

## שני מערכי שיעור (2 \* 90 דק') :

### א. "מחשבון פשוט"

### ב. "מחשבון כולל If מקונן ושימוש בפעולות"

## 1. כללי

---

### א. סביבת עבודה

- נעבוד בסביבה של Window Application. סביבה המכילה בקירבה עצמים הניתנים לשימוש מייד, תוך כדי גרירה בלבד, וזאת בלי דרישה של ידע מוקדם.
- סביבה זו מאפשרת לנו להבין בקלות את המהות של עבודה עם עצמים.
- בסביבה זו ניתן לבנות בקלות תוכניות (משחקים) שיגרמו לתלמידים לאהוב את עולם התיכנות ולהתקרב אליו.
- הכוונה לעורר בתלמידים אלו את הרצון להמשיך ולבחור במגמה זו לאחר מכן, אי לכך חשוב שהשיעור יועבר בצורה מעניינת ומגרה אך לא בצורה קשה ומורכבת מדי ....

### ב. נושאי השיעורים

- הכרה ועבודה עם משתנים מסוגים שונים והמרת ערכים מסוג אחד למשנהו
- תירגול נוסף של הוראת התנאי If כולל מצב של תנאי מקונן
- שימוש במחלקת Math לחישוב פעולות מורכבות.

### ג. ידע מוקדם

- יש צורך להכיר את סביבת העבודה וכן שימוש בפקדים בסיסיים + משתנים בסיסיים, כלומר זה שיעור המשך למערך שיעור (כפול) הכולל היכרות + משימת הרמזור.

## ד. מטרות השעור

- התלמיד ילמד לעבוד בסביבת window Application
- התלמיד ילמד לעבוד בסביבת EvenDriven, תכנות מונחה אירועים.
- התלמיד ילמד לעבוד עם עצמים, כמו כפתור, ועם התכונות שלו (כגון צבע).
- התלמיד יכיר בצורך בשימוש בהוראת תנאי, כולל תנאי מקונן.
- התלמיד יכיר בצורך בשימוש במשתנים מסוגים שונים
- התלמיד ילמד לעבוד עם משתנים מסוג string, char, double
- התלמיד יכיר בייחודיות של טיפוסים המשתנים השונים

## 2. מהלך השיעורים מחשבו:

### א. מבנה השיעורים:

#### שיעור מחשבון פשוט (1#)

- השיעור בכללותו הוא באורך שתי שעות הוראה פרונטליות (2 \* 45 דקות).
- הצגת משימת מחשבון : (זמן משוער 5 דקות)
- הוראת סוגי משתנים string + char והמרות משתנים : (זמן משוער 20 דק)
- תרגול עצמי, התחלה של בניית מחשבון : (זמן משוער 20 דקות)
- הוראת הודעות שגיאה ותרגול : (זמן משוער 3\*15 = 45 דקות).

#### שיעור מחשבון (2#), כולל שימוש ב-if מקונן ובפעולות

- השיעור בכללותו הוא באורך שתי שעות הוראה פרונטליות (2 \* 45 דקות).
- חזרה על משימת מחשבון כולל שימוש בהודעות שגיאה : (זמן משוער 10 דקות)
- הוראת טיפוס משתנה double : (זמן משוער 10 דקות)
- תרגול עצמי – הוספת פעולת חילוק : (זמן משוער 25 דקות)
- סיכום בייניים : (זמן משוער 5 דקות)
- הוראת שימוש בפעולות : (זמן משוער 20 דק)
- תרגול עצמי : (זמן משוער 20 דקות)
- סיכום : (זמן משוער 5 דקות)

## שיעורי המשך אופציונאליים במחשבון :

בשעורים אלו נחזור ונחזק את השימוש בפעולות ובמשתנים השונים וזאת ע"י הוספת אופציות ורכיבים למחשבון אותו התחלנו:

נשתמש ברכיב radio לבחירת הפעולה במחשבון, ונחזק את הפעולות

נוסיף שימוש בפעולות % וחילוק של שלמים מול חילוק שנותן תוצאה מממטית.

נשתמש ברכיב comboBox לבחירת הפעולה של המחשבון ובכך נחזק את השימוש ב-if מקונן

נמיר את השימוש ב-if מקונן בפעולת switch

נוסיף למחשבון אופציות המשתמשות בפעולות סטנדרטיות של מחלקת Math כמו : Max, Min, Pow,

Log, Round, Sqrt ועוד ככול העולה על רצוננו

ניתן גם להוסיף פעולות כמו חישוב היקף ושטח של עיגול אשר משתמשים בקבוע  $\pi$  (פאי) ממחלקת Math.

נהפוך את המחשבון ללומדה:

נגריל (ע"י Random) מספרים (בטווח של 1 ל 100, למשל) שייכנסו בתיבות הטקסט המשתמש יכניס את התוצאה בתיבת טקסט ואנו נשווה את התוצאה לערך אותו כבר חישבנו, בתרגיל המחשבון, (ונמצא בתיבת טקסט שאינה מוצגת למשתמש) המשתמש ייקבל הודעה "צדקת" או "טעית" בהתאם. ניתן להוסיף גם מונה של הצלחות / כשלונות ולתת למשתמש ציון בהתאם.



