

# יסודות מדעי המחשב

## בשפת C# – נספח

תמר בניה וד"ר מיכל ארמוני – ראשי צוות הכתיבה

יעל בילצ'יק

נעה גרדוביץ

עדי גרין

אתי מנשה (סעיפי התבניות)

הילה קדמן (נספח)

ייעוץ: ד"ר דוד גינת

עריכה: לירון ברגר

תשע"א 2010



יסיטת תל-אביב החוג להוראת המדעים

מטה מל"מ המרכז הישראלי להוראת המדעים ע"ש עמוס דה-שליט

משרד החינוך האגף לתכנון ולפיתוח תכניות לימודים



# יסודות מדעי המחשב בשפת C# – נספח

תמר בניה וד"ר מיכל ארמוני – ראשי צוות הכתיבה

יעל בילצ'יק

נעה גרדוביץ

עדי גרין

אתי מנשה (סעיפי תבניות)

הילה קדמן (נספח)

ייעוץ: ד"ר דוד גינת

עריכה: לירון ברגר

כל הזכויות שמורות © 2010

השראה הוצאה לאור, ת"ד 19022, חיפה 31190

טל': 04-8254752, פקס: 1534-8254752

E-Mail: [books@hashraa.co.il](mailto:books@hashraa.co.il)

[www.hashraa.co.il](http://www.hashraa.co.il)



**השראה הוצאה לאור**

עיצוב העטיפה: טל גרין

אין לשכפל, להעתיק, לצלם, לתרגם, להקליט, לאחסן במאגר מידע כלשהו, לשדר או לקלוט בכל דרך או בכל אמצעי אלקטרוני, אופטי או מכני (לרבות צילום, הקלטה, אינטרנט, מחשב ודואר אלקטרוני), כל חלק שהוא מהחומר שבספר זה. שימוש מסחרי מכל סוג בחומר הכלול בספר זה אסור בהחלט, אלא ברשות מפורשת בכתב מהמוציא לאור ומהגורמים המפורטים להלן.



כל הזכויות שמורות  
משרד החינוך

# פעולות סטטיות ומחלקות שירות

רוב תכניות המחשב, המבצעות משימה משמעותית, מורכבות ממגוון של תת-משימות. הדרך היעילה להתמודד עם תכנון התכנית, עם בנייתה ועם תחזוקתה, היא חלוקתה ליחידות קטנות יחסית שכל אחת מהן מבצעת תת-משימה מוגדרת.

היתרונות בחלוקת המשימה הכללית לתת משימות:

- **שיפור הקריאות** – התכנית הראשית עוסקת ב**מה** ולא ב**איך**. במקום הרבה שורות קוד שקשה לעקוב אחריהן ולהבין מה הן מבצעות, תכלול התכנית סדרה של הוראות הפעלה לתת-משימות מוגדרות.
  - **תחזוקה** – קל לאתר שגיאות בתכנית ולתקן. מכיוון שכל פעולה עוסקת במשימה מוגדרת, ניתן להגיע ישירות לקטע הקוד הבעייתי, ולטפל בו.
  - **חיסכון בכתיבה** – כאשר יש משימה מסוימת הצריכה להתבצע יותר מפעם אחת במהלך התכנית, ניתן לכתוב פעולה המבצעת את המשימה ולזמן אותה מספר פעמים לפי הנדרש.
  - **חלוקת העבודה** – ניתן לחלק את המשימה הגדולה לכמה מתכנתים, כך שכל אחד מהם מטפל בתת-משימה מוגדרת שתהווה חלק מהמשימה הכללית.
  - **שימוש יעיל בספריות קיימות** – לדוגמא: השימוש במחלקה Math המכילה אוסף של פעולות מתמטיות. כאשר ניגשים לפתור בעיה תכנותית, יש לפרק את המשימה הכללית לתת-משימות המבצעות כל אחת פעולה מוגדרת, ואחר כך לחבר הכול לתכנית השלמה. כל תת-משימה היא פעולה עצמאית הפועלת על משתנים משלה, והתכנית הראשית מזמנת את הפעולות לפי הצורך.
- קריאה לפעולה מתוך התכנית גורמת לתכנית להסתעף למקום אחר, לבצע את ההוראות הכתובות בו ולחזור להמשך ביצוע התכנית מהשורה שאחרי השורה שגרמה לסיעוף.

**את הפעולה ניתן לזמן מספר פעמים במהלך התכנית.**

# 1. פעולה שאינה מקבלת דבר ואינה מחזירה דבר

## קציה 1

מטרת הבעיה ופתרונה: הצגת פירוק לתת-בעיות ופתרון כל תת-בעיה בפעולה נפרדת.

להלן תכנית המאפשרת להציג על המסך צורה גיאומטרית, בהתאם לבחירת המשתמש. הצורות האפשריות תהיינה: ריבוע, מלבן או משולש.

### פירוק הבעיה לתת משימות

1. ציור ריבוע.
2. ציור מלבן.
3. ציור משולש.
4. כדי לאפשר בחירה בין האפשרויות השונות, נוסיף תפריט בחירה.

לו היינו כותבים את התכנית בכלים שרכשנו עד כה, קרוב לוודאי שהייתה מתקבלת תכנית עמוסה בהוראות קוד ושהיה צורך בטבלת מעקב או הרצה כדי להבין מה היא מבצעת. נציג פתרון המשתמש בפעולות מיוחדות לביצוע כל אחת מהמשימות שקבענו:

### האלגוריתם

1. הצג גפריט-בוירה ( )
2. קאוט את בויהג האמאש - choice
3. אק ערכו של choice הוא 1  
ציי-ריבוע ( )
4. אק ערכו של choice הוא 2  
ציי-מלבן ( )
5. אק ערכו של choice הוא 3  
ציי-משולש ( )

### התכנית

```
using System;
public class Shapes
{
    //--- פעולה המציירת ריבוע בגודל 5x5 ---
    static void DrawSquare()
    {
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
                Console.Write(" *");
            Console.WriteLine();
        }
    }
}
```

```

//--- 5x10 פעולה המציירת מלבן בגודל ---
static void DrawRectangle()
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 10; j++)
            Console.Write(" *");
        Console.WriteLine();
    }
}

//--- פעולה המציירת משולש ישר-זווית ---
//--- שאורך כל אחד מניצביו הוא 5 ---
static void DrawTriangle()
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j <= i; j++)
            Console.Write(" *");
        Console.WriteLine();
    }
}

//--- פעולה המציגה תפריט בחירה ---
static void ShowMenu()
{
    Console.WriteLine("Geometric shape menu: ");
    Console.WriteLine(" 1. Square ");
    Console.WriteLine(" 2. Rectangle ");
    Console.WriteLine(" 3. Triangle ");
    Console.WriteLine();
}

static void Main()
{
    int choice;

    ShowMenu();
    Console.Write("Type shape number --> ");
    choice = int.Parse(Console.ReadLine());

    if (choice == 1)
        DrawSquare();
    if (choice == 2)
        DrawRectangle();
    if (choice == 3)
        DrawTriangle();
}
}

```

ההוראות drawTriangle(), drawRectangle(), drawSquare() ו-showMenu() הן משפטי זימון לפעולות (methods) המבצעות כל אחת תפקיד מוגדר.

לכל אחת מהפעולות מבנה דומה המתחיל בכותרת הפעולה, ואחריה גוף הפעולה התחום בתוך סוגריים מסולסלים:

```
--- תיעוד הפעולה ---  
static void שם-הפעולה ()  
{  
    // גוף הפעולה  
}
```

כותרת הפעולה מזהה את הפעולה, ומהווה ממשק בינה לבין התכנית המזמנת את הפעולה, כלומר - מספקת מידע איך להשתמש בפעולה:

**רמת הרשאה** – רכיב זה מאפשר לקבוע למי מותר לגשת לפעולה. המילה **public** מציינת גישה ציבורית, כלומר ניתן לגשת לפעולה מכל מקום בתכנית. בשלב זה של העבודה, רכיב זה הוא רשות ולא חובה, ולכן לא השתמשנו בו בתכנית.

**static** – פעולה של מחלקה מתחילה תמיד במילה **static** המציינת שהפעולה היא **class member**, כלומר – פעולה השייכת למחלקה ולא פעולה השייכת לעצם.

**טיפוס הערך המוחזר** – יש פעולות שמחזירות ערך ובמקרה זה הן חייבות לציין את טיפוס הערך המוחזר. פעולות שלא מחזירות ערך, חייבות לציין זאת באמצעות המלה השמורה **void** (ריק). הפעולות המוצגות בתכנית הן פעולות מסוג זה.

**שם הפעולה** – השם שבו נזמן את הפעולה. מקובל לתת לפעולה שם משמעותי המרמז על תפקידה. שם פעולה יתחיל באות גדולה. שם פעולה המכיל שתי מילים או יותר, יחובר למילה אחת בדרך המקובלת בשפת **C#**, כלומר – כל מילה תתחיל באות גדולה. למשל: **DrawSquare()**. שים ♥: אין לתת לפעולה שם שהוא מילה שמורה או הוראה בשפה.

**גוף הפעולה** – גוף הפעולה יהיה תחום בתוך סוגריים מסולסלים ויכלול את הוראות הביצוע של הפעולה. ההוראות הן הוראות השפה המוכרות.

הפעולה יכולה להגדיר משתני עזר לביצוע החישובים. משתנים אלו קרויים **משתנים פנימיים** או **משתנים מקומיים** והם מוכרים רק בתוך הפעולה שבה הם הוגדרו, גם אם קיימים משתנים בשם זהה בפעולה אחרת. שם המשתנה הפנימי יהיה לפי כללי מתן שמות בשפת **C#**. שים ♥: אין לתת למשתנה פנימי שם זהה לשם הפעולה שהוא מוגדר בה.

**תיעוד הפעולה** – מקובל לתעד כל פעולה בתכנית. (תכנית הלימודים מחייבת תיעוד). התיעוד יכלול: תיאור של מה שהפעולה מבצעת ומה היא מחזירה. אם הפעולה מקבלת ערכים (פרמטרים), נרשום תיאור של הערכים האלו. אם על הערכים האלה לקיים תנאי כלשהו כדי שהפעולה תפעל כראוי, נרשום תנאים אלו כהנחות שאנו מניחים לגביהם (למשל: הנחה: המספר אינו שלילי).

דוגמאות לתיעוד:

- פעולה המקבלת ... (תיאור הפרמטרים) ... ומחזירה ... (מה מחזירה הפעולה) ... הנחות: ... (רשימת ההנחות שמניחה הפעולה לגבי הפרמטרים) ...
- פעולה המקבלת ... (תיאור הפרמטרים) ... ומבצעת ... (מה מבצעת הפעולה) ... הנחות: ... (רשימת ההנחות שמניחה הפעולה לגבי הפרמטרים) ...
- טענת כניסה: ... (אילו פרמטרים מקבלת הפעולה ומהן ההנחות לגביהם) טענת יציאה: ... (מה עושה / מחזירה הפעולה)

ההוראה בתכנית המפעילה את הפעולה נקראת **משפט זימון**. את הפעולה מזמנים בציון שמה בתוספת סוגריים עגולים. למעשה, ההבחנה בין שם של משתנה לשם של פעולה, הוא בתוספת הסוגריים העגולים בשם הפעולה. למשל. למשל: `ShowMenu()`, `Console.WriteLine()`.

את הפעולות אפשר לכתוב במחלקה הראשית לפני הפעולה **Main** או אחריה.

סוף פתרון קציה 1

## 2. פעולה המקבלת פרמטרים ואינה מחזירה דבר

### קציה 2

מטרת פתרון הבעיה: העברת פרמטרים לפעולה

נשנה את התכנית כך שתצייר צורות גיאומטריות בגדלים המותאמים לבקשת המשתמש בתכנית. לאחר שיוצג התפריט ויבחר המשתמש את הצורה הרצויה, הוא יתבקש להקליד את מידות הצורה. ערכים אלו יועברו כ**פרמטרים** לפעולה המתאימה, והיא תצייר את הצורה במידות המבוקשות.

פעולה המקבלת ערך מהתכנית

//--- תיעוד הפעולה ---

**static void** שם-הפעולה (שם-פרמטר-1 **טיפוס-פרמטר-1**, שם-פרמטר-2 **טיפוס-פרמטר-2**)

{

// גוף הפעולה

}

הפרמטרים שבשורת הכותרת, הנקראים גם **פרמטרים פורמאליים**, הם משתנים השייכים לפעולה, ומקבלים את ערכם מהתכנית באמצעות ההוראה המזמנת בתכנית, ושם הם נקראים: **פרמטרים אקטואליים** או **ארגומנטים**.

**טיפוס-הפרמטר** – הוא טיפוס הנתונים של הפרמטר המועבר לפעולה.

**שם-הפרמטר** – הוא שם המשתנה כפי שיוכר בפעולה.

- מספר הפרמטרים המועברים אינו מוגבל, ויש להצהיר על כל אחד מהם בנפרד. אם יש יותר מפרמטר אחד, יש להפריד ביניהם בסימני פסיק.
- שם הפרמטר הפורמאלי (המוכר בפעולה) אינו חייב להיות זהה לשם הפרמטר האקטואלי (המוכר בתכנית המזמנת).
- סדר הרישום ומספר הארגומנטים שיופיעו במשפט הזימון לפעולה, חייב להיות תואם לסדר ולטיפוס המוצהר ברשימת הפרמטרים. למשל: פעולה בשם `compute` המקבלת כפרמטר שלושה משתנים מספריים: שלם, שלם וממשי:

**static void Compute (int a, int b, double x)**

משפטי זימון אפשריים לפעולה יהיו:

```
compute(3, 12, 4.25);
```

או :

```
int n1 = -4, n2 = 5;
double x = 0.25;
compute (n1, n2, x);
```

ניסיון לזמן את הפעולה עם ארגומנטים בסדר שונה מהסדר שנקבע בכותרת הפעולה יגרום לשגיאה.

- הפרמטר מקבל את ערכו מהפעולה המזמנת. המשמעות – אין לקלוט בתוכו ערך חדש בתוך הפעולה.
- פרמטרים מטיפוס בסיסי (int, double וכד') מעבירים לפעולה את ערכם, כלומר – בפעולה נוצר עותק שלהם. כל שינוי בערכי הפרמטרים בתוך הפעולה, לא ישפיעו או ישנו את ערכי הארגומנטים בתכנית המזמנת.

### התכנית המקבלת את גודל הצורה הגיאומטרית

```
using System;
public class Shapes2
{
    //--- טענת כניסה: side אורך צלע הריבוע ---
    //--- טענת יציאה: מצויר ריבוע שאורך צלעו side ---
    static void DrawSquare(int side)
    {
        for (int i = 0; i < side; i++)
        {
            for (int j = 0; j < side; j++)
                Console.Write(" *");
            Console.WriteLine();
        }
    }

    //--- טענת כניסה: length ו-width אורך צלעות המלבן ---
    //--- טענת יציאה: מצויר מלבן שאורך צלעותיו length ו-width ---
    static void DrawRectangle(int length, int width)
    {
        for (int i = 0; i < width; i++)
        {
            for (int j = 0; j < length; j++)
                Console.Write(" *");
            Console.WriteLine();
        }
    }

    //--- טענת כניסה: trigSide אורך ניצב משולש ישר זווית ---
    //--- טענת יציאה: מצויר משולש שאורך כל ניצב הוא trigSide ---
    static void DrawTriangle(int trigSide)
    {
        for (int i = 0; i < trigSide; i++)
        {
            for (int j = 0; j <= i; j++)
                Console.Write(" *");
            Console.WriteLine();
        }
    }
}
```



```

//--- פעולה המציגה תפריט בחירה ---
static void ShowMenu()
{
    Console.WriteLine("Geometric shape menu: ");
    Console.WriteLine(" 1. Square ");
    Console.WriteLine(" 2. Rectangle ");
    Console.WriteLine(" 3. Triangle ");
    Console.WriteLine();
}

static void Main()
{
    int choice;
    int sidel, side2;

    ShowMenu();
    Console.Write("Type shape number --> ");
    choice = int.Parse(Console.ReadLine());

    if (choice == 1)
    {
        Console.Write("type square side --> ");
        sidel = int.Parse(Console.ReadLine());
        DrawSquare(sidel);
    }

    if (choice == 2)
    {
        Console.Write("type rectangle length --> ");
        sidel = int.Parse(Console.ReadLine());
        Console.Write("type rectangle width --> ");
        side2 = int.Parse(Console.ReadLine());
        DrawRectangle(sidel, side2);
    }

    if (choice == 3)
    {
        Console.Write("type trianble base --> ");
        sidel = int.Parse(Console.ReadLine());
        DrawTriangle(sidel);
    }
}
}

```

## סוף פתרון גליה 2

### הפעולה Main()

מבט נוסף בתכנית, מראה שגם הפעולה הראשית Main() היא פעולה סטטית המקבלת פרמטר ומחזירה void. ההבדל בינה לבין הפעולות האחרות:

- הפעולה Main() חייבת לקבל הרשאת גישה public כדי להיות מופעלת.
- כאשר מריצים את התכנית, המחשב מחפש את הפעולה Main() וממנה תתחיל הריצה.

### 3. פעולה המקבלת פרמטרים ומחזירה ערך

בעיה 3 - שאלה 9 בגרות 2000

מטרת פתרון הבעיה: הדגמת השימוש בפעולות המחזירות ערך, שימוש במסננת קלט.

נגדיר "משקל" של מספר תלת-ספרתי כסכום של מכפלת שתי הספרות הראשונות של המספר ושל מכפלת שתי הספרות האחרונות שלו.

לדוגמא: ה"משקל" של המספר 327 הוא:  $3 * 2 + 2 * 7 = 20$

א. כתוב פעולה שתקבל מספר תלת ספרתי ותחזיר את ה"משקל" שלו.

ב. כתוב קטע תכנית (או פעולה) שיקלוט מספרים תלת-ספרתיים ויחשב עבור כל מספר את ה"משקל" שלו, באמצעות הפעולה שכתבת בסעיף א'.

קטע התכנית יחשב וידפיס את סכום ה"משקלים". הקלט יסתיים כאשר סכום ה"משקלים" יהיה גדול מ-100.

#### פירוק הבעיה לתת-משימות

1. פעולה המקבלת מספר תלת ספרתי ומחזירה את המשקל שלו.
2. קלט מספרים תלת-ספרתיים, חישוב והדפסת משקלו של כל מספר.
3. הדפסת סכום המשקלים.

#### פירוק מעמיק יותר של הבעיה לתת-משימות

1. פעולה המקבלת מספר תלת ספרתי ומחזירה את המשקל שלו.  
פעולת עזר: פעולה המקבלת מספר דו-ספרתי ומחזירה את מכפלת ספרותיו.
2. קלט מספרים תלת-ספרתיים סיכום והדפסת משקלו של כל מספר.  
פעולת עזר: פעולה המחייבת קליטת מספר תלת-ספרתי חיובי, ומחזירה אותו.
3. הדפסת סכום המשקלים.

#### בחירת משתנים

- num – מספר שלם, שומר את המספר התלת ספרתי.  
sum – מספר שלם, שומר את סכום המשקלים.

#### האלגוריתם

1. אפס את הסכום
2. כל עוד סכום המשקלים קטן או שווה ל-100 בצד:  
2.1 קלוט מספר גיא ספרתי במשתנה num

פעולה המחזירה ערך:

```
//--- תיעוד הפעולה ---
static טיפוס-הערך-המוחזר (רשימת פרמטרים) שם-הפעולה
{
    // גוף הפעולה
    return ערך להחזרה ;
}
```

**טיפוס הערך המוחזר** – כאמור, פעולות יכולות להחזיר ערך. ללא מידע לגבי הערך המוחזר, לא נדע כיצד להפעיל את הפעולה בהצלחה. עד כה, ראינו פעולות שלא מחזירות ערך (void), ועתה נכיר פעולות שמחזירות ערך כלשהו, למשל מספר שלם או ממשי, ערך בוליאני או תו. המשמעות היא שאם מוחזר ערך שאינו void, התכנית המזמנת תציין מה לעשות עם הערך המוחזר. הערך חוזר אל השורה שהתבצע בה הזימון לפעולה.

**משפט return** – החזרת ערך מפעולה מתבצעת באמצעות משפט return. טיפוס הערך המוחזר מוגדר בכותרת הפעולה.

אם נקבע כי הפעולה מחזירה ערך, יוחזר הערך באמצעות משפט return.  
אם נקבע כי הפעולה אינה מחזירה ערך, לא יופיע משפט ה-return בגוף הפעולה.

כאמור, במשפט זימון לפעולה המחזירה ערך, יש לציין מה לעשות עם הערך המוחזר (השמה למשתנה, הדפסתו, בדיקה אם הוא מקיים תנאי מסויים או העברתו כארגומנט לפעולה אחרת). אם מתבצעת השמה למשתנה הרי טיפוס המשתנה המקבל את הערך המוחזר חייב להיות תואם את טיפוס הערך המוחזר.

אם נקבע בכותרת הפעולה שהיא מחזירה ערך, חובה להחזיר ערך במשפט return.

כדי לחשב את משקל המספר, יש לחשב פעמיים מכפלה של מספר דו-ספרתי. פעם אחת את מכפלת שתי הספרות השמאליות של המספר (מאות ועשרות) ופעם שנייה את מכפלת שתי הספרות הימניות של המספר (עשרות ואחדות):

```
//--- פעולה המקבלת מספר דו-ספרתי ---
//--- ומחזירה את מכפלת ספרותיו ---
static int MultiplyDigits(int num)
{
    int d1 = num / 10;
    int d2 = num % 10;
    return d1 * d2 ;
}
```

## הדגשים

- הפעולה הצהירה במשפט הכותרת כי היא מחזירה מספר שלם. המספר מוחזר בהוראת return.
- הפעולה מניחה כי המספר שהתקבל הוא מספר דו-ספרתי, ואין זה מתפקידה לוודא זאת. תפקידה של התכנית המשתמשת בפעולה לבדוק ולהבטיח את קיומו של התנאי, כלומר לשלוח כפרמטר מספר דו-ספרתי תקין.

## משפטי זימון אפשריים לפעולה MultiplyDigits()

השמת הערך המוחזר במשתנה:

`int mult = MultiplyDigits(37);`

שילוב הערך המוחזר בבדיקת תנאי לוגי:

`if (MultiplyDigits(num) > x) ... (הוראה) ...;`

הדפסת הערך המוחזר מהפעולה:

`Console.WriteLine("Multiply Digits : " + MultiplyDigits(2*b + 3));`

שליחת הערך המוחזר מהפעולה כפרמטר לפעולה אחרת:

`double x = Math.Sqrt(MultiplyDigits(m));`

שים ♥ : הארגומנטים (הערכים הנשלחים) לפעולה יכולים להיות קבועים מספריים, שמות של משתנים, ביטויים חישוביים או הערך המוחזר מפעולה אחרת. תחילה יחושב ערך הביטוי או הפעולה שבסוגריים ורק אחר כך תתבצע הפעולה.

"משקלו" של מספר מוגדר כסכום המכפלות של שתי הספרות השמאליות ושל שתי הספרות הימניות במספר תלת-ספרתי, ולכן נוכל להגדיר את הפעולה באופן הבא:

```
static int NumWeight(int num)
{
    int left = MultiplyDigits(num / 10);
    int right = MultiplyDigits(num % 100);
    return left + right;
}
```

או באופן הבא:

```
static int NumWeight(int num)
{
    return MultiplyDigits(num / 10) + MultiplyDigits(num % 100);
}
```

התכנית אמורה לקלוט סדרה של מספרים תלת-ספרתיים ולחשב את משקלם. כדי להיות בטוחים שהמספרים שייקלטו יהיו מספרים תלת-ספרתיים, ניצור מסננת קלט שתחזיר מספרים מתאימים.

מסננת קלט היא קטע קוד לקליטת ערך העונה לקריטריון מסוים. הדרישה שלנו היא לקליטת מספר תלת-ספרתי חיובי, כלומר – מספר שנמצא בין 100 ל-999. מכאן שמספר שאינו עונה לקריטריון הוא מספר הקטן מ-100 או מספר הגדול מ-999.

## האלגוריתם למסננת הקלט

1. קלט מספר בגודל המסלול `num`.

2. כן אם `num` קטן מ-100 או `num` גדול מ-999.

2.1 קלט מספר בגודל המסלול `num`

נכתוב פעולה בשם GetNum שתחזיר מספר תלת-ספרתי כנדרש:

```
static int GetNum()
{
    int num;

    Console.WriteLine("type a positive 3 digits number --> ");
    num = int.Parse(Console.ReadLine());
    while (num < 100 || num > 999)
    {
        Console.WriteLine(" Error ! ");
        Console.WriteLine("type a positive 3 digits number --> ");
        num = int.Parse(Console.ReadLine());
    }
    return num;
}
```

הערה: בתכנית הלימודים לא נדרשת מסננת קלט, אלא אם נכתב במפורש שחובה לבדוק את תקינות הקלט.

### התכנית המלאה המחשבת את סכום המשקלים

```
/**
 *      "משקל" של מספר - סכום מכפלת שתי הספרות הראשונות
 *      ומכפלת שתי הספרות האחרונות
 *      א. פעולה המקבלת מספר תלת ספרתי ומחזירה את משקלו
 *      ב. תכנית הקולטת מספרים תלת-ספרתיים ומדפיסה את
 *          סכום המשקלים עד שיתקבל סכום גדול מ-100
 *
 */
public class T9_2000
{
    //--- פעולה המקבלת מספר דו-ספרתי ---
    //--- ומחזירה את מכפלת ספרותיו ---
    static int MultiplyDigits(int num)
    {
        int d1 = num / 10;
        int d2 = num % 10;
        return d1 * d2 ;
    }

    //--- פעולה המקבלת מספר תלת-ספרתי ---
    //--- ומחזירה את "משקלו" עפ"י ההגדרה ---
    static int NumWeight(int num)
    {
        return MultiplyDigits(num / 10) + MultiplyDigits(num % 100);
    }

    //--- פעולה המחזירה מספר תלת-ספרתי חיובי ---
    static int GetNum()
    {
        int num;

        Console.WriteLine("type a positive 3 digits number --> ");
        num = int.Parse(Console.ReadLine());
        while (num < 100 || num > 999)
        {
            Console.WriteLine(" Error ! ");
            Console.WriteLine("type a positive 3 digits number -->");
        }
    }
}
```

```
        num = int.Parse(Console.ReadLine());
    }
    return num;
}

//--- התכנית הראשית ---
static void Main()
{
    int num;
    sum = 0;

    while (sum <= 100)
    {
        num = GetNum();
        sum = sum + NumWeight (num);
    }
    Console.WriteLine("total weight: " + sum);
}
}
```

סוף פתרון בעיה 3

## 4. פעולות על מערכים

קצ'ה 4 – שאלה 7 בגרות 2000

מטרת פתרון הבעיה: הדגמת השימוש בפעולות על מערכים.

נתון מערך חד-ממדי בגודל 80 המכיל מספרים. כתוב תכנית שתקלוט נתון למשתנה k. התכנית תבדוק אם סכום k האיברים הראשונים במערך גדול מסכום שאר איברי המערך. אם כן – תדפיס את סכום k האיברים הראשונים במערך. התכנית תבדוק את תקינות הקלט k שיתאים לתנאי הבעיה.

מערך הוא עצם. הכרזה על מערך בתכנית יוצרת הפנייה לעצם מסוג מערך. בניית מחלקה ובה תכונה מסוג עצם חורגת מנושאי תכנית הלימודים, ולפיכך יופעלו על המערך פעולות סטטיות.

### פעולה המקבלת מערך כפרמטר

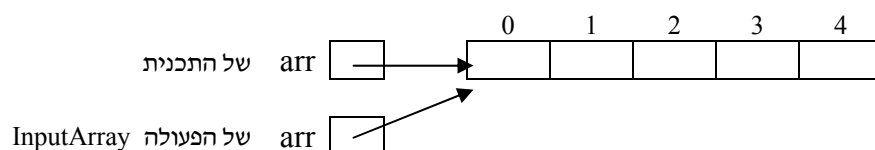
כאשר אנו שולחים מערך כפרמטר לפעולה, אנו בעצם שולחים את ההפניה למערך עצמו, ולכן, כל שינוי במערך בתוך הפעולה משפיע גם על המערך שבתכנית המזמנת את הפעולה.

### פעולה הקולטת ערכים לתוך מערך

הפעולה מקבלת מערך כפרמטר וקולטת ערכים לאיבריו. מכיוון שמועברת הפנייה למערך עצמו, כל שינוי בערכי המערך בתוך הפעולה ישנה את ערכי המערך שהפנייתו הועברה כפרמטר לפעולה. לכן אין החזרת ערך מהפעולה ולפיכך אין בפעולה משפט return.

```
static void InputArray(int[] arr)
{
    for (int i = 0 ; i < arr.Length ; i++)
    {
        Console.WriteLine("type number " + i + " --> ");
        arr[i] = int.Parse(Console.ReadLine());
    }
}
```

האיור הבא ממחיש את דרך העברת המערך כפרמטר לפעולה:



## פעולה המחזירה ערך מסוג מערך

את המערך נוכל להגדיר ב-main אך נוכל גם לכתוב פעולה שתקלוט מהמשתמש את גודל המערך, תיצור מערך חדש ותקלוט בתוכו ערכים בלולאה פנימית או באמצעות הפעולה InputArray ולבסוף תחזיר את ההפניה למערך זה:

```
//--- פעולה המחזירה מערך מאותחל ומלא בערכים ---  
static int[] GetArray()  
{  
    Console.WriteLine(" Type array size --> ");  
    int n = int.Parse(Console.ReadLine());  
  
    int [] arr = new int[n];  
  
    for (int i = 0 ; i < arr.Length ; i++)  
    {  
        Console.WriteLine("type number " + i + " --> ");  
        arr[i] = int.Parse(Console.ReadLine());  
    }  
  
    return arr;  
}
```

אפשר להמיר קטע זה  
במשפט הזימון:  
InputArray(arr);

משפט הזימון לפעולה, מתוך התכנית, יהיה: `int [] arr = GetArray();`

הערה: שאלות בגרות רבות, העוסקות במערכים, מניחות כי המערך נתון, ולכן אין צורך בפעולות הקלט. בפתרון הבחינה נוכל להסתפק בכתיבת השורה: `int [] arr = GetArray();` המגדירה את המערך הנתון וקובעת את שמו בתכנית. מובן שאם נרצה להריץ את התכנית, נצטרך לממש את הפעולה `GetArray`.

## האלגוריתם לפתרון הבעיה

- (1) הפזרי את `arr` כמערך של  $n$  אלמנטים בגודל 80 וקלוט בגובה מספרים שלמים
- (2) קלוט  $k$ - מספר שלם בין 0 לגודל המערך (כולל)
- (3) גשב `sum1` את סכום  $k$  האיברים הראשונים במערך
- (4) גשב `sum2` את סכום האיברים ממקום  $k$  ועד סוף המערך
- (5) אם הערך של `sum1` גדול מהערך של `sum2` הדפס את `sum1`

הערה:

- יכולנו להחליף את שורות 3-5 באלגוריתם בהוראה:

(3) אם סכום  $k$  האיברים הראשונים גדול מסכום שאר האיברים  
הדפס את סכום  $k$  האיברים הראשונים

דבר שהיה אולי קריא יותר, אבל פחות יעיל שכן היינו צריכים להפעיל את ההוראה המחשבת את סכום  $k$  האיברים הראשונים פעמיים.



```

public class T7_2000
{
    //--- פעולה המחזירה מספר שלם וחיובי ---
    //--- בתחום איברי המערך (0..n) ---
    static int GetNum(int n)
    {
        int num;

        do {
            Console.WriteLine("type a positive int between 0 and "
                + n + " --> ");
            num = int.Parse(Console.ReadLine());
        } while (num < 0 || num > n);

        return num;
    }

    //--- פעולה המחזירה מערך מאותחל ומלא בערכים ---
    static int[] GetArr()
    {
        Console.WriteLine(" type array size --> ");
        int n = int.Parse(Console.ReadLine());

        int [] arr = new int[n];
        for (int i = 0 ; i < arr.Length ; i++)
        {
            Console.WriteLine("type number " + i + " --> ");
            arr[i] = int.Parse(Console.ReadLine());
        }

        return arr;
    }

    //--- פעולה המחזירה את סכום איברי המערך החל ממקום from ---
    //--- ועד המקום to (לא כולל) ---
    static int ArrSum(int [] arr, int from, int to)
    {
        int sum = 0;

        for (int i = from ; i < to ; i++)
            sum = sum + arr[i];
        return sum;
    }

    public static void Main()
    {
        int [] arr = GetArr();
        int k = GetNum(arr.Length);

        int sum1 = ArrSum(arr, 0, k);
        int sum2 = ArrSum(arr, k, arr.Length);

        if (sum1 > sum2)
            Console.WriteLine("sum of first elements is: " + sum1);
    }
}

```

סוף פתרון בציה 4

## 5. מחלקת שירות

לאחר שהרצנו כמה תכניות המטפלות במערכים, גילינו שיש פעולות החוזרות על עצמן בכל תכנית. למשל – יצירת מערך, קלט למערך, הדפסת מערך, חיפוש במערך, סכום האיברים המערך וכד'.

פתרון אחד לתרגילים אלו יהיה לכתוב את הפעולות החוזרות הדרושות בכל תכנית ותכנית. הדרך הנכונה יותר תהיה לקבץ את כל הפעולות האלה בתוך מחלקה מיוחדת, שתספק את כל השירותים למערך, ושילובה של המחלקה בתכנית.

מחלקת שירות היא מחלקה המכילה אוסף של פעולות, שניתן להשתמש בהן ממחלקות אחרות. אוסף הפעולות במחלקת שירות מסוימת עוסק בדרך כלל באותו נושא. למשל - הכרנו את מחלקת השירות `Math` המכילה אוסף של פעולות המבצעות חישובים מתמטיים. נוכל לכתוב מחלקת שירות משלנו בשם `Array` שתספק פעולות המטפלות במערך חד-ממדי, או מחלקת שירות בשם `Matrix` המספקת פעולות המטפלות במערך דו-ממדי (מטריצה).

היתרון שביצירת מחלקת שירות, מעבר לחיסכון בכתיבה חוזרת של אותו קוד, הוא בכך שאנו משתמשים בפעולות שנבדקו בעבר וידוע שהן פועלות כהלכה.

מחלקת השירות נכתבת כמחלקה נפרדת. כדי שהתכנית תוכל להשתמש בפעולות של מחלקת השירות, יש לספק למהדר את מיקומה. בשלב זה נדאג שמחלקת השירות תישמר באותה תיקייה שנמצאת בה מחלקת התכנית.

בתוך המחלקה נגדיר את התכונות הדרושות למחלקה – אובייקט הקלט, ואת אוסף הפעולות המטפלות במערך. מכיוון שהפעולות תהיינה במחלקה `Array` ונשתמש בהן מתוך מחלקה אחרת, נקפיד להוסיף לכל פעולה הרשאת גישה `public`.

### יצירת מחלקת שירות לטיפול במערך חד-ממדי

```
public class Array
{
    //--- פעולה הקולטת מספרים לתוך מערך של שלמים ---
    public static void InputArray(int [] arr)
    {
        for (int i = 0 ; i < arr.Length ; i++)
        {
            Console.WriteLine("type number " + i + " --> ");
            arr[i] = int.Parse(Console.ReadLine());
        }
    }

    //--- פעולה הקולטת מספרים לתוך מערך של ממשיים ---
    public static void InputArray(double [] arr)
    {
        for (int i = 0 ; i < arr.Length ; i++)
        {
            Console.WriteLine("type number " + i + " --> ");
            arr[i] = double.Parse(Console.ReadLine());
        }
    }
}
```

```

//--- פעולה המקבלת מערך של מספרים שלמים ---
//--- ומדפיסה את ערכיו בשורה ---
public static void PrintArray(int [] arr)
{
    for (int i = 0 ; i < arr.Length ; i++)
        Console.Write(arr[i] + " ");
    Console.WriteLine ();
}

//--- פעולה המקבלת מערך של מספרים ממשיים ---
//--- ומדפיסה את ערכיו בשורה ---
public static void PrintArray (double [] arr)
{
    for (int i = 0 ; i < arr.Length ; i++)
        Console.Write(arr[i] + " ");
    Console.WriteLine ();
}
}

```

נוכל להוסיף פעולות נוספות לפי הצורך.

## העמסת פעולות

במחלקה הוגדרו שתי פעולות בשם `InputArray` ושתי פעולות בשם `PrintArray`. מצב שבו יש כמה פעולות באותו שם, הנבדלות זו מזו בשורת הפרמטרים, נקרא **העמסת פעולות** (method overloading). פעולה אחת מקבלת כפרמטר מערך של מספרים שלמים, והפעולה האחרת מקבלת מערך של מספרים ממשיים. המהדר ידע לזהות את הפעולה המתאימה לפי סוג המערך המועבר.

## תכנית המשתמשת בפעולות המוגדרות במחלקת השירות Array

```

public class ArrCheck
{
    public static void Main()
    {
        int [] intArr = new int [5];
        double [] doubleArr = new double [7];

        Array.InputArray(intArr);
        Array.InputArray(doubleArr);

        Array.PrintArray(intArr);
        Array.PrintArray(doubleArr);
    }
}

```

כדי שהמהדר ידע שמדובר בפעולות של מחלקת השרות, נוסיף לכל משפט זימון את שם המחלקה שבה מוגדרת הפעולה, בדיוק כפי שעשינו כשהשתמשנו בפעולות של המחלקה `Math`:

`Array.InputArray(intArr);`

כדי להציג את איברי המערך, נציין שהפעולה `PrintArray` נמצאת במחלקה `Array`. שליחת המערך `intArr` כפרמטר לפעולה, תגרום לבחירת הפעולה המקבלת מערך של `int`.