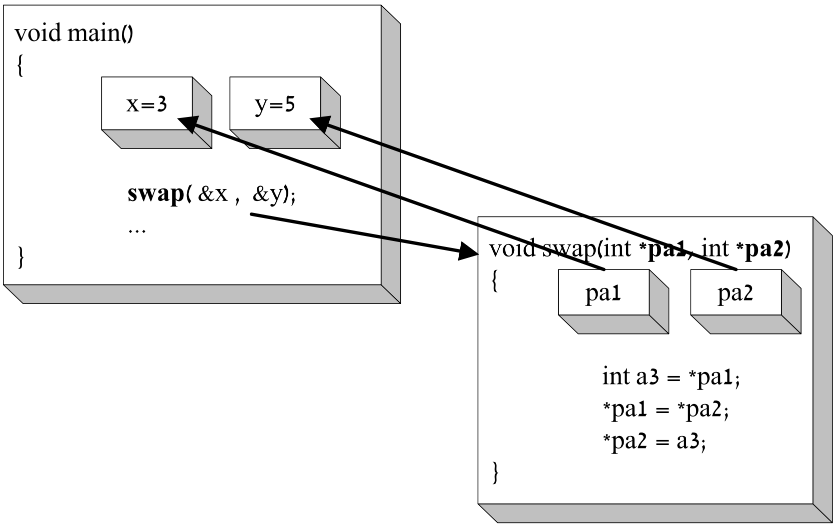
<u>הסבר</u>

הפונקציה swap מקבלת כפרמטרים 2 מצביעים:

```
void swap(int *pa1, int *pa2)
```

הפונקציה מחליפה בין ערכי המשתנים המוצבעים ע"י pa1 ו- pa2, שהם המשתנים x,y המוגדרים ב- main ואשר כתובותיהם הועברו כפרמטרים:

```
swap(&x,&y);
```




בעמ' 198-199 מוצגת **מחסנית הקריאות (Call Stack)** לאורך מהלך התכנית. עיין/י בעמודים אלו.

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## דוגמא נוספת - שימוש בפונקציה swap במיון

נתייחס לפונקצית מיון המערך שראינו בפרק 7, "**מערכים**". השתמשנו בפונקצית sort שביצעה מיון בשיטת "מיון בועות". תוך שימוש בפונקציה swap נוכל לפשט את פעולת sort:

– הפונקציה ()swap:

```
void swap(int *pa1, int *pa2)
{
	int a3 = *pa1;

	*pa1 = *pa2;
	*pa2 = a3;
}
```

– הפונקציה ()sort:

```
void sort(int arr[], int size)
{
	int i,j;

	for(i=0; i < size - 1; i++)
		for(j=0; j < size -1 - i; j++)
			if(arr[j] > arr[j+1])
				swap(&arr[j], &arr[j+1]);
}
```

<u>הסבר</u>: במיון "בועות", כאשר איבר מסויים גדול מהאיבר העוקב לו, הפונקציה ()sort קוראת ל-()swap להחלפה בין שני איברים במערך ע"י העברת **כתובותיהם** כפרמטרים:

```
swap(&arr[j], &arr[j+1]);
```

ע"י העברת מצביע כפרמטר, יכולות הפונקציות הקוראת והנקראת להתייחס לאותה כתובת בזיכרון. מבחינת העברת פרמטרים לפונקציות, אנו רואים שהמצביע "מתנהג" כמו מערך.

## שאלת חזרה

הסבר/י - תוך התייחסות להעברת פרמטרים ע"י ערך וע"י התייחסות - מדוע בקריאה לפונקציה scanf יש צורך להעביר כפרמטרים את כתובות המשתנים.

כמו כן, הסבר/י מדוע בפונקציה printf אין בכך צורך.


מרכז ההדרכה 2000 © כל הזכויות שמורות

# מצביעים ומערכים

## שקילות בין מערך ומצביע

בשפת C מערך הוא מצביע, כלומר, שם המערך הוא כתובת תחילתו בזיכרון. לדוגמא, נגדיר מערך שלמים קצרים בשם array:

```
short array[5] = {45,56,12,98,12};
```

תמונת הזיכרון של המערך במערכת שבה short הוא בגודל 2 בתים:

| | **כתובת** | **ערך** |
|---|---|---|
| array[] = | 243890 | 45 |
| | 243892 | 56 |
| | 243894 | 12 |
| | 243896 | 98 |
| | 243898 | 12 |

array הוא בעצם מצביע לאיבר הראשון במערך. כלומר ערכו של array בדוגמא זו הוא 243890.
לכן ניתן לבצע עליו פעולות מצביעים כגון:

```
*array = 0;               /* array[0] = 0 */
*(array+2) =8;   /* array[2] = 8 */
```

כמו כן ניתן להגדיר מצביע מתאים ולבצע פעולות חוזרות על איברי המערך:

```
short *p = array;
for(i=0; i<5; i++)
        *(p+i) = i;
```

<u>העברה כפרמטר לפונקציה</u>

ניתן להכריז על פרמטר שפונקציה מקבלת כעל מצביע ובפועל לקבל מערך ולהפך. לדוגמא, הפונקציה הבאה מקבלת כפרמטר מצביע למערך תווים, ומשנה את תכנו:

```
#include <stdio.h>
void convert(char *p)
{
        *p      = 'A';
```

```
        *(p+1) = 'h';
        *(p+2) = 'l';
        *(p+3) = 'a';
        *(p+4) = 'n';
}
```

הפונקציה המשתמשת מעביר ל- ()convert מערך תווים:

```
void main()
{
        char s[] = {'H', 'e', 'l', 'l', 'o'};
        int i;

        convert(s);
        for(i=0; i<5; i++)
                putchar(s[i]);
}
```

הפלט:

```
Ahlan
```

מכיוון שמערך הוא מצביע, שינוי איבריו בפונקציה הנקראת משפיע אצל הפונקציה הקוראת.

דוגמאות נוספות מובאות בעמ' 202.

## מצביעים והקצאות זיכרון

כאשר מגדירים מערך, בדרך כלל מקצים לאיבריו מקום בזיכרון. לדוגמא ההגדרה

*char s[5];*

או לחילופין,

*char s[] = {'H', 'e', 'l', 'l', 'o'};*

יוצרת מערך ומקצה 5 עמדות בזיכרון עבור איבריו. לעומת זאת אם נגדיר מצביע

*char *s;*

לא מוקצות עמדות זיכרון לאיברים כלשהם.

דוגמא לבעיה:

```
#include <stdio.h>

void main()
{
	char s[5];
	char *p;

	int i;

	/* read 5 chars into the array pointed to by s */
	for(i=0; i<5; i++)
		s[i] = getchar();   /* OK */

	/* read 5 chars into the array pointed to by p */
	for(i=0; i<5; i++)
		p[i] = getchar();
			/* Runtime error!!  no memory allocated for the elements - memory access
			violation */
}
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

תמונת הזיכרון:

| **כתובת** | **ערך** |
|---|---|
| | |
| **p=839642** | |
| | |
| **s=839644** | s[0] |
| | s[1] |
| | s[2] |
| | s[3] |
| | s[4] |

**הבעיה**: כאשר התכנית מנסה לגשת למקום p[i] , היא חורגת אל תא זיכרון שאינו שמור עבור p, וזה עלול לפגוע בתוכן של משתנים אחרים בתכנית. לדוגמא ניסיון לכתוב

*p[3] = 'r';*

ישנה (בהנחת תמונת הזיכרון הנ"ל) את ערכו של s[1] ל- 'r' !!! אנו נראה בפרק 12 כיצד ניתן להקצות מקום בזיכרון - באופן מפורש - עבור מצביעים.

<u>הקבוע NULL</u>

בשפת C מוגדר קבוע בשם NULL בעל ערך 0 , שבהצבתו למצביע מציין "שום מקום". לדוגמא ההוראה

*int  *p = NULL;*

מגדירה מצביע בשם p ומאתחלת אותו להצביע על "שום מקום", כלומר ערכו הוא 0. ללא אתחול זה ערכו של p איננו מוגדר, כלומר הוא מכיל "זבל".

אנו נראה בפרק 12, "**הקצאת זיכרון דינמית ורשימות מקושרות**", שימושים להצבות NULL למצביעים.

הקבוע NULL מוגדר בקבצי הממשק של הספריות התקניות stdio.h, stdlib.h.

## האם קיים בכל זאת הבדל בין מצביע ומערך?

מצביע ומערך הם כמעט זהים בתפקידם, להוציא הבדל אחד: מצביע הוא מקום בזיכרון שניתן להתייחס אליו.

למשל, ניתן לשנות את ערכו (כך שיצביע על ערך אחר) וכן לקבל את כתובתו (מצביע למצביע). מערך לעומת זאת, אינו תא פיזי בזיכרון, אלא התייחסות **שמית** לכתובת תחילת איברי המערך.

לכן לא ניתן לשנות את ערכו, וגם לא לקבל את כתובתו.

התכנית והסברה בעמ' 204-205 ממחישים את ההבדל בין מצביע למערך.

## תרגיל

***בצע/י את התרגיל בעמ' 205-206.***

# פעולות חשבוניות על מצביעים

פעולות חשבוניות על מצביע מבוצעות בהתאם לטיפוסו, כלומר בהתאם לטיפוס הנתון עליו הוא מצביע. ככלל, מצביע **ptr** לטיפוס **type**

***<type>* *<ptr>;***

מקודם עפ״י גודל הטיפוס type. למשל אם p מוגדר כמצביע לממשי (* float) אזי הביטוי

**p++**

יגרום לקידומו של p ב- 4 בתים, במערכת בה ממשי הוא בגודל 4. באופן כללי מתקיים

ptr + x **שקול ל-** ptr + x*sizeof(type)

ptr - x **שקול ל-** ptr - x*sizeof(type)

לדוגמא, נניח שמוגדרים המצביעים הבאים:

```
char    *pc;
int     *pi;
double  *pd;
float   array[5];
```

ובהנחה שנתונים גדלי הטיפוסים

sizeof(char) = 1

sizeof(int) = 4

sizeof(float) = 4

sizeof(double) = 8

הביטויים הבאים שקולים:

| **פירוש בחשבון של בתים** | **ביטוי** |
|---|---|
| pc = pc + 1 | pc++ |
| pi  + 2*4 | pi + 2 |
| pd = pd - 8 | pd-- |
| pc - 4 | pc - 4 |
| array + 3*4 | array + 3 |

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# מצביעים לפונקציות

שפת C מאפשרת להגדיר מצביעים לפונקציות ובכך מספקת יכולת תמרון גבוהה בהפעלתן. התחביר בהגדרת מצביעים לפונקציות הוא מעט מורכב ומבלבל.

למעשה, שמה של פונקציה הוא גם מצביע אליה (בדומה לכך ששם מערך הוא מצביע לתחילתו).

לדוגמא, נתונה הפונקציה

```
int func1(int x)
{
	printf("\nfunc1: x=%d",x);
	return x+1;
}
```

הפונקציה מקבלת כפרמטר שלם ומחזירה ערך שלם. func1 הוא שם הפונקציה והוא גם כתובתה. לכן ניתן לקרוא לפונקציה באופן המקובל

```
func1(17)
```

או ע"י כתובתה:

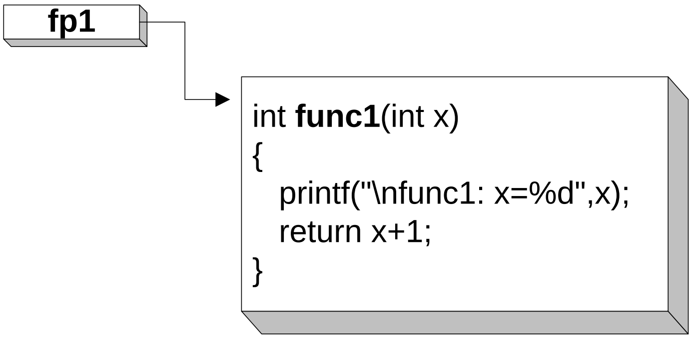
```
(*func1)(17);
```

ניתן להגדיר מצביע לפונקציה באופן הבא:

```
int (*fp1)(int);
```

משמעות ההגדרה היא "fp1 הוא מסוג מצביע לפונקציה המקבלת כפרמטר שלם ומחזירה ערך שלם". כעת ניתן להציב ל- fp1 את כתובת הפונקציה func1 ע"י:

```
fp1 = func1;
```




<u>הסבר</u>: func1 הוא שם הפונקציה ולכן הוא גם כתובתה. לכן ניתן להציבו למצביע מאותו הטיפוס, כלומר למצביע לפונקציה המקבלת כפרמטר שלם ומחזירה שלם.

וכעת ניתן לקרוא לפונקציה ע"י המצביע fp1:

```
printf("\nfunc1 returned %d.", fp1(3));
```

הפלט:

```
func1: x=3
func1 returned 4.
```

הסוגריים סביב המצביע (fp1*) בהגדרת המצביע חשובות, מכיוון שבלעדיהן

*int *__fp1__(int);*

fp1 יוגדר כפונקציה המחזירה מצביע ל- int.

דוגמא נוספת - נגדיר פונקציה שנייה, func2:

```
int func2(int x, int y)
{
	printf("\nfunc2: x=%d y=%d", x, y);
	return x+y;
}
```

נגדיר מצביע לפונקציה ונציב לו את כתובתה:

```
int (*fp2)(int,int);
fp2 = func2;
```

וכעת נקרא לפונקציה ע"י המצביע:

```
printf("\nfunc2 returned %d.", fp2(8,10));
```

והפלט:

```
func2: x=8 y=10
func2 returned 18.
```

## שימושים במצביעים לפונקציות

בעמ' 208-210 מובאת דוגמת שימוש במצביעים לפונקציות באמצעות פונקצית הספרייה qsort().

## הגדרת טיפוס מצביע ומערכי מצביעים לפונקציות

בגלל מורכבות ההכרזות על מצביעים לפונקציות, רצוי להגדיר טיפוס מצביע לפונקציה ע"י **typedef**:

```
typedef int (*FPTR)(int) ;
```

כעת **FPTR** הוא מסוג מצביע לפונקציה המקבלת כפרמטר int ומחזירה int, וניתן להשתמש בו כך:

```
FPTR  fp1 =  func1;
printf("\nfunc1 returned %d.", fp1(3));
```

כמו כן ניתן להגדיר מערך מצביעים לפונקציות ע"י

```
FPTR farr[3];
```

וכעת ניתן להציב ל- farr את כתובתה של func1 ולקרוא לה:

```
farr[0] = func1;
farr[0](34);
```

# סיכום

- **מצביע** הוא משתנה המכיל כתובת של משתנה אחר בזיכרון. ניתן להתייחס למשתנה המוצבע דרך המצביע - לקרוא את ערכו, לשנותו וכו'.
- משמעות האופרטורים * ו- & :
  - הסימן * פירושו "הערך המוצבע על ידי"
  - הסימן & פירושו "כתובתו של".
- אחד השימושים הנפוצים במצביעים בשפת C הוא בהעברת כתובות משתנים כפרמטרים (**העברה ע"י התייחסות**) במקום את הפרמטרים עצמם (**העברה ע"י ערך**). באופן זה יכולה פונקציה **נקראת** לבצע שינוי של משתנה אצל הפונקציה **הקוראת**.
- פעולות חשבוניות על מצביע מבוצעות בהתאם לטיפוסו, כלומר בהתאם לטיפוס הנתון עליו הוא מצביע.
- מערך ומצביע הם בעלי תפקידים כמעט זהים - ניתן להתייחס למצביע כאל מערך ולהפך. ניתן להציב מערך למצביע (אך לא להפך!). ניתן לבצע פעולות על איברי המערך בשתי הצורות - כמערך וכמצביע. כמו כן ניתן להגדיר פרמטר לפונקציה כמערך ובפועל להעביר מצביע ולהפך.
- מצביעים לפונקציות ב- C מאפשרים להעביר פונקציות כפרמטרים לפונקציות אחרות. ניתן להגדיר טיפוס מצביע לפונקציה וכן מערכי מצביעים לפונקציות.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום בעמ' 212.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

# 9. מחרוזות

- הגדרת מחרוזות
- כללים
- קלט / פלט מחרוזות
- פעולות על מחרוזות
- מערכי מחרוזות
- תמיכה בשפות בינלאומיות ו– Unicode
- סיכום
- תרגיל מסכם



מרכז ההדרכה 2000 © כל הזכויות שמורות

# הגדרת מחרוזת

בשפת C לא קיים טיפוס מחרוזת טבעי. מחרוזת מיוצגת ע"י מערך תווים המסתיים בתו מסיים מחרוזת, לדוגמא:

*char home[] = "**www.mh2000.co.il**"; /* home page of this book */*

home היא מחרוזת בת 16 תווים + תו מסיים מחרוזת (בלתי נראה):

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| w | w | w | . | m | h | 2 | 0 | 0 | 0 | . | c | o | . | i | l | \0 |

התו '\0' הוא תו סוף מחרוזת, שערך ה- ASCII שלו הוא 0 (אפס). גודל המערך חייב להיות כמספר התווים המרביים שהמחרוזת אמורה להכיל + 1.

ערך קבוע (ליטרל) של מחרוזת מצוין ע"י גרשיים משני צידיה. דוגמאות:

"hello"

"1"

""

יש לשים לב ש- "1" היא מחרוזת ואילו '1' הוא תו - מלבד העובדה ששני נתונים אלו מאוחסנים בזיכרון בגודל שונה (מדוע?) הם משתייכים לטיפוסים שונים.

המחרוזת האחרונה, "", היא מחרוזת ריקה, כלומר היא מכילה תו יחיד - התו '\0' - תו סוף מחרוזת.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# כללים

- מחרוזת היא מערך לכל דבר - ניתן לאתחל אותה ע"י ציון גודל המערך, או בלעדיו:

```
1. char str[] = "hello";
2. char str[6] = "hello";
```

כמו כן, ניתן להתייחס אליה כאל מצביע, לדוגמא:

```
char *str = "hello";
```

- לא ניתן להציב מחרוזת לאחר ההגדרה, לדוגמא:

```
char str[6];
str = "hello";   /* Error! */
```

כמו כן לא ניתן להציב מחרוזת אחת לשנייה, לדוגמא:

```
char str1[6] = "hello";
char str2[6];
str2 = str1;         /* Error! */
```

כפי שנראה בהמשך, קיימת פונקציה המטפלת בהעתקת מחרוזות.

- לא ניתן לבדוק שוויון מחרוזות ע"י האופרטור "==". לדוגמא

```
if(str1 == "hello")  /* Error */
        ...
```

הוא ביטוי לא חוקי. גם לכך קיימת פונקצית ספריה שנכיר בהמשך.

- חריגה מגבולות המערך תגרום לשגיאה בזמן ריצה - המהדר לא יזהה את החריגה בזמן ההידור ולא יודיע על שגיאה, לדוגמא:

```
char str[6];

    ...

str[0] = 'h';
str[1] = 'e';
str[2] = 'l';
str[3] = 'l';
str[4] = 'o';
str[5] = '\0';
str[6] = 'x';         /* Runtime error: Array out-of-range ! */
```

- ערך ה- ASCII של התו '\0' הוא 0 (אפס). לכן שתי ההוראות הבאות שקולות:

```
1. str[5] = '\0';
2. str[5] = 0;
```

- כפי שכבר ראינו, הקבוע NULL מהווה שם חליפי ל- 0 (אפס). באתחול, ניתן להציבו למצביע, בכדי לציין שהוא כרגע "אינו מצביע על אף נתון":

```
char *p = NULL;
```

- בתכניות העושות שימוש עתיר במחרוזות, כדאי להגדיר טיפוס מחרוזת גדול מספיק, כפי

שכבר ראינו, ע"י typedef:

```
typedef char String[256];
```

כעת ניתן להגדיר משתני מחרוזת מסוג String בצורה נוחה:

```
String str1 = "hello";
```

החסרון בשיטה זו הוא בזבוז המקום בזיכרון במקרה של מחרוזות קטנות, אולם בתכניות רבות שיקול זה זניח ביחס לכמות הזכרון במחשב וביחס לנוחות שבהגדרה זו.

- ניתן לשרשר מספר ליטרלים של מחרוזות - המהדר יצרף אותם למחרוזת בודדת. דוגמאות:

```
1) char * str =  "first " "second " "third";/* str="first second third" */
2) printf("hello " "world"); /* output: hello world */
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

# קלט / פלט מחרוזות

ראינו שניתן לקרוא ולהדפיס מחרוזות ע"י הפונקציות printf ו- scanf באמצעות מציין הטיפוס %s. לדוגמא, בכדי להדפיס את המחרוזת שהגדרנו קודם

*char str1[6] = "hello";*

נבצע

*printf("The string is %s", str1);*

יודפס:

```
The string is hello
```

כמו כן, ניתן להדפיס את המחרוזת כמחרוזת הבקרה:

*printf(str1);*

יודפס:

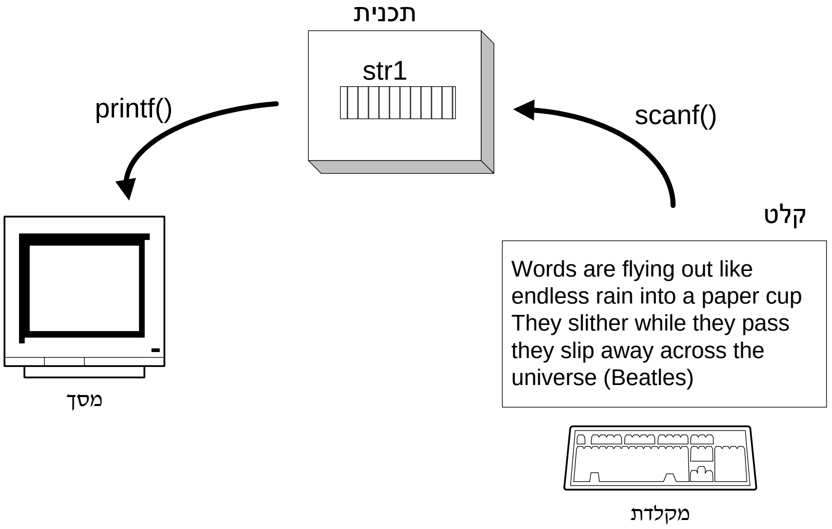
```
hello
```

באופן דומה, scanf מבצעת קריאה מהקלט של מחרוזת:

*char str1[6];*
*scanf("%s", str1);*

כפי שכבר ראינו, scanf - בשונה מ- printf - מקבלת ברשימת הפרמטרים מצביעים למשתנים. במקרה של מחרוזת, הואיל והיא בעצם מערך, שמה הוא כתובתה.

scanf() ו- printf() מבצעות קלט/פלט מחרוזות מהקלט התקני לפלט התקני:



## הפונקציות sscanf ו- sprintf

עד עתה ביצענו פעולות קלט / פלט תקני - מקור הקלט היה המקלדת, ופלט התכנית היה המסך.

בשפת C קיימת אפשרות להתייחס למחרוזות כאל מקורות קלט ויעדי פלט ע"י שימוש בפונקציות **sscanf** ו- **sprintf**.

הפונקציה **sscanf** משמשת לקריאת קלט ממחרוזת עפ"י פורמט נתון. לדוגמא:

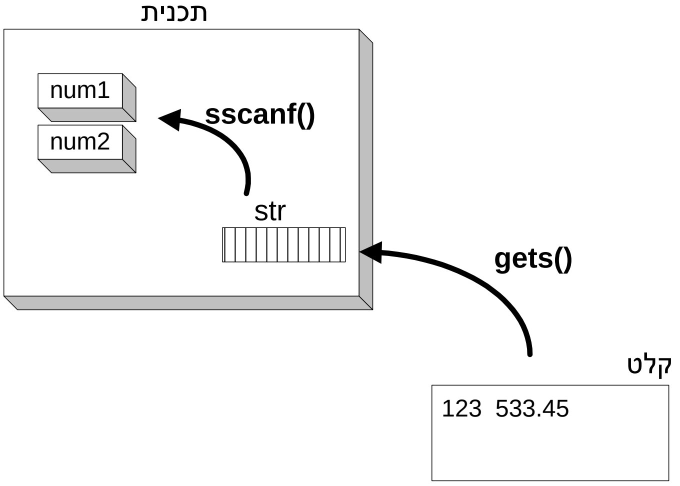
```
typedef char String[256];
String  str;
int    num1;
float num2;

gets(str);  /* read the whole line as string*/
sscanf(str, "%d %f", &num1, &num2);  /* read numbers from string */
printf("num1=%d, num2=%.2f", num1, num2);
```

הסבר: פעולת הקריאה ממחרוזת מבוצעת ע"י sscanf עפ"י התחביר

*sscanf( <string> , <format>, parameter1, parameter2, ...);*

הקריאה מבוצעת בדומה לקריאת קלט ע"י scanf ולפי כללי מחרוזת הפורמט ורשימת הפרמטרים שהכרנו:

*sscanf(str, "%d %f", &num1, &num2);*




בדומה, ניתן לכתוב משתנים עפ"י פורמט לתוך מחרוזת ע"י **sprintf** :

```
String  str;
int    num1 = 3;
float num2 = 54.8;

sprintf(str, "num1=%d, num2=%.2f", num1, num2);
puts(str);
```

יודפס:

```
num1=3, num2=54.80
```

*הערה: גם ()sscanf, בדומה ל- ()scanf, מחזירה שלם המציין את מספר הפרמטרים שנקראו נכונה. ניתן להשתמש בערך מוחזר זה לבדיקת הצלחת פעולת הקלט.*


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# פעולות על מחרוזות

כפי שכבר ראינו, מחרוזת בשפת C היא מערך, ולא טיפוס בסיסי. לכן אין אפשרות לבצע עליה פעולות כגון:

- העתקה/הצבה: str1=str2
- השוואה: if (str1==str2)

לצורך ביצוע הפעולות הנ"ל קיימות פונקציות בספרייה string.h.

כדי להבין איך הפונקציות עובדות, ננסה תחילה לחקות את פעולתן. בכל הדוגמאות, נתייחס לטיפוס מחרוזת **String** המוגדר כמו קודם:

*typedef char **String**[256];*

בעמודים 220-226 מובאות פונקציות חיקוי לפונקציות הספרייה:

- דוגמא 1: חיקוי פעולת strcpy
- דוגמא 2: חיקוי פעולת gets ו- puts
- דוגמא 3: השוואה בין מחרוזות
- דוגמא 4: השוואה ללא התחשבות בגדלי האותיות האנגליות (Case Ignored)

עיין/י בספר בסעיף זה ובצע/י את התרגילים המצורפים.

## פונקציות הספרייה string.h

הפונקציות שממשנו לעיל הן פונקציות תקניות הקיימות בספרייה string.h של שפת C. הספרייה כוללת פונקציות רבות לטיפול במחרוזות - השכיחות שבהן:

- **strcmp** השוואה בין 2 מחרוזות:

***int strcmp(const char *s1, const char *s2);***

- **stricmp** כמו strcmp אך ללא התחשבות בהבדלי אותיות גדולות/קטנות:

***int stricmp(const char *s1, const char *s2);***

- **strcpy** העתקת מחרוזת אחת למחרוזת שנייה:

***char * strcpy(char *s1, const char *s2);***

- **strlen** החזרת אורך מחרוזת:

***size_t strlen(const char *s);***

size_t הוא גודל טיפוס המוגדר ע"י typedef כשלם חיובי.

- **strcat** שרשור מחרוזת אחת למחרוזת שנייה :

***char * strcat(char *dest, const char *src);***

לפונקציות **strcmp**, **strcpy** ו- **strcat** יש פונקציות מקבילות הפועלות על מספר נתון של תווים: **strncmp** משווה בין 2 מחרוזות מספר נתון של תווים לכל היותר, **strncpy** מעתיקה מספר נתון של תווים לכל היותר ממחרוזת אחת לשנייה, ו- **strncat** משרשרת מספר נתון של תווים לכל היותר ממחרוזת אחת לשנייה.

<u>תכנית דוגמא: העתקה ושרשור</u>

הפונקציה ()strcat מקבלת כפרמטרים שתי מחרוזות str1,str2 ומוסיפה את התווים של מחרוזת str2 לקצה המחרוזת str1 , ותו סיום המחרוזת עובר לקצה המחרוזת החדשה שנוצרה.

לדוגמא, אם נתונות שתי המחרוזות:

str1: H e l l o ' ' '\0'

str2: w o r l d '\0'

לאחר קריאה לפונקציה strcat(str1, str2) יהיה ערכה של str1:

str1: H e l l o ' ' w o r l d '\0'

הערות:

– str2 לא משתנה כתוצאה מהפעולה.

– המערך str1 צריך להיות גדול מספיק כדי לקלוט את תווי str2 לתוכו - באחריות הפונקציה הקוראת.

התכנית הבאה מבצעת העתקה ושרשור של מחרוזות ע"י פונקציות הספרייה strcpy ו- strcat :

```
/* file: concat.c */
#include <string.h>
#include <stdio.h>

typedef char String[256];

void main( void )
{
  String str;
  strcpy( str, "Hello " );
  strcat( str, "from " );
  strcat( str, "www.MH2000.co.il !" );  /* home page of this book */
  puts(str);
}
```

והפלט:

```
Hello from www.MH2000.co.il !
```

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## פונקציות לפעולות חיפוש טקסט

קיימות פונקציות ספרייה נוספות ב- string.h לביצוע פעולות חיפוש בטקסט:

**strchr** מציאת מיקום של המופע הראשון של תו מסוים במחרוזת:

***char * strchr(const char *s, int c);***

**strrchr** מציאת מיקום של תו מסוים במחרוזת החל מהסוף (reversed):

***char * strrchr(const char *s, int c);***

**strstr** מציאת מיקום של תת-מחרוזת במחרוזת:

***char * strstr(const char *s, const char *sub_str);***

**strtok** מציאת מילה במחרוזת, עפ"י תווי פיסוק נתונים. תווי הפיסוק מוחלפים בתו מסיים מחרוזת, והמילה מוחזרת:

***char * strtok(char *str, const char *delimit);***

**strspn** פועלת כמו strtok, אך לא מכניסה סימן סוף מחרוזת. במקום זאת, היא מחזירה את אינדקס התו הבא שלאחר סימני הפיסוק:

***size_t strspn(const char *str, const char *delimit);***

**strpbrk** מקבלת כפרמטרים מחרוזת ותת-מחרוזת. מחזירה מצביע למופע הראשון של תו כלשהו מתת-המחרוזת במחרוזת, ואם לא נמצא מופע כזה מוחזר NULL:

***char * strpbrk(const char *str, const char *sub_str);***

<u>תכנית דוגמא: ניתוח טקסט ע"י strtok()</u>

התכנית הבאה מנתחת ומפרקת שורת טקסט על פי סימני פיסוק תוך שימוש בפונקציה strtok המוכרזת כך:

**char *strtok( char ****str***, const char ****delimit*** );**

הפרמטר הראשון הוא מחרוזת והפרמטר השני הוא אוסף תווים (מחרוזת) המהווים סימני פיסוק במחרוזת הראשונה. פונקציה זו פועלת בשני אופנים:

– בפעם הראשונה שהיא נקראת מעבירים לה את שני הפרמטרים. היא מחפשת את תו הפיסוק הראשון במחרוזת, מחליפה אותו בסימן סוף מחרוזת ('\0') ומחזירה מצביע לתת-המחרוזת שנוצרה.

– בפעמים הבאות ש- strtok נקראת, הפרמטר הראשון הוא NULL והיא ממשיכה לחפש סימני פיסוק מהמקום האחרון בו הפסיקה. היא מחזירה שוב מצביע לתת-המחרוזת החדשה שנוצרה (אם הייתה כזו).

מרכז ההדרכה 2000 © כל הזכויות שמורות www.mh2000.co.il

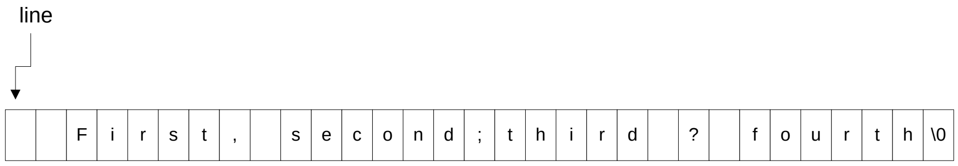
קוד התכנית:

```
/* file: strtok.c */
#include <stdio.h>
#include <string.h>

typedef char String[256];


void main()
{
	String line = "  First, second;third ? fourth";

	char *ptr = strtok(line, " ,;?");

	while(ptr != NULL)
	{
		printf("%s\n", ptr);
		ptr = strtok(NULL, " ,;?");
	}

}
```

פלט התכנית:

```
First
second
third
fourth
```

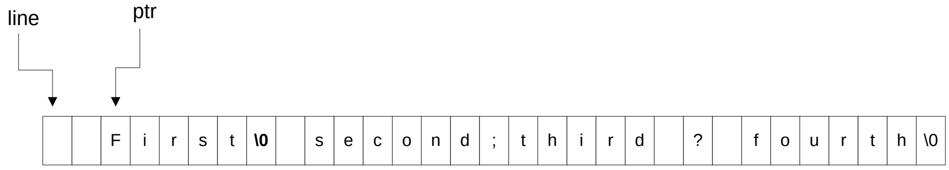
הסבר: השורה line נראית כך בזיכרון לפני הניתוח -




בפעם הראשונה נקראת הפונקציה strtok עם הפרמטר line ומחרוזת תווי הפיסוק:

```
char *ptr = strtok(line, " ,;?");
```

המצביע המוחזר ע"י strtok ומוצב ל- ptr הוא מצביע למילה הראשונה שהתגלתה עפ"י סימני הפיסוק:





*מרכז ההדרכה 2000 © כל הזכויות שמורות*

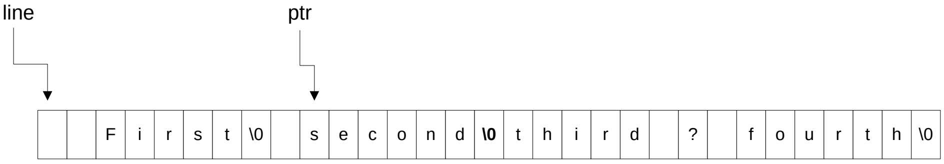
strtok() החליפה את התו "," בתו מסיים המחרוזת. לכן בהדפסת ptr מודפס

```
First
```

בקריאה הבאה ל- strtok() (בתוך הלולאה) מועבר כפרמטר ראשון NULL המציין עבורה המשך סריקה של המחרוזת האחרונה שהועברה לה (line):

*ptr = strtok(NULL, " ,;?");*

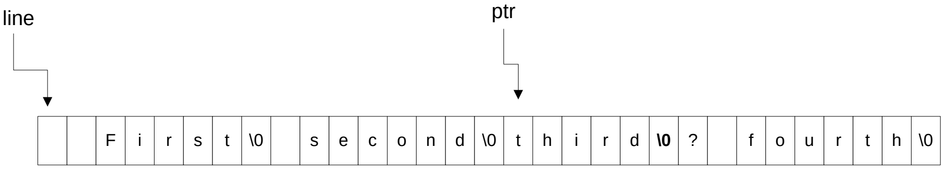
הפונקציה מחזירה שוב מצביע לתת-המחרוזת שנמצאה, "second":



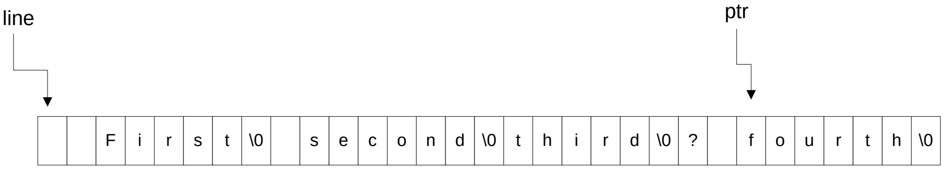

ולכן מודפס

```
second
```

באופן דומה, לאחר הקריאה הבאה ל- strtok() מוחזרת תת-המחרוזת "third"




ומודפסת. לבסוף מוחזרת ומודפסת תת-המחרוזת "fourth":




בקריאה הבאה ל- strtok() מוחזר הערך NULL, ולכן הלולאה מסתיימת:

*while(ptr != NULL)*

הערות:

1. יש לשים לב ש- strtok() משנה את המחרוזת המקורית המועברת לה ע"י החלפת סימן הפיסוק המתאים בתו מסיים מחרוזת.

2. לצורך שמירת המקום האחרון בו הפסיקה, strtok() מגדירה משתנה מקומי סטטי ש"זוכר"


מרכז ההדרכה 2000 © כל הזכויות שמורות

את כתובת התו האחרון שנסרק במחרוזת.

3. במידה ויש מספר תווי פיסוק עוקבים, ()strtok מזהה זאת ומדלגת עליהם, עד למציאת תת-מחרוזת המכילה לפחות תו אחד שאינו תו פיסוק.

## פונקציות לבדיקת סוג התו

קימות פונקציות לבדיקת סוג התו המוגדרות בקובץ ctype.h. הפונקציות מקבלות כפרמטר תו ומחזירות ערך בוליאני המציין האם התו שייך לקטגוריה מסוימת.

לדוגמא, הפונקציה ()isalpha מחזירה ערך "אמת" (שונה מ- 0) אם תו נתון הוא אות אנגלית ו- 0 אחרת. היא מוכרזת כך:

```
int isalpha( int c );
```

שימוש לדוגמא בפונקציה:

```
int ch = getchar();

if(isalpha(ch))
	printf("The char %c is a letter from a-z or A-Z", ch);
```

הפונקציה ()isupper בודקת ומחזירה ערך אמת (שונה מ- 0) אם התו הוא אות אנגלית גדולה:

```
int ch = getchar();
if( isalpha(ch))
	if( isupper(ch))
		printf("The char %c is a letter from A-Z", ch);
	else
		printf("The char %c is a letter from a-z", ch);
```

***רשימת הפונקציות המלאה לבדיקת סוג תו מופיעה בטבלה בעמ' 231.***

## תרגילים

***קרא/י סעיף זה בספר ובצע/י את תר' 4-1 שבעמ' 232.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# מערכי מחרוזות

מערך מחרוזות הוא מערך שכל כניסה בו היא מחרוזת. מגדירים מערך מחרוזות כך:

```
char strings[6][15];
```

או ע"י שימוש בטיפוס מחרוזת המוגדר ע"י typedef:

```
typedef char String[15];
String strings[6];
```

זוהי הגדרה של מערך בן 6 מחרוזות שארכה של כל אחת בו היא עד 15 תווים. ניתן למשל להעתיק מחרוזות לאיברי המערך ע"י strcpy

```
strcpy(strings[0], "first");
strcpy(strings[1], "second");
strcpy(strings[2], strings[0]);
```

ולשרשר מחרוזות אחת לשנייה:

```
strcat(strings[2], strings[1]);
```

תרשים המערך:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | f | i | r | s | t | \0 | | | | | | | | | |
| 1 | s | e | c | o | n | d | \0 | | | | | | | | |
| 2 | f | i | r | s | t | s | e | c | o | n | d | \0 | | | |
| 3 | ... | | | | | | | | | | | | | | |
| 4 | ... | | | | | | | | | | | | | | |
| 5 | ... | | | | | | | | | | | | | | |

***תכנית הדוגמא שבעמ' 233 קוראת מהקלט 8 שורות לתוך מערך מחרוזות ולאחר מכן מדפיסה אותן לפלט.***


מרכז ההדרכה 2000 © כל הזכויות שמורות

## מערך מצביעים לעומת מערך של מערכים

מבחינים בין שני סוגי מערכי מחרוזות:

- מערך מחרוזות המיוצגות ע"י מערכים:

*char colors1[8][15] = {"red", "black", "white", "purple", "yellow", "pink", "green", "blue" };*

בהגדרה כזו מוקצה עבור כל מחרוזת זיכרון עבור מלוא המערך, גם אם רק חלקו מנוצל. תמונת מערך המחרוזות בזיכרון:

**colors1**[]

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | r | e | d | \0 | | | | | | | | | | | |
| 1 | b | l | a | c | k | \0 | | | | | | | | | |
| 2 | w | h | i | t | e | \0 | | | | | | | | | |
| 3 | p | u | r | p | l | e | \0 | | | | | | | | |
| 4 | y | e | l | l | o | w | \0 | | | | | | | | |
| 5 | p | i | n | k | \0 | | | | | | | | | | |
| 6 | g | r | e | e | n | \0 | | | | | | | | | |
| 7 | b | l | u | e | \0 | | | | | | | | | | |

- מערך מחרוזות המיוצגות ע"י מצביעים:

*char* colors2[8] = {"red", "black", "white", "purple", "yellow", "pink", "green", "blue" };*


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

במקרה זה המימד השני של מערך התווים הדו-ממדי הוא אינו קבוע: לכל מחרוזת מוקצה מקום בהתאם לארכה:

**colors2**[]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | r | e | d | \0 | | | |
| 1 | b | l | a | c | k | \0 | |
| 2 | w | h | i | t | e | \0 | |
| 3 | p | u | r | p | l | e | \0 |
| 4 | y | e | l | l | o | w | \0 |
| 5 | p | i | n | k | \0 | | |
| 6 | g | r | e | e | n | \0 | |
| 7 | b | l | u | e | \0 | | |

כפי שניתן לראות, המערך הראשון הוא בזבזני יותר הן מבחינת הקצאת הזיכרון והן מבחינת אתחול המערך הכולל (מקומות ריקים במערך מאותחלים ל- 0).

איזו שיטה עדיפה? רצוי להשתמש במערכי מערכים בייצוג של טקסט קבוע, כשלא אמורים לשנות את המחרוזות או את מיקומן.

לעומת זאת, כאשר נדרשות פעולות מניפולציה על המחרוזות - כגון מיון - רצוי להגדיר מערכי מצביעים.

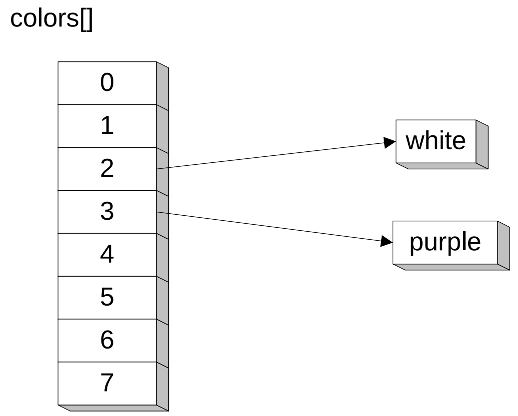
הבעיה בשיטה 2 היא שכרגע איננו יודעים להקצות זיכרון מפורש עבור מחרוזת - דבר שנכיר רק בפרק 12. לעת עתה, ניתן להשתמש בשיטת מערכי מצביעים רק עבור מחרוזות המאותחלות בזמן הגדרתן.

## מיון מערך מחרוזות

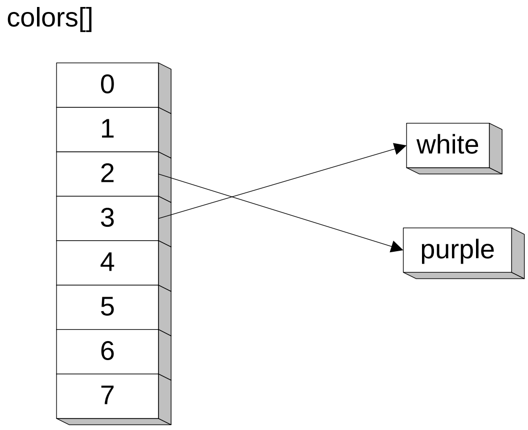
נכתוב תכנית שממיינת מערך מחרוזות. מערך המחרוזות יוגדר כמערך של מצביעים למחרוזת, תוך שימוש בטיפוס String המוגדר הפעם כ- * char.

נשתמש בשיטת "מיון בועות" שראינו בפרק 7, "**מערכים**".

מיון המערך מתבצע כך שהמצביעים למחרוזות מוחלפים. לדוגמא, אם שתי המחרוזות "white" ו- "purple" ממוקמות כך לפני ההחלפה ביניהן :




לאחר ההחלפה המערך ייראה כך :




***קוד התכנית מובא בעמ' 235-236.***

## פרמטרים לתכנית: argc, argv ו- env

תכניות רבות מאפשרות למשתמש להפעיל אותן עם אופציות שונות. לדוגמא, תכנית המבצעת העתקת קובץ אחד לשני מאפשרת להעביר את שמות הקבצים כפרמטרים:

– במערכות ההפעלה Windows/Dos:

```
copy file1 file2
```

– ב- Unix:

```
cp file1 file2
```

יתרה מזאת, הפונקציות מאפשרות להוסיף מציינים נוספים כדי לבחור את אופן הפעולה.

לדוגמא, בתכנית ההעתקה הנ"ל, אם קובץ המקור הוא ספרייה קיימת אופציה להעתקת תתי-הספריות באופן רקורסיבי.

כיצד ניתן להעביר פרמטרים לתכנית בשפת C? הפונקציה **main**, שעד כה הגדרנו אותה כפונקציה שלא מקבלת פרמטרים, יכולה להיות מוכרזת גם כך:

***int main( int argc, char *argv[ ]);***

הפונקציה מקבלת 2 פרמטרים:

**argc**: מספר הפרמטרים המועברים

**argv**: מערך של מחרוזות הפרמטרים

המחרוזת הראשונה ב- argv היא שם התכנית. לדוגמא, אם שם התכנית הוא prog.c, והיא הודרה לקובץ ביצוע בשם prog.exe, הפעלתה עם הפרמטרים הבאים

```
prog  hello 22 0.5
```

תגרום לכך ש- argc ו- argv ייראו כך:

**argc = 4**

**argv[] =**

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | p | r | o | g | '\0' | |
| 1 | h | e | l | l | o | '\0' |
| 2 | 2 | 2 | '\0' | | | |
| 3 | 0 | . | 5 | '\0' | | |

כלומר, הפרמטרים מועברים לתכנית כמחרוזות. אם התכנית מעוניינת לשלוף את המספרים

מהמחרוזות היא יכולה לבצע זאת ע"י sscanf, לדוגמא:

```
sscanf(argv[2], "%d" , &num1);  /* num1 = 22 */
sscanf(argv[3], "%f" , &num2);  /* num2 = 0.5 */
```

## תכנית דוגמא: FTP

בעמ' 239 מובאת כדוגמא תכנית **FTP** (File Transfer Protocol) שהיא תכנית נפוצה להעברת קבצים בין מחשבים מרוחקים דרך האינטרנט. התכנית מופעלת ב- 2 צורות עיקריות:

- מקבלת כפרמטר את כתובת ה- FTP של השרת ומנסה להתחבר אליו
- לא מקבלת אף פרמטר, מחכה לפקודת open מהמשתמש

תרגול

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 241.***

# ***תמיכה בשפות בינלאומיות ו- Unicode***

***עיין/י בסעיף זה להרחבה בנושא Unicode ותמיכה בשפות בינלאומיות.***

# סיכום

- בשפת C לא קיים טיפוס בסיסי מסוג מחרוזת. מחרוזת מיוצגת ע"י מערך תווים המסתיים בתו מיוחד ('**\0**').
- קלט/פלט מחרוזות בנות מילה בודדת מבוצע ע"י הפונקציות **scanf** ו- **printf** עם מציין הטיפוס **%s**. קלט/פלט של מחרוזת שורה שלמה מבוצע ע"י הפונקציות **gets** ו- **puts**.
- ניתן לבצע פעולות קלט פלט על מחרוזות כאילו היו מקור קלט או יעד הפלט. הפונקציות המשמשות לכך הן **sscanf** ו- **sprintf**.
- מערך מחרוזות הוא מערך שכל כניסה בו היא מערך תווים או מצביע למחרוזת. בשיטה הראשונה המחרוזות במערך הן בעלות אורך זהה. בשיטה השנייה לכל מחרוזת אורך עפ"י איתחולה.
- הספרייה string.h כוללת פונקציות שונות לטיפול במחרוזות, כגון: העתקה, שירשור, קבלת אורך מחרוזת, חיפוש תווים ותת-מחרוזת במחרוזת ועוד.
- ניתן להגדיר את הפונקציה **main()** כך שתקבל פרמטרים מהמשתמש דרך מערכת ההפעלה:

***int main( int argc, char *argv[ ], char *env[]);***

**argc**: מספר הפרמטרים המועברים

**argv**: מערך של מחרוזות הפרמטרים

**env**: מערך פרמטרי סביבה במערכת ההפעלה

- לצורך תמיכה בשפות מורחבות, הוגדר תקן **Unicode** שבו כל תו מיוצג ע"י 16 סיביות, במקום 8. שפת C כוללת תמיכה ב**תווים רחבים** (Wide Chars): קיים טיפוס תו רחב **wchar_t** באורך 16 סיביות. כמו כן כוללת הספרייה התקנית גרסאות "תו-רחב" לפונקציות לטיפול בתווים ובמחרוזות.

# תרגיל מסכם

***בצע/י את התרגיל המסכם שבעמ' 245.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# 10. מבנים

- הגדרת מבנה והתייחסות לשדותיו
- מערכי מבנים
- מצביעים למבנים
- העברת מבנים כפרמטרים לפונקציות
- פעולות על מבנים במערך
- union
- סיכום
- תרגילי סיכום


מרכז ההדרכה 2000 © כל הזכויות שמורות

# הגדרת מבנה והתייחסות לשדותיו

הכרנו עד עתה משתנים מטיפוס נתונים יחיד: שלם, ממשי, תו או מערך מסוג מסוים. לדוגמא:

```
int    x;
float  y;
char   str[20];
```

**מבנה (structure)** ב - C מאפשר הגדרת אוסף משתנים מטיפוסים שונים כיחידה אחת. מבנה נקרא לפעמים גם **רשומה (record)**.

לדוגמא, נניח שאנו רוצים לכתוב תוכנה לניהול אוסף תקליטורי מוזיקה. לכל תקליט נרצה לשמור את המשתנים הבאים:

| <u>**שם השדה**</u> | <u>**הטיפוס המייצג**</u> |
|---|---|
| שם התקליט | char name[20] |
| שם הלהקה | char band[40] |
| סוג המוסיקה | char category[12] |
| מחיר | float cost |
| מספר סידורי | int number |

מבנה ב- C מוגדר ע"י המילה השמורה **struct**. קיימות שתי דרכים להגדיר טיפוס מבנה:

- הגדרת טיפוס מבנה ע"י typedef
- הגדרת מבנה כשם תג

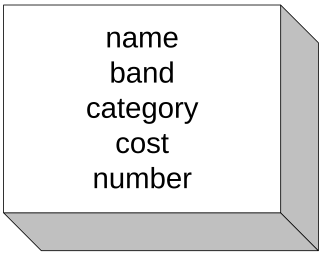
## דרך א': הגדרת מבנה כטיפוס

בשיטה זו, הכרזה על מבנה נעשית תוך שימוש בהוראה typedef :

```
typedef struct
{
        char  name[20];
        char  band [40];
        char  category[12];
        float  cost;
        int     number;
} CD;
```

ההגדרה יוצרת טיפוס נתונים חדש CD המכיל הגדרות משתנים מטיפוסים שונים. משתנים אלו נקראים **שדות** המבנה. הם מאוגדים יחדיו בבלוק זכרון אחד :

**CD**




המבנה התחבירי בהגדרת מבנה בשיטה זו :

```
typedef struct
{
    <טיפוס>  <שם משתנה>;
    <טיפוס>  <שם משתנה>;
         :
} <שם-טיפוס>;
```

כעת נגדיר משתנים מסוג CD :

```
CD  disc1, disc2;
```

disc1 ו- disc2 הם מבנים מהטיפוס CD שהגדרנו לעיל. בדומה למערך, ניתן לאתחל מבנה בזמן הגדרתו, לדוגמא :

```
CD cd1 = {"Best Hits", "Tiny Tim", "pop music",12.5,12};
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## דרך ב': הגדרת מבנה כשם תג

בשיטה זו מגדירים מבנה ללא שימוש ב- typedef. הכרזה על מבנה :

```
struct CD
{
        char  name[20];
        char  band[40];
        char  category[12];
        float  cost;
        int     number;
};
```

המבנה התחבירי בהגדרת מבנה בשיטה זו:

```
struct <שם תג>

{

        <טיפוס>  <שם משתנה>;

        <טיפוס>  <שם משתנה>;

        ...

} ;
```

בשונה מהשיטה הקודמת, בהגדרת משתנים מסוג CD חובה לציין את שם התג, יחד עם המילה השמורה **struct**:

```
struct CD  disc1, disc2;
```

ניתן גם להגדיר משתנים בו זמנית עם הגדרת טיפוס המבנה, לדוגמא:

```
struct CD
{
        char  name[20];
        char  band[40];
        char  category[12];
        float  cost;
        int     number;
} disc1, disc2;
```

וכמו כן ניתן לאתחל מבנה בזמן הגדרתו:

```
struct CD cd1 = {"Best Hits", "Tiny Tim", "pop music",12.5,12};
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## איזו דרך עדיפה?

הדרך הראשונה - הגדרת טיפוס על ידי typedef - עדיפה מ- 2 סיבות:

1. בהעברת המבנה כפרמטר לפונקציה, למהדר קל יותר לבצע בדיקת התאמת טיפוסים כאשר הטיפוס מוגדר ע"י typedef מאשר בלעדיו (מהדרים מסוימים לא יודיעו על שגיאה בשיטה ב').

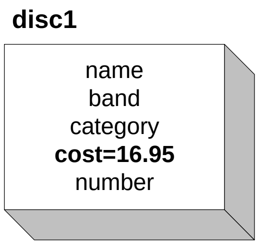
2. נוח יותר להשתמש בטיפוס המוגדר ע"י typedef מאשר לרשום בכל פעם את המילה struct.

אנו נשתמש בדרך א' מעתה ואילך, למעט מקרה מסוים של התייחסות עצמית, שנעסוק בו ברשימות מקושרות בפרק 12, "**הקצאת זיכרון דינמית ורשימות מקושרות**".

## גישה לשדות המבנה

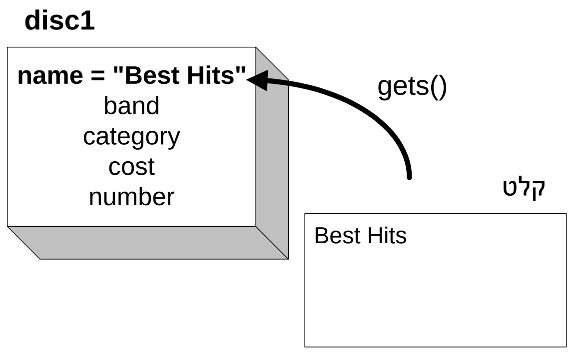
נניח שאנו רוצים לשנות את השדה cost ב- disc1. כדי לגשת לשדה מסוים משתמשים באופרטור "." בין שם המבנה לשם השדה:

*disc1.cost = 16.95;*




דוגמא נוספת - אתחול שם התקליטור מהקלט:

*gets(disc1.name);*




בהגדרת המחרוזות במבנה ניתן להשתמש בטיפוס מחרוזת המוגדר כמערך של 256 תווים:

```
typedef char String[256];

typedef struct
{
	String name;
	String band;
	String category;
	float cost;
	int number;
} CD;
```

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## תכנית דוגמא

קריאת נתוני תקליטור מהקלט למשתנה מסוג תקליטור והדפסתם לפלט:

```
/* file: cd1.c */
#include <stdio.h>
typedef char String[256];

typedef struct
{
	String name;
	String band;
	String category;
	float cost;
	int number;
} CD;

void main()
{
	CD cd1;

	puts("Enter CD name, band and category, separated by newlines:");
	gets(cd1.name);
	gets(cd1.band);
	gets(cd1.category);
	printf("Enter cost:");
	scanf("%f", &cd1.cost);
	cd1.number = 1;

	/* print */
	puts("cd1 = ");
	printf("\tname=%s\n",cd1.name);
	printf("\tband=%s\n",cd1.band);
	printf("\tcategory=%s\n",cd1.category);
	printf("\tcost=%.2f\n",cd1.cost);
	printf("\tnumber=%d\n",cd1.number);
}
```

עבור הקלט

```
Enter CD name, band and category, separated by
newlines:
Changes
David Bowie
pop
Enter cost:23.5
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

יודפס הפלט:

```
cd1 =
    name=Changes
    band=David Bowie
    category=pop
    cost=23.50
    number=1
```

<u>הסבר התכנית</u>

התכנית מגדירה משתנה מסוג רשומת CD

*CD cd1;*

ולאחר מכן מבקשת מהמשתמש להקליד את נתוני התקליטור:

*puts("Enter CD name, band and category, separated by newlines:");*

השדות שמטיפוס מחרוזות נקראים מהקלט ע"י ()gets:

*gets(cd1.name);*
*gets(cd1.band);*
*gets(cd1.category);*

זאת בכדי לאפשר לקרוא יותר ממילה אחת עבור שם התקליטור, שם הלהקה והקטגוריה.

הנתון המספרי, cost, נקרא ע"י ()scanf תוך העברת **כתובת** השדה במבנה, בדומה לקריאת משתנה מספרי רגיל:

*printf("Enter cost:");*
*scanf("%f", **&cd1.cost**);*

המספר הסידורי מאותחל ל- 1:

*cd1.number = 1;*

בשלב הבא מודפס המבנה ע"י :

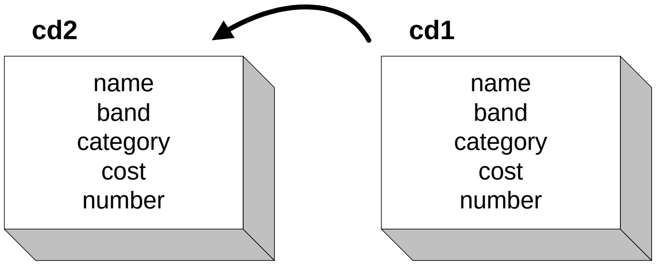
```
/* print */
puts("cd1 = ");
printf("\tname=%s\n",cd1.name);
printf("\tband=%s\n",cd1.band);
printf("\tcategory=%s\n",cd1.category);
printf("\tcost=%.2f\n",cd1.cost);
printf("\tnumber=%d\n",cd1.number);
```

## הצבת מבנה למבנה אחר

ניתן להציב מבנה אחד למבנה אחר, בתנאי שהם מאותו הטיפוס, לדוגמא:

***cd2 = cd1;***

מה שמבוצע במקרה זה הוא העתקת תוכן המבנה הראשון למבנה השני שדה-שדה:




*הערה: יש לשים לב שכאשר המבנה מכיל מצביעים המצביע הוא המועתק, ולא תוכנו.*

***הרחבת הדוגמא הקודמת - עיין/י בתכנית שבעמ' 254-255.***

<u>תרגילים</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-4 שבעמ' 255.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## מבנה המכיל משתנה ממבנה אחר

אחד השדות של מבנה יכול בעצמו להיות טיפוס מבנה אחר. באופן זה ניתן להגדיר היררכיה של טיפוסים מורכבים.

לדוגמא, בתכנית לניהול מוצרים בסופרמרקט, נגדיר מבנה המתאר ספק:

```
typedef struct
{
        char        name[30];
        char        address[50];
        char        phone_num[15];
        char        fax_num[15];
} Supplier ;
```

נגדיר מבנה נוסף המתאר את המלאי עבור מוצר נתון - מקסימום, מינימום וכמות נוכחית:

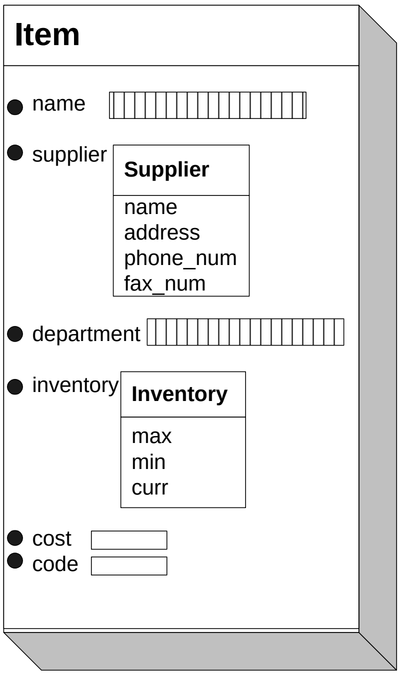
```
typedef struct
{
        int max;
        int min;
        int curr;
} Inventory;
```

כעת נגדיר פריט (Item) - הפריט יתאר סוג של מוצר הקיים בסופרמרקט:

```
typedef struct
{
        char        name[30];
        Supplier    supplier;
        char        department[15];
        Inventory   inventory;
        float       cost;
        int         code;
} Item ;
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

תרשים הזיכרון של רשומת Item :




<u>הסבר</u>: לכל פריט קיימים שדות המתארים את שמו, המחלקה לה הוא משתייך, מחיר וקוד.
בנוסף, שדות מסוימים מוגדרים מטיפוסי המבנים שהוגדרו קודם:

*Supplier supplier;*
*...*
*Inventory inventory;*

> *הערה: יש לשים לב שהטיפוס מוגדר עם אות גדולה בתחילתו, ואילו שם השדה עם אות קטנה. מקובל להבדיל כך בין טיפוסים לבין משתנים.*

***התכנית המשתמשת מובאת בעמ' 257.***

<u>תרגיל</u>

***בצע/י את התרגיל שבעמ' 258.***

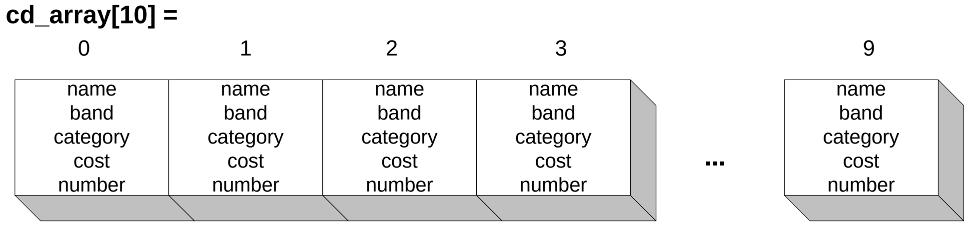
 מרכז ההדרכה 2000 © כל הזכויות שמורות

# מערכי מבנים

כמו בכל טיפוס משתנה אחר, ניתן להגדיר מערך של מבנים. לדוגמא, הגדרת מערך של 10 תקליטורים:

*CD cd_array[10];*

תמונת הזיכרון של המערך:




התייחסות לשדות במבנה הנמצא במערך היא ע"י ציון אינדקס המבנה במערך , בצירוף אופרטור הנקודה ".", לדוגמא:

1. *cd_array[3].cost = 17.2;*

2. *gets(cd_array[6].name);*

***בעמ' 259 מובאת תכנית המרחיבה את תכנית התקליטורים לטיפול במערך תקליטורים (CD). עיין/י בקוד התכנית ובהסבר.***

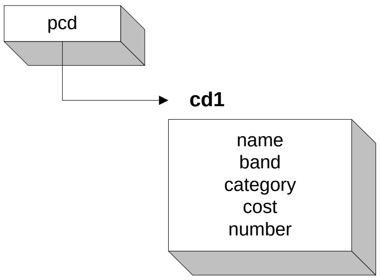
<u>תרגיל</u>

***בצע/י את התרגיל שבעמ' 261.***

# מצביעים למבנים

בדומה לטיפוסים הבסיסיים האחרים, ניתן להגדיר מצביע גם לטיפוס מבנה. לדוגמא:

```
CD  cd1;
CD *pcd = &cd1;
```




ההתייחסות לשדה במבנה בגישה דרך מצביע היא ע״י

```
(*pcd).cost = 4.5;
```

צורה זו היא מסורבלת - הסימן → (צירוף הסימנים ״>״ ו ״-״) משמש כחלופה נוחה לשני האופרטורים ״*״ ו- ״.״ . לדוגמא, במקום הביטוי הנ״ל ניתן לרשום

```
pcd->cost = 4.5;
```

דוגמא לקריאת כל נתוני התקליטור מהקלט:

```
gets(pcd->name);
gets(pcd->band);
gets(pcd->category);
printf("Enter cost:");
scanf("%f", &pcd->cost);
```

כמו בטיפוסים רגילים, קידום מצביע מבוצע עפ״י גודל המבנה. לדוגמא

```
CD  cd;
CD *pcd = &cd;
pcd++;
```

יבוצע קידום של המצביע בגודל של מבנה שלם, כלומר הביטוי ++pcd שקול לביטוי

```
pcd = pcd + sizeof(CD);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

# העברת מבנים כפרמטרים לפונקציות

ראינו כמה מסורבל לקרוא בכל פעם מחדש מבנה מהקלט וכן להדפיסו.

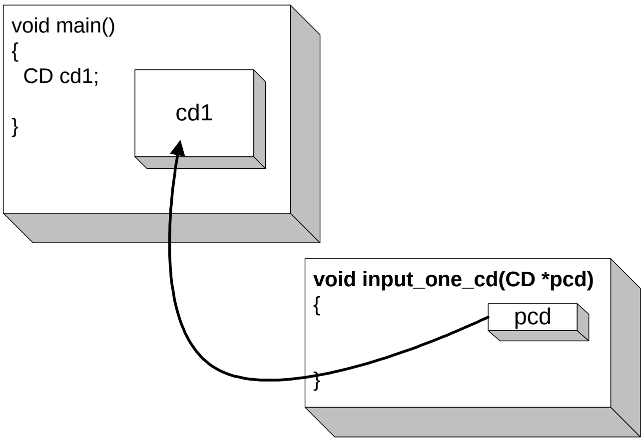
ניתן להגדיר פונקציה המקבלת כפרמטר מבנה ו/או מחזירה אותו כפרמטר, לדוגמא :

```
void output_one_cd(CD cd); /* output a given CD structure */
CD  input_one_cd();            /* initialize CD from input and return it */
```

גישה זו אינה יעילה מכיוון שבהעברת מבנה כפרמטר לפונקציה ובהחזרתו כערך הוא מועתק למחסנית הקריאות, כלומר מספר גדול של בתים - עשרות ואולי מאות - מועתקים בכל העברה כזו.

פתרון עדיף הוא להעביר מצביעים למבנים. ממשק הפונקציות :

```
void output_one_cd(CD *pcd);
void input_one_cd(CD *pcd);
```




בשיטה זו מועתק רק המצביע למבנה (כ- 4 בתים) למחסנית ולא נוצרים עותקים מיותרים של המבנה. הגדרת הפונקציות :

```
void output_one_cd(CD *pcd)
{
	printf("\ncd %d:\n", pcd->number);
	printf("\tname=%s\n",pcd->name);
	printf("\tband=%s\n",pcd->band);
	printf("\tcategory=%s\n",pcd->category);
	printf("\tcost=%.2f\n",pcd->cost);
}

void input_one_cd(CD *pcd)
{
	char temp[80];
	puts("Enter CD name, band and category, separated by newlines:");
	gets(pcd->name);
	gets(pcd->band);
	gets(pcd->category);
```

מרכז ההדרכה 2000 © כל הזכויות שמורות

```
        printf("Enter cost:");
        scanf("%f", &pcd->cost);
        gets(temp); /* to read up to the end of line */
}
```

הפונקציות לקריאה ולכתיבה של מערך תקליטורים שלם קוראות לשתי הפונקציות הנ"ל:

```
void input_cd_array(CD arr[], int array_size)
{
        int i;

        for (i=0; i<array_size; i++)
        {
                input_one_cd(&arr[i]);
                arr[i].number = i+1;
        }
}

void output_cd_array(CD arr[], int array_size)
{
        int i;

        printf("\nArray elements:\n");
        for (i=0; i<array_size; i++)
                output_one_cd(&arr[i]);
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

# ***פעולות על מבנים במערך***

סעיף זה מובאות מספר דוגמאות לפעולות על מבנים במערך:

- פונקציה למציאת מבנה עם ערך מקסימום:

***int find_dearest_cd(CD arr[], int array_size);***

הפונקציה מחפשת ומחזירה את אינדקס התקליטור היקר ביותר:

– משתנה האינדקס מאותחל לאיבר הראשון, 0.

– מתבצעת לולאה על איברי המערך, החל מאיבר 1.

– בכל שלב בו נמצא תקליטור יקר מהקודם, מעודכנים הן הערך המקסימלי והן משתנה האינדקס.

– הפונקציה מחזירה את האינדקס.

- פונקציה לבדיקת קיום של מבנה במערך לפי מספר:

***int exist(CD arr[], int array_size, int num);***

- פונקציה לחיפוש תקליטור עפ"י שמו:

***int find_cd_by_name(CD arr[], int array_size, char name[]);***

- פונקציה למיון מערך המבנים: כדי למיין את איברי המערך נגדיר פונקציות בשם swap ו- sort שתבצענה מיון בשיטת **מיון בועות** (Bubble sort) שהכרנו בפרק 7, "**מערכים**":

***void swap(CD *pcd1, CD *pcd2);***
***void sort_by_cost(CD arr[], int array_size);***

המיון מתבצע בפונקציה sort עפ"י מחירי התקליטורים: הפונקציה משווה בין כל זוג תקליטורים סמוכים במערך.

אם יש צורך בהחלפה, הפונקציה מעבירה את **כתובות** המבנים במערך (מדוע?) לפונקציה swap, וזו מחליפה בין ערכיהם.

***עיין/י בקוד הפונקציות וקוד התכנית המשתמשת שבעמ' 264-266.***

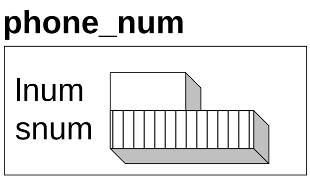
# union

union הוא סוג משתנה היכול להכיל בזמנים שונים נתון מגודל ומטיפוס שונה.

לדוגמא, אם נרצה לייצג בתכנית מספר טלפון פעם כמחרוזת ופעם כשלם ארוך (long) נוכל להגדירו כ- union :

```
union phone_num
{
	long lnum;
	char snum[30];
};
```

המהדר יקצה למשתנה phone_num זכרון לפי הנתון הגדול יותר:




union מוגדר בדומה למבנה שהייצוגים האפשריים שלו מוגדרים כשדות בתוכו. במקרה זה ניתן לייצג מספר טלפון כשלם ארוך או כמחרוזת.

הגדרת משתנה מסוג ה- union :

```
union phone_num p1;
```

וכעת אם רוצים להתייחס למשתנה כאל long :

```
p1.lnum = 48934562;
```

לעומת זאת אם נרצה לשמר את הקידומת - 04 - בצורה נוחה נייצג אותו כמחרוזת:

```
strcpy(p1.snum, "04-8934562");
```

כלומר ההתייחסות לסוג הייצוג היא ברישום של גישה לשדות במבנה.

מרכז ההדרכה 2000 © כל הזכויות שמורות www.mh2000.co.il

## כללים

- בדומה ל- struct ניתן להגדיר את ה- union כטיפוס ע"י typedef.
- המהדר דואג להקצות למשתנה מטיפוס union את הגודל המקסימלי האפשרי, כלומר את גודל השדה המקסימלי המוגדר ב- union.

  בדוגמא הנ"ל, המהדר יקצה למשתנה מסוג phone_num 30 בתים, כגודל המחרוזת.
- בדומה למבנים ניתן לגשת לייצוגים השונים של ה- union ע"י מצביע והאופרטור "->".

# סיכום

- **מבנה (structure)** ב - C מאפשר הגדרת אוסף משתנים מטיפוסים שונים כיחידה אחת. הוא נקרא גם **רשומה (record)**.
- המבנה מוגדר ע"י המילה השמורה **struct** באחת משתי הצורות - הגדרת טיפוס או שימוש בשם תג. בהתאם לכך מבוצעת הגדרת המשתנים מהמבנה.
- המשתנים במבנה נקראים **שדות.** שדה יכול להיות מטיפוס כלשהו, כולל משתנה מטיפוס מבנה אחר. הגישה לשדה דרך שם המבנה היא ע"י אופרטור הנקודה ".".
- ניתן להציב מבנה אחד למבנה אחר ע"י אופרטור ההצבה אך יש לשים לב שאם המבנה מכיל מצביע כשדה, המצביע הוא המועתק ולא התוכן המוצבע.
- ניתן להגדיר מצביע לטיפוס מבנה בדומה להגדרת מצביע לטיפוסים אחרים - תוך שימוש באופרטור "*". כדי להתייחס לשדות המבנה דרך מצביע משתמשים באופרטור "->" .
- ניתן גם להגדיר מערך מבנים. ההתייחסות לשדה באחד מאיברי המערך היא ע"י האופרטור [] בצירוף האופרטור ".".
- **union** הוא סוג משתנה היכול להכיל בזמנים שונים נתון מגודל ומטיפוס שונה.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבסוף פרק זה.***


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# 11. ניהול קבצי התוכנה בפרוייקט

- חלוקת התוכנה למספר קבצים
- הכללת קבצי ממשק
- שלב הקישור
- דוגמא: חלוקת תוכנית לקבצים
- הקדם–מעבד (Pre-Processor)
- סיכום
- תרגילי סיכום


מרכז ההדרכה 2000 © כל הזכויות שמורות

# חלוקת התוכנה למספר קבצים

כאשר כותבים תוכנה בהיקף גדול - כלומר תוכנה המכילה מאות, אלפי או מיליוני שורות קוד - נעשה קשה מאוד לתחזק אותה בקובץ יחיד. במצב זה מקובל לחלק את התוכנה למספר קבצים, הנקראים גם **מודולים**.

**מודול** היא יחידת תוכנה העוברת הידור נפרד - בשפת C זהו בדרך כלל בקובץ עם סיומת ".c" . בשלב **ההידור** כל מודול מהודר בנפרד, ללא תלות בהידור מודולים אחרים.

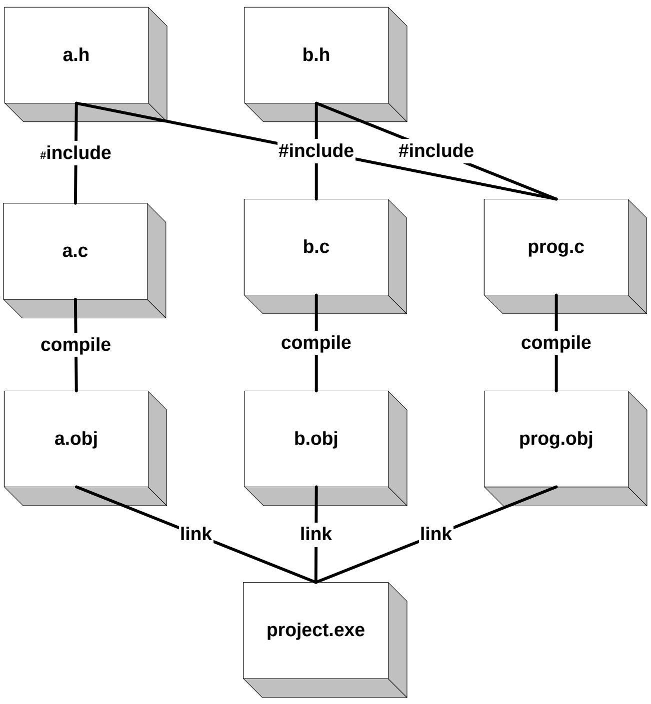
קריאות לפונקציות בין מודולים וכן שימוש במשתנים גלובליים (המוגדרים במודול יחיד) מושארות פתוחות עד לשלב הקישור.

לכל מודול מוגדר בדרך כלל קובץ נוסף, בעל סיומת h. : קובץ זה מכיל הכרזות של קבועים, טיפוסים, משתנים גלובליים ופונקציות שהמודול מייצא למודולים אחרים.

הכרזות אלה נדרשות לצורך ביצוע קריאות לפונקציות בין מודולים.

בשלב **הקישור** מאוגדים כל המודולים המהודרים לקובץ הרצה בודד. בשלב זה נפתרות כל הקריאות לפונקציות והשימושים במשתנים גלובליים בין מודולים.

לדוגמא: תוכנה מכילה שלושה מודולים a, b, ו- prog




קבצי ה- c. מבצעים **הכללה** (#include) לקבצי ה- h., וכל אחד מהם עובר **הידור** נפרד לקובץ אובייקט (obj.). בשלב הסופי מאוגדים קבצי האובייקט לקובץ ביצוע יחיד ע"י **קישור** (link).


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## קבצי ממשק וקבצי מימוש

מכיוון שקבצי ה- h. מכילים הכרזות של טיפוסים ופונקציות – ללא הגדרת גוף פונקציה או הגדרת משתנים - הם נקראים **קבצי ממשק**.

הטיפוסים, הקבועים והפונקציות המוכרזים הם הממשק למשתנים ולפונקציות המוגדרים בקבצי המקור.

קבצי המקור נקראים גם **קבצי המימוש** מכיוון שהם מכילים את המימוש של הפונקציות והמשתנים שהוכרזו בקבצי הממשק.

מודול מייצא את הממשק אליו דרך קובץ הממשק למודולים אחרים. כלומר הכרזות הטיפוסים, הקבועים והפונקציות הקיימות בקובץ הממשק מאפשרות למודול המכליל אותו לקרוא לפונקציות המוכרזות.

שיטה זו מאפשרת לנפות את שגיאות הריצה ביתר קלות הואיל והיא מקטינה את הקשרים בין המודולים למינימום.

חלוקה זו של התוכנה **לממשק** (**Interface**) **ולמימוש** (**Implementation**) היא חשובה מאוד ובאה לידי ביטוי במרבית שפות וגישות התכנות הקיימות.

בתכנות מונחה עצמים ההבחנה בין ממשק ומימוש היא אף חדה יותר: תכונות המודולריות, הסתרת מידע, טיפוסיות חזקה ופולימורפיזם מחייבות הפרדה בין ממשק המחלקות לבין מימושן. להרחבה, עיין/י בספרים "**Java על כוס קפה**" ו-"**תכנות מונחה עצמים ב- ++C**" בהוצאה זו.

# הכללת קבצי ממשק

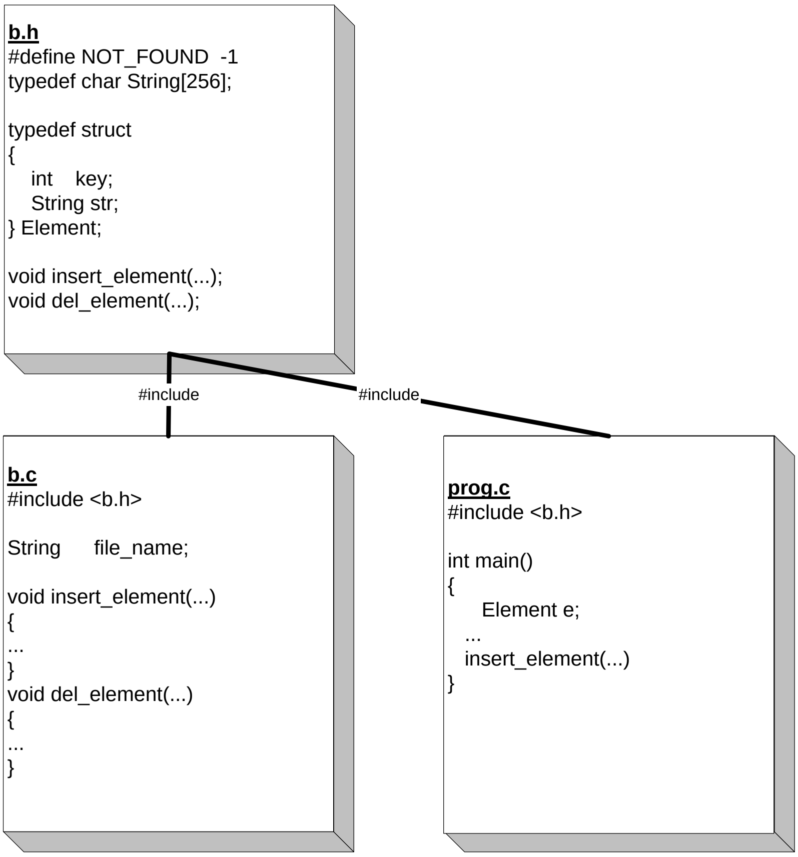
הכללת קבצי הממשק(סיומת h.) מבוצעת בחלקו הראשון של שלב ההידור ע״י **הקדם-מעבד (pre-processor)**. הוראות אלו מתחילות כולן בפקודה #include.

בקבצי הממשק מוכרזות הגדרות טיפוסים, אבטיפוסי פונקציות, והצהרות משתנים לצורך שימוש גלובלי, כלומר ע״י מספר מודולים.

כאשר רוצים לשתף טיפוס חדש, פונקציה או משתנה בין מספר מודולים, מצהירים עליהם בקובץ ממשק (h.) ומבצעים הכללה (#include) בכל אחד מהמודולים.

*הערה : **הגדרת** הפונקציות או המשתנים חייבת להופיע במודול יחיד!!*

דוגמא :



## פרישת הקובץ המוכלל בקובץ המקור

מה מבצע המהדר (ליתר דיוק הקדם-מעבד שבמהדר) בהכללת קובץ? הפעולה היא פשוטה מאוד: המהדר פורש את קובץ ה- h. בקובץ שבו בוצע לו include# ובכך למעשה הופך אותם לקובץ יחיד המוכן לשלב ההידור.

לדוגמא, לאחר מעבר הקדם-מעבד ולפני תחילת שלב ההידור ייראו הקבצים b.c ו- prog.c כך:

```
b.c
#define NOT_FOUND  -1
typedef char String[256];

typedef struct
{
   int    key;
   String str;
} Element;

void      insert_element(...);
void      del_element(...);

String        file_name;

void insert_element(...)
{
...
}
void del_element(...)
{
...
}
```

```
prog.c
#define NOT_FOUND  -1
typedef char String[256];

typedef struct
{
   int    key;
   String str;
} Element;

void      insert_element(...);
void      del_element(...);

int main()
{
   Element e;
     ...
   insert_element(...)
}
```

<u>הערות</u>

1. יש לשים לב שהכללה של הגדרות טיפוסים ואבטיפוסי פונקציות מספר פעמים בקבצים שונים אינה שגיאה בשפת C.

למעשה הגדרות כפולות (וזהות) אלו חוקיות אפילו אם הן מבוצעות באותו קובץ!

2. תיאור פרישת הקובץ הנ"ל אינו מדויק - הקדם-מעבד עובר על כל ההוראות המתחילות ב- # ומבצע אותן לפני שלב ההידור. לכן, גם הגדרת הקבוע

*#define NOT_FOUND  -1*

לא תופיע - הקדם-מעבד יחליף כל מופע של הקבוע NOT_FOUND בערך 1-.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## הכרזה על משתנה גלובלי

ניתן להכריז על משתנה גלובלי המוגדר בקובץ מסויים כך שיוכר גם בקבצים אחרים: כשם שניתן להגדיר אבטיפוס (prototype) לפונקציה כך ניתן להגדיר מעין אבטיפוס למשתנה גלובלי ע"י הוראת **extern**.

לדוגמא, נניח שמתוך התכנית הראשית בקובץ prog.c אנו מעוניינים לגשת למשתנה הגלובלי file_name המוגדר ב- b.c:

*String file_name;*

לשם כך, יש להכריז על המשתנה הגלובלי בקובץ b.h כך:

***extern*** *String file_name;*

המילה **extern** מציינת שהמשתנה מוגדר באיזשהו קובץ ושמעתה ניתן להשתמש בו.

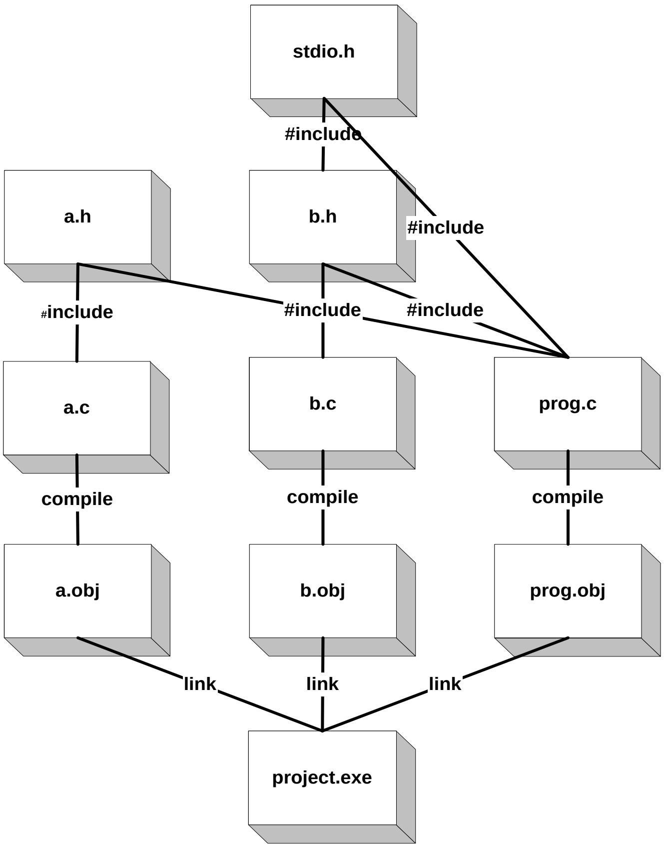

*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הכללת קבצי h. בקבצי h.

לפעמים קבצי h. מכילים בעצמם הוראות הכללה של קבצי h. אחרים. לדוגמא, אם בקובץ b.h הנ"ל נעשה שימוש בהגדרות הקיימות בקובץ stdio.h, יש לבצע הכללה של הקובץ:

```
b.h
#include <stdio.h>
#define NOT_FOUND  -1
typedef char String[256];

typedef struct element
{
        int         key;
        String      str;
} Element;

void  insert_element(Element *);
void  del_element(int);
```

נניח שגם הקובץ prog.c משתמש בספריית הקלט / פלט stdio.h. תרשים המודולים ייראה כעת כך:




כלומר, כעת הקובץ stdio.h מוכלל פעמיים במודול prog: פעם אחת כתוצאה מהכללה ישירה שלו ופעם שנייה כתוצאה מהכללת b.h המכליל בעצמו את stdio.h.

## מניעת הכללה מרובה

כדי למנוע הכללה מרובה של קבצי h. משתמשים בטכניקה המשלבת הוראות תנאי של הקדם-מעבד בצורה הבאה:

```
#ifndef STDIO_H
#define STDIO_H
int printf(...);
int scanf(...);
int getchar();
int putchar(int ch);
...
#endif
```

כדי למנוע הכללה כפולה של stdio.h בודקים בתחילתו אם המאקרו STDIO_H (השרירותי) אינו מוגדר.

בדיקה זו מבוצעת ע"י ההוראה #ifndef: תשובה חיובית (כלומר המאקרו לא מוגדר) גורמת לביצוע כל קטע הקוד שעד ההוראה המסיימת #endif.

לעומת זאת תשובה שלילית (המאקרו מוגדר) גורמת לדילוג על כל קטע הקוד שעד ההוראה המסיימת #endif. במילים אחרות, קוד זה לא ייפרש בקובץ המכליל.

בפעם הראשונה שהקדם-מעבד קורא את הקובץ stdio.h המאקרו STDIO_H עדיין לא מוגדר, ולכן התשובה להוראה

```
#ifndef STDIO_H
```

היא חיובית וגוף הקובץ מבוצע, כלומר מגדירים את המאקרו

```
#define STDIO_H
```

וגוף הקובץ נפרש בקובץ המכליל:

```
int printf(...);
int scanf(...);
int getchar();
int putchar(int ch);
...
```

בפעמים הבאות שהקדם-מעבד קורא את הקובץ stdio.h המאקרו STDIO_H מוגדר ולכן התשובה להוראה

```
#ifndef STDIO_H
```

היא שלילית - והקובץ לא נפרש שוב.

*מרכז ההדרכה 2000 © כל הזכויות שמורות*      www.mh2000.co.il

# שלב הקישור

שלב הקישור מבוצע בדרך כלל בסביבת הפיתוח ע"י הגדרת פרוייקט (project) והכנסת קבצי המימוש לתוכו (אין צורך להכניס קבצי ממשק).

בביצוע הרצה סביבת הפיתוח מבצעת את הקישור באופן אוטומטי ע"י מרכיב במהדר הנקרא **מקשר** (**Linker**). דוגמאות לסביבות פיתוח:

- בסביבת הפיתוח **MS-DevStudio**, יוצרים פרוייקט ע"י בחירה בתפריט File/New/Projects, ובוחרים בסוג הפרוייקט (בחירה ב- "Win32 Console Application" עבור יישומי DOS).

  לאחר מכן יוצרים קבצי h. ו- c. ע"י File/New/files ובוחרים בקובץ מקור c. או בקובץ ממשק h. תוך ציון הוספתו לפרוייקט.

- בסביבת הפיתוח של **Borland**, **Turbo C++**, קיים תפריט Project ליצירת פרוייקט ולהוספת קבצים לתוכו.

  את הקבצים יוצרים בנפרד ע"י File/new, עורכים אותם ומוסיפים אותם (רק קבצי מימוש!) לפרוייקט ע"י Project/Add item.

  בדרך כלל, שם קובץ ה- exe. הנוצר הוא כשם קובץ הפרוייקט.

- בסביבת פיתוח טקסטואליות הוראות ההידור והקישור נכתבות באופן מפורש בשורת הפקודה ע"י המתכנת, או מתוך קובץ script.

  לדוגמא, במערכת Unix המהדר מופעל ע"י הפקודה **cc** :

**cc -c** a.c b.c prog.c

פקודה זו תגרום להידור קבצי ה- c. לקבצי אובייקט עם שם זהה אך עם סיומת o.. בכדי לבצע קישור של המודולים נקרא למהדר עם האופציה o- :

***cc -o** prog a.o b.o prog.o*

פקודה זו תגרום לקישור המודולים a.o, b.o ו- prog.o לקובץ ביצוע בשם prog.

## מה מתבצע בקישור?

בשלב הקישור נפתרות כל הקריאות לפונקציות חיצוניות והשימוש במשתנים הגלובליים: אם המקשר לא מצליח למצוא פונקציה או משתנה גלובלי מסויים, הוא מודיע על שגיאה.

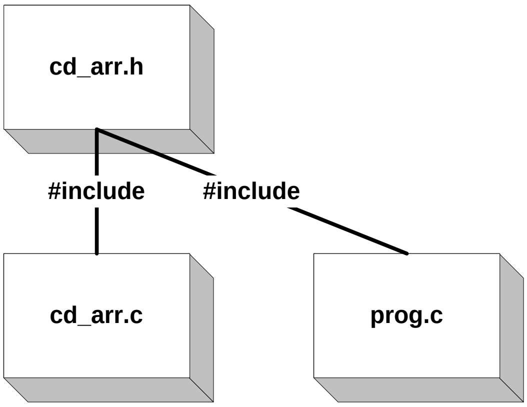
כמו כן בשלב הקישור מצרף המקשר לתכנית את **גוף** פונקציות הספרייה שקראנו להן.

לדוגמא, אם בתכנית קיימות קריאות לפונקציות printf ו- scanf שהממשק שלהן מוגדר ב- stdio.h , המקשר דואג לצרף את גוף הפונקציות בזמן קישור מתוך הספרייה התקנית של שפת C (הנקראת גם ספריית זמן ריצה, Standard C runtime library).

# דוגמא: חלוקת תוכנית לקבצים

נבצע חלוקה של תוכנית תקליטורי המוסיקה למספר קבצים:

| | |
|---|---|
| cd_arr.h | הכרזות הקבועים, הטיפוסים והפונקציות של מערך התקליטורים |
| cd_arr.c | הגדרות המשתנים והגדרות (גוף) הפונקציות של מערך התקליטורים |
| cd_main.c | התכנית המשתמשת: קוראת לפונקציות במערך התקליטורים |




ספריית הפרוייקט וקובץ הפרוייקט ייקראו **cd_proj**.

***קוד התכנית מובא בעמ' 277-282.***

# הקדם-מעבד (Pre-Processor)

הקדם-מעבד הוא תת-תכנית במהדר העוברת על הקוד שבקובץ ומבצעת את כל ההוראות המתחילות בסימן #.

הקדם-מעבד מטפל בהוראות ממספר סוגים:

- הכללות קבצים - הוראת include#
- הגדרות קבועים - הוראות define#, undef#
- הגדרות פונקציות מאקרו - הוראת define# עם פרמטרים
- הידור מותנה - הוראות if#, ifdef#, ifndef#, elif#, else#
- הוראות נוספות - הוראות error, #line, #pragma#

### הכללות קבצים

ראינו שההוראה include# מכלילה את הקובץ ששמו נתון לה כפרמטר בקובץ המהודר.

לדוגמא ההוראה

*#include <stdio.h>*

בקובץ prog.c תפרוש בקובץ prog.c את תכולת הקובץ stdio.h בשורת ההוראה. אם stdio.h כולל הוראות include# בעצמו, הן יבוצעו באופן רקורסיבי.

כאשר מבצעים הכללה תוך ציון שם הקובץ בין גרשיים

*#include "myfile.h"*

המהדר יחפש את הקובץ בספריית המחדל שלו - כלומר הספרייה הנוכחית שבה נמצא הפרוייקט או הקובץ המהודר.

## הגדרות קבועים

כפי שראינו, ניתן להגדיר קבועים ע"י הקדם-מעבד, כך שהערכים המוגדרים יוחלפו בשמות הקבועים בשלב הראשוני של ההידור:

```
#define NOT_FOUND  -1
```

ניתן להגדיר קבוע מכל סוג שהוא: שלם, ממשי, תוי, מחרוזת. דוגמאות:

```
#define INUM     454
#define FNUM     66.78f
#define FEXP     2.55E5
#define CH       'A'
#define HELLO    "hello"
```

כמו כן אפשר להגדיר הוראות שלמות לביצוע ע"י קבוע, לדוגמא:

```
#define CHECK_POINT  printf("CP: line %d, file %s", __LINE__, __FILE__);
```

(המאקרואים __FILE__ , __LINE__ מוסברים להלן) ומתוך התכנית ניתן להפעיל את המאקרו בכל שלב בכדי לדעת שהגענו לנקודה מסויימת בתכנית בזמן ריצה:

```
int x, ret;
printf("Enter an integer:");
ret = scanf("%d", &x);
CHECK_POINT
```

ניתן גם להגדיר קבועים ללא ערך, כך שישמשו כדגלים בהידור מותנה (ראה/י להלן):

```
#define FLAG
```

ניתן לבטל קבוע לאחר שהוגדר ע"י ההוראה undef#. לדוגמא, ניתן לבטל את הדגל הקודם ע"י

```
#undef FLAG
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## הגדרת פונקציות מאקרו

ניתן להגדיר ע"י הקדם-מעבד פעולות לביצוע עם פרמטר, בדומה לפונקציות. דוגמאות:

```
#define    MAX(a,b)        a > b? a : b
#define    POWER2(a)       a*a
```

וכעת ניתן לקרוא לפונקציות המאקרו כאילו היו פונקציות רגילות:

```
int x,y, ret;
printf("\nEnter 2 integers:");
scanf("%d %d", &x, &y);
printf("\nThe maximum is %d", MAX(x,y));
printf("\nx^2 = %d", POWER2(x));
```

פונקציות מאקרו נפרשות בקוד בשלב המעבר של הקדם-מעבד, והן אינן מתורגמות לפונקציות אמיתיות. כתוצאה מכך הן מהירות יותר לביצוע מכיוון שנחסך המעבר דרך מנגנון **מחסנית הקריאות** (ראה/י פרק 6, "**פונקציות**").

מצד שני, פרישה מחודשת בכל מקום של פונקציות המאקרו עלולה להגדיל את **קטע הקוד** (Code Segment) של התכנית.

**מסקנה**: רצוי להשתמש בפונקציות מאקרו עבור פונקציות קטנות במיוחד לצורך ייעול התכנית.

***שיפור ב C++ :*** בשפת C++ ניתן להגדיר פונקציה כ- **inline** ובכך לגרום לכך שהיא תיפרש בקוד, בדומה לפונקציית מאקרו. לדוגמא, פונקציה המחזירה את המינימלי מבין שני מספרים שלמים:

```
inline int MIN(int x, int y)
{
   return x < y ? x: y;
}
```

השיפור העיקרי כאן הוא בכך שהמהדר הוא זה שמטפל בפונקציה, ולכן לא קיימות הבעיות שלעיל. בנוסף לכך המהדר בודק התאמה של טיפוסים בקריאה לפונקצית inline, בעוד שבפונקציות מאקרו אין בדיקת טיפוס.

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הידור מותנה

ניתן להתנות הידור של קטעי קוד בתכנית כתלות בתנאי קדם-מעבד כלשהו. דוגמא שכיחה לכך היא מניעת הכללה מרובה של קבצים.

לדוגמא, קובץ ממשק myfile.h יוקף ע"י ההוראות הבאות

```
#ifndef MYFILE_H
#define MYFILE_H
...
#endif
```

בכדי למנוע הכללה מרובה של הקובץ באותה יחידת הידור.

שימוש נפוץ נוסף הוא לצורך ניפוי (debug). לדוגמא, אם מעונינים שקוד בדיקה מסוים יהיה קיים רק בשלב הפיתוח, ניתן להגדיר קבוע ולהתנות את קטע הקוד הנ"ל בהגדרתו.

לדוגמא:

```
#ifdef MY_DEBUG
void my_debug(char *buff, int size)
{
	...
}
#endif
```

כעת הידורה של הפונקציה ביחידת ההידור בה היא נמצאת תלויה בהגדרת המאקרו MY_DEBUG: אם הוא הוגדר לפני כן ע"י

```
#define MY_DEBUG
```

אזי קטע הקוד ייכלל ביחידת ההידור. אחרת, הפונקציה my_debug כלל לא תיפרש בהידור.

מרכז ההדרכה 2000 © כל הזכויות שמורות

## מקרואים מוגדרים מראש

***עיין/י ברשימת המקרואים המוגדרים מראש שבעמ' 287.***

## הוראות נוספות

***הקדם-מעבד כולל את הוראות נוספות המפורטות בעמ' 288.***

# סיכום

- נהוג לחלק תוכניות גדולות למספר קבצים, כאשר כל קובץ מטפל בנושא מסוים בתוכנית. **מודולריות** היא חלוקה של התוכנה למספר **מודולים** עפ"י נושאים.

  כל מודול כולל קובץ **מימוש** (.c) וקובץ **ממשק** (.h).

- קובץ **הממשק** כולל הכרזות על קבועים, טיפוסים ואבטיפוסים של פונקציות. קובץ **המימוש** מכיל הגדרות של משתנים גלובליים ואת גוף הפונקציות.

- קריאה לפונקציה ממודול אחד לשני מבוצעת ע"י **הכללת** (#include) קובץ הממשק שלו בשלב **ההידור**.

  בזמן **הקישור המקשר (linker)** יוצר קובץ ביצוע ( exe. ) יחיד וכל קוד הקבצים משולב לתוכו, תוך פתירת הקריאות בין המודולים.

- כדי למנוע הכללה מרובה של קבצי הממשק, משתמשים בטכניקה המשלבת הוראות תנאי של הקדם-מעבד: #ifndef, #define, #endif.

- הקדם-מעבד כולל הוראות מסוגים שונים:

  – הכללת קבצי ממשק

  – הגדרות קבועים

  – הגדרות פונקציות מאקרו

  – הידור מותנה

  – הגדרות על הקובץ המהודר

  – הגדרות תלויות סביבה למהדר

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבסוף פרק זה.***

# 12. הקצאת זיכרון דינמית ורשימות מקושרות

- הקצאות מקום למשתנים ושיקולי צריכת זיכרון
- תכנית דוגמא: ספרייה
- מערכי מצביעים
- רשימות מקושרות
- סיכום
- תרגילי סיכום

מרכז ההדרכה 2000 © כל הזכויות שמורות

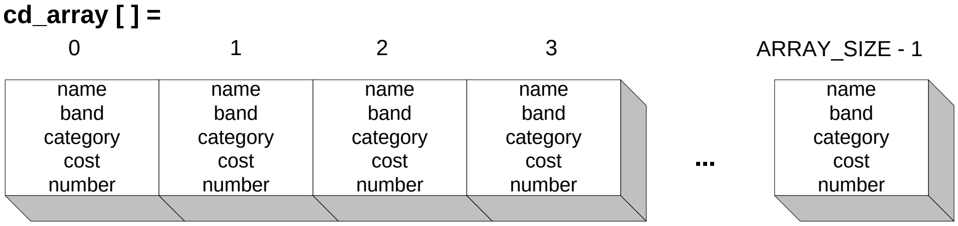
# הקצאות מקום למשתנים ושיקולי צריכת זיכרון

עד עתה טיפלנו במשתנים שמקומם בזיכרון הוקצה ע"י המהדר. לא נדרשה לשם כך התערבות שלנו - לא ביצענו הקצאת זיכרון למשתנים וכן לא טיפלנו בשחרור הזיכרון בתום השימוש.

אופן העבודה בו המהדר מטפל בהקצאת המקום למשתנים - ללא הקצאת זיכרון מפורשת ע"י המתכנת - הוא פשוט ונוח מצד אחד, אך בעל חסרון משמעותי מצד שני: כאשר אנו עובדים עם מבני נתונים בגודל שאינו ידוע מראש אנו מקצים את הזיכרון המקסימלי האפשרי ובכך מבזבזים מקום.

לדוגמא, את מערך התקליטורים הגדרנו בפרקים הקודמים כך:

*CD cd_array[ARRAY_SIZE];*




אם ARRAY_SIZE הוא 500 למשל, ובתכנית השתמשנו רק ב- 50 רשומות, היא מנצלת בפועל רק 10 אחוזים מהזיכרון שהוקצה עבורה.

יתירה מזאת, שדות המבנים עצמם מכילים הקצאות המתייחסות למקסימום: שם התקליטור (name), שם הלהקה (band) והקטגוריה (category) מוגדרים כולם כמחרוזות בעלות אורך מקסימלי אפשרי.

דוגמא נוספת: בתכנית לניהול ספרייה (שניתנה כתרגיל בפרקים הקודמים) מחרוזות הכלולות ברשומת "ספר" מוגדרות כמערכים בגודל המקסימלי האפשרי, מטיפוס String :

```
typedef char String[256];

typedef struct
{
	String name;
	String writer;
	String publisher;
	int  year;
	...
} Book;
```

גם כאן מבזבזים זיכרון ע"י התייחסות ל"מקרה הגרוע ביותר".

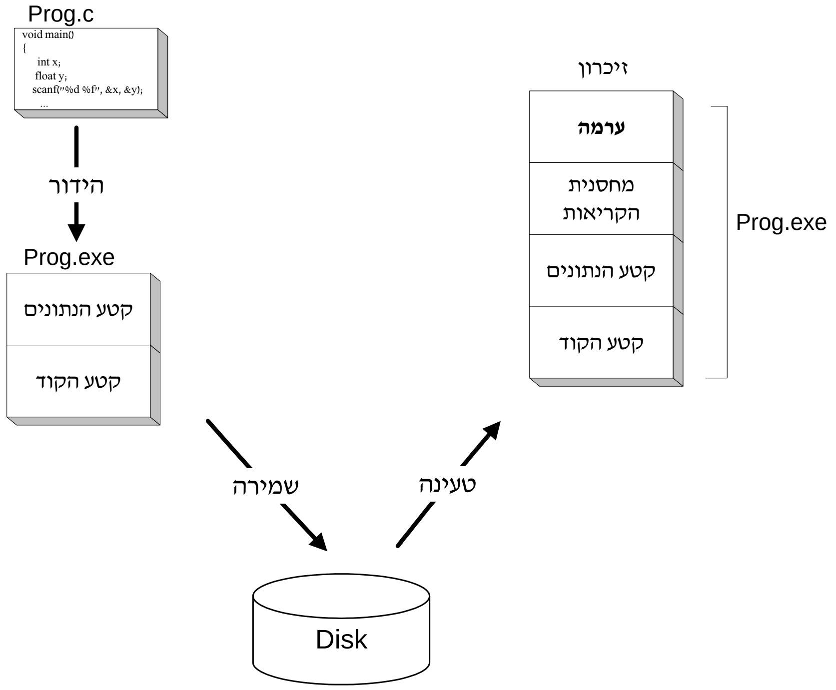
## מנגנון הקצאת זיכרון דינמית

בכדי לשפר את ניצולת הזיכרון בתכניות, הוגדר בשפת C מנגנון להקצאת זיכרון מפורשת ע"י המתכנת במהלך התכנית.

הקצאת זיכרון מפורשת נקראת גם **הקצאת זיכרון דינמית (dynamic memory allocation)**.

הקצאות זיכרון דינמיות מבוצעת מתוך מאגר שמור של זיכרון הנקרא **ערמה** (**heap**).

מאגר זיכרון מסוג זה מסופק לכל תכנית בתחילת ריצתה ומאפשר לה יכולת הרחבת משאבי הזיכרון שלה. כפי שראינו בפרק 6, "**פונקציות**", תהליך יצירת וטעינת התכנית לזכרון נראה כך:




אספקת ערמה לכל תכנית היא באחריות מערכת ההפעלה: מבנה הערמה, גודלה ופרטי מימוש נוספים תלויים בסוג מערכת ההפעלה בה התכנית מורצת.

עם זאת, הפונקציות המבצעות את הקצאת הזיכרון ושחרורו בשפת C הן תקניות ואינן תלויות מערכת.

קיימות 2 פונקציות עיקריות לטיפול בהקצאה ושחרור דינמיים של זיכרון:

**malloc()** פונקציה להקצאת זיכרון דינמית: מספקים לפונקציה את גודל הזיכרון הנדרש והיא מקצה זיכרון ומחזירה מצביע אליו. היא מבצעת זאת ע"י פנייה למנהל הזיכרון של מערכת ההפעלה. אבטיפוס הפונקציה:

*void *malloc( size_t size );*

size_t הוא טיפוס של גדלי משתנים. הוא מוגדר פשוט כשלם בלתי מסומן בקובץ stddef.h (ובקבצים נוספים):

*typedef unsigned int **size_t**;*

**free()** פונקציה לשחרור זיכרון שהוקצה דינמית. אבטיפוס הפונקציה:

*void **free**( void *ptr);*

תכנית דוגמא:

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

void main()
{
	char *str;

	/* allocate 80 chars for str */
	str = malloc(80);
	if( str == NULL )
		puts("Insufficient memory available");
	else
	{
		puts("Enter a string:");
		gets(str);
		puts("The string entered");
		puts(str);

		/* free the allocated memory */
		free(str);
		puts("Memory freed");
	}
}
```

<u>הסבר</u>

כדי להשתמש בפונקציות ההקצאה והשחרור של הזיכרון יש להכליל את הספרייה **stdlib.h**:

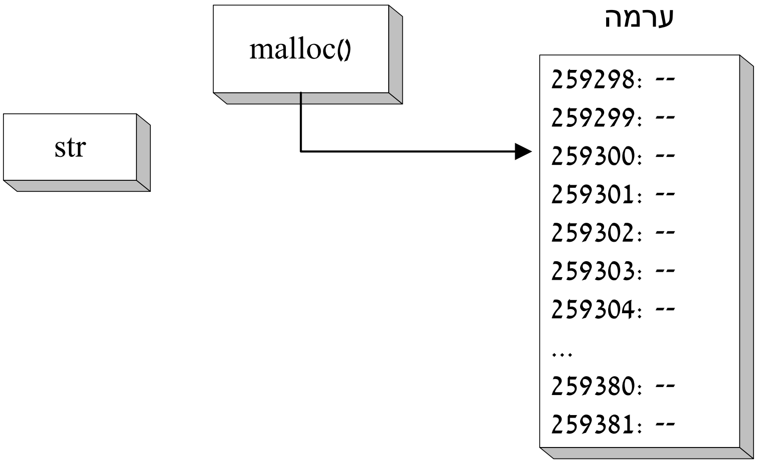
```
#include <stdlib.h>
```

התכנית מגדירה מצביע לתו בשם str ומקצה לו זיכרון בגודל של 80 תווים:

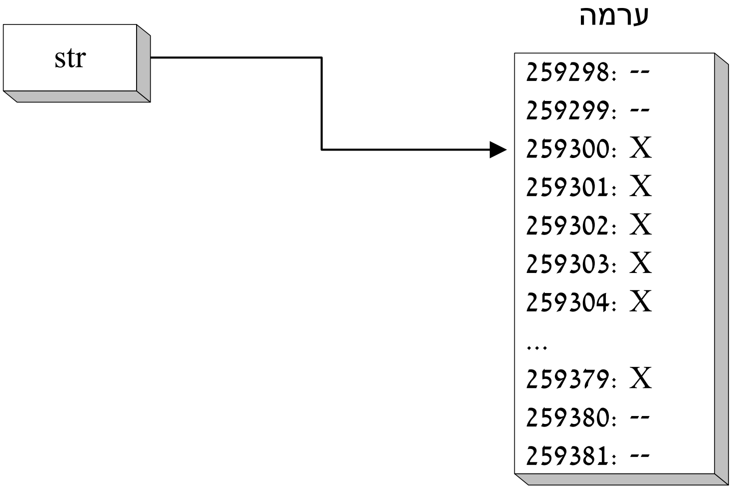
```
char *str;
...
str =  malloc(80);
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

הפונקציה ()malloc מבצעת הקצאת זיכרון מהערמה בגודל הנדרש ומחזירה את כתובתו, שמוצבת למצביע. תרשים הזיכרון בזמן ההקצאה:




תרשים הזיכרון לאחר ההקצאה:




בלוק הבתים שהוקצו מסומן ע"י מנהל הזיכרון כתפוס.

אם הפונקציה malloc נכשלת - כלומר אם אין בנמצא זיכרון להקצאה - מוחזר NULL. הבדיקה מבוצעת ע"י

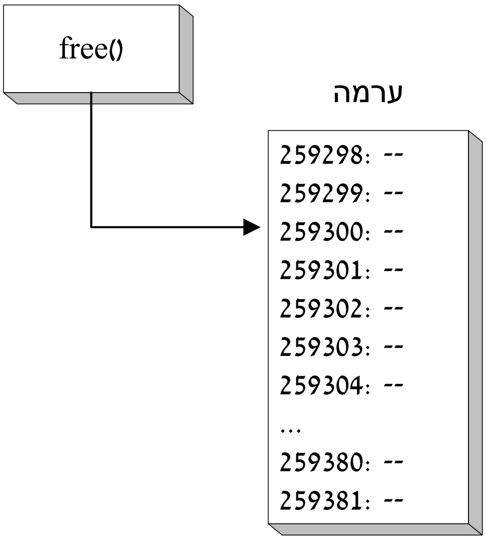
```
if( str == NULL )
	puts("Insufficient memory available");
```

לאחר הקריאה ל- ()free משוחרר הבלוק שהוקצה:

```
free(str);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

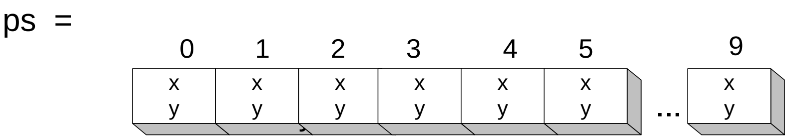
תרשים הזיכרון לאחר השחרור :




דוגמא נוספת - הקצאת זיכרון עבור מבנים :

```
typedef struct
{
        int   x;
        float y;
} S;

S  *ps;

ps =  malloc(10 * sizeof(S));
```

בדוגמא זו הקצינו 10 מבנים מסוג S :




ניתן כעת להתייחס אל הזיכרון המוקצה כאל מערך, לדוגמא :

```
for(i=0; i<10; i++)
{
        ps[i].x = i;
        ps[i].y = i*0.5;
}
```

שחרור הזיכרון :

```
free(ps);
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# תכנית דוגמא: ספרייה

בסעיף זה נממש תוכנה לניהול ספרייה כתוכנית המשך לתרגיל הסיכום מפרק 10, "**מבנים**".

התוכנה תאפשר ביצוע פעולות שונות כגון: הוספת ספרים, מחיקה, מיון, הדפסת דו"ח וכו'.

**דרישות**

- ספר מיוצג ע"י רשומה הכוללת שם, שם מחבר, שם הוצאה לאור , שנת הוצאה לאור, מחיר ומספר מזהה.
- התוכנה תאפשר להוסיף ספרים באופן הדרגתי כשהם ממוינים עפ"י המספר המזהה, לבצע חיפוש ספר עפ"י שם או מספר סידורי וכן להדפיס דו"ח של רשימת הספרים.
- כמו כן נדרשת הפעלה של התוכנה ע"י ממשק משתמש נוח.

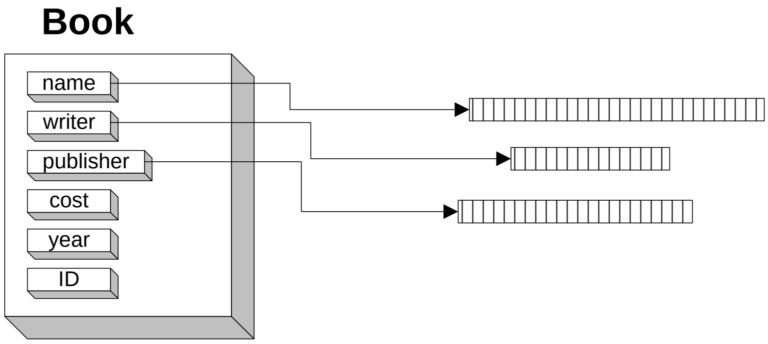

מרכז ההדרכה 2000 © כל הזכויות שמורות

## הקצאה דינמית של מחרוזות

ראשית נגדיר רשומת ספר - את המחרוזות נגדיר כמצביעים ולא כמערכים:

```
/* Book type */
typedef struct
{
	char  *name;
	char  *writer;
	char  *publisher;
	float  cost;
	int    year;
	int    ID;
} Book;
```

וכעת רשומת ספר נראית כך בזכרון:




את גדלי המחרוזות נקצה בזמן ביצוע הקלט, עפ"י אורך המחרוזת הנקראת. לדוגמא, בכדי לקלוט את שם הספר (name) לתוך מבנה בשם book נבצע:

– הגדרת מחרוזת מקומית, temp

```
String temp;
```

– קריאה מהקלט לתוך המחרוזת:

```
gets(temp);
```

– הקצאת זיכרון לשם הספר עפ"י אורך המחרוזת temp + 1:

```
book.name =  malloc((strlen(temp)+1);
```

– העתקת המחרוזת temp ל- book.name:

```
strcpy(book.name, temp);
```


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## הפונקציה strdup

הפעולה שביצענו לעיל היא מסורבלת - אם נצטרך לבצע אותה על כל מחרוזת התכנית תהיה מורכבת ורגישה לשגיאות.

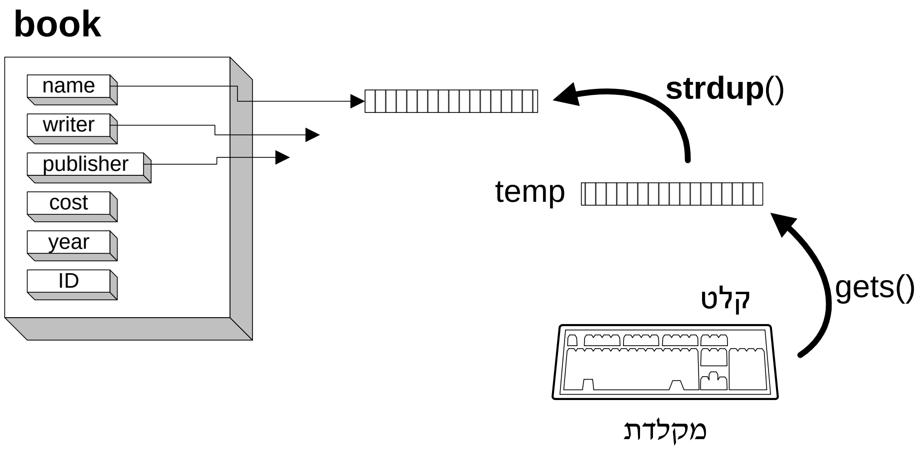
בספרייה string.h קיימת פונקציה בשם **strdup** - תפקידה הוא לבצע את שתי הפעולות האחרונות בבת-אחת ובפשטות:

*gets(temp);*
*book.name = **strdup**(temp);*

<u>הסבר</u>: הפונקציה strdup מוגדרת כך:

*char *strdup( const char *str);*

הפונקציה מקבלת כפרמטר מחרוזת, מקצה זיכרון בגודל מתאים, מעתיקה את המחרוזת המקורית לחדשה ומחזירה מצביע למחרוזת החדשה:




מכיוון שבפועל מבוצעת הקצאה דינמית, יש לדאוג לשחרור מתאים בסוף השימוש במחרוזת ע"י:

***free(book.name);***

***פונקצית הקלט עבור רשומת ספר מובאת בעמ' 300.***

<u>תרגיל</u>

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 301.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# מערכי מצביעים

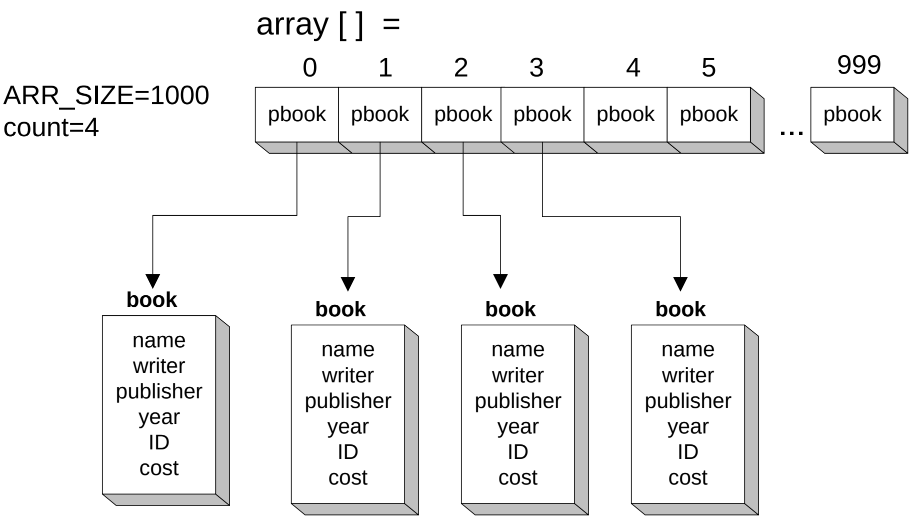
התכנית המטפלת בספרים עדיין אינה יעילה מבחינת הקצאת הזיכרון - המבנים עצמם מוקצים באופן סטטי. נשנה את הגדרת המבנים כך שגם הקצאתם תהיה דינמית.

נגדיר מערך מצביעים למבנים:

```
#define ARR_SIZE 1000

Book *array[ARR_SIZE];
int count=0;
```

הקבוע ARR_SIZE מציין את גדלו המקסימלי של המערך. המשתנה מונה את מספר האיברים הנוכחי במערך.

לדוגמא, תרשים זיכרון של מערך בגודל 1000 הכולל הקצאות של 4 ספרים:




אופן הוספת רשומה חדשה למערך המצביעים:

```
if(count<ARR_SIZE)
{
        array[count] =   malloc(sizeof(Book));
        book_input(array[count]);
        count++;
}
```

לצורך הנוחות, ניתן להגדיר טיפוס **מצביע לספר** באופן הבא:

```
typedef Book* BookPtr; /* a pointer to Book struct */
```

ואת המערך ניתן כעת להגדיר כך:

```
BookPtr     array[ARR_SIZE];
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

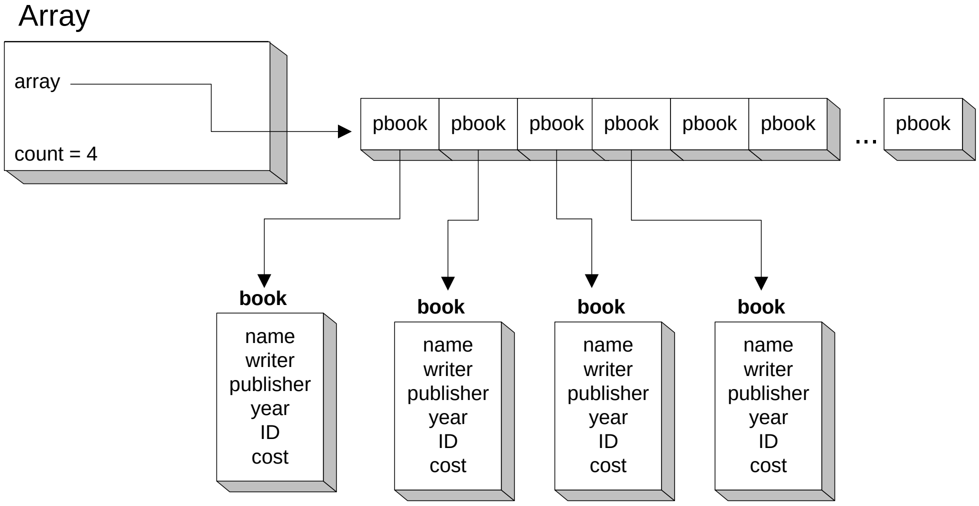
## הגדרת מבנה מערך מודולרי

לצורך מודולריות התכנית, רצוי להגדיר **מבנה** (struct) המתאר את המערך.

מבנה זה יכלול את המצביע array למערך, את מספר האיברים הנוכחי ברשימה ומידע נוסף - לפי הצורך. לדוגמא, עבור רשימת הספרים נגדיר את המבנה:

```
/* array structure */
typedef struct
{
        BookPtr        array[ARR_SIZE];   /* pointer to the book array */
        int            count;             /* number of elements in the array */
} Array;
```

המבנה המלא של המערך מוצג בתרשים הבא:




כעת התכנית המשתמשת יכולה להגדיר מערך ע"י

*Array arr1;*

אופן זה של הגדרת מערך מאפשר שימוש חוזר באותה תכנית להגדרת מספר מערכים, לדוגמא:

*Array arr1, arr2, arr3;*

***בעמ' 303-304 מובאות פונקציות לטיפול במערך:***

- פונקציה להוספת ספר למערך
- פונקציה למחיקת ספר מהמערך


מרכז ההדרכה 2000 © כל הזכויות שמורות

## מיון מערך מצביעים

נניח שאנו מעוניינים למיין את מערך הספרים. לשם כך נכתוב פונקציה בשם ()array_sort הממיינת בשיטת מיון בועות.

פונקציה זו נעזרת בפונקציה ()swap להחלפה בין שני איברים סמוכים.

הפונקציה swap מקבלת כפרמטרים **מצביעים למצביעים** לרשומות ספר:

```
void swap(BookPtr *ppbook1, BookPtr *ppbook2)
{
	BookPtr temp = *ppbook1;
	*ppbook1 = *ppbook2;
	*ppbook2 = temp;
}
```

ופונקצית המיון עצמה, array_sort :

```
void array_sort(Array *parray)
{
	int i,j;
	for(i=0; i < parray->count - 1; i++)
		for(j=0; j < parray->count- 1 - i; j++)
			if(parray->array[j]->ID > parray->array[j+1]->ID)
				swap(&parray->array[j], &parray->array[j+1]);
}
```

<u>מיון ע"י פונקצית הספרייה ()qsort</u>

כפי שראינו בפרק 8, "**מצביעים**", הספרייה התקנית של שפת C כוללת פונקצית מיון בשם ()qsort, הפועלת בשיטת **מיון מהיר** (**Quick Sort**) על מערך איברים מטיפוס כללי.

שיטת מיון זו ממיינת את המערך ע"י ביצוע פעולות חלוקה ומיון רקורסיביים, והיא יעילה במרבית המקרים משיטת **מיון בועות**.

אבטיפוס הפונקציה ()qsort מוכרז כך בקובץ stdlib.h :

```
void qsort( void *array,
	size_t count,
	size_t element_size,
	int compare(const void *element1, const void *element2 ) );
```

הפרמטרים שמקבלת הפונקציה :

**array** - כתובת הבסיס של המערך

**count** - מספר האיברים למיון

**element_size** - גודל זיכרון של כל איבר

**compare** - פונקציה המבצעת השוואה בין שני איברים.

הפונקציה ()qsort תשתמש בפונקציה ()compare בעת ביצוע המיון, כדי לבדוק האם להחליף בין איברים במערך.

לדוגמא, נכתוב פונקצית השוואה בין שני מצביעים לספרים:

```
/* compare 2 pointers to books and return value:
        > 0         if book1 > book2
        < 0         if book1 < book2
        = 0         if book1== book2
*/
int book_compare(const void *ppbook1, const void *ppbook2)
{
    BookPtr pbook1 = *(BookPtr *)ppbook1;
    BookPtr pbook2 = *(BookPtr *)ppbook2;

    return pbook1->ID - pbook2->ID;
}
```

כפי שניתן לראות, הפונקציה מגדירה את המצביעים מסוג * const void, בכדי שהפונקציה תתאים לממשק הכללי שהוגדר ב- ()qsort.

בתוך הפונקציה מבצעים המרה מפורשת של המצביעים ל- * BookPtr, ומציבים את הנתון המוצבע על ידם (מסוג BookPtr) למשתנים המתאימים בפונקציה:

```
    BookPtr pbook1 = *(BookPtr *)ppbook1;
    BookPtr pbook2 = *(BookPtr *)ppbook2;
```

לבסוף משווים בין שני הנתונים ע"י חיסור שדה ה- ID:

```
    return pbook1->ID - pbook2->ID;
```

וכעת נוכל לקרוא לפונקציה ()qsort מתוך הפונקציה array_sort :

```
void array_sort(Array *parray)
{
    qsort((void *)parray->array,
        (size_t)parray->count,
        sizeof(BookPtr),
        book_compare);
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## הוספת ממשק למשתמש

עד עתה ביצענו את הפעולות בתכנית הראשית באופן שרירותי: הגדרנו מערך מבנים, אתחלנו אותו, מיינו עפ״י שדה מסוים וכו' - כל זאת ללא התערבות המשתמש.

בדרך כלל מהלך התכנית כולו מוכתב ע״י המשתמש: על התכנית לספק לו אפשרויות בחירה שונות בכל שלב ולאפשר לו לקבוע את מהלכה.

מנגנון מסוג זה נקרא **ממשק למשתמש (User Interface).**

קיימים סוגים רבים של ממשקי משתמש וסגנונות שונים. הם נחלקים ל- 2 קטגוריות עיקריות:

– ממשקים גרפיים (GUI = Graphic User Interface)

– ממשקים טקסטואליים

בשפת C אין הגדרה תקנית לספריית ממשקים גרפיים. ספריות אלו הן תלויות במערכת ובסביבת הפיתוח בה עובדים.

לעומת זאת קלט/פלט תקני כן מוגדר בשפה (stdio) ובאמצעותו ניתן לבנות ממשק משתמש טקסטואלי.

מכיוון שאנו כותבים תכניות לשפת C התקנית (ANSI-C), נבנה ממשק טקסטואלי לתכנית.

הממשק למשתמש יבוא לביטוי בדרך כלל בפונקציה שתציג לו את תפריט האפשרויות השונות:

```
void menu()
{
	printf("\n\n\n\n\n");
	puts("Book program main menu");
	puts("-----------------------------------");
	puts("a. Add a book to the array");
	puts("d. Delete a book from the array");
	puts("i. Find a book by ID");
	puts("n. Find a book by name");
	puts("s. Sort the book array by ID");
	puts("p. Print all books");
	puts("x. Exit");
	puts("\n\n\n\n\n\n\n\n");
}
```

פונקצית עזר נוספת נדרשת בתכנית:

```
void pause()
{
	puts("\nHit Enter to continue");
	getchar();
}
```

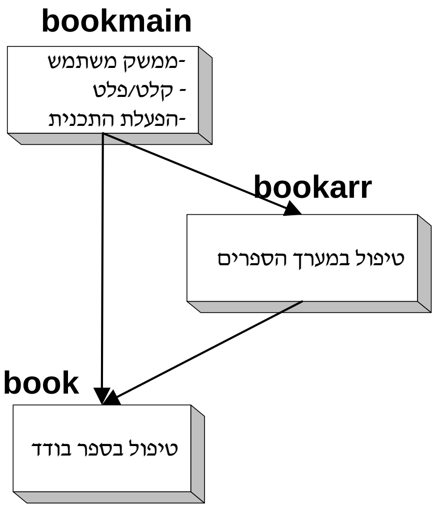
בפונקציה ()main תופעל הפונקציה ()menu, בחירת המשתמש תיקרא ובהתאם תבוצע ההוראה. קוד הפונקציה מובא בעמ' 309.

## חלוקת התכנית לקבצים

תכנית הספרייה הכוללת מחולקת לקבצים באופן הבא:

- book.h, book.c - מודול המטפל בספר יחיד.
- bookarr.h, bookarr.c - מודול המטפל ברשימת ספרים.
- bookmain.c - מודול המכיל את הפונקציה הראשית וממשק המשתמש.

החלוקה למודולים מתוארת בתרשים המודולים הבא:




***קוד התכנית בכללותו במלואו מובא בעמ' 311-316.***

## הרחבת מערך ע"י realloc

מערך המצביעים שראינו יעיל יותר, מבחינת הקצאת זיכרון, ממערכים רגילים אולם עדיין קיימת מגבלה רצינית בשימוש בו:

– גדלו הסטטי, הנקבע מתחילת התכנית גורם מצד אחד לבזבוז מקום במידה ויש מעט איברים במערך

– מצד שני למגבלה קשה כאשר נדרש להחזיק מספר איברים גדול.

הפתרון לבעיה הוא הגדלת זיכרון המערך לפי הצורך. הרעיון: נקצה את המערך באופן דינמי לגודל התחלתי מסוים.

אם נגיע למצב שהתפוסה מלאה ואין מקום להכניס איברים נוספים, נקצה זיכרון גדול יותר - למשל, פי 2 מהקיים - ונעתיק את תוכן המערך הישן לחדש.

פונקצית הספרייה **realloc** עושה בדיוק פעולה זו: היא מקבלת כפרמטר מצביע לזיכרון וגודל הקצאה:

**void *realloc( void ****ptr***, size_t** *size* **);**

realloc פועלת באחת משתי הדרכים:

1. מנסה להגדיל את האזור המוקצה לטובת המצביע ptr לגודל size.

2. אם זה לא ניתן, realloc מקצה בלוק זיכרון חדש בגודל size במקום אחר, מעתיקה את תוכן הבלוק הישן לחדש ומשחררת את הבלוק הישן.

## תרגילים

***קרא/י סעיף זה בספר ובצע/י את תר' 1-2 שבעמ' 318.***

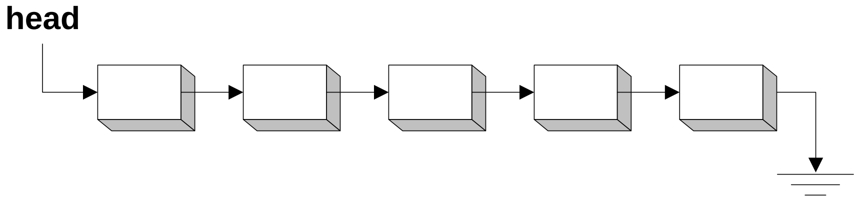
# רשימות מקושרות

עד עתה השתמשנו במבנה נתונים בעל גודל סטטי: **מערך**. כפי שראינו, המערך הוא בעל מספר איברים מקסימלי שלא תמיד מנוצל במלואו.

בעיה נוספת הקיימת במערכים היא הקושי בהוצאת ובהכנסת איברים חדשים למקום מסוים באמצע. לדוגמא, אם נרצה להכניס איבר חדש למערך ממוין, ונרצה שהוא יוכנס במקום המתאים, נצטרך להזיז את כל האיברים העוקבים לו מקום אחד קדימה - פעולה מאוד לא יעילה.

**רשימה מקושרת** (**Linked List**) היא מבנה נתונים שגודלו דינמי וגמיש: ניתן להוסיף או להוציא איברים עפ"י הצורך ללא צורך בהכרזה מראש על מספר איברים מקסימלי.

כמו כן פעולות הכנסה והוצאה מרשימה מקושרת הן פשוטות ויעילות יותר מאשר במערך. מבנה רשימה מקושרת:




רשימה זו מקושרת בכיוון אחד (singly linked list) : head הוא מצביע לאיבר הראשון ברשימה, וכל איבר ברשימה מכיל מצביע לאיבר הבא.

האיבר האחרון מצביע ל- NULL, והוא מסומן בתרשים ע"י חץ לאדמה.

כל איבר ברשימה נקרא **צומת** (**Node**) והוא מוגדר באופן הבא:

```
struct Node
{
	int data;
	struct Node *next;
};
```

המשתנה head מוגדר כך:

```
struct Node *head;
```

<u>הסבר</u>: צומת מוגדר כרשומה המכילה בנוסף לנתונים גם מצביע מסוג הרשומה עצמה. head הוא מצביע לצומת הראשון ברשימה.

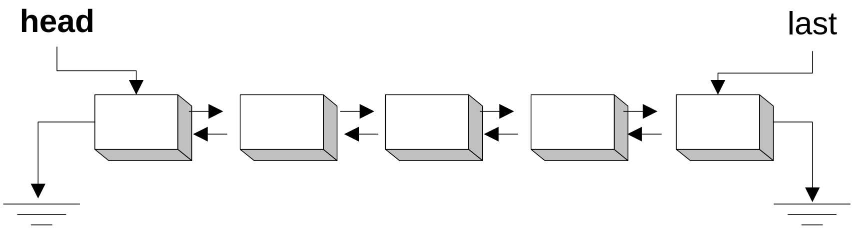
המשתנה next שלו יצביע לצומת הבא ברשימה, המצביע next של הצומת הבא יצביע לצומת הבא אחריו ברשימה וכן הלאה.


מרכז ההדרכה 2000 © כל הזכויות שמורות

## רשימות מסוגים נוספים

קיימות רשימות מקושרות השונות במבניהן:

<u>רשימה מקושרת דו-כוונית:</u>




במקרה זה קיימים שני מצביעים: המצביע head מצביע לתחילת הרשימה והמצביע last מצביע לסופה.

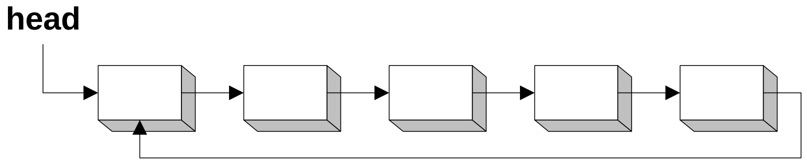
כל צומת ברשימה מכיל שני מצביעים - אחד לצומת הקודם ואחד לצומת הבא.

צומת הרשימה יוגדר כך:

```
struct Node
{
	int data;
	struct Node *prev;
	struct Node *next;
};
```

המצביע **prev** של הצומת הראשון מצביע ל- NULL (מסומן כמחובר לאדמה), וכך גם המצביע **next** של הצומת האחרון.

<u>רשימה מקושרת חד-כוונית מעגלית:</u>




הרשימה היא חד כוונית, כאשר האיבר האחרון מכיל מצביע לאיבר הראשון ולא ל- NULL. מבנה צומת ברשימה זו הוא כמו ברשימה החד-כוונית שלעיל.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## מימוש רשימה מקושרת בתכנית הספרייה

נראה כעת כיצד ממשים **רשימה מקושרת חד-כוונית**. לשם כך נחזור לתכנית הספרייה ונחליף את מבנה הנתונים של הספרים - מערך - ברשימה מקושרת.

ספר יוכנס לרשימה המקושרת עפ"י מספרו המזהה, כלומר הרשימה תהיה ממוינת תמיד על פיו. ניתן גם להוציא ספרים מהרשימה (תוך שחרור רשומת הספר).

תזכורת: רשומת הספר מוגדרת כך

```
/* Book type */
typedef struct
{
	char  *name;
	char  *writer;
	char  *publisher;
	float  cost;
	int    year;
	int    ID;
} Book;

typedef Book* BookPtr; /* a pointer to Book struct */
```

<u>הגדרת צומת ברשימה</u>

הגדרת טיפוס צומת, ListNode:

```
/* list node */
typedef struct ListNode_tag
{
	BookPtr      pbook;
	struct       ListNode_tag  *next;
} ListNode;
```

צומת הרשימה מכיל מצביע לרשומת ספר. הספר יוקצה דינמית בהכנסת צומת חדש לרשימה.

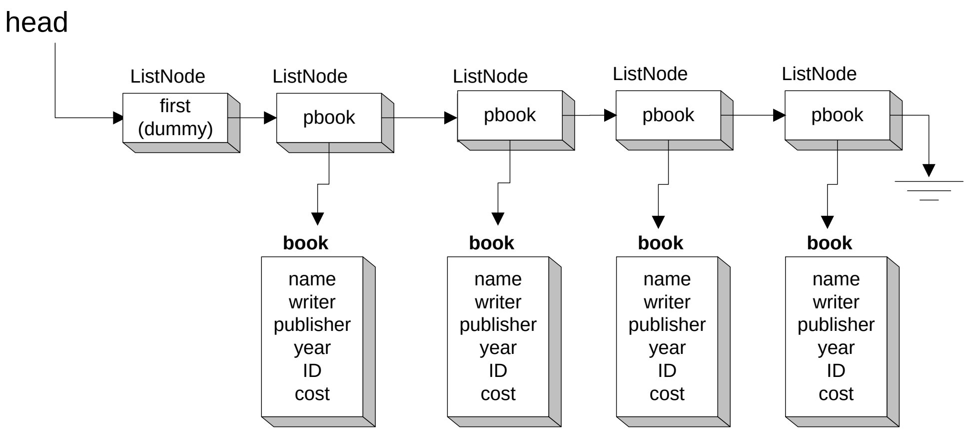
הגדרת צומת ראשון כצומת דמי (dummy) מפשטת את הפעולות על הרשימה:

```
ListNode  first;
```

צומת זה משמש כעוגן הקיים באופן תמידי. הגדרת מצביע לרשימה - head - מצביע ל- first :

```
ListNode  *head = &first;
```

התרשים הבא מראה את תמונת הרשימה:




יש לשים לב שברשימה, בניגוד למערך, אין הגבלה (מלבד מגבלת הזיכרון הפיזי של המערכת) על מספר האיברים שבה!

<u>הגדרת מבנה הרשימה באופן מודולרי</u>

לצורך מודולריות התכנית, רצוי להגדיר מבנה המתאר את הרשימה המקושרת.

מבנה זה יכלול את המצביע head לתחילתה, את הצומת הראשון, את מספר האיברים הנוכחי ברשימה ומידע נוסף - עפ"י הצורך.

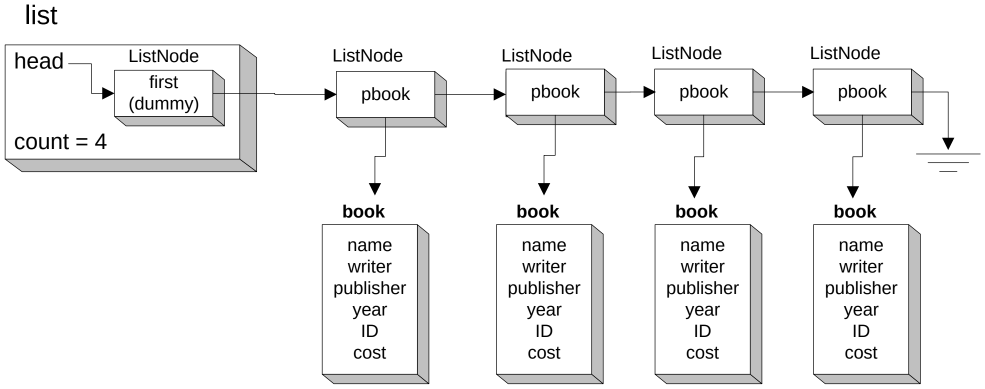
לדוגמא, עבור רשימת הספרים נגדיר את המבנה :

```
/* list structure */
typedef struct
{
        ListNode    *head;      /* head of the book list */
        ListNode    first;      /* first dummy element */
        int         count;
} List;
```

כעת התכנית המשתמשת יכולה להגדיר רשימה ע"י

```
List   list;
```

המבנה המלא של הרשימה מוצג בתרשים הבא :




אופן זה של הגדרת הרשימה מאפשר שימוש חוזר באותה תכנית לשימוש במספר רשימות.

לדוגמא, ניתן להגדיר מספר רשימות בתכנית אחת:

```
List   list1, list2, list3;
```

אתחול הרשימה

הרשימה המקושרת תאותחל ע"י פונקציה המקבלת מבנה מסוג List כפרמטר:

```
void list_init(List *list)
{
        list->first.next = NULL;
        list->head = &list->first;
        list->count = 0;
}
```

כעת, מתוך התכנית המשתמשת נוכל להגדיר רשימה ולהעביר את כתובתה כפרמטר לפונקציה לשם אתחולה:

```
void main()
{
        List   list;
        list_init(&list);
        ...
}
```

הוספת רשומת ספר לרשימה

פונקציה להכנסת רשומת ספר חדשה לרשימה:

```
void list_add_book(List *plist, BookPtr pbook)
{
        ListNode *new_node, *p;

        new_node = malloc(sizeof(ListNode));
        new_node->pbook = pbook;

        for(p=plist->head; p->next!=NULL ; p=p->next)
                if(new_node->pbook->ID < p->next->pbook->ID)
                        break;

        /* at this point p->next is the place to insert */
        new_node->next = p->next;
        p->next = new_node;
        plist->count++;
}
```

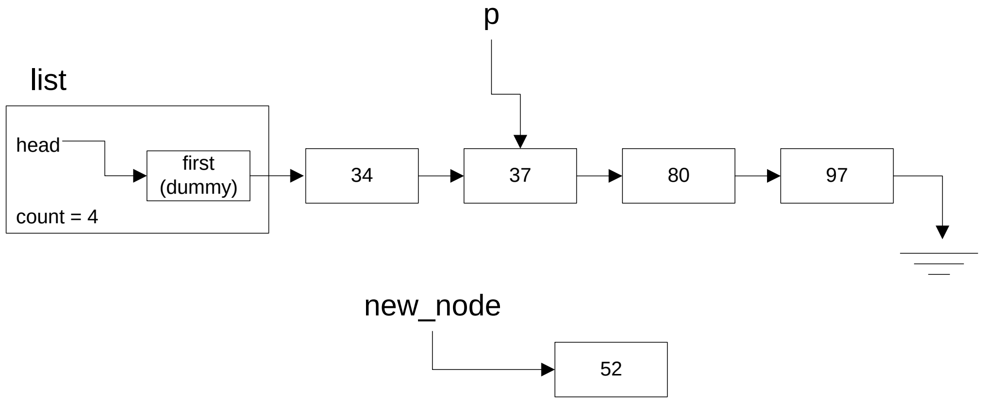
הסבר: new_node הוא מצביע לצומת החדש המוקצה מפורשות ע"י malloc(). לאחר הקצאתו, מוצב לשדה pbook שבו המצביע לספר:

```
        new_node = malloc(sizeof(ListNode));
        new_node->pbook = pbook;
```

בשלב הבא עוברים בלולאה על איברי הרשימה עד למציאת המקום המתאים להכנסה - עפ"י מספר הזהות:

```
for(p=list->head; p->next!=NULL ; p=p->next)
        if(new_node->pbook->ID < p->next->pbook->ID)
                break;
```

לאחר יציאה מלולאת ה- for האיבר העוקב ל- p צריך להיות האיבר החדש. לדוגמא, אם המספר המזהה של הספר החדש הוא 52, תרשים הרשימה לאחר סיום הלולאה :




בשלב זה מוכנסת הרשומה החדשה למקומה ע"י ההוראות

```
new_node->next = p->next;
p->next = new_node;
```

קוד התכנית המשתמשת לאתחול ולהוספת ספר לרשימה ייראה כך:

```
List   list;
list_init(&list);
...
pbook = malloc(sizeof(Book));
book_input(pbook);
list_add_book(&list, pbook);
```

<u>הוצאה (מחיקה) מהרשימה</u>

הוצאת רשומת ספר מהרשימה מבוצעת ע"י הפונקציה ()list_del_book עפ"י מספר מזהה נתון של ספר.

***עיין/י בקוד הפונקציה בעמ' 325-326.***

<u>פונקצית ההדפסה</u>

בפונקציה זו עוברים על כל איברי הרשימה (עד הגעה ל- NULL) ומדפיסים אותם ע"י קריאה לפונקציה ()book_output .

***עיין/י בקוד הפונקציה שבעמ' 327.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

<u>פונקציות החיפוש</u>

במימוש ע"י מערך, פונקציות החיפוש החזירו אינדקס של הרשומה שנמצאה במערך, ואם לא נמצאה החזירו ערך מיוחד (-1).

במימוש הנוכחי ע"י רשימה, הפונקציות מחזירות מצביע לרשומה שנמצאה, ואם לא נמצאה מוחזר NULL.

פעולות החיפוש:

- חיפוש עפ"י שם
- חיפוש עפ"י מספר זהות
- האם איבר עם מספר זהות נתון קיים ברשימה?

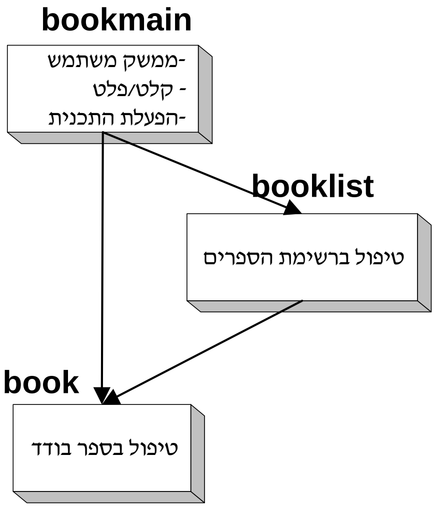
***עיין/י בקוד הפונקציות שבעמ' 327.***

## חלוקת התכנית לקבצים

תכנית הספרייה הכוללת מחולקת לקבצים באופן הבא:

- book.h, book.c - מודול המטפל בספר יחיד, מוגדר כמו קודם.
- booklist.h, booklist.c - מודול המטפל ברשימת ספרים.
- bookmain.c - מודול המכיל את הפונקציה הראשית וממשק המשתמש - דומה לגרסה הקודמת, תוך החלפת קריאות לפונקציות המערך באילו של הרשימה.

תרשים המודולים:




***קובץ הממשק של הרשימה booklist.h מובא בעמ' 328-329, כמו גם הפנייה לקבצי המקור של התכנית המלאה.***

# סיכום

- הקצאת זיכרון מפורשת ע"י המתכנת נקראת הקצאת זיכרון דינמית. היא מבוצעת ע"י פונקציות הספרייה stdlib.h :

  **malloc**() מבצעת הקצאת זיכרון עפ"י גודל זיכרון נתון

  **free**() משחררת זיכרון שהוקצה ע"י malloc()

- הקצאות זיכרון דינמיות מבוצעות מתוך מאגר שמור של זיכרון הנקרא **ערמה** (**heap**). מאגר זיכרון מסוג זה מסופק לכל תכנית בתחילת ריצתה ומאפשר לה יכולת הרחבת משאבי הזיכרון שלה.

- אחד השיפורים בניצולת הזיכרון בתכנית מושג ע"י שימוש במחרוזות בעלות אורך משתנה, שהזיכרון שלהן מוקצה דינמית בהתאם לאורכן.

- גם מבנים ניתנים להגדרה באופן דינמי: הגדרת מערך מצביעים למבנים במקום מערך מבנים משפרת משמעותית את ניצולת הזיכרון בתכנית.

  המבנים מוקצים עפ"י דרישה ע"י הקצאת זיכרון דינמי.

- ייעול נוסף של התכנית הוא שימוש במבנה נתונים דינמי: **רשימה מקושרת**. רשימה מקושרת, בניגוד למערך, היא מבנה נתונים שמספר איבריו ניתן להגדלה או להקטנה.

  איברי הרשימה נקראים צמתים. הצמתים מכילים מלבד המידע (רשומת הספר בדוגמא) מצביע לצומת הבא. הם אינם מסודרים באופן רציף בזיכרון, אלא משורשרים אחד לשני ע"י המצביעים.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבעמ' 330.***

# 13. קלט / פלט עם קבצים

- סוגי קבצים
- פתיחת קובץ
- קריאה / כתיבה לקובץ
- הוספת שמירה לקובץ לתוכנית הספרייה
- סיכום
- תרגילי סיכום


מרכז ההדרכה 2000 © כל הזכויות שמורות

# סוגי קבצים

כדי לשמור מידע בין ריצות עוקבות של התוכנה מבצעים שמירת נתונים לקבצים בדיסק. ניתן לשמור נתונים בשני סוגי פורמט:

**פורמט טקסט (ascii)** - הנתונים מאוחסנים בקובץ כתווים.

**פורמט בינרי** - הנתונים מאוחסנים בקובץ כפי שהם בזיכרון.

קיימים 3 שלבים בטיפול בקבצים:

1. פתיחת קובץ
2. קריאה/כתיבה לקובץ
3. סגירת הקובץ

בפתיחת קובץ מקבלים ממערכת ההפעלה מצביע מטיפוס **FILE**. טיפוס זה הוא רשומה ששדותיה מתארים תכונות שונות של הקובץ.

פונקציות הקלט/פלט לקבצים מקבלות כולן מצביע ל- **FILE** כפרמטר. הטיפוס **FILE** ופונקציות הקלט/פלט לקבצים מוגדרים בספרייה stdio.h.

קיימים 3 קבצי קלט/פלט תקניים הנפתחים ע"י מערכת ההפעלה בתחילת התכנית ומוגדרים להם 3 מצביעים בהתאם:

**stdin** - מצביע לקובץ הקלט התקני

**stdout** - מצביע לקובץ הפלט התקני

**stderr** - מצביע לקובץ השגיאה התקני


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

# פתיחת קובץ

ראשית מגדירים מצביע לקובץ:

```
FILE  *in_fp;     /* input file pointer */
FILE  *out_fp;  /* output file pointer */
```

קוראים לפונקציה ()**fopen** ומעבירים לה פרמטרים: שם קובץ לפתיחה ומוד הפתיחה

```
in_fp = fopen("file.dat", "rt");
out_fp = fopen("file.dat", "wt");
```

המחרוזת "rt" מציינת פתיחה במוד קריאה ובפורמט טקסט, המחרוזת "wt" מציינת פתיחה במוד כתיבה ובפורמט טקסט.

במידה ופעולת הפתיחה/סגירה נכשלת מוחזר NULL ע"י הפונקציה ()fopen.

דוגמא לפתיחת קבצים במוד בינרי:

```
in_fp = fopen("file.dat", "rb");
out_fp = fopen("file.dat", "wb");
```

האות b מציינת פורמט בינרי. קיימים מודי פתיחה נוספים - פתיחה לקריאה וכתיבה("rw") , פתיחה להוספה (append - "a") כאשר ברירת המחדל - אם לא צוין אחרת - היא פורמט קובץ טקסט.

# קריאה / כתיבה לקובץ

קיימים 2 פורמטים לאחסון נתונים בקובץ : פורמט **טקסט** ופורמט **בינרי**.

בפורמט טקסט הנתונים נשמרים עפ״י הייצוג המחרוזתי שלהם, כלומר בקוד ה- ASCII של התווים.

בפורמט בינרי לעומת זאת נשמר הנתון בדיוק כפי שהוא מיוצג בזיכרון.

לדוגמא, המספר ההקסה-דצימלי F123456 יאוחסן בקובץ בפורמט טקסט ע״י 7 תווי ה- ASCII המרכיבים אותו :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **70** | **49** | **50** | **51** | **52** | **53** | **54** | | |
| | | | | | | | | |

(כל משבצת מתארת בית בודד בקובץ). 70 הוא קוד ASCII של האות 'F', 49 הוא קוד ASCII של הספרה '1', 50 הוא קוד ה- ASCII של הספרה '2' וכן הלאה.

לעומת זאת, בפורמט בינרי המספר יישמר במערכת שבה שלם תופס 4 בתים כך :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **0F** | **12** | **34** | **56** | | | | | |
| | | | | | | | | |

**מסקנה** : פורמט בינרי הוא חסכוני יותר ואילו פורמט טקסט הוא נוח יותר מבחינת היכולת לקרוא ולהבין את הקובץ.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## קריאת וכתיבת טקסט לקובץ

הקריאה והכתיבה לקובץ טקסט דומה לפעולות המקבילות בקלט/פלט התקני: לפונקציות הקלט/פלט התקני מתאימות פונקציות קלט/פלט לקובץ עם האות f בתחילתן, כאשר פרמטר נוסף המועבר הוא המצביע לקובץ.

לדוגמא, המקבילות לפונקציות printf ו- scanf הן **fprintf** ו- **fscanf**:

```
fscanf(in_fp, "%d %s", &num, str);
fprintf(out_fp, "%d %s", num, str);
```

הפונקציות fputc ו-fgetc כותבות וקוראות תו בודד אל/מקובץ. ממשק הפונקציות:

```
int fputc( int c, FILE *fp);
int fgetc( FILE *fp);
```

***עיין/י בתכנית הדוגמא המובאת בעמ' 334 המעתיקה את הקובץ autoexec.bat לקובץ גיבוי autoexec.bak.***


מרכז ההדרכה 2000 © כל הזכויות שמורות

## קריאה וכתיבה בינריים לקובץ

כתיבה בינרית לקובץ מבצעת העתקה של תמונת הזיכרון של הנתונים בזיכרון לקובץ. קריאה בינרית מבצעת את הכיוון ההפוך. כתיבה וקריאה בינריים מבוצעות ע"י הפונקציות **fread()** ו-**fwrite()** :

```
size_t   fread( void *buffer,    size_t size, size_t count, FILE *fp );
size_t   fwrite( const void *buffer,     size_t size, size_t count, FILE *fp );
```

שתי הפונקציות מקבלות את הפרמטרים:

**buffer** - מצביע לחוצץ הנתונים המיועדים לכתיבה / לקריאה

**size** - גודלו של איבר בודד במערך בבתים

**count** - מספר האיברים : 1 לנתון בודד, למערך - מספר האיברים הנכתבים

**fp** - מצביע לקובץ

הפונקציה fwrite מעתיקה מהמקום בזיכרון ש- buffer מצביע עליו size*count בתים לקובץ. בדומה, fread מבצעת את הקריאה.

הערך המוחזר - מטיפוס size_t - משתי הפונקציות הוא מספר הבתים שנקראו / נכתבו. size_t הוא מטיפוס (typedef) שלם לא מסומן המציין גודל של נתון בזיכרון.

***תכנית הדוגמא שבעמ' 336 מדגימה כתיבה וקריאה של סוגי נתונים שונים לקובץ בפורמט בינרי.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## חיפוש בקבצים בינריים

אחת הפעולות השכיחות בטיפול בקבצים בינריים היא חיפוש של רשומה כלשהי.

שימוש נפוץ לחיפוש מעין זה קיים בתוכנות בסיסי נתונים, בביצוע שאילתות או בהפקת דוח״ות. בסעיף זה נראה כיצד לחפש רשומות בקובץ בינרי.

הפונקציה **fseek** מאפשרת להזיז את סמן הקריאה/כתיבה של הקובץ ממקומו הנוכחי למיקום מסויים בקובץ:

int **fseek**( FILE ***fp**, long ***offset***, int ***origin*** );

הפונקציה מקבלת 3 פרמטרים:

- **fp** - מצביע לקובץ
- **offset** - מספר הבתים להזזה ביחס ל- origin
- **origin** - מיקום התחלתי. ערך זה יכול להיות אחד מהשלושה:

SEEK_SET - התחלת הקובץ

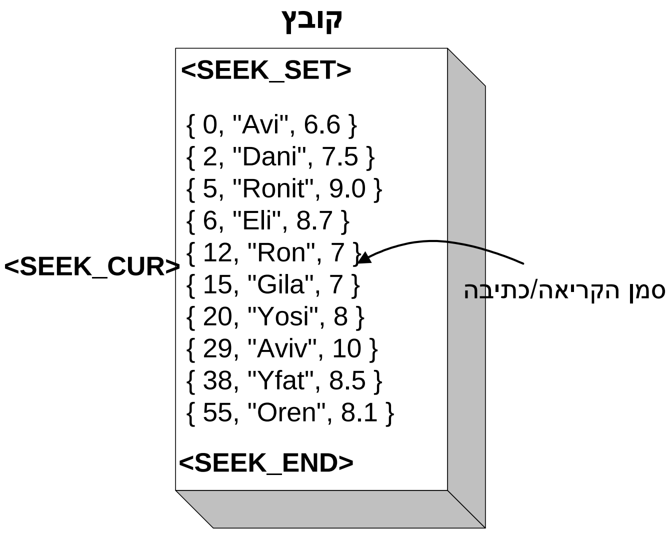
SEEK_CUR - המיקום הנוכחי של סמן הקריאה/כתיבה של הקובץ

SEEK_END - סוף הקובץ

לדוגמא, נתון קובץ בינרי של רשומות מטיפוס Student הכולל מספר מזהה, שם וממוצע ציונים:

```
typedef struct
{
	int		ID;
	String		name;
	double		average;
} Student;
```

בקובץ מערך רשומות סטודנטים ממויינות עפ״י מספר הזהות:



הקובץ כולל 10 רשומות. סמן הקריאה/כתיבה של הקובץ נמצא לאחר הרשומה החמישית ולפני הרשומה השישית. למשל, ניתן להזיז את הסמן לרשומה השלישית ע"י

```
fseek(file, 2*sizeof(Student), SEEK_SET);
```

הסבר: הפונקציה fseek מזיזה את סמן הקריאה/כתיבה ביחס לראשית הקובץ(SEEK_SET) למיקום 2*sizeof(Student), שהוא מקום התחלת הרשומה השלישית.

כמו כן, ניתן לקרוא את הרשומה מהקובץ ע"י **fread** ולהדפיס את ערכיה.

לדוגמא, הפונקציה הבאה מזיזה את הסמן לרשומה התשיעית (אינדקס 8), קוראת ומדפיסה את ערכי הרשומה:

```
void f(FILE *file)
{
	Student s;
	fseek(file, 8*sizeof(Student), SEEK_SET);
	fread(&s, sizeof(Student), 1, file); /* read record */
	printf("\nRecord %d name is %s, avg=%f",s.ID, s.name, s.average);
}
```

מודפס:

```
Record 38 name is Yfat, avg=8.500000
```

פונקציה לחיפוש בינרי בקובץ

הפונקציה הבאה מבצעת חיפוש בינרי של רשומה בקובץ. הפונקציה מקבלת כפרמטרים מצביע לקובץ ומספר מזהה לחיפוש:

```
void bin_search(FILE *file, int ID)
{
	int file_size;
	int min, max, avg;
	Student s;
	Boolean found;
```

- ראשית מחשבים את גודל הקובץ בבתים ע"י הזזת הסמן לסוף הקובץ, וקריאה לפונקציה **ftell**, המחזירה את מיקום הסמן ביחס לתחילת הקובץ בבתים:

```
	fseek(file, 0, SEEK_END);
	file_size = ftell(file);
```

- כעת מתחיל תהליך החיפוש: מאתחלים את שני המשתנים, min ו- max:

```
	min=0;
	max=file_size/sizeof(Student) - 1;
	printf("\nSearching for record %d...reading records ", ID);
```

- בלולאה הבאה מבצעים את החיפוש הבינרי: בכל חזרה, מחשבים את החציון, avg, ומשווים את ערך המספר המזהה שמחפשים עם זה של הרשומה שבחציון. בהתאם לכך ממשיכים לחפש בחציון העליון, או התחתון:


מרכז ההדרכה 2000 © כל הזכויות שמורות

```
	for(found=FALSE; min <= max; )
	{
		avg = (max + min)/2;
		fseek(file, avg*sizeof(Student), SEEK_SET); /* move to the record avg */
		fread(&s, sizeof(Student), 1, file); /* read record */
		printf("%d ", s.ID);
		if(s.ID==ID)
		{
			printf("\nRecord %d found: name is %s",s.ID, s.name);
			found = TRUE;
			break;
		}
		else
			if(ID < s.ID)
				max = avg - 1;
			else
				min = avg + 1;
	}
	if(!found)
		printf("\nRecord %d not found!", ID);
}
```

בכל חזרה מזיזים את min או max לכיוון החציון (עפ"י תוצאת ההשוואה) עד למציאת הרשומה, או עד ש- min ו- max בעלי ערך זהה, מה שמציין שהרשומה שמחפשים לא קיימת בקובץ.

התכנית המשתמשת

***התכנית המשתמשת מובאת בעמ' 340.***

 מרכז ההדרכה 2000 © כל הזכויות שמורות

# הוספת שמירה לקובץ לתוכנית הספרייה

**בעיה**: הנתונים שהמשתמש מקליד בכל הרצה של תכנית הספרייה שכתבנו הולכים לאיבוד בסופה. בכל הרצה של התכנית מתחילים מאפס.

כיצד נשמור את הנתונים בין ריצות עוקבות של התכנית?

**פתרון**: נשמור את הנתונים לקובץ, ובהרצת התכנית נקרא אותם מתוכו.

## הוספת פונקציות שמירה (Save)

נוסיף למודול book פונקציה לשמירת רשומת ספר בודד לקובץ בפורמט טקסט:

```
void book_save_to_file(FILE *fp, BookPtr pbook)
{
	fprintf(fp, "\n%s\n%s\n%s\n%d\n%f\n%d",
		pbook->name,
		pbook->writer,
		pbook->publisher,
		pbook->year,
		pbook->cost,
		pbook->ID);
}
```

נתוני הספר נכתבים לקובץ ע"י fprintf נתון בשורה. למודול booklist נוסיף פונקציה לשמירת כלל הרשימה לקובץ:

```
void  list_save_to_file(List *plist, char *file_name);
```

***קוד הפונקציה והסבר מובאים בעמ' 342.***

## פונקציות טעינה מהקובץ

מאחר שנתוני התכנית נשמרים לקובץ, יש להגדיר פונקציה שקוראת אותם מהקובץ למשתנים בזיכרון.

לפני כן, נגדיר פונקצית עזר לקריאת שורה מקובץ:

```
char *getline(FILE *fp)
{
	char  temp[80];

	fgets(temp, 80, fp);
	temp[strlen(temp)-1] = '\0'; /* instead of the '\n' inserted by fgets */
	return strdup(temp);
}
```

<u>הסבר</u>: הפונקציה קוראת שורת טקסט מהקלט ע"י fgets ומחזירה שיכפול שלה - ע"י strdup - לפונקציה הקוראת.

לפני ההחזרה, הפונקציה מתקנת בעיה שנגרמת ע"י הפונקציה fgets(): השורה שנקראה מהקלט מכילה גם את תו השורה החדשה '\n' כתו אחרון במחרוזת (לפני תו סיום המחרוזת '\0').

כדי למחוק את התו האחרון '\n' מציבים את תו סיום המחרוזת במקומו:

```
temp[strlen(temp)-1] = '\0';
```

הפונקציה הקוראת רשומת ספר מהקובץ מוגדרת כך במודול book :

```
void book_read_from_file(FILE *fp, BookPtr pbook)
{
	String temp;
	pbook->name = getline(fp);
	pbook->writer = getline(fp);
	pbook->publisher  = getline(fp);
	fscanf(fp, "%d", &pbook->year);
	fscanf(fp, "%f", &pbook->cost);
	fscanf(fp, "%d", &pbook->ID);
	fgets(temp, MAX_STR_LEN, fp); /* to read up to the end of line */
}
```

במודול booklist מוגדרת הפונקציה הטוענת את כל רשומות הספרים מהקובץ לרשימה המקושרת:

```
void  list_load_from_file(List *plist, char *file_name);
```

***קוד הפונקציה והסבר מובאים בעמ' 343.***

מרכז ההדרכה 2000 © כל הזכויות שמורות

## הפונקציה הראשית ()main

הקריאה לפונקצית השמירה לקובץ מבוצעת בתכנית הראשית כתגובה לבחירת המשתמש באופציה s - שמירה לקובץ - שהוספה לתפריט:

```
void menu()
{
	printf("\n\n\n\n\n");
	puts("Book program main menu");
	.
	.
	puts("s. Save books records to file");
	puts("x. Exit");
	puts("\n\n\n\n\n\n\n\n");
}
```

הקריאה לפונקצית הטעינה לעומת זאת, נקראת תמיד בתחילת התכנית בכדי לטעון את הנתונים הקיימים:

```
int main()
{
	char  *DATA_FILE_NAME = "book.out";
	...
	list_init(&list);

	list_load_from_file(&list, DATA_FILE_NAME);
	for(;;)  /* forever */
	{
		...
		switch(c)
		{
			.
			.
		case 's': /* save to file */
			list_save_to_file(&list,DATA_FILE_NAME);
			pause();
			break;
		default:
			break;
		}
	}
	return 0;
}
```

מובן שלצורך כך יש להכריז על 2 הפונקציות הנ"ל בקובץ הממשק booklist.h.

## שמירה ביציאה מהתכנית

המשתמש יכול כעת לבצע שמירה יזומה לתכנית, בכל רגע שירצה. אולם מה אם הוא ישכח ויבקש לצאת מהתכנית לפני ששמר את השינויים האחרונים שביצע?

כאשר המשתמש מבקש לסיים את התכנית, נהוג לבדוק אם הוא ביצע שינויים במבני הנתונים מאז השמירה האחרונה ואם כן, לשאול אותו האם ברצונו לבצע שמירה. עפ"י החלטתו מבוצעת השמירה.

**שאלה**: כיצד נדע בבקשת היציאה אם בוצע שינוי בתכנית מאז השמירה האחרונה?

**תשובה**: נחזיק משתנה בוליאני - בעל ערך "אמת" או "שקר" - שיציין בכל רגע נתון האם נתוני התכנית שונו מאז השמירה האחרונה או לא.

ערכו של הנתון יהיה בתחילת התכנית "שקר", ולאחר כל ביצוע שינוי - קרי הוספת ספר או הסרת ספר - הוא ישונה ל- "אמת".

לאחר שמירת הנתונים לקובץ שוב יוצב למשתנה ערך "שקר".

נגדיר משתנה בוליאני בטיפוס **הרשימה** שיציין האם נתוני הרשימה שונו מאז השמירה האחרונה לקובץ:

```
typedef struct
{
        ListNode   *head;        /* head of the book list */
        ListNode   first;        /* first dummy element */
        Boolean    changed_fl; /* list change flag */
        int        count;
} List;
```

ערכו של המשתנה מאותחל ל- FALSE ומעודכן בכל שינוי במבנה הנתונים.

***עיין/י בקוד הפונקציות בהן הוא מעודכן בעמ' 345-347.***

# סיכום

- כדי לשמור את הנתונים שהמשתמש עורך בתכנית בין ריצות עוקבות שלה, יש לשמור אותם לקובץ.
- קיימים 2 פורמטים לכתיבה/קריאה ל/מקובץ:
  - פורמט בינרי
  - פורמט טקסט

בפורמט בינרי הנתונים נשמרים לקובץ בדיוק כפי שהם מאוחסנים בזיכרון. בפורמט טקסט הנתונים מפורקים לתווים והם נשמרים בייצוג התווי שלהם.

- קיימים מספר שלבים בשמירת / שחזור נתונים אל/מקבצים:
  - פתיחת הקובץ במוד המתאים (קריאה/כתיבה, טקסט/בינרי)
  - ביצוע פעולות הקריאה/כתיבה
  - סגירת הקובץ
- הפונקציות לפתיחת וסגירת קובץ הן **fopen** ו- **fclose** בהתאמה.
- הפונקציות לקריאה וכתיבה בינריים הן **fread()** ו- **fwrite()**. הפונקציות לכתיבה וקריאה במוד טקסט הן פונקציות מקבילות לפונקציות הקלט/פלט התקניות בתוספת האות f בתחילתן: **fscanf(), fprintf(), fgetchar()** וכו'.
- הפונקציה **fseek** משמשת להזזת סמן הקריאה/כתיבה של הקובץ למיקום מסויים. הפונקציה **ftell** מחזירה את מרחק הסמן מתחילת הקובץ בבתים.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבסוף פרק זה.***

# 14. טיפוסי נתונים מופשטים (ADT)

- טיפוסים מופשטים
- מדד ליעילות: סיבוכיות
- מבנה יישום ADT
- רשימה מקושרת
- מחסנית
- תור
- עץ בינרי
- סיכום


מרכז ההדרכה 2000 © כל הזכויות שמורות

# טיפוסים מופשטים

הכרנו את הטיפוסים הבסיסיים הקיימים בשפת C : שלם (int), ממשי (float), תו (char), מחרוזת ([] char), מערך וכן תתי-טיפוסים שלהם.

לעתים, הטיפוסים הבסיסיים אינם מספיקים ואנו מגדירים טיפוסים חדשים.

לדוגמא, בכדי להגדיר טיפוס מחרוזת נוח, השתמשנו בהוראה typedef :

```
typedef char String[256];
```

באופן זה יצרנו טיפוס מחרוזת חדש המבוסס על מערך בגודל 256 תווים.

בדומה, כאשר רצינו לייצג רשימה מקושרת (ראה/י פרק 12, "**הקצאת זיכרון דינמית ורשימות מקושרות**") הגדרנו טיפוס רשימה ע"י

```
typedef struct
{
        ListNode    *head;        /* head of the book list */
        ListNode    first;        /* first dummy element */
        Boolean     changed_fl; /* list change flag */
        int         count;
} List;
```

וכן הגדרנו טיפוסים ופונקציות לטיפול בצמתי הרשימה.

**טיפוסי נתונים מופשטים** (**ADT**, Abstract Data Types) הם טיפוסים המורכבים מטיפוסים בסיסיים יותר וכוללים גם פונקציונליות לטיפול בנתונים.

אנחנו נכיר בפרק זה את הטיפוסים המופשטים הבאים:

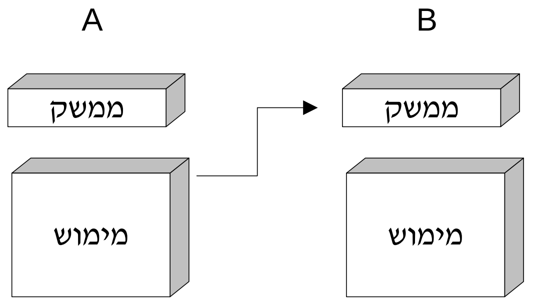
- **רשימה מקושרת כללית**
- **מחסנית**
- **תור**
- **עץ בינרי**

## הפרדה בין מימוש לממשק

העיקרון החשוב ביותר ביצירת טיפוסי נתונים מופשטים הוא הפרדה בין ממשק הטיפוס, כלומר אבטיפוסי הפונקציות, הטיפוסים והקבועים שהוא מייצא לבין מימושם בפועל.

עקרון זה חשוב מאוד לצורך פיתוח תוכנה באופן מודולרי: התלות בין מודולים שונים במערכת צריכה להיות מינימלית.

למשל, אם מודול A משתמש במודול B, הוא אינו צריך לדעת כיצד ממומש מודול B:

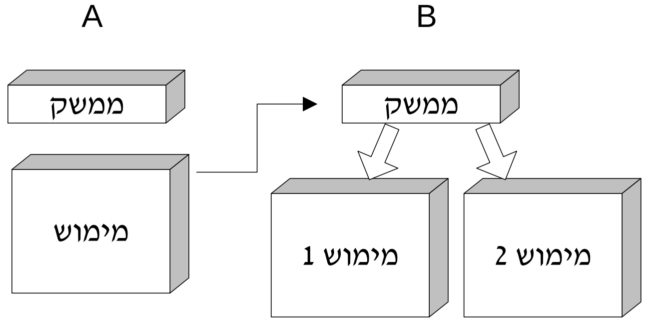



יש לשים לב לכך שהמהדר אינו יודע את פרטי המימוש בזמן ההידור: הוא מבצע את בדיקות ההידור רק עפ"י הממשק.

לדוגמא, בהידור המודול A, המהדר בודק שימוש בשמות פונקציות קיימות, שימוש בטיפוסים קיימים, טיפוסי פרמטרים מתאימים לפונקציות ושימוש בקבועים.

כל הבדיקות הן עפ"י קובץ הממשק (B.h) של המודול B (וכמובן גם עפ"י קובץ הממשק של A עצמו, A.h).

יתרון חשוב בהפרדה בין ממשק ומימוש של מודול מסוים הוא היכולת להחליף את המימוש של המודול בשלב מסוים, אם מסיבות של שיפור יעילות התכנית או עקב תיקוני שגיאות, וזאת מבלי שיהיה צורך לשנות את המודול המשתמש:




<u>הסתרת מידע</u>

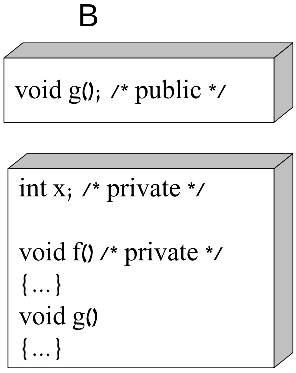
**הסתרת מידע** היא הטכניקה שבה מפרידים בין הממשק למימוש. אם מודול A משתמש במודול B, מודול B **מייצא** את כל המרכיבים (פונקציות, טיפוסים וקבועים) המהווים את הממשק אליו, ומצד שני **מסתיר** את מרכיבי המימוש (מבני נתונים, פונקציות פנימיות, משתנים).

קיימות שפות התומכות באופן מובנה בהסתרת מידע, בעיקר שפות לתכנות מונחה עצמים.

לדוגמא, שפות ++C ו- Java מאפשרות להגדיר אילו חלקים הם **פרטיים** (private) במודול (מחלקה) ואילו **ציבוריים** (public).

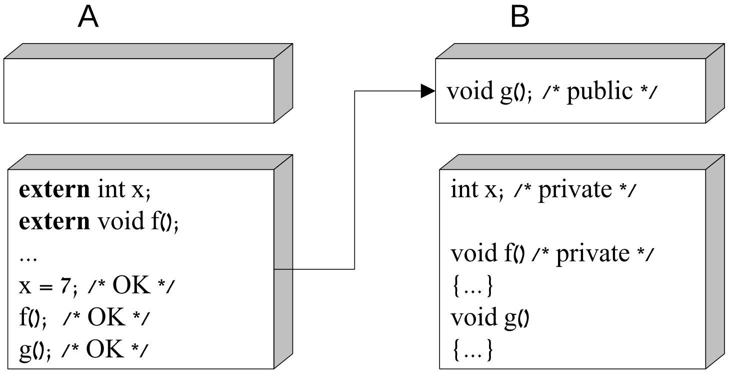
בשפת C האפשרויות מוגבלות יותר, אך ניתן בכל זאת להסתיר מרכיבים מסוימים:

- פונקציה או משתנה גלובלי שאינם מוכרזים בקובץ הממשק אינם מיוצאים למודולים אחרים, ובכך הם למעשה פרטיים:




כעת למודול A גישה רק לפונקציה ()g שב- B, והוא אינו "רואה" את המשתנה הגלובלי x ואת הפונקציה ()f, מכיוון שהם פרטיים.

אולם מודול A יכול לבצע ייצוא עצמאי ע"י הוראת **extern** ובכך לעקוף את קובץ הממשק של B:

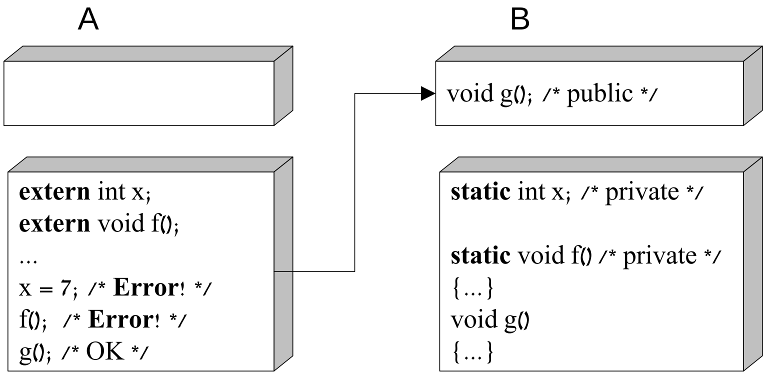



הערה: בייצוא פונקציה ההוראה **extern** היא אופציונלית, כלומר אבטיפוס הפונקציה הוא כשלעצמו ייצוא חוקי.

בדוגמא הנ"ל ניתן לרשום ייצוא עבור הפונקציה ()f גם כך:

```
void f();
```

- בכדי למנוע את האפשרות של ייצוא עצמאי ע"י **extern** במודולים האחרים, ניתן לציין משתנים ופונקציות כ**סטטיים** **(static)** במודול B:




המציין **static** גורם למשתנה גלובלי או לפונקציה להיות מקושרים באופן פנימי בלבד למודול שבו הוגדרו.

השגיאות במודול A יתקבלו ע"י **המקשר** **(linker)** בזמן הקישור של A, והן יהיו "לא נמצא המשתנה הגלובלי x" ו- "לא נמצאה הפונקציה הגלובלית f()".

# *מדד ליעילות: סיבוכיות*

**סיבוכיות כפרמטר ביצועי**

משווים בין סוגי מבני נתונים ואלגוריתמים עפ"י **סיבוכיות** הביצוע של פעולות שונות עליהם, כלומר עפ"י משך הזמן וכמות הזיכרון שהן צורכות. הסיבוכיות מצויינת ע"י סדר גודל בלבד.

נהוג לציין את הסיבוכיות ע"י האות **O** (קיצור של Order) בצירוף סדר הסיבוכיות בסוגריים: **O(<סדר של n>)**.

דוגמאות:

- סיבוכיות לינארית מצויינת ע"י **O(n)** כקיצור ל- "**Order of n**", כאשר **n** הוא מספר האיברים שבמבנה הנתונים. סדר סיבוכיות זה כולל את הביטויים הלינאריים של **n**, לדוגמא:

*n*
*n/2*
*3n*
*4n*
*23n + 255*
*2000000n + 60000*

כלומר, בחישוב הסיבוכיות לא מתחשבים בקבועים ובגורמים המעורבים בביטוי, אלא רק בסדר של **n** בביטוי.

- הסיבוכיות **$O(n^2)$** מתארת סדר סיבוכיות ריבועי של n, לדוגמא:

$5n^2$
$3n^2 + 2n + 5$

- אם מספר הפעולות שדורש חישוב מסויים אינו תלוי ב- n, הסיבוכיות היא מסדר קבוע והיא מצויינת ע"י **O(1)**. למשל:

*23*
*0*
*1000000000*

***בטבלה שבעמ' 354 מובאים סדרי הסיבוכיות הקיימים.***

## סוגי סיבוכיות

קיימים שני סוגי סיבוכיות: סיבוכיות **זמן** וסיבוכיות **מקום**.

### סיבוכיות זמן

סיבוכיות זמן מתארת את סדר הגודל של כמות הזמן הנדרשת לביצוע פעולה נתונה על מבנה נתונים נתון.

דוגמאות:

**1.** סיבוכיות חיפוש איבר במערך בעל n איברים היא מסדר מספר האיברים במערך **O(n)**. במערך ממויין, ניתן לייעל את הפעולה ל- O(log n) ע"י ביצוע חיפוש בינרי.

**2.** הכנסת איבר באמצע מבנה הנתונים - לא כולל חיפוש מיקום ההכנסה - היא מסיבוכיות שונה בין מערך ורשימה: במקרה של **מערך** היא O(n) מכיוון שמתחייבת הזזה של כל האיברים העוקבים מקום אחד קדימה (בממוצע n/2 איברים), בעוד שעבור **רשימה מקושרת** הכנסת איבר כוללת מספר קבוע (וקטן) של פעולות קישור הצומת החדש. מציינים סיבוכיות זו ע"י **O(1)** , כלומר מסדר גודל קבוע.

**3.** סיבוכיות הזמן של מיון מערך מגודל n בשיטת **מיון בועות** היא $O(n^2)$: אנו עוברים n פעמים על איברי המערך בחזרה החיצונית, ובחזרה הפנימית על n (פחות אינדקס החזרה החיצונית, שלא משנה את סדר גודל הסיבוכיות) - סה"כ סדר $n^2$.

4. **מיון מהיר** (**Quick Sort**) היא שיטה הממיינת באופן רקורסיבי את המערך. סיבוכיות מיון זה היא במקרה הממוצע O(n log n), ובמקרה הגרוע $O(n^2)$.

### סיבוכיות מקום

סיבוכיות מקום מתארת את סדר הגודל של כמות הזיכרון הנדרשת לביצוע פעולה נתונה על מבנה נתונים נתון.

דוגמאות:

**1.** מיון איברי המערך בשיטת **מיון בועות** אינה צורכת כמות גדולה של זיכרון: בכל חזרה שבה מבצעים החלפה מקצים זיכרון למשתנה זמני יחיד לצורך ההחלפה, כלומר איבר יחיד נדרש בכל חזרה, לכן הסיבוכיות היא O(1), כלומר מסדר קבוע.

**2.** סיבוכיות המקום בשיטת **מיון מהיר** היא O(log n) מכיוון שהיא עושה שימוש במחסנית הקריאות לצורך הרקורסיה.


מרכז ההדרכה 2000 © כל הזכויות שמורות

## השוואה בין מערך ורשימה

הטבלה הבאה משווה בין מערך ורשימה מבחינת סיבוכיות <u>הזמן</u> בביצוע פעולות שונות על איבריהם:

| **<u>פעולה</u>** | **<u>מערך</u>** | **<u>רשימה</u>** |
|---|---|---|
| גישה אקראית | O(1) | O(n) |
| הכנסה / הוצאה באמצע | O(n) | O(1) |
| הכנסה / הוצאה בסוף או בהתחלה | O(1) | O(1) |
| חיפוש | O(n) | O(n) |
| מיון בשיטת בועות | $O(n^2)$ | $O(n^2)$ |
| מיון מהיר (Quick sort) | O(n log n) | O(n log n) |

# מבנה יישום ADT

## קובץ הגדרות כללי

כאשר כותבים תוכנה הכוללת מספר רב של מודולים, מקובל להציב הגדרות בסיסיות משותפות בקובץ יחיד, ולהכלילו בכל המודולים.

דוגמאות להגדרות כאלו הן הגדרת מחרוזת כמערך, טיפוס בוליאני ועוד.

לדוגמא, נגדיר קובץ בשם base.h באופן הבא:

```
/* base.h - basic type and constant definitions */
#ifndef BASE_H
#define BASE_H

/**************** TYPES ********************************/
/* a boolean type */
typedef enum {FALSE=0, TRUE=1} Boolean;

/* a String type */
typedef char String[256];

#endif
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

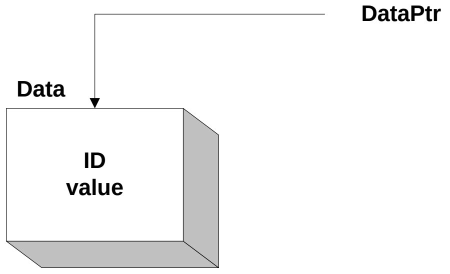
# טיפוס איבר כללי Data

בטיפוסי הנתונים המופשטים נטפל באיבר מסוג כללי שיוגדר כך:

```
/* Data type */
typedef struct
{
	int	ID;
	float	value;
} Data;

typedef Data *   DataPtr;
```

הטיפוס **Data** - המקביל לטיפוס Book מהפרקים הקודמים - ישמש כשם טיפוס כללי לנתונים כלשהם: בדוגמא זו, המבנה **Data** מכיל שני נתונים - שלם וממשי.

- השלם מציין מספר מזהה כלשהו והמספר הממשי מציין ערך.
- כמו כן הגדרנו טיפוס שימושי, **DataPtr** , כמצביע לטיפוס **Data**:

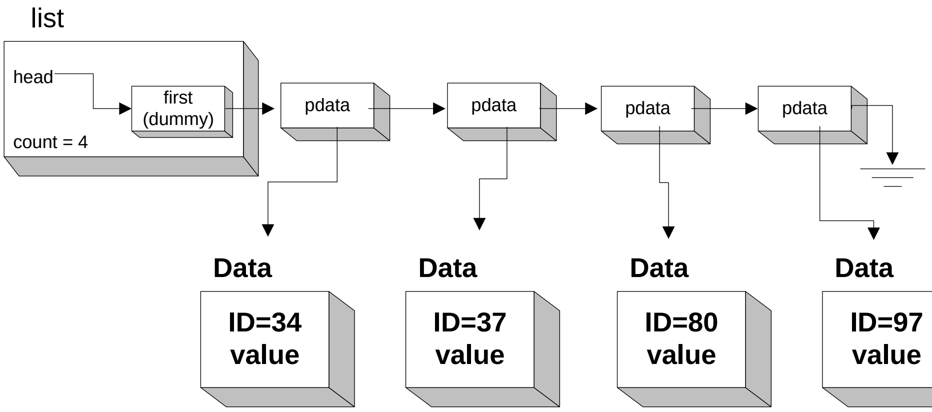



***קבצי הממשק והמימוש מובאים בעמ' 356-357.***

# רשימה מקושרת

בפרק 12, "**הקצאת זיכרון דינמית ורשימות מקושרות**", הכרנו את מבנה **הרשימה המקושרת** ומימשנו באמצעותה את רשימת הספרים בתכנית הספרייה.

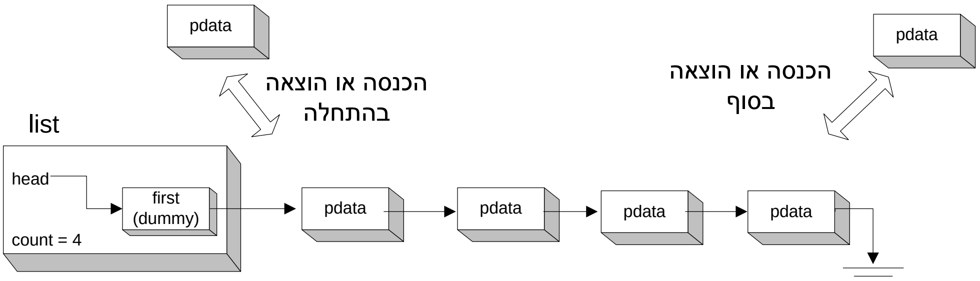
בסעיף זה נגדיר רשימה מקושרת כללית, המתאימה לא רק לספרים - אלא לטיפול ברשומת Data כללית:




הממשק הנדרש מרשימה מקושרת כללית:

- אתחל רשימה (list_init)
- נקה ושחרר את איברי הרשימה (list_clear)
- האם הרשימה ריקה? (list_is_emty)
- הכנס בסוף (list_push_back)
- הכנס בהתחלה (list_push_front)
- הוצא מהסוף (list_pop_back)
- הוצא מההתחלה (list_pop_front)
- הוסף לרשימה עפ"י פונקצית השוואה של האיבר (list_add)
- האם איבר נתון קיים ברשימה? (list_exist)
- הדפס את הרשימה (list_print)

תרשים פעולות ההכנסה וההוצאה :




## קובץ הממשק, list.h

קובץ הממשק מובא בעמ' 359.

## קובץ המימוש, list.c

קובץ המימוש מובא בעמ' 360-363, עפ"י חלוקה לפונקציות פרטיות וציבוריות, כלומר, פונקציות שאינן מיוצאות וכאלו שמיוצאות.

תכנית הבדיקה מובאת בעמ' 363-365.

## סיבוכיות זמן פעולות על הרשימה המקושרת

סיבוכיות הרשימה המקושרת היא מסדר גודל קבוע (O(1)) עבור פעולות הכנסה והוצאה בתחילת הרשימה.

הכנסה והוצאה מסופה דורשות מעבר על כלל צמתי הרשימה עד להגעה לסופה, ולכן גם הן מסיבוכיות O(n) (ראה/י תרגיל להלן).

הכנסת איבר באופן ממוין דורשת, בממוצע, מעבר על n/2 איברים, ולכן גם היא מסיבוכיות O(n).

הטבלה הבאה מציגה את סיבוכיות הפעולות על הרשימה המקושרת:

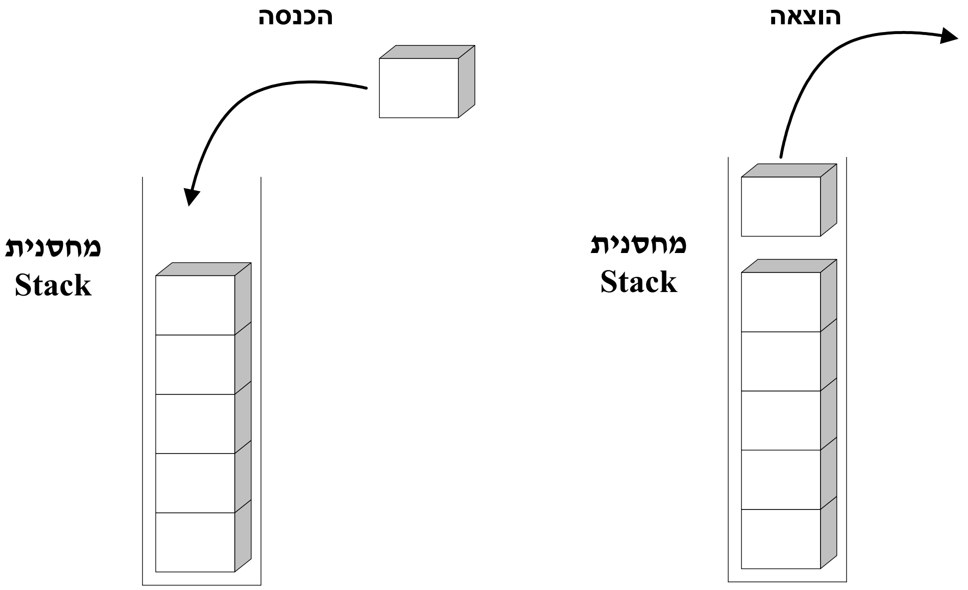
| פעולה | סיבוכיות |
|---|---|
| אתחול | O(1) |
| ניקוי | O(n) |
| הכנסה באופן ממוין | O(n) |
| הכנסה/הוצאה מתחילת הרשימה | O(1) |
| הכנסה/הוצאה מסוף הרשימה | O(n) |
| בדיקה אם הרשימה ריקה | O(1) |
| האם איבר מסוים קיים ברשימה | O(n) |
| הדפסה | O(n) |

תרגול

***קרא/י סעיף זה בספר ובצע/י את תר' 3-1 שבעמ' 366.***

# מחסנית

**מחסנית** (**Stack**) היא מבנה נתונים שאיבריו נשלפים בסדר ההפוך מזה שבו הוכנסו:




- הכנסה - איבר חדש מוכנס בראש המחסנית
- הוצאה - איבר מוצא מראש המחסנית

סדר זה של הכנסת והוצאת נתונים נקרא **LIFO** (**L**ast **I**n **F**irst **O**ut) - האיבר האחרון שהוכנס מוצא ראשון. שימושים נפוצים למחסנית הם ייצוג פונקציות בזמן ריצה (**מחסנית הקריאות**) וניתוח תחבירי במהדרים.

בכדי לממש מחסנית של איברים מסוג מצביע לטיפוס Data, נגדיר מבנה Stack באופן הבא:

```
#define STACK_MAX_SIZE 500

/* stack structure */
typedef struct
{
	DataPtr	array[STACK_MAX_SIZE];
	int		count;
} Stack;
```

הסבר: המחסנית מוגדרת כמערך של מצביעים לטיפוס Data, DataPtr. המשתנה count מציין את מספר האיברים הקיימים בפועל במחסנית.

### פעולות על המחסנית

נגדיר את הפעולות הבאות על המחסנית:

- אתחל מחסנית (stack_init)
- שחרר/נקה את המחסנית (stack_clear)
- הכנס למחסנית (stack_push)
- הוצא מהמחסנית (stack_pop)
- החזר איבר עליון (stack_top)
- האם המחסנית ריקה? (stack_is_empty)
- האם המחסנית מלאה? (stack_is_full)
- הדפס את המחסנית (stack_print)

הפונקציה להחזרת האיבר העליון (stack_top) מחזירה את האיבר שבראש המחסנית **מבלי להוציאו**.

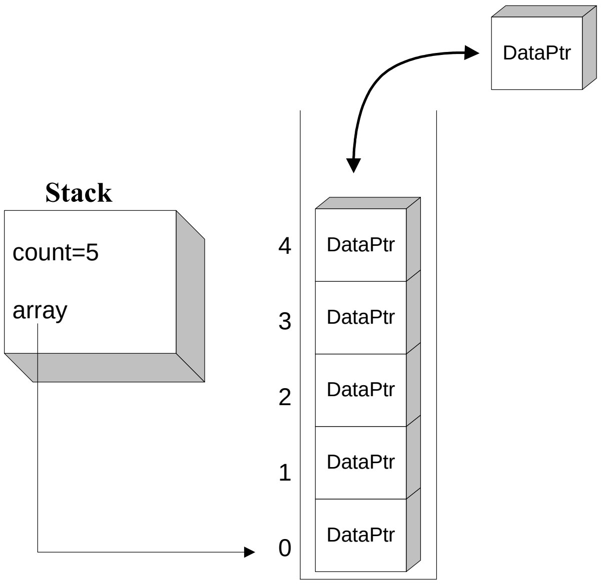
ממשק הפונקציות:

```
void      stack_init(Stack *pstack);
void      stack_clear(Stack *pstack);
void      stack_push(Stack *pstack, DataPtr pdata);
DataPtr   stack_pop(Stack *pstack);
DataPtr   stack_top(Stack *pstack);
Boolean   stack_is_empty(Stack *pstack);
Boolean   stack_is_full(Stack *pstack);
void      stack_print(Stack *pstack);
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## מימוש המחסנית ע"י מערך

אנו נראה כעת מימוש של פעולות המחסנית תוך שימוש במערך לשמירת האיברים:




בשלב מאוחר יותר נראה מימוש שונה של המחסנית ע"י רשימה מקושרת, וזאת **מבלי לשנות את ממשק הפעולות על המחסנית**.

הפונקציה stack_init מאתחלת את המחסנית:

```
void stack_init(Stack *pstack)
{
	pstack->count = 0;
}
```

הפונקציה stack_clear מנקה ומשחררת את נתוני המחסנית:

```
void stack_clear(Stack *pstack)
{
	int i;
	for(i=0; i<pstack->count; i++)
		data_clear(pstack->array[i]);
	pstack->count = 0;
}
```

הפונקציה stack_push דוחפת איבר לראש המחסנית (כולל בדיקה שהמחסנית לא מלאה):

```
void stack_push(Stack *pstack, DataPtr pdata);
```

הפונקציה stack_pop מוציאה איבר מראש המחסנית ומחזירה אותו (אם המחסנית ריקה מוחזר NULL):

```
DataPtr  stack_pop(Stack *pstack);
```

הפונקציה stack_top מחזירה את האיבר שבראש המחסנית, מבלי להוציא אותו (אם המחסנית ריקה מוחזר NULL):

***DataPtr stack_top(Stack *pstack);***

הפונקציה stack_is_empty מחזירה ערך בוליאני - האם המחסנית ריקה?

***Boolean stack_is_empty(Stack *pstack);***

הפונקציה stack_is_full מחזירה ערך בוליאני - האם המחסנית מלאה?

***Boolean stack_is_full(Stack *pstack);***

הפונקציה stack_print עוברת ומדפיסה את כל איברי המחסנית, החל מהתחתית :

***void stack_print(Stack *pstack);***

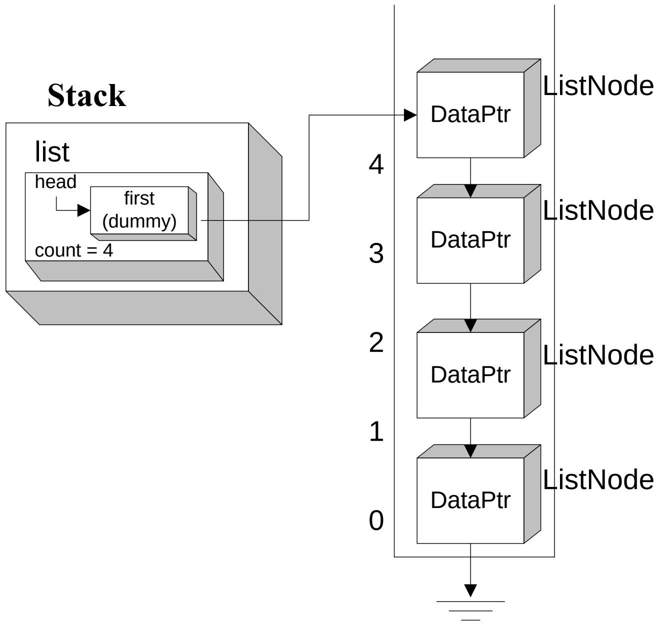
***עיין/י בקוד הפונקציות המלא בעמ' 369-370.***

***פונקצית הבדיקה מובאת בעמ' 370-372.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*

## מימוש המחסנית ע"י רשימה

מבנה הנתונים **מחסנית** אינו מחייב מימוש מסוים - בדוגמא הקודמת, מימשנו את המחסנית ע"י מערך בעל גודל קבוע, אך ניתן למשל לממש אותה ע"י **רשימה מקושרת**:




מבנה המחסנית מכיל כעת משתנה מסוג רשימה מקושרת, list. דרך משתנה זה יבוצעו כל הפעולות על המחסנית.

בקובץ הממשק של המחסנית, list_l.h (הסיומת l_ מציינת מימוש ע"י רשימה), יוכרז מבנה המחסנית כך:

```
typedef struct
{
	List list;
} Stack;
```

שים/י לב: ממשק הפונקציות של המחסנית נשאר כפי שהיה - התכנית המשתמשת במחסנית אינה משתנה כתוצאה משינוי המימוש!

נראה כעת מספר פונקציות בקובץ המימוש, stack_l.c :

– פונקצית הדחיפה למחסנית:

```
void stack_push(Stack *pstack, DataPtr pdata)
{
	if(stack_is_full(pstack))
		return;
	list_push_front(&pstack->list, pdata);
}
```

– פונקצית ההוצאה:

```
DataPtr  stack_pop(Stack *pstack)
{
	if(stack_is_empty(pstack))
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

```
        return NULL;
    else
        return list_pop_front(&pstack->list);
}
```

– הפונקציה top :

```
DataPtr  stack_top(Stack *pstack)
{
    if(stack_is_empty(pstack))
        return NULL;
    else
        return pstack->list.head->next->pdata;
}
```

– פונקציה לבדיקה אם המחסנית מלאה:

```
Boolean stack_is_full(Stack *pstack)
{
    return FALSE;
}
```

יש לשים לב שבמימוש ע"י רשימה מקושרת אין מגבלה על מספר האיברים שבמחסנית, לכן הפונקציה stack_is_full במימוש זה מחזירה תמיד ערך **שקר**.

## פונקצית הבדיקה

פונקצית הבדיקה היא כמו קודם מכיוון שתכנית הבדיקה לא השתנתה כתוצאה משינוי מימוש המחסנית!


מרכז ההדרכה 2000 © כל הזכויות שמורות

## סיבוכיות זמן פעולות על המחסנית

כפי שניתן לראות, המחסנית היא מבנה נתונים פשוט יחסית המיועד למצב בו מדיניות הכנסה והוצאת הנתונים היא LIFO.

לכן פעולות ההכנסה וההוצאה מהמחסנית הן בעלות סיבוכיות O(1). הטבלה הבאה מציגה את סיבוכיות הפעולות:

| פעולה | סיבוכיות |
|---|---|
| אתחול | O(1) |
| ניקוי | O(n) |
| הכנסה | O(1) |
| הוצאה | O(1) |
| איבר עליון | O(1) |
| בדיקה אם המחסנית ריקה | O(1) |
| בדיקה אם המחסנית מלאה | O(1) |
| הדפסת המחסנית | O(n) |

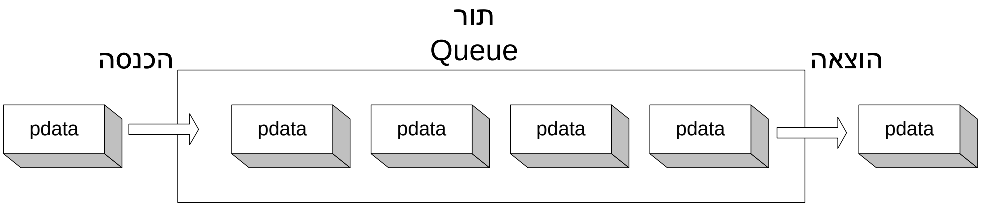
<u>תרגיל</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-3 שבעמ' 375.***

# תור

**תור** (**Queue**) הוא מבנה נתונים בעל מדיניות "נכנס-ראשון-יוצא-ראשון" (First In First Out), או בקיצור **FIFO**.

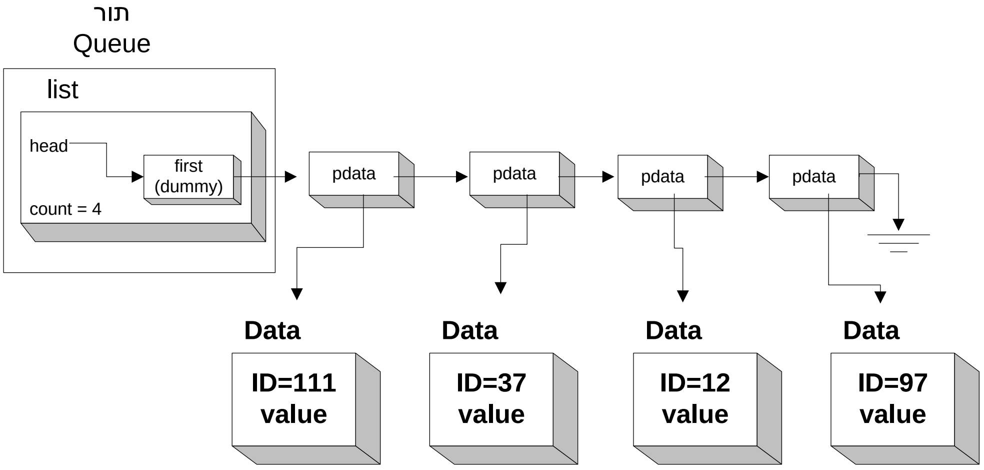
התור, כשמו, שימושי למתן שרות עפ"י סדר הגעה:




בדומה למחסנית, ניתן לממש תור הן כמערך סטטי והן ע"י רשימה מקושרת. אנו נייצג את התור ע"י רשימה מקושרת בלבד:

```
typedef struct
{
        List        list;
} Queue;
```

תרשים התור ממומש ע"י רשימה:





מרכז ההדרכה 2000 © כל הזכויות שמורות

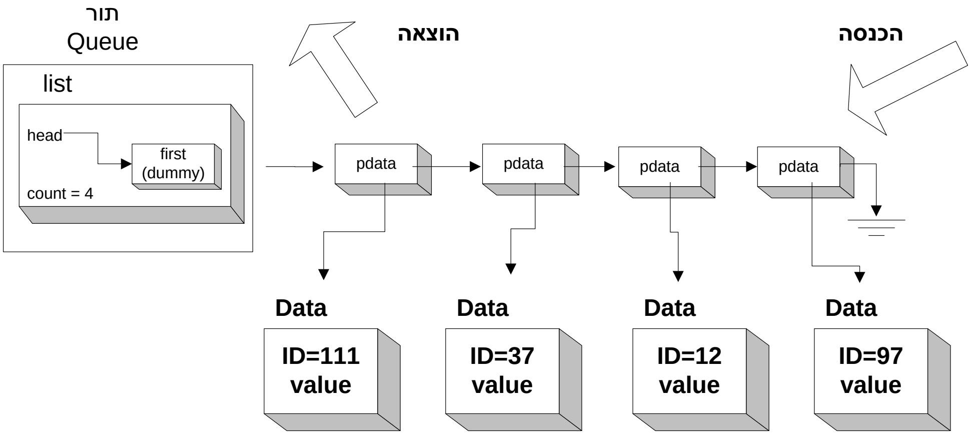
## פעולות על התור

נגדיר את הפעולות הבאות על התור:

- אתחל תור (queue_init)
- שחרר/נקה את התור(queue_clear)
- הכנס לתור (queue_push)
- הוצא מהתור (queue_pop)
- החזר את האיבר שבראש התור(queue_top)
- האם התור ריק? (queue_is_empty)
- הדפס את התור (queue_print)

הכנסת איבר חדש מבוצעת בסוף הרשימה, והוצאת האיבר הראשון בתור מבוצעת ע"י הוצאת האיבר מתחילתה.

התרשים הבא מציג את כוון ההכנסה וההוצאה מהתור הממומש ע"י רשימה:




הפונקציה להחזרת האיבר העליון (queue_top) מחזירה את האיבר שבראש התור **מבלי להוציאו**. בתרשים הנ"ל, הפונקציה תחזיר את האיבר 111.

ממשק הפונקציות:

```
void      queue_init(Queue *pqueue);
void      queue_clear(Queue *pqueue);
void      queue_push(Queue *pqueue, DataPtr pdata);
DataPtr   queue_pop(Queue *pqueue);
DataPtr   queue_top(Queue *pqueue);
Boolean   queue_is_empty(Queue *pqueue);
void      queue_print(Queue *pqueue);
```

מימוש הפונקציות:


מרכז ההדרכה 2000 © כל הזכויות שמורות

– אתחול התור:

```
void queue_init(Queue *pqueue)
{
	list_init(&pqueue->list);
}
```

– שחרור הזיכרון:

```
void queue_clear(Queue *pqueue)
{
	list_clear(&pqueue->list);
}
```

– דחיפת איבר לסוף התור:

```
void queue_push(Queue *pqueue, DataPtr pdata)
{
	list_push_back(&pqueue->list, pdata);
}
```

– הוצאת איבר מראש התור:

```
DataPtr  queue_pop(Queue *pqueue)
{
	if(queue_is_empty(pqueue))
		return NULL;
	else
		return list_pop_front(&pqueue->list);
}
```

***עיין/י בקוד שאר הפונקציות בהמשך.***

## פונקצית הבדיקה

פונקצית הבדיקה של התור מובאת בעמ' 379-378.

 מרכז ההדרכה 2000 © כל הזכויות שמורות

## סיבוכיות זמן פעולות על התור

סיבוכיות ההכנסה בסוף התור, הממומש ע"י רשימה מקושרת, היא O(n). לעומת זאת ההוצאה מראש התור היא מיידית ומסיבוכיות O(1).

פעולות הניקוי וההדפסה הן מסדר גודל של מספר האיברים בתור (O(n)).

הטבלה הבאה מציגה את סיבוכיות הפעולות על התור:

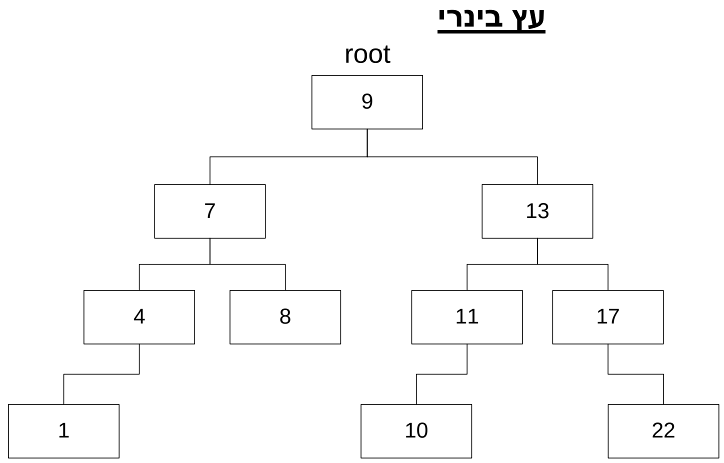
| **פעולה** | **סיבוכיות** |
|---|---|
| אתחול | O(1) |
| ניקוי | O(n) |
| הכנסה בסוף | O(n) |
| הוצאה מההתחלה | O(1) |
| בדיקה אם התור ריק | O(1) |
| הדפסה | O(n) |

<u>תרגול</u>

***קרא/י סעיף זה בספר ובצע/י את תר' 1-3 שבעמ' 380.***

# עץ בינרי

**עץ בינרי** (**Binary Tree**) הוא מבנה נתונים הבנוי בצורת עץ הפוך: שורש העץ מצוייר למעלה וה"ענפים" למטה:




כל צומת בעץ מכיל מצביע לנתונים, pdata, ושני מצביעים לשני צמתים בנים: בן שמאלי (left son) ובן ימני (right son). הצומת שבראש העץ הוא צומת השורש (root).

ערכי הנתונים מסודרים כך שהערך בכל צומת הוא מקסימלי ביחס לכל הצמתים שבתת-העץ השמאלי שלו, ומינימלי ביחס לאלו שבתת-העץ הימני שלו.

מבנה הצומת מוגדר כך:

```
typedef struct BTreeNode_tag
{
	DataPtr		pdata;
	struct		BTreeNode_tag  *left;
	struct		BTreeNode_tag  *right;
} BTreeNode;
```

אם לצומת אין בן מסוים, ערך המצביע לבן יהיה NULL. מבנה העץ בכללותו מוגדר כך:

```
typedef struct
{
	BTreeNode *root;	/* root of the binary tree */
	int		count;	/* number of elements in the tree */
} BTree;
```

מבנה העץ מכיל מצביע לצומת שבשורש העץ ומונה מספר האיברים.


מרכז ההדרכה 2000 © כל הזכויות שמורות

## פעולות על העץ

נגדיר את הפעולות הבאות על העץ:

- אתחל עץ(btree_init)
- הוספת איבר לעץ (btree_add)
- האם איבר מסוים קיים בעץ? (btree_exist)
- האם העץ ריק? (btree_is_empty)
- הדפס את העץ (btree_print)

## אתחול העץ

הפונקציה הבאה מאתחלת מבנה עץ המועבר לה כפרמטר:

```
void btree_init(BTree *pbtree)
{
	pbtree->root = NULL;
	pbtree->count = 0;
}
```

שורש העץ מצביע ל- NULL ומספר האיברים מאופס.

## הוספת איבר לעץ

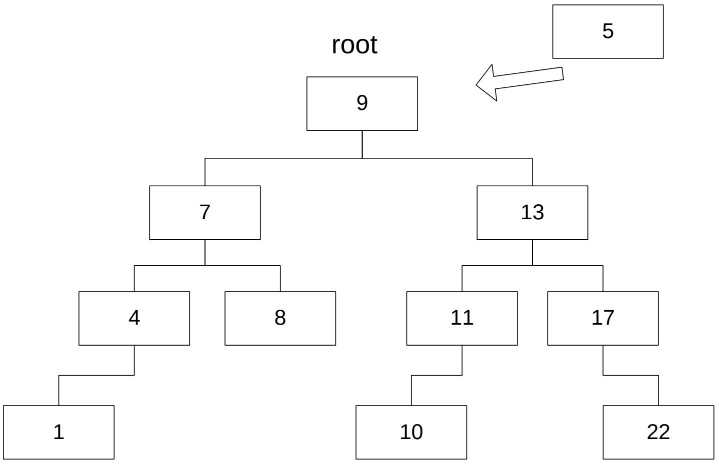
בכדי להוסיף איבר למקום המתאים בעץ, עלינו להשוות את ערכו עם זה של כל צומת, החל בשורש.

- אם ערך האיבר החדש גדול מזה של הצומת, עוברים לבן הימני ומבצעים את ההשוואה איתו.
- אחרת, עוברים לבן השמאלי וההשוואה מבוצעת איתו.

פעולה זו מתרחשת ב**רקורסיה**: במעבר לבן השמאלי או הימני מבוצעת ההשוואה מחדש, כמו בצומת האב. ושוב, אם ערך האיבר החדש גדול מזה של הצומת עוברים לבן הימני, אחרת לבן השמאלי.

פעולת הרקורסיה נעצרת כאשר אנו מגיעים לנתיב המוביל ל- NULL, כלומר לקצה העץ, ואז פשוט מוסיפים את האיבר החדש ע"י הוספת צומת חדש לעץ וקישורו במקום המתאים.

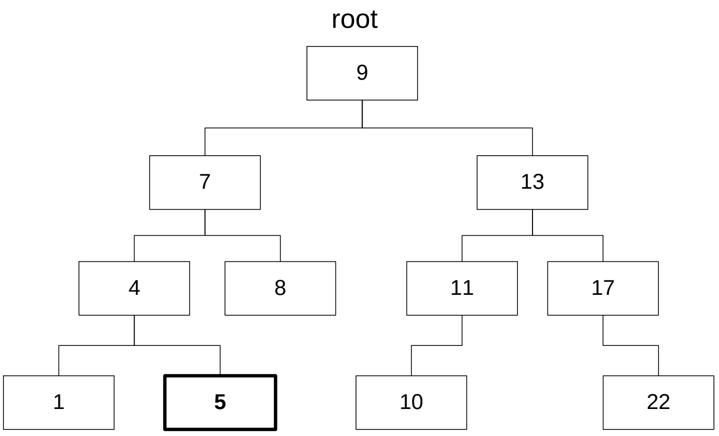
לדוגמא, בהינתן העץ הקודם, נניח שמוכנס איבר חדש עם ערך 5:




- משווים את האיבר החדש, 5, עם צומת השורש ומכיוון שהוא קטן ממנו עוברים לבן השמאלי, 7.
- בהשוואה עם צומת 7 , 5 קטן יותר ואנו עוברים שוב לבן השמאלי, 4.
- מכיוון ש- 5 גדול מ- 4 עוברים לבן הימני, שהוא NULL.
- מוסיפים את הצומת החדש לעץ כבן ימני של צומת 4

 מרכז ההדרכה 2000 © כל הזכויות שמורות

תרשים העץ לאחר ההוספה:




עקב האופי הרקורסיבי של הטיפול באיברים בהכנסה ובהוצאת איברים מהעץ, הגדרת פונקציות רקורסיביות היא הדרך הפשוטה והטבעית לכך.

לתזכורת לגבי פונקציות רקורסיביות, ראה/י פרק 6, "**פונקציות**", סעיף "רקורסיה".

<u>הקצאת צומת ואתחולו</u>

ראשית, נגדיר פונקציה המקצה ומאתחלת צומת בודד:

```
static BTreeNode *btree_new_node(DataPtr pdata)
{
	BTreeNode *pnode = malloc(sizeof(BTreeNode));
	pnode->pdata = pdata;
	pnode->left = NULL;
	pnode->right = NULL;
	return pnode;
}
```

<u>הסבר</u>: הפונקציה מקצה זיכרון עבור צומת חדש ומציבה לו את הפרמטר, מצביע ל- Data, כערך הצומת. לאחר מכן מאופסים המצביעים לבנים.

הערך המוחזר של הפונקציה הוא המצביע לצומת החדש. הפונקציה מוגדרת כ**סטטית** (**static**) מכיוון שהיא לצורך פנימי של מודול העץ הבינרי, כלומר **פונקציה פרטית**.

<u>הוספה רקורסיבית לעץ</u>

לצורך הוספת איבר לעץ, נגדיר פונקציה פרטית המקבלת כפרמטרים מצביע לתת-עץ (כלומר, צומת כלשהו בעץ) וצומת חדש להוספה.

הפונקציה בודקת את ערך הצומת החדש ביחס לערך שבצומת, ובהתאם מוסיפה אותו בצידו השמאלי או הימני:

```
static void btree_add_to_subtree(BTreeNode *root, BTreeNode *pnode)
{
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

```
    if(data_compare(pnode->pdata, root->pdata) < 0) /* add as left son */
    {
        if(root->left == NULL)
            root->left = pnode;
        else
            btree_add_to_subtree(root->left, pnode);
    }
    else  /* add as right son */
    {
        if(root->right == NULL)
            root->right = pnode;
        else
            btree_add_to_subtree(root->right, pnode);
    }
}
```

הסבר: לאחר השוואת ערך הצומת החדש עם זה של שורש תת-העץ, נבחר הצד המתאים בעץ - שמאל או ימין. אם נבחר הצד השמאלי, בודקים אם קיים בן:

אם לא (NULL) - מוסיפים את הצומת החדש כבן שמאלי

אם כן - קוראים באופן רקורסיבי לפונקציה **btree_add_to_subtree**.

בדיקה מקבילה מבוצעת בצד הימני של העץ.

הפונקציה **btree_add_to_subtree** קוראת לעצמה באופן **רקורסיבי** עד להגעה ל"עלה" בעץ - כלומר לצומת שאין לו בן ימני/שמאלי - ומוסיפה את הצומת החדש במקום המתאים.

מודול העץ הבינרי מספק פונקציה ציבורית להוספת נתון לעץ. פונקציה זו קוראת ל-**btree_add_to_subtree** להוספה רקורסיבית לעץ:

```
void btree_add(BTree *pbtree, DataPtr pdata)
{
    if(btree_is_empty(pbtree))
        pbtree->root = btree_new_node(pdata);
    else
        btree_add_to_subtree(pbtree->root, pdata);
    pbtree->count++;
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## האם איבר מסוים קיים בעץ

כדי לבדוק אם איבר מסוים קיים בעץ יש צורך לסרוק את העץ החל מהשורש כלפי מטה, ובכל צומת להשוות את ערך האיבר עם זה של הצומת:

- אם ערך האיבר שווה לזה שבצומת, התשובה חיובית וסיימנו.
- אחרת, אם ערך האיבר קטן מזה שבצומת, עוברים לחיפוש בתת-העץ השמאלי.
- אחרת (ערך האיבר גדול מזה שבצומת), עוברים לחיפוש בתת-העץ הימני.

הפונקציה הפרטית **btree_exist_in_subtree** מבצעת בדיוק את האלגוריתם הנ"ל באופן רקורסיבי. הפונקציה מקבלת כפרמטר מצביע לתת-עץ ומצביע לנתון.

***static Boolean btree_exist_subtree(BTreeNode *root, DataPtr pdata);***

***קוד הפונקציה מובא בעמ' 385.***

פונקציה זו נקראת ע"י הפונקציה הציבורית, **btree_exist**, הבודקת קיום של איבר בעץ ע"י קריאה ל- **btree_exist_subtree** עם מצביע לשורש:

```
Boolean btree_exist(BTree *pbtree, DataPtr pdata)
{
	return btree_exist_subtree(pbtree->root, pdata);
}
```

*© כל הזכויות שמורות*

## שחרור זיכרון העץ

גם מחיקת איברי העץ מבוצעת ע"י פונקציה רקורסיבית פרטית:

```
static void btree_del_subtree(BTreeNode *root)
{
	if(root==NULL)
		return;
	btree_del_subtree(root->left);
	btree_del_subtree(root->right);
	data_clear(root->pdata); /* free the data */
	free(root); /* free the node */
}
```

הסבר: הפונקציה **btree_del_subtree** היא פונקציה **פרטית** של המודול, הנקראת ע"י הפונקציה הציבורית **btree_clear** (להלן). הפונקציה בודקת את תנאי העצירה של הרקורסיה - הגעה לצומת NULL - וחוזרת אם התשובה חיובית:

```
	if(root==NULL)
		return;
```

בשלב הבא היא מבצעת קריאה רקורסיבית לעצמה למחיקת תת-העץ השמאלי ואח"כ הימני:

```
	btree_del_subtree(root->left);
	btree_del_subtree(root->right);
```

לבסוף משוחררים נתוני הצומת הנוכחי והצומת עצמו:

```
	data_clear(root->pdata); /* free the data */
	free(root); /* free the node */
```

הפונקציה הרקורסיבית **brtee_del_subtree** נקראת ע"י הפונקציה **btree_clear** המוחקת את כלל העץ ומאפסת את המבנה ע"י קריאה ל- **btree_init**:

```
void btree_clear(BTree *pbtree)
{
	btree_del_subtree(pbtree->root);
	btree_init(pbtree);
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

## סיורים רקורסיביים בעץ

בהתאם למבנה של עץ בינרי קיימות שלוש דרכים לסריקה רקורסיבית של האיברים שבו:

- סרוק את השורש, את הבן שמאלי ולבסוף את הבן ימני (**Pre-Order**)
- סרוק את הבן שמאלי, את השורש ולבסוף את הבן ימני (**In-Order**)
- סרוק את הבן שמאלי, את הבן ימני ולבסוף את השורש (**Post-Order**)

<u>סיור Pre-Order</u>

לדוגמא, הפונקציה הרקורסיבית (הפרטית) הבאה מדפיסה תת-עץ נתון ב- pre-order:

```
static void btree_preorder(BTreeNode *root)
{
	if(root)
	{
		data_output(root->pdata);
		btree_preorder(root->left);
		btree_preorder(root->right);
	}
}
```

<u>הסבר</u>: הפונקציה מקבלת כפרמטר מצביע לצומת שורש, root. אם המצביע אינו NULL מבצעים בסדר הבא:

– נתוני הצומת מודפסים

*data_output(root->pdata);*

– מודפס תת-העץ השמאלי של root

*btree_preorder(root->left);*

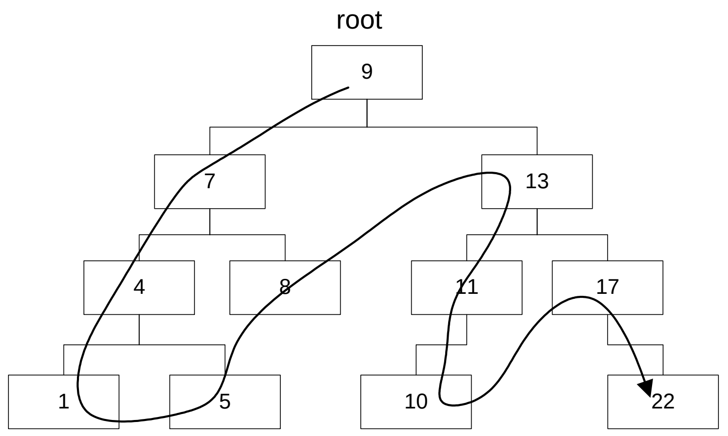
– מודפס תת-העץ הימני של root

*btree_preorder(root->right);*

תנאי העצירה של הרקורסיה הוא הגעה ל- NULL.

*מרכז ההדרכה 2000 © כל הזכויות שמורות* www.mh2000.co.il

תרשים הסיור:




פלט הפונקציה עבור העץ שבתרשים:

```
ID=9, value=99.000000
ID=7, value=77.000000
ID=4, value=44.000000
ID=1, value=11.000000
ID=5, value=55.000000
ID=8, value=88.000000
ID=13, value=133.000000
ID=11, value=111.000000
ID=10, value=100.000000
ID=17, value=177.000000
ID=22, value=222.000000
```

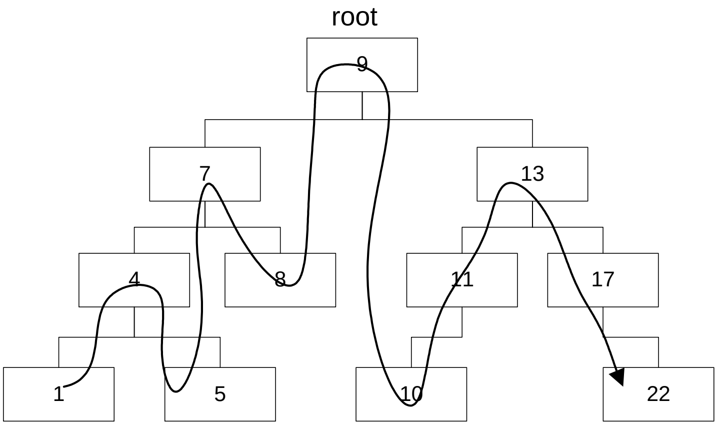
<u>סיור in-order</u>

הפונקציה ()btree_inorder סורקת את צמתי העץ ומדפיסה את ערכיהם ב- In-Order:

```
static void btree_inorder(BTreeNode *root)
{
	if(root)
	{
		btree_inorder(root->left);
		data_output(root->pdata);
		btree_inorder(root->right);
	}
}
```


מרכז ההדרכה 2000 © כל הזכויות שמורות

תרשים הסיור :




פלט הפונקציה :

```
ID=1, value=11.000000
ID=4, value=44.000000
ID=5, value=55.000000
ID=7, value=77.000000
ID=8, value=88.000000
ID=9, value=99.000000
ID=10, value=100.000000
ID=11, value=111.000000
ID=13, value=133.000000
ID=17, value=177.000000
ID=22, value=222.000000
```

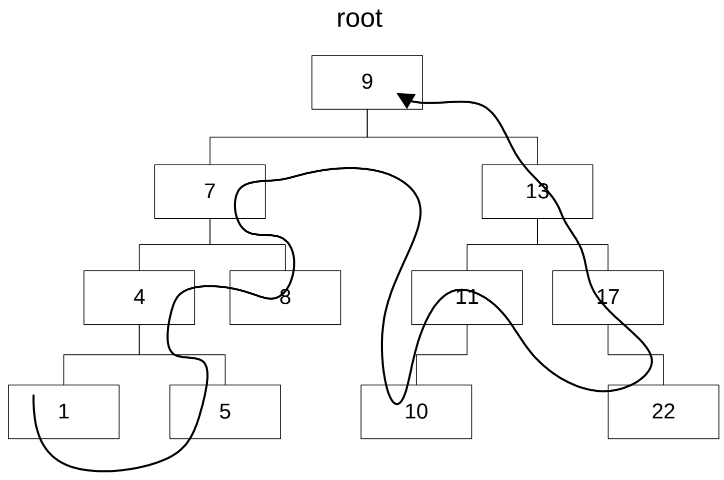
## סיור Post-Order

הפונקציה ()btree_postorder מבצעת סיור Post-Order :

```
static void btree_postorder(BTreeNode *root)
{
	if(root)
	{
		btree_postorder(root->left);
		btree_postorder(root->right);
		data_output(root->pdata);
	}
}
```

 מרכז ההדרכה 2000 © כל הזכויות שמורות

תרשים הסיור:




פלט הפונקציה:

```
ID=1, value=11.000000
ID=5, value=55.000000
ID=4, value=44.000000
ID=8, value=88.000000
ID=7, value=77.000000
ID=10, value=100.000000
ID=11, value=111.000000
ID=22, value=222.000000
ID=17, value=177.000000
ID=13, value=133.000000
ID=9, value=99.000000
```

<u>פונקציה להדפסה כללית של העץ</u>

הפונקציה הציבורית btree_print() מדפיסה את איברי העץ בכל שלוש הצורות, ע"י קריאה מתאימה לכל אחת מהפונקציות הפרטיות.

***קוד הפונקציה מובא בעמ' 390.***

## פונקצית הבדיקה

פונקצית הבדיקה של העץ הבינרי מובאת בעמ' 391.


*מרכז ההדרכה 2000 © כל הזכויות שמורות*

## סיבוכיות זמן פעולות על העץ הבינרי

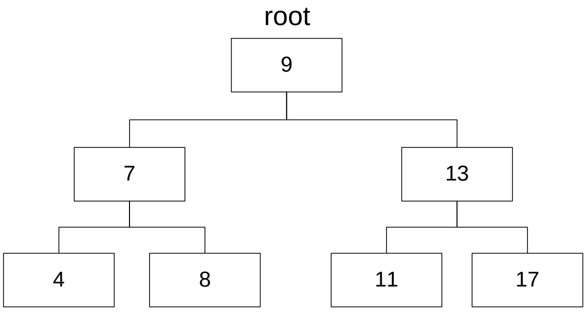
סיבוכיות פעולות ההכנסה והחיפוש של איבר בעץ תלויה בגובה העץ, h: בממוצע, בכדי להכניס איבר לעץ או בחיפושו נידרש לעבור על h/2 איברים, ולכן הסיבוכיות היא O(h).

**שאלה**: מהו גבהו של העץ, h, ביחס למספר האיברים, n?

**תשובה**: אם נניח שלכל צומת שני בנים והעץ מאוזן, אזי נתונה הנוסחה:

$n = 2^h - 1$

כלומר, בכל רמה בעץ גדל מספר האיברים פי 2. לדוגמא, עבור עץ בגובה 3:




מספר האיברים הוא:

$n = 2^3 - 1 = 7$

מכיוון שאנו עוסקים בסיבוכיות, ניתן להשמיט את המספר 1 מהמשוואה

$n = 2^h$

וכעת ניתן לבטא את h ע"י לוגריתם:

$h = \log n$

בסיס הלוגריתם הוא 2, אך נהוג להשמיטו בחישובי סיבוכיות. לכן סיבוכיות הכנסה וחיפוש של איבר בעץ הבינרי היא O(log n).

פעולות הניקוי וההדפסה הן מסיבוכיות O(n) מכיוון שהן מחייבות מעבר על כלל האיברים.

הטבלה הבאה מציגה את סיבוכיות הפעולות על העץ:

| **פעולה** | **סיבוכיות** |
|---|---|
| אתחול | O(1) |
| ניקוי | O(n) |
| הכנסה | O(log n) |
| בדיקה אם איבר נתון קיים בעץ | O(log n) |
| בדיקה אם העץ ריק | O(1) |
| הדפסה | O(n) |

## סוגי עצים נוספים

קיימים עצים מסוגים שונים ובוריאציות שונות, בין הנפוצים:

- **עץ מאוזן** - עץ ששני צידיו מאוזנים. בעץ זה מובטח שגובה העץ יהיה תמיד מסדר גודל log n. האיזון ממומש ע"י הכנסה באופן מאוזן לעץ, תוך ביצוע "גלגולים" במידת הצורך.
- **עץ 2-3** - עץ שכל צומת בו יכול להכיל 2 או שלושה צמתים בנים.
- **ערמה** - עץ שבו האיבר המקסימלי (או מינימלי) נמצא בשורש, ונגיש בסיבוכיות O(1).

תרגיל

***קרא/י סעיף זה בספר ובצע/י את התרגיל שבעמ' 393.***

# סיכום

- **טיפוסי נתונים מופשטים** (**ADT**, Abstract Data Types) הם טיפוסים המורכבים מטיפוסים בסיסיים יותר וכוללים גם פונקציונליות לטיפול בנתונים.

  ניתן לממש טיפוס מופשט במספר צורות. בהפשטה אנחנו מתרכזים במאפיינים הרלוונטיים של הטיפוס ולא במימוש שלו.

- **סיבוכיות** היא מדד ביצוע של מבני נתונים ואלגוריתמים. קיימים שני סוגי סיבוכיות: סיבוכיות **זמן** וסיבוכיות **מקום**, המתייחסות למשך הזמן ולכמות הזיכרון הנדרשים באלגוריתם.

  רמת הסיבוכיות מצוינת ע"י האות האנגלית O בצירוף סדר גודל הסיבוכיות בסוגריים.

- **רשימה מקושרת** (Linked List) היא מבנה נתונים דינמי המורכב כשרשרת של צמתים. כל צומת מכיל מצביע לנתונים ומצביע לצומת הבא ברשימה.

- **מחסנית** (Stack) היא מבנה נתונים שאיבריו נשלפים בסדר הפוך לזה שבו הוכנסו, כלומר "נכנס אחרון יוצא ראשון" (**FIFO**, **L**ast **I**n **F**irst **O**ut).

  ראינו שניתן לממש את המחסנית הן ע"י מערך והן ע"י רשימה מקושרת, מבלי לשנות את הממשק שלה.

- **תור** (**Queue**) הוא מבנה נתונים מופשט המספק פונקציות להכנסת איברים ולהוצאה לפי הכלל "נכנס ראשון יוצא ראשון" (**FIFO**, **F**irst **I**n **F**irst **O**ut).

- **עץ בינרי** (**Binary Tree**) הוא מבנה נתונים הבנוי בצורת עץ הפוך: שורש העץ מצוייר למעלה וה"ענפים" למטה.

  - כל צומת בעץ מכיל מצביע לנתונים, pdata, ושני מצביעים לשני צמתים בנים: בן שמאלי (left son) ובן ימני (right son).
  - הצומת שבראש העץ הוא צומת השורש (root).
  - פעולות על העץ מבוצעות בד"כ באופן רקורסיבי.

# תרגילי סיכום

***בצע/י את תרגילי הסיכום שבסוף פרק זה.***

 *מרכז ההדרכה 2000 © כל הזכויות שמורות*