## ב. משימת מחשבון (#1):

במשימת המחשבון המשימה שלנו תהיה לתכנן, לעצב ולתכנת מחשבון בסיסי, כלומר מחשבון המכיל את הפעולות: חיבור, חיסור וכפל. כל פעם המשתמש של המחשבון יכניס שני מספרים, ילחץ על כפתור הפעולה הרצויה, ואנחנו נבנה את המחשבון כך שהלחיצה תגרום להופעת תשובת הביטוי המתמטי המתאים. טופס העבודה שלנו ייראה כך:

המשימה היא כזאת:

1. על המשתמש להכניס שני מספרים שלמים (מטיפוס int) לשתי תיבות-הטקסט שבראש הטופס.
2. לאחר מכן המשתמש יבחר בפעולה שהוא רוצה להפעיל על שני המספרים שהכניס, וילחץ על הכפתור המתאים.
3. תשובת הביטוי המתמטי שרצה התלמיד לפתור תופיע בתיבת-טקסט שבסוף הטופס.
4. **הערות**
  - תוויות ינחו את המשתמש בטופס שלנו. בעזרתן הוא יידע היכן עליו להכניס את המספר הראשון ואת המספר השני, וכן היכן עליו לצפות לקבל את התשובה.
  - נרצה להודיע למשתמש ולהסב את תשומת ליבו, בדרך זו או אחרת, במקרה שבו הוא הכניס טקסט שאינו מספר שלם במקום שבו אנו מצפים לקבל ממנו מספר כזה. במידה שכך קרה, נרצה שלא יוצג דבר בתיבת התוצאה, כלומר שהיא תיוותר ריקה.
  - נדאג לכך שהמשתמש לא יוכל לשנות את התוצאה שחישבנו עבורו. כלומר, אם הוצגה למשתמש תוצאה, הוא לא יוכל להחליפה בטקסט אחר אלא רק להעתיקה או לסמנה.
  - צבע תיבת-הטקסט של התוצאה שונה באופן אוטומטי על-ידי המערכת מלבן לאפור. הסיבה לכך היא הנקודה השלישית לעיל, ונגיע לכך ביתר פירוט בהמשך.

## כיצד נבצע את כל הדברים הללו?

נלמד על:

1. טיפוסים המשתנים string (מחרוזת) ו-char (תו)
2. שימוש בסוגריים בחישוב של ביטויים
3. תיבת טקסט - TextBox
4. Labels - שימושן במחשבון

5. המאפיין ReadOnly
6. קלט (Input) והמרתו לטיפוס שונה
7. try ו-catch (וקצת על Exceptions)
8. ספק שגיאות - ErrorProvider
9. MessageBox
10. פעולות - Methods

## טיפוסי המשתנים string (מחרוזת) ו-char (תו)



### תווים (char)

תו הוא אות, ספרה או סימן. קיימים תווים הנראים על המקלדת, וקיימים תווים נוספים (לדוגמה, לב) שאינם נמצאים על המקלדת. יש לציין שגם רווח (שמופיע במקלדת) נחשב לתו בפני עצמו. תו בודד מוגדר על ידי הטיפוס char. קבוע תווי מפורש יש לתחום בגרש משני צידיו. לדוגמה: '#'.  
לכל תו קיים ערך מספרי קבוע מראש הנקרא Unicode.

### מחרוזות (string)

מחרוזת היא רצף של תווים. רצף תווים מוגדר על ידי הטיפוס string. קבוע מפורש של מחרוזת יש לתחום משני צידיו במירכאות משני צידיו.

לדוגמה: "C# is a programming language".

לתווים השונים במחרוזת מסוימת יש מספרים סידוריים (אינדקסים) שהם מספרים שלמים החל מ-0 ועד למספר השלם הקטן ב-1 מאורך המחרוזת. לדוגמה, במחרוזת "Hello" האינדקס התו 'H' הוא 0, האינדקס של התו 'o' הוא 4, וכו'. אם רוצים לפנות לתו בודד במחרוזת, ניתן לעשות זאת על-פי האינדקס שלו בצורה הבאה:

**[ אינדקס ] המחרוזת** ;

למקום שבו נכתוב הוראה כזו יחזור תו (מטיפוס char, כמובן) שנוכל לשמור אותו במשתנה ולעשות בו שימוש. נעיר שבמקום של "המחרוזת" יכול להופיע קבוע מפורש מטיפוס string, משתנה מטיפוס string, או ביטוי שטיפוסו הוא int.  
לדוגמה:

```
string str = "world!";  
char index2 = str[2];  
this.label1.Text = index2.ToString();
```

כתוצאה מהקוד הזה התו שנמצא במקום מס' 2 במחרוזת, כלומר התו 'r', יהיה למעשה הערך של מאפיין ה-Text של התווית label1. במילים אחרות, נראה שהטקסט בתווית הוא r.  
שימו לב שפעולת התרגום למחרוזת חיונית משום שטיפוס המאפיין Text הוא string ולא char.

## שימוש במחרוזות ובתווים – לשם מה?

בתהליך פתרון בעיות יש לעיתים צורך להשתמש בתווים שהם רצף של תווים (כלומר מחרוזת). למשל: שמות למיניהם.

כמו כן, יש לא אחת צורך להשתמש בתווים שהם תו בודד. למשל: יום בשבוע (יום א', יום ב', וכו').

### פעולת השרשור

שרשור היא פעולה שמטרתה חיבור מספר מרכיבים לכדי מחרוזת אחת. הפעולה תבצע בעזרת סימן הