



מגמת הנדסת אלקטרוניקה ומחשבים

שפת תכנות C#

תוכן עניינים

2	דבר המפמ"ר.....
3	פתח דבר.....
4	פרק 1 – מבוא.....
20	פרק 2 – מושגי יסוד בתכנות.....
49	פרק 3 – ביצוע מותנה.....
82	פרק 4 – ביצוע חוזר.....
112	פרק 5 – ייצוג טיפוס נתונים חדש על ידי מחלקה.....
130	פרק 6 – פעולות סטטיות.....
154	פרק 7 - מחרוזות.....
167	פרק 8 - מבנים סדרתיים - מערך חד ממדי.....
181	פרק 9 - טיפוסים מורכבים.....
201	פרק 10 - רשימה.....
227	פרק 11 - מטריצות, חיפושים ומיונים.....

דבר המפמ"ר

אנו עומדים בפתחה של תקופה של חידושים טכנולוגיים, אתגרים חשיבתיים והתנסויות מרתקות. במגמת הנדסת אלקטרוניקה ומחשבים אתם תחשפו לסביבת למידה מתקדמת ומשמעותית. זכרו, כי אתם חלק חשוב מהמחר ובידכם הכוח והידע לקחת חלק בפיתוח דור העתיד.

הפקדתי בידי צוות מורי המגמה, אחריות רבה, לחשוף אתכם לידע חדשני, להנחות אתכם בפרויקטים מאתגרים ולהוביל אתכם להצלחה.

שלומי אדמונד אחנין

מפמ"ר מגמות הנדסת אלקטרוניקה ומחשבים ומערכות בקרה ואנרגיה

פתח דבר

ברוכים הבאים לעולם התכנות באמצעות שפת התכנות #C.

ספר זה מיועד לתלמידים ולכל מי שחפץ ללמוד לתכנת בשפת תכנות זו. אנו מניחים כי לקורא המיועד אין רקע תכנותי בשפה כלשהי ואף לא ידע בסיסי, ולכן הספר בנוי כך שההתקדמות היא איטית, הדרגתית ומלווה בדוגמאות רבות.

הספר כולל הסברים תיאורטיים, שאלות חשיבה, איורים, ולפעמים אף שאלות אתגר. אנו ממליצים בכל לב לעבור מלימוד תיאורטי של הנושאים אל שלב כתיבת הקוד ולכתוב פרויקטים שונים בעולם התכנות.

שפת התכנות #C היא שפה מדהימה, עשירה מאוד ביכולות שלה. היא נגזרת מהשפה C ומהשפה C++, ולכן, מי שעבד עם השפות הללו ימצא כי המעבר ל-#C מהיר בדרך כלל.

#C תומכת בטיפוסי נתונים המובנים בשפה, ואף מאפשרת עבודה עם טיפוסי נתונים מופשטים (Abstract Data Type) שאותם יכול להגדיר המפתח.

כמו כן, #C תומכת בראיית עולם של תכנות מונחה עצמים (Object-Oriented Programming), עבודה עם כימוס (Encapsulation), עבודה עם תורשה (Inheritance), ותכנות רב-צורתי (פולימורפיזם).

שפת #C מתממשת בקלות יחסית עם רכיבי חומרה והתקנים שונים, וכן עם עולמות תוכן שונים ומגוונים. לדוגמה, #C עובדת בצורה הרמונית עם סביבת Arduino.

סביבת העבודה

מחד, נעבוד עם Visual Studio 2017, ולכן יהיו משויכים אליה צילומי מסך, דוגמאות קוד, פלטים, הודעות שגיאה ועוד.

מאידך, מהדרים (Compilers) אחרים או גרסאות ישנות יותר של Visual Studio מתפקדים למעשה בצורה דומה, לכן קל מאוד להקיש בין מהדרים שונים.

אנחנו מאחלים לך עבודה פורייה והצלחה רבה, ומקווים מאוד כי הספר יהיה מועיל ומעשיר, ויפתח עבורך עולמות תוכן חדשים.

צוות הפיתוח, ירושלים, ישראל, 2018

פרק 1 - מבוא

1.1 הדגמת תפקידו וחשיבותו של מקצוע הנדסת המחשבים

באמצעות דיון באתגרים חישוביים מתחומי ידע שונים

מחשבים הם מכונות מבוססות פעולות חישוב שונות, המסוגלות לבצע מטלות.

מחשבים מסוגים שונים (נייחים, סמארטפונים, טאבלטים ועוד) מסוגלים לבצע אך ורק פקודות שאנו פוקדים עליהם, והם מתנהגים בצורה צפויה הניתנת לחיזוי.

דוגמאות למשימות שונות המתבצעות על ידי מחשבים:

1. הפקת תלושי משכורת:

- קלט - המידע שהמחשב מקבל - שם העובד, מספר זהות, ותק, תפקיד, שכר לשעה ועוד.
- פלט - המוצר שהמחשב מפיק/מדפיס - תלוש שכר.

2. מנוע חיפוש:

- קלט - מילות חיפוש שהמשתמש הקליד.
 - פלט - דפי אינטרנט שבהם מופיעה המילה שהמשתמש הקליד, או המילה שהמשתמש הקליד בצירוף תכנים נוספים.
- לדוגמה, חיפשו את המילה "בית" וקיבלנו גם את המונח "בית ספר".

3. מחשב של מטוס נוסעים:

- קלט - מיקום נוכחי, כיוון הרוח ועוצמתה, כמות הדלק במכלים, המרחק מהיעד ועוד.
- פלט - המלצה למסלול טיסה או לשינוי כיוון.

אתגרים חישוביים:

נניח כי יש לנו מאגר מידע הכולל כמה מיליוני אנשים. מחשב מסוגל לבצע בשנייה אחת מיליוני פעולות לפחות. לדוגמה, אם מבקשים לאתר אדם עוין בכניסה לשדה תעופה – המחשב יהיה מסוגל בדרך כלל לאתר אותו תוך מספר שניות/דקות קטן יחסית.

לעומת המחשב, לאדם זוהי משימה בלתי אפשרית כמעט. נסו לדמיין אדם המדפדף בקלסר משרדי ובוחן תמונות של אנשים עוינים. סביר להניח שעד שאותו אדם ימצא את התמונה המתאימה, הגורם העוין כבר יגיע ליעדו או ימות מזקנה.

באופן דומה מאוד, באתרי קניות המוניים ברשת האינטרנט, כגון eBay או AliExpress, קיימים מאות אלפי מוצרים לפחות. כאשר אדם מזמין מוצר מסוים, חשוב להימנע ממצב שבו הוא מזמין את הפריט, משלם עבורו, ורק אז החברה מגלה שהפריט חסר במלאי.

בכל רגע נתון מתבצעות ברחבי הגלובוס מיליוני הזמנות של אנשים שונים. חייבת להיות מכונת חישוב שבודקת **בכל רגע נתון** את מצב המלאי ואת תקינותן של ההזמנות. את כל אלו מבצעים מחשבים לפי פקודות שניתנו להם.

1.2 הכרת משימות חישוביות פשוטות: ניתח המשימה, ניסוח

אלגוריתמי של פתרון אפשרי

בהמשך לדוגמה האחרונה, בואו ננתח מקרה כזה:

ניתוח משימה מספר 1:

אדם מבקש לבצע קנייה באינטרנט – לדוגמה, באתר ebay.

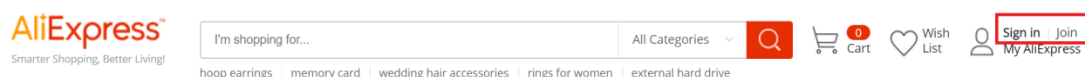
- א. בשלב הראשון עליו להקליד את שם המשתמש שלו ואת הסיסמה.
- ב. בשלב השני מעביר המחשב שלו את הנתונים לשרת המרכזי (Server) ובודק אם הפרטים של האדם נמצאים במאגר הנתונים.
- ג. בשלב השלישי סורק השרת (או המחשב המרכזי) את מאגר הנתונים של האנשים שיש להם חשבון פעיל ב-ebay כדי לאתר את שם המשתמש והסיסמה של אותו אדם.
- ד. אם הפרטים נמצאו, אפשר להתקדם. האתר מכיר את האדם, את כתובתו, את אופן התשלום שהוא מעדיף – ואפשר להתקדם עם הקנייה.

לדוגמה, אתר ebay מזהה את הלקוח ופונה אליו בשמו: "שלום, אודי!"



- ה. אם פרטי הלקוח לא נמצאו במאגר – אמורה להיות אפשרות **להיכנס שוב** או **להירשם**. הבעיה (במקרה זה, האתר AliExpress לא הצליח לזהות לקוח) והפתרון האלגוריתמי שלה (להירשם מחדש או להיכנס שוב) חייבים להיות ברורים וחד-משמעיים.

לדוגמה, ניסינו להיכנס לאתר AliExpress ולא הצלחנו. האתר מאפשר לנו להיכנס שוב (**Sign in**) או להירשם (**Join**):



ניתוח משימה מספר 2:

אדם מבקש להתחבר לאתר הבנק שלו על מנת לבצע פעולות מסוימות בחשבון.

א. בשלב הראשון מבקש אתר הבנק את הקלטים הבאים:

	שם משתמש: (קלט ראשון) סיסמה: (קלט שני)
---	--

ב. בשלב השני מעביר המחשב שלו את הנתונים לשרת המרכזי (Server) של הבנק ובודק האם הפרטים של האדם נמצאים במאגר.

ג. בשלב הבא סורק השרת של הבנק (או המחשב המרכזי שלו) את הפרטים של כל האנשים שיש להם חשבון פעיל בבנק כדי לאתר את שם המשתמש של אותו לקוח.

ד. יש שתי תוצאות אפשריות לבדיקה האם הלקוח נמצא במאגר.

- אפשרות ראשונה: הלקוח לא הצליח להיכנס לאתר, והוא עשוי לקבל הודעה כזו:

"לקוח/ה נכבד/ה,

פרטי ההזדהות שמסרת שגויים.

נא ודא/י תקינות נתוני ההזדהות."

2. אפשרות שנייה: הפעולה **הצליחה**, והלקוח יכול לבצע בחשבון מגוון פעולות. לדוגמה,

העברות ותשלומים	
העברה בין חשבונות	פעולה ראשונה (להעביר כסף לחשבון אחר)
העברה מחזורית (הוראת קבע)	פעולה שנייה (לשלם באופן קבוע תשלומים כגון חשבון חשמל)
העברות קודמות	פעולה שלישית (הצגת כל התשלומים הקודמים)
ריכוז הוראות קבע והעברות עתידיות	פעולה רביעית (הצגת תשלומים עתידיים)
תשלום חשבונות	פעולה חמישית (תשלום חשבון חד-פעמי)

ניתוח משימה מספר 3:

אדם מבקש להתחבר למשחק ב-XBOX שלו. נניח כי שם המשחק הוא FORTNINE

קלט: סיסמת הרשת האלחוטית הביתית, פרטי חשבון המשתמש ב-Microsoft וכדומה.

פלט: **הצלחה:** המשחק עובד (אפשר לעדכן מפה, למשל); או,

כישלון: לדוגמה, הסיסמה לכניסה לרשת אינה נכונה.

1.3 הכרת הושג "שפת תכנות"

שפת התכנות C# מורכבת מ-200 פקודות פחות או יותר.

בעולם התוכן הזה לא נדבר על מילים (כמו ב"שפה" רגילה) אלא על **פקודות** או **הוראות**.

כל הוראה גורמת למחשב לבצע פעולה חד-משמעית וסופית. כל פקודה מורכבת ממספר מסוים של תווים – כמו שכל מילה (בשפה העברית או בכל שפה אחרת) מורכבת ממספר מסוים של אותיות.

המהות של עולם התכנות היא לשלוט בעבודת המחשב בכל הרמות. נעשה זאת בעזרת פקודות שמורות ופקודות שהמתכנת יכתוב. הפקודות מכונות גם הוראות. רצפים אלה של הוראות נקראים תוכניות מחשב.

תכנות כרוך בתיאור הפעולה שהמשתמש מבקש מהמחשב לבצע, באמצעות רצף סופי של צעדים המכונה אלגוריתם. אלגוריתם הוא סדרה של הוראות/פקודות מדויקות וחד-משמעיות לביצוע משימה נתונה.

ההוראות הבונות את האלגוריתם חייבות להיות חד-משמעיות ומובנות למי שאמור לבצע אותן (אדם, מכונה או מחשב).

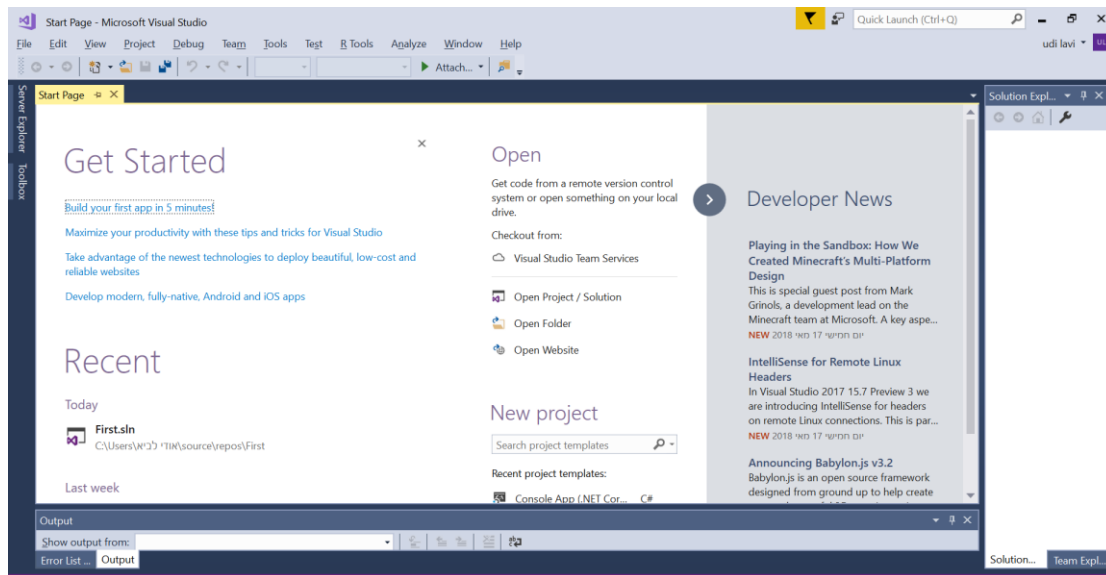
מתכנתים הם האנשים שיוצרים את ההוראות האלה, השולטות במחשבים. הוראות אלה נקראות תוכניות. קיימות תוכניות רבות, והן נוצרות תוך שימוש בשפות תכנות שונות.

1.4 הכרת מושג התוכנית: קריאה, כתיבה, הרצה, בדיקה

ותיקון תוכניות פשוטות

עבודה עם Visual Studio 2017

עם הפעלת Visual Studio 2017 יופיע המסך הבא:

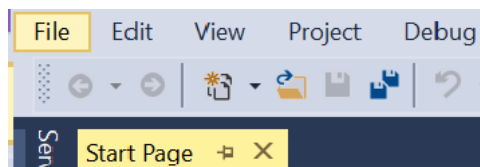


בעבודה עם Visual Studio, כל פרויקט בנוי ממחלקה אחת לפחות ומפונקציות (פונקציה ראשית לפחות), ומכיל תוכניות, פעולות ומחלקות. בכל פעם שנרצה לכתוב תוכנית, למעשה נפתח פרויקט חדש.

כאמור, כל פרויקט בנוי ממחלקה (Class) אחת לפחות. בהמשך נסביר מהי מחלקה.

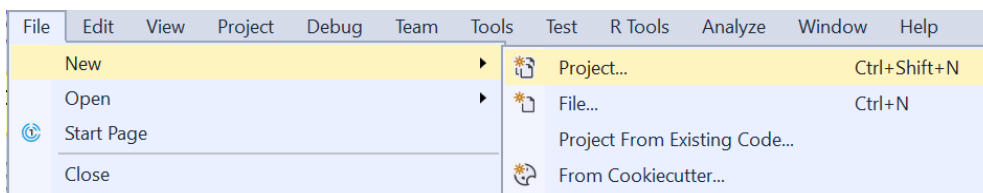
את פתיחת הפרויקט נבצע על ידי לחיצה על **New Project**.

אלה השלבים:



שלב 1:

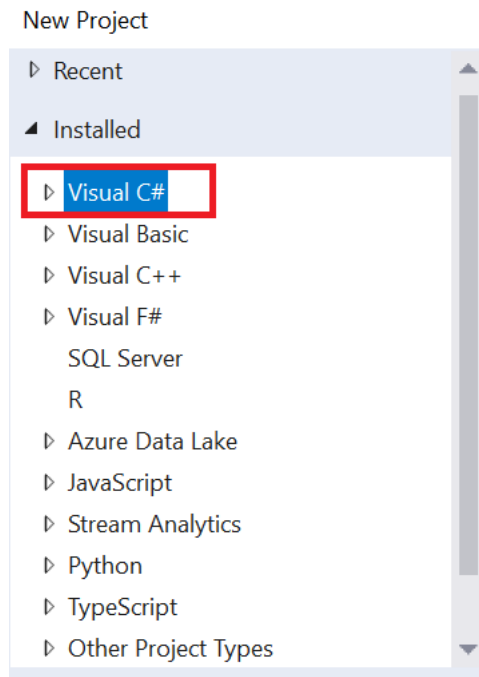
נלחץ על **File** כדי לפתוח את התפריט.



שלב 2:

בתפריט שנפתח נבחר **New** ובתת-תפריט שנפתח נבחר **Project**.

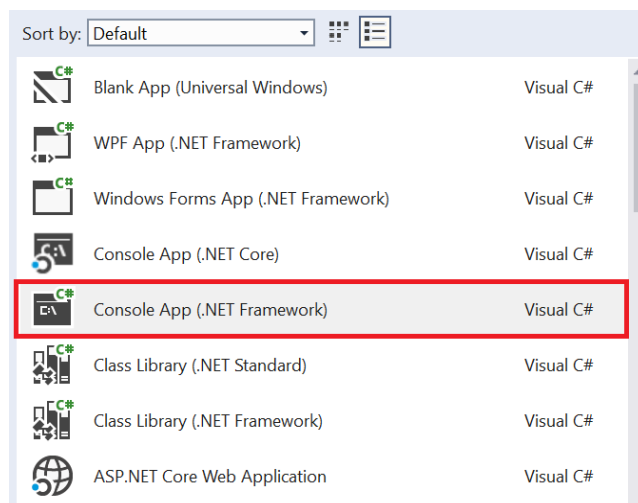
שלב 3:



מרשימת השפות (הרשימה נקבעה בהתקנה) המופיעה בצד שמאל, נבחר בשפה C# (המסומנת כאן באדום):

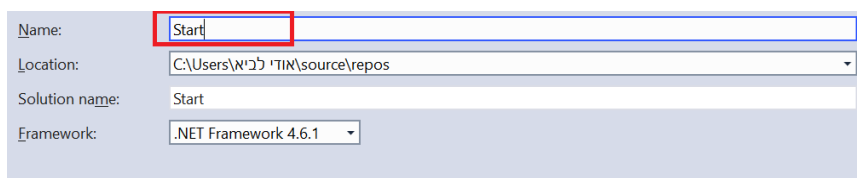
שלב 4:

מרשימת תת-האפשרויות שנפתחה מימין למסך הקודם נבחר באפשרות המסומנת באדום.

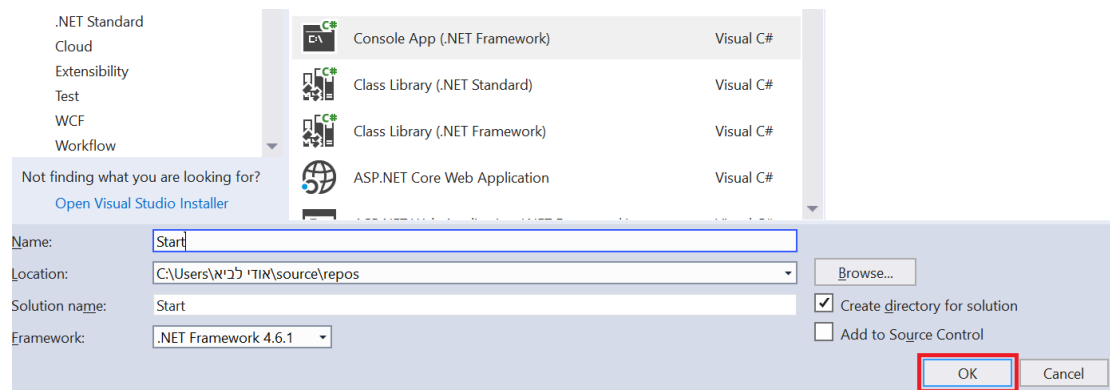


שלב 5:

בתחתית המסך יש לתת שם לפרויקט (רק באנגלית, כמובן):

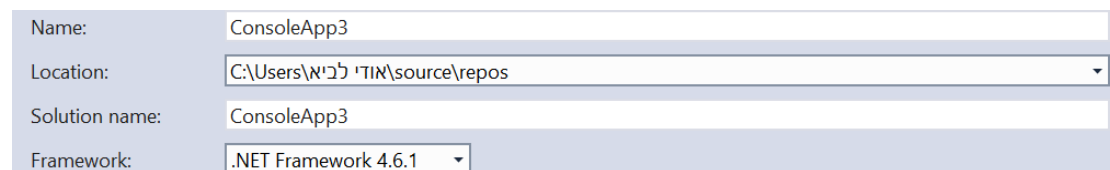


שלב 6: נאשר את הבחירה בלחיצה על כפתור OK:



הסבר:

אנחנו נעבוד תחת Console Application (.Net Framework) (האפשרות הראשונה, כאמור). יש לשים לב לכך ששינוי שם הפרויקט גורם לשינוי שם הפתרון (Solution). בהמשך נבחן את המשמעות של פתרון.



את השורות המסומנות (שורות 2-5) אפשר יהיה למחוק – הן מיותרות לנו:

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;

```

מסך העבודה שלנו נראה כך:

```

1  using System;
2
3  namespace Start
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              //
10             //
11             //
12         }
13     }
14 }

```

בשלב זה אפשר למחוק גם את "מרחב העבודה" (השורות המסומנות באדום). המסך שלנו הצטמצם שוב, וכעת הוא נראה כך:

```

1  using System;
2
3  class Program
4  {
5      static void Main(string[] args)
6      {
7          //
8          //
9          //
10     }
11 }

```

הסברים:

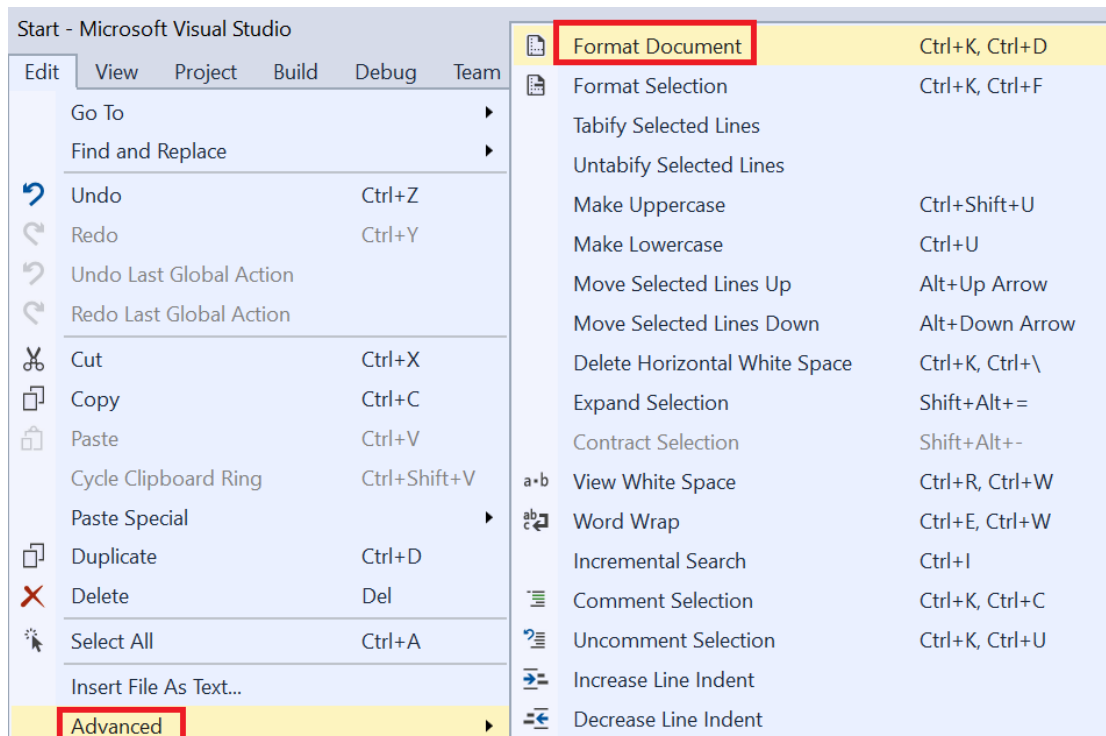
מספרי השורה/שורות	משמעות
1	עבודה עם משאבי המערכת – רכיב חובה בכל פרויקט.
3-9	המחלקה שלנו. בגוף המחלקה נצטרך לרשום פקודות.
5-8	הפונקציה הראשית בפרויקט – פונקציית ה-Main. מעתה נכתוב את הפקודות שלנו בתוך הפונקציה הנ"ל.

מוסכמה: i

נשמור על הזחה. הזחה היא ריווח בתחילת השורה (מצד שמאל).

כמו כן, חשוב לשים לב שכל סוגר-סוגר ('}') נמצא בדיוק מתחת לסוגר-פותח ('{'). חשוב מאוד להקפיד לשמור על המוסכמה הזו. היא תחסוך לנו זמן יקר של מציאת שגיאות.

לפעמים נוריד קבצים או תוכניות מרשת האינטרנט או ממקור אחר, וההזחה עלולה להיות לא-מושלמת. אפשר לקבוע הזחה אוטומטית באופן הבא: בשורת התפריטים נבחר Edit, ומשם נתקדם על פי הסימונים באדום:



1.5 מחלקה ופעולת Main כמסגרת בסיסית לכתיבת תוכנית

התוכנית העיקרית נמצאת, כאמור, בפונקציית Main:

נרשום שתי פקודות ראשונות:

```

3  class Program
4  {
5      static void Main(string[] args)
6      {
7          Console.WriteLine("My First C# Program !");
8          Console.ReadKey();
9      }
10 }
```

הסברים:

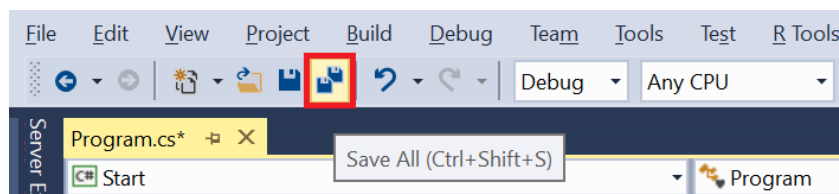
שורה 7: הפקודה מדפיסה על המסך את הטקסט המוקף בגרשיים.

שורה 8: התוכנית מחכה לבחירה (לחיצה) כלשהי מהמשתמש.

ננסה להריץ את הפרויקט, כלומר, להביא את התוכנית למצב שבו המחשב מבצע את כל שורות הקוד בזו אחר זו. עלינו לבצע שני שלבים:

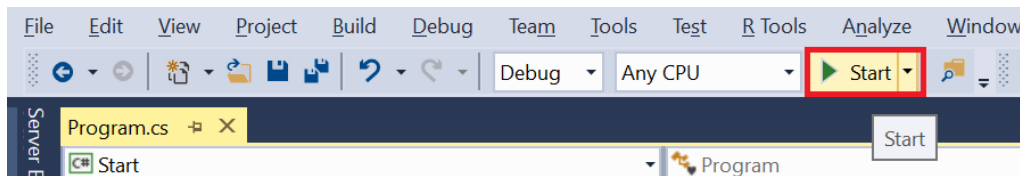
א. שלב ראשון – שמירה על קוד המקור:

עלינו לשמור את כל הקבצים (במקרה שהבאנו כדוגמה יש קובץ בודד). שימו לב כי ליד שם התוכנית (Program.cs*) מופיעה כוכבית. הכוכבית מציינת שהיו שינויים בקוד אך הם לא נשמרו. על מנת לא לאבד מידע, אנו ממליצים לשמור את כל רכיבי הפרויקט באמצעות לחיצה על הלחצן המסומן באדום:

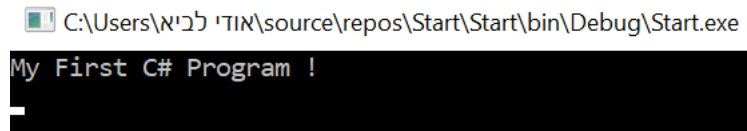


שימו לב כי לאחר השמירה נעלמה הכוכבית שהופיעה קודם ליד שם הקובץ (עד השינוי הבא).

ב. שלב שני – לחיצה על כפתור ההרצה:



אם עברנו שני שלבים אלו בהצלחה – אמור להופיע הפלט הבא:



שימו לב כי הפלט (תוצאת ההרצה) מוצג על המסך. נלחץ על מקש כלשהו, והשליטה/בקרה תחזור ל-C#.

טיפול בשגיאות תחביר

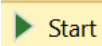
שגיאות תחביר (syntax) נפוצות מאוד. בדרך כלל מדובר באיות לא-נכון של פקודה או באי-כתיבה של סוגריים ו/או של נקודה ופסיק.

אין אדם שאינו טועה בשלב כזה או אחר. טעויות תחביר או טעויות הידור (קומפילציה) הן בדרך כלל טעויות שקל למצוא ואשר חוזרות על עצמן.

חשוב מאוד (!): בטעויות קומפילציה מצוין מספר השורה שבה מתרחשת הטעות.

Visual Studio מציגה טעויות תחביר בכמה דרכים:

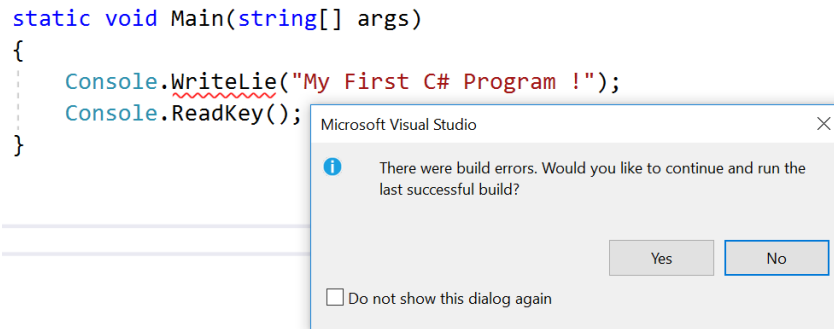
א. הדרך המקובלת ביותר היא סימון בצבע אדום בזמן אמת, כלומר, לפי שלב הרצת

התוכנית; כלומר, לפני הלחיצה על הכפתור הבא: 

לדוגמה,

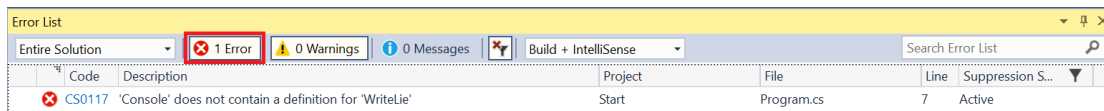
```
static void Main(string[] args)
{
    Console.WriteLine("My First C# Program !");
    Console.ReadKey();
}
```

ב. דרך נוספת היא לסמן לאחר שלב ניסיון ההרצה:



בכל פעם שאנו מקמפלים מלא, נוצר כאן קובץ בר-הרצה. כלומר, תהליך ההידור לא הושלם. עלינו לבחור ב-**No**, ולחזור לתקן את הטעות/טעויות. קובץ-בר הרצה הוא קובץ שהסיומת שלו היא EXE, והוא מסוגל לרוץ באופן עצמאי.

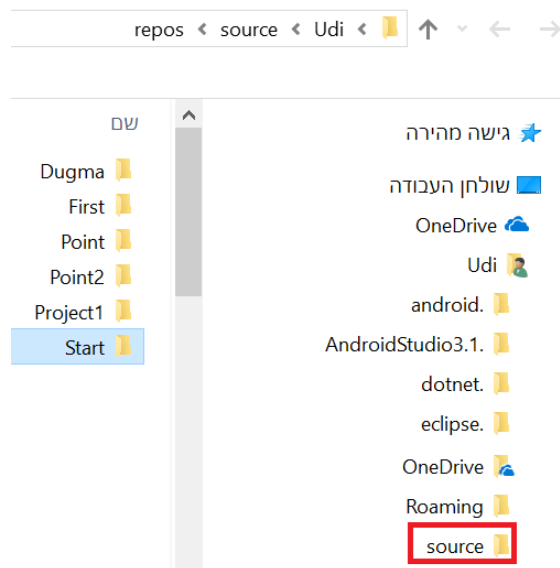
ג. בהמשך לסעיף הקודם, לאחר שלחצנו על **No**, נוכל לראות את מסך השגיאות:



נחזור ונאמר כי עלינו לתקן את השגיאה ולשמור את הפרויקט.

שמירה על פרויקט שלם בשפת C#

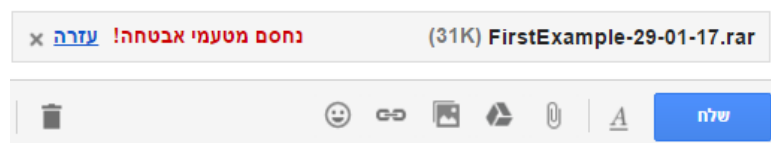
אם ברצונכם לשמור על גבי כונן נשלף (USB/Disk On Key) את התיקייה שעליה עבדתם במעבדה, כל שעליכם לעשות הוא להגיע לתיקייה שבה נשמר הפרויקט (אחרי ששמרתם את כל השינויים, כמובן) ולהעתיק אותה אל הכונן:



שליחת הפרויקט במייל

אם תרצו לשלוח לעצמכם למייל פרויקט שלכם, תיתקלו במספר בעיות:

1. לא ניתן לשלוח למייל תיקייה של קבצים, לכן צריך לכווץ את קובצי התיקייה לקובץ RAR אחד.
2. אם תנסו להעלות קובץ RAR לתיבת המייל שלכם, תגלו במהרה ש-Gmail לא מאפשר לשמור קבצים הכוללים מרכיבים שעלולים להיות פוגעניים (כמו למשל קבצים שהסיומת שלהם היא EXE – ובפרויקט שלנו אכן יש קבצים שזו הסיומת שלהם). במקרה כזה, תתקבל ההודעה הבאה:



הפתרון הוא אחת משתי האפשרויות הבאות:

1. לשמור את הקובץ המכווץ (של כל הפרויקט) ולהעלות אותו ל-Google Drive או ל-Dropbox.
2. לשלוח את קובץ ה-RAR דרך אתר <https://wettransfer.com> או דרך אתר www.jumbomail.co.il (או כל אתר שיתוף קבצים אחר):

שלב ראשון:

שלב שני:

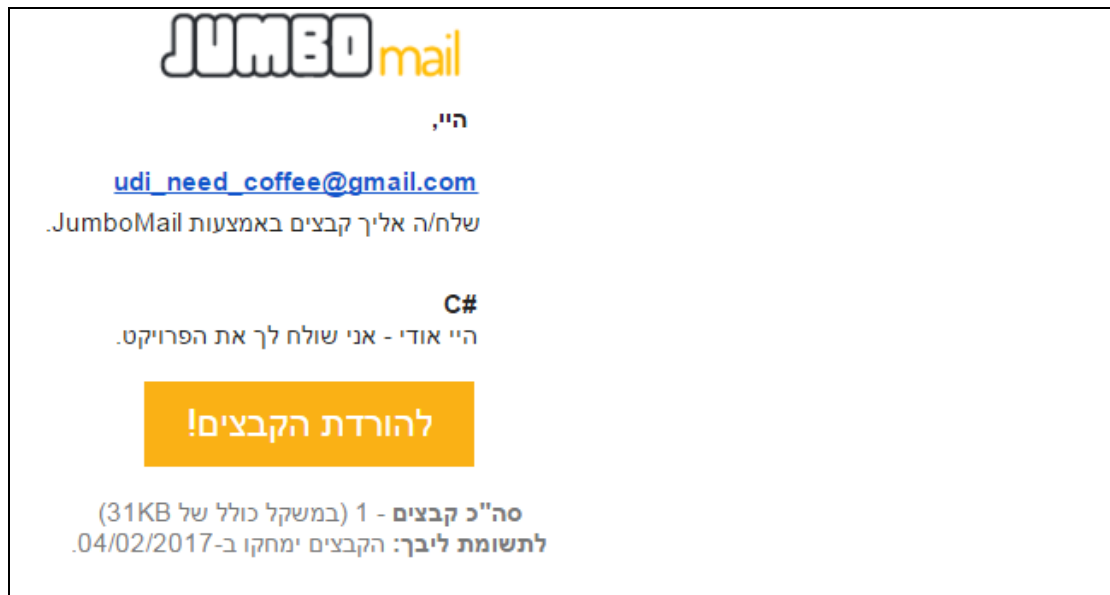
שלב שלישי:

נלחץ על:

שלב רביעי:

קבלת המייל עם הקישור להורדה:

ולאחר מכן הקובץ ייטען:



פתיחת פרויקט קיים

כדי לפתוח פרויקט קיים יש להיכנס לתיקייה המתאימה ולהפעיל קובץ שהסיומת שלו היא SLN (קיצור של Solution):

לחיצה על קובץ זה תפתח את כל הפרויקט:

שם	תאריך שינוי	סוג	גודל
.vs	03/06/2018 17:41	תיקיית קבצים	
Start	06/06/2018 18:40	תיקיית קבצים	
Start.sln	03/06/2018 17:41	Visual Studio Solution	2 KB

1.6 הכרת המושג עצם

בפרק 5 נדבר בהרחבה על מחלקות ועצמים.

עצם/אובייקט או מופע (Object או Instance) מורכב משילוב של נתונים (או שילוב של תכונות) ופעולות (התנהגויות) שהעצם "יודע" לבצע. לדוגמה, לעצם מסוג סמארטפון יש את הנתונים הבאים:

- א. שם יצרן
- ב. מספר אנשי קשר
- ג. צבע
- ד. מצב סוללה

באופן דומה, כל אובייקט (או עצם) מסוג סמארטפון מסוגל לבצע:

- א. חיוג למספר כלשהו.
- ב. שליחת SMS למספר כלשהו.
- ג. כיבוי הסמארטפון.
- ד. הדלקת הסמארטפון.

1.7 קריאה והבנה של ממשק פשוט של מחלקה קיימת, לצורך יצירת עצמים וזימון פעולה (Methods) על עצמים

בעולם התוכן של C# נדבר תמיד על אובייקט. האובייקט יהיה תמיד מושא הפעולה. איך נגדיר בשפת C# פקודה לסמארטפון לחייג לאיש קשר כלשהו (לדוגמה, "אבי")? זה ייראה כך:

```
("אבי")חייג_לאיש_קשר.סמארטפון
```

ואם נרצה לבצע פעולת "הנמך טמפרטורה" במזגן?

```
()הנמך_טמפרטורה.מזגן
```

ובאופן דומה, איך נגדיר את הפעולה "נעל את הבית"?

```
();נעל.בית
```

מה מבצעת הפקודה הבאה?

```
();הפעל.מזגן
```

הפקודה הזו מפעילה את האובייקט "מזגן".

בשפת C# נציין תחילה את האובייקט, ואחר כך נכתוב נקודה ואת שם המתודה המבצעת את המשימה.

1.8 יצירת עצמים באמצעות פקודת new

נניח כי אנו מבקשים "לייצר" סמארטפון.

בשפת C#, הפקודה תיראה כך:

```
PHONE one_plus_6 = new PHONE ("dani",054-2233942,"Tel-Aviv");
```

הפקודה new יוצרת אובייקט חדש מסוג "סמארטפון".

תובנות מהדוגמה הזו:

- א. קיים בזיכרון המחשב תא (ב-RAM) המכונה one_plus_6.
- ב. הפקודה new מייצרת אובייקט מסוג "סמארטפון" בכתובת מסוימת בזיכרון ה-RAM.

1.9 זימון וביצוע פעולות על עצמים

לדוגמה, הוראות הדפסה:

```
Console.WriteLine("same Line");
```

הפקודה Write מקבלת טקסט (מחרוזת) ומדפיסה אותו. הסמן נשאר באותה השורה. שימו לב כי הפקודה עובדת על מחלקת Console. מה זה אומר? בשלב זה עדיין איננו יודעים מהי המשמעות של זה, אבל חשוב להבין שחייבים להיות מחלקה או אובייקט שעליהם "עובדת" הפונקציה.

יש פונקציית הדפסה נוספת, דומה לקודמת:

```
Console.WriteLine("new Line");
```

הפעם הסמן יורד שורה לאחר ההדפסה.

נחזור ונדגיש כי תמיד יש:

```
מתודה.אובייקט או מחלקה
```

דוגמאות:

א.

```
()הצג_מצב_סוללה.סמארטפון
```

כל סמארטפון מסוגל להראות את מצב הסוללה שלו. אין זה משנה אם הדבר נעשה באמצעות ציור סוללה פיזית או על ידי ציון האחוזים. הרעיון הוא היכולת הזו באופן עקרוני.

ב. באופן דומה, סמארטפון מסוג מסוים (נניח, Samsung Galaxy Note 9) מסוגל להציג את 10 השיחות האחרונות שבוצעו בו.

```
(10)הצג_שיחות_אחרונות.Samsung_Galaxy_Note9
```

בדוגמה א' – הכוונה היא למחלקה. כל סמארטפון מכל סוג שהוא מסוגל להציג את מצב הסוללה שלו.

בדוגמה ב' – בחרנו סמארטפון בודד (אובייקט) מסוג המחלקה ועליו הפעלנו מתודה מסוימת.

פרק 2 – מושגי יסוד בתכנות

אלגוריתם

אלגוריתם הוא אוסף של הוראות (אחת או יותר) חד-משמעיות הניתנות לביצוע, אשר ביצוען יביא לפתרון של בעיה מסוימת. ההוראות חייבות להיות ממוספרות, וברורות למבצע המיועד (אדם או מכונה).

אלגוריתם חייב להסתיים לאחר מספר סופי של צעדים/פקודות.
האלגוריתם מתבצע לפי המספור שניתן על ידי המתכנת או כותב האלגוריתם.

כיצד מייצגים אלגוריתם? קיימות מספר דרכים:

דרך א' – אלגוריתם מילולי

רצף של הנחיות בשפה כלשהי או בשפה הדומה מאוד לשפה המדוברת. דוגמה קלאסית:

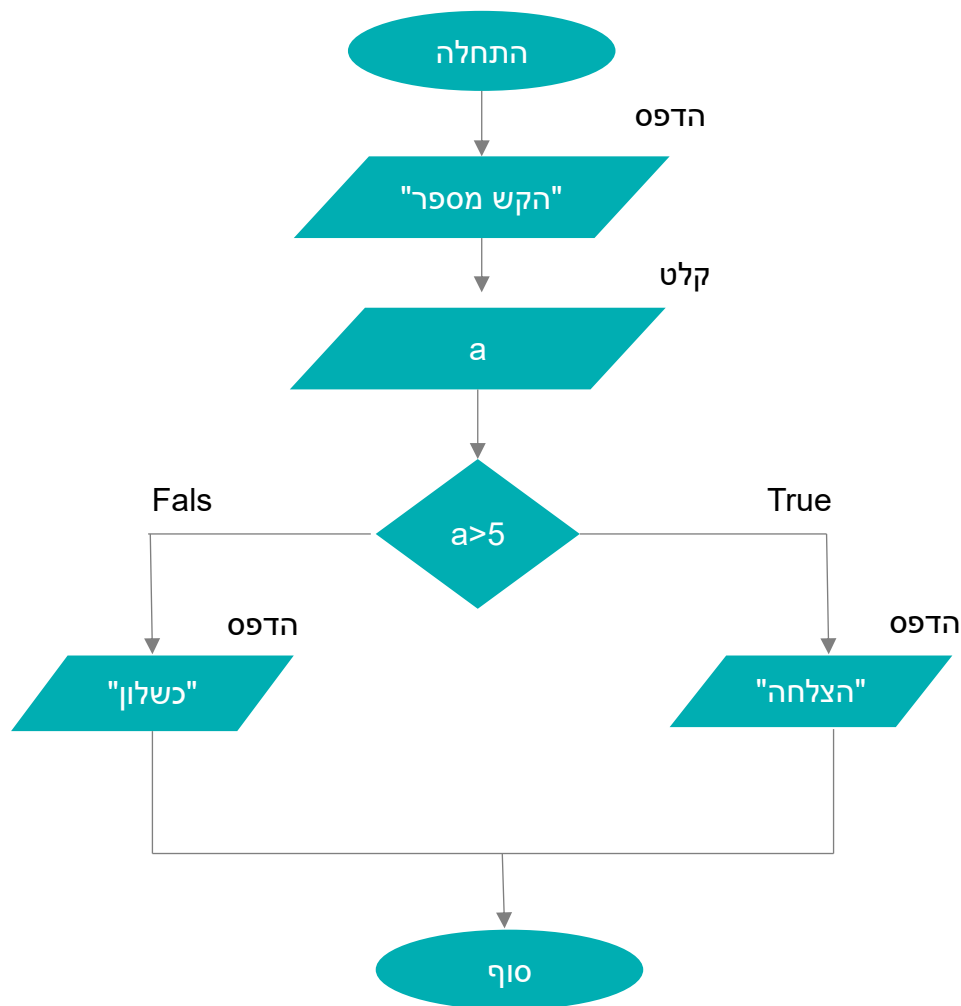
1. הדפס למסך את הפלט "הקש מספר".
2. קלוט מספר שלם לתוך תא זיכרון ששמו a.
3. אם ערכו של a גדול מ-5 הדפס "הצלחה"
אחרת, הדפס "כישלון".

דרך ב' – תרשים זרימה

זוהי למעשה שפה אוניברסלית – כל אדם (ללא תלות בשפה כזו או אחרת) יכול להבין אותה ולבצע את ההוראות. לשפה הזו יש מוסכמות, והן מופיעות בצורת תרשימים:

המשמעות	שם הצורה	הביטוי הגרפי
מציינת "התחלה" או "סוף" של האלגוריתם. בעלת תפקיד כפול.	אליפסה	
מציין השמה – הצבת ערך מסוים בתא, כלומר, הצבת משתנה מסוים בזיכרון.	מלבן	
מציינת קליטת נתונים או הדפסתם. בעלת תפקיד כפול.	מקבילית	
מציין בדיקת תנאי מסוים. תוצאת הבדיקה יכולה להיות "אמת" או "שקר".	מעוין	

דוגמה:



דרך ג' – פסאודו-קוד

שפה מילולית קרובה מאוד לשפת תכנות.

1. הדפס למסך את הפלט "הקש מספר".
2. קלוט מספר שלם לתא זיכרון בשם a.
3. אם הערך הנקלט גדול מ-5 הדפס "הצלחה" אחרת, הדפס "כישלון".

דרך ד' – שפת מחשב

כאן נרשום בשפת C#. אלו יהיו רוב הדוגמאות בספר זה. הרבה יותר קל לעבור לשפת מחשב כאשר מבינים לעומק את המונח אלגוריתם. למעשה, תוכנית המחשב היא סוג של אלגוריתם.

משתנה

זיכרון המחשב בנוי ממאות אלפים (לפחות) של תאים שונים. לכל תא במחשב יש כתובת פיזית (שנקבעת על ידי מערכת ההפעלה) ושם לוגי (שנקבע על ידי המתכנת). בכל תא שכזה יכול להיות ערך בודד, והוא עשוי להשתנות. זוהי מהות המונח *משתנה*. כל משתנה תופס מספר שונה של בתים (bytes), בהתאם לגודלו.

לדוגמה, נניח כי קיים בזיכרון המחשב תא מסוים ששמו grade ("ציון", באנגלית) והוא אמור להכיל ערך שלם:

```
int number = 89;
```

איננו יכולים לקבוע היכן תקצה לנו מערכת ההפעלה את התא הזה; אך, כאמור, יהיו לו שני שמות:

number	השם הלוגי של המשתנה
89	הערך התורן של המשתנה
3420	הכתובת הפיזית (או השם הפיזי) של המשתנה בזיכרון

כל משתנה שוכן בכתובת זיכרון מסוימת. שמות המשתנים קיימים רק לנוחותם של בני האדם.

טבלת מעקב

טבלת מעקב מאפשרת לנו לעקוב אחר האלגוריתם כדי לוודא שהוא עובד בצורה תקינה. טבלת המעקב צריכה להכיל עמודה עבור כל משתנה ועמודה עבור הפלט. כל פעולה/הוראה תופיע בשורה חדשה בטבלת המעקב.

נבדוק בהמשך את התנהלותם של פתרונות מסוימים באמצעות טבלאות מעקב.

דוגמה עבור האלגוריתם הבא:

- א. קלוט מספר שלם למשתנה בשם a.
- ב. אם הערך הנקלט גדול מ-5 הדפס "הצלחה" אחרת, הדפס "כישלון".

אפשרות ראשונה:

ערך המשתנה a	בדיקת תנאי	פלט (למסך)
		קלוט מספר שלם (לתא זיכרון בשם a).
12		
	$12 > 5 \rightarrow \text{true}$	
		"הצלחה"

אפשרות שנייה:

ערך המשתנה a	בדיקת תנאי	פלט (למסך)
		קלוט מספר שלם (לתא זיכרון בשם a).
2		
	$2 > 5 \rightarrow \text{false}$	
		"כישלון"

טיפוסי נתונים בסיסיים

קבוצה ראשונה

מספרים שלמים מסומנים (signed) – כלומר, גם מספרים חיוביים וגם מספרים שליליים:

שם הטיפוס	גודל הזיכרון (בבתים – bytes)	תחום הערכים
sbyte	1 בית (בייט)	128 עד 127
short	2 בתים	32,768 עד 32,767
int (integer)	4 בתים	2,147,483,648 עד 2,147,483,647
long	8 בתים	עד 9,223,372,036,854,775,808 עד 9,223,372,036,854,775,807

קבוצה שנייה

מספרים שלמים בלתי מסומנים (unsigned) – כלומר, מספרים חיוביים (ואפס) בלבד. ניסוח חלופי – מספרים "אי-שליליים".

שם הטיפוס	גודל הזיכרון (בבתים – bytes)	תחום הערכים
byte	1 בית	0 עד 255
ushort	2 בתים	0 עד 65,535
uint	4 בתים	0 to 4,294,967,295
ulong	8 בתים	0 to 18,446,744,073,709,551,615

קבוצה שלישית

מספרים שלמים מסומנים (signed) – כלומר, גם מספרים חיוביים (ואפס) וגם מספרים שליליים:

שם הטיפוס	גודל הזיכרון (בבתים – bytes)	תחום הערכים	רמת הדיוק / שבר עשרוני
float	4 בתים	$1.5 \cdot 10^{-45}$ עד $3.4 \cdot 10^{38}$	רמת דיוק של 7 ספרות אחרי הנקודה העشرונית
double	8 בתים	$5 \cdot 10^{-324}$ עד $1.7 \cdot 10^{308}$	רמת דיוק של 15 ספרות אחרי הנקודה העشرונית

קבוצה רביעית

שם הטיפוס	גודל הזיכרון (בבתים – bytes)	תחום הערכים	הערות
bool	1 בית	"False" ("שקר") או "True" ("אמת")	ביטוי בוליאני (בהמשך פרק זה)
char	1 בית	כל תו בודד הנמצא בטבלת ה-Unicode	טבלת Unicode עברית ב-Unicode
string	דינאמי בהתאם לגודלה של הטבלה	אוסף של תווים (אחד או יותר) הנמצאים בטבלת ה-Unicode	מופיע תחת גרשיים

א. מיינו את הערכים שלפניכם לסוגי משתנים אידיאליים (כלומר, משתנים התופסים כמות זיכרון מינימלית):

מספר	טיפוס נתונים	מספר	טיפוס נתונים
33000		240	
-1244		31,890	
345678		'\$'	
"Enter number"		75	
14.12345678		7>5	
89.575		1,787,444	
7,143,148,294,111,222,333		'Z'	
19,818,147,497,811,922,724		65,500	
-14,400		"F"	

פתרון:

מספר	טיפוס נתונים	מספר	טיפוס נתונים
33000	ushort	240	byte
-1244	short	31,890	short
345678	ushort	'\$'	char
"Enter number"	string	75	sbyte , byte
14.12345678	double	7>5	bool
89.575	float	1,787,444	int
7,143,148,294,111,222,333	long	'Z'	char
19,818,147,497,811,922,724	ulong	65,500	int
-14,400	short	"F"	string

פקודות השמה

האלגוריתם הבא מחשב ומדפיס את שכרו של אדם העובד כ-8 שעות ביום, 25 ימים בחודש, ומרוויח 89.4 ₪ לשעה.

```
int hours = 8;
// הסבר: הערך 8 מוצב במשתנה (מסוג שלם) hours.

int days = 25;
// הסבר: הערך 25 מוצב במשתנה (מסוג שלם) yamim.

double payment = 89.4;
// הסבר: הערך 89.4 מוצב במשתנה (מסוג ממשי) payment.

double total = yamim * hours * payment;
```

הסבר:

במשתנה total יחושב שכרו הכולל של העובד.

```
Console.WriteLine(total);
```

אפשר לשלב בין ההדפסות של ערך משתנה ושל טקסט. לדוגמה, מבנה כללי של פקודת ההדפסה:

```
Console.WriteLine("Salary :"+total);
```

או:

```
Console.WriteLine("Salary :{0}",total);
```

בשני המקרים, הפלט למסך יהיה זהה.

בכל מקרה מדובר בשרשור – משמע, חיבור של טקסט וערך של משתנה שהוסב לטקסט. למשל, אם נבקש לשרשר מחרוזת להדפסה הכוללת שני משתנים, נכתוב:

```
Console.WriteLine("Days : "+ yamim + " Salary: " + total);
```

או:

```
Console.WriteLine("Days : {0} Salary: {1}", yamim, total);
```

הפונקציה WriteLine מדפיסה טקסט מסוים, והסמן יורד שורה באופן אוטומטי.

הפונקציה Write מדפיסה טקסט מסוים, והסמן נשאר באותה שורה.

עבודה עם קלטים ופלטים במחשב

עבודה עם קלטים

מקרה א': שם המשתנה הוגדר למעלה (בתחילת התוכנית)

ב-5 השורות הראשונות מופיעה הצהרת משתנים.

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int integer;
        double floatPoint;
        char character;
        bool flag;
        string text;

        Console.WriteLine("Please enter : int,double,tav,bool & text.");
        integer = int.Parse(Console.ReadLine());
        floatPoint = double.Parse(Console.ReadLine());
        character = char.Parse(Console.ReadLine());
        flag = bool.Parse(Console.ReadLine());
        text = Console.ReadLine(); // No need to Parse it !

        Console.WriteLine("int is {0}, double is {1},char is {2} ,
                           bool is {3}, string is {4}.",
                           integer, floatPoint, character, flag, text);

        Console.ReadKey();
    }
}
```

פלט התוכנית לאחר הרצה:

```
Please enter : int,double,tav,bool & text.
9
4.6
$
true
test_text
int is 9, double is 4.6,char is $ ,bool is True, string is test_text.
```

טבלת מעקב

integer	floatPoint	character	flag	text	פלט
					Please enter : int,double,tav,bool & text.
9					
	4.6				
		\$			
			true		
				test_text	
					int is 9, double is 4.6,char is \$,bool is True, string is test_text.

הערה: הפלט יופיע באותה שורה (בטבלת המעקב הנ"ל אין מספיק מקום).

מקרה ב': הצהרת משתנים וקליטה באותה שורה

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Please enter : int,double,tav,bool & text.");
        int integer = int.Parse(Console.ReadLine());
        double floatPoint = double.Parse(Console.ReadLine());
        char character = char.Parse(Console.ReadLine());
        bool flag = bool.Parse(Console.ReadLine());
        string text = Console.ReadLine();

        Console.WriteLine("int is {0}, double is {1},char is {2} ,
                           bool is {3}, string is {4}.",
                           integer, floatPoint, character, flag, text);

        Console.ReadKey();
    }
}
```

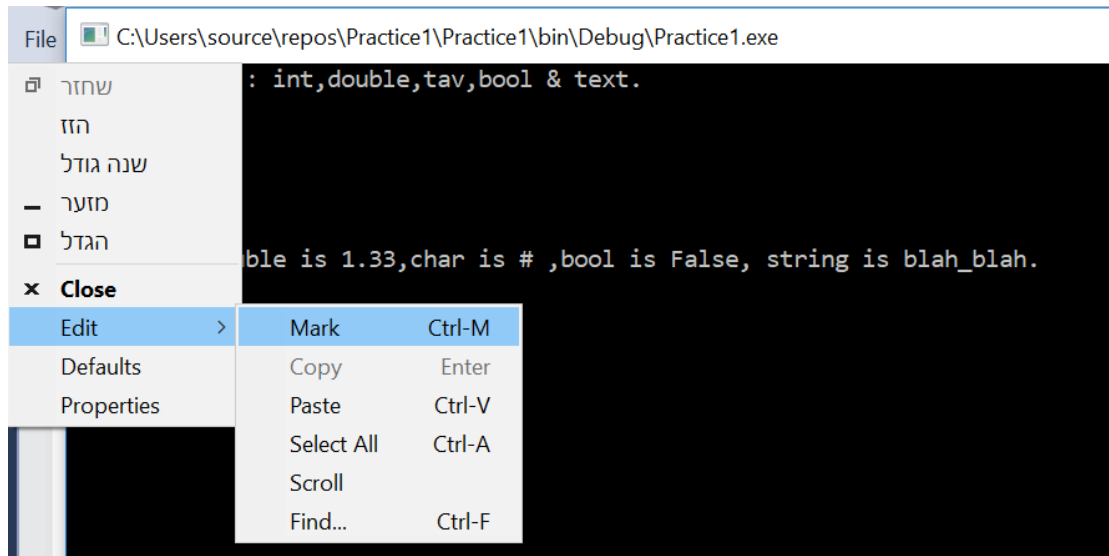
פלט התוכנית לאחר הרצה:

```
Please enter : int,double,tav,bool & text.
8
1.33
#
false
blah_blah
int is 8, double is 1.33,char is # ,bool is False, string is blah_blah.
_
```

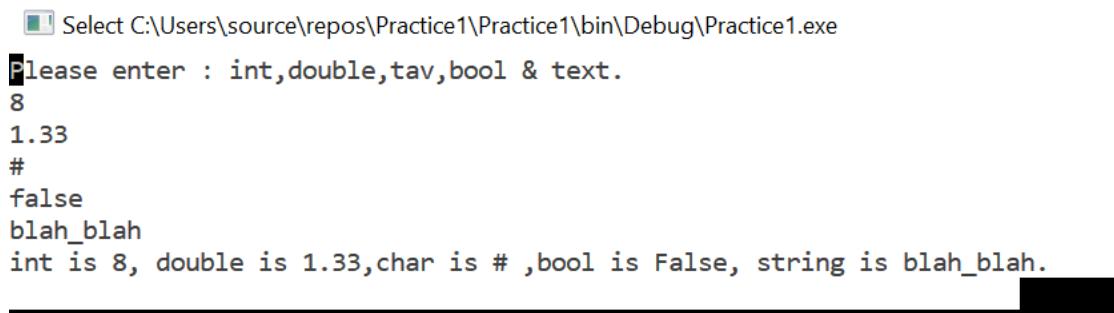
שמירת הפלט

א. נמקם את מצביע העכבר בפינה השמאלית-עליונה של חלון ה-Console (הפלט). שימו לב למלבן המסומן.

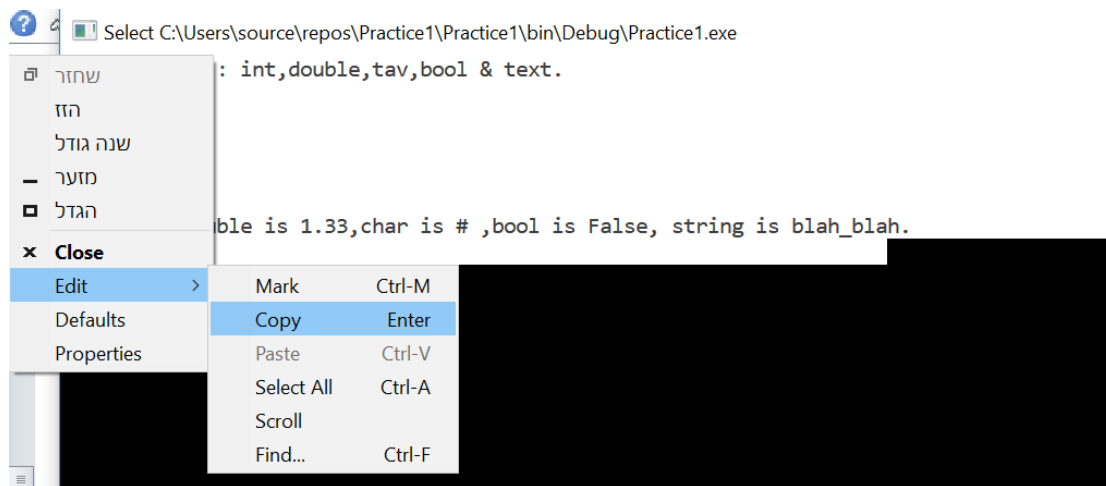
ב. נלחץ על הלחצן השמאלי של העכבר, ובתפריט שנפתח נבחר **Edit** ולאחר מכן **Mark**:



ג. נסמן – פיזית – את הפלט המבוקש (בצבע לבן):



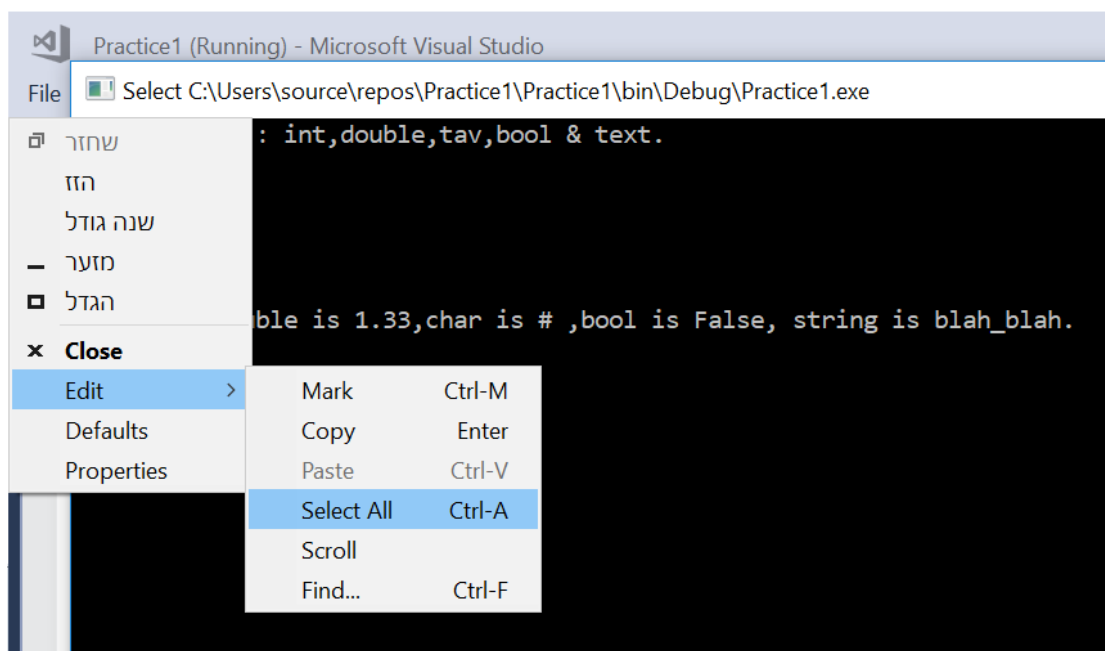
ד. נמקם שוב את המצביע של העכבר בפינה השמאלית-עליונה של חלון הפלט, נלחץ על הלחצן השמאלי ונבחר **Edit** ולאחר מכן **Copy**.



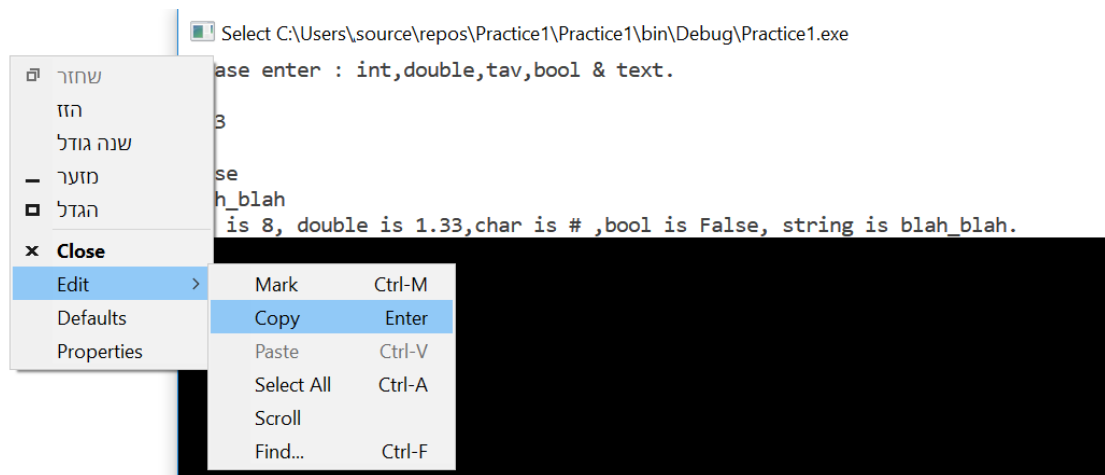
ה. בשלב זה, הפלט של התוכנית נמצא בלוח העבודה ואפשר להעתיק אותו (או את חלקו) לקובץ WORD, למשל, ו/או לערוך אותו בהמשך:

```
Please enter : int,double,tav,bool & text.
4
2.55
#
false
text_text
int is 4, double is 2.55,
char is # , bool is false, string is text_text.
```

וזוהי אפשרות מהירה יותר להעתקת הפלט כולו:



לאחר מכן נחזור על השלב הבא:



ואפשר להעתיק/לערוך את הטקסט:

```
Please enter : int,double,tav,bool & text.
8
1.33
#
false
blah_blah
int is 8, double is 1.33,char is # ,bool is False, string is blah_blah.
```

אופרטורים חשבוניים – חיבור, חיסור, חילוק, כפל ושארית

פעולת חיבור

+ (פלוס) הוא אופרטור בינארי, כלומר, הוא כולל **שני ערכים**:

```
int p;
p=3 + 8;
```

הוא מחבר את האופרנד השמאלי (3) לאופרנד הימני (8), ואת התוצאה הוא מציב במשתנה בשם p.

באופן דומה, האופרטור + עובד גם על מספרים ממשיים:

```
double x,y=3.2,z=1.1;
x=y+z;
```

שימו לב: אם נחבר מספר שלם למספר ממשי, התוצאה עתידה להיות מספר ממשי.

כך:

```
z=p+x=11+4.3=15.3
```

הערה: תוצאת חיבור המספרים 1.5 + 0.5 תהיה מסוג ממשי (ולא שלם).

פעולת חיסור

```
int a=5-2;
```

המשתנה a יקבל את הערך 3.

```
double c=10-a;
```

המשתנה c יקבל את הערך 7.0.

הסבר: c הוא משתנה מסוג ממשי (שבר עשרוני); לכן, בכל מקרה אמורה להיות נקודה עשרונית.

פעולת חילוק

כאשר מדובר ב-2 מספרים שלמים – לעולם תהיה התוצאה מספר שלם.

כלומר, תוצאת הפעולה $10/3$ (10 לחלק ל-3) היא **3** (ולא 3 ושליש!).

```
Console.WriteLine(10/3);
```

ההדפסה תהיה, כאמור, 3.

ההדפסה של הפקודה הבאה שונה:

```
Console.WriteLine(10/3.0);
```

ההדפסה תהיה 3.0 (כלומר, ערך ממשי/שבר).

וזוהי אפשרות זאת לקבלת ערך ממשי:

```
Console.WriteLine(10/(double)3);
```

למעשה התבצעה כאן **המרה**: ברגע החלוקה הפכנו את הערך 3 לשבר. למעשה, הערך לא השתנה אך תוצאת החילוק השתנתה.

פעולת כפל

באופן דומה, האופרטור * (כוכבית) משמש כפעולת פלט.

גם במקרה זה האופרטור הוא בינארי, כלומר, הוא כולל **שני ערכים**:

```
static void Main(string[] args)
{
    int a = 3, b = 5;
    double c = 1.5, d = 4;
    Console.WriteLine("{0} X {1} = {2}", a, b, a*b);
    Console.WriteLine("{0} X {1} = {2}", c, d, c*d);

    Console.ReadKey();
}
```

והפלט יהיה בהתאמה:

$$3 \times 5 = 15$$

$$1.5 \times 4 = 6$$

פעולת שארית

10 לחלק ל-2: התוצאה (השלמה) היא 5 ואין שארית. או, במילים אחרות, התוצאה היא אפס. למעשה:

$$10 / 2 = 5$$

$$10 \% 2 = 0$$

כמה זה?

$$14 \% 5 = ?$$

השארית היא 4. תוצאת הפעולה 14 לחלק ל-5 היא 2. השארית היא 4.

על כן, אפשר לכתוב כך: $14 \% 5 = 4$.

טעות נפוצה(!): כמה זה?

$$2 / 5 = ?$$

התוצאה היא אפס – מהסיבה הפשוטה שהערך 5 לא "נכנס" ב-2. אין חלוקה. לעומת זאת, מהי תוצאת הפעולה הבאה?

$$2 \% 5 = ?$$

התוצאה היא 2 (ולא אפס!) – מפני שכלל לא היתה חלוקה ולכן "נשארנו" עם אותו ערך, כלומר, 2.

פעולות מתמטיות בסיסיות: sqrt, round, pow, min, max, abs

בשפת C# קיימת מחלקה מובנית בשם Math, המאגדת בתוכה לא מעט פונקציות המטפלות באוסף פעולות (פונקציות) מתמטיות.

הנה כמה דוגמאות:

מה מבצעות הפעולות הבאות?	
דוגמה מספר 1	<code>Console.WriteLine(Math.Abs(-3));</code>
תשובה	קיימת פונקציה מובנית המקבלת מספר שלם (int) ומחזירה את הערך המוחלט שלו. הפלט יהיה 3.
דוגמה מספר 2	<code>Console.WriteLine(Math.Abs(-3.32));</code>
תשובה	באופן דומה, קיימת פונקציה מובנית המקבלת מספר ממשי (float/double) ומחזירה את הערך המוחלט שלו. הפלט יהיה 3.32.
דוגמה מספר 3	<code>Console.WriteLine(Math.Round(95.45));</code>
תשובה	הפעולה מקבלת מספר ממשי ומעגלת אותו למספר השלם הקרוב. הפלט יהיה 95.
דוגמה מספר 4	<code>Console.WriteLine(Math.Round(95.75));</code>
תשובה	באופן דומה, מה יהיה הפלט עבור הפקודה הבאה? הפלט יהיה 96.

במחלקת Math קיימות גם הפונקציות הבאות, שהן פונקציות טריגונומטריות:

הערה: כל הפונקציות הללו מקבלות ומחזירות ערכים ממשיים בצורת רדיאנים (ולא מעלות).

מטרת הפונקציה	דוגמת הפעלה	פלט
חישוב סינוס	<code>Console.WriteLine(Math.Sin(-90));</code>	-0.893996663600558
פעולה הפוכה לסינוס	<code>Console.WriteLine(Math.Asin(0.3));</code>	0.523598775598299
חישוב קוסינוס	<code>Console.WriteLine(Math.Cos(25));</code>	0.991202811863474
פעולה הפוכה לקוסינוס	<code>Console.WriteLine(Math.Acos(0.7));</code>	0.795398830184144
פעולת טנגנס	<code>Console.WriteLine(Math.Tan(25));</code>	-0.133526407021536
פעולה הפוכה לטנגנס	<code>Console.WriteLine(Math.Atan(0.7));</code>	0.610725964389209

פונקציות נוספות במחלקת Math:

מטרת הפונקציה	דוגמת הפעלה	פלט
מציאת המספר הגדול ביותר	<code>Console.WriteLine(Math.Max(2,4));</code>	4
	<code>Console.WriteLine(Math.Max(2.1,4.6));</code>	4.6
מציאת המספר הקטן ביותר	<code>Console.WriteLine(Math.Min(11,-2));</code>	-2
	<code>Console.WriteLine(Math.Min(11.1,-2.99));</code>	-2.99
חישוב שורש ריבועי	<code>Console.WriteLine(Math.Sqrt(16));</code>	4
פעולת חזקה	<code>Console.WriteLine(Math.Pow(4,2));</code>	16

נציג כמה דוגמאות:

```

16 static void Main(string[] args)
17 {
18
19     Random rnd = new Random(); // new -> to be continue ...
20     int mis1 = rnd.Next(10); // 0 .. 9
21     Console.WriteLine("mis1 = {0}", mis1);
22     int mis2 = rnd.Next(5, 10); // 5 .. 9
23     Console.WriteLine("mis2 = {0}", mis2);
24     int mis3 = rnd.Next(); // return random integer value
25     Console.WriteLine("mis3 = {0}", mis3);
26
27     Console.ReadKey();
28 }

```

מספר השורה	הסבר
19	בשורה זו אנו יוצרים אובייקט מטיפוס Random. נרחיב על כך בפרק 5.
20	המשתנה mis1 יקבל לתוכו ערך רנדומלי (אקראי) בין 0 ל-10, כלומר, 9.
21	בשורה זו יודפס ערכו של המשתנה mis1.
22	המשתנה mis2 יקבל לתוכו ערך רנדומלי (אקראי) בין 5 ל-10, כלומר, 9.
23	בשורה זו יודפס ערכו של המשתנה mis2.
24	המשתנה mis3 יקבל לתוכו ערך רנדומלי (אקראי) בגבולות ה-integer.
25	בשורה זו יודפס ערכו של המשתנה mis3.

הערה: את הפקודות העתקנו לכאן רק לצורך העריכה:

```

Random rnd = new Random(); //new משמעות הפקודה
int mis1 = rnd.Next(10); // 0 .. 9
Console.WriteLine("mis1 = {0}", mis1);
int mis2 = rnd.Next(5, 10); // 5 .. 9
Console.WriteLine("mis2 = {0}", mis2);
int mis3 = rnd.Next(); // return random integer value
Console.WriteLine("mis3 = {0}", mis3);

```

3 דוגמאות הרצה

mis1 = 7	mis1 = 4	mis1 = 8
mis2 = 5	mis2 = 7	mis2 = 5
mis3 = 583491644	mis3 = 883725993	mis3 = 1344387900

חזרה למחלקת Math

בנוסף למתודות (פונקציות), כוללת מחלקת Math קבועים מסוימים. לדוגמה:

א. קבוע פאי (PI):

```
Console.WriteLine(Math.PI);
```

א. קבוע אפסילון (E):

```
Console.WriteLine(Math.E);
```

והפלט יהיה:

3.14159265358979

2.71828182845905

דוגמאות לפיתוח אלגוריתם

דוגמה מספר 1

כתוב תוכנית מחשב שתקלוט את מחירו של כרטיס כניסה למתחם פעילות בקניון (מעין ג'ימבורי), וכן את מספר הילדים האמורים להיכנס למתחם. האלגוריתם יחשב וידפיס את מחירם הכולל של הכרטיסים.

שלב ראשון – הצהרת משתנה:

```
int kids;  
double price;  
double total;
```

יש לרשום עבור כל תוכנית "מקרא משתנים". כלומר, לכל משתנה יש לציין מהו סוג המשתנה ומה הוא מייצג. בדוגמה הזו:

Kids – משתנה מסוג שלם, מייצג את מספר הילדים האמורים להיכנס למתחם הפעילות.

Price – משתנה מסוג ממשי, מייצג את מחירו של כרטיס כניסה בודד.

Total – משתנה מסוג ממשי, מייצג את מחירם הכולל של כרטיסי הכניסה (עבור כל הילדים שאמורים להיכנס למתחם).

שלב שני – הדפסת הנחיות למשתמש:

ככלל, לפני כתיבת בקשה כלשהי מהמשתמש (הקלט), תמיד **נקליד בצורה מסודרת** למה התוכנית מצפה. רק לאחר שנקליד בקשה ברורה וחד-משמעית נוכל להתקדם: לבצע קלטים, לבצע חישובים, ובדרך כלל להציג בסוף התוכנית פלטים מסודרים.

עבור המשתנה הראשון (מחיר כרטיס בודד)

פלט: הנחיה למשתמש – "אנא הקש מחיר כרטיס":

```
Console.WriteLine("Please enter ticket's price");
```

קלט: מידע שהמשתמש מקיש –

```
price = double.Parse(Console.ReadLine());
```

עבור המשתנה השני (מספר הילדים)

פלט: הנחיה למשתמש – "אנא הקש את מספר הילדים שאמורים להיכנס"

```
Console.WriteLine("Please enter How many childrens");
```

קלט: מידע שהמשתמש מקיש –

```
kids = int.Parse(Console.ReadLine());
```

עבור המשתנה השלישי (המחיר הכולל)

כאן אין לנו צורך בקלט, אלא בחישוב המחיר הכולל:

```
total = price * kids;
```

ובסוף, הדפסת הפלט (המחיר הסופי):

```
Console.WriteLine("Total price is {0}.", total);
```

כעת נציג את הקוד במלואו:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int kids;
        double price;
        double total;

        Console.WriteLine("Please enter ticket's price");
        price = double.Parse(Console.ReadLine());

        Console.WriteLine("Please enter How many childrens");
        kids = int.Parse(Console.ReadLine());

        total = price * kids;

        Console.WriteLine("Total price is {0}.", total);

        Console.ReadKey();
    }
}
```

טבלת מעקב

Price	Kids	Total	פלט
			אנא הקש מחיר לכרטיס
28.5			
	20		
		570.0	
			המחיר הכולל הוא 570.

דוגמה מספר 2

כתוב תוכנית מחשב שהקלטים שלה (מספרים ממשיים) יהיו:

- א. אורך צלע של משולש.
- ב. גובה של משולש.

יש לבצע את חישוב השטח על פי הנוסחה הבאה:

$$\text{שטח משולש} = (\text{גובה} \times \text{בסיס}) / 2$$

שלב ראשון – הצהרת משתנה:

```
double orech;  
double gova;  
double shetach;
```

שלב שני – הדפסת הנחיות למשתמש:

פלט (הנחיה עבור שני קלטים):

```
Console.WriteLine("Please enter orech and gova");
```

קלט לשני המשתנים:

```
orech = double.Parse(Console.ReadLine());  
gova = double.Parse(Console.ReadLine());
```

חישוב השטח:

```
shetach =(orech * gova)/2;
```

פלט של התוצאה:

```
Console.WriteLine("The surface is {0}.", shetach);
```

כעת נציג את הקוד במלואו:

```
using System;  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        double orech;  
        double gova;  
        double shetach;  
        Console.WriteLine("Please enter orech and gova");  
        orech = double.Parse(Console.ReadLine());  
        gova = double.Parse(Console.ReadLine());  
  
        shetach =(orech * gova)/2;  
  
        Console.WriteLine("The surface is {0}.", shetach);  
        Console.ReadKey();  
    }  
}
```

הערה: במקרה זה לא היינו צריכים לבצע המרה לערכו של המשתנה shetach מאחר שמדובר במשתנה מסוג שבר עשרוני.

תרגול

בית חולים לילדים ביקש למכור כרטיסי כניסה למתחם הג'ימבורי על מנת לאפשר לילדים המאושפזים לקבל מתנות לחג הפסח הקרב ובא.

הקלט

א. מספר הילדים המאושפזים בבית החולים

ב. מחיר כרטיס כניסה

יש לחשב ולהדפיס:

א. כ-15% מההכנסות עתידות לממן את עלות הדפסת הכרטיסים. יש להקליד סכום זה.

ב. כ-18% מההכנסות עתידות לממן את עלות כרטיסי הכניסה למתחם הג'ימבורי עבור הילדים המאושפזים בבית החולים.

יש לחשב את העלות הנ"ל ולהקליד את הנתון המתקבל.

ג. שאר הכסף אמור להיות מופנה לטובת בית החולים לילדים.

יש לחשב את הסכום הנ"ל ולהקליד אותו.

האופרטור קידום/הפחתה עצמיים

את האופרטור ++ או -- אפשר לכתוב לפני (pre) המשתנה או אחרי (post).

כאשר נשתמש באופרטור לפני המשתנה, ראשית כול יקודם המשתנה ולאחר מכן נוכל להשתמש בערכו החדש.

לעומת זאת, כאשר האופרטור יופיע אחרי המשתנה, נשתמש תחילה בערכו הקודם של המשתנה, ורק לאחר מכן יופחת או יקודם ערכו.

דוגמה:

<code>static void Main(string[] args)</code>	
<code>{</code>	
<code>int a = 5, b = 10, q = 8, p = 13;</code>	
<code>int c, d, e, f;</code>	
<code>c = a++;</code>	כאן האופרטור מופיע אחרי המשתנה a; לכן, ראשית כול תבוצע ההשמה, ורק לאחר מכן יתבצע הקידום.
	בסוף הוראה זו, ערכו של המשתנה c יהיה 5 ואילו ערכו של a יהיה 6.
<code>d = ++b;</code>	האופרטור מופיע לפני המשתנה b, ולכן ראשית כול יקודם המשתנה b ורק לאחר מכן תתבצע ההשמה.
	ביום הפקודה, ערכו של b יהיה 11 וגם ערכו של d יהיה 11.
<code>e = p--;</code>	
<code>f = --q;</code>	
<code>Console.WriteLine("a={0} b={1} c={2} d={3} ",a,b,c,d);</code>	
<code>Console.WriteLine("a={0} b={1} c={2} d={3}. ",e,f,p,q);</code>	

טבלת מעקב

a	b	q	p	c	d	e	f	פלט
5								
	10							
		8						
			13					
				5				
6								
	11							
					11			
						13		
			12					
		7						
							7	
								6 11 5 11 13 7 12 7
								הורדת הסמן

חלוקת עבודה תחת פרויקט – מיועד למורים בלבד

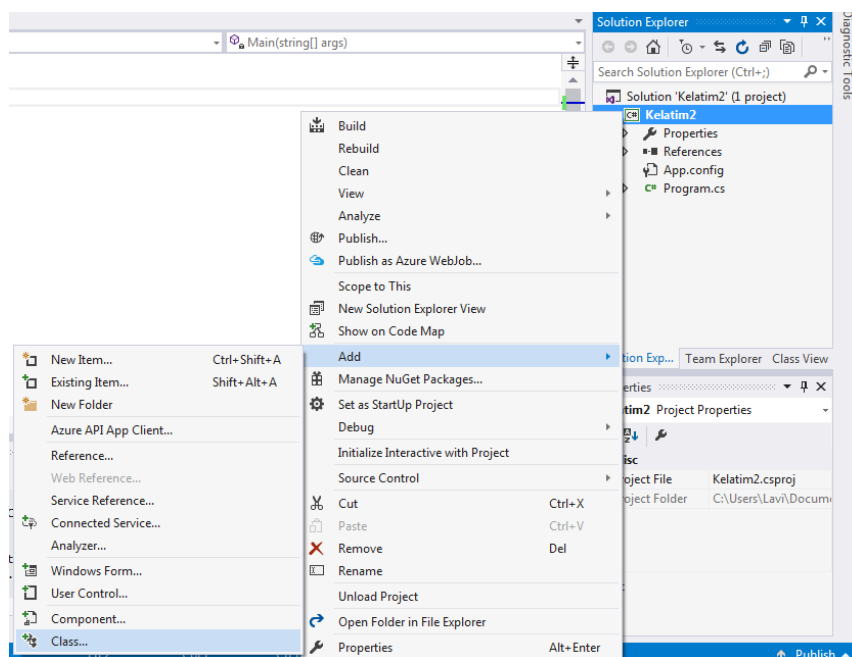
א. עבודה עם פרויקט בעל מחלקה אחת בלבד

למעשה, מה שביצענו עד עכשיו.

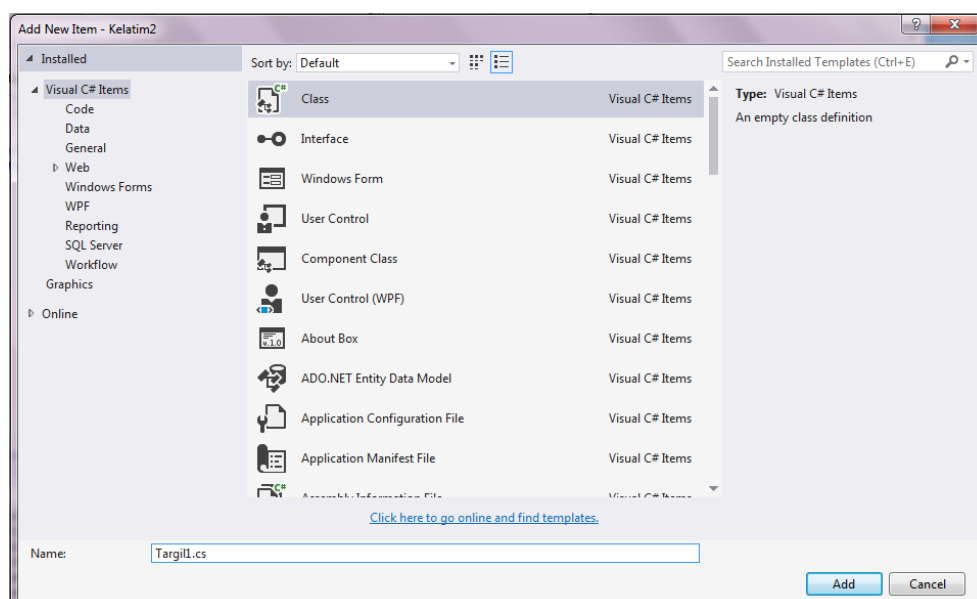
ב. עבודה עם פרויקט בעל יותר ממחלקה אחת:

נפתח פרויקט חדש בשם Kelatim2. נמקם את העכבר על שם הפרויקט:

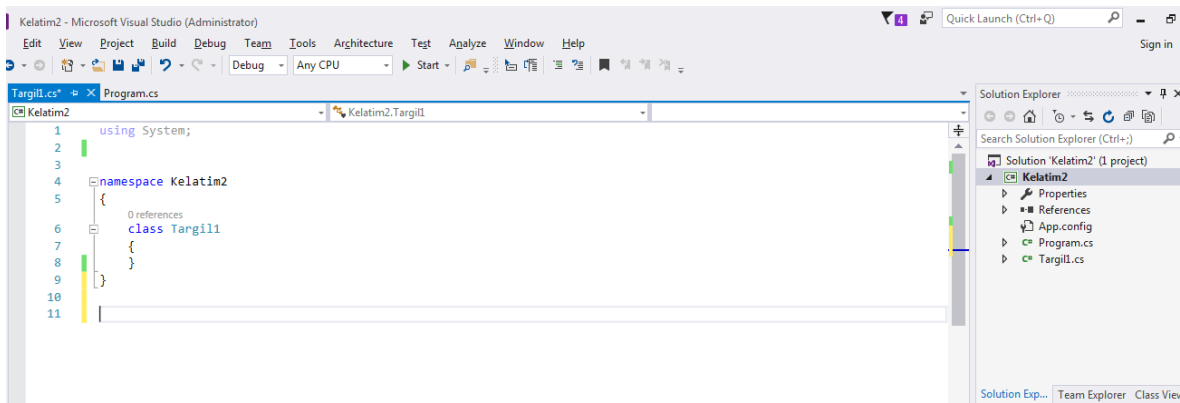
Project Name → Add → Class (ימני עכבר)



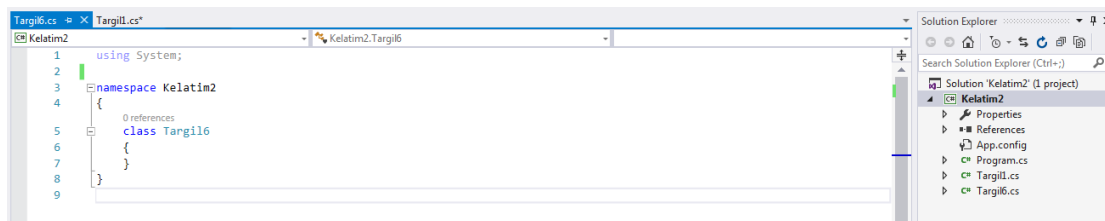
יפתח החלון הבא:



נשנה את שם המחלקה בהתאמה (בניח, "תרגיל 1"), ונתאים אותו למוסכמה:



נחזור על פעולה זו וניצור את מחלקת "תרגיל 6":



נעתיק את הפונקציות בהתאמה (אל דאגה – נציג פתרון מפורט בהמשך).

טעויות נפוצות

א. טעות קלאסית היא לא לתת לפונקציה הרשאה ציבורית (public). כמו למשל עבור הפונקציה הבאה:

```
using System;

class Targil1
{
    public static void Daf1_Ex1() // // Add the public notation ! without it, we have a problem !
    {
        Console.WriteLine("Solution of Ex1.");
        Console.WriteLine("Please enter double number.");
        int shalem;
        double shever = double.Parse(Console.ReadLine());
        Console.WriteLine("The number is {0}.", shever);
        shalem = (int)shever; // Allocate int part of it.
        Console.WriteLine("The int section of it is: {0}.", shalem);
        Console.WriteLine("The real section of it is {0}.", shever - shalem);
    }
}
```

דוגמה נוספת: עבור המחלקה הראשית (שבתוכה ממוקמת הפונקציה static void Main(string[] args):

```
using System;
0 references
class Program
{
0 references
static void Main(string[] args)
{
Console.WriteLine("Please enter question number (1-15). \n[for now question 1 or 6]");
int task = int.Parse(Console.ReadLine()); // scanf("%d",&task);
// Nested if
if (task == 1)
Targil1.Daf1_Ex1();
else if (task == 6)
Targil6.Daf1_Ex6();
else Console.WriteLine("No Question to solve (yet).");

Console.ReadKey();
}
```

1 Error 0 Warnings 0 Messages Build + IntelliSense

Description	Project	File	Line	Suppression
'Targil1.Daf1_Ex1()' is inaccessible due to its protection level	Kelitim2	Program.cs	10	Active

ב. טעות קלאסית נוספת היא לכתוב – סביר להניח שמתוך הרגל – פונקציית Main נוספת במחלקה אחרת. נניח שבמחלקת Targil6 התווספה הפונקציה המדוברת:

```
using System;

class Targil6
{
    public static void Daf1_Ex6() // Add the public notation ! without it, we have a problem !
    {
        int tlat;
        Console.WriteLine("Solution of Ex6:");
        Console.WriteLine("Please enter three digits number");
        tlat = int.Parse(Console.ReadLine());
        if (!(tlat >= 100) && (tlat <= 999))
        {
            Console.WriteLine("It is not three digits number.");
        }
        else
        {
            Console.WriteLine(tlat / 100 + " " + tlat / 10 % 10 + " " + tlat % 10);
        }
    }

    static void Main(string[] args)
    {
    }
}
```

שימו לב להודעת השגיאה הבאה:

1>C:\Users\Lavi\Documents\Visual Studio
2015\Projects\electronics\Kelatim2\Kelatim2\Program.cs(4,21,4,25): error CS0017: Program has
more than one entry point defined. Compile with /**main to specify the type that contains the entry
point.**

===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====

אפשרות א'

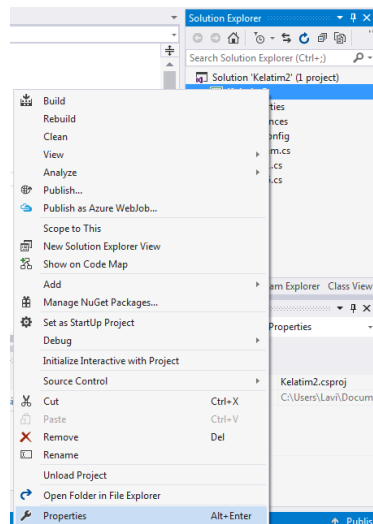
האפשרות הראשונה היא, כמובן, לוודא שהפונקציה Main תופיע בפרויקט פעם אחת בלבד.

אפשרות ב'

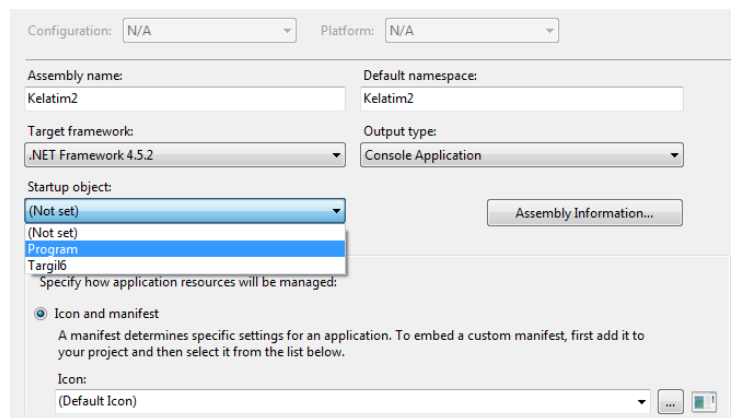
להורות למערכת VS לאיזה קובץ Main ברצוננו להתייחס.

נעמוד על שם הפרויקט:

Project Name (לחצן ימני של העכבר) □ Properties



ובמסך הבא נחליט לאיזו פונקציה אנחנו מתכוונים:



נציג כאן, כמובטח, את הקוד המלא (ללא טעויות), ואחר כך קישור להורדה.

המחלקה הראשית:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Please enter question number (1-15). \n[for now question 1 or 6]");
        int task = int.Parse(Console.ReadLine()); // scanf("%d",&task);
        // Nested if
        if (task == 1)
            Targil1.Daf1_Ex1();
        else if (task == 6)
            Targil6.Daf1_Ex6();
        else Console.WriteLine("No Question to solve (yet).");

        Console.ReadKey();
    }
}
```


המחלקה השנייה:

```
using System;

class Targil1
{
    public static void Daf1_Ex1() // // Add the public notation ! without it, we have a problem !
    {
        Console.WriteLine("Solution of Ex1:");
        Console.WriteLine("Please enter double number.");
        int shalem;
        double shever = double.Parse(Console.ReadLine());
        Console.WriteLine("The number is {0}.", shever);
        shalem = (int)shever; // Allocate int part of it.
        Console.WriteLine("The int section of it is: {0}.", shalem);
        Console.WriteLine("The real section of it is {0}.", shever - shalem);
    }
}
```

המחלקה השלישית:

```
using System;

class Targil6
{
    public static void Daf1_Ex6() // Add the public notation ! without it, we have a problem !
    {
        int tlat;
        Console.WriteLine("Solution of Ex6:");
        Console.WriteLine("Please enter three digits number");
        tlat = int.Parse(Console.ReadLine());
        if (!(tlat >= 100) && (tlat <= 999))
        {
            Console.WriteLine("It is not three digits number.");
            Console.WriteLine("Please shoot yourself !");
        }
        else
        {
            Console.WriteLine(tlat / 100 + " " + tlat / 10 % 10 + " "
                               + tlat % 10);
        }
    }
}
```

פרק 3 – ביצוע מותנה

בפרק זה נלמד לעבוד עם מגוון **משפטי בקרה** על מנת לאפשר כתיבת תוכניות יעילות. משפט בקרה הוא משפט המגדיר אילו חלקים בתוכנית עתידים להתבצע על פי תנאים מסוימים. נלמד להגדיר את משפטי הבקרה השונים בשפת #C, ונציג דוגמאות רבות ומגוונות לשימוש בהם.

טיפוס הנתונים bool

טיפוס הנתונים מסוג bool יכול להכיל ערך מסוג "True" (כלומר, "אמת") או ערך מסוג "False" (כלומר, "שקר"). נשתמש בו על מנת לבדוק את קיומם של מצבים מסוימים בהמשך – לדוגמה, האם מספר מסוים שנקלט מהווה ציון חוקי.

דוגמה מספר 1

בתוכנית הבאה, a הוא משתנה מסוג bool וערכו מאותחל ל-"true" ("אמת"). לכן ההדפסה עתידה להיות: "a contains True." (שימו לב לנקודה בסוף המשפט).

התוכנית:

```
static void Main(string[] args)
{
    bool a=true;
    Console.WriteLine("a contains {0}.",a);
    Console.ReadKey();
}
```

פלט התוכנית:

a contains True.

הערה: ערך הביטוי הוא "true" למרות שבפלט מופיעה המילה True. לא להתבלבל!

ביטויים בוליאניים

פעולה בוליאנית היא פעולה (או פקודה) שהערך שלה (כתוצאה) יכול להיות "אמת" או "שקר".

דוגמה מספר 2

בדוגמה זו יחושב תחילה ערך הביטוי הבוליאני. כלומר, האם ערכו של המספר 5 גדול מערכו של המספר 8?

5 אינו גדול מ-8, ולכן ערך פעולה (או השוואה) זו מחזיר את הערך הבוליאני "False" ("שקר").
לכן ההדפסה עתידה להיות "a contains False." (שימו לב לנקודה בסוף המשפט).

```
static void Main(string[] args)
{
    bool a=(5>8);
    Console.WriteLine("a contains " + a + ".");
    Console.ReadKey();
}
```

פלט התוכנית:

a contains False.

יחס: כמו למשל שווה, שונה, גדול, קטן, גדול או שווה, קטן או שווה

יחס הוא ביטוי לוגי שהתוצאה שלו היא ערך מטיפוס בוליאני. נציג מספר יחסים:

בכל הדוגמאות הבאות נניח כי $a=3$ וכי $b=5$.

דוגמה	המשמעות המילולית	כתיבה ב-C#
הביטוי $a==b$ יחזיר את הערך False.	האם ערכו של a שווה לערכו של b?	$a==b$
הביטוי $a!=b$ יחזיר את הערך True.	האם ערכו של a שונה (כלומר, לא שווה) מערכו של b?	$a!=b$
הביטוי $a>b$ יחזיר את הערך False.	האם ערכו של a גדול מערכו של b?	$a>b$
הביטוי $a>=b$ יחזיר את הערך False.	האם ערכו של a גדול או שווה לערכו של b?	$a>=b$
הביטוי $a<b$ יחזיר את הערך True.	האם ערכו של a קטן מערכו של b?	$a<b$
הביטוי $a<=b$ יחזיר את הערך True.	האם ערכו של a קטן או שווה לערכו של b?	$a<=b$

דוגמה מספר 3

מה יהיה הפלט של התוכנית הבאה?

```
static void Main(string[] args)
{
    int a = 3;
    int b = 5;

    Console.WriteLine(a == b); // (1)
    Console.WriteLine(a != b); // (2)
    Console.WriteLine(a > b); // (3)
    Console.WriteLine(a >= b); // (4)
    Console.WriteLine(a < b); // (5)
    Console.WriteLine(a <= b); // (6)

    Console.ReadKey();
}
```

פתרון והסבר:

שורה מספר	הפלט הוא	הסבר
(1)	False	הערך 3 אינו שווה לערך 5. לכן ערך הביטוי הוא "שקר".
(2)	True	הערך 3 שונה מהערך 5. לכן ערך הביטוי הוא "אמת".
(3)	False	הערך 3 אינו גדול מהערך 5. לכן ערך הביטוי הוא "שקר".
(4)	False	הערך 3 אינו גדול או שווה לערך 5. לכן ערך הביטוי הוא "שקר".
(5)	True	הערך 3 קטן מהערך 5. לכן ערך הביטוי הוא "אמת".
(6)	True	הערך 3 קטן או שווה לערך 5. לכן ערך הביטוי הוא "אמת".

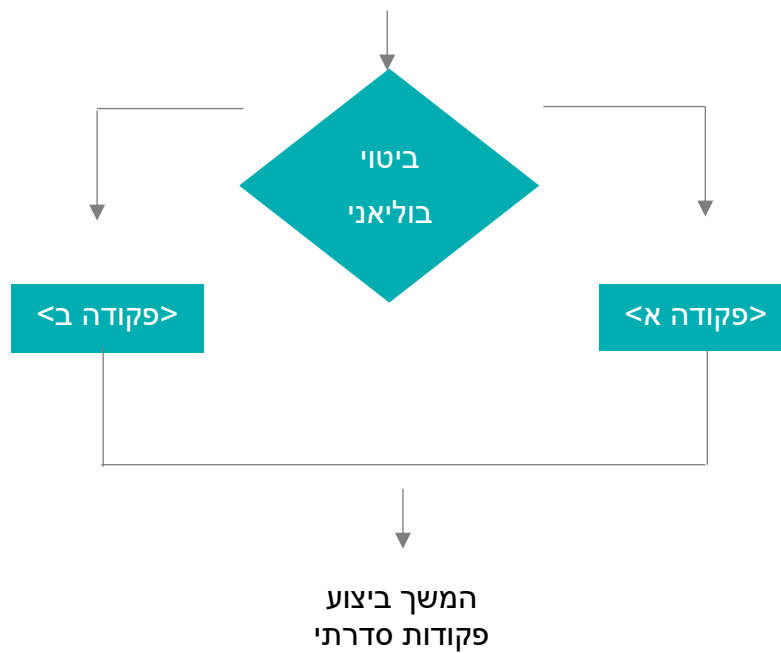
ביצוע מותנה if

כל משפט if ("האם?") בנוי בבסיסו ממשפט תנאי אשר נשאלת בו שאלה והתשובה המתקבלת היא **בוליאנית**.

אם ערך הביטוי הבוליאני הוא "אמת", תתבצענה סדרת הפקודות הבאות אחריו;

אחרת – כלומר, אם ערך המשפט הבוליאני הינו "שקר" – תתבצענה סדרת הפקודות הבאות אחרי הפקודה "else" ("אחרת").

לשם המחשה נשתמש בתרשימי זרימה:



איור 1 – תרשים ביצוע מותנה if

דוגמה משלבת

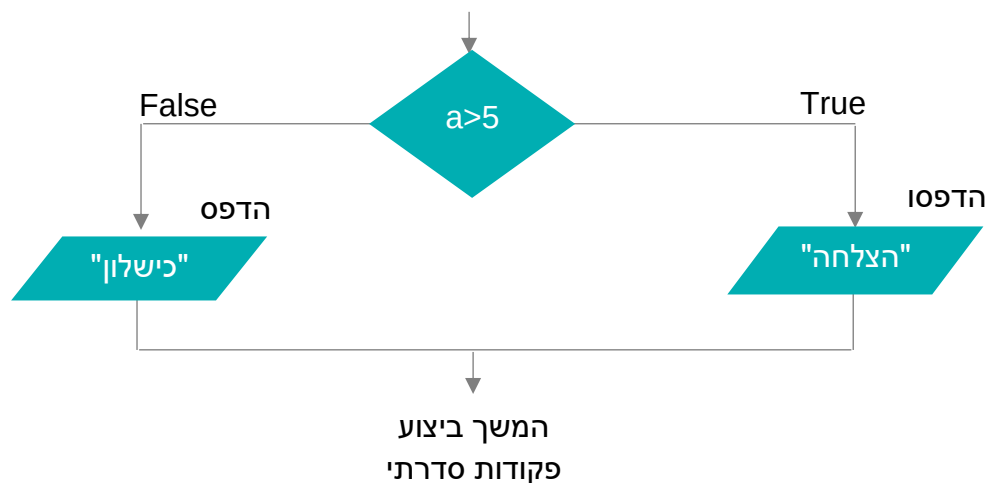
נתון האלגוריתם המילולי הבא:

אם ערכו של a גדול מ-5

הדפס את הטקסט "הצלחה"

אחרת, הדפס את הטקסט "כישלון".

נבנה כעת תרשים זרימה לאלגוריתם המילולי הנ"ל:



איור 2 - תרשים זרימה לאלגוריתם מילולי

לפני שנעבור לפקודות מחשב בשפת #C, נשים לב למספר הנחיות חשובות:

א. מקובל כי כל פקודת `else` צריכה להיות ממוקמת (מבחינת העמודה) מתחת לפקודת `if`.

בצורה זו נוח יותר לקרוא את הקוד.

ב. במקרה שיש לבצע יותר מפקודה בודדת – יש לפתוח סוגריים מסולסלים, ולסגור אותם בסיום סדרת הפקודות.

הערה: בדוגמה הבאה אין צורך לעשות זאת, מפני שמדובר בפקודה בודדת בין שהתנאי מתקיים ובין שאינו מתקיים.

ג. אחרי פתיחת סוגריים מסולסלים יש "לזוז" מספר רווחים קבוע או להשתמש ב-`Tab`. למעשה, Visual Studio עושה זאת עבורנו.

ד. כל פקודת `else` משויכת לפקודת `if` האחרונה שאינה בתוך סוגריים מסולסלים.

כעת נבנה – על בסיס תרשים זרימה של האלגוריתם שהוצג – קטע מתוכנית מחשב:

```

if (a>5)           // תנאי בוליאני
    Console.WriteLine("הצלחה"); // פקודה זו תתבצע אם ערך התנאי הבוליאני הוא אמת
else              // אחרת
    Console.WriteLine("כישלון"); // פקודה זו תתבצע אם ערך התנאי הבוליאני הוא שקר
  
```

השלם את החסר:

א.	עבור הערך $a=3$ יודפס	___?___
ב.	עבור הערך $a=5$ יודפס	___?___
ג.	עבור הערך $a=7$ יודפס	___?___
ד.	עבור הערך $a=2$ יודפס	___?___
ה.	עבור הערך $a=(-5)$ יודפס	___?___

פתרון:

א.	עבור הערך $a=3$ יודפס	"כישלון"
ב.	עבור הערך $a=5$ יודפס	"כישלון"
ג.	עבור הערך $a=7$ יודפס	"הצלחה"
ד.	עבור הערך $a=2$ יודפס	"כישלון"
ה.	עבור הערך $a=(-5)$ יודפס	"כישלון"

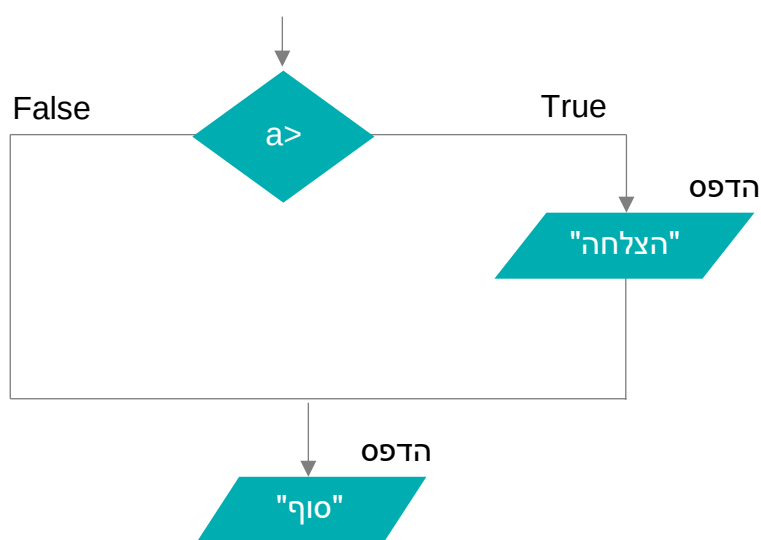
פקודת if פשוטה ללא פקודת else

תיתכן בדיקה בוליאנית באמצעות פקודת if, ללא הפקודה else.

דוגמה

פקודת if פשוטה ללא פקודת else:

תיתכן בדיקה בוליאנית באמצעות פקודת if, ללא הפקודה else. למשל:



איור 3 – פקודת if ללא פקודת else

תרגום לשפת c#

```
if (a>b)
    Console.WriteLine("הצלחה");
Console.WriteLine("סוף");
```

כעת נשים לב, באמצעות טבלאות מעקב, כי ההדפסה השנייה ("סוף") תתבצע בכל מקרה. כלומר, אין זה משנה מהו ערכו של הביטוי הבוליאני.

הערות	פלט	תנאי קיום בוליאני	b	a
דוגמת הרצה מס' 1				3
		$(3 > 5) \rightarrow \text{False}$	5	
	סוף			

הערות	פלט	תנאי קיום בוליאני	b	a
דוגמת הרצה מס' 2				3
		$(3 > 2) \rightarrow \text{True}$	2	
	הצלחה			
	סוף			

הערות	פלט	תנאי קיום בוליאני	b	a
דוגמת הרצה מס' 3				5
		$(5 > 5) \rightarrow \text{False}$	5	
	סוף			

פקודת if עם יותר מהוראת ביצוע בודדת

נסתכל על האלגוריתם המילולי הבא:

קלוט מספר ממשי המייצג טמפרטורת גוף של אדם מסוים.

אם הטמפרטורה גבוהה או שווה ל-36.5 מעלות, **אזי** בצע את הפעולות הבאות:

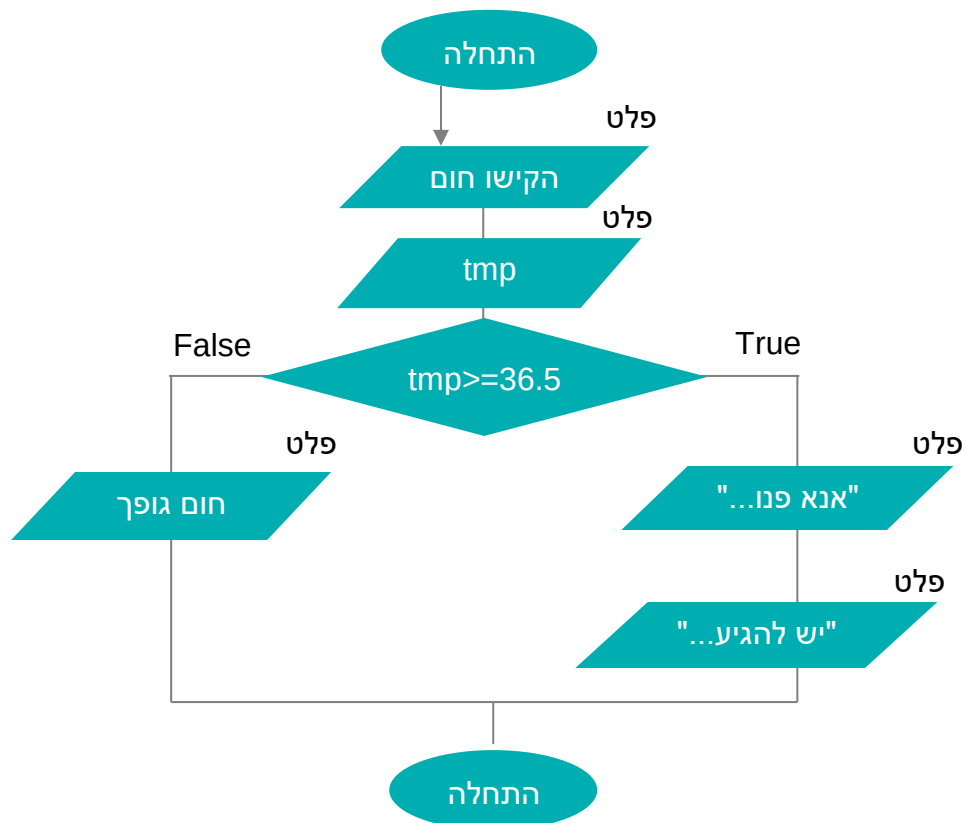
- א.** הדפס "אנא פנה לרפואת חירום".
- ב.** הדפס "יש להגיע עם כרטיס קופת חולים".

אחרת,

הדפס "חום גופך תקין".

נתרגם את האלגוריתם המילולי הנ"ל לתרשים זרימה:

נתרגם את האלגוריתם המילולי הנ"ל לתרשים זרימה:



איור 4 – ייצוג תרשים זרימה לאלגוריתם

תרגום לשפת C#

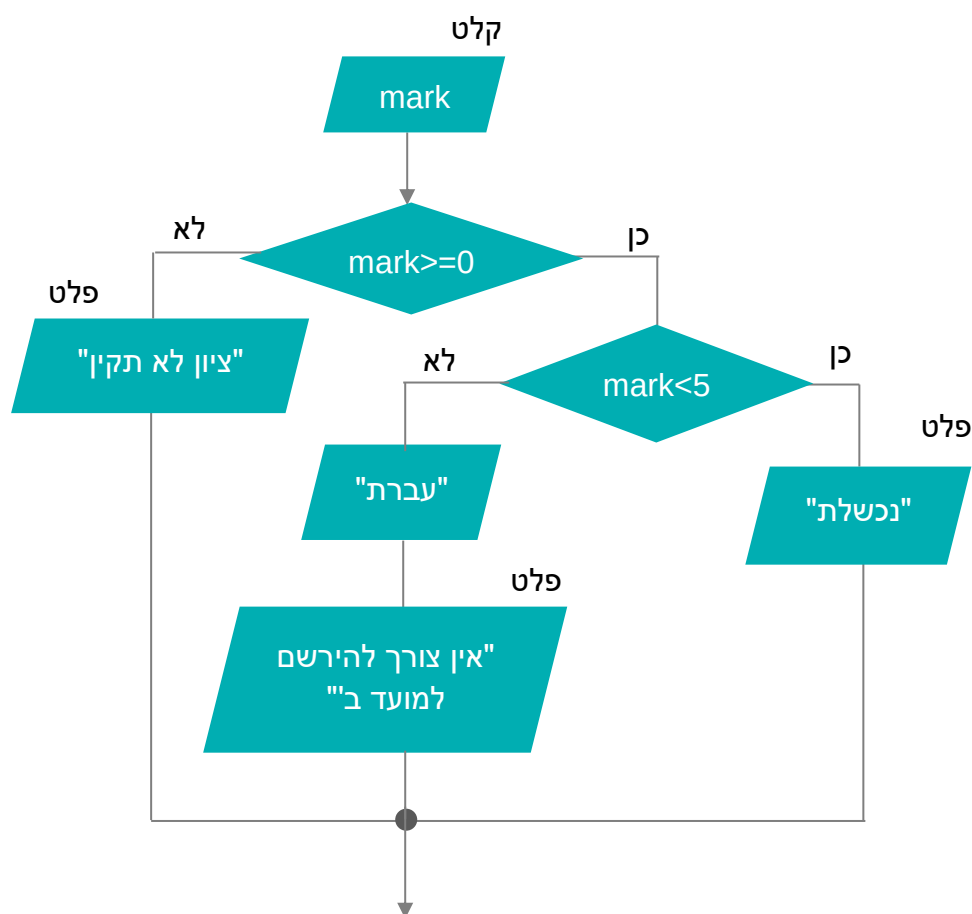
נשים לב כי במקרה שטמפרטורת הגוף גבוהה או שווה ל-36.5, תתבצעה **שתי פקודות**. כלומר, מדובר ביותר מפקודה בודדת, ולכן יש לפתוח סוגריים מסולסלים. הפעם נביא דוגמת קוד משולבת, הכוללת תוכנית מחשב מלאה:

```
static void Main(string[] args)
{
    double tmp;
    Console.WriteLine("Please enter body temperature:");
    tmp = double.Parse(Console.ReadLine());
    if (tmp >= 36.5)
    {
        Console.WriteLine("Please refer to emergency medicine.");
        Console.WriteLine("You must arrive with a health card");
    }
    else
        Console.WriteLine("your body temperature is normal");

    Console.ReadKey();
}
```

פקודת if מקוננת:

If מקונן הוא if שיכול להתבצע רק אם if בודד לפניו התבצע – כלומר, ערכו היה True. נסתכל על תרשים הזרימה הבא:



נסתכל על התנאי השני ($mark < 55$): אין אפשרות להגיע אליו – **אלא רק** לאחר שעברנו ב-`if` הראשון וערכו היה `True`.

אם ערכו היה `False` מלכתחילה – לעולם לא נגיע לתנאי השני (או לתנאי הפנימי או המקונן).

כעת נתרגם זאת לשפת `C#`:

```
static void Main(string[] args)
{
    int mark;
    Console.WriteLine("Please enter your mark:");
    mark = int.Parse(Console.ReadLine());
    if (mark >= 0)
        if (mark < 55)
            Console.WriteLine("You failed!");
        else
        {
            Console.WriteLine("You passed!");
            Console.WriteLine("No need for moed-b");
        }
    else
        Console.WriteLine("Invalid mark!");

    Console.ReadKey();
}
```

כעת נתרגל באמצעות טבלאות מעקב את כל סוגי הפלט האפשריים:

הערות	פלט	תנאי קיום 2	תנאי קיום 1	mark
דוגמת הרצה מס' 1				
	Please enter your mark:			
				75
			$75 \geq 0$ (T)	
		$75 \leq 55$ (F)		
	You passed!			
	No need for moed-b			

הערות	פלט	תנאי קיום 2	תנאי קיום 1	mark
דוגמת הרצה מס' 2				
	Please enter your mark:			
				42
			$42 \geq 0$ (T)	

		42<=55 (T)		
			You failed!	

הערות	פלט	תנאי קיום 2	תנאי קיום 1	mark
דוגמת הרצה מס' 3				
	Please enter your mark:			
				-12
			-12>=0 (F)	
	Invalid mark!			

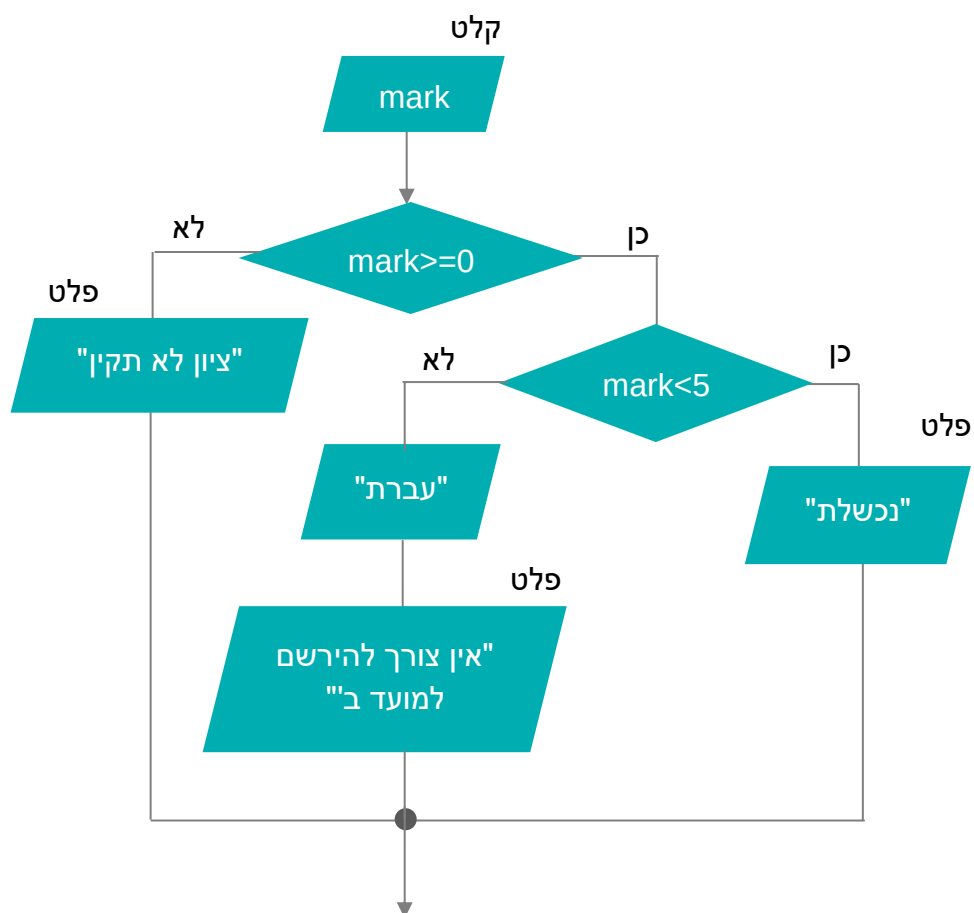
האם התוכנית הנ"ל תקינה? מה יהיה הפלט שלה עבור הציון הלא-חוקי 150?

תשובה: התוכנית אינה תקינה!

נדגים זאת באמצעות טבלת מעקב:

הערות	פלט	תנאי קיום 2	תנאי קיום 1	mark
דוגמת הרצה מס' 4				
	Please enter your mark:			
				150
			150>=0 (T)	
		150<55 (F)		
	You passed! No need for moed-b			

כלומר, עבור הקלט (הלא חוקי) 150 מתקבל פלט לא הגיוני – שהתלמיד עבר את המבחן ואינו זקוק למועד ב'. נתקן זאת:



כעת נתרגם זאת לשפת #C:

```
static void Main(string[] args)
{
    int mark;
    Console.WriteLine("Please enter your mark:");
    mark = int.Parse(Console.ReadLine());
    if (mark >= 0)
    {
        if (mark <= 100)
        {
            if (mark < 55)
                Console.WriteLine("You failed!");
            else
            {
                Console.WriteLine("You passed!");
                Console.WriteLine("No need for moed-b");
            }
        }
        else // Bigger than 100
        {
            Console.WriteLine("mark must be smaller than 101");
        }
    }
    else // negative
    {
        Console.WriteLine("mark cannot be negative number");
    }
    Console.ReadKey();
} // Main function
```

תרגילים לפעולות תנאי

הנחייה לתרגילים אלו: אפשר לרשום באמצעות פקודת if ואפשר לעשות זאת גם באמצעות הדפסה בלבד. לדוגמה, `.Console.WriteLine(a > 5);`

א. כתבו תוכנית מחשב שתקלוט שלושה ציונים. התוכנית תחשב ותדפיס את ממוצע הציונים של התלמיד. התוכנית תדפיס (בצורה בוליאנית) האם התלמיד "עבר" או "נכשל".

ב. כתבו תוכנית מחשב הקולטת את מספר הבנים והבנות בכיתה. התוכנית תדפיס (בצורה בוליאנית) האם מספר הבנים שווה למספר הבנות.

ג. כתבו תוכנית מחשב שתקלוט מספר שלם ותדפיס (בצורה בוליאנית) האם הוא בין 40 (כולל) ל-65 (כולל).

ד. כתבו תוכנית מחשב שתקלוט שני מספרים שלמים ותדפיס (בצורה בוליאנית) האם ההפרש ביניהם גדול מ-10.

לפניכם אלגוריתמים מילוליים. עבור כל אחד מהם:

א. תרגמו אותו לתרשים זרימה מסודר.

ב. תרגמו אותו לתוכנית מחשב מסודרת.

א.

1. קלוט מספר המייצג מספר תלמידים בכיתה.
2. קלוט מספר המייצג מספר כיתות בשכבה.
3. האלגוריתם יחשב וידפיס כמה תלמידים יש בשכבה.

ב.

1. קלוט תעריף לשעת עבודה (מספר ממשי).
2. מספר שעות שאדם עבד ביום רגיל (מספר שלם).
3. מספר ימי עבודה בחודש.
4. האלגוריתם יחשב וידפיס שכר חודשי (מספר ממשי) לפי ההיגיון הבא:
 - 4.1. עבור עד 8 שעות עבודה (כולל) – ישולמו 100%.
 - 4.2. עבור שעות 9-10 עבודה (כולל) – ישולמו 125%.
 - 4.3. עבור שעה 11 ואילך – ישולמו 150%.

ג.

1. קלוט שני ציוני תלמיד: את הציון הראשון למשתנה rishon, ואת הציון השני למשתנה sheni.
2. אם $rishon \geq 75$,
 - 2.1. קלוט למשתנה ממשי בשם third את הציון של התלמיד.
 - 2.2. הדפס את ממוצע הציונים הראשון והשלישי.
 - 2.3. אם ממוצע הציונים הראשון והשלישי גדול מהציון השני – הדפס "השיעור מתקיים בימי ב'",
אחרת, הדפס "השיעור מתקיים בימי ה'".
3. אחרת, הדפס את הציון השני.

ד.

בתיכון "מחויבים להצלחה" קיימים 4 בניינים. בכל בניין יש 3 קומות, ובכל קומה יש 5 כיתות. בכל כיתה אפשר לשבץ עד 40 תלמידים.

1. קלוט את מספר התלמידים שנרשמו ללימודים.
2. אם קיימים מספיק מקומות עבור התלמידים, אזי
2.1. הדפס את מספר המקומות הריקים הקיימים.
2.2. אחרת, הדפס "חריגה" ואת מספר ההרשמות החורגות מהכמות המרבית האפשרית.

ה.

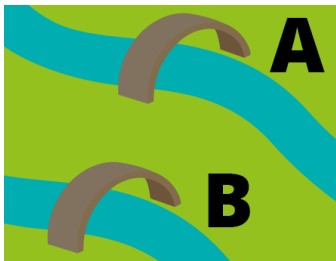
1. קלוט מספר שלם למשתנה בשם bdika1.
2. אם bdika1 מהווה מספר זוגי, אזי
2.1. הדפס את הערך של 100 פחות ערכו של bdika1.
2.2. הדפס את ערכו של bdika1 בריבוע. אחרת,
2.3. הדפס bdika1 (הערך התורן) הוא מספר זוגי.
3. הדפס "סוף התוכנית".

תנאים עם קשרים לוגיים מורכבים

קיים גשר מעל נחל שורץ תנינים.

אם הגשר מורם (כלומר, נמצא במצב "True")
אפשר להגיע לגדה השנייה של הנחל.

אחרת, הגשר מורד (כלומר, נמצא במצב "False")
ואי אפשר להגיע לגדה השנייה.



קשר לוגי "וגם" / AND / בשפת #C מסומן &&

טבלת אמת

המשמעות:

A	B	תוצאה
T	F	F
F	T	F
F	F	F
T	T	T

בביטוי המורכב מאוסף של "AND" ימים, מספיק שערך של ביטוי בודד יהיה "False" על מנת שערך כל הביטוי הבוליאני יהיה "False".

נמחיש עיקרון זה באמצעות הדוגמה הבאה:

```
int a = 3, b = 5, c = 4;
if((b - 2 == a) && (a > b) && (c > a))
    Console.WriteLine("Lego");
else
    Console.WriteLine("Damka");
```

ננתח את הביטוי:

נציב את ערכי המשתנים בביטוי הבוליאני הראשון: $(5-2 == 3)$.

קיבלנו שאלה בוליאנית $(3 == 3)$. האם הערך 3 שווה לערך 3?

התשובה היא 'כן', ולכן ערכו "True".

האם יש טעם להמשיך?! כן! נתקדם לביטוי הבוליאני הבא: $(3 > 5)$

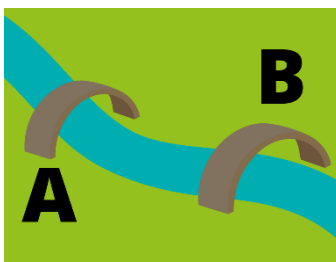
הערך 3 אינו גדול מהערך 5, ולכן ערך הביטוי הנ"ל הוא "False". האם יש טעם להמשיך? לא!

ברור כי ערך כל הביטוי הבוליאני המורכב הוא "False". לא ייתכן מצב שבו ערך הביטוי

הבוליאני המורכב יהיה "True" בהמשך.

לכן, הפלט של התוכנית הזו יהיה "Damka".

קשר לוגי "או" / OR / בשפת #C מסומן ||



טבלת אמת

המשמעות:

A	B	B A
T	F	T
F	T	T
T	T	T
F	F	F

בביטוי המורכב מאוסף של "OR" ימים", מספיק שערך בודד של ביטוי בודד יהיה "True" על מנת שערך כל הביטוי הבוליאני יהיה "True".

נמחיש עיקרון זה באמצעות הדוגמה הבאה:

```
int a = 5, b = 11, c = 9;
if ((a == b) || (b != a) || (b > c))
    Console.WriteLine("Shesh-Besh");
else
    Console.WriteLine("Remi-cube");
```

ננתח את הביטוי:

ערך הביטוי הראשון (5 == 11) הוא "False".

האם יש טעם להמשיך? כן! נתקדם לביטוי הבוליאני הבא ונציב את ערכי

המשתנים: (11 != 5)

ערכו "True".

האם יש טעם להמשיך? לא!

ברור שערך כל הביטוי הבוליאני המורכב הינו "True". הפלט יהיה **Shesh-Besh**.

קשר לוגי "לא" / NOT / מהפך / בשפת #C מסומן ב-! (סימן קריאה)

!(T)=F

!(F)=T

נסתכל על הדוגמה הבאה:

```
int a = 3, c = 5;
Console.WriteLine(!(a>c));
```

ננתח את הביטוי:

$a > c$ הוא ביטוי המחזיר ערך בוליאני "False" מפני שאין 3 אינו גדול מ-5.

כעת יש להפעיל על הערך המוחזר את קשר ה-NOT, ולכן:

$!(a > c) \rightarrow !(False) \rightarrow True$

היות וההיפך של שקר הוא אמת – זוהי התוצאה הסופית של הביטוי הבוליאני.

דגשים חשובים בעבודה עם ביטויים לוגיים מורכבים

א. קשר "לא" קודם או עדיף על האחרים.

ב. קשר AND קודם לקשר OR, בדיוק כמו שכפל קודם לחיבור.

ג. אם יש סוגריים בביטוי (דוגמאות בהמשך) – אז יש להן עדיפות.

דוגמאות לפקודת if מקוננת מורכבת עם קשרים לוגיים

נסתכל על האלגוריתם המילולי הבא:

קלוט מספר המהווה ציון (כלומר, אמור להיות בין הערכים 0 [כולל] ל-100 [כולל]).

אם הציון הוא בין 55 (כולל) ל-74 (כולל), אזי

א. הדפס "עברת".

ב. הדפס את ההפרש בין הציון 100 לציון של התלמיד.

אחרת,

אם הציון הוא בין 75 (כולל) ל-100 (כולל), אזי

א. הדפס "יפה מאוד".

ב. הדפס "המשך כך".

אחרת,

אם הציון קטן מ-0 או גדול מ-100, הדפס "ציון לא חוקי".

אחרת, הדפס "נכשלת!".

כעת נתרגם זאת לשפת C#:

```
static void Main(string[] args)
{
    int grade;
    Console.WriteLine("please enter your grade (0-100)");
    grade = int.Parse(Console.ReadLine());
    if ((grade >= 55) && (grade <= 74))
    {
        Console.WriteLine("You passed!");
        Console.WriteLine(100 - grade);
    }
    else
        if ((grade >= 75) && (grade <= 100))
        {
            Console.WriteLine("well done");
            Console.WriteLine("keep on doing the good work");
        }
}
```

דוגמה מסכמת

נציג שלוש דוגמאות לבדיקת "ציון חוקי". כאמור, ציון חוקי הוא ערך בין 0 (כולל) ל-100 (כולל).

פתרון לפי דרך א'

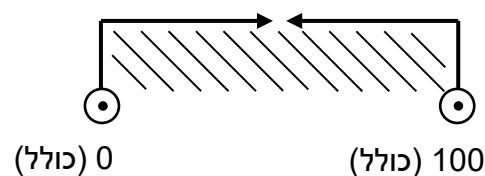
נסתכל על הניסוח הבא:

אם המספר הנקלט גדול או שווה ל-0 **וגם** קטן או שווה ל-100,

אזי הוא מהווה ציון חוקי

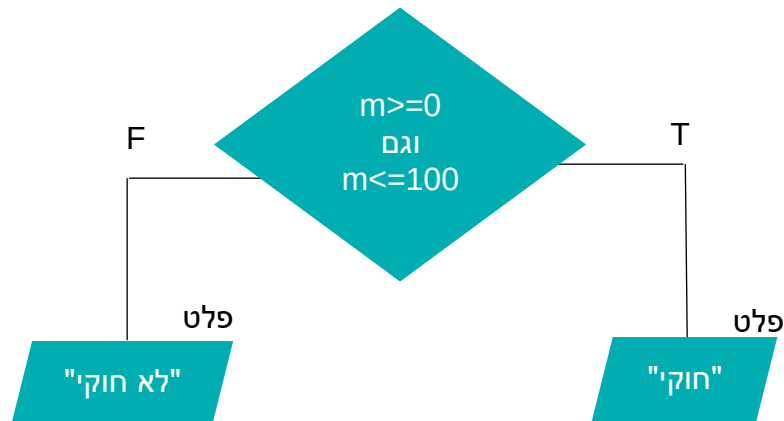
אחרת, מדובר במספר שאינו מהווה ציון חוקי.

בצורה גרפית, אפשר לתאר זאת כך



איור 6

ובתרשים זרימה התנאי הנ"ל ייראה כך:



איור 7

בשפת #C ייראה הקוד כך:

```

if ((grade >= 0) && (grade <= 100))
    Console.WriteLine("legal grade (OK)");
else
    Console.WriteLine("illegal grade (NOT OK)");
    
```

ננתח מספר דוגמאות הרצה:

א. נניח כי הקלט (למשתנה grade) הוא 86. האם מדובר בציון חוקי?

נציב ונבדוק:

$((86 \geq 0) \text{ AND } (86 \leq 100))$

הביטוי הראשון מתקיים היות ואכן 86 גדול או שווה ל-100, אז אפשר לפשט אותו:

$(T \text{ AND } (86 \leq 100))$

גם הביטוי השני מתקיים היות ו-86 קטן או שווה ל-100, ולכן קיבלנו:

$(T \text{ AND } T) \rightarrow T$

משמע כי כל הביטוי הבוליאני מחזיר את הערך "True", ולכן תתבצע הפקודה

הראשונה שאחרי משפט ה-if ויודפס "ציון חוקי" (legal grade).

ב. נניח כי הקלט (למשתנה grade) הוא 42. האם מדובר בציון חוקי?

נציב ונבדוק:

$((42 \geq 0) \text{ AND } (42 \leq 100))$

הביטוי הראשון מתקיים היות ואכן 42 גדול או שווה ל-100, אז אפשר לפשט אותו:

$(T \text{ AND } (42 \leq 100))$

גם הביטוי השני מתקיים היות ו-42 קטן או שווה ל-100, ולכן קיבלנו:

$(T \text{ AND } T) \rightarrow T$

משמע כי כל הביטוי הבוליאני מחזיר את הערך "True", ולכן תתבצע הפקודה

הראשונה שאחרי משפט ה-if ויודפס "ציון חוקי" (legal grade).

חשוב לא להתבלבל(!): אמנם הציון 42 הוא ציון נכשל – אך כציון הוא חוקי בהחלט.

ג. נניח כי הקלט (למשתנה grade) הוא -12. האם מדובר בציון חוקי?

נציב ונבדוק:

$((-12 > 0) \text{ AND } (-12 \leq 100))$

הביטוי הראשון מחזיר "False" מפני שהערך -12 אינו גדול או שווה ל-0. הביטוי

הפך להיות:

$(F \text{ AND } (-12 \leq 100))$

למעשה אפשר לעצור בנקודה זו, היות ולמדנו כי כשיש לנו ביטוי המורכב מ-AND'ים

בלבד – מספיק ערך בודד שהוא "False" על מנת שכל הביטוי הבוליאני המורכב יחזיר

"False".

למרות הנאמר לעיל, נמשיך את ההצבה למען הסר ספק.

הערך -12 אכן קטן או שווה ל-100, ולכן ערכו "True". הביטוי נראה כעת כך:

$(F \text{ AND } T) \rightarrow F$

למדנו, על פי טבלת האמת של קשר "וגם", כי במקרה כזה הערך המוחזר הוא

"False". למעשה, אפשר היה לעצור ולהחזיר את הערך "False" לאחר חישוב תת-

הביטוי הראשון.

פתרון לפי דרך ב'

נסתכל גם על הניסוח הבא:

אם המספר הנקלט קטן מ-0 או גדול מ-100,

אזי הוא מהווה ציון לא חוקי

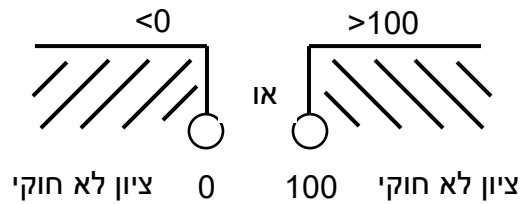
אחרת, מדובר במספר המהווה ציון חוקי.

שימו לב: כאשר המשפט הבוליאני יחזיר "True", המשמעות היא שהציון **לא חוקי**.

אין סתירה בין הדברים.

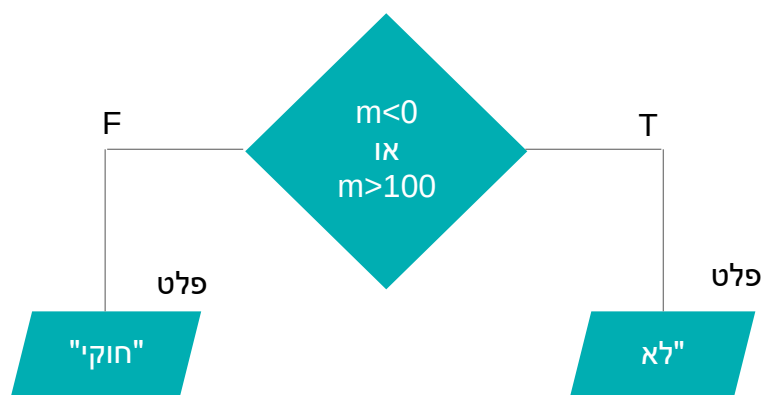
תמיד לאחר פקודת ה-if יתבצע החלק אם הביטוי מחזיר 'True'.

בצורה גרפית, אפשר לתאר זאת כך:



איור 8

ובתרשים זרימה התנאי הנ"ל ייראה כך:



איור 9

בשפת #C ייראה הקוד כך:

```
if ((grade < 0) || (grade > 100))
    Console.WriteLine("illegal grade (NOT OK)");
else
    Console.WriteLine("legal grade (OK)");
```

ננתח מספר דוגמאות הרצה:

א. נניח כי הקלט (למשתנה grade) הוא 77. האם מדובר בציון חוקי?

נציג ונבדוק:

$((77 < 0) \parallel (77 > 100))$

$\Downarrow \qquad \Downarrow$

$((F) \parallel (F))$

$\Downarrow$

הערך המוחזר מהביטוי הבוליאני המורכב הוא (F)

שימו לב(!): היות והערך הבוליאני הוא "False", תתבצע הפקודה שאחרי פקודת ה-else; ואכן הציון 77 הוא חוקי והפלט יהיה **legal grade**.

ב. נניח כי הקלט (למשתנה grade) הוא -20. האם מדובר בציון חוקי?

נציג ונבדוק:

$((-20 < 0) \parallel (20 - > 100))$

$\Downarrow$

$((T) \parallel \quad)$

ברור כי הערך שיחזור מהביטוי הבוליאני המורכב הוא "True" היות ויש לנו תוצאה של "True" בתת-הביטוי הראשון.

שימו לב(!): היות והערך הבוליאני הוא "True", תתבצע הפקודה שאחרי פקודת ה-if; ואכן הציון -20 הוא ציון לא חוקי והפלט יהיה **illegal grade**, כלומר, ציון **לא** חוקי.

ג. נניח כי הקלט (למשתנה grade) הוא 120. האם מדובר בציון חוקי?

נציג ונבדוק:

$((120 < 0) \parallel (120 > 100))$

$\Downarrow$

$((F) \parallel \quad)$

יש טעם להמשיך ולבדוק כי ייתכן שבהמשך המשפט נקבל ערך "True":

$((120 < 0) \parallel (120 > 100))$

$\Downarrow$

$((F) \parallel (T))$

$\Downarrow$

(T)

ושוב, היות והערך הבוליאני הוא "True", תתבצע הפקודה שאחרי פקודת ה-if. ואכן הציון 120 הוא ציון לא חוקי והפלט יהיה `illegal grade`, כלומר, ציון לא חוקי.

פתרון לפי דרך ג'

דרך חשיבה זו עלולה לבלבל מעט:

"אם הציון הוא לא (בין 0 [כולל] ל-100 [כולל]), אזי הציון לא חוקי

אחרת, הציון חוקי".

למעשה מדובר בהיפוך של דרך א'!

בואו נראה את הקוד ב-C#:

```
if (!(grade >= 0) && (grade <= 100))
    Console.WriteLine("illegal grade (NOT OK)");
else
    Console.WriteLine("legal grade (OK)");
```

ננתח מספר דוגמאות הרצה:

א. נניח כי הקלט (למשתנה grade) הוא 77. האם מדובר בציון חוקי?

```

(!((77>=0) && (77<=100)))
  ↓           ↓
(! (  (T)  && (T)  ))
  ↓
(! (    (T)    ))
  ↓
(!      (T)  ) → F

```

שימו לב(!): מאחר שהערך הבוליאני הוא "False", תתבצע הפקודה שאחרי פקודת

ה-else; ואכן הציון 77 הוא חוקי והפלט יהיה `legal grade`, כלומר, ציון חוקי.

ב. נניח כי הקלט (למשתנה grade) הוא 101. האם מדובר בציון חוקי?

```

(!((101>=0) && (101<=100)))
  ↓           ↓
(! (  (T)  && (F)  ))
  ↓
(! (    (F)    ))
  ↓
(!      (F)  )

```

(! (F)) → T

שימו לב(!): היות והערך הבוליאני הוא "True", תתבצע הפקודה שאחרי פקודת

ה-if; ואכן הציון 102 הוא חוקי והפלט יהיה illegal grade, כלומר, ציון לא חוקי.

ג. נניח כי הקלט (למשתנה grade) הוא -20. האם מדובר בציון חוקי?

(!((-20 >= 0) && (-20 <= 100)))

↓ ↓
(!((F) && (T)))

↓
(!((F)))

↓
(! (F)) → T

שימו לב(!): בגלל שהערך הבוליאני הוא "True" תתבצע הפקודה שאחרי פקודת ה-

if; ואכן הציון -20 הוא חוקי והפלט יהיה illegal grade, כלומר, ציון לא חוקי.

הגדרת if מקוצר

```
int number = 4;

int AbsNumber = number >= 0 ? number : -number;
```

פקודה זו שקולה ל-if הבא:

```
if (number >= 0)
    Absnumber = number;
else
    Absnumber = -number;
```

סימון ה-"?" (סימן שאלה) מהווה למעשה if, והאופרטור ":" (נקודתיים) מהווה else. מצד שמאל של הנקודתיים נמצא הביטוי שאמור להתבצע אם התנאי מתקיים, ואילו מימין להם נמצא הביטוי שאמור להתבצע אם התנאי אינו מתקיים.

מבנה בקרה שני – בּוּרֶר (switch)

פקודה זו מאפשרת לנו לטפל במספר רב של מקרים מבלי להשתמש ב-if מספר רב של פעמים. נניח כי המשתמש מקליד יום בשבוע (מספר מ-1 עד 6, המייצג בהתאמה יום ראשון עד שישי).

הפלט של התוכנית הוא שעות הפתיחה של מעבדת מחשבים.

מקרה א' – אפשרויות בודדות

בכל יום המעבדה פתוחה בשעות שונות:

שעות פתיחה	מספר מייצג	יום בשבוע
11:00-16:00	1	ראשון
16:00-19:00 וכן 09:00-13:00	2	שני
16:00-22:00	3	שלישי
10:00-16:00	4	רביעי
13:30-19:30	5	חמישי
08:45-14:15	6	שישי

התוכנית

```

static void Main(string[] args)
{
    int day;
    Console.WriteLine("Please enter number between 1..6");
    day = int.Parse(Console.ReadLine());

    switch (day)
    {
        case 1: Console.WriteLine("11:00-16:00");
                break;

        case 2: Console.WriteLine("09:00-13:00");
                Console.WriteLine("16:00-19:00");
                break;

        case 3: Console.WriteLine("16:00-22:00");
                break;

        case 4: Console.WriteLine("10:00-16:00");
                break;

        case 5: Console.WriteLine("13:30-19:30");
                break;

        case 6: Console.WriteLine("08:45-14:15");
                break;

        default: Console.WriteLine("Invalid day");
                 break;
    }
}

```

הסברים

- בהתאם לערכו של משתנה ה-switch המתאים (בדוגמה הנ"ל – day), התוכנית תבחר את ה-case המתאים עד סיומו של ה-switch.
- אם הערך הנקלט (day) אינו מופיע באף אחד מה-case'ים (לדוגמה, קלט 7 המייצג את יום שבת), אזי יתבצע קטע הקוד ב-default.
- Default מייצג "ברירת מחדל" – כלומר, אפשרות שלא הופיעה ברשימה.
- כש-case מסוים מתבצע – כל הפקודות הבאות עתידות להתבצע עד הופעת הפקודה break.
- אין חשיבות לסדר ה-case'ים.

מקרה ב' – אפשרויות משותפות

בחלק מהימים שעות הפתיחה של המעבדה זהות:

שעות פתיחה	מספר מייצג	יום בשבוע
11:00-16:00	1	ראשון
09:00-13:00 וכן 16:00-19:00	2	שני
08:45-14:15	3	שלישי
09:00-13:00 וכן 16:00-19:00	4	רביעי
08:45-14:15	5	חמישי
09:00-14:00	6	שישי

שימו לב:

- יום א' (1) – אפשרות בודדת
- ימים ב' ו-ד' (2 או 4) – אפשרות משותפת (באדום)
- ימי ג' ו-ה' (3 ו-5) – אפשרות משותפת (בירוק)
- יום ו' (6) – אפשרות בודדת

```
static void Main(string[] args)
{
    int day;
    Console.WriteLine("Please enter number between 1..6");
    day = int.Parse(Console.ReadLine());

    switch (day)
    {
        case 1: Console.WriteLine("11:00-16:00");
                break;

        case 2:
        case 4: Console.WriteLine("09:00-13:00");
                Console.WriteLine("16:00-19:00");
                break;

        case 3:
        case 5: Console.WriteLine("08:45-14:15");
                break;

        case 6: Console.WriteLine("09:00-14:00");
                break;

        default: Console.WriteLine("Invalid day");
                 break;
    }
}
```

הנחיות

עליכם לבנות לכל שאלה תרשים זרימה, לכתוב קוד ב-#C, וכן לבדוק את עבודתכם באמצעות טבלאות מעקב.

תרגול נוסף לפרק זה

1. בנו תוכנית מחשב שתקלוט את הנתונים הבאים:

א. שם התלמיד (מחרוזת).

ב. שלושה ציונים של תלמיד (מספר שלם).

התוכנית תדפיס את הציון הסופי (מספר ממשי) על פי ההיגיון הבא:

- הציון הראשון מהווה כ-20% מהציון הכללי.
- הציון השני, מהווה כ-20% מהציון הכללי.
- הציון השלישי מהווה כ-60% מהציון הכללי.

הדפס את הציון הסופי, וציין האם הוא שווה לאחד הציונים האחרים.

פתרון:

בשלב הראשון נגדיר את המשתנים שאיתם נעבוד בתוכנית:

מטרה	שם המשתנה	סוג
שמירת שם התלמיד	shem	מחרוזת (string)
ציון ראשון	mark1	מספר שלם
ציון שני	mark2	מספר שלם
ציון שלישי	mark3	מספר שלם
ציון סופי	sofi	ממשי/שבר (double)

בשלב השני נבצע קליטה של הערכים בהתאם לשאלה:

פעולה/ות תורנית/יות	פקודות מחשב
קליטת שם התלמיד	Console.WriteLine("please enter your name:"); shem = Console.ReadLine();
קליטת שלושת הציונים	Console.WriteLine("please enter 3 marks"); mark1 = int.Parse(Console.ReadLine()); mark2 = int.Parse(Console.ReadLine()); mark3 = int.Parse(Console.ReadLine());

בשלב השלישי נחשב את הציון הסופי ונדפיס אותו:

```
sofi = 0.20 * mark1 + 0.20 * mark2 + 0.60 * mark3;
Console.WriteLine("Final mark is {0}",sofi);
```

בשלב הרביעי נבדוק האם הציון הסופי זהה לאחד הציונים:

```
if ((sofi==mark1) || (sofi == mark2) || (sofi == mark3))
    Console.WriteLine("sofi equals to one of the mark");
else
    Console.WriteLine("sofi is not equals to one of the mark");
```

וכעת, כמובטח, נציג את כל הקוד:

```
static void Main(string[] args)
{
    string shem;
    int mark1, mark2, mark3;
    double sofi;
    Console.WriteLine("please enter your name:");
    shem = Console.ReadLine();
    Console.WriteLine("please enter 3 marks");
    mark1 = int.Parse(Console.ReadLine());
    mark2 = int.Parse(Console.ReadLine());
    mark3 = int.Parse(Console.ReadLine());
    sofi = 0.20 * mark1 + 0.20 * mark2 + 0.60 * mark3;
    Console.WriteLine("Final mark is {0}",sofi);
    if ((sofi==mark1) || (sofi == mark2) || (sofi == mark3))
        Console.WriteLine("sofi equals to one of the mark");
    else
        Console.WriteLine("sofi is not equals to one of the mark");
    Console.ReadLine();
}
```

2. עליכם לתרגם את האלגוריתם המילולי הבא לתוכנית מחשב ותרשים זרימה:

קיימים שני ציונים של תלמיד – במתמטיקה ובאנגלית (מספרים ממשים): a ו-b.

אם הציון במתמטיקה (a) גדול או שווה ל-85, אזי

אם הציון באנגלית (b) גדול או שווה ל-88, אזי

הדפס "התקבלת ללימודים במכללה".

הדפס את ממוצע הציונים.

אחרת,

הדפס "עליך לשפר ציונים".

אחרת,

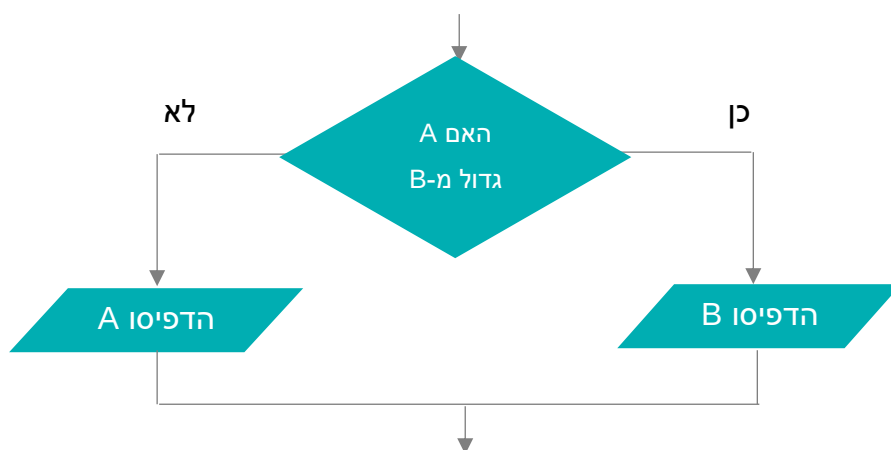
אם הציון באנגלית (b) גדול או שווה ל-90, אזי

הדפס "יש לך פטור מקורס אנגלית".

אחרת, הדפס "עליך להשלים קורס קיץ".

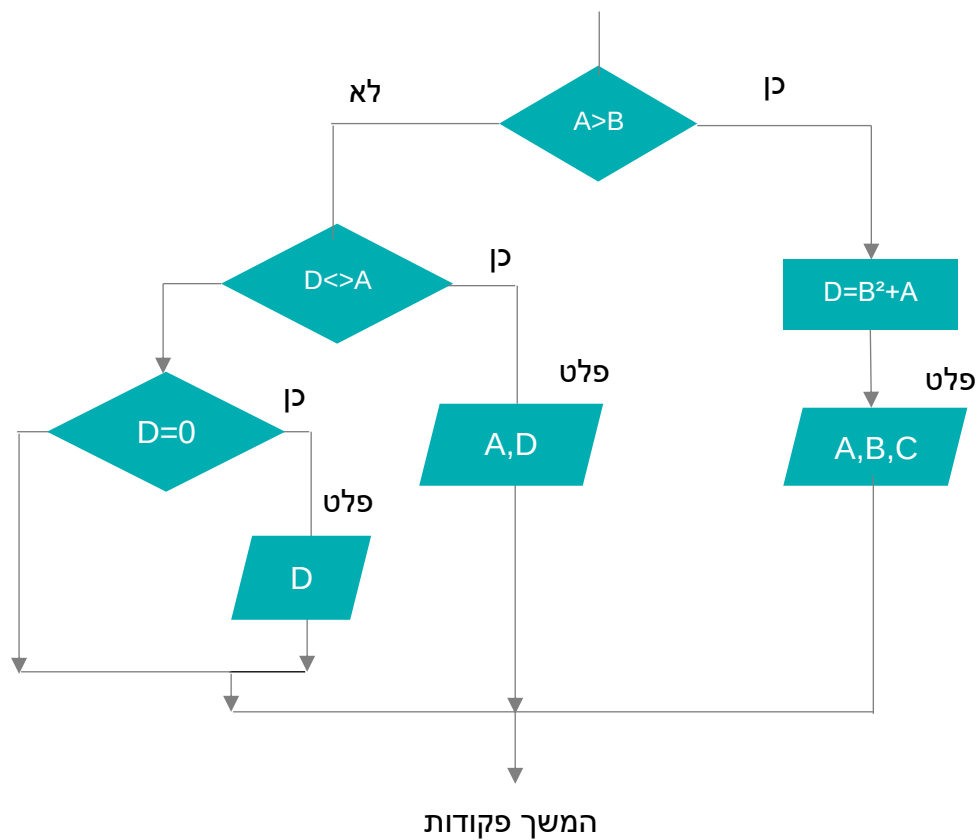
הנחיה: לפניכם תרשימי זרימה – עליכם לתרגם אותם לתוכנית מחשב ולבדוק את תקינותם באמצעות טבלאות מעקב.

3.



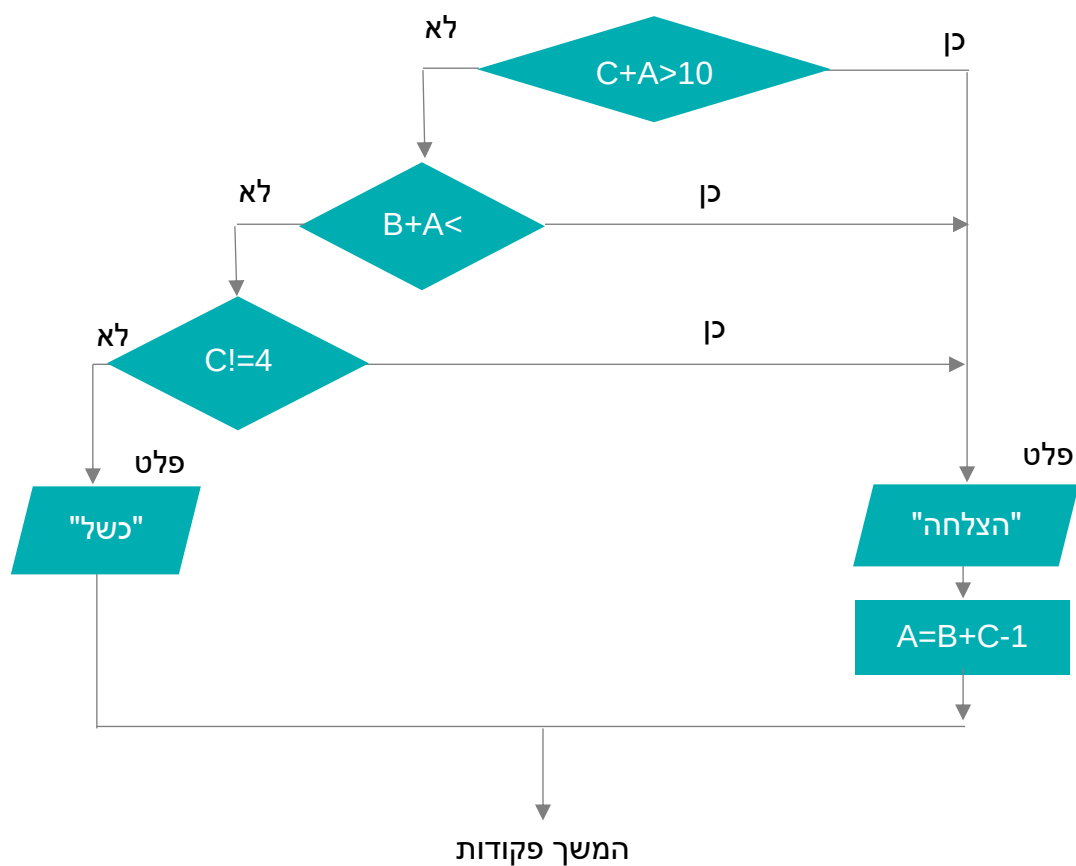
איור 10

4.



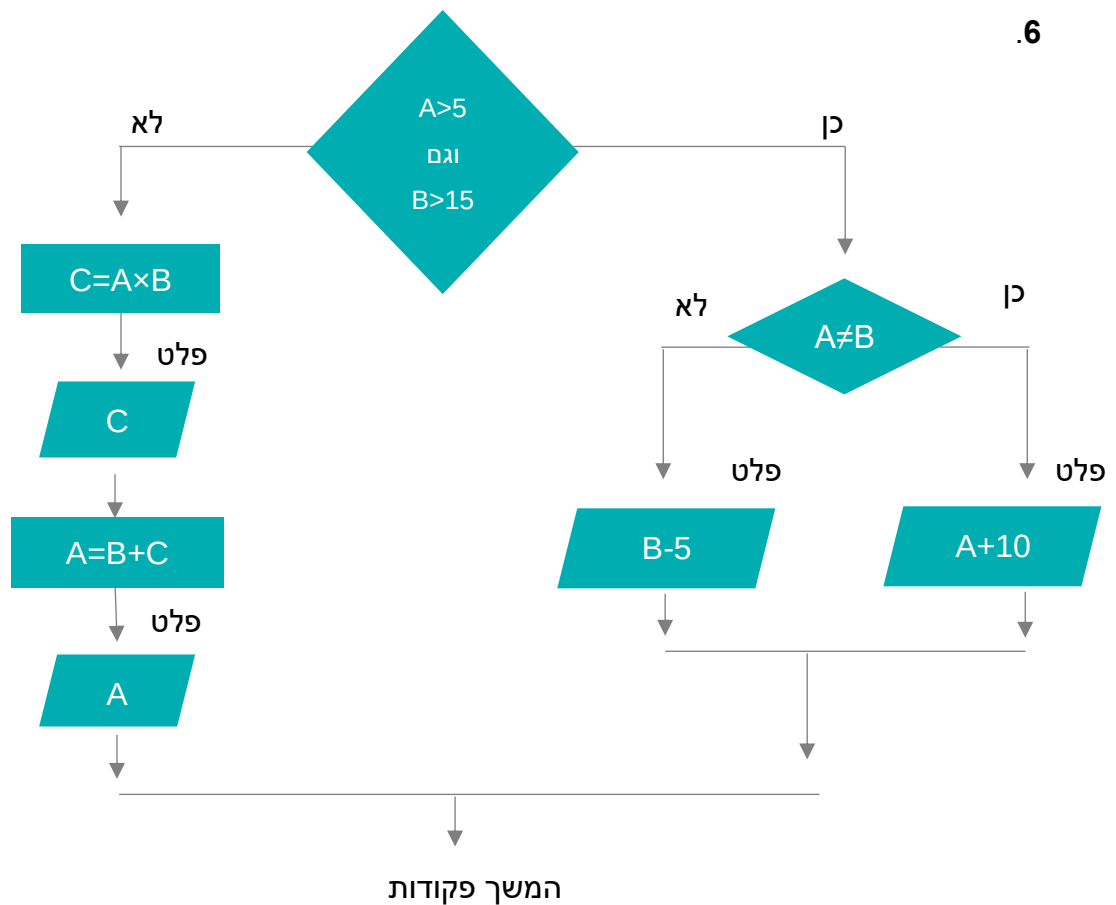
איור 11

5.



איור 12

6.



איור 13

פרק 4 – ביצוע חוזר

לולאה היא קטע קוד (כלומר, 0 או יותר שורות קוד) המתבצע מספר פעמים מסוים.

נשתמש בלולאה במקרים שונים, כגון:

- א. נתונים 50 מוצרים בחנות, ואנו מבקשים להדפיס את המוצר היקר ביותר.
- ב. נתונים 10 תלמידים, ואנו מבקשים לחשב את ממוצע הציונים שלהם.
- ג. קלוט מספרים שונים עד אשר הסכום שלהם יהיה 100 או יותר.

לולאה מספר 1 – לולאת for

לולאת for היא לולאה המתאימה בדרך כלל למקרים שבהם מספר הצעדים ידוע מראש.

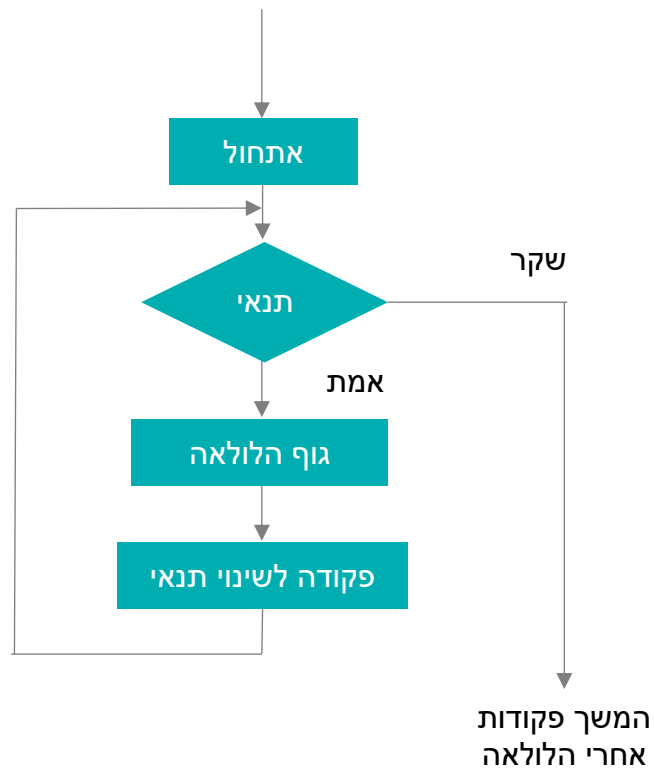
דוגמאות קלאסיות:

- א. חישוב ממוצע הציונים של 100 תלמידי השכבה.
- ב. מציאת המחיר הגבוה ביותר מבין 500 מוצרים בחנות.
- ג. חבר את כל המספרים הזוגיים שבין 200 ל-400.

מבנה הלולאה:

```
for (פקודה לשינוי תנאי ; תנאי קיום ; פקודת אתחול)
{
    גוף_הלולאה
}
```

תרשים זרימה



סדר הביצוע:

- ראשית כול תתבצע פקודת האתחול **פעם אחת ויחידה**.
- לאחר מכן ייבדק תנאי הקיום הבוליאני (ת.ק.).
- אם ערכו "אמת" – תתבצענה כל הפקודות בגוף הלולאה.
- לאחר ביצוע גוף הלולאה תתבצע **פקודה לשינוי תנאי**, ואז ייבדק שוב תנאי הקיום.
- אם ערכו "אמת" – ניכנס לגוף הלולאה, וחוזר חלילה.
- בסופו של דבר יחזיר תנאי הקיום הבוליאני ערך "שקר", ואז תתבצענה הפקודות שאחרי גוף הלולאה.

לולאת for – דוגמה מספר 1

```
int i;
for (i = 1; i < 3; i++)
{
    Console.WriteLine(" i={0}", i);
}
Console.WriteLine("sof");
```

שלבי הביצוע

כאמור, ראשית כול תתבצע שורת האתחול. כלומר, ערכו של i יהיה 1.

לאחר מכן ייבדק תנאי הקיום: ערכו של $i=1$, ולכן $1 < 3$ יחזיר ערך "אמת".

כעת נבצע את גוף הלולאה, והפלט שיודפס הוא:

$i=1$

לאחר מכן תבוצע פקודה לשינוי תנאי, וערכו של i יהיה 2.

שוב ייבדק תנאי הקיום "האם $2 < 3$?"

גם הפעם התשובה היא "אמת", ולכן ניכנס שוב לגוף הלולאה – והפלט יהיה:

$i=2$

שוב תתבצע פקודה לשינוי תנאי, והפעם ערכו של i יהיה 3.

הפעם, ערך הביטוי הבוליאני (האם $3 < 3$?) הוא "שקר".

כעת תתבצענה הפקודות שאחרי גוף הלולאה (במקרה זה – פקודה בודדת), ויודפס "sof".

הערה: גוף הלולאה במקרה הנ"ל מורכב מפקודה בודדת. לכן אפשר היה לוותר על הסוגריים המסולסלים.

קל יותר לעקוב אחר ביצוע של לולאת for ולהבינו באמצעות שימוש בטבלת מעקב:

הערות	פלט	תנאי קיום	i
רק ברישום ערך "תנאי קיום" אפשר לא לרדת שורה.		1<3 (true)	1
	i=1		
מתבצעת פקודה לשינוי תנאי.			2
		2<3 (true)	
	i=2		
מתבצעת פקודה לשינוי תנאי.			3
		3<3 (false)	
מדלגים מעל גוף הלולאה ומגיעים לפקודה שאחרי גוף הלולאה.	sof		

לולאת for – דוגמה מספר 2

מה מבצעת התוכנית הבאה?

```
static void Main(string[] args)
{
    int i;
    int result = 1;
    for (i = 1; i <= 3; i++)
        result *= 2; // ⇔ result = result * 2;
    Console.WriteLine("result = {0}.", result);
}
```

נעקוב באמצעות טבלת מעקב:

פלט	תנאי קיום (ת.ק.)	result	i
		1	
	1<=3 (true)		1
		1*2=2	
	2<=3 (true)		2
		2*2=4	
	3<=3 (true)		3
		4*2=8	
	4<=3 (false)		4
result = 8.			

לולאת for – דוגמה מספר 3

לולאת for יורדת:

<pre>static void Main(string[] args) { int i; int result = 0; for (i = 10; i >= 0; i--) { if (i % 2 == 0) { result += i; Console.WriteLine("i is : " + i); } } Console.WriteLine("result = {0}.", result); Console.ReadLine(); }</pre>	הסבר: משתנה i הוא מונה לולאה. הוא מאותחל בערך 10 וערכו יורד (בצעדים של אחד) עד הערך 0. משתנה result מאותחל בערך אפס. רק כשערכו של i מתחלק ב-2 ללא שארית (כלומר מספר זוגי) – אזי result צובר אותו. לסיכום: משתנה result מסכם את כל המספרים הזוגיים שבין 10 ל-0. נעקוב אחר סדר הביצוע באמצעות טבלת מעקב:
---	---

פלט	תנאי קיום ב' (if)	תנאי קיום (for)	result	i
			0	
		10 >= 0 → True		10
	10 % 2 == 0 → 0 == 0 → True			
			= 0 + 10 = 10	
i is 10.				
		9 >= 0 → True		9
	9 % 2 == 0 → 1 == 0 → False			
		8 >= 0 → True		8
	8 % 2 == 0 → 0 == 0 → True			
			= 10 + 8 = 18	
i is 8.				
		7 >= 0 → True		7
	7 % 2 == 0 → 1 == 0 → False			
		6 >= 0 → True		6
	6 % 2 == 0 → 0 == 0 → True			
			= 18 + 6 = 24	

5		5>=0 → True		i is 6.
			5%2==0 → 1 == 0 → False	
4		4>=0 → True		
			4%2==0 → 0==0 → True	
	=24+4=28			
				i is 4.
3		3>=0 → True		
			3%2==0 → 1 == 0 → False	
2		2>=0 → True		
			2%2==0 → 0==0 → True	
	=28+2=30			
				i is 2.
1		1>=0 → True		
			1%2==0 → 1 == 0 → False	
0		0>=0 → True		
			0%2==0 → 0==0 → True	
				i is 0.
-1				
		-1>=0 → False		
				result=30.

תכונות מיוחדות של לולאת for

א. אתחול ו/או פקודה לשינוי תנאי סימולטני

מותר לבצע מספר פקודות אתחול ו/או פעולות פקודה לשינוי תנאי.

```
static void Main(string[] args)
{
    int a,b;
    for ( a = 0, b = 4; a <= 3 ; a++, b--)
    {
        Console.WriteLine("a={0} b={1}",a,b);
    }
    Console.ReadKey();
}
```


- מותר לבצע מספר פקודות עדכון.
- פקודת עדכון יכולה להיות כל פקודה המשנה את ערך משתנה התנאי.

ננתח את הדוגמה הנ"ל:

- במקרה זה **האתחול** כפול. אין כאן סתירה: החלק הראשון של לולאת for הוא אתחול והוא מתבצע פעם אחת ויחידה.
- שלבי הביצוע זהים: תנאי קיום, ביצוע גוף לולאה, פקודה לשינוי תנאי וחוזר חלילה.
- גם **הפקודה לשינוי תנאי** כפולה במקרה זה. ושוב אין כאן סתירה – החלק השלישי של לולאת for הוא הפקודה לשינוי תנאי.

נעקוב באמצעות טבלת מעקב:

הסבר	פלט	תנאי קיום	b	a
ביצוע פקודת האתחול הראשונה				0
ביצוע פקודת האתחול השנייה			4	
כרגיל, בדיקת "תנאי קיום"		$0 \leq 3 \rightarrow \text{True}$		
ביצוע גוף הלולאה (פקודה בודדת)	$a=0 \ b=4$			
				1
			3	
		$1 \leq 3 \rightarrow \text{True}$		
	$a=1 \ b=3$			
				2
			2	
		$2 \leq 3 \rightarrow \text{True}$		
	$a=2 \ b=2$			3
			1	
		$3 \leq 3 \rightarrow \text{True}$		
	$a=3 \ b=1$			
				4
			0	
		$4 \leq 3 \rightarrow \text{False}$		

הפלט הסופי ייראה כך (קיימת ירידת שורה לאחר כל הדפסה):

```
a=0 b=4
a=1 b=3
a=2 b=2
a=3 b=1
```

ב. אתחול חיצוני

מותר לבצע אתחול מחוץ ללולאה. ננתח את הדוגמה הבאה:

```
static void Main(string[] args)
{
    int i = 2;
    for ( ; i <= 4; i++)
    {
        Console.WriteLine("i = {0}", i);
    }
    Console.ReadKey();
}
```

מונה הלולאה i מאתחל לפני גוף הלולאה. אפשר לאתחל אותו כך:

```
int i = 2;
```

ואפשר לאתחל אותו גם באמצעות קליטה:

```
Console.WriteLine("Please enter number");
i=int.Parse(Console.ReadLine());
```

שימו לב: אם ערך i יהיה 4 או יותר – לולאת ה-for כלל לא תתבצע!

דוגמאות ריצה באמצעות טבלאות מעקב:

דוגמה ראשונה:

i	תנאי קיום	פלט	הסבר
2			i מקבל ערך מסוים לפני לולאת ה-for
	$2 \leq 4 \rightarrow T$		בדיקת "תנאי קיום" כרגיל
		$i = 2$	ביצוע גוף הלולאה
3			פקודה לשינוי תנאי
	$3 \leq 4 \rightarrow T$		
		$i = 3$	
4			
	$4 \leq 4 \rightarrow T$		
		$i = 4$	
5			
	$5 \leq 4 \rightarrow F$		סיום לולאת ה-for ומעבר לפקודה שאחריה

דוגמה שנייה:

```
static void Main(string[] args)
{
    int i;
    Console.WriteLine("Please enter number");
    i=int.Parse(Console.ReadLine());
    for ( ; i <= 4; i++)
    {
        Console.WriteLine("i = {0}", i);
    }
}
```

הסבר	פלט	תנאי קיום	i
ביצוע פקודת הדפסה	Please enter number		
			4
		$4 \leq 4 \rightarrow T$	
	i = 4		
ביצוע פקודה לשינוי תנאי			5
		$5 \leq 4 \rightarrow F$	
סיום לולאת ה-for ומעבר לפקודה שאחריה			

דוגמה שלישית:

```
static void Main(string[] args)
{
    int i;
    Console.WriteLine("Please enter number");
    i=int.Parse(Console.ReadLine());
    for ( ; i <= 4; i++)
    {
        Console.WriteLine("i = {0}", i);
    }
}
```

הסבר	פלט	תנאי קיום	i
ביצוע פקודת הדפסה	Please enter number		
			6
		$6 \leq 4 \rightarrow F$	
סיום לולאת ה-for ומעבר לפקודה שאחריה			

בהרצה זו כלל לא היה פלט, ולולאת ה-for לא התבצעה!

ג. לולאה ללא גוף

גוף הלולאה מתבטל למעשה!

```
static void Main(string[] args)
{
    int i;
    Console.WriteLine("Start:");
    for (i = 1; i <= 3; i++);
        Console.WriteLine("i = {0}", i) ;
    Console.WriteLine("End:");
    Console.ReadKey();
}
```

הסבר:

- עצם הקיום של ; (נקודה ופסיק) לאחר שורת ה-for **מנטרל** את גוף לולאת ה-for.
- השורה המסומנת בצהוב תבוצע **פעם אחת בלבד** לאחר שתנאי הקיום יחזיר "שקר".
- מייד לאחר בדיקת תנאי הקיום תתבצע הפקודה לשינוי תנאי – כיון שלמעשה (בגלל ה-;) כלל אין כאן גוף לולאה!

נעקוב אחר סדר הביצוע באמצעות טבלת מעקב:

הסבר	פלט	תנאי קיום (ת.ק.)	i
	Start:		
בדיקת "תנאי קיום" כרגיל		$1 \leq 3 \rightarrow T$	1
אין גוף לולאה – מעבר לפקודה לשינוי תנאי!			
			2
		$2 \leq 3 \rightarrow T$	
			3
		$3 \leq 3 \rightarrow T$	
			4
תנאי הקיום החזיר "שקר".		$4 \leq 3 \rightarrow F$	
ביצוע הפקודות שאחרי ה-for:			
ביצוע הפקודה המסומנת בצהוב	i = 4		
	End:		

כאמור, הפלט בסופו של דבר יהיה:

Start:

i = 4

End:

ד. לולאה אינסופית:

הסבר	
• גוף הלולאה יתבצע "לנצח" • הפלט הראשון יהיה "Start:" • לאחר מכן יהיה הפלט i=1 וחוזר חלילה • לעולם לא נגיע לפלט האחרון, שהוא "End:"	<pre>static void Main(string[] args) { int i = 1; Console.WriteLine("Start:"); for (; ;) { Console.WriteLine("i = {0}", i); } Console.WriteLine("End:"); Console.ReadKey(); }</pre>

ה. תנאי קיום מורכב

נניח כי אנו משחקים במשחק מחשב "בול פגיעה". המחשב בוחר מספר אקראי בין 1 ל-100, והשחקן האנושי צריך לנחש מהו המספר. יש לו עד 5 ניחושים.

דוגמה:

נניח כי המחשב בחר במספר 8.

בסוף שתי הדוגמאות הבאות נציג קוד מקור מסודר שבו המחשב יבחר מספר בין 1 ל-100.

נעבור על שורות הקוד הבא:

חלק ראשון: הצהרת משתנים

bool game_over = false;	משתנה בוליאני – בודק האם המשתמש ניחש את המספר הסודי
int secret_number = 8;	המספר שהמחשב בחר
int uses_guess;	הניחוש של המשתמש
int i;	מונה לולאה (עד 5 ניחושים)

חלק שני: לולאת ה-for עם הבדיקה הכפולה

הלולאה תמשיך להתבצע כל עוד **לא היו 5 ניחושים** ו/או המשתמש לא הצליח **לנחש** את המספר:

for (i=1; (i<=5) && (!game_over); i++)	שילוב שני התנאים
{	
Console.WriteLine("Guess number = {0}", i);	הצג את מספר הניחוש
uses_guess = int.Parse(Console.ReadLine());	קלוט את הניחוש מהשחקן
// Nested If:	שימוש ב-if מקונן:
if (uses_guess > secret_number)	אם הניחוש גדול מהמספר שנבחר
Console.WriteLine("Too big !");	הצג "גדול מדי"
else if (uses_guess < secret_number)	אחרת, אם הניחוש קטן מדי
Console.WriteLine("Too small !");	הצג "קטן מדי"
else game_over = true;	אחרת,
}	אתחל את game_over ל"אמת"

חלק שלישי: בדיקה אם המשתנה הבוליאני שווה ל"אמת"

אם המשתנה שווה ל"אמת" – המשמעות היא שהשחקן ניחש את המספר הסודי ולכן יודפס "כל הכבוד", אחרת, יודפס "בהצלחה בפעם הבאה...":

```
if (game_over) // ⇔ if (game_over==true)
    Console.WriteLine("Well Done!");
else
    Console.WriteLine("Better luck next time...");
```

כעת נעקוב אחר התוכנית ונתאר **דרך ראשונה** לסיים אותה:

המשתמש הצליח לנחש את המספר הסודי בניחוש מספר 3. כלומר, הוא הצליח לנחש את המספר בפחות מ-5 ניחושים. לכן, המשחק עתיד להסתיים ויודפס "כל הכבוד!":

פלט	ת.ק.	i	user_guesss	secret_number	game_over
					false
				8	
	$1 \leq 5 \ \&\& \ \text{true} \Leftrightarrow T \ \&\& \ T \rightarrow T$	1			
ניחוש מספר 1					
			9		
גדול מדי!					
	$2 \leq 5 \ \&\& \ \text{true} \Leftrightarrow T \ \&\& \ T \rightarrow T$	2			
ניחוש מספר 2					
			7		
קטן מדי!					

			3	$3 \leq 5 \ \&\& \ \text{true} \Leftrightarrow T \ \&\& \ T \rightarrow T$	
					ניחוש מספר 3
		8			
true					
			4	$4 \leq 5 \ \&\& \ \text{false} \Leftrightarrow T \ \&\& \ F \rightarrow F$	
					כל הכבוד!

כעת נעקוב אחר התוכנית ונתאר **דרך שנייה** לסיים אותה:

המשתמש לא הצליח לנחש את המספר. אחרי 5 ניחושים יסתיים המשחק, ויודפס "בהצלחה בפעם הבאה...":

פלט	ת.ק.	i	user_guesss	secret_number	game_over
					false
				8	
	$1 \leq 5 \ \&\& \ \text{true} \Leftrightarrow T \ \&\& \ T \rightarrow T$	1			
ניחוש מספר 1					
			10		
גדול מדי!					
	$2 \leq 5 \ \&\& \ \text{true} \Leftrightarrow T \ \&\& \ T \rightarrow T$	2			
ניחוש מספר 2					
			6		

					קטן מדי!
			3	3<=5 && true ⇔ T && T → T	
					ניחוש מספר 3
		9			
					גדול מדי!
			4	4<=5 && true ⇔ T && T → T	
					ניחוש מספר 4
		7			
					קטן מדי!
			5	5<=5 && true ⇔ T && T → T	
					ניחוש מספר 5
		4			
			6	6<=5 && true ⇔ F && T → F	
					בהצלחה בפעם הבאה...

לולאה מספר 2 – לולאת while:

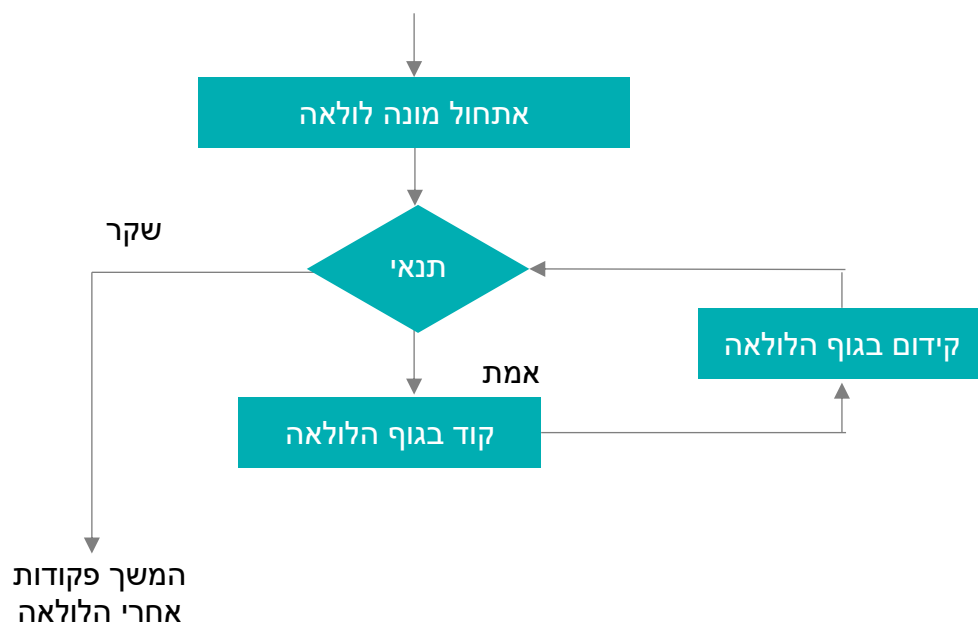
לולאה זו מכונה לולאה עם "בקרת כניסה". כלומר, לפני הכניסה לגוף הלולאה ייבדק תנאי הקיום (בדומה ללולאת for).

רק אם ערכו של תנאי הקיום הוא "אמת" – יתבצע גוף הלולאה.

עלול להיות מצב שבו גוף הלולאה כלל לא יתבצע.

נציג מגוון דוגמאות לשימוש בלולאה זו.

תרשים זרימה



סדר הביצוע

- ראשית כול יש לדאוג לאתחול לפני גוף הלולאה.
- לאחר מכן ייבדק תנאי הקיום הבוליאני (ת.ק.).
- **כל עוד** ערכו "אמת", תתבצעה כל הפקודות בגוף הלולאה.
- **חשוב מאוד:** על מנת למנוע לולאה אינסופית – יש לדאוג לפקודה לשינוי תנאי משתנה הלולאה בתוך גוף הלולאה.
- בסופו של דבר יחזיר תנאי הקיום הבוליאני ערך "שקר", ואז תתבצעה הפקודות שאחרי גוף הלולאה.

לולאת while – דוגמה מספר 1

```
static void Main(string[] args)
{
    int i = 1; // שורת האתחול
    while (i <= 3) // תנאי קיום
    {
        Console.WriteLine("{0}", i); // פקודה מספר 1 בגוף הלולאה
        i = i + 1; // פקודה מספר 2 בגוף הלולאה המהווה פקודה לשינוי תנאי
    }
    Console.WriteLine("Sof");
    Console.ReadLine();
}
```

הסבר

הפקודה הראשונה שתבצע היא הכנסת ערך 1 למשתנה i.

לאחר מכן ייבדק תנאי הקיום: האם $1 \leq 3$. התשובה היא "אמת", ולכן ניכנס לגוף הלולאה. תודפס ההדפסה הראשונה "1" (ערכו של i), משתנה i יקודם לערך 2, ונחזור שוב לבדיקת תנאי הקיום.

הפעם הבדיקה היא האם $2 \leq 3$. התשובה היא "אמת", ושוב ניכנס לגוף הלולאה.

תודפס ההדפסה השנייה "2" (ערכו של i), משתנה i יקודם לערך 3, ונחזור שוב לבדיקת תנאי הקיום.

הפעם, באופן דומה, הבדיקה היא האם $3 \leq 3$. התשובה היא "אמת", ושוב ניכנס לגוף הלולאה. תודפס ההדפסה השנייה "3" (ערכו של i), משתנה i יקודם לערך 4, ונחזור שוב לבדיקת תנאי הקיום.

הפעם ערכו של תנאי הקיום הוא "שקר", מפני שערכו של הביטוי הבוליאני $4 \leq 3$ הוא "שקר". כעת נדלג על גוף הלולאה, ותתבצענה הפקודות שאחרי הגוף – במקרה הנ"ל, פקודת הדפסה שתדפיס "sof".

נעקוב אחר סדר הביצוע באמצעות טבלת מעקב:

פלט	תנאי קיום (ת.ק.)	i
	$1 \leq 3$ (T)	1
1		
	$2 \leq 3$ (T)	2
2		
	$3 \leq 3$ (T)	3
3		
	$4 \leq 3$ (F)	4
sof		

לולאת while – דוגמה מספר 2

```
static void Main(string[] args)
{
    int result = 1, i = 1;
    while (i <= 3)
    {
        result *= 2;
        i=i+1;
    }

    Console.WriteLine("The result is {0}.", result);
    Console.ReadLine();
}
```

פלט	תנאי קיום (ת.ק.)	result	i
		1	
	$1 \leq 3$ (true)		1
		$1*2=2$	
	$2 \leq 3$ (true)		2
		$2*2=4$	
	$3 \leq 3$ (true)		3
		$4*2=8$	
	$4 \leq 3$ (false)		4
result = 8.			

לולאת while – דוגמה מספר 3

```
static void Main(string[] args)
{
    int help;
    Console.WriteLine("Enter an int value:");
    int num = int.Parse(Console.ReadLine());
    int copy = num, build = 0;
    while (copy > 0)
    {
        help = copy % 10;
        build = build * 10 + help;
        copy /= 10;
    }
    Console.WriteLine(num == build);
}
```

מה מבצעת התוכנית הבאה?

א. בנה טבלת מעקב עבור הערך הנקלט 123.

ב. בנה טבלת מעקב עבור הערך הנקלט 121.

נעקוב אחר מקרה א' באמצעות טבלת מעקב:

help	num	copy	build	ת.ק.	פלט
					הקש מספר שלם
	123				
		123			
			0		
				$123 > 0 \rightarrow T$	
3					
			$= 0 \times 10 + 3 = 3$		
		12		$12 > 0 \rightarrow T$	
2					
			$= 3 \times 10 + 2 = 32$		
		1		$1 > 0 \rightarrow T$	
1					

			$=32 \times 10 + 1 = 321$		
		0		$0 > 0 \rightarrow F$	
					false

הסבר:

בסוף הפלט מופיע "שקר" מפני שבשורה האחרונה נבדק התנאי הבוליאני הבא:

`Console.WriteLine(num == build);`

השאלה שנשאלה היא – האם 123 (ערכו של num) שווה לערכו של build (ערכו של 321).

נעקוב אחר מקרה ב' באמצעות טבלת מעקב:

פלט	ת.ק.	build	copy	num	help
הקש מספר שלם					
				121	
			121		
		0			
	$121 > 0 \rightarrow T$				
					1
		$=0 \times 10 + 1 = 1$			
	$12 > 0 \rightarrow T$		12		
					2
		$=1 \times 10 + 2 = 12$			
	$1 > 0 \rightarrow T$		1		
					1
		$=12 \times 10 + 1 = 121$			
	$0 > 0 \rightarrow F$		0		
true					

הסבר:

בסוף הפלט מופיע "אמת" מפני שבשורה האחרונה נבדק התנאי הבוליאני הבא:

```
Console.WriteLine(num == build);
```

השאלה שנשאלה היא – האם 121 (ערכו של num) שווה לערכו של build (ערכו של 121).

מה, למעשה, היה פה?

התוכנית בודקת האם מספר נקלט מהווה **פלינדרום**. **פלינדרום** (בעברית, "מילה מתהפכת") הוא מספר שקריאתו מימין לשמאל זהה לקריאתו בצורה הרגילה, כלומר, משמאל לימין.

לולאת while מתאימה במקרה זה מפני שאיננו יודעים מראש מהו אורך **המספר** שיקיש משתמש מסוים.

הסבר למשתני התוכנית:

ezer	מייצג את ספרת האחדות התורנית
num	מייצג את המספר המקורי הנקלט
copy	מייצג העתק של המספר המקורי הנקלט – בסופו של דבר ערכו אפס
bulid	מייצג את המספר החדש הנבנה במהלך לולאת while

לולאת while – דוגמה מספר 4

יש לקלוט מספר באורך לא ידוע. "נקצץ" את המספר מצד שמאל, ונספור בכל פעם את הספרות הזוגיות ואת הספרות האי-זוגיות. דוגמאות:

א. עבור המספר 1653, הפלט יהיה:

number of Even: 1

number of Odd : 3

אכן, מספר הספרות הזוגיות (Even) הוא 1, ומספר הספרות האי-זוגיות (Odd) הוא 3. כלומר, בביצוע התוכנית יהיה עלינו להשתמש בשני מונים. המונה הראשון ישמור את מספר הספרות הזוגיות (even), והמונה השני ישמור את מספר הספרות האי-זוגיות (Odd).

ב. עבור המספר 75289, הפלט יהיה:

number of Even: 2

number of Odd : 3

התוכנית במלואה:

<code>static void Main(string[] args)</code>	
<code>{</code>	
<code>int number;</code>	הצהרה על המספר המקורי הנקלט
<code>int copy;</code>	הצהרה על ההעתק של המספר המקורי
<code>int digit;</code>	הצהרה על ספרת האחדות
<code>int even = 0;</code>	איפוס מונה הספרות הזוגיות
<code>int odd = 0;</code>	איפוס מונה הספרות האי-זוגיות
<code>Console.WriteLine("Enter number:");</code>	הדפסה למסך
<code>number = int.Parse(Console.ReadLine());</code>	קליטת ערך למשתנה number
<code>copy = number;</code>	העתקת הערך למשתנה copy
<code>while (copy > 0)</code>	כל עוד המספר הקיים (גדול מאפס)
<code>{</code>	גוף הלולאה:
<code>digit = copy % 10;</code>	א. בידוד ספרת האחדות התורנית (מימין)
<code>if ((digit % 2) == 0)</code>	ב. בדיקה האם היא זוגית
<code>even++;</code>	אם כן, נוסף למונה even אחד
<code>else</code>	אחרת,
<code>odd++;</code>	נוסף למונה odd אחד
<code>copy /= 10;</code>	ג. חלוקת המספר ב-10 ("קיצוץ המספר")
<code>}</code>	סוף גוף הלולאה:
<code>Console.WriteLine("number of Even {0}.", even);</code>	הדפסת מונה הספרות הזוגיות
<code>Console.WriteLine("number of Odd {0}.", odd);</code>	הדפסת מונה הספרות האי-זוגיות
<code>Console.ReadLine();</code>	
<code>}</code>	

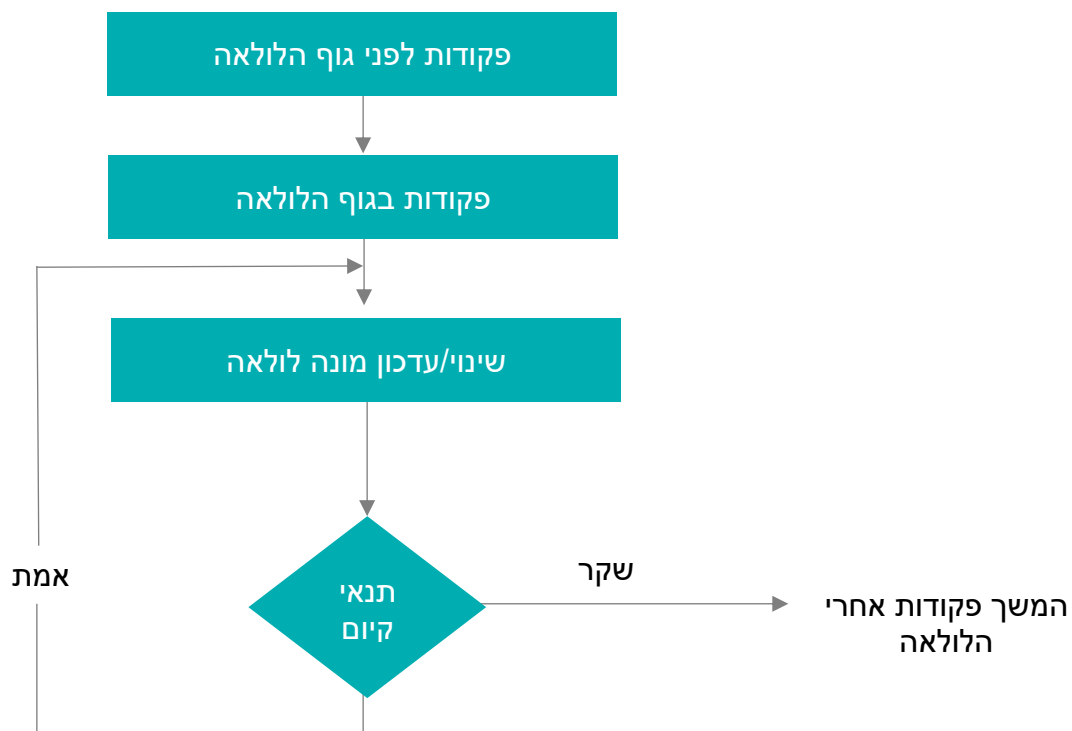
**הערה:**

המספר הנקלט המקורי – number – מקוצץ במהלך התוכנית. למרות שאיננו עושים בו שימוש בהמשך התוכנית, אנו ממליצים לשמור העתק שלו (במשתנה copy) לצורך השוואות עתידיות.

לולאה מספר 3 – לולאת do..while

לולאה זו מכונה לולאה עם "בקרת יציאה". כלומר, תנאי הקיום נבדק רק לאחר ביצוע גוף הלולאה. רק אם ערכו של תנאי הקיום הוא "אמת" – יחזור ויתבצע גוף הלולאה. גוף הלולאה יתבצע פעם אחת לפחות.

לולאה זו מכונה "לולאת קלט עקשנית", והיא מהווה מעין "מסננת" (filter) לקלטים. באופן יוצא דופן, מקובל לרשום את פקודת ה-while אחרי הסוגריים הסוגרים את הלולאה.

תרשים זרימה

סדר הביצוע:

- ראשית כול יש לדאוג לאתחול לפני גוף הלולאה.
- לאחר מכן תתבצעה כל הפקודות בגוף הלולאה.
- רק לאחר שתנאי הקיום הבוליאני ייבדק – וכל עוד הוא יחזיר ערך "אמת" – ישובו להתבצע הפקודות שבגוף הלולאה.



חשוב מאוד: על מנת למנוע לולאה אינסופית – יש לדאוג לפקודה לשינוי תנאי משתנה הלולאה בתוך גוף הלולאה.

לולאת do..while – דוגמה מספר 1

הסבר לפקודות

<code>static void Main(string[] args)</code>	
<code>{</code>	
<code>int i = 6;</code>	הצהרה ואתחול מונה לולאה
<code>do</code>	בתרגום חופשי "בצע!"
<code>{</code>	תחילת גוף הלולאה:
<code>Console.WriteLine("{0}", i);</code>	פקודה ראשונה בגוף הלולאה
<code>i=i-2;</code>	פקודה שנייה בגוף הלולאה (פקודה לשינוי תנאי)
<code>} while (i >= 0);</code>	כל עוד ⇔ מתאר באיזו מקרה להמשיך בלולאה
<code>}</code>	

נעקוב אחר סדר הביצוע באמצעות טבלת מעקב:

הסבר	פלט	תנאי קיום	i
אתחול מונה לולאה			6
כניסה מיידית לגוף הלולאה	6		
פקודה לשינוי תנאי בתוך גוף הלולאה			4
בדיקת ת.ק. בסוף הלולאה		$4 \geq 0 \rightarrow T$	
כניסה שנייה לגוף הלולאה	4		
פקודה לשינוי תנאי בתוך גוף הלולאה			2
בדיקת ת.ק. בסוף הלולאה		$2 \geq 0 \rightarrow T$	
כניסה שלישית לגוף הלולאה	2		
פקודה לשינוי תנאי בתוך גוף הלולאה			0
בדיקת ת.ק. בסוף הלולאה		$0 \geq 0 \rightarrow T$	
	0		
			-2
תנאי הקיום החזיר "שקר", ולכן הסתיימה הלולאה			$-2 \geq 0 \rightarrow F$

לולאת do..while – דוגמה מספר 2

<pre>static void Main(string[] args) { int distance; do { Console.WriteLine("Enter distance (1..30):"); distance = int.Parse(Console.ReadLine()); } while (distance <= 0 distance>30); Console.ReadLine(); }</pre>	כאן מדובר ב"מסננת קלט עקשנית": כל עוד המשתמש לא יקיש ערך בין 1 ל-30 – התוכנית לא תסתיים.
--	---

דוגמת הרצה:

```
Enter distance (1..30):
0
Enter distance (1..30):
31
Enter distance (1..30):
32
Enter distance (1..30):
30
```

כל עוד תנאי הקיום של הלולאה מחזיר ערך "אמת" – המשמעות היא שהוקלד ערך לא חוקי והלולאה תתבצע שוב.

התוכנית הסתיימה כי תנאי הקיום של הלולאה החזיר את הערך "שקר":

```
while (distance <= 0 || distance>30)
    ↓   ||   ↓
    30<=0 || 30>30
    ↓   ||   ↓
    F   ||   F → false
```

הערה

לולאת for שקולה ללולאת while מבחינת "בקרת כניסה". לעומת זאת, לולאת do..while שונה מהן בכך שגוף הלולאה מתבצע פעם אחת לפחות.

לולאות מקוננות

דוגמה מספר 1

מורה התבקש להקיש שלושה ציונים חוקיים (כלומר, בין 0 ל-100) ולחשב את הממוצע שלהם.
ננתח מספר מקרים:

מקרה א':

קלט ראשון נקלט 84 (חוקי – ציון ראשון)
קלט שני נקלט 91 (חוקי – ציון שני)
קלט שלישי נקלט 88 (חוקי – ציון שלישי)

התקבלו שלושה ציונים בסך הכול, וכל השלושה חוקיים. לכן אפשר לחשב ממוצע:

$$(84+91+88)=263/3=87.66$$

מקרה ב':

קלט ראשון נקלט 84 (חוקי - ציון ראשון)
קלט שני נקלט 112 (לא חוקי)
קלט שלישי נקלט 45 (חוקי – ציון שני)
קלט רביעי נקלט 30- (לא חוקי)
קלט חמישי נקלט 86 (חוקי – ציון שלישי)

התקבלו חמישה ציונים בסך הכול, וכעת יש שלושה ציונים חוקיים. לכן אפשר לחשב ממוצע:

$$(84+45+86)=215 / 3 = 71.66$$

מקרה ג':

קלט ראשון נקלט 74 (חוקי - ציון ראשון)
קלט שני נקלט 81 (חוקי – ציון שני)
קלט שלישי נקלט 111 (לא חוקי)
קלט רביעי נקלט 77 (חוקי – ציון שלישי)

התקבלו ארבעה ציונים בסך הכול, וכעת יש שלושה ציונים חוקיים. לכן אפשר לחשב ממוצע:

$$(74+81+77) = 232 / 3 = 77.33$$

כעת נתקדם עם שלבי הפתרון:

א. האם אנחנו יודעים לבצע מסננת קלט עקשנית כדי לקלוט ציון חוקי (בין 0 ל-100)?

התשובה היא – כן:

```
int grade;
do
{
    Console.WriteLine("Enter grade (0..100):");
    grade = int.Parse(Console.ReadLine());

} while (grade < 0 || grade > 100);
Console.ReadLine();
```

ב. כעת, את הקוד המוצג יהיה עלינו לבצע בתוך לולאה אחרת כל עוד לא התקבלו

שלושה ציונים חוקיים (בדומה למקרים א'-ג' שהוצגו לעיל).

ג. אם התקבל ציון לא חוקי, התוכנית תשוב ותקלוט ציון (שלב א') עד אשר יתקבל ציון חוקי.

ד. אם התקבל ציון חוקי – יודפס הציון בצירוף הטקסט "ציון חוקי".

נציג כעת את הקוד המלא בצירוף הסברים:

<code>static void Main(string[] args)</code>	
{	
<code>int maxGrades = 3; //</code>	הצהרה ואתחול של מספר הציונים החוקיים המבוקש
<code>int legal = 0; //</code>	זהו מונה הסופר את מספר הציונים החוקיים שהתקבלו
<code>int grade; //</code>	זהו הציון התורן (כל אחד בתורו) שנקלט
<code>double avg = 0; //</code>	כאן יחושב ממוצע הציונים החוקיים. בהתחלה הוא מאותחל לאפס.
שימוש ב צובר	
<code>while (legal < maxGrades) //</code>	זוהי הלולאה ה חיצונית . כל עוד לא התקבלו כל הציונים החוקיים
{	
<code>do //</code>	זוהי הלולאה ה פנימית . קבלת ציון חוקי
{	
<code>Console.WriteLine("Enter grade (0..100):");</code>	
<code>grade = int.Parse(Console.ReadLine());</code>	
<code>} while (grade < 0 grade > 100);</code>	
<code>Console.WriteLine("{0} legal grade",grade); //</code>	הדפסת הציון החוקי שהתקבל
<code>legal++; //</code>	שימוש ב מונה – עדכון מספר הציונים החוקיים שהתקבלו
<code>avg = avg + grade; //</code>	שימוש ב צובר – חיבור הציון החוקי האחרון לסכום הקיים עד כה
}	
<code>avg = avg /maxGrades; //</code>	חישוב ממוצע הציונים החוקיים
<code>Console.WriteLine("The average is {0}",avg); //</code>	הדפסת הממוצע
<code>Console.ReadLine();</code>	
}	

פרק 5 – ייצוג טיפוס נתונים חדש על ידי מחלקה

באופן דומה לטיפוסים שונים (כגון int, double ועוד) הקיימים בשפת #C, אפשר להגדיר טיפוס נתונים חדשים.

מחלקה היא "תבנית יוצרת", מעין "בית חרושת לאובייקטים".

אובייקטים הם "מוצרים" שהמחלקה מייצרת. מחלקה אחת יכולה, רעיונית, לייצר אינספור אובייקטים. למעשה, כל אובייקט נגזר ממחלקת האם שלו בשלב כזה או אחר.

אובייקט יכול להיות דבר פיזי-מוחשי כגון שולחן, סמארטפון או קיר, אך הוא יכול להיות גם דבר מופשט כגון קורס או טיסה.

מבנה המחלקה

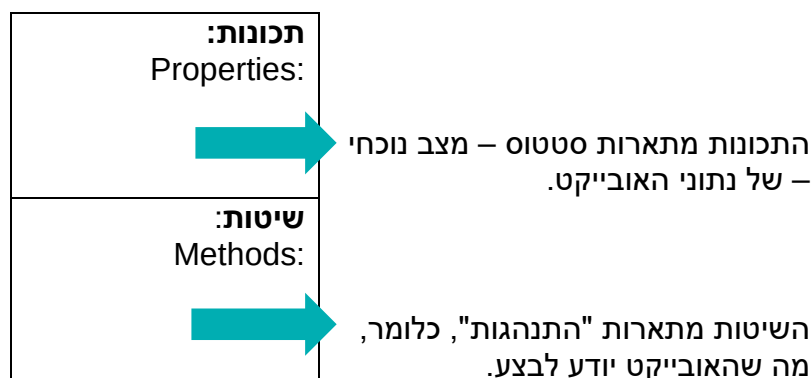
מחלקה בנויה משני חלקים עיקריים.

החלק הראשון הוא אוסף המאפיינים (Properties) או השדות של האובייקט.

החלק השני הוא אוסף השיטות (methods) או הפעולות/הפונקציות (functions) המשויכות לאובייקט.

חשוב להבין כי פעולות אלו מגדירות את אופי התנהגותו של האובייקט.

כל אובייקט מסוג מחלקה בנוי משני חלקים:



שני החלקים האלו מכונים "חברי מחלקה" (Class Members):



דוגמה מספר 1:

בואו נגדיר את מחלקת "נקודה" (Point).

נתחיל עם התכונות:

	לכל אובייקט מסוג נקודה יש שתי תכונות: ציר ה-x של הנקודה וציר ה-y של הנקודה. למשל, מהן הקואורדינטות של הנקודה האדומה? ערך ציר ה-x שלה הוא 8, וערך ציר ה-y שלה הוא 4.
	התכונות הנ"ל הן מידע פרטי. כלומר, אי אפשר לגשת לתכונות הללו מחוץ למחלקת Point. האפשרות היחידה לגשת אליהן היא באמצעות שיטות (פעולות) ציבוריות הניגשות לתכונות הפרטיות.
	נסקור את רכיבי המחלקה ונסביר בצורה מפורטת:

<pre>class Point { private int x; //(1) private int y; //(2) public int GetX() // (5) { return this.x; } public void SetX(int val) // (3) { x = val; } public int GetY() // (6) { return this.y; } }</pre>	בשורות (1) ו-(2) מוגדרות תכונות המחלקה. כאמור, התכונות הן פרטיות. כמוסכמה – לכל תכונה פרטית נכתוב פונקציה ציבורית שתוכל לקבוע ערך של תכונה פרטית או לשנות אותו. כאן, זו הפונקציה המסומנת ב-(3) המסוגלת לשנות את ערך ה-x של התכונה x, ובאופן דומה הפונקציה המסומנת ב-(4) המסוגלת לשנות את ערך ה-x של התכונה y. באופן דומה – לכל תכונה פרטית נכתוב פונקציה ציבורית המחזירה את ערך התכונה הפרטית. הפונקציה המסומנת ב-(5) מחזירה
---	---

<pre>public void SetY(int val) // (4) { this.y = val; } public Point() //(7) Default C'tor { this.x = 512; y = 320; } public Point(int i) : this(i, 300) //(8) { } public Point(int x, int y) // (9) { this.x = x; this.y = y; } public Point(Point Other) // (10) { x = Other.x; this.y = Other.GetY(); } public override bool Equals(Object obj) //(11) { Point PObj = obj as Point; //(א) if (PObj == null) // (ב) return false; if (this == PObj) // (ג) { Console.WriteLine("Same Point !"); return (false); } return (x.Equals(PObj.x) && y.Equals(PObj.y)); } public override string ToString() // (12) { return string.Format("x={0} y={1}\n", x, y); } }</pre>	את ערך ה-x, ובאופן דומה – הפונקציה המסומנת ב-(6) מחזירה את ערך ה-y. בשלב זה אנו מגדירים פונקציות פונקציות בונות. אלו הן: פונקציות המסוגלות לייצר אובייקט מסוג המחלקה: בנאי ראשון: זהו בנאי ללא פרמטרים, המכונה בנאי ברירת מחדל. ערכה של כל נקודה שהוא מייצר הוא: ציר ה-x שווה ל-512 וציר ה-y שווה ל-320. בנאי שני (8): בנאי (סוג של פונקציה) המקבל פרמטר אחד ומזמן את בנאי מספר 3 (9). בנאי שלישי (9): בנאי מפורש – כלומר, מקבל את שתי התכונות (x ו-y) ומייצר אובייקט. חשוב(!): שמות הפרמטרים זהים לשמות התכונות ולכן חובה להשתמש במילה השמורה (בצהוב) this. בנאי רביעי (10): בנאי המקבל כתובת של אובייקט (קיים) ומכונה בנאי העתקה – Copy Constructor. פונקציית השוואה (11): הפונקציה משווה שני אובייקטים מסוג נקודה ומחזירה ערך בוליאני. פונקציית הדפסה (12): הפונקציה מציגה את נתוני האובייקט בצורה טקסטואלית.																		
<pre>class Program { static void Main(string[] args) { Point p1; p1 = new Point(); Point p2 = new Point(512, 320); Console.WriteLine("p1 is " + p1); Console.WriteLine("p2 is " + p2); Console.WriteLine(p1.Equals(p2)); Console.ReadKey(); } }</pre> פלט התוכנית: <pre>p1 is x=512 y=320 p2 is x=512 y=320</pre>	הפונקציה הראשית: P1 הוא אובייקט שנוצר באמצעות בנאי ברירת המחדל, ולכן ערכו הוא: <table><tr><td></td><td>x</td><td>y</td></tr><tr><td>p1</td><td>512</td><td>320</td></tr><tr><td></td><td>440</td><td></td></tr></table> P2 אף הוא אובייקט מסוג Point – יש לו את אותם ערכי תכונות אך הוא ממוקם בכתובת אחרת בזיכרון: <table><tr><td></td><td>x</td><td>y</td></tr><tr><td>p2</td><td>512</td><td>320</td></tr><tr><td></td><td>612</td><td></td></tr></table> פקודה זו –		x	y	p1	512	320		440			x	y	p2	512	320		612	
	x	y																	
p1	512	320																	
	440																		
	x	y																	
p2	512	320																	
	612																		

True	Console.WriteLine(p1.Equals(p2)); תדפיס "True", מפני שמדובר בשני אובייקטים שהתכונות שלהם (ולא הכתובת!) זהות.
------	--

דוגמה מספר 2:

בואו נגדיר את מחלקת "רכב" – vehicle:

תכונות: Attributes: שם_יצרן string מהירות_מרבית int צבע string מהירות_נוכחית int
--

נתחיל עם התכונות:
כלומר, לכל אובייקט מסוג "רכב" יש:
א. שם יצרן (מוגדר כמחרוזת)
ב. מהירות מרבית (מוגדרת כמספר שלם)
ג. צבע (מוגדר כמחרוזת)
ד. מהירות נוכחית (מוגדרת כמספר שלם)
שימו לב כי תכונה מאופיינת כך שבסוף הגדרתה יש ; (נקודה
ופסיק).
כל התכונות הן פרטיות כלומר לא ניתן לגשת אליהן מחוץ
למחלקה.

הסבר: ההרשאה הדיפולטית (ברירת המחדל) של כל תכונה היא "פרטית".

כלומר, אפשר לרשום כך:

שם_יצרן string;

ולמעשה גם בצורה מפורשת – המשמעות זהה:

private string שם_יצרן;

נמשיך עם השיטות:

לכל אובייקט מסוג "רכב" יש התנהגויות מובנות שנגדיר לפי ראות עינינו או לפי שאלה נתונה
מסוימת.

שיטות:
Methods:
public string get_שם_יצרן()
public int get_מהירות_מרבית()
public int get_מהירות_נוכחית()
public void set_מהירות_נוכחית(
מהירות_חדשה)

כל אובייקט מסוג "רכב" מסוגל לבצע את
הפעולות הבאות:

א. אחזר שם יצרן – פונקציה
המחזירה ערך string (ללא set).
ב. אחזר מהירות מקסימלית –
פונקציה המחזירה ערך שלם
(ללא set).

- ג. אחזר מהירות_נוכחית() – פונקציה המחזירה ערך שלם.
- ד. קבע_מהירות_נוכחית(מהירות חדשה) – פונקציה set המקבלת מהירות חדשה.

שימו לב כי שיטה מאופיינת בכך שבסוף הגדרתה יש סוגריים.

חשוב להבין: על מנת שנוכל להפעיל את השיטות של המחלקה על אובייקט מסוגה – הן חייבות להיות ציבוריות (public).

מדוע יש צורך בתכונות פרטיות?

הרעיון הוא שלא כל אדם יוכל לשנות נתונים "רגישים" (כמו מספרי ת.ז. או חשבונות בנק של אנשים). רק אדם שיש לו הרשאה מיוחדת – כגון מנהל – יכול לעשות זאת. גם אז הוא יעשה זאת באמצעים מיוחדים כדוגמת סיסמה או כרטיס עובד או מפתח מיוחד (בדומה למפתח שיש ללוחמי האש במעליות מסוימות).

ניצור, באופן רעיוני, אובייקטים מטיפוס "מכונית":

אובייקט מספר 1

נגדיר את תכונותיו:

1. יצרן: Toyota
2. מהירות מרבית: 220 קמ"ש
3. צבע: כחול (BLUE)
4. מהירות נוכחית: 95 קמ"ש

אובייקט מספר 2

נגדיר את תכונותיו:

1. יצרן: Suzuki
2. מהירות מרבית: 187 קמ"ש
3. צבע: ירוק (GREEN)
4. מהירות נוכחית: 140 קמ"ש

כעת נעבור לכתיבת קוד בצורה מסודרת.

נתחיל עם התכונות:

```
class Vehicle // הגדרת מחלקת המכונית
{
    private string manufacturer; // שם יצרן
    private int maxSpeed; // מהירות מרבית
    private string color; // צבע רכב
    private int currSpeed; // מהירות נוכחית
```

נמשיך עם הפונקציות:

לכל אובייקט (או מופע) מסוג/מטיפוס "מכונית" יש התנהגויות מובנות:

פונקציה מספר 1:

בהקשר של תכונה מספר 1 – אי אפשר "לקבוע" יצרן. למעשה, אפשר רק לבנות פונקציה שתאחזר את שם היצרן בהתאם לתכנון המקורי שלה:

אחזר שם יצרן – פונקציה המחזירה ערך string (ללא set).

בואו נסתכל על הקוד שלה:

```
public string GetManufacturer()
{
    return (manufacturer);
}
```

פונקציה מספר 2:

בהקשר של תכונה מספר 2 – אי אפשר "לקבוע" מהירות מקסימלית. לכל רכב יש מהירות אופיינית משלו. גם במקרה זה אפשר רק לבנות פונקציה שתאחזר את המהירות המרבית של הרכב התורן.

אחזר מהירות מרבית – פונקציה המחזירה ערך שלם (ללא set):

```
public int GetMaxSpeed()
{
    return this.maxSpeed;
}
```

פונקציה מספר 3:

אחזר מהירות_נוכחית() – פונקציה המחזירה ערך שלם:

```
public int GetcurrSpeed()
{
    return currSpeed;
}
```

פונקציה מספר 4:

קבע_מהירות_נוכחית(מהירות חדשה) – פונקציה set המקבלת מהירות חדשה:

```
public void SetcurrSpeed(int s)
{
    currSpeed = s;
}
```

כעת נוסיף עוד פונקציות:

פונקציה מספר 1:

ייצוג טקסטואלי של נתוני האובייקט.

הסבר: הפונקציה מציגה בצורה מילולית את נתוני המכונית. לדוגמה, במקרה של הטיוטה הכחולה – נתוני המכונית יוצגו בצבע כחול. לאחר ההדפסה (בפונקציה הראשית) נדאג שצבע ההצגה יהיה רגיל (לבן):

```
public override string ToString()
{
    switch (color)
    {
        case "BLUE":
            Console.ForegroundColor = ConsoleColor.Blue;
            break;
        case "RED":
            Console.ForegroundColor = ConsoleColor.Red;
            break;
        case "GREEN":
            Console.ForegroundColor = ConsoleColor.Green;
            break;
        case "YELLOW":
            Console.ForegroundColor = ConsoleColor.Yellow;
            break;
        case "WHITE":
            Console.ForegroundColor = ConsoleColor.White;
            break;
    }
    return string.Format("{0} {1} {2} {3}\n", manufacturer, maxSpeed,
        color, currSpeed);
}
```

הערה: בפרק הבא, פרק מספר 6, נסקור ונסביר את כל סוגי הפונקציות.

פונקציה מספר 2:

זוהי פונקציה בונה/פונקציית בנאי (Constructor).

לפונקציה בונה יש שני מאפיינים בולטים:

- א. שמה יהיה תמיד כשמה של המחלקה.
- ב. אין לה ערך מוחזר, אפילו לא void. אין עוד פונקציה כזו!

```
public Vehicle(string m, int ms, string c, int cs)
{
    manufacturer = m;
    maxSpeed = ms;
    color = c;
    currSpeed = cs;
}
```

זהו בנאי מפורש – בנאי המקבל את כל (3) התכונות.

חשוב לשים לב: ההרשאה חייבת להיות ציבורית (public) על מנת שאפשר יהיה להפעיל את הפונקציה הבונה מבחוץ (בדרך כלל – מהפונקציה הראשית Main) למחלקה שלה, ולייצר אובייקטים.

קיימת אפשרות של העמסה/חפיפה – OverLoading – של בנאים:

המשמעות היא שיהיו מספר אפשרויות לייצר אובייקטים.

דוגמאות:

בנאי ברירת המחדל – Default Constructor:

שימו לב כי זהו הבנאי השני במחלקה זו.

זהו בנאי המקבל 0 נתונים. יהיה עלינו להחליט מהו "רכב" ברירת המחדל. בדוגמה זו הוחלט כי הבנאי הדיפולטי (ברירת המחדל) ייצר אובייקט לפי ההיגיון הבא:

הרכב יהיה מסוג Daihatsu, מהירותו המקסימלית היא 155 קמ"ש, צבעו אדום ומהירותו הנוכחית 0 קמ"ש. כעת נתרגם זאת לקוד:

```
public Vehicle()
{
    manufacturer = "Daihatsu";
    maxSpeed = 155;
    color="RED";
    currSpeed = 0;
}
```


כעת ניצור את בנאי מספר 3:

נגדיר בנאי חלקי (יצרן), המקבל שם יצרן בלבד, וקובע:

- מהירות מקסימלית 250 קמ"ש
- מהירות נוכחית 105 קמ"ש
- צבע צהוב

קוד:

```
public Vehicle(string yaztran)
{
    manufacturer = yaztran;
    maxSpeed = 250;
    color = "YELLOW";
    currSpeed = 105;
}
```

בנאי מספר 4 הוא בנאי מעתיק (Copy Constructor):

זהו בנאי שמקבל כתובת של אובייקט קיים (כלומר, אובייקט שנוצר בעבר) ויוצר אובייקט חדש שיש לו נתונים זהים לאובייקט הקיים.

קוד:

```
public Vehicle(Vehicle other)
{
    manufacturer = other.manufacturer;
    maxSpeed = other.maxSpeed;
    color = other.color;
    currSpeed = other.currSpeed;
}
```

הסבר – בשורה הבאה:

```
manufacturer = other.manufacturer;
```

אנו פונים לשדה (מאפיין) של הפרמטר המועבר לפונקציה הבונה.

דוגמאות:

מכונת א' (this) תהיה שווה למכונת ב' אם יתרחשו התנאים הבאים:

א. אם סכום המהירות המרבית והמהירות הנוכחית הוא 200 לפחות.

פתרון:

שורה מספר	
	<code>public override bool Equals(Object obj)</code>
	<code>{</code>
1	<code>Vehicle second = (Vehicle)obj; // (1) Casting / הסבה / המרה</code>
2	<code>if (this != second) // (2) Avoid same Vehicle!</code>
3	<code>{</code>
4	<code>//(3) ==> ראשית כל נחבר את המהירויות של הרכב הראשון</code>
5	<code>int first = this.maxSpeed + this.getcurrSpeed();</code>
6	<code>// 220 + 125 ==>345</code>
7	<code> //(4) ==> Calc first (second Vehicle)</code>
8	<code>int sum = second.maxSpeed + second.currSpeed;</code>
9	<code> //(5) → Calc second 120+80=200</code>
10	<code>return (first > 200 && sum > 200);</code>
11	<code>}</code>
12	<code>else</code>
13	<code>{</code>
14	<code>Console.WriteLine("It is the same Vehicle");</code>
15	<code>return (false);</code>
16	<code>}</code>
17	<code>}</code>

הסברים:

מספר שורה	הסבר
1	פונקציית ההשוואה מקבלת פרמטר בשם <code>obj</code> מסוג <code>Object</code> . כאן מתבצעת המרה כלפי מטה. נניח שיש לנו מנכ"ל חברה מסוים. אנחנו מבקשים את מספר הזהות שלו על מנת להדפיס עבורו תלוש משכורת. התכונה הזו (מספר ת.ז.) קיימת לכל עובד. לכן, יהיה עלינו "להסתכל" על המנכ"ל כעובד. זוהי ההמרה.
2	כאן אנו מבקשים להימנע ממצב שמשווה רכב מסוים לעצמו. אם זה אכן מתרחש – השליטה תעבור לשורה 14, תודפס הודעה ש"זה אותו הרכב" והפונקציה (בשורה 15) תחזיר "שקר".
5	נגדיר משתנה בשם <code>first</code> שיסכם את המהירות המרבית ואת המהירות הנוכחית של מי שהפעיל את פונקציית ההשוואה. למשל, עבור <code>v.Equals(v2)</code> ⇔ הרכב <code>v</code> הוא זה שעליו מתבצעת ההשוואה.
8	נגדיר משתנה בשם <code>second</code> שיסכם את המהירות המרבית ואת המהירות הנוכחית של מי שהועבר לפונקציית ההשוואה. למשל, עבור <code>v.Equals(v2)</code> ⇔ הרכב <code>v2</code> הוא זה שמועבר לפונקציית ההשוואה.
10	בשורה זו יוחזר ערך בוליאני (אמת או שקר) הבודק האם הרכבים "שווים". כאן נבדק התנאי הבא: <code>If (345>200) AND (200>200)</code> <code>T AND F</code> <code>F</code> זאת לפי ההיגיון הבא: <code>Vehicle v = new Vehicle("Toyota", 220, "BLUE", 125); //220+125→345</code> <code>Vehicle v2 = new Vehicle("Fiat", 120, "GREEN", 80); // 120+80→200</code> <code>Console.WriteLine(v.Equals(v2));</code>

בשלב זה נציג את כל קוד המקור של פרויקט זה:

מחלקה ראשונה בפרויקט:

```
using System;

namespace Dugma
{
    class Vehicle
    {
        private string manufacturer; // יצרן_שם
        private int maxSpeed; // מירבית_מהירות
        private string color; // צבע
        private int currSpeed; // נוכחית_מהירות

        public string getmanufacturer()
        {
            return (manufacturer);
        }
        public int getmaxSpeed()
        {
            return this.maxSpeed;
        }

        public void setColor(string newColor)
        {
            this.color = newColor;
        }

        public string getColor()
        {
            return this.color;
        }

        public int getcurrSpeed()
        {
            return currSpeed;
        }

        public void setcurrSpeed(int s)
        {
            currSpeed = s;
        }

        public override string ToString()
        {

```

המשך הקוד בעמוד הבא <<<

```

        switch (color)
        {
            case "BLUE":
                Console.ForegroundColor = ConsoleColor.Blue;
                break;
            case "RED":
                Console.ForegroundColor = ConsoleColor.Red;
                break;
            case "GREEN":
                Console.ForegroundColor = ConsoleColor.Green;
                break;
            case "YELLOW":
                Console.ForegroundColor = ConsoleColor.Yellow;
                break;
            case "WHITE":
                Console.ForegroundColor = ConsoleColor.White;
                break;
        }
        return string.Format
            ("{0} {1} {2} {3}\n", manufacturer, maxSpeed, color, currSpeed);
    }

    public Vehicle(string m, int ms, string c, int cs)
    {
        manufacturer = m;
        maxSpeed = ms;
        color = c;
        currSpeed = cs;
    }

    public override bool Equals(Object obj)
    {
        Vehicle second = (Vehicle)obj; // (1) Casting / הסבה / המרה
        if (this != second) // (2) Avoid same Vehicle!
        {
            //(3) ==> Calc first (this)
            int first = this.maxSpeed + this.getcurrSpeed();
            // 220 + 95 ==>315
            //(4) ==> Calc first (second Vehicle)
            int secnod = second.maxSpeed + second.currSpeed;
            //(5)
            return (first > 200 && secnod > 200);
        }
        else
        {
            Console.WriteLine("It is the same Vehicle");
            return (false);
        }
    }
}
}

```

מחלקה שניה בפרויקט:

```
using System;

namespace Dugma
{
    class Program
    {
        static void rc() // restore_(white)_color()
        {
            Console.ResetColor();
        }

        static void Main(string[] args)
        {
            Vehicle v = new Vehicle("Toyota", 220, "BLUE", 125);
            Vehicle v2 = new Vehicle("Fiat", 120, "GREEN", 80);
            Console.WriteLine(v);
            Console.WriteLine(v2);
            rc(); // מחזירה את צבע לבן כהדפסה רגילה

            Console.WriteLine(v.Equals(v2)); // הפעלת פונקציה השוואה

            Console.ReadKey();
        }
    }
}
```

פלט ההרצה:

```
Toyota 220 BLUE 125
Fiat 120 GREEN 80
False
```

משימות לדוגמה ופתרונות

משימה ראשונה:

מכונת א' (this) תהיה שווה למכונת ב' אם יתרחשו התנאים הבאים:

- א. שתיהן יוצרו על ידי אותו יצרן.
- ב. הפרש המהירות המקסימלית הוא עד 15 קמ"ש.

פתרון:

```
public override bool Equals(Object obj)
{
    Vehicle second = (Vehicle)obj; // (1) Casting / הסבה / המרה
    if (this != v) // (2) Avoid same Vehicle!
    {
        int mehirut = Math.Abs(maxSpeed - second.getmaxSpeed()); // (3)
        return (manufacturer.Equals(second.manufacturer) && (mehirut <= 15)); // (4)
    }
    else
    {
        Console.WriteLine("It is the same Vehicle");
        return (false);
    }
}
```

דוגמת הרצה:

```
static void Main(string[] args)
{
    Vehicle v = new Vehicle("Toyota", 220, "BLUE", 125);
    Vehicle v2 = new Vehicle("Toyota", 212, "GREEN", 80);
    Console.WriteLine(v);
    Console.WriteLine(v2);
    rc(); // חזרה לבצע לבן בהדפסות
    Console.WriteLine(v.Equals(v2));
    Console.ReadKey();
}
```

פלט ההרצה:

```
Toyota 220 BLUE 125
Toyota 212 GREEN 80
True
```

הסברים:

```
int mehirut = Math.Abs(maxSpeed - second.getmaxSpeed()); // (3)
```

בשורה זו אנו מגדירים משתנה מקומי בשם "מהירות". מה הוא מקבל?

```
Vehicle v = new Vehicle("Toyota", 220, "BLUE", 125);
Vehicle v2 = new Vehicle("Toyota", 212, "GREEN", 80);

Console.WriteLine(v.Equals(v2));

. לכן הנתונים שלו הם: האובייקט ("רכב") עליו עובדת פונקציית ההשוואה הוא
("Toyota", 220, "BLUE", 125);
int mehirut = Math.Abs(maxSpeed - second.getMaxSpeed());
```

ועכשיו נציב: 220 (מהירות מקסימלית של האובייקט הראשון) פחות 212 (מהירות מקסימלית של האובייקט השני):

```
int mehirut = Math.Abs(220 - 212) = |8| = 8 קמ"ש
```

הוא מקבל את ההפרש בין המהירויות בערך מוחלט. בואו נחשב:

```
return (manufacturer.Equals(second.manufacturer) && (mehirut <= 15)); // (4)
```

כעת אפשר להמשיך ולהציב:

manufacturer הוא שם היצרן של הרכב הראשון (מסומן בצהוב):

```
("Toyota", 220, "BLUE", 125);
```

second.manufacturer הוא שם היצרן של הרכב השני – בפונקציה הראשית שמו v2 ובפונקציית ההשוואה שמו second. גם כאן שם היצרן הוא Toyota.

עכשיו אפשר לחשב את הביטוי:

```
return (manufacturer.Equals(second.manufacturer) && (mehirut <= 15)); // (4)
```

```
return (true && (8<=15)); // קיבלנו "אמת" כי את שתי המכונות ייצר אותו יצרן
```

```
return (true && true); ==> // קיבלנו "אמת" שני (בצהוב) כי הערך 8 קטן מהערך 15 או שווה לו
```

```
return (true); → כלומר, לפי ההיגיון הזה – של פונקציית ההשוואה – המכונות זהות.
```

דוגמה ב':

אם שתי המכונות יוצרו על ידי יצרן יפני (Honda, Mitsubishi, Nissan, Suzuki, Toyota, Daihatsu או Subaru)

פתרון:

ראשית כול נבנה פונקציית מחלקה העובדת על אובייקט מסוג רכב ומחזירה "אמת" אם שם היצרן הוא יפני.

ההרשאה של הפונקציה תהיה פרטית – מפני שפונקציית ההשוואה (Equals) תפעיל אותה מתוך המחלקה:

<pre>private bool from_japan() { bool made_in_japan = false; switch (this.manufacturer) { case "Honda": case "Mitsubishi": case "Daihatsu": case "Toyota": case "Suzuki": case "Nissan": case "Subaru": made_in_japan = true; break; } return (made_in_japan); }</pre>	הסבר: הפונקציה עובדת על אובייקט מסוג רכב. קיים בפונקציה משתנה מקומי ששמו <code>made_in_japan</code>. המשתנה הזה מאותחל ל"שקר". אם קיבלנו שם יצרן שאינו יפני – לדוגמה, "פיאט" – הפונקציה תחזיר אותו. כלומר, יוחזר "שקר". אם קיבלנו שם יצרן יפני (אחד מ-7 היצרנים שנכתבו ברשימה) – אזי ערך המשתנה ישונה ל"אמת", ובסופו של דבר הפונקציה תחזיר "אמת".
--	--

כעת נרשום את פונקציית ההשוואה:

<pre>public override bool Equals(Object obj) { Vehicle second = (Vehicle)obj; // (1) if (this != second) // (2) { return (this.from_japan().Equals(second.from_japan())); } else { Console.WriteLine("It is the same Vehicle"); return (false); } }</pre>	הסבר: הפונקציה תחזיר "אמת" אם שתי המכוניות מיוצרות ביפן. ושוב – <code>this</code> מייצגת את האובייקט השמאלי (או הראשון) ו-<code>second</code> מייצגת את האובייקט שאחרי הנקודה (או השני).
---	---

דוגמת הרצה ראשונה:

```
Vehicle v = new Vehicle("Suzuki", 220, "BLUE", 125);
Vehicle v2 = new Vehicle("Mitsubishi", 245, "RED", 180);
Console.WriteLine(v);
Console.WriteLine(v2);
rc();
```


פלט ההרצה:

```
Suzuki 220 BLUE 125
Mitsubishi 245 RED 180
True
```

הסברים:

הפעלת הפונקציה הבוליאנית `from_japan()` על הרכב הראשון – `v`
`Console.WriteLine(v.Equals(v2));`

החזירה "אמת", שהרי היצרן "`Suzuki`" מופיע ברשימת היצרנים מיפן.
 באופן דומה, הפעלת הפונקציה הבוליאנית `from_japan()` על הרכב השני – `v2`

`Console.WriteLine(v.Equals(v2));`

החזירה גם היא "אמת", שהרי גם היצרן "`Mitsubishi`" מופיע ברשימת היצרנים מיפן.

הפקודה שמשווה זאת נמצאת בפונקציית `Equals` והיא:

```
return (this.from_japan().Equals(second.from_japan()));
```

דוגמת הרצה שנייה:

```
Vehicle v = new Vehicle("Renault", 145, "BLUE", 115);
Vehicle v2 = new Vehicle("Mitsubishi", 195, "RED", 133);
Console.WriteLine(v);
Console.WriteLine(v2);
rc();
```

פלט ההרצה:

```
Renault 145 BLUE 115
Mitsubishi 195 RED 133
False
```

הסברים:

הפעלת הפונקציה הבוליאנית `from_japan()` על הרכב הראשון – `v`
`Console.WriteLine(v.Equals(v2));`

החזירה "שקר", שהרי היצרן "**Renault**" אינו מופיע ברשימת היצרנים מיפן.
 באופן דומה, הפעלת הפונקציה הבוליאנית `from_japan()` על הרכב השני – `v2`

`Console.WriteLine(v.Equals(v2));`

החזירה "אמת", שהרי היצרן "**Mitsubishi**" מופיע ברשימת היצרנים מיפן.

כעת מדובר בתנאי של:

False AND True

ולכן מדובר בערך מוחזר מסוג "שקר".

תרגול

- א. צרו שני אובייקטים נוספים והפעילו על אחד מהם (לבחירתכם) את פונקציית ההשוואה מהמשימה הראשונה – כך שפעם אחת היא תחזיר "אמת" ופעם שנייה "שקר".
- ב. יש להקליד בצורה טקסטואלית את פרטי כל האובייקטים.
- ג. הוסיפו בנאי מעתיק (Copy Constructor).
- ד. הרכב יהיה מסוג "Daihatsu", מהירותו המקסימלית היא 155 קמ"ש, צבעו אפור ומהירותו הנוכחית היא 0 קמ"ש.
- ה. צרו אובייקט ("רכב") מסוג הבנאי השני.
- ו. צרו שני אובייקטים נוספים והפעילו על אחד מהם (לבחירתכם) את פונקציית ההשוואה מהמשימה הראשונה – כך שפעם אחת היא תחזיר "אמת" ופעם שנייה "שקר".
- ז. יש להקליד בצורה טקסטואלית את פרטי כל האובייקטים.

פרק 6 – פעולות סטטיות

פעולה/פונקציה היא קטע קוד בעל שם מסוים, המסוגל לקבל ערכי קלט לתוך משתנים מסוימים ולהחזיר ערך בודד.

השימוש בפעולות/פונקציות מאפשר לנו לעשות בהן שימוש חוזר – כלומר, להשתמש באותו קטע קוד יותר מפעם אחת במהלך התוכנית שאנו כותבים.

לפונקציה יש מטרה ברורה ומוגדרת בצורה-חד משמעית, כמו למשל חישוב ממוצע, הדפסת הערך הגבוה ביותר וכדומה.

מוסכמות

- א. שם הפונקציה (בדומה לשם משתנה) חייב להיות בעל משמעות (כגון Max, Greater וכדומה).
- ב. אסור לקרוא לפונקציה בשמות של פקודות שמורות (קיימות) כגון if, WriteLn וכדומה.
- ג. שם פונקציה (שוב, בדומה למשתנה) חייב להתחיל באות.
- ד. שם פונקציה אינו יכול להכיל תווים כגון %, ^, #, +, \$ ועוד.
- ה. מקובל להשתמש בקו תחתון או בתו מינוס בשמות פונקציות. לדוגמה, min_value או max-number.

מטרת השימוש בפונקציות

המטרה העיקרית של שימוש בפונקציות היא לחלק את התוכנית בהתאם למרכיביה השונים (לדוגמה, קלט, חישוב מסוים ופלט).

כמו כן, נכתוב פונקציה ראשית (Main) קטנה ויעילה עד כמה שאפשר.

הפונקציה הראשית מזמנת, בשלב כזה או אחר, את הפונקציות האחרות בהתאם להקשר הלוגי של התוכנית. השימוש בפונקציות מאפשר לנו להימנע משכפול קוד וכן משימוש חוזר בקוד במהלך התוכנית ו/או תוכניות אחרות. לכן, החיסכון בזמן הכתיבה משמעותי מאוד: נוכל להגדיר משימה **פעם אחת** בלבד ולהשתמש בה שוב ושוב במהלך חיי התוכנית ואף בתוכניות אחרות.

1. דוגמה ראשונה לשימוש בפונקציה

אנו משתמשים בפונקציה Write או WriteLine – הנמצאת במחלקת Console:

```
Console.WriteLine("Chapter6");
```

נשתמש במילה השמורה return ("החזר"), ומיד אחריה נציין את הערך שנבקש להחזיר לפונקציה המזמנת.

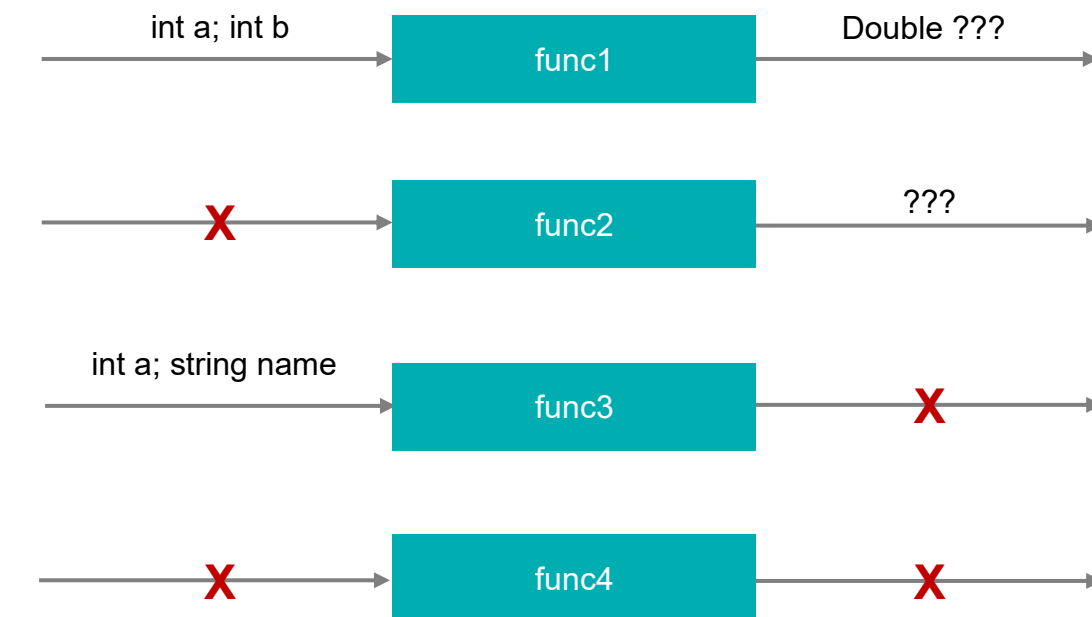
2. דוגמה שנייה לשימוש בפונקציה

נוכל להשתמש בפונקציה השמורה של חישוב ערך מוחלט:

```
Console.WriteLine(Math.Abs(-12.20));
```

הפונקציה הזו מובנית (כלומר, מישהו כתב אותה בעבר) במחלקה בשם Math.

קיימים ארבעה סוגי תבניות של פונקציות:



הסברים

- פונקציה סטטית (static) היא פונקציה שאפשר להפעילה בכל מקרה מבלי ליצור אובייקט מסוג המחלקה.
- public היא הרשאה ציבורית. כלומר, אפשר להפעיל פונקציה מסוימת מכל מחלקה בתוכנית ולא דווקא במחלקה שבה היא הוגדרה.

תבנית מספר 1

משמעות

פונקציה המקבלת פרמטר אחד (או יותר) ומחזירה ערך בודד.

חשוב להבין

פונקציה עשויה לקבל ערך בודד או מספר ערכים, אבל מסוגלת להחזיר ערך אחד ויחיד.

דוגמה:

רואה חשבון מקבל את הנתונים הבאים של עובד: שם מלא, מספר זהות, ותק, תעריף לשעה ועוד. רואה החשבון מחזיר – כלומר, נותן לאותו עובד – בסופו של דבר תלוש משכורת (אחד ויחיד).

דוגמאות לאבות-טיפוס

1. דוגמה ראשונה:

```
double func1(int a, int b);
```

שם הפונקציה: func1

מקבלת: הפונקציה מקבלת שני ערכים שלמים.

מחזירה: הפונקציה מחזירה ערך ממשי.

2. דוגמה שנייה:

```
string GetName(int a, char b);
```

שם הפונקציה: GetName

מקבלת: הפונקציה מקבלת שני ערכים: ערך מסוג שלם וערך מסוג תו.

מחזירה: הפונקציה מחזירה ערך מסוג מחרוזת.

דוגמאות קוד

1. דוגמת קוד ראשונה:

```
public static double CalcAvg(int mark1,int mark2)
{
    return((mark1 + mark2) / 2.0);
}
```

הסברים:

<code>public static double CalcAvg (int mark1,int mark2)</code> נשים לב כי מימין לשם הפונקציה מופיעים הפרמטרים שהפונקציה מקבלת וסוגיהם. בדוגמה זו, הפונקציה מקבלת שני מספרים שלמים (int).	א.
<code>public static double CalcAvg Avg(int mark1,int mark2)</code> משמאל לשם הפונקציה יש לציין את סוג טיפוס הנתונים שהפונקציה מחזירה. להבדיל מיכולתה של הפונקציה לקבל מספר פרמטרים – היא מסוגלת להחזיר ערך אחד בלבד.	ב.
<code>public static double CalcAvg (int mark1,int mark2)</code> <code>{</code> <code>return((mark1 + mark2) / 2.0);</code> <code>}</code> בגוף הפונקציה (בתוך הסוגריים המסולסלים) מופיעה הפקודה return – המציינת החזרה של ערך מסוים לפונקציה המזמנת.	ג.
ד. אפשר לעשות עם ערך מוחזר של פונקציה דברים נוספים. לדוגמה, להקליד את הערך, להציב אותו בתוך משתנה מסוים או לשלוח אותו כפרמטר לפונקציה אחרת.	

הצגת קוד

```

7 namespace ConsoleApp3
8 {
9     class Program
10    {
11        public static double calcAvg(int mark1,int mark2)
12        {
13            return((mark1 + mark2) / 2.0);
14        }
15        static void Main(string[] args)
16        {
17            double avg = (calcAvg(72, 88));
18            Console.WriteLine(calcAvg(80,65));
19            Console.WriteLine("avg is: "+avg);
20
21            Console.ReadKey();
22
23        }
24    }
25 }

```

בשורה 17 מזומנת הפונקציה CalcAvg, והערך המוחזר שלה מוצג למשתנה avg.
 בשורה 18 שוב מזומנת הפונקציה CalcAvg – אך הפעם הערך המוחזר מודפס ישירות למסך.

בשורה 19 אנו מקלידים את הערך שהוצב בשורה 17 במשתנה avg.

2. דוגמת קוד שנייה:

נכתוב פונקציה המקבלת ציון ומחזירה ערך בוליאני ("אמת" או "שקר") – האם הציון חוקי (כלומר, בין 0 [כולל] ל-100 [כולל]).

את הפונקציה הזו נציג בשלוש דרכים (המופיעות גם בפרק 3).

א.

דרך א':

שימוש בקשר לוגי and בתוך הפונקציה:

```
public static bool IsLegal(int mark)
{
    if (mark >= 0 && mark <= 100)
        return (true);
    else
        return (false);
}
```

ב.

דרך ב':

שימוש בקשר לוגי or. האם הפונקציה הבאה שקולה (כלומר, מבצעת את אותה המשימה)?

```
public static bool IsLegal (int grade)
{
    if (grade < 0 || grade > 100)
        return (false);
    return (true);
}
```

כן! בסופו של דבר, שם הפרמטר שהפונקציה מקבלת שונה, אך עדיין נבדק קיומו של ציון חוקי.

בפונקציה זו לא השתמשנו בפקודה else בגוף הפונקציה. האם זו טעות?

לא!

```
public static bool IsLegal(int mark)
{
    if (mark >= 0 && mark <= 100)
        return (true);
    else
        return (false);
}
```

ביצוע הפונקציה נפסק בביצוע המילה השמורה return.

אם הציון אינו חוקי – הפונקציה תחזיר את הערך "שקר" (בשורה 14).

פונקציה אינה יכולה להחזיר ערך יותר מפעם אחת. אם הציון חוקי – הפונקציה תתקדם בביצוע הקוד ותחזיר את הערך "אמת" בשורה 15.

א. דרך ג':

דוגמה הפוכה מדוגמה א' בשימוש קשר לוגי (not)!:

```
public static bool IsLegal(int grade)
{
    if (!((grade < 0 || grade > 100)))
        return (true);
    return (false);
}
```

תבנית מספר 2

משמעות:

פונקציה שאינה מקבלת פרמטרים – ומחזירה ערך בודד. עבור פונקציה שאינה מקבלת פרמטרים נכתוב סוגריים ריקים או סוגריים שבתוכם מופיעה המילה void.

דוגמאות לאבות-טיפוס:

1. דוגמה ראשונה:

```
public static string GetTime();
```

שם הפונקציה: GetTime()

אינה מקבלת ערך.

מחזירה: הפונקציה מחזירה ערך מסוג מחרוזת, המייצג את השעה הנוכחית במחשב.

2. דוגמה שנייה:

```
public static int printRnd1_10();
```

שם הפונקציה: printRnd1_10();

אינה מקבלת ערך.

מחזירה: הפונקציה מגרילה (בדומה לזריקת קובייה) מספר בין 1 ל-10, ומדפיסה בלולאה את המחרוזת "Type_B" מספר פעמים לפי המספר שהוגרל.

הפונקציה מחזירה את המספר שהוגרל.

דוגמאות קוד:

שימו לב כי אנו משרשרים (בדומה לחיבור) את המחרוזת שלב אחר שלב:

1. דוגמת קוד ראשונה:

```
public static string GetTime()
{
    DateTime now = DateTime.Now;

    string output="";
    output += now.Hour;
    output += ":";
    output += now.Minute;
    output += ":";
    output += now.Second;

    return(output);
}
```

הסברים:

<pre>public static string GetTime()</pre>	א. הפונקציה אינה מקבלת פרמטרים כלשהם.
<pre>DateTime now = DateTime.Now;</pre>	ב. אנו מגדירים בפונקציה אובייקט בשם now, השומר את הזמן והתאריך המוגדרים במחשב.
<pre>string output="";</pre>	ג. אנו מגדירים מחרוזת יעד – שאותה תחזיר הפונקציה. בשלב זה המחרוזת ריקה (כלומר, ללא תווים).
<pre>output+=now.Hour;</pre>	ד. תחילת השרשור. כלומר, חיבור המחרוזת. בשלב זה המחרוזת מכילה רק את השעה. אם השעה היא 08:00 בבוקר, אזי המחרוזת היא "8".
	ה. המשך השרשור. אנו ממשיכים לחבר למחרוזת את הסימן ":" (נקודתיים), את הדקה, ושוב ":". בסיום אנו משרשרים למחרוזת (כלומר, מחברים) את השנייה של הזמן הנוכחי במחשב.
<pre>return(output);</pre>	ו. בסופו של דבר, המחרוזת שתוחזר מתארת את הזמן במחשב. לדוגמה, היא עשויה להיות "12:6:49".

בדוגמה זו השעה במחשב היא 12 (בצהריים), 6 דקות ו-49 שניות. מה צריך לבצע על מנת שההדפסה תהיה כמו תצוגת שעה בשעון "אמיתי" (כלומר, כך: 12:06:49)?

פתרון:

אם השעה או הדקה או השנייה קטנות מערך דו-ספרתי, נחבר (נשרשר) "0" (אפס) למחרוזת.

שלבי הפתרון:

- א. אם השעה היא 0 (חצות) עד 9 (תשע בבוקר) – נוסיף, כאמור, את התו "0".
- ב. אותו עיקרון גם לגבי השעה ולגבי הדקה.

נכתוב מחדש את הפונקציה עם השינויים הנדרשים:

```
public static string get_Time()
{
    DateTime now = DateTime.Now;

    string output="";
    output+=now.Hour;
    if (now.Hour < 10) output+="0";
    output+= ":";
    if (now.Minute < 10) output+="0";
    output+= now.Minute;
    output+= ":";
    if (now.Second < 10) output+="0";
    output+= now.Second;

    return (output);
}
```

כעת הפלט יהיה דומה לתצוגת שעה בשעון:

כלומר, 12:06:49 (כולל הספרה 0 לפני הספרה 6).

2. דוגמת קוד שנייה:

```
class Program
{
    public static int printRnd1_10()
    {
        int number;
        int i;
        Random r = new Random();
        number = r.Next(10) + 1;
        return(number);
    }

    static void Main(string[] args)
    {
        Console.WriteLine(printRnd1_10());
        Console.ReadKey();
    }
}
```

הסברים:

א.	הפונקציה אינה מקבלת פרמטר כלשהו.
ב.	R הוא אובייקט מסוג Random.
ג.	הפקודה r.Next(10) מחוללת מספר אקראי בין 0 ל-9. הוספת הערך 1 למספר האקראי הופכת אותו למספר בין 1 ל-10.
ד.	בולאת ה-for מודפסת המחרוזת כמספר הפעמים שנבחר. הערך הנ"ל מוצב במשתנה number.
ה.	ערך המשתנה number מוחזר לתוכנית הראשית ומודפס (בפונקציית ה-Main): Console.WriteLine(printRnd1_10());

פלט לדוגמה:

תבנית מספר 3

משמעות

פונקציה המקבלת פרמטר אחד לפחות ואינה מחזירה ערך כלשהו.

עבור פונקציה שאינה מחזירה ערך כלשהו – נרשום את המילה השמורה void.

דוגמאות לאבות-טיפוס

1. דוגמה ראשונה:

```
public static void GetInterval(int low,int high,int value);
```

שם הפונקציה: GetInterval

מקבלת: שלושה ערכים מסוג שלם (int).

מחזירה: אין ערך מוחזר.

2. דוגמה שנייה:

```
public static void Roots(int a,int b,int c);
```

שם הפונקציה: Roots

מקבלת: שלושה ערכים מסוג שלם (int).

מחזירה: אין ערך מוחזר.

3. דוגמה שלישית:

```
public static void Mantisa(double t,int q)
```

שם הפונקציה: Mantisa

מקבלת: שני ערכים – האחד מסוג שלם (int) והשני מסוג ממשי (double).

מחזירה: אין ערך מוחזר.

דוגמאות קוד:

1. דוגמת קוד ראשונה:

```
class Program
{
    public static void GetInterval(int low,int high,int value)
    {
        if ((value >= low) && (value <= high))
            Console.WriteLine("Between");
        else
            Console.WriteLine("Not Between");
    }

    static void Main(string[] args)
    {
        GetInterval(20, 45, 50);
        GetInterval(5, 55, 25);

        Console.ReadKey();
    }
}
```

הסברים:

<pre>public static void GetInterval(int low,int high,int value)</pre> א. הפונקציה מקבלת שלושה פרמטרים: גבול תחתון, גבול עליון וערך לבדיקה.
ב. אם הערך (value) – שהוא הפרמטר השלישי המועבר לפונקציה – נמצא בין הערכים low ו-high, הפונקציה תדפיס "Between". אחרת היא תדפיס "Not Between".
ג. דוגמה להדפסה הראשונה – הזימון הראשון בפונקציה הראשית: <pre>GetInterval(20, 45, 50);</pre> הערך 50 אינו נמצא בין 20 (גבול תחתון) ל-45 (גבול עליון), ולכן ההדפסה תהיה: "Not Between"
ד. דוגמה להדפסה השנייה – הזימון הראשון בפונקציה הראשית: <pre>GetInterval(20, 45, 50);</pre> הערך 25 נמצא בין 5 (גבול תחתון) ל-55 (גבול עליון), ולכן ההדפסה תהיה: "Between"

דוגמת קוד שנייה:

```

class Program
{
    public static void roots(int a,int b,int c)
    {
        double delta = Math.Pow(b, 2) - 4 * a * c;
        if ((delta)>0)
            Console.WriteLine("There are two roots to the equation.");
        else if (Math.Sqrt(delta) == 0)
            Console.WriteLine("There is only one root to the equation.");
        else
            Console.WriteLine("There is no solution to the equation!");
    }

    static void Main(string[] args)
    {
        roots(2, 7, 3);
        roots(1, 4, 4);
        roots(9, 4, 2);
        Console.ReadKey();
    }
}

```

הסברים:

<code>public static void roots(int a,int b,int c)</code> א. הפונקציה מקבלת שלושה פרמטרים המייצגים פרמטרים של משוואה ריבועית. לא נפתור את המשוואה הריבועית הבאה: $X_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
ב. הפונקציה תחשב את הערך של הדלתא $\sqrt{b^2 - 4ac}$, כלומר, את תוצאת החישוב של הוצאת השורש. אם התוצאה גבוהה מ-0, אזי יש שני פתרונות (שני שורשים שונים); אם היא שווה ל-0, אזי יש פתרון יחיד (כלומר, שני שורשים זהים). אם הערך של הדלתא שלילי (כלומר, קטן מ-0) – אזי אין פתרון למשוואה.
ג. שימו לב לשימוש הנעשה כאן ב-<code>if</code> מקונן.
ד. חישבנו את ערכו של <code>b</code> בריבוע (<code>b</code> בחזקת 2) באמצעות הפונקציה השמורה <code>pow</code>: <code>Math.pow(p,2);</code>
ה. בצורה דומה, חישבנו את ערכו של שורש הדלתא באמצעות הפונקציה השמורה <code>Sqrt</code>: <code>Math.Sqrt(delta)>0)</code>
ו. מה יהיה הפלט של כל אחד משלושת הזימונים בפונקציה הראשית?

2. דוגמת קוד שלישית:

"מנטיסה" היא רמת הדיוק אחרי הנקודה העשרונית:

```
public static void Mantisa(double t,int q)
{
    Console.WriteLine("{0:N"+q+"} ", t);
}

static void Main(string[] args)
{
    double value = 1.235678;
    Console.WriteLine(value);
    Mantisa(value ,2);
    Mantisa(value, 3);
    Mantisa(value, 4);
    Console.ReadKey();
}
```

הסברים:

<pre>public static void Mantisa(double t,int q)</pre> א. כאמור, הפונקציה מקבלת שני פרמטרים. הפרמטר הראשון הוא המספר שאותו היא מקבלת, והפרמטר השני הוא מציין רמת הדיוק – כלומר, כמה ספרות עשרוניות יודפסו אחרי הנקודה העשרונית.	
<pre>Console.WriteLine(value);</pre> ב. בפקודה זו אנו מקלידים את המספר כמו שהוא.	
<pre>Mantisa(value ,2);</pre> ג. בפקודה זו אנו שולחים את המספר המקורי לפונקציה, והוא מודפס עם דיוק בן שתי ספרות: 1.24 שימו לב לעיגול ש-C# מבצעת: בגלל החלק הממשי (.356) – המספר שונה ל-1.24.	
<pre>Mantisa(value ,3);</pre> ד. בפקודה זו אנו שולחים את המספר המקורי לפונקציה, והוא מודפס עם דיוק בן שלוש ספרות: 1.236 שימו לב לעיגול ש-C# מבצעת: בגלל החלק הממשי (.678), המספר שונה ל-1.236.	
<pre>Mantisa(value ,4);</pre> ה. בפקודה זו אנו שולחים את המספר המקורי לפונקציה, והוא מודפס עם דיוק בן ארבע ספרות: 1.2357 שימו לב לעיגול ש-C# מבצעת: בגלל החלק הממשי (.78), המספר שונה ל-1.2357.	

תבנית מספר 4

משמעות:

פונקציה שאינה מקבלת דבר ואינה מחזירה דבר.

כזכור, עבור פונקציה שאינה מחזירה ערך – נרשום את המילה השמורה void.

דוגמאות לאבות-טיפוס:

1. דוגמה ראשונה:

```
public static void Instructions();
```

שם הפונקציה: Instructions

מקבלת: אינה מקבלת דבר.

מחזירה: אין ערך מוחזר.

2. דוגמה שנייה:

```
public static void Cube();
```

שם הפונקציה: Cube

מקבלת: אינה מקבלת דבר.

מחזירה: אין ערך מוחזר.

דוגמאות קוד:

1. דוגמת קוד ראשונה:

```
public static void Instructions()
{
    Console.WriteLine("Press 8 To move up");
    Console.WriteLine("Press 2 To move down");
}

static void Main(string[] args)
{
    Instructions();
    Console.ReadKey();
}
```

הסברים:

א. שימו לב לזימון של הפונקציה Instructions מהפונקציה הראשית.
הפונקציה מציגה, כביכול, הסברים למשחק מחשב מסוים.

2. דוגמת קוד שנייה:

```
public static void Cube()
{
    int number;
    Random r = new Random();
    number = r.Next(1, 7); // will be number between 1..6
    Console.WriteLine(number);
}

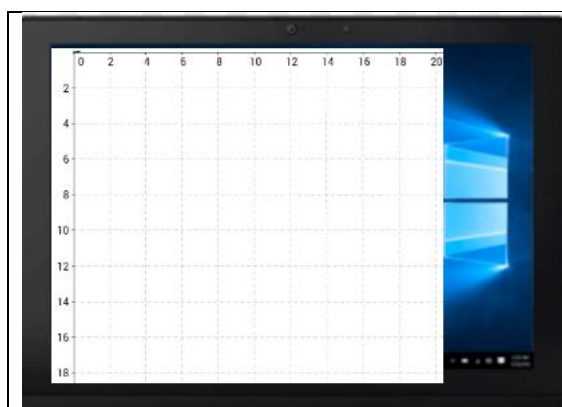
static void Main(string[] args)
{
    Cube();
    Console.Read()
}
```

הסברים:

א. שימו לב לזימון של הפונקציה Cube מהפונקציה הראשית.
הפונקציה מציגה בכל הרצה מספר אקראי בין 1 ל-6 (כמו קובייה).

ב. חשוב להבין כי הפקודה `r.Next(1, 7)` מחוללת מספר אקראי בין 1 ל-6 (ולא 7).

פונקציה המקבלת אובייקט



נניח שיש לנו טאבלט:

תצוגת הטאבלט בנויה מאוסף של פיקסלים (נקודות). מיקומה של הפינה השמאלית-עליונה של מסך הטאבלט הוא (0,0).

לשם הדוגמה נניח כי ציר ה-X (הציר האופקי) נע בין 0 ל-1024, וציר ה-Y (הציר האנכי) נע בין 0 ל-640. ניסיון להדפיס נקודה מחוץ לגבולות אלו יגרור לכך שהנקודה לא תוצג.

ניזכר במחלקת "נקודה" (Point) שהגדרנו בפרק 5.

נוסיף לה מספר פעולות:

1. דוגמה מספר 1

<pre>public static bool InScreen(Point p) { bool inside = true; //(1) if ((p.GetX() < 0) (p.GetX() > 1024)) //(2) inside = false; if ((p.GetY() < 0) (p.GetY() > 640)) //(3) inside = false; return (inside); //(4) }</pre>	פונקציה המקבלת אובייקט מסוג "נקודה" ומחזירה "אמת" אם הנקודה בתוך גבולות מסך הטאבלט, אחרת היא תחזיר "שקר".
---	--

הסברים:

<pre>bool inside = true;</pre> א. בשורה זו אנו מגדירים משתנה בוליאני ומאתחלים אותו בערך "אמת". אם נגלה במהלך התוכנית ששדה x ו/או שדה y נמצאים מחוץ לגבולות הטאבלט – נשנה אותו ל"שקר".	
<pre>if ((p.GetX() < 0) (p.GetX() > 1024)) inside = false;</pre> ב. בשורה זו אנו בודקים האם ערך ה-x של הנקודה נמצא בין גבולות ה-x של הטאבלט, כלומר, בין 0 ל-1024. אם לא – ערך המשתנה הבוליאני משתנה ל"שקר".	
<pre>if ((p.GetY() < 0) (p.GetY() > 640)) inside = false;</pre> ג. בשורה זו אנו בודקים האם ערך ה-y של הנקודה נמצא בין גבולות ה-y של הטאבלט, כלומר, בין 0 ל-640. אם לא – ערך המשתנה הבוליאני משתנה ל"שקר".	
<pre>return (inside);</pre> ד. ערכו של inside מוחזר. אם הוא לא השתנה (בשליבים ב' ו-ג') – הוא נשאר "אמת". אחרת ערכו משתנה ל"שקר".	

כעת נבצע על בסיס פונקציה זו מספר הדגמות באמצעות טבלת מעקב:

דוגמה 1.1:

הנקודה הנבדקת P	Inside	הסברים/הערות	ערך מוחזר של הפונקציה
P x 10 y 2			
	true	בשורה מספר (1) – ערכו של inside מאותחל לערך "אמת".	
		בשורה מספר (2) אנו בודקים האם ערכו של השדה x חורג מגבולות המסך: האם $10 < 0$? התשובה היא "שקר". האם $10 > 1024$? התשובה היא "שקר". קיבלנו את הביטוי <code>False False</code> ; ערכו הוא False, ולכן inside שומר על ערכו והוא עדיין "אמת".	
		בשורה מספר (3) אנו בודקים האם ערכו של השדה y חורג מגבולות המסך: האם $2 < 0$? התשובה היא "שקר". האם $2 > 640$? התשובה היא "שקר". קיבלנו את הביטוי <code>False False</code> ; ערכו הוא False, ולכן inside שומר על ערכו והוא עדיין "אמת".	
			ערכו של inside לא השתנה והפונקציה תחזיר "אמת".

דוגמה 1.2:

הנקודה הנבדקת P	Inside	הסברים/הערות	ערך מוחזר של הפונקציה
P x -4 y 15			
	true	בשורה מספר (1) – ערכו של inside מאותחל לערך "אמת".	
	false	בשורה מספר (2) אנו בודקים האם ערכו של השדה x חורג מגבולות המסך: האם $-4 < 0$? התשובה היא "אמת". האם $-4 > 1024$? התשובה היא "שקר". קיבלנו את הביטוי <code>False True</code> ; ערכו הוא <code>True</code> , ולכן inside משנה את ערכו ל"שקר".	
		בשורה מספר (3) אנו בודקים האם ערכו של השדה y חורג מגבולות המסך: האם $15 < 0$? התשובה היא "שקר". האם $15 > 640$? התשובה היא "שקר". קיבלנו את הביטוי <code>False False</code> ; ערכו הוא <code>False</code> , ולכן inside שומר על ערכו והוא עדיין "שקר".	
			ערכו של inside השתנה (בשורה 2), והפונקציה תחזיר "שקר".

דוגמה 1.3:

הנקודה הנבדקת P	Inside	הסברים/הערות	ערך מוחזר של הפונקציה				
P <table><tr><td>x</td><td>12</td></tr><tr><td>y</td><td>720</td></tr></table>	x	12	y	720			
x	12						
y	720						
	true	בשורה מספר (1) – ערכו של inside מאותחל לערך "אמת".					
		בשורה מספר (2) אנו בודקים האם ערכו של השדה x חורג מגבולות המסך: האם $12 < 0$? התשובה היא "שקר". האם $12 > 1024$? התשובה היא "שקר". קיבלנו את הביטוי <code>False False</code> ; ערכו הוא False, ולכן inside שומר על ערכו והוא עדיין "שקר".					
	false	בשורה מספר (3) אנו בודקים האם ערכו של השדה y חורג מגבולות המסך: האם $720 < 0$? התשובה היא "שקר". האם $720 > 640$? התשובה היא "אמת". קיבלנו את הביטוי <code>False True</code> ; ערכו הוא True, ולכן inside משנה את ערכו ל"שקר".					
			ערכו של inside השתנה (בשורה 3), והפונקציה תחזיר "שקר".				

דוגמה מספר 2

<pre>public static bool MergePoints(Point a,Point b) { int new_x, new_y; // (1) new_x = a.GetX() + b.GetX(); // (2) new_y = a.GetY() + b.GetY(); // (3) Point MergePoint = new Point(new_x, new_y);//(4) if ((InScreen(MergePoint) == true)) // (5) return (true); else return (false); }</pre>	פונקציה המקבלת שני אובייקטים מסוג "נקודה" ומחזירה "אמת". הנקודה נמצאת באמצע המסך.
<pre>public static bool MergePoints(Point a, Point b) { int new_x, new_y; // (1) new_x = a.GetX() + b.GetX(); //(2) new_y = a.GetY() + b.GetY(); //(3) Point MergePoint = new Point(new_x, new_y);//(4) return ((InScreen(MergePoint))); //(5) }</pre>	גרסה מקוצרת של הפונקציה

הסברים:

א. הפונקציה מקבלת שני אובייקטים מטיפוס "נקודה" (האחד בשם a והשני בשם b).
ב. בשלב זה (1) נוצרים שני משתנים מקומיים חדשים, האמורים להחזיק את הקואורדינטות של הנקודה החדשה.
ג. בשלב זה (2) אנו מחברים את ערכי צירי ה-x של שתי הנקודות שקיבלנו לתוך המשתנה שקראנו לו new_x.
ד. בשלב זה (3) אנו מחברים את ערכי צירי ה-y של שתי הנקודות שקיבלנו לתוך המשתנה שקראנו לו new_y.
ה. בשלב זה (4) אנו יוצרים נקודה חדשה המאוחתלת בשני הערכים שחיברנו בשורות (2) ו-(3).
ו. בשלב זה (5) אנו בודקים האם הנקודה החדשה היא בגבולות מסך הטאבלט או לאו.

כעת, על בסיס פונקציה זו, נציג דוגמה באמצעות טבלת מעקב.

הפונקציה הראשית מוצגת כאן:

```
static void Main(string[] args)
{
    Point p1;
    p1 = new Point();
    Point p2 = new Point(9, 12);
    Console.WriteLine(Point.MergePoints(p1, p2)); // (א)
    Console.ReadKey();
}

public static bool MergePoints(Point a, Point b)
{
    int new_x, new_y; // (1)
    new_x = a.GetX() + b.GetX(); //(2)
    new_y = a.GetY() + b.GetY(); //(3)
    Point MergePoint = new Point(new_x, new_y); //(4)
    return ((InScreen(MergePoint) == true)); //(5)
}
```

p1 היא הנקודה הראשונה שהפונקציה MergePoints מקבלת, וערכה הוא (x=512,y=320).
p2 היא הנקודה השנייה שהפונקציה מקבלת, וערכה הוא (x=9,y=12).

a	b	new_x	new_y	MergePoint	InScreen(MergePoint)	ערך מוחזר של הפונקציה
		0				
			0			
a x512 y320						
	b x9 y12					
		512+9 ⇔ 521				
			320+12 ⇔ 432			
				MergePoint x521 y432		
					הזימון של הפונקציה יחזיר "אמת" מפני שאין חריגה מהגבולות.	
						הפונקציה תחזיר "אמת" והזימון שלה (מסומן ב-(א)) ידפיס "אמת".

הפלט שיוצג:

True

פונקציה המחזירה אובייקט

נשנה את הדוגמה הקודמת: הפונקציה תקבל שתי נקודות (כמו במקרה הקודם). אם הנקודה החדשה נמצאת בגבולות המסך – הפונקציה תחזיר אותה. אם הנקודה החדשה חורגת מגבולות המסך – תיווצר נקודה שהיא ברירת מחדל (באמצע המסך). בפונקציה הראשית תודפס הנקודה שהוחזרה.

1. דוגמת הרצה ראשונה

```
static void Main(string[] args)
{
    Point p1;
    p1 = new Point(); // יצירת הנקודה הראשונה
    Point p2 = new Point(9, 12); // יצירת הנקודה השנייה
    Point p3;
    Console.WriteLine("p1 is " + p1);
    Console.WriteLine("p2 is " + p2);
    p3 = Point.CreatePoint(p1, p2); // הנקודה השלישית תקבל את החיבור של שתי הקודמות
    Console.WriteLine("p3 is "+p3);
    Console.ReadKey();
}

public static Point CreatePoint(Point a, Point b)
{
    int new_x, new_y;
    new_x = a.GetX() + b.GetX();
    new_y = a.GetY() + b.GetY();
    Point MergePoint = new Point(new_x, new_y);
    if (InScreen(MergePoint)) // אם הנקודה החדשה בגבולות המסך
        return (MergePoint); // אז תחזיר אותה לפונקציה העליונה
    else
        return (new Point()); // אם לא – צור נקודת אמצע מסך והחזר אותה לפונקציה העליונה
}
```

פלט ההרצה

```
p1 is x=512 y=320
p2 is x=9 y=12
p3 is x=521 y=332
```

הסברים

- א. הנקודה הראשונה (p1) נוצרה בצורת ברירת מחדל (כאמור, באמצע המסך).
- ב. הנקודה השנייה (p2) נוצרה באמצעות בנאי מפורש (מקבל את שתי התכונות).
- ג. הפונקציה CreatePoint החזירה את הנקודה is x=521 y=332 לנקודה ששמה p3.
- ד. שימו לב שערכי p3 לא חוזרים מהמסך. בפונקציה CreatePoint הופעל למעשה ה-return הראשון.
- ה. בפונקציה הראשית p3 (בצהוב) נקבל את הנקודה שאותה מחזירה הפונקציה CreatePoint.

2. דוגמת הרצה שנייה

```
static void Main(string[] args)
{
    Point p1 = new Point(900,500);
    Point p2 = new Point(100, 200);
    Point p3;
    Console.WriteLine("p1 is " + p1);
    Console.WriteLine("p2 is " + p2);
    p3 = Point.CreatePoint(p1, p2);
    Console.WriteLine("p3 is "+p3);
    Console.ReadKey();
}

public static Point CreatePoint(Point a, Point b)
{
    int new_x, new_y;
    new_x = a.GetX() + b.GetX();
    new_y = a.GetY() + b.GetY();
    Point MergePoint = new Point(new_x, new_y);
    if (InScreen(MergePoint)) // אם הנקודה החדשה בגבולות המסך
        return (MergePoint); // אז תחזיר אותה לפונקציה העליונה
    else
        return (new Point()); // אם לא – צור נקודת אמצע מסך והחזר אותה לפונקציה העליונה
}
```

פלט ההרצה

```
p1 is x=900 y=500
p2 is x=100 y=200
p3 is x=512 y=320
```

הסברים

- א. הנקודה הראשונה (p1) נוצרה באמצעות בנאי מפורש (מקבל את שתי התכונות).
- ב. הנקודה השנייה (p2) נוצרה באמצעות בנאי מפורש (מקבל את שתי התכונות).
- ג. הפונקציה CreatePoint החזירה את הנקודה is x=512 y=320 לנקודה ששמה p3. הנקודה החדשה חורגת מגבולות המסך (החיבור של ציר ה-y מחושב כ-500+200, והתוצאה גדולה מערך ציר ה-y, שהוא 640. לכן, הפונקציה CreatePoint מייצרת נקודת ברירת מחדל ומחזירה אותה לתוכנית הראשית. בפונקציה CreatePoint הופעל למעשה ה-return השני.
- ד. בפונקציה הראשית p3 (בצהוב) נקבל את הנקודה שאותה מחזירה הפונקציה CreatePoint.

פרק 7 - מחרוזות

הגדרה ⓘ

תו הוא סימן בודד. הוא יכול להיות אות בשפה כלשהי, סימן פיסוק, ספרה, או כל סימן אחר – גם כאלה שאינם מופיעים במקלדת (לדוגמה, $\sqrt{\quad}$ [שורש] או ☺ [פרצוף מחייך]).
תו בודד יסומן בגרש בודד משני צדדיו. לדוגמה, '\$' או '4' או 'A'.

הגדרה ⓘ

מחרוזת מורכבת מרצף של תווים (תו אחד לפחות) ומסומנת בגרשיים משני צדדיה. לדוגמה, "\$" או "sasi" או "179".

בפתרון בעיות אלגוריתמיות נשתמש במחרוזות עבור שמות, כתובות, מספרי טלפון וכדומה.

```
static void Main(string[] args)
{
    string name1 = "Daniela"; // הגדרת מחרוזת בת 7 תווים
    string name2 = "";        // הגדרת מחרוזת ריקה בת 0 תווים
    Console.ReadKey();
}
```

מבנה המחרוזת

כאמור, מחרוזת היא אוסף של תווים המופיעים בה ברצף. תווים אלה ממוספרים החל מהמספר 0 – שהוא המיקום/אינדקס של התו הראשון במחרוזת.

נסתכל על המחרוזת name1:

	0	1	2	3	4	5	6
name1	D	a	n	i	e	l	a

לכל מחרוזת יש תכונה בשם Length – אורך המחרוזת. כלומר, מספר התווים המרכיבים את המחרוזת.

שאלה לדיון

מה יהיה הפלט של הפקודות הבאות?

```
Console.WriteLine(name1.Length);
Console.WriteLine(name2.Length);
```

פתרון

ערך ההדפסה הראשונה יהיה 7 – מספר התווים במחרוזת "Daniela".
ערך ההדפסה השנייה יהיה 0 – מאחר שבמחרוזת ששמה name2 אין תווים כלל.

אפשרויות להדפסת מחרוזת

- אפשרות א': להדפיס תו אחר תו באמצעות הסימן המוסכם [], המציין אינדקס. כך:

<pre>static void Main(string[] args) { string name1 = "Daniela"; Console.Write(name1[0]); Console.Write(name1[1]); Console.Write(name1[2]); Console.Write(name1[3]); Console.Write(name1[4]); Console.Write(name1[5]); Console.Write(name1[6]); }</pre>	הפלט יהיה של תו אחר תו, ברצף, ללא ירידת שורה: Daniela _ (סמן) פקודת ההדפסה האחרונה מורידה את הסמן לשורה הבאה.
--	--

- אפשרות ב': לבצע את אותה המשימה – כלומר, הדפסת תו אחר תו – באמצעות לולאת for:

<pre>static void Main(string[] args) { string name1 = "Daniela"; int i; for(i=0; i<name1.Length; i++) Console.Write(name1[i]); Console.WriteLine(); // For new line }</pre>	גם במקרה זה הפלט יהיה של תו אחר תו, ללא ירידת שורה: Daniela – פקודת ההדפסה האחרונה מורידה את הסמן לשורה הבאה.
---	---

- האפשרות הקלאסית היא להשתמש בפקודה ToString – המוגדרת במחלקת string – ולהדפיס באמצעות הפקודה הרגילה:

<pre>static void Main(string[] args) { string name1 = "Daniela"; Console.WriteLine(name1); Console.ReadKey(); }</pre>	הפלט זהה: Daniela – פקודת ההדפסה האחרונה מורידה את הסמן לשורה הבאה.
---	---

הגדרה

שרשור הוא חיבור של מחרוזת אחת למחרוזת אחרת, או הפיכת מספר למחרוזת שאותה מצרפים למחרוזת קיימת.

מחרוזת → מחרוזת + מחרוזת

מחרוזת → מספר + מחרוזת

מחרוזת → מחרוזת + מספר

שרשור רגיל קיים בין מספרים.
לדוגמה,

מספר → מספר + מספר
10 + 1.5 → 11.5

דוגמאות לשרשור

• דוגמה א':

יצירת מחרוזת ריקה:

```
String pelet="";  
pelet=pelet+"Danni";
```

– המשמעות היא

חיבור של מחרוזות:

```
"" + "Danni" → "Danni"
```

נמשיך ונשרשר:

```
pelet+=" cohen";
```

– המשמעות היא

```
"Danni" + " cohen" → "Danni cohen"
```

• דוגמה ב':

בכל פעם שאנו משתמשים בהדפסת ייצוג טקסטואלי של אובייקט ב-ToString –
למעשה אנו משתמשים בשרשרת:

```
public override string ToString()
{
    return string.Format("x=" + x + " y="+ y);
}
```

↓ ↓ ↓ ↓

מספר + מחרוזת + מספר + מחרוזת

כלומר, הכול שורשר (חובר) למחרוזת אחת והיא הודפסה למסך.

פעולות על מחרוזות

• Length

הפונקציה עובדת על המחרוזת ומחזירה מספר שלם המייצג את אורך המחרוזת, כלומר, את מספר התווים שמהם היא מורכבת. אם המחרוזת ריקה – יוחזר הערך 0 (אפס).

מה יהיה הפלט של התוכנית הבאה?

```
static void Main(string[] args)
{
    string name = "Udi";
    Console.WriteLine(name.Length);
    Console.ReadKey();
}
```

הפלט של התוכנית יהיה 3.

המחרוזת name אכן מורכבת משלושה תווים:

	0	1	2
name	u	d	i

• Equals

הפונקציה עובדת על מחרוזת אחת ומקבלת כפרמטר מחרוזת אחרת. הפעולה תחזיר ערך אמת אם ורק אם:

- אורך (Length) של שתי המחרוזות זהה.
- כל תו במחרוזת הראשונה זהה – הן במיקום והן בתוכן – לכל תו במחרוזת השנייה (המועברת כפרמטר).

דוגמה א'

מה יהיה הפלט של התוכנית הבאה?

```
static void Main(string[] args)
{
    string s1 = "dan";
    string s2 = "dan";
    Console.WriteLine(s1.Equals(s2));
    Console.ReadKey();
}
```

	0	1	2
s1	d	a	n
	800		

	0	1	2
s2	d	a	n
	606		

שימו לב: שתי המחרוזות הללו זהות מבחינת הנתונים (ערכים) – אך הן שונות מבחינת הכתובות שלהן.

דוגמה ב'

מה יהיה הפלט של התוכנית הבאה?

```
static void Main(string[] args)
{
    string s1 = "dani";
    string s2 = "danI";
    Console.WriteLine(s1.Equals(s2));
    Console.ReadKey();
}
```

	0	1	2	3
s1	d	a	n	i
	800			
	</			

זוהי השוואה המבוססת על סדר אותיות האלפבית (בדומה לרשימת אנשי הקשר בסמארטפון שלנו), כלומר, היא מבוססת על רצף תווים נתון וקבוע ולכן היא אפשרית.

הפונקציה `CompareTo`:

- תחזיר ערך 0 (אפס) אם שתי המחרוזות זהות.
- תחזיר ערך שלילי אם המחרוזת הראשונה קטנה מהמחרוזת השנייה.
- תחזיר ערך חיובי אם המחרוזת הראשונה גדולה מהמחרוזת השנייה.

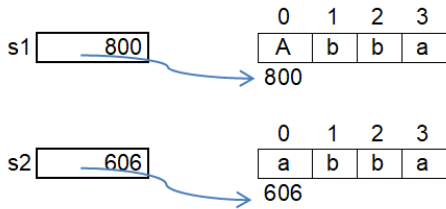
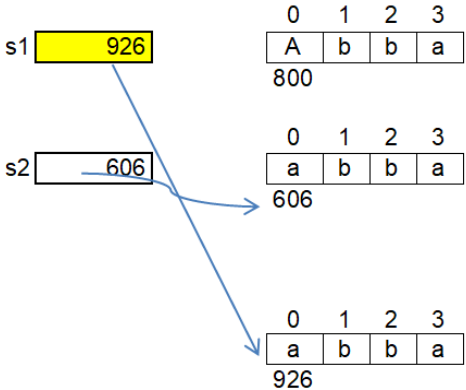
בסיס ההשוואה מכונה גם **השוואה מילונית** או **לקסיקוגרפית**.

הפלט של התוכנית יהיה 1.-. הפונקציה החזירה מספר שלישי, כי אכן מבחינה טקסטואלית (כלומר, לפי סדר האלפבית) המילה "beni" תופיע לפני המילה "gadi".	דוגמה א' <pre>static void Main(string[] args) { string a = "beni"; string b = "gadi"; Console.WriteLine(a.CompareTo(b)); Console.ReadKey(); }</pre> מה יהיה הפלט של התוכנית הבאה?
הפלט של התוכנית יהיה 1. המחרוזת "dan" קטנה יותר מבחינה טקסטואלית מהמחרוזת "udi".	דוגמה ב' <pre>static void Main(string[] args) { string a = "udi"; string b = "dan"; Console.WriteLine(a.CompareTo(b)); Console.ReadKey(); }</pre> מה יהיה הפלט של התוכנית הבאה?

דוגמה ג'	
מה יהיה הפלט של התוכנית הבאה? <pre>static void Main(string[] args) { string a = "abba"; string b = "abba"; Console.WriteLine(a.CompareTo(b)); Console.ReadKey(); }</pre>	הפלט של התוכנית יהיה 0 (אפס). המחרוזות זהות לחלוטין.

• ToLower

פעולה זו עובדת על מחרוזות ומחזירה – בכתובת חדשה – מחרוזת חדשה בעלת אותיות קטנות בלבד.

מה יהיה הפלט של התוכנית הבאה?	שליבי הביצוע של התוכנית
<pre>static void Main(string[] args) { string s1 = "Abba"; // (1) string s2 = "abba"; // (2) s1=s1.ToLower(); //(3) Console.WriteLine(s1.Equals(s2)); Console.ReadKey(); }</pre>	בשתי הפקודות הראשונות (1) + (2) מתבצעות הצבעות:  בפקודה השלישית (3) נוצרת מחרוזת חדשה בכתובת חדשה, ו-s1 מצביעה עליה: 

כאן מצביעה s1 על הכתובת 926 ולא על הכתובת 800. פעולה ההשוואה – <code>Console.WriteLine(s1.Equals(s2));</code> תדפיס אמת: True	
--	--

• ToUpper

פעולה זו עובדת על מחרוזת ומחזירה – בכתובת חדשה – מחרוזת חדשה בעלת אותיות גדולות בלבד.

שלבי הביצוע של התוכנית

בשתי הפקודה הראשונות (1) + (2) מתבצעות הצבעות:

The diagram illustrates the memory state for two string variables, s1 and s2. For s1, a box labeled 's1' contains the memory address '800'. An arrow points from this box to a character array starting at address '800'. The array has four indices: 0 (A), 1 (b), 2 (B), and 3 (a). For s2, a box labeled 's2' contains the memory address '606'. An arrow points from this box to a character array starting at address '606'. The array has four indices: 0 (A), 1 (B), 2 (B), and 3 (A).

בפקודה השלישית (3) נוצרת מחרוזת חדשה בכתובת חדשה, ו-s1 מצביעה עליה:

מה יהיה הפלט של התוכנית הבאה?

```
static void Main(string[] args)
{
    string s1 = "AbBa"; //(1)

    string s2 = "ABBA"; // (2)

    s1=s1.ToUpper(); // (3)

    Console.WriteLine(s1.Equals(s2));

    Console.ReadKey();
}
```

s1 420

0	1	2	3
A	b	B	a

800

s2 606

0	1	2	3
A	B	B	A

606

0	1	2	3
A	B	B	A

420

כעת מצביעה s1 על הכתובת 420 ולא על
 הכתובת 800.
 פעולה ההשוואה –
 Console.WriteLine(s1.Equals(s2)
);
 תדפיס אמת: True

IndexOf •

הפעולה מחזירה את המיקום (אינדקס) של התו – או המחרוזת – שאנו מחפשים בפעם הראשונה. אם התו/מחרוזת לא נמצא – יוחזר הערך 1-.

א'דוגמה

```
static void Main(string[] args)
{
    string msg = "you have a nice
                  dog";

    if (msg.IndexOf("dog")!=-1)
        Console.WriteLine("string
                           contains dog");

    Console.ReadKey();
}
```

מה יהיה הפלט של התוכנית הבאה?

ב'דוגמה

```
static void Main(string[] args)
{
    string msg = "you have a nice
                  dog";

    Console.WriteLine(msg.IndexOf('o'));
    Console.ReadKey();
}
```

מה יהיה הפלט של התוכנית הבאה?

ג'דוגמה

```
static void Main(string[] args)
{
    string msg = "you have a nice
                  dog";

    Console.WriteLine(msg.IndexOf("oo"));
    Console.ReadKey();
}
```

מה יהיה הפלט של התוכנית הבאה?

הפלט של התוכנית יהיה:

string contains dog

הפונקציה החזירה את הערך 16, שהוא המופע הראשון (האחרון) של המחרוזת "dog".

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
msg	y	o	u		h	a	v	e		a		n	i	c	e		d	o	g

הפלט של התוכנית יהיה 1.

הסבר: התו 'ס' (שימו לב שהוא מסומן בגרש בודד ['] ולא בגרשיים ["]) מופיע פעמיים:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
msg	y	o	u		h	a	v	e		a		n	i	c	e		d	o	g

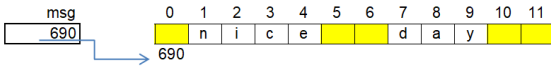
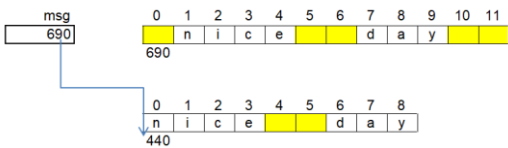
התוכנית החזירה את האינדקס הראשון שנמצא – שהוא, כאמור, 1.

הפלט של התוכנית יהיה 1-.

הסבר: המחרוזת 'z' (שימו לב שהיא מסומנת בגרשיים ["]) ולא בגרש בודד [']) אינה מופיעה כלל, ולכן הוחזר הערך 1-.

Trim •

הפעולה עובדת על המחרוזת ומחזירה מחרוזת חדשה ללא רווחים בתחילת המחרוזת ובסופה.

מה יהיה הפלט של התוכנית הבאה?	הפלט של התוכנית יהיה:
<pre>static void Main(string[] args) { string msg = " nice day "; //(1) Console.WriteLine(msg); // (2) msg = msg.Trim(); // (3) Console.WriteLine(msg); // (4) Console.ReadKey(); }</pre>	לאחר אתחול המחרוזת – בשורה מספר (1) – היא נראית כך:  <u>שימו לב:</u> א. הרווחים מסומנים בצהוב. ב. אורך המחרוזת הוא 11 תווים. לאחר ביצוע שורה מספר (2), ההדפסה של המחרוזת תהיה: nice day – <u>שימו לב:</u> א. בהדפסה לא רואים את שני הרווחים שבסוף המחרוזת. ב. בתחילת המחרוזת יש רווח אחד בלבד. בשורה (3) נוצרת מחרוזת חדשה על בסיס המחרוזת הקודמת ללא רווחים בסוף ובהתחלה. כלומר המחרוזת החדשה (בכתובת שונה) נראית כך: 

שימו לב:	
א. למחרוזת החדשה (בכתובת 440) יש שני רווחים באמצע – כמו במחרוזת הקודמת – אך אין רווחים בתחילתה ובסופה. ב. המחרוזת החדשה קצרה יותר: אורכה הוא 8 תווים בלבד. ג. ההדפסה שלה בשורה (4) תיראה כך: nice day –	

פרק 8 - מבנים סדרתיים - מערך חד ממדי

הגדרה

מערך חד-ממדי הוא טיפוס נתונים מסוג מסוים, שאיבריו ממוקמים בזיכרון בצורה רציפה (סמוכים זה לזה).

כל מערך הוא מטיפוס מסוים, יש לו שם משלו (בדומה למשתנה) וגודל (Length) משלו (בדומה למחרוזת).

את המיקום הסידורי של כל איבר במערך מציין **האינדקס**. אפשר לפנות לכל אחד מהאיברים על פי המספר המייצג את המיקום (אינדקס) שלו במערך – כלומר, את מיקומו הסידורי ברצף. כל מערך, מכל סוג שהוא, מתחיל באינדקס 0.

דוגמאות למערכים

א. מערך של מספרים – בגודל 5

לפנינו מערך בשם grades (ציונים), השומר את הציונים של חמשת התלמידים היושבים בשורה הראשונה של הכיתה:

grades	0	1	2	3	4
690	79	100	88	89	93

690

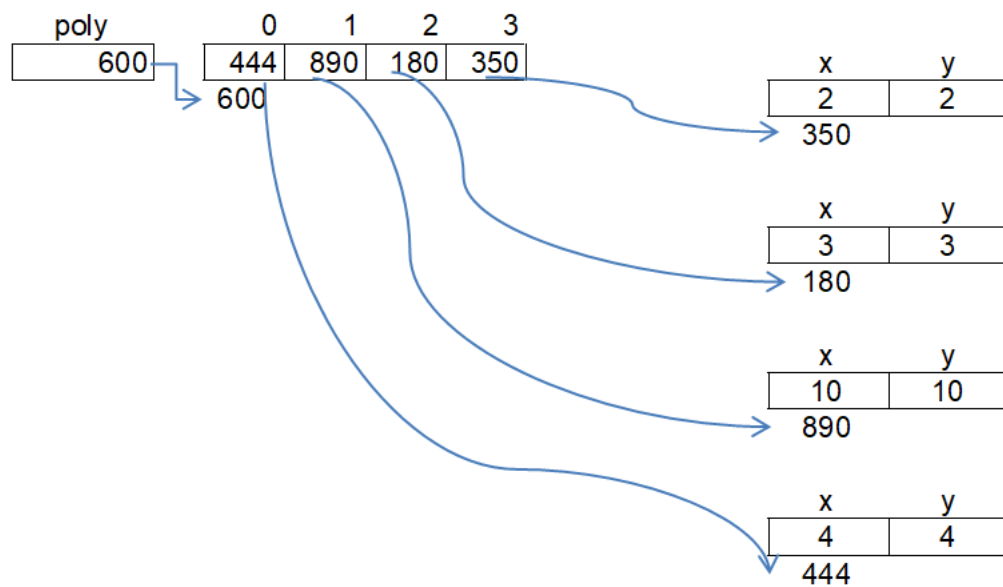
ב. מערך של מספרים ממשיים – בגודל 4

לפנינו מערך בשם prices (מחירים), המאחסן ארבעה מחירים:

prices	0.00	1.00	2.00	3.00
440	90.40	11.70	80.30	112.45

440

מערך של מצביעים לאובייקטים – בגודל 4



יצירת מערך

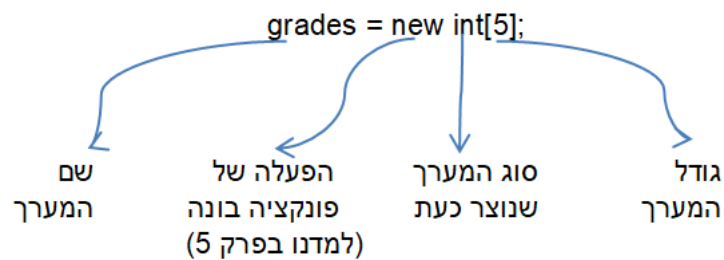
בשלב הראשון נגדיר מצביע למערך:

```
int []grades;
```

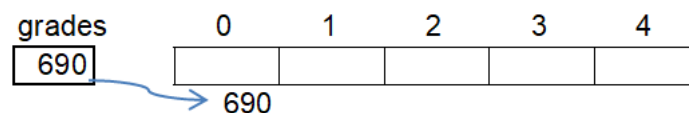
הסימון [] מצייין כי מדובר במערך חד-ממדי. בשלב הזה עדיין לא הוקצה מערך בזיכרון היות ועוד לא מופיעה המילה new.

בשלב השני נצהיר מהו גודל המערך ונשייך אותו לשמו:

```
grades = new int[5];
```



כך נראית תמונת הזיכרון בשלב הזה:



פעולות על איברי המערך

1. אתחול מערך

אפשר ליצור מערך ולאתחל אותו בצורה המופיעה כאן. החיסרון של שיטה זו הוא, שעבור חמישה ציונים נוספים (לדוגמה, שורת התלמידים השנייה בכיתה) נצטרך

```
int[] grades = { 79, 100, 88, 89, 93 };
```

לשנות את הערכים ולהריץ את התוכנית פעם נוספת.

דוגמה לאתחול והדפסת המערך (ללא ירידת שורה):

```
static void Main(string[] args)
{
    int[] grades = { 79, 100, 88, 89, 93 };
    int i; // ניצור משתנה שיספור את איברי המערך

    for (i=0; i<grades.Length; i++) // נשתמש בפונקציית אורך המערך
        Console.Write(grades[i]+" ");

    Console.ReadKey();
}
```

פלט ההרצה:

79 100 88 89 93 _ (סמן)

2. קליטת איברים לתוך המערך

לאחר שיצרנו את המערך, ניצור משתנה שיספור את איבריו. נדפיס הוראה למשתמש כדי שידע כמה ציונים עליו להקליד. גם כאן נשתמש בפונקציה השמורה – גודל המערך (Length):

```
static void Main(string[] args)
{
    int []grades = new int[5];
    int i;
    Console.WriteLine("please enter "+grades.Length+ " grades:");
    for (i = 0; i < grades.Length; i++)
        grades[i] = int.Parse(Console.ReadLine());

    for (i = 0; i < grades.Length; i++)
        Console.Write(grades[i] + " ");
    Console.ReadKey();
}
```

פלט ההרצה:

```
please enter 5 grades:
59
93
84
82
94
59 93 84 82 94
```

3. הדפסת המערך

```
for (i = 0; i < grades.Length; i++)
    Console.Write(grades[i] + " ");
Console.Write(); // New Line
```

4. קביעת אורך מערך על פי משתנה

במקרה זה נשאל את המשתמש לכמה איברים (בדוגמה הבאה מדובר במחירים) הוא זקוק, ובהתאם לתשובתו נקבע את גודל המערך. הלולאות תתבצענה בהתאם למספר האיברים שהמשתמש ביקש. כעת נציג בשלמותה את התוכנית המבצעת מטלה זו, ונשלב בה הסברים:

<pre>static void Main(string[] args) { double[] prices; int i; int size;</pre>	הצהרה על מצביע למערך. מונה לולאה. גודל המערך המבוקש. avg (בשלב הנוכחי הוא יסכם את המחירים, ובהמשך הוא יחשב את הממוצע שלהם).
<pre>Console.WriteLine("How many prices you need ?"); size = int.Parse(Console.ReadLine()); prices = new double[size];</pre>	נשאל את המשתמש לכמה מחירים הוא זקוק. קליטת המספר ל-size. יצירת המערך בהתאמה. לולאת for- מבצעת: א. קליטת מחיר לאינדקס מתאים במערך. ב. סיכום המחירים (על מנת לחשב בהמשך את הממוצע).
<pre>for (i = 0; i < prices.Length; i++) { prices[i]=double.Parse(Console.ReadLine()); avg = avg + prices[i]; }</pre>	הדפסת המערך בדיוק של 2 ספרות אחרי הנקודה. ירידת שורה. חישוב הממוצע. הדפסת הממוצע.

```
for (i = 0; i < prices.Length; i++)  
    Console.Write("{0:N2}    ",  
prices[i]);  
  
    Console.WriteLine(); // New Line  
  
    avg = avg / size;  
  
    Console.WriteLine("Average is: "  
+ avg);  
  
    Console.ReadKey();
```

דוגמאות להרצת התוכנית

1.

How many prices do you need?

4

1.559

20.306

55.559

40.10

1.56 20.31 55.56 40.10

Average is: 29.381

2.

How many prices do you need?

5

100.768

50.555

90.11

40

153.666

100.77 50.56 90.11 40.00 153.67

Average is: 87.0198

פעולות המקבלות מערך כפרמטר

- כשאנו מעבירים מערך לפונקציה – הפונקציה מקבלת את הכתובת של תחילת המערך.
- המשמעות היא, שכל השינויים שנבצע בתוך הפונקציה במערך, יישמרו כאשר נצא ממנה.
- גם אם הפונקציה אינה מחזירה ערך (void) – עדיין יישמרו השינויים במערך שהועבר.

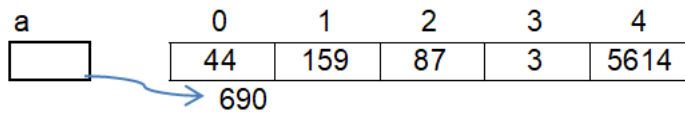
דוגמאות

1. נכתוב פונקציה (השייכת לתבנית ג' בפרק 6) המקבלת את כתובת המערך הקיים בתוכנית הראשית וקולטת לתוכו נתונים.

שימו לב העובדה שהפונקציה לא מחזירה ערך (void) – אינה סותרת את העובדה שהערכים הנקלטים נשמרים בסיום פעולת הפונקציה.

```
static void Get_Array(int []a)
{
    int i;
    Console.WriteLine("Please enter {0} numbers.",a.Length);
    for (i = 0; i < a.Length; i++)
        a[i] = int.Parse(Console.ReadLine());
}
```

2. נכתוב פונקציה (השייכת אף היא לתבנית ג' בפרק 6) המקבלת מערך של מספרים שלמים ומדפיסה את נתוניו. הפונקציה תדפיס את ערכי המערך במרווחים של חמישה תווים/מקומות מוצמדים לשמאל. לדוגמה, אם המערך הוא:



זה הפלט שיוצג על המסך:

4	4				1	5	9			8	7				3					5	6	1	4	
רווחים 3					רווחים 2					רווחים 3					רווחים 4					רווח בודד				

```
static void Show_Array(int[] a)
{
    int i;
    Console.WriteLine("The array is");
    for (i = 0; i < a.Length; i++)
        Console.WriteLine("{0,-5}",a[i]);
    Console.WriteLine();
}
```

כעת נשלב את שתי הפונקציות בתוכנית הראשית:

```
static void Main(string[] args)
{
    int[] array = new int[5]; // יצירת מערך
    Get_Array(array); // קליטת נתונים למערך
    Show_Array(array); // הפונקציה הקודמת
    Console.ReadKey();
}
```

פלט ההרצה

```
Please enter 5 numbers.
132
19
5432
2
77
The array is: 132 19 5432 2 77
```

פעולות המחזירות ערך

1. נכתוב פונקציה (השייכת לתבנית ב' בפרק 6) שאינה מקבלת דבר (void) ומחזירה מערך של איברים שלמים בגודל 7:

```
static int[] Create_Array()
{
    int[] a = new int[7];
    return (a);
}
```

אם נשתמש, שוב, בפונקציית ההדפסה Show_Array ונזמן אותה בפונקציה הראשית – אזי כל איברי המערך יהיו אפסים:

הפלט:	התוכנית הראשית:
The array is: 0 0 0 0 0 0 0 _(סמן)	<pre>static void Main(string[] args) { int[] array; array = Create_Array(); Show_Array(array); }</pre>

2. נכתוב פונקציה (השייכת לתבנית א' בפרק 6) המקבלת מספר ומחזירה מערך בגודל של המספר הזה. המערך מאוחל במספרים אקראיים בין 1 ל-10:

```
static int[] GetRandomArray(int size)
{
    int[] a = new int[size];
    int i;
    Random r = new Random();
    for (i = 0; i < size; i++)
        a[i] = r.Next(0, 10) + 1; // המספר הופך ל-1 עד 10
    return (a);
}
```

התוכנית הראשית:

```
static void Main(string[] args)
{
    int[] array;
    array = GetRandomArray(5);
    Show_Array(array);
}
```

פלט הרצה לדוגמה:

The array is: 10 8 8 7 6
- (סמן)

מערכי מונים

ניקח כדוגמה את הבעיה הבאה:
משתמש מקליד מספר באורך לא-ידוע. אנו מבקשים להדפיס את מספר הפעמים שכל אחת מהספרות 1 עד 9 הופיעה בכל מספר שנקלט.

דוגמאות:

1721

עבור הקלט:

הפלט יהיה:

הספרה 1 הופיעה פעמיים.

הספרה 2 הופיעה פעם אחת.

הספרה 7 הופיעה פעם אחת.

שימו לב

- א. כאמור, איננו יודעים מראש מהו אורכו של המספר.
- ב. נדפיס רק את כמות המופעים של ספרות הנמצאות במספר.

דרך לפתרון הבעיה

- א. נשתמש בתבנית שלמדנו בפרק 4. נסתכל מימין לשמאל על המספר שנקלט, ונקצץ אותו בכל פעם עד שיהפוך ל-0.
- ב. נגדיר מערך של מונים. כלומר, בתחילת התוכנית, כל הספרות (0 עד 9) המיוצגות על ידי מערך מאותחלות ב-0:

monim	0	1	2	3	4	5	6	7	8	9
690	0	0	0	0	0	0	0	0	0	0

690

כאמור (כמו שלמדנו בפרק 4), נסתכל על ספרת האחדות של המספר שנקלט. במקרה שלנו, עבור הערך הנקלט 1721, מדובר בספרה '1'. על כן, נמקם את הערך '1' באינדקס 1:

`monim[1]=monim[1]+1;`

או:

`monim[1]++;`

כלומר, המערך ייראה כך:

monim	0	1	2	3	4	5	6	7	8	9
690	0	1	0	0	0	0	0	0	0	0

690

בשלב זה הופיעה הספרה '1' פעם אחת בלבד. השינוי מסומן בצהוב.

בשלב הבא נקצץ את המספר (על ידי חלוקתו ב-10) והוא יהפוך ל-172. נבודד פעם נוספת את ספרת האחדות, במקרה זה – הספרה '2'. נמקם את הערך באינדקס המייצג את הספרה, והמערך ייראה עכשיו כך:

monim	0	1	2	3	4	5	6	7	8	9
690	0	1	1	0	0	0	0	0	0	0

690

לאחר קיצוץ נוסף – המספר הוא '17'. נמקם את הערך באינדקס המתאים –
`monim[7]++;`
 והמערך ייראה כך:

monim	0	1	2	3	4	5	6	7	8	9
690	0	1	1	0	0	0	0	1	0	0

690

נקצץ שוב את המספר. המספר שנותר הוא '1'. גם הפעם נמקם את הערך באינדקס המתאים
 (1):

`monim[1]++;`
 המערך ייראה כך:

monim	0	1	2	3	4	5	6	7	8	9
690	0	2	1	0	0	0	0	1	0	0

690

לאחר הקיצוץ (1 חלקי 10), ערך המספר הוא '0'. לכן, האלגוריתם הסתיים.
 כעת עלינו לעבור על כל איבר של מערך המונים, **ורק אם ערכו שונה מ-0** – נדפיס אותו.

```
for (i = 0; i < monim.Length; i++)
    if (monim[i] != 0)
        Console.WriteLine("The digit {0} appears {1} time(s)", i, monim[i]);
```

כעת נרשום את הקוד המלא בצורה מסודרת:

1. פונקציה המקבלת מספר שאורכו אינו ידוע ומחזירה מערך מונים מאותחל בהתאם לספרות המרכיבות את המספר.
2. פונקציה המקבלת מערך מונים – שאותו החזירה הפונקציה בסעיף א' – ומדפיסה את מערך המונים באינדקסים שבהם הוא שונה מ-0.
3. פונקציה ראשית, כמובן – שתזמן את שתי הפונקציות הקודמות.

```

class Program
{
    static int[] GetArrMonim(long num)
    {
        int[] monim = new int[10]; // 0 .. 9
        while (num != 0)
        {
            monim[num % 10]++;
            num = num / 10;
        }
        return (monim);
    }

    static void Print_Array(int[] monim)
    {
        int i;
        for (i = 0; i < monim.Length; i++)
        {
            if (monim[i] != 0)
                Console.WriteLine("The digit {0} appears {1}
                                   time(s).", i, monim[i]);
        }
        Console.WriteLine();
    }

    static void Main(string[] args)
    {
        int[] monim = new int[10];
        long number;
        Console.WriteLine("Please enter your number:");
        number = long.Parse(Console.ReadLine());
        monim = GetArrMonim(number);
        Print_Array(monim);
        Console.ReadKey();
    }
}

```

דוגמאות ריצה

.1

Please enter your number:

1358856

The digit 1 appears 1 time(s).

The digit 3 appears 1 time(s).

The digit 5 appears 2 time(s).

The digit 6 appears 1 time(s).

The digit 8 appears 2 time(s).

.2

Please enter your number:

19809

The digit 0 appears 1 time(s).

The digit 1 appears 1 time(s).

The digit 8 appears 1 time(s).

The digit 9 appears 2 time(s).

פרק 9 - טיפוסים מורכבים

הגדרה

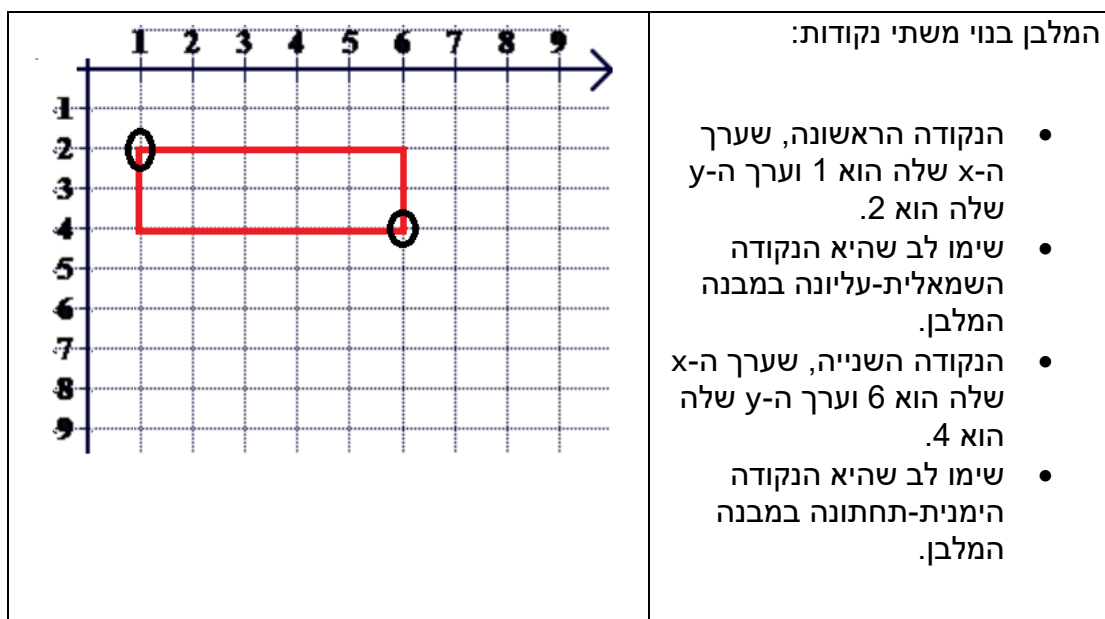
טיפוס מורכב הוא טיפוס הכולל בהגדרתו גם תכונות מטיפוס נוסף. למעשה, כל מחלקה המכילה **תכונה שאינה יחידה** ואינה מסוג מספרי (int, double וכדומה) או מסוג bool – היא **מורכבת**.

למעשה, כמעט כל הדוגמאות שביצענו עד השלב הנוכחי הן דוגמאות של מחלקות מורכבות (כולל מחלקת string, רכב, נקודה ועוד).

דוגמאות לטיפוסים מורכבים

א. מחלקת "נקודה" (Point) שעמה עבדנו, המורכבת משתי תכונות מסוג int.

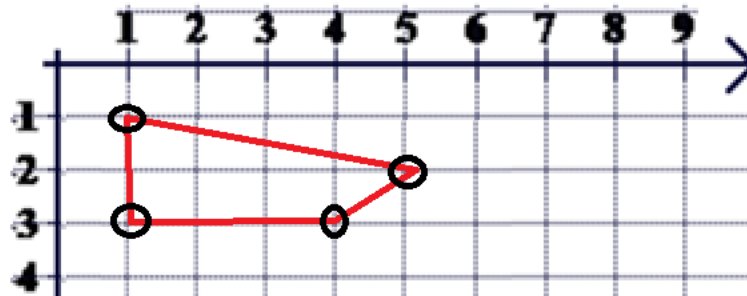
ב. מחלקת "מלבן", הבנויה משתי נקודות: נקודה שמאלית-עליונה ונקודה ימנית-תחתונה. נסתכל על המלבן הבא:



ג. מחלקת "סטודנט", הבנויה מהתכונות הבאות:

1. שם הסטודנט.
2. מערך בן חמישה ציונים (המייצגים חמישה מקצועות).
3. ממוצע ציוני הסטודנט.

ד. מחלקת "פוליגון", הבנויה ממערך של "נקודות". לדוגמה, נסתכל על הפוליגון הבנוי מהנקודות הבאות: (1,1), (5,2), (4,3), (1,3), (1,1).



הערה: הנקודה האחרונה זהה לנקודה הראשונה, כך שהפוליגון יהיה "סגור".

נסתכל על המחלקה הראשונה – מחלקת "נקודה" (Point). שימו לב: פעולת ההדפסה ToString() שונתה מעט, ואינה כוללת הדפסת שורה חדשה ("\\n").

<pre> class Point { private int x; private int y; public int GetX() { return this.x; } public void SetX(int val) { x = val; } public int GetY() { return this.y; } public void SetY(int val) { </pre>	<pre> public Point(Point Other) { x = Other.x; this.y = Other.GetY(); } public override bool Equals(Object obj) { Point PObj = obj as Point; if (PObj == null) return false; if (this == PObj) { Console.WriteLine("same Point !"); return (false); } return (x.Equals(PObj.x) && y.Equals(PObj.y)); } public override string ToString() { </pre>
---	---

<pre> this.y = val; } public Point() { this.x = 5; y = 3; } public Point(int i) : this(i, 5) { } </pre>	<pre> return string.Format("x={0} y={1}", x, y); } } </pre>
---	---

ייצוג של טיפוס מורכב

נסתכל על המחלקה השנייה – מחלקת "מלבן" (Rectangle) – הבנויה משני אובייקטים מסוג "נקודה" (Point):

<pre> class Rectangle { Point p1; // תכונה ראשונה Point p2; // תכונה שנייה public Rectangle(Point a, Point b) { this.p1 = a; this.p2 = b; } public Rectangle(Point a) : this(a, new Point(a.GetX()+4, a.GetY()+2)) { } public override string ToString() { return string.Format(" Rectangle consist of: First Point ({0}), Second Point ({1})" , p1.ToString(), p2.ToString()); </pre>	רכיבי המחלקה: כל מלבן בנוי משתי נקודות: p1 ו-p2. פונקציה בונה: הפונקציה מקבלת שתי נקודות ומייצרת מלבן. הנקודה הראשונה (a) מאתחלת את הנקודה p1, והנקודה השנייה (b) מאתחלת את הנקודה p2. פונקציה בונה: הפונקציה מקבלת נקודה (a), ומזמנת את הפונקציה הבונה (הקודמת) עם ערך x של a ועוד 4, וכן עם ערך y ועוד 2.
--	---

<pre> } } </pre>	פונקציית ההדפסה: מדפיסה את נתוני המלבן. הפונקציה מזמנת את ההדפסה של הנקודה הראשונה – <code>p1.ToString</code> – ומשרשרת לתוצאה את נתוני ההדפסה של הנקודה השנייה – <code>p2.ToString</code>.
----------------------	--

כעת נסתכל על הפעולה הראשית:

```

static void Main(string[] args)
{
    Point p1 = new Point();           // (1)
    Point p2 = new Point(6, 8);       // (2)

    Rectangle r1 = new Rectangle(p1, p2); // (3)
    Console.WriteLine(r1);           // (4)

    Rectangle r2 = new Rectangle(new Point(2,4)); // (5)
    Console.WriteLine(r2);           // (6)

    Console.ReadKey();
}

```

כעת נפרט את תהליך הביצוע, שלב אחר שלב:

```

Point p1 = new Point();           // (1)

```

בשלב הראשון נוצר אובייקט מסוג "נקודה", והוא מאוחלל בצורה **דיפולטית**. כלומר, הופעל בנאי **ללא פרמטרים**. לכן, זוהי הפונקציה שתזומן ממחלקת `Point`:

```
public Point() // Default C'tor
{
    this.x = 5;
    y = 3;
}
```

בשלב זה יש אובייקט בודד מסוג "נקודה" בכתובת 550 בזיכרון:



```
Point p2 = new Point(6, 8); // (2)
```

בשלב השני נוצר אובייקט נוסף מסוג "נקודה", והוא מאותחל בצורה **מפורשת**. כלומר, הופעל בנאי המקבל **שתי תכונות**. לכן, זוהי הפונקציה שתזומן ממחלקת Point:

```
public Point(int x, int y)
{
    this.x = x;
    this.y = y;
}
```

בשלב זה מצוי אובייקט שני מסוג "נקודה" בכתובת 620 בזיכרון:

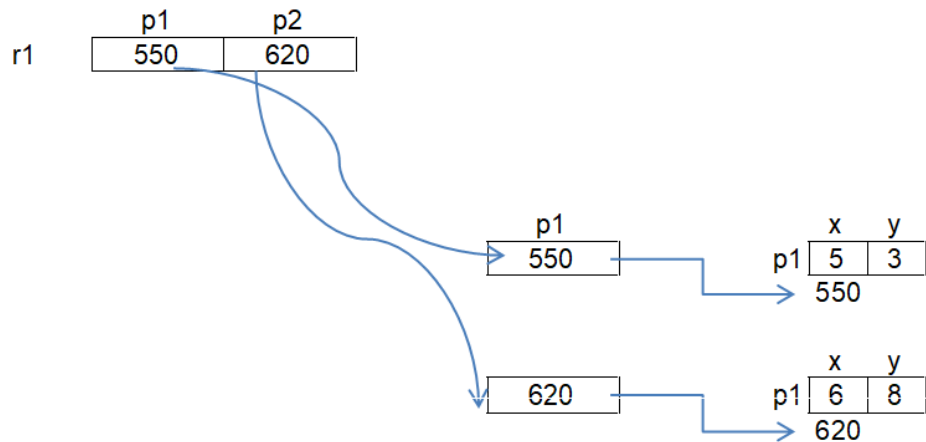


```
Rectangle r1 = new Rectangle(p1, p2); // (3)
```

בשלב השלישי מצויים בזיכרון שני אובייקטים מסוג "נקודה". כעת נוצר אובייקט מסוג "מלבן". הפונקציה הבונה של "מלבן" מקבלת את שתי הנקודות הקיימות:

```
public Rectangle(Point a, Point b)
{
    this.p1 = a;
    this.p2 = b;
}
```

למעשה, הנקודה a מצביעה לנקודה p1 והנקודה b מצביעה לנקודה p2.
 כך נראה בזיכרון האובייקט r1 מסוג "מלבן":



```
Console.WriteLine(r1);
```

```
// (4)
```

בשלב הרביעי נדפיס את נתוני r1 על בסיס הפונקציה הזו:

```
public override string ToString()
{
    return string.Format("Rectangle consist of: First Point ({0}),
                          Second Point ({1})",
                          , p1.ToString(), p2.ToString());
}
```

המחרוזת שהפונקציה הנ"ל בונה כעת היא:

```
" Rectangle consist of: First Point ("
```

בהמשך מזומנת הפונקציה ToString() של "Point" – של נקודת p1: p1.ToString().
 לכן, כעת תזומן הפונקציה הזו:

```
public override string ToString()
{
    return string.Format("x={0} y={1}", x, y);
}
```

כאמור, היא מופעלת על p1, ולכן המחרוזת שמשורשרת היא "x=5 y=3".
 בשלב הנוכחי המחרוזת היא:

```
" Rectangle consist of: First Point (x=5 y=3)"
```

בהמשך פונקציית ToString() של "מלבן" משורשר הטקסט הבא:

```
"}), Second Point ("
```

כעת, המחרוזת הכוללת היא:

```
" Rectangle consist of: First Point (x=5 y=3), Second Point ("
```

בשלב הזה מופעלת הפונקציה הבאה על הנקודה השנייה, p2:

```
public override string ToString()  
{  
    return string.Format("x={0} y={1}", x, y);  
}
```

כאמור, הפונקציה מופעלת על p2, ולכן המחרוזת שמשורשרת היא "x=6 y=8".
בסופו של דבר מדפיסה פונקציית ToString() של מחלקת "מלבן" את המחרוזת הזו:

```
" Rectangle consist of: First Point (x=5 y=3), Second Point (x=6 y=8)"
```

```
PolygonRectangle r2 = new Rectangle(new Point(2,4)); // (5)
```

בשלב החמישי נוצר באופן דומה מלבן שני, והבנאי שמופעל הוא הבנאי שמקבל **נקודה בודדת**:

```
public Rectangle(Point a) : this(a, new Point(a.GetX()+4, a.GetY()+2))  
{  
}  
}
```

כעת מפעילה הפונקציה הבונה הזו את הבנאי הקודם ומעבירה לו את הנקודות הבאות:

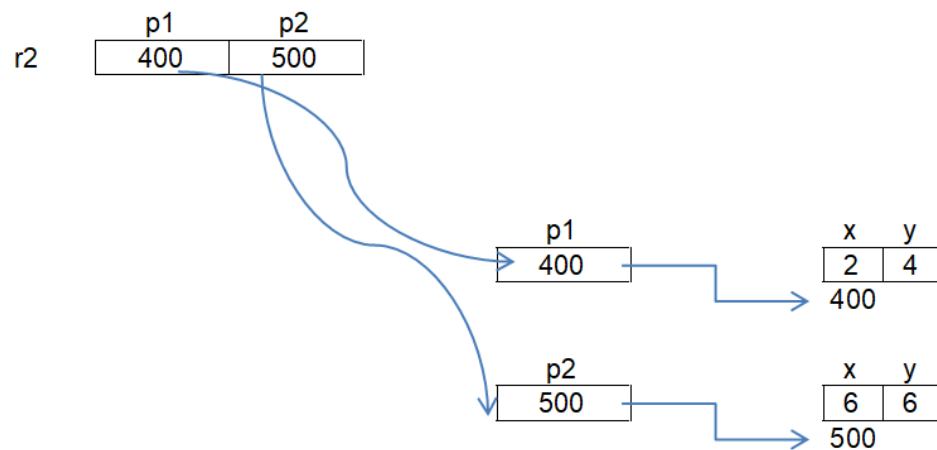
```
Rectangle((2,4),(2+4,4+2)) → Rectangle((2,4),(6,6);
```

```
Console.WriteLine(r2); // (6)
```

באופן דומה, ToString() של מחלקת Point בונה ומחזירה למסך את המחרוזת הבאה:

```
" Rectangle consist of: First Point (x=2 y=4), Second Point (x=6 y=6)"
```

תמונת הזיכרון תראה כך:



מעריך עצמים כתכונה

על בסיס מחלקת "נקודה" (Point) – נבנה את מחלקת "פוליגון" (Polygon):

```

class Polygon
{
    Point []p;
    public Polygon(Point []recv)
    {
        int i;
        p = new Point[recv.Length];
        for (i=0; i<recv.Length; i++)
            p[i]=new Point(recv[i].GetX(),recv[i].GetY());
    }

    public override string ToString()
    {
        int i;
        string pelet = "Polygon consist of: ";
        for (i=0; i<p.Length;i++)
        {
            pelet += "("+p[i].ToString()+")";
        }
        pelet += (" Total " + i + " Point(s).");
        return (pelet);
    }
}

```

כל פוליגון בנוי ממספר שונה של נקודות. זו הסיבה לכך שיש מצביע למערך נקודות – אך אין גודל מסוים שלהן:

```
Point[] p;
```

נסתכל על הפונקציה השונה של הפוליגון:

```

public Polygon(Point []recv)
{
    int i;
    p = new Point[recv.Length];
    for (i=0; i<recv.Length; i++)
        p[i]=new Point(recv[i].GetX(),recv[i].GetY());
}

```

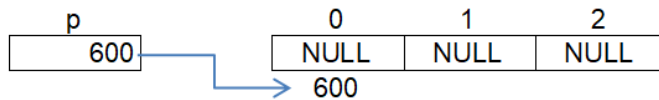
הבנאי מקבל מהמחלקה הראשית מערך של נקודות בגודל מסוים. p מאותחל בגודל של אותו מערך.

לדוגמה, אם בפונקציה הראשית (נביא דוגמאות בהמשך) הוגדר מערך בן שלושה מקומות –

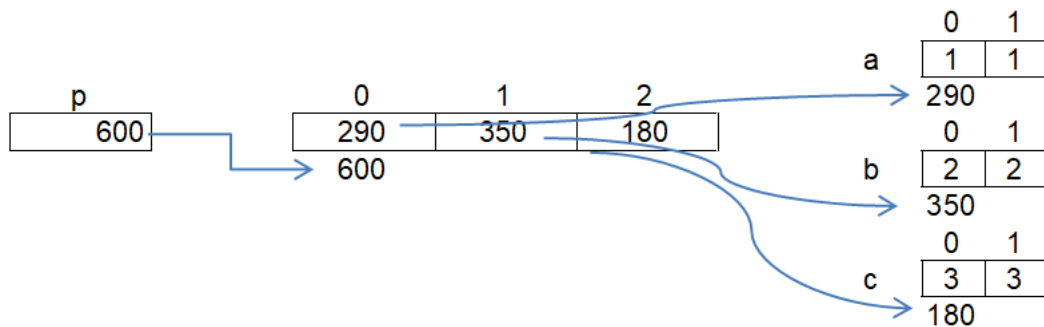
```
Point a = new Point(1, 1);
Point b = new Point(2, 2);
Point c = new Point(3, 3);
```

```
Point[] shape1 = { a, b, c };
```

אזי מערך p יראה כך:



זהו מערך של מצביעים לאובייקטים מסוג "נקודה".
לולאת ה-for מייצרת עבור כל מקום אובייקט מסוג "נקודה", והמערך מצביע עליהן:



פונקציית ההדפסה:

```
public override string ToString()
{
    int i;
    string pelet = "Polygon consist of: ";
    for (i=0; i<p.Length;i++)
    {
        pelet += (" "+p[i].ToString()+"");
    }
    pelet += (" Total " + i + " Point(s).");
    return (pelet);
}
```

עבור כל פוליגון, הפונקציה מדפיסה "הפוליגון מורכב מ:", ומשרשרת את הדפסת הנקודות – על בסיס המערך שהוצג קודם לכן.
נוסף לכך, הפונקציה משרשרת את הטקסט "סך הכול " + i + " נקודות".
כלומר, כל הדפסה מציינת את מספר הנקודות שמהן מורכב הפוליגון.

כעת נציג את התוכנית הראשית ואת פלט ההרצה:

```
static void Main(string[] args)
{
    Point a = new Point(1, 1);
    Point b = new Point(2, 2);
    Point c = new Point(3, 3);
    Point d = new Point(4, 4);
    Point[] shape1 = { a, b, c };

    Point[] shape2 = { a, b, c ,new Point(10,10),d};

    Polygon polygon1 = new Polygon(shape1);
    Console.WriteLine(polygon1);

    Polygon polygon2 = new Polygon(shape2);
    Console.WriteLine(polygon2);

    Console.ReadKey();
}
```

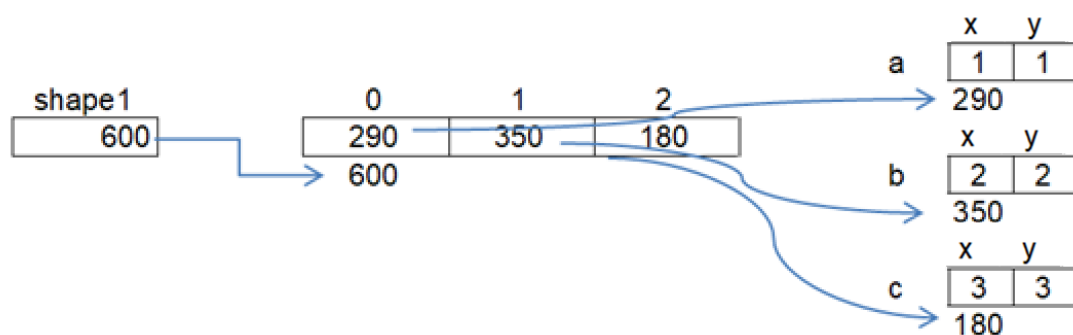
פלט ההרצה:

Polygon consist of: (x=1 , y=1)(x=2 , y=2)(x=3 , y=3) Total 3 Point(s).

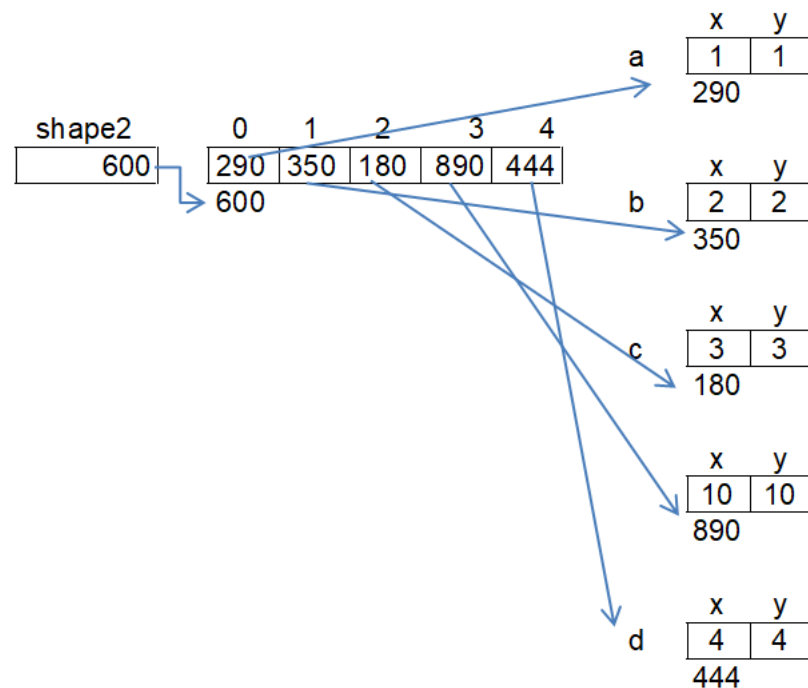
Polygon consist of: (x=1 , y=1)(x=2 , y=2)(x=3 , y=3)(x=10 , y=10)(x=4 , y=4)
Total 5 Point(s).

תמונת זיכרון

כך נראה המערך **shape1**:



כך נראה המערך shape2:



תרשים UML – Unified Modeling Language

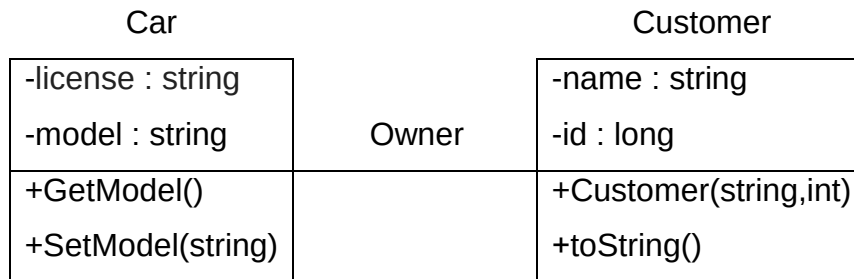
UML הוא דרך חזותית/גרפית להמחשת קשרים בין אובייקטים בתכנות מונחה עצמים. רישום תכונות:

- תכונה שההרשאה שלה היא פרטית (private) תסומן במינוס (-).
 - תכונה שההרשאה שלה היא ציבורית (public) תסומן בפלוס (+).
- רישום הפעולות ייעשה בצורה דומה – אך יהיו בהן סוגריים.

היחסים בין המחלקות: קיימים (בשלב זה) שני סוגים של יחסים בין מחלקות:

1. אסוציאציה

Owner – לקוח – הוא הבעלים של הרכב. זה הקשר בין המחלקות.



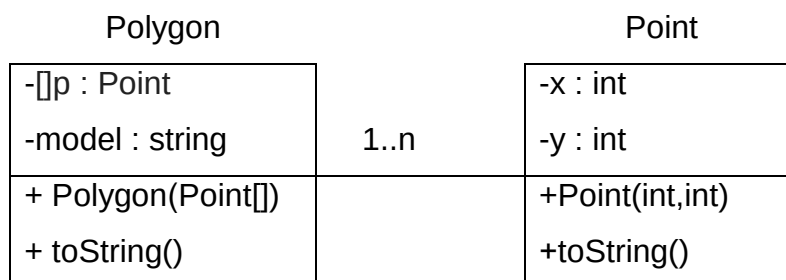
2. הכלה/הרכבה: מדובר באובייקט (מסוג מחלקה מסוימת) שאינו יכול להתקיים

באופן עצמאי.

לדוגמה, לנקודות אין "זכות קיום" אלא אם כן הן חלק מפוליגון.

היחסים יכולים להיות:

- a. 1...1: כלומר, אחד לאחד. לדוגמה, לבית אחד יש דלת כניסה אחת.
- b. 1....n או 1.....*: כלומר, אחד לרבים. לדוגמה, לרופא אחד יש מטופלים רבים.
- c. n....m: כלומר, רבים לרבים. לדוגמה, לרופא אחד יש מטופלים רבים, אבל במקביל יש למטופל אחד יותר מרופא אחד.



העמסת פעולות לטיפול בטיפוסים שונים

1. נעמים (OverLoad) את הפעולה IsBigger. הנקודה a תהיה גדולה מהנקודה b

אם שדה ה-x שלה גדול משדה ה-x של הנקודה b ושדה ה-y שלה גדול משדה ה-y של הנקודה b.

```
public static bool IsBigger(Point a, Point b)
{
    if (a.GetX() > b.GetX() && a.GetY() > b.GetY())
    {
        return true;
    }
}
```

בצורה דומה נעמיס את הפעולה הנ"ל ונבדוק האם נקודה a גדולה יותר ממערך של

נקודות:

```
public static bool IsBigger(Point a, Point []b)
{
    int i;
    for (i = 0; i < b.Length; i++)
        if (a.GetX() <= b[i].GetX() && a.GetY() <= b[i].GetY())
        {
            return false;
        }
    return true;
}

return false;
}
```

נפעיל את שתי הפונקציות שכתבנו.

נתונה הפונקציה הראשית הבאה:

```
static void Main(string[] args)
{
    Point myPoint = new Point(10, 10);
    Point other = new Point(20, 20);
    Point a = new Point(1, 1);
    Point b = new Point(2, 2);
    Point c = new Point(3, 3);
    Point []shape1 = { a, b, c };

    Console.WriteLine(Point.IsBigger(other, myPoint));
    Console.WriteLine(Point.IsBigger(other, shape1));

    Console.ReadKey();
}
```

מה יהיה הפלט עבור השורה הבאה? בואו ננתח זאת:

```
Console.WriteLine(Point.IsBigger(other, myPoint));
```

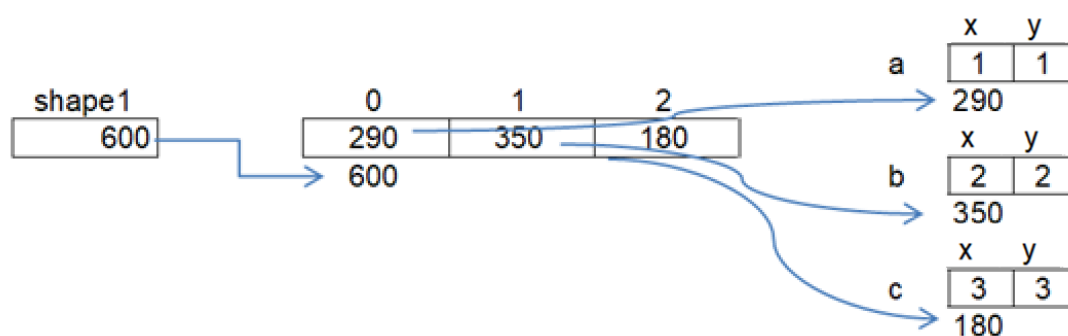
		האם				
a	<table><tr><td>x</td><td>y</td></tr><tr><td>20</td><td>10</td></tr></table> 290	x	y	20	10	?a>b
x	y					
20	10					
b	<table><tr><td>x</td><td>y</td></tr><tr><td>20</td><td>10</td></tr></table> 350	x	y	20	10	(a.x > b.x) && (a.y > b.y) → ? (20 > 10) && (20 > 10) → T && T → T אם כך, הפלט שיוצג יהיה True.
x	y					
20	10					

מה יהיה הפלט עבור השורה הבאה?

בואו ננתח זאת:

```
Console.WriteLine(Point.IsBigger(other , shape1));
```

כך נראה מערך shape1:



גם במקרה זה הפלט יהיה True, מפני שערכי הנקודה a (שבפונקציה הראשית מכונה `other`) גדולים מהערכים של כל אחת מהנקודות המכילות את מערך `arr`.

כעת נציג את הקוד המלא של המחלקה "Point":

```
class Point
{
    private int x;
    private int y;

    public int GetX()
    {
        return this.x;
    }

    public void SetX(int val)
    {
        x = val;
    }

    public int GetY()
    {
        return this.y;
    }

    public void SetY(int val)
    {
        this.y = val;
    }
}
```

```

public Point() // Default C'tor
{
    this.x = 5;
    y = 3;
}

public Point(int i) : this(i, 5) // this->From c'tor call other C'tor !
{
}

public Point(int x, int y) // C'tor מפורש
{
    this.x = x; // must write the word "this" !
    this.y = y; //
}
public Point(Point Other)
{
    x = Other.x;
    this.y = Other.GetY();
}

public override bool Equals(Object obj)
{
    Point PObj = obj as Point;
    if (PObj == null)
        return false;
    if (this == PObj)
    {
        Console.WriteLine("It is the same Point !");
        return (false);
    }
    return (x.Equals(PObj.x) && y.Equals(PObj.y));
}

public override string ToString()
{
    return string.Format("x={0} , y={1}", x, y);
}

public static bool IsBigger(Point a, Point b)
{
    if (a.GetX() > b.GetX() && a.GetY() > b.GetY())
    {
        return true;
    }
    return false;
}

public static bool IsBigger(Point a, Point []b)
{
    int i;
    for (i = 0; i < b.Length; i++)
        if (a.GetX() <= b[i].GetX() && a.GetY() <= b[i].GetY())
        {
            return false;
        }
    return true;
}
} // class Point

```

שתי הפונקציות שנכתבו הן סטטיות, כלומר, אין צורך באובייקט מסוג המחלקה על מנת להפעילן.

על מנת להפעילן יש לציין את שם **המחלקה** ואחריה נקודה, ואחר כך את שם הפונקציה.

```
Console.WriteLine(Point.IsBigger(other , myPoint));
Console.WriteLine(Point.IsBigger(other , shape1));
```

נבצע את אותה המשימה – אך הפעם נכתוב פונקציות סטטיות. כלומר, כאלה שחייב להיות בהן אובייקט שיפעיל את הפונקציה:
כתיבת הפונקציות:

```
public bool IsBigger(Point b)
{
    if (this.GetX() > b.GetX() && this.GetY() > b.GetY())
    {
        return true;
    }
    return false;
}

public bool IsBigger(Point []b)
{
    int i;
    for (i = 0; i < b.Length; i++)
        if (this.GetX() <= b[i].GetX() && this.GetY() <= b[i].GetY())
        {
            return false;
        }
    return true;
}
```

הפעלת הפונקציות מהתוכנית הראשית:

```
static void Main(string[] args)
{
    Point myPoint = new Point(10, 10);
    Point other = new Point(20, 20);
    Point a = new Point(1, 1);
    Point b = new Point(2, 2);
    Point c = new Point(3, 3);
    Point []arr = { a, b, c };

    Console.WriteLine(other.IsBigger(myPoint));
    Console.WriteLine(other.IsBigger(arr));

    Console.ReadKey();
}
```

האובייקט other (ולא שם המחלקה כפי שהיה בדרך א') הוא זה שמפעיל את הפונקציות השונות.

הפלט הוא אותו פלט.

כעת נוסיף למחלקה פונקציה המבצעת מיון בועות על מערך נקודות:

```
public static void sort(Point[] array) // Bubble Sort
{
    Point temp = new Point(0, 0);
    for (int i = 0; i < array.Length; i++)
    {
        for (int j = 0; j < array.Length - 1; j++)
        {
            if (array[j].IsBigger(array[j + 1]))
            {
                temp = array[j + 1];
                array[j + 1] = array[j];
                array[j] = temp;
            }
        }
    }
}
```

שלב הזה נקבל את ביצוע פעולת המיון כעבודה נתונה. בהמשך, בפרק "רשות", יופיע הסבר מפורט בנושא מיון בועות.

נציג את פונקציית ה-Main, ובה נדפיס את מערך הנקודות לפני ואחרי המיון:


```
static void Main(string[] args)
{
    Point myPoint = new Point(10, 10);
    Point other   = new Point(20, 20);
    Point a = new Point(1, 1);
    Point b = new Point(2, 2);
    Point c = new Point(3, 3);
    Point []arr = { c, b, a };

    Console.WriteLine("Before Sorting");
    foreach (var item in arr)
    {
        Console.WriteLine(item.ToString());
    }
    Point.sort(arr); // הפעלת המיון על מערך הנקודות
    Console.WriteLine("After Sorting");
    foreach (var item in arr)
    {
        Console.WriteLine(item.ToString());
    }

    Console.ReadKey();
}
```

ובהתאמה, נציג את פלט ההרצה:

```
Before Sorting
x=3 , y=3
x=2 , y=2
x=1 , y=1
After Sorting
x=1 , y=1
x=2 , y=2
x=3 , y=3
```

פרק 10 - רשימה

עם כל יתרונותיו של המערך בביצוע פעולות שונות עם משתנים/אובייקטים, יש בשימוש בו גם כמה חסרונות:

- איבריו של המערך ממוקמים בזיכרון בצורה רציפה. פעמים רבות יש צורך לאפשר לאיברים נוספים להיכנס למערך, ואז נאלצת מערכת ההפעלה להזיז את כל איברי המערך לכתובות אחרות.
- לעומת זאת, במקרים מסוימים נוצרים במערך "חורים" בעקבות שליפה של איברים ו/או כתוצאה מעדכונים שנעשו בדרך.
- מערך נוצר בגודל מסוים (לדוגמה, `int []a=new int[5]`) – ואי אפשר לשנותו. אבל לא תמיד אנו יודעים מראש לכמה איברים נזדקק כדי לפתור בעיה אלגוריתמית מסוימת.

לסיכום, אם אנו מייצגים את הנתונים באמצעות מערך, עלינו להעתיק מחדש את המערך כולו בכל פעם שנצטרך להוסיף איבר, ו"לצופף" את המערך כולו כשנצטרך למחוק פריט כלשהו מאמצע המערך.

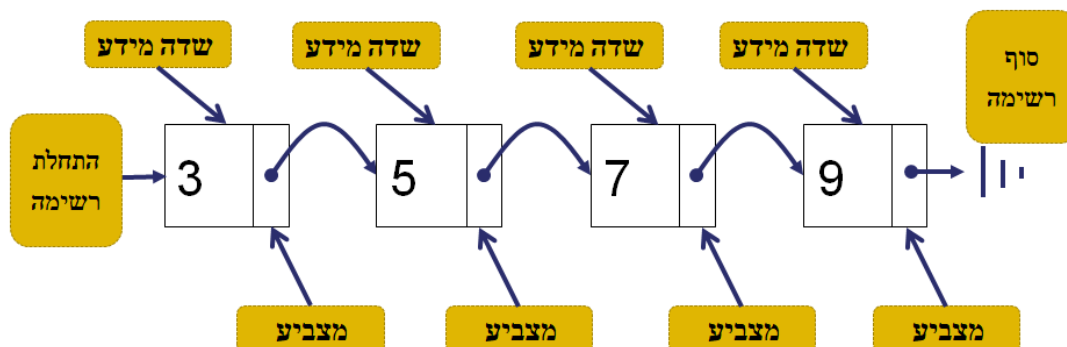
לכן, במקרים מסוימים עדיף להשתמש ברשימה.

הגדרה ⓘ

רשימה מקושרת היא רצף לוגי של משתנים מסוג המחלקה/הטיפוס הנתון. כל איבר ברשימה בנוי משני שדות:

השדה הראשון – המכונה "value" – מחזיק את ערך האיבר (מסוג מסוים).

השדה השני – המכונה "מצביע" – מחזיק את כתובתו של האיבר הבא (אם קיים כזה).



כל איבר ברשימה הזו נמצא בכתובת מסוימת. אך בשונה ממערך, כאן האיברים אינם סמוכים זה לזה.

מאפייניה של רשימה

1. רשימה מקושרת היא מבנה נתונים דינמי אשר יכול להתרחב לפי דרישת המנגנון שאותו הוא אמור לשרת.
2. אנו נדרשים להוסיף איברים במהלך ריצת התוכנית.
3. רשימות מקושרות יכולות – ואף נועדו – להתרחב ולהצטמצם באותה מידה בכל עת לפי הצורך.
4. מעצם טבעה של רשימה (שיש בה תוספות ומחיקות), מספר האיברים שלה אינו ידוע מראש.
5. התא האחרון ברשימה מצביע תמיד על סוף הרשימה, המצוין תמיד באמצעות הערך NULL ("סוף רשימה").

יתרונותיה של רשימה

1. אפשר להקצות באופן דינמי מספר רב של נתונים.
2. הוספה ו/או מחיקה של נתונים אינה מחייבת הזזת איברים בזיכרון.
3. הנתונים אינם מסודרים בזיכרון ברצף, ולכן הקצאת הזיכרון בטוחה יותר כי בכל פעם מקצים איבר בודד.
4. רשימה (List) היא חלק מאוסף (Collection) ב-#C, ומעצם הגדרתה זו היא מקבלת תכונות ופעולות נוספות שאיננו צריכים להגדיר. פעולות אלה יחסכו לנו זמן רב בהמשך.

אופן כתיבת רשימה

בדומה למערך – רשימה יכולה להיות מסוג מסוים. כלומר, ההגדרה הראשונית היא `List<T>`. T מציין איבר מסוג `Template`, כלומר, אפשר להעביר את סוג הרשימה כפרמטר (בדומה לפונקציה). נזדקק לתוספת של עבודה עם אוספים: ()

```
using System.Collections.Generic;
```

יצירת רשימה

שימו לב שכאן <> מוגדר סוג הרשימה. בדוגמה הבאה, שם הרשימה הוא grades והסוג שלה הוא מספר שלם (int).

```
List<int> grades = new List<int>();
```

תכונות של רשימה

לרשימה, מכל סוג שהוא, יש שתי תכונות:

1. מספר האיברים ברשימה – Count.
2. קיבולת מקסימלית אפשרית – Capacity.

מתודות של רשימה

1. add – מוסיפה איבר לסוף הרשימה.
2. insert – מוסיפה איבר במקום מסוים ברשימה.
3. Remove – מסירה מהרשימה איבר מסוים.
4. RemoveAt – מסירה איבר במקום מסוים ברשימה.
5. Sort – ממיינת את הרשימה.
6. Reverse – הופכת את סדר האיברים של הרשימה.
7. IndexOf – מקבלת אינדקס ומחזירה את האיבר במקום המבוקש (או -1).
8. BinarySearch – מקבלת רשימה ממוינת ומחזירה את המיקום של האיבר המבוקש (או מספר שלילי המציין כי האיבר לא נמצא).

דוגמאות לשימוש ברשימה (List)

1.

קוד מקור	פלט והסברים
<pre>static void Main(string[] args) { List<int> lst1 = new List<int>(); int i; lst1.Add(4); lst1.Add(12); lst1.Add(8); lst1.Add(6); }</pre>	מטרה: יצירת רשימה, הכנסת איברים והצגת דרך א' להדפסת רשימה. בשלב הראשון נוצרת רשימה ריקה (כלומר, ללא איברים) בשם lst1. לאחר הוספת איברים, הרשימה נראית

<pre> lst1.Add(10); for (i=0; i< lst1.Count; i++) Console.WriteLine(lst1[i]+" "); Console.WriteLine(); Console.ReadKey(); } </pre>	כך: Lst1->4->12->8->6->10 לכן, הפלט של התוכנית הזו ייראה כך: 4 12 8 6 10 שימו לב שבגוף הלולאה הפקודה היא Write, ולכן אין ירידת שורה.
---	--

2.

קוד מקור	פלט והסברים
<pre> static void Main(string[] args) { List<int> lst1 = new List<int>(); int i; lst1.Add(4); lst1.Add(12); lst1.Add(8); lst1.Add(6); lst1.Add(10); Console.WriteLine("Capacity: {0}", lst1.Capacity); Console.WriteLine("Count: {0}", lst1.Count); Console.ReadKey(); } </pre>	מטרה: הכרת תכונות של רשימה. לאחר יצירת הרשימה נדפיס את ערכי התכונות: Capacity: 8 Count: 5 כרגע, הקיבולת (Capacity) של הרשימה היא עד 8 איברים, וכמות האיברים הנוכחית (Count) היא 5.

3.

קוד מקור	פלט והסברים
<pre>static void Main(string[] args) { List<int> lst1 = new List<int>(); int i; lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); lst1.Insert(2,99); lst1.Insert(0, 14); lst1.Insert(4, 33); lst1.Insert(1, 57); Console.WriteLine(); for (i = 0; i < lst1.Count; i++) Console.Write(lst1[i] + " "); Console.WriteLine("\nCapacity: {0}", lst1.Capacity); Console.WriteLine("Count: {0}", lst1.Count); Console.ReadKey(); }</pre>	מטרה: הדגמת הפונקציה Insert ועבודה עם תכונות הרשימה. זוהי הרשימה המקורית: Lst1->4->12->8->6->10 בפקודה <code>lst1.Insert(2,99);</code> – אנו מוסיפים את האיבר 99 במקום השני. כלומר, הרשימה נראית עכשיו כך: Lst1->4->12->99->8->6->10 לאחר מכן, בפקודה הבאה – <code>lst1.Insert(0, 14);</code> – אנו מוסיפים את האיבר 14 לראש הרשימה (המקום הראשון): Lst1->14->4->12->99->8->6->10 לאחר ההוספה השלישית, הרשימה תיראה כך: Lst1->14->4->12->99->33->8->6->10 ולאחר ההוספה האחרונה היא תיראה כך: Lst1->14->57->4->12->99->33->8->6->10 נשים לב להדפסות:

	Capacity: 16 Count: 9 כרגע, הקיבולת (Capacity) של הרשימה היא עד 16 איברים. הגענו לגבול העליון הקודם (8), ולכן הקיבולת השתנתה. ואכן, כמות האיברים הנוכחית (Count) היא 9.
--	---

4.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12);</pre>	מטרה: יצירת רשימה, הכנסת איברים והצגת דרך ב' להדפסת רשימה (באמצעות foreach). זוהי הרשימה המקורית: Lst1->4->12->8->6->10 בפקודה <code>lst1.Insert(2,99);</code> אנו מוסיפים את האיבר 99 במקום השני. כלומר, עכשיו הרשימה נראית כך: Lst1->4->12->99->8->6->10 לאחר מכן, בפקודה הבאה – <code>lst1.Insert(0, 14);</code> אנו מוסיפים את האיבר 14 לראש הרשימה (המקום הראשון): Lst1->14->4->12->99->8->6->10

<pre> lst1.Add(8);lst1.Add(6); lst1.Add(10); lst1.Insert(2,99); lst1.Insert(0, 14); lst1.Insert(4, 33); lst1.Insert(1, 57); showList(lst1); // זימון הפונקציה Console.ReadKey(); } } </pre>	לאחר ההוספה השלישית, הרשימה תיראה כך: Lst1->14->4->12->99->33->8->6->10 ולאחר ההוספה האחרונה היא תיראה כך: Lst1->14->57->4->12->99->33->8->6->10 נשים לב להדפסות: Capacity: 16 Count: 9 כרגע, הקיבולת (Capacity) של הרשימה היא עד 16 איברים. הגענו לגבול העליון הקודם (8), ולכן הקיבולת השתנתה. ואכן, כמות האיברים הנוכחית (Count) היא 9.
---	--

5.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); showList(lst1); lst1.Remove(6); // יימחק האיבר 6 showList(lst1); lst1.RemoveAt(2); // יימחק האיבר 8 showList(lst1); Console.ReadKey(); }</pre>	מטרה: הדגמת הפונקציות Remove ו-RemoveAt. זוהי הרשימה המקורית: Lst1->4->12->8->6->10 לאחר הסרת האיבר '6' – Lst1->4->12->8->10 הרשימה תיראה כך: Lst1->4->12->8->10 לאחר הסרת האיבר שמיקומו 2 (האיבר 8) – הרשימה תיראה כך: Lst1->4->12->10 פלט התוכנית: Capacity: 8 Count: 5 List: 4 12 8 6 10 Capacity: 8 Count: 4 List: 4 12 8 10 Capacity: 8 Count: 3 List: 4 12 10

6.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); showList(lst1); lst1.Sort(); showList(lst1); Console.ReadKey(); }</pre>	מטרה: הדגמת הפונקציה Sort. זוהי הרשימה המקורית: Lst1->4->12->8->6->10 לאחר המיון – lst1.Sort(); הרשימה תיראה כך: Lst1->4->6->8->10->12 פלט התוכנית: Capacity: 8 Count: 5 List: 4 12 8 6 10 Capacity: 8 Count: 5 List: 4 6 8 10 12

7.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); lst1.Sort(); showList(lst1); Console.WriteLine(lst1.BinarySearch h(7)); Console.WriteLine(lst1.BinarySearch h(8)); Console.ReadKey(); }</pre>	מטרה: הדגמת הפונקציה BinarySearch. זוהי הרשימה המקורית: Lst1->4->12->8->6->10 לאחר המיון – lst1.Sort(); הרשימה תיראה כך: Lst1->4->6->8->10->12 הפונקציה "חיפוש בינארי" (BinarySearch) עובדת על רשימה ממוינת בלבד. הפקודה Console.WriteLine(lst1.BinarySearch (7)); תדפיס ערך שלילי, שמשמעותו היא שהפעולה לא הצליחה. הפעולה לא הצליחה מכיוון שאין ברשימה איבר שערכו '7'. לעומת זאת, הפקודה Console.WriteLine(lst1.BinarySearch(8)); תדפיס את הערך 2 מכיוון שזהו האינדקס של האיבר שחיפשנו: Lst1->4->6->8->10->12

<pre>} </pre>	פלט התוכנית: Capacity: 8 Count: 5 List: 4 6 8 10 12 -3 2
---------------	--

8.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); </pre>	מטרה: הדגמת הפונקציה Reverse. זוהי הרשימה המקורית: Lst1->4->12->8->6->10 לאחר הפעלת הפונקציה lst1.Reverse (); הרשימה תיראה כך: Lst1->10->6->8->12->4 פלט התוכנית: Capacity: 8 Count: 5 List: 4 12 8 6 10 Capacity: 8 Count: 5

<pre>showList(lst1); lst1.Reverse(); showList(lst1); Console.ReadKey(); }</pre>	<pre>List: 10 6 8 12 4</pre>
---	------------------------------

9.

קוד מקור	פלט והסברים
<pre>static void showList(List<int> ls) { Console.WriteLine("Capacity: {0}", ls.Capacity); Console.WriteLine("Count: {0}", ls.Count); Console.WriteLine("List:"); foreach (int item in ls) { Console.Write(item + " "); } Console.WriteLine(); } static void Main(string[] args) { List<int> lst1 = new List<int>(); lst1.Add(4);lst1.Add(12); lst1.Add(8);lst1.Add(6); lst1.Add(10); showList(lst1); Console.WriteLine(lst1.IndexOf(6)); }</pre>	מטרה: הדגמת הפונקציה IndexOf : זוהי הרשימה המקורית: Lst1->4->12->8->6->10 לאחר הפעלת הפונקציה Console.WriteLine(lst1.IndexOf(6)); – הפונקציה תדפיס 3, כי זהו המיקום של האיבר '6'. לאחר הפעלת הפונקציה Console.WriteLine(lst1.IndexOf(9)); הפונקציה תדפיס 1 – כי אין ברשימה איבר כזה. פלט התוכנית: Capacity: 8 Count: 5 List: 4 12 8 6 10

Console.WriteLine(lst1.IndexOf(9));	3
Console.ReadKey();	-1
}	

פעולות על סיביות (BiteWise)

שפת #C, בדומה לשפות אחרות, מאפשרת לנו לבצע פעולות לוגיות ברמה של סיביות.

נסתכל על הפעולות הבאות:

1. קשר לוגי – NOT (מסומן על ידי ~):

<pre>class Program { static void Main(string[] args) { int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = ~a; /* -61 = 1100 0011 */ Console.WriteLine("Value of c is {0}", c); Console.ReadKey(); } // Main function }</pre>	הפלט של התוכנית הזו הוא -61. מדוע? ננתח זאת:
--	---

עברנו לבסיס בינארי, ולכן החזקות בבסיס זה יהיו (2^n) :

ננתח את ערכו של משתנה a:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	128	64	32	16	8	4	2	1	ערך:
a	0	0	1	1	1	1	0	0	

טבלת אמת של ~ (NOT) – קדימות מספר 1:

X	!(X)
0	1
1	0

ננתח את משמעות הפקודה הבאה:

$$c = \sim a;$$

נהפוך	את	תוכן	הסיביות	ב-a:					
	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	
ערך:	1	2	4	8	16	32	64	128	
	0	0	1	1	1	1	0	0	a

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	128	64	32	16	8	4	2	1	ערך:
$\sim a$	1	1	0	0	0	0	1	1	

↓ נבצע משלים ל-2:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	1	1	1	0	1
3				D			

עברנו לבסיס הקסדצימלי (Hexadecimal), ולכן החזקות בבסיס זה יהיו 16:

$$3D \rightarrow D \times 16^0 + 3 \cdot 16^1 \Rightarrow 13 \cdot 1 + 13 \cdot 16 \Rightarrow 13 + 38 \Rightarrow 61$$

ביט הסימן (משמאל) דלוק, ולכן ערכו של c הוא 61-.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-------	-------	-------	-------	-------	-------	-------	-------

ערך:

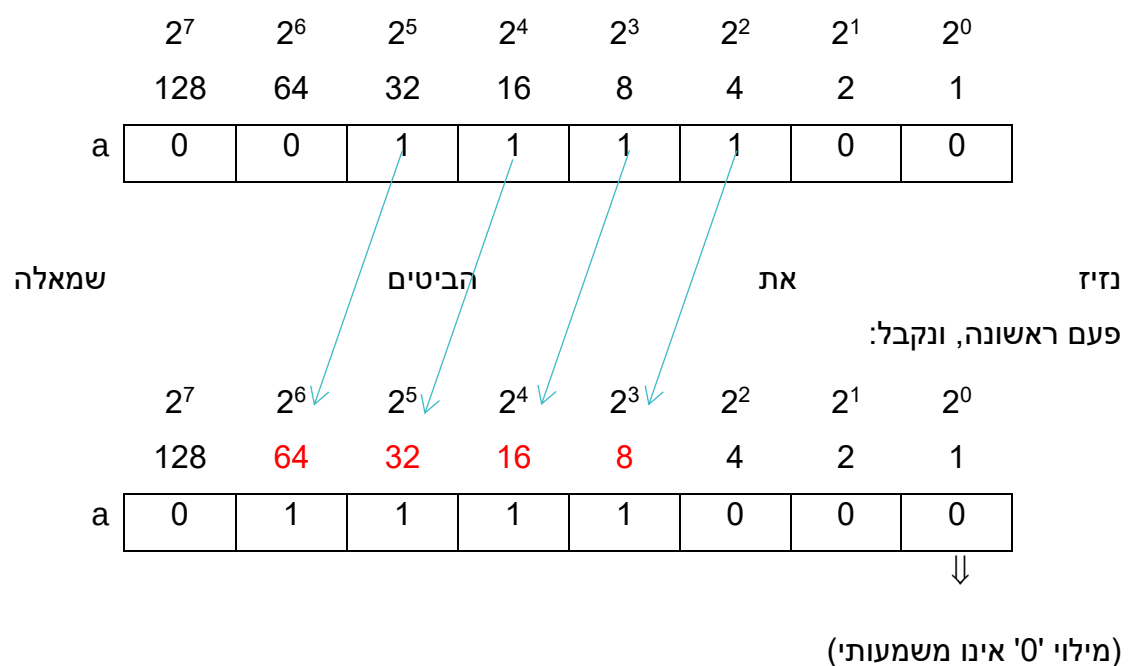
	128	64	32	16	8	4	2	1
a	0	0	1	1	1	1	0	0

2. קשר לוגי – ShiftLeft (מסומן על ידי <<):

<pre> class Program { static void Main(string[] args) { int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = a << 2; /* 240 = 1111 0000 */ Console.WriteLine("Value of c is {0}", c); Console.ReadKey(); } // Main function } </pre>	הפלט של התוכנית הזו הוא 240. מדוע? ננתח זאת:
--	---

הפקודה שביצענו כאן היא $c = a \cdot 2^2$. כלומר, שתי הזזות שמאלה. למעשה, הפקודה שביצענו שקולה לפקודה $a = a \cdot 4$. כל הזזה של הסיביות שמאלה מכפילה את המספר ב-2.

ערכו של a בהתחלה:



כעת, ערכו של a הוא $120 = 64 + 32 + 16 + 8$. תוצאה זו מתאימה ללוגיקה, כי הערך המקורי של a – שהיה 60 – אכן הוכפל ב-2.

נזיז את הביטים שמאלה פעם שנייה, ונקבל:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
a	1	1	1	1	0	0	0	0

↓

(מילוי 0 אינו משמעותי)

לאחר ההזזה השנייה ערכו של a הוא $240 = 128 + 64 + 32 + 16$.
אכן, עבור שתי הזזות שמאלה – ערכו המקורי של a (60) הוכפל ב-4.

3. קשר לוגי – ShiftLeft (מסומן על ידי >>):

<pre>class Program { static void Main(string[] args) { int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = a >> 2; /* 15 = 0000 1111 */ Console.WriteLine("Value of c is {0}", c); Console.ReadKey(); } // Main function }</pre>	הפלט של התוכנית הזו הוא 15. מדוע? ננתח זאת:
--	---

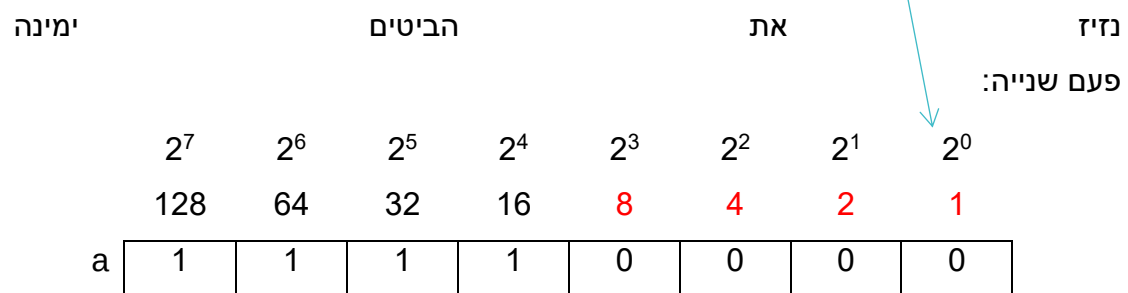
כאן ביצענו את הפקודה $c = a/2^2$ – כלומר, שתי הזזות שמאלה. למעשה, הפקודה הזו שקולה לפקודה $a = a/4$. כל הזזה של הסיביות שמאלה מחלקת את המספר ב-2.

ערכו של a בהתחלה:



	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
a	0	0	0	1	1	1	1	0

כעת ערכו של a הוא $2 + 4 + 8 + 16 = 30$. תוצאה זו מתאימה ללוגיקה, כי הערך המקורי של a – שהיה 60 – התחלק ב-2.



לאחר ההזזה השנייה, ערכו של a הוא $1 + 2 + 4 + 8 = 15$. ואכן, עבור שתי הזזות ימינה – ערכו המקורי של a (60) התחלק ב-4.

4. קשר לוגי – BitWise OR (מסומן על ידי |):

```
class Program
{
    static void Main(string[] args)
    {
        int a = 60;    /* 60 = 0011 1100 */
        int b = 13;    /* 13 = 0000 1101 */
        int c = 0;
        c = a | b;
```

טבלת אמת של OR:

a	b	a b
0	0	0
1	0	1
0	1	1
1	1	1

<pre> Console.WriteLine("Value of c is {0}", c); Console.ReadKey(); } // Main function </pre>	
---	--

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
a	0	0	1	1	1	1	0	0

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
b	0	0	0	0	1	1	0	1



כך ייראה הרגיסטר c לאחר פעולת ה-OR:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
c	0	0	1	1	1	1	0	1



שימו לב שערכו של c הוא $1 + 4 + 8 + 16 + 32 = 61$.

5. קשר לוגי – BitWise XOR (מסומן על ידי ^):

```
class Program
{
    static void Main(string[] args)
    {
        int a = 60;    /* 60 = 0011 1100 */
        int b = 13;    /* 13 = 0000 1101 */
        int c = 0;
        c = a ^ b;

        Console.WriteLine("Value of c is {0}",
c);
        Console.ReadKey();
    }
}
```

טבלת אמת של XOR:

a	b	a ^ b
0	0	0
1	1	1
1	0	1
0	1	1

```
} // Main function
}
```

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
a	0	0	1	1	1	1	0	0

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
b	0	0	0	0	1	1	0	1



כך יראה הרגיסטר c לאחר פעולת ה-XOR:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
c	0	0	1	1	0	0	0	1



שימו לב שערכו של c הוא $1 + 16 + 32 = 49$.

6. קשר לוגי – BitWise AND (מסומן על ידי &):

<pre>class Program { static void Main(string[] args) { int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = a & b; Console.WriteLine("Value of c is {0}", c); Console.ReadKey(); } // Main function }</pre>	טבלת אמת של AND: <table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>b & a</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	a	b	b & a	1	1	1	0	0	0	1	0	0	0	1	0
a	b	b & a														
1	1	1														
0	0	0														
1	0	0														
0	1	0														

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
a	0	0	1	1	1	1	0	0

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
b	0	0	0	0	1	1	0	1

↓

כך ייראה הרגיסטר c לאחר פעולת ה-AND:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
c	0	0	0	0	1	1	0	0

↓

שימו לב שערכו של c הוא $4 + 8 = 12$.

7. פעולות מיסוך

המטרה היא קריאת הערך של סיבית מסוימת מתוך מילה.

לדוגמה, אנו מבקשים לבדוק האם ערכו של הביט החמישי במספר $0x75$ (0x) – קידומת הקסדצימלית) הוא 1.

מהלך הפעולה:

נשתמש בפעולת AND למיסוך ביטים. את הביטים שנרצה לבודד נכפול ב-'1', ואילו את הביטים האחרים נכפול ב-'0'. אם התוצאה שהתקבלה לאחר הבידוד היא '1' – המשמעות היא שהביט שבודדנו הוא '1'; אחרת הוא '0'.

נסתכל על הערך $0x75$:

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
b	0	1	1	1		0	1	0	1

$$4 + 2 + 1 = 7$$

$$4 + 1 = 5$$

מאחר שעברנו לבסיס 16, אנו מבודדים כל ספרה בנפרד. אנו מבקשים לבדוק אם ערכו של הביט החמישי מימין (המסומן בצהוב) הוא '1'. לכן, הערך המתאים לבדיקה הוא '0' (בספרה הימנית) ו-'1' (בספרה השמאלית): 0x10:

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
b	0	0	0	1		0	0	0	0

כעת נבצע את הבדיקה: if (x & 01) שונה מ-0:

הערך 0x75:

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
b	0	1	1	1		0	1	0	1

הערך 0x10:

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
b	0	0	0	1		0	0	0	0

התוצאה המתקבלת:

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
b	0	0	0	1		0	0	0	0

אכן, הביט החמישי במספר 0x75 הוא '1'.

דוגמת קוד:

```
class Program
{
    static void Main(string[] args)
    {
        int x = 0x75;           /* 0111 0101 */
                                /* 7(10) 5(10) */

        int check = 0x10;
        int res = x & check;

        if (res==check)
            Console.WriteLine("The 5th bit (from the right) is 1");
        else
            Console.WriteLine("The 5th bit (from the right) is 0");
        Console.ReadLine();
    }
}
```

8. פעולת כיבוי ביטים

"כיבוי" (הפיכה ל-'0') הביט השמיני (מימין) במספר 0x0397 (num).

מהלך הפעולה: נשתמש בפעולת AND למיסוך ביטים, בדיוק כפי שעשינו בדוגמה הקודמת – אלא שהפעם יישמר ערך המילה החדשה. כלומר, "נדרוס" את הערך הישן.

ננתח את ערכו של num:

`int num = 0x0397;`

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
0	0	0	0	0	0	1	1	1	0	0	1	0	1	1	1
						2	1	8			1		4	2	1

כעת, על מנת "לכבות" את הביט השמיני מימין (מסומן באדום) נכפול אותו ב-'0', ואילו את השאר נכפול ב-'1':

`int check = 0xff7f;`

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0

1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

תוצאת הפעולה:

num

0	0	0	0	0	0	1	1	1	0	0	1	0	1	1	1

&

check

1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

היא:

0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

שימו לב שהתוצאה זהה למילה הראשונה, למעט ביט מספר 8 (בצהוב) שהתאפס (או "כובה").

ערך num **לאחר השינוי** הוא 0317 בבסיס הקסדצימלי – זהה ל-791 בבסיס 10 (שהוא ערך ההדפסה בתוכנית).

דוגמת קוד	הפלט
<pre>namespace Chap10 { class Program { static void Main(string[] args) { int num = 0x0397; int check = 0xff7f; num = num & check; Console.WriteLine("num=" + num); Console.ReadLine(); } } }</pre>	num=791

9. בדיקה האם המספר זוגי

א. אם המספר זוגי – אזי הביט הראשון (מימין) הוא '0'. נבצע AND עם הערך '1'.

נסתכל על הערך '60' (בבסיס 2):

הערך 0x3c (בבסיס 16):

	2^3	2^2	2^1	2^0		2^3	2^2	2^1	2^0
	8	4	2	1		8	4	2	1
Num	0	0	1	1		1	1	0	0

$$3C \rightarrow C \cdot 16^0 + 3 \cdot 16^1 \rightarrow 12 \cdot 1 + 3 \cdot 16 \rightarrow 12 + 48 \rightarrow 60 \text{ (בבסיס 10)}$$

דוגמת קוד	הפלט
<pre> namespace Chap10 { class Program { static void Main(string[] args) { int num = 0x3c; // 0011 1100(base 2) <=> 60(base 10) */ int check = 0x1; int zug1 = num & check; if (zug1==0) Console.WriteLine(num+ " zug1"); else Console.WriteLine(num+" e-zug1"); Console.ReadLine(); } } } </pre>	60 zug1

ב. כאמור, אם המספר זוגי – אזי הביט הראשון (מימין) הוא '0'. נבצע AND עם הערך '1'.

נסתכל על הערך '61' (בבסיס 2) שאינו זוגי:

הערך 0x3d (בבסיס 16):

	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
	8	4	2	1	8	4	2	1
Num	0	0	1	1	1	1	0	1

(בבסיס 10) $3D \rightarrow D \cdot 16^0 + 3 \cdot 16^1 \rightarrow 13 \cdot 1 + 3 \cdot 16 \rightarrow 13 + 48 \rightarrow 61$

פלט	דוגמת קוד
61 e-zugi	<pre> namespace Chap10 { class Program { static void Main(string[] args) { int num = 0x3d; // 0011 1101(base 2) // 61(base 10) int check = 0x1; int zug1 = num & check; if (zug1==0) Console.WriteLine(num+ " zug1"); else Console.WriteLine(num+" e-zug1"); Console.ReadLine(); } } } </pre>

פרק 11 - מטריצות, חיפושים ומיונים

מטריצות

מערך דו-ממדי הוא מערך המורכב ממספר מסוים של שורות ומספר מסוים של עמודות, במבנה של טבלה. כל האיברים במערך חייבים להיות מאותו סוג.

פנייה לתא במערך דו-ממדי

כדי לפנות אל תא במערך, עלינו לציין את מספר השורה ואת מספר העמודה. נמחיש זאת על ידי אחסון ציוני בחינות הבגרות של שלושה תלמידים בארבעה מקצועות:

```
int [,]grades = new int[3,4]
{
    {80,85,90,97},
    {75,80,85,90},
    {85,95,70,66}
```

הגדרנו כאן מערך דו-ממדי בעל שלוש שורות וארבע עמודות. כל שורה מציינת תלמיד, וכל עמודה מציינת מקצוע. שמות התלמידים הם יוסי, אורי ומשה. כלומר, שורה 0 היא מערך חד-ממדי המכיל את ציוני הבחינות של יוסי, וכך הלאה. ועמודה 0 היא המקצוע הראשון שבו נבחנות שלושת התלמידים, וכך הלאה.

	עמודה 0	עמודה 1	עמודה 2	עמודה 3
שורה 0 יוסי	80	85	90	97
שורה 1 אורי	75	80	85	90
שורה 2 משה	85	95	70	66

הפנייה לאיברים השונים במערך מתבצעת על ידי ציון מספר השורה ואחריו מספר העמודה. כך:

```
grades [0][3]=97;
```

זה הציון של יוסי במקצוע הרביעי (באינדקס/עמודה 3).

המטריצה מופיעה בזיכרון ברצף, כלומר, היא אינה מופיעה באמת בצורה של טבלה. כאשר אנו ניגשים לתא מסוים במטריצה, מערכת ההפעלה מחשבת את המקום בזיכרון לפי הנוסחה הבאה:

מספר עמודות בשורה i * j --> מספר עמודות בשורה i * j מביא אותנו לתחילת תת-המערך הרצוי בזיכרון, והוספת j מביא אותנו לתא המתאים בתת-המערך.

לדוגמה, במקרה של מערך grades:

פנייה לתא grades[2][3] משמעותו פנייה לתא $3 + 4 * 2$ --> תא 11 בזיכרון ביחס לתחילת המערך הדו-ממדי בזיכרון.

	עמודה 0	עמודה 1	עמודה 2	עמודה 3
שורה 0 יוסי	80	85	90	97
שורה 1 אורי	75	80	85	90
שורה 2 משה	85	95	70	66

0	1	2	3	4	5	6	7	8	9	10	11
80	85	90	97	75	80	85	90	85	95	70	66

אתחול מערך דו-ממדי

= [עמודות][שורות] סוג_משתנה new שם_מערך [,] סוג_משתנה

```
{
{ נתוני שורה 1 },
{ נתוני שורה 2 },
{..... },
```

{ נתוני שורה n }

};

אפשר להגדיר את המערך ולגשת לתאים ספציפיים ולהזין לתוכם ערכים על ידי הצבה (כמו בדוגמה הקודמת) או קלט (כמו בדוגמה הבאה):

```
static void Main(string[] args)
{
    int [,]matrixInt = new int[3,4] // הגדרת מטריצה ואתחולה
    {
        {80,85,90,97},
        {75,80,85,90},
        {85,95,70,66}
    };

    int rowCount = matrixInt.GetLength(0); // מספר השורות
    int colCount = matrixInt.GetLength(1); // מספר העמודות
    int row;
    int col;
    // get data into matrix:
    Console.WriteLine("Please enter {0} numbers:", rowCount*colCount);
    Console.WriteLine("The matrix is:");
    for (row = 0; row < rowCount; row++)
    {
        for (col = 0; col < colCount; col++)
            matrixInt[row, col] = int.Parse(Console.ReadLine());
    }

    // Print the matrix:
    for (row = 0; row < rowCount; row++)
    {
        for (col = 0; col < colCount; col++)
            Console.Write(String.Format("{0}\t", matrixInt[row, col]));
        Console.WriteLine();
    }
    Console.ReadLine();
}
```

שימו לב לשורות הקוד הבאות:

```
for (row = 0; row < rowCount; row++)
{
    for (col = 0; col < colCount; col++)
        Console.Write(String.Format("{0}\t", matrixInt[row, col]));
    Console.WriteLine();
}
```

ה-for הראשון אחראי על כל השורות. ב-for השני אנו מציגים את כל האיברים בשורה ולאחר מכן יורדים שורה. לאחר מכן נחזור לביצוע לולאת ה-for הראשונה, ונשוב ונדפיס את כל השורות הבאות.

בנוסף, שימו לב לשימוש בקבועים להגדרת ממדי המערך:

```
class Program
{
    const int shurot = 3; // הגדרת קבועים
    const int amudot = 4;
    static void Main(string[] args)
    {
        int [,]matrixInt = new int[shurot, amudot] // הגדרת מטריצה ואתחולה
        {
            {80,85,90,97},
            {75,80,85,90},
            {85,95,70,66}
        };
        int rowCount = matrixInt.GetLength(0); // מספר השורות
        int colCount = matrixInt.GetLength(1); // מספר העמודות
        int row;
        int col;
        // get data into matrix:
        Console.WriteLine("Please enter {0}numbers:", rowCount*colCount);
        Console.WriteLine("The matrix is:");
        for (row = 0; row < rowCount; row++)
        {
            for (col = 0; col < colCount; col++)
                matrixInt[row, col] = int.Parse(Console.ReadLine());
        }

        // Print the matrix:
        for ( row = 0; row < rowCount; row++)
        {
            for ( col = 0; col < colCount; col++)
                Console.Write(String.Format("{0}\t", matrixInt[row, col]));
            Console.WriteLine();
        }
        Console.ReadLine();
    }
}
```

כעת נבצע את אותן המטלות באמצעות פונקציות:

הפונקציה הראשונה מקבלת כתובת של מטריצה וקולטת לתוכה נתונים:

```
static void GerMatrix(int [,]mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int row;
    int col;
    // get data into matrix:
    Console.WriteLine("Please enter {0} numbers:", rowCount * colCount);
    Console.WriteLine("The matrix is:");
    for (row = 0; row < rowCount; row++)
    {
        for (col = 0; col < colCount; col++)
            mat[row, col] = int.Parse(Console.ReadLine());
    }
}
```

הפונקציה השנייה מקבלת כתובת של מטריצה ומדפיסה את נתונה:

```
static void PrintMatrix(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int row;
    int col;
    // Print the matrix:
    Console.WriteLine("The matrix is:");
    for (row = 0; row < rowCount; row++)
    {
        for (col = 0; col < colCount; col++)
            Console.Write(String.Format("{0}\t", mat[row, col]));
        Console.WriteLine();
    }
}
```

נציג כאן בצורה מסודרת את קוד המקור בשלמותו:


```
using System;
using System.Collections.Generic;

namespace Chap5
{
    class Program
    {
        const int shurot = 3; // הגדרת קבועים
        const int amudot = 2;

        static void GerMatrix(int [,]mat)
        {
            int rowCount = mat.GetLength(0); // מספר השורות
            int colCount = mat.GetLength(1); // מספר העמודות
            int row;
            int col;
            // get data into matrix:
            Console.WriteLine
                ("Please enter {0} numbers:", rowCount * colCount);
            Console.WriteLine("The matrix is:");
            for (row = 0; row < rowCount; row++)
            {
                for (col = 0; col < colCount; col++)
                {
                    mat[row, col] = int.Parse(Console.ReadLine());
                }
            }
        }
    }
}
```

```
static void PrintMatrix(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int row;
    int col;
    // Print the matrix:
    Console.WriteLine("The matrix is:");
    for (row = 0; row < rowCount; row++)
    {
        for (col = 0; col < colCount; col++)
            Console.Write(String.Format(
                "{0}\t", mat[row, col]));
        Console.WriteLine();
    }
}

static void Main(string[] args)
{
    int[,] matrixInt = new int[shurot, amudot];
    GerMatrix(matrixInt);
    PrintMatrix(matrixInt);
    Console.ReadLine();
}
}
```

סיכום שורה במטריצה:

```
static int sum_Row(int r, int[,] mat) // סכום שורה
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = 0; i < colCount; i++)
        sum += mat[r, i];
    return (sum);
}
```

	0	1	2	3	4
0					
1	ש	ש	ש	ש	ש
2					
3					
4					

סיכום עמודה במטריצה:

```

static int sum_Col(int c, int[,] mat) // סכום עמודה
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = 0; i < rowCount; i++)
        sum += mat[i, c];
    return (sum);
}

```

	0	1	2	3	4
0			ע		
1			ע		
2			ע		
3			ע		
4			ע		

מערך דו-ממדי ריבועי

מערך דו-ממדי ריבועי הוא מערך שבו מספר השורות שווה למספר העמודות.

	0	1	2
0			
1			
2			

במערך ריבועי יש שני אלכסונים: אלכסון ראשי ואלכסון משני.

אלכסון ראשי

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

סכום איברי האלכסון הראשי

נבנה פונקציה שתחשב את סכום איברי האלכסון לפי ההיגיון הבא:

```
sum=0;
for (i=0; i<מס'מס'; i++)
    sum=sum+a[i,i];
```

החישוב בדוגמה זו מבוסס על $1 + 5 + 9 = 15$.

הקוד המלא:

```
static int SumMainAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i;
    for (i = 0; i < rowCount; i++)
        sum = sum + mat[i, i];
    return (sum);
}
```

אלכסון משני

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

סכום איברי האלכסון המשני

החישוב מתבצע על פי ההיגיון הבא:

```
sum=0;
for (i=0;i<השורות;i++)
    sum= sum+a[i][1-i]; // מספר שורות - i - 1;
```

החישוב בדוגמה זו מבוסס על $3 + 5 + 7 = 15$.

הקוד המלא:

```
static int SumSecondAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i;
    for (i = 0; i < rowCount; i++)
        sum = sum + mat[i, rowCount - 1 - i];
    return (sum);
}
```

עבודה עם מטריצה לא-סימטרית (מספר שורות שונה בכל פעם)

ניהול זיכרון

ניצור מטריצה כזו:

דנה	81	91				
יוסי	93	100	89	81	91	93
דני	98			100	164	104
				108	117	120

0	1	2
Unit	Unit	Unit
100	108	120

ניצור מערך מצביעים לפי מספר השורות.

זו תמונת זכרון של מטריצה לא סימטרית

סכום האיברים מעל האלכסון הראשי


```
sum=0; // סכום האיברים
for (i=0;i<n-1;i++) // שורות
    for (j=i+1;j<n;j++) // עמודות
        sum=sum+a[i,j];
```

הקוד המלא:

```
static int SumOverMainAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = 0; i < rowCount-1; i++)
        for (j = i + 1; j < colCount; j++)
            sum = sum + mat[i, j];
    return (sum);
}
```

סכום האיברים מתחת לאלכסון הראשי


```
sum=0;
for (i=1;i<n;i++)->שורות
    for (j=0;j<i;j++)->עמודות
        sum=sum+a[i,j];
```

הקוד המלא:

```
static int SumBelowMainAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    sum = 0;
    for (i = 1; i < rowCount; i++)
        for (j = 0; j < i; j++)
            sum = sum + mat[i, j];

    return (sum);
}
```

סכום האיברים מעל לאלכסון המשני


```
sum=0;
for (i=0;i<n-1;i++)->שורות
    for (j=0;j<n-i-1;j++)->עמודות
        sum=sum+a[i,j];
```

הקוד המלא:

```
static int SumOverSecondAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = 0; i < rowCount - 1; i++)
        for (j = 0; j < colCount - i - 1; j++)
            sum = sum + mat[i, j];
    return (sum);
}
```

סכום האיברים מתחת לאלכסון המשני


```
sum=0;
for (i=1;i<=n-1;i++)->שורות
    for (j=n-i;j<=n-1;j++)->עמודות
        sum=sum+a[i][j];
```

הקוד המלא:

```
static int SumBelowSecondAlachson(int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = 1; i <= rowCount - 1; i++)
        for (j = rowCount-i; j <= rowCount - 1; j++)
            sum = sum + mat[i, j];
    return (sum);
}
```

חישוב סכום ה"שכנים" של איבר במערך דו-ממדי

X – מספר השורה; y – מספר העמודה.

נחשב את סכום האיברים הנמצאים סביב האיבר שנמצא במקום [x][y]:

הקוד המלא:

```

sum = 0;

static int sumNeighbors(int x,int y,int[,] mat)
{
    int rowCount = mat.GetLength(0); // מספר השורות
    int colCount = mat.GetLength(1); // מספר העמודות
    int sum = 0, i, j;
    for (i = x - 1; i <= x + 1; i++)
        for (j = y - 1; j <= y + 1; j++)
            sum = sum + mat[i,j]; // הלולאה סיכמה גם את האיבר הנתון
    sum = sum - mat[x,y]; // לכן נחסר אותו מערך הצובר
    return (sum);
}

```

חיפושים ומיונים

מיון בועות – Bubble Sort

המטרה: קבלת מערך לא-ממוין, ומיונו על פי סדר עולה או סדר יורד.
לדוגמה, אם מתקבל המערך הבא –

	0	1	2	3
arr	3	8	1	4

לאחר המיון הוא ייראה כך:

	0	1	2	3
arr	1	3	4	8

עקרונות הפעולה:

- בכל שלב נבצע השוואה בין שני איברים סמוכים. אם האיבר השמאלי גדול מהאיבר הימני – אזי עתידה להתבצע החלפה.
- המעבר הזה מתבצע עבור על כל זוג איברים סמוכים. בסופו של דבר, האיבר הגדול ביותר "צף" אל המקום (האינדקס) הגבוה ביותר במערך.
- בכל השוואה תורנית (איטרציה) נוותר על האינדקס הגבוה ביותר מפני שמובטח לנו כי האיבר הגדול ביותר נמצא שם. בתום הסריקות נקבל מערך ממוין בסדר עולה, מהקטן לגדול.

נדגים את תהליך המיון על המערך הבא:

	0	1	2	3
arr	3	8	1	4

מעבר ראשון על המערך:

- נשווה בין האיבר הראשון לאיבר השני: האם $3 > 8$? התשובה היא "לא" – לכן לא נבצע דבר, ונתקדם.

- נשווה בין האיבר השני לאיבר השלישי. האם $8 > 1$? התשובה היא "כן" – לכן תתבצע החלפה בין האיברים:

	0	1	2	3
arr	3	1	8	4

- נשווה בין האיבר השלישי לאיבר הרביעי. האם $8 > 4$? התשובה היא "כן" – ולכן תתבצע החלפה בין האיברים:

	0	1	2	3
arr	3	1	4	8

- כעת ברור כי האיבר המקסימלי (הגדול ביותר) נמצא באינדקס האחרון:

	0	1	2	3
arr	3	1	4	8
				ממיון
			לא-ממיון	

מעבר שני על המערך:

כאמור, האיבר המקסימלי נמצא בשלב הזה במקום (אינדקס) האחרון, ולכן נשארו לנו פחות איברים למיין. נעבוד כעת רק על החלק השמאלי:

- נשווה בין האיבר הראשון לאיבר השני: האם $3 > 1$? התשובה היא "כן" – ולכן תתבצע החלפה:

	0	1	2	3

arr	1	3	4	8
	לא-ממוין			ממוין

- נשווה בין האיבר השני לאיבר השלישי. האם $3 > 4$? התשובה היא "לא" – ולכן לא נבצע דבר.

- הגענו לגבול העליון של המערך (האיבר הרביעי מונח במקומו מהאיטרציה הקודמת) – ולכן ברור לנו בשלב זה כי האיבר המקסימלי התורן (4) נמצא באינדקס האחרון התורן (אינדקס 2):

	0	1	2	3
arr	1	3	4	8
	לא-ממוין		ממוין	

מעבר שלישי על המערך:

כעת נעבוד שוב רק על החלק השמאלי, שעדיין אינו ממוין.

- נשווה בין האיבר הראשון לאיבר השני: האם $1 > 3$? התשובה היא "לא" – ולכן לא תבצע החלפה.

הגענו לגבול העליון של המערך. אין משמעות למיון של איבר בודד – ולכן הוא נשאר במקומו.

למעשה, כל המערך ממוין:

	0	1	2	3
arr	1	3	4	8
	ממוין			

פונקציה המבצעת מיון בועות על מערך שלמים:

```
public static void BubbleSort(int[] array) // Bubble Sort
{
    int temp;
    for (int i = 0; i < array.Length; i++)
    {
        for (int j = 0; j < array.Length - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                temp = array[j + 1];
                array[j + 1] = array[j];
                array[j] = temp;
            }
        }
    }
}
```

דוגמה להפעלת הפונקציה:

```
static void Main(string[] args)
{
    int[] arr = { 3, 8, 1, 4 };
    Console.WriteLine("Before Sorting\n");
    foreach (var item in arr)
        Console.Write(item + " ");
    BubbleSort(arr);
    Console.WriteLine("\nAfter Sorting\n");
    foreach (var item in arr)
        Console.Write(item + " ");

    Console.ReadKey();
}
}
```

הפלט על המסך:

Before Sorting

3 8 1 4

After Sorting

1 3 4 8

מיין הכנסה – Insertion Sort

במיון זה נחפש באיטרציה הראשונה את הערך המינימלי, ונמקם אותו באינדקס 0.

- לאחר מכן נחליף (swap) בין האיבר הממוקם באינדקס 0 (אפס) לבין הערך המינימלי.
- באיטרציה הבאה (השנייה ומעלה) נחפש את האיבר הנמוך ביותר החל מאינדקס 1 (ומעלה), ונחליף אותו בערך הממוקם באינדקס 1 (ומעלה).

המערך לפני המיון:

	0	1	2	3	4
arr	6	4	2	1	8

מעבר ראשון על המערך:

- נסרוק את כל המערך, ונמצא כי הערך 1 (באינדקס 3) הוא הערך המינימלי:

	0	1	2	3	4
arr	6	4	2	1	8

Min

- נחליף את האיבר המינימלי התורן (1) באיבר הראשון באינדקס 0. ידוע לנו כי כעת ממוקם באינדקס 0 הערך המינימלי:

	0	1	2	3	4
arr	1	4	2	6	8
	ממוין	לא-ממוין			

מעבר שני על המערך:

- נסרוק את המערך מאינדקס 1 ונמצא (באינדקס 2) את האיבר המינימלי:

	0	1	2	3	4
arr	1	4	2	6	8

min	ממוין
-----	-------

- נחליף את האיבר המינימלי התורן (2) באיבר השני באינדקס 1:

	0	1	2	3	4
arr	1	2	4	6	8
		ממוין			

- נסרוק את המערך **מאינדקס 2** (האינדקסים שלפניו ממוינים), ונמצא (באינדקס 2) את האיבר המינימלי:

	0	1	2	3	4
arr	1	2	4	6	8
			Min		

- האינדקס של האיבר המינימלי התורן (2) זהה למיקום שבו אמור להימצא האיבר החדש (אינדקס 2), ולכן לא תתבצע כל החלפה:

	0	1	2	3	4
arr	1	2	4	6	8
	ממוין	ממוין	ממוין		

- נסרוק את המערך **מאינדקס 3** (האינדקסים שלפניו ממוינים) ונמצא (באינדקס 3) את האיבר המינימלי:

	0	1	2	3	4
arr	1	2	4	6	8
				Min	

- האינדקס של האיבר המינימלי התורן (3) זהה למיקום שבו אמור להימצא האיבר החדש (אינדקס 3), ולכן לא תתבצע כל החלפה:

	0	1	2	3	4
arr	1	2	4	6	8
	ממוין	ממוין	ממוין	ממוין	

- נשאר איבר בודד שלא עבר מיון, ולכן בהכרח הוא הגדול ביותר.

למעשה, כל המערך ממוין!

```
public static void InsertionSort(int[] array)
{
    int temp, j;
    for (int i = 1; i < array.Length; i++)
    {
        temp = array[i];
        j = i - 1;
        while (j >= 0 && array[j] > temp)
        {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = temp;
    }
}
```

חיפוש סדרתי

חיפוש סדרתי במערך לא-ממוין

נחפש ערך (value) בתוך המערך.

אם הערך קיים – הפונקציה תחזיר את האינדקס שבו נמצא הערך.

אחרת, הפונקציה תחזיר את הערך -1 (אין אינדקס כזה).

```
public static int Is_exist1(int[] a, int lookfor)
{
    bool flag = true;
    Console.WriteLine("Lookfor " + lookfor + ": ");
    int i;
    for (i = 0; i < a.Length && flag; i++)
    {
        if (a[i] == lookfor)
        {
            flag = false;
            return (i);
        }
    }
    return (-1);
}

static void Main(string[] args)
{
    int[] arr1 = { 12, 7, 5, 3, 8 };

    Console.WriteLine(Is_exist1(arr1, 4));
    Console.WriteLine(Is_exist1(arr1, 5));

    Console.ReadKey();
}
```

עבור חיפוש הערך 4 – יודפס -1.
עבור חיפוש הערך 5 – יודפס 2.

חיפוש סדרתי במערך ממין

המשמעות היא חיסכון בזמן, מפני שאיננו חייבים להמשיך לחפש את הערך שאנו מבקשים למצוא.
נניח כי קיים המערך הבא:

	0	1	2	3	4	5
arr	6	9	11	17	21	30

נחפש ערך (value) בתוך המערך. נניח כי הערך שאנו מחפשים במערך הוא 10.
ברגע שהגענו לערך 11 (באינדקס מספר 2) – אין טעם להמשיך בחיפוש.


```

int Find_value_in_sorted_array(int[] a, int val)
{
    int i;
    for (i = 0; i < a.Length && val >= a[i]; i++)
    {
        Console.WriteLine("check " + a[i] + " ");
        if (a[i] == val)
        {
            Console.WriteLine("found " + val + " !");
            return (i + 1);
        }
    }
    return (-1); // not found
}

static void Main(string[] args)
{
    int[] arr = { 6, 9, 11, 17, 21, 30 };
    int look_for = 16;

    Console.WriteLine(Find_value_in_sorted_array(arr, look_for));
    Console.ReadKey();
}

```

חיפוש בינארי

החיפוש הבינארי יעבוד אך ורק על מערך ממוין ומטרתו היא למצוא את מיקומו של איבר מסוים (או להחזיר את הערך [-1]).

עקרונות הפעולה:

כל עוד קיימים איברים במערך –

נבדוק האם האיבר המבוקש הוא האמצעי. אם כן, נחזיר את המיקום (אינדקס) שלו; אחרת, אם הוא קטן מהאיבר האמצעי – נחפש אותו בתת-המערך השמאלי; אחרת, נחפש אותו בתת-המערך הימני. אם הגענו לכאן משמע כי האיבר נמצא – ונחזיר את הערך (-1).

דוגמה:

נניח כי אנו מחפשים את האיבר 24 במערך הבא:

0	1	2	3	4	5	6	7	8	9	10	11
5	8	9	12	15	17	20	21	24	27	29	31

תחילה נחלק את המערך לשניים:

0	1	2	3	4	5	6	7	8	9	10	11
5	8	9	12	15	17	20	21	24	27	29	31
שמאלי חצי						ימני חצי					

הערך 24, אם הוא קיים, יופיע בחצי הימני של המערך המקורי. נחצה גם אותו:

6	7	8	9	10	11
20	21	24	27	29	31
שמאלי חצי			ימני חצי		

ערך 24, אם הוא קיים, יופיע בחצי השמאלי:

6	7	8
20	21	24
שמאלי חצי		ימני חצי

21 הוא הערך האמצעי. $24 > 21$; ולכן, אם הערך 24 קיים – הוא יופיע מימינו. הערך (24) קיים – ולכן יוחזר הערך 8 (כאינדקס).

דוגמה נוספת

נניח כי אנו מחפשים את הערך 10 במערך הבא:

0	1	2	3	4	5
2	4	6	9	15	19

הערך 10, אם הוא קיים, אמור להופיע בחצי הימני:

0	1	2	3	4	5
2	4	6	9	15	19
שמאלי חצי			ימני חצי		

על כן, הוא אמור להופיע כאן:

3	4	5
9	15	19
	ימני	חצי

10 < 15 (15 הוא הערך האמצעי); ולכן, אם הערך 10 קיים – הוא אמור להופיע משמאל לערך 15. הוא אינו מופיע שם (משמאלו של הערך 15 מופיע הערך 9) – ולכן יוחזר (-1) כאינדקס שאינו יכול להתקיים.

קוד מקור ודוגמאות הרצה

```

public static int BinarySearch(int[] arr,int LookForValue)
{
    int low = 0;
    int high = arr.Length - 1;
    int middle=0;
    bool found = false;
    while (low <= high && !found)
    {
        middle = (low + high) / 2;
        if (LookForValue == arr[middle])
        {
            Console.WriteLine("{0} found at {1}", LookForValue, middle);
            found = true;
        }
        else if (!found && LookForValue < arr[middle])
            high = middle - 1;
        else
            low = middle + 1;
    } // while
    if (!found) // == false
    {
        Console.WriteLine("{0} not found !",LookForValue);
        return (-1);
    }
    return (middle); // found !
}

static void Main(string[] args)
{
    int[] Arr = { 5, 8, 9, 12, 15, 17, 20, 21, 24, 27, 29, 31 };
    Console.WriteLine(BinarySearch(Arr,24));
    Console.WriteLine(BinarySearch(Arr,22));
    Console.ReadLine();
}
}

```

פלט ההרצה

24 found at 8

8

22 not found!

-1

מיזוג מערכים ממוינים

תנאי מקדים למיזוג מערכים הוא, ששני המערכים יהיו ממוינים לפני המיזוג.

כל עוד שני המערכים אינם ריקים –

נבדוק האם האיבר הנוכחי במערך הראשון קטן או שווה לאיבר הנוכחי במערך השני:

- אם התשובה היא "כן" – עלינו להעתיק את האיבר הנוכחי מהמערך הראשון למיקום הנוכחי במערך השלישי (הממוזג), ואז לקדם את מיקום האיבר הנוכחי.
- אחרת, עלינו להעתיק את האיבר הנוכחי מהמערך השני למיקום הנוכחי במערך השלישי (הממוזג), ואז לקדם את מיקום האיבר הנוכחי.

אחד משני המצבים הבאים עתיד להתרחש:

- נצטרך להעתיק את כל האיברים שנותרו במערך הראשון – אל המערך השלישי (הממוזג).
- נצטרך להעתיק את כל האיברים שנותרו במערך השני – אל המערך השלישי (הממוזג).

קוד מקור ודוגמאות הרצה

```
public static int []MergeSortedArrays(int []first, int []second)
{
    int []newArr = new int[first.Length + second.Length];
    int i = 0, j = 0;
    while (i < first.Length && j < second.Length)
    {
        if (first[i] <= second[j])
            newArr[i + j] = first[i++];
        else
            newArr[i + j] = second[j++];
    }

    while (i < first.Length) // copy the rest of the first array
        newArr[i + j] = first[i++];

    while (j < second.Length) // copy the rest of the second array
        newArr[i + j] = second[j++];

    return newArr; // return the address of the array
}

static void Main(string[] args)
{
    int []sorted;
    int []a = {2,4,5,7};
    int []b = {3,6,7,8,9};
    sorted = MergeSortedArrays(a, b);

    foreach (var item in sorted)
        Console.Write(item + " ");

    Console.ReadLine();
}
```

פלט ההרצה:

2 3 4 5 6 7 7 8 9

נפרט

בתחילת התהליך נתונים המערכים הבאים:

	0	1	2	3
first	2	4	5	7

i

Length=4

	0	1	2	3	4
second	3	6	7	8	9

j

Length=5

בדיקת תנאי קיום (לולאת while):

$0 < 4 \ \&\& \ 0 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):

$First[0] \leq second[0] \Leftrightarrow 2 \leq 3 \Leftrightarrow T$

ולכן תתבצע הפקודה הבאה:

$newArr[i+j] \rightarrow newArr[0+0] \rightarrow newArr[0]=first[i++]$

	0	1	2	3
first	2	4	5	7

i

Length=4

	0	1	2	3	4
second	3	6	7	8	9

j

Length=5

	0	1	2	3	4	5	6	7	8
newArr	2	3	4	5	6	7	7		

בדיקת תנאי קיום בגוף לולאת ה-while (בפעם השנייה):

$1 < 4 \ \&\& \ 0 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):

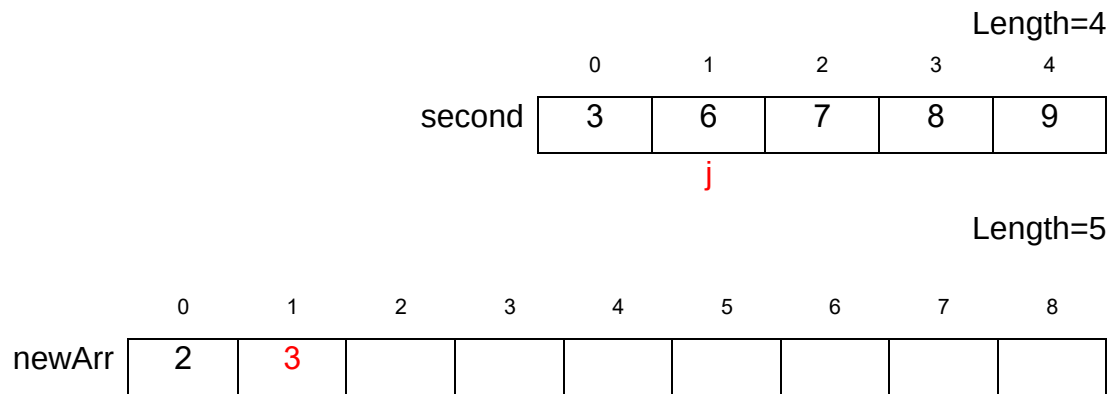
$first[1] \leq second[0] \Leftrightarrow 4 \leq 3 \Leftrightarrow F$

ולכן תתבצע הפקודה הבאה:

$newArr[i+j] \rightarrow newArr[1+0] \rightarrow newArr[1]=second[j++]$

	0	1	2	3
first	2	4	5	7

i



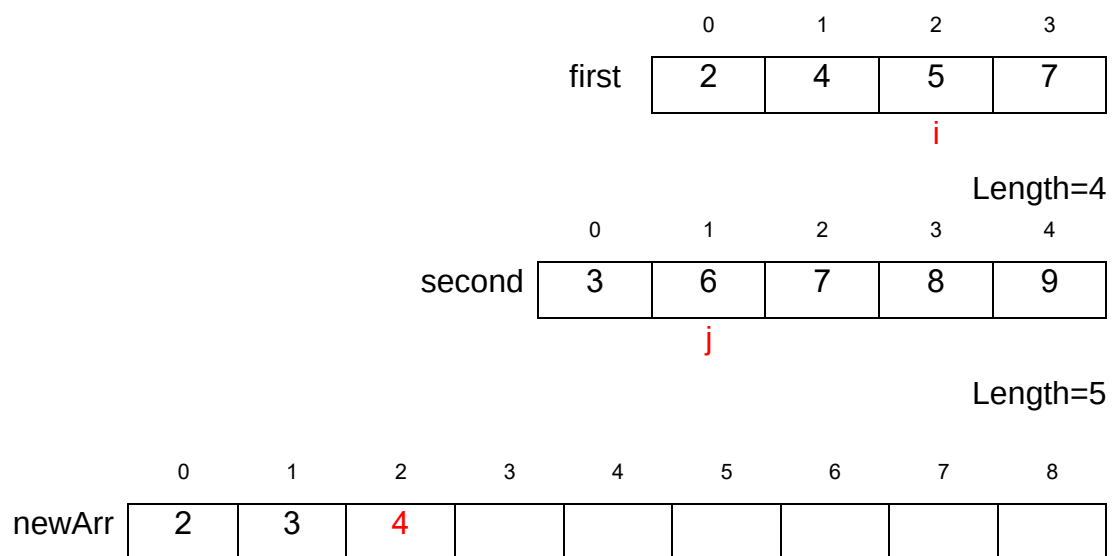
בדיקת תנאי קיום בגוף לולאת ה-while (בפעם השלישית):
 $1 < 4 \ \&\& \ 1 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):

$First[1] \leq second[1] \Leftrightarrow 4 \leq 6 \Leftrightarrow T$

ולכן תתבצע הפקודה הבאה:

$newArr[i+j] \rightarrow newArr[1+1] \rightarrow newArr[2]=first[i++]$



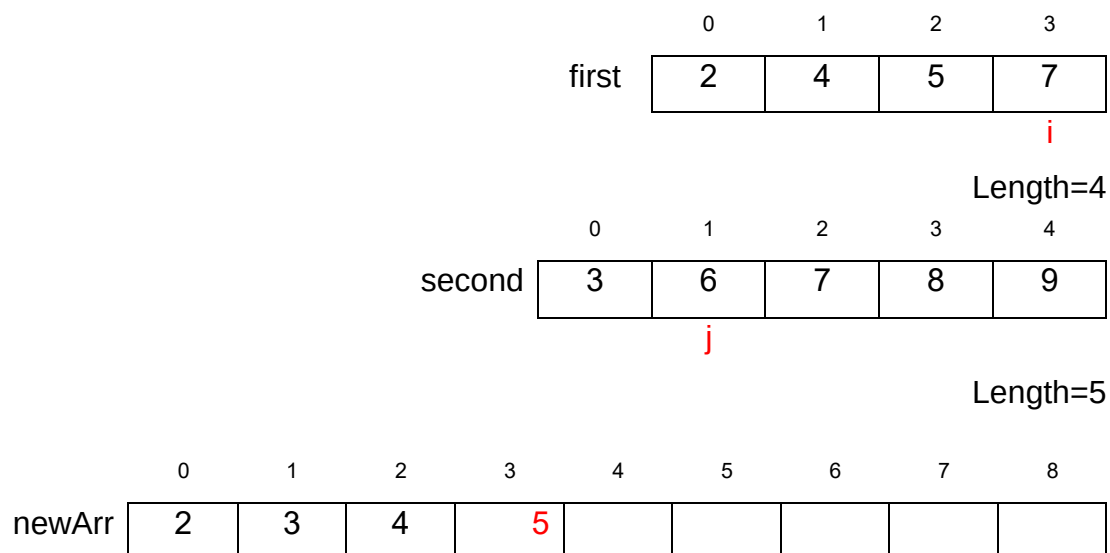
בדיקת תנאי קיום בגוף לולאת ה-while (בפעם הרביעית):
 $2 < 4 \ \&\& \ 1 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):

$First[2] \leq second[1] \Leftrightarrow 5 \leq 6 \Leftrightarrow T$

ולכן תתבצע הפקודה הבאה:

$newArr[i+j] \rightarrow newArr[2+1] \rightarrow newArr[3]=first[i++]$

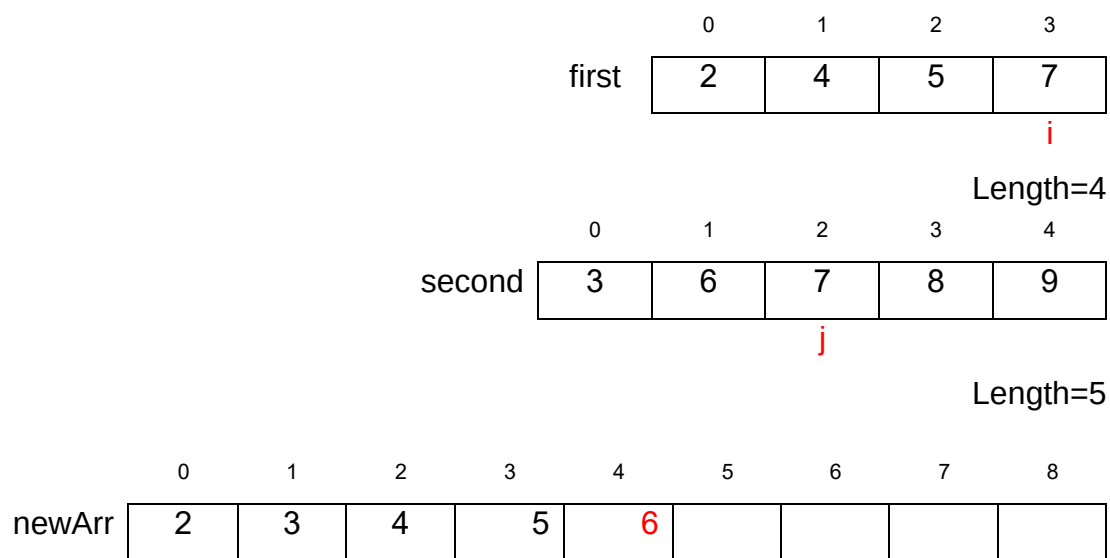


בדיקת תנאי קיום בגוף לולאת ה-while (בפעם החמישית):
 $3 < 4 \ \&\& \ 1 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):
 $First[3] \leq second[1] \Leftrightarrow 7 \leq 6 \Leftrightarrow F$

ולכן תתבצע הפקודה הבאה:

$newArr[i+j] \rightarrow newArr[3+1] \rightarrow newArr[4]=second[j++]$



בדיקת תנאי קיום בגוף לולאת ה-while (בפעם השישית):
 $3 < 4 \ \&\& \ 2 < 5 \Leftrightarrow T$

בדיקת שורת ה-if (בתוך גוף לולאת ה-while):

$\text{First}[3] \leq \text{second}[2] \Leftrightarrow 7 \leq 7 \Leftrightarrow T$

ולכן תתבצע הפקודה הבאה:

$\text{newArr}[i+j] \rightarrow \text{newArr}[3+2] \rightarrow \text{newArr}[5] = \text{first}[i++]$

0123

first

2457

4 הוא i של ערכו

Length=4

01234

second

36789

j

Length=5

012345678

newArr

234567

בדיקת תנאי קיום בגוף לולאת ה-while (בפעם השביעית):

```
while (i < first.Length && j < second.Length)
```

$4 < 4 \rightarrow F$

כעת נגיע אל לולאת ה-while השנייה:

```
while (i < first.Length) // copy the rest of the first array
    newArr[i + j] = first[i++];
```

נבדוק אם $4 < 4$. התשובה היא F. המשמעות היא, שהמערך הראשון כבר מוזג למערך השלישי.

כעת נגיע אל לולאת ה-while השלישית:

```
while (j < second.Length) // copy the rest of the second array
    newArr[i + j] = second[j++];
```

נבדוק אם $2 < 5$. התשובה היא T.

ניכנס לפקודה (היחידה) בגוף הלולאה בפעם הראשונה ונבצע אותה:

newArr[4] + 2] = newArr[6] = second[2]
newArr[6] = 7

וכן, לא נשכח לקדם את ערכו של j:

	0	1	2	3	4
second	3	6	7	8	9

j
Length=5

	0	1	2	3	4	5	6	7	8
newArr	2	3	4	5	6	7	7		

כעת נבצע שוב את גוף לולאת ה-while השלישית:

```
while (j < second.Length)
    newArr[i + j] = second[j++];
```

newArr[4] + 3] = newArr[7] = second[3]
newArr[7] = 8

ושוב, לא נשכח לקדם את ערכו של j:

	0	1	2	3	4
second	3	6	7	8	9

j
Length=5

	0	1	2	3	4	5	6	7	8
newArr	2	3	4	5	6	7	7	8	

נבצע (בפעם השלישית) את גוף לולאת ה-while השלישית (מאחר ש- 5 < 4):

```
while (j < second.Length)
    newArr[i + j] = second[j++];
```

newArr[4] + 4] = newArr[8] = second[4]
newArr[8] = 9

ערכו של j קודם לערך 5.

בשלב הזה לא ניכנס פעם נוספת לגוף לולאת ה-while השלישית, מאחר שהשאלה הבאה –

5 < 5 – מניבה את הערך "שקר".

המשמעות היא, שגם המערך השני כבר מוזג למערך השלישי.

תמונת זיכרון כוללת:

	0	1	2	3					
first	2	4	5	7					
	4 הוא	i	של	ערכו					
	Length=4								
	0	1	2	3	4				
second	3	6	7	8	9				
	5	הוא	j	של	ערכו				
	Length=5								
	0	1	2	3	4	5	6	7	8
newArr	2	3	4	5	6	7	7	8	9
1200									

כעת יקבל ערכו של המשתנה sorted את הערך 1200.

1200 היא כתובתו של המערך הממוזג.

```
sorted = MergeSortedArrays(a, b);
```

הפלט, בסוף התהליך, הוא הדפסה של המערך הממוין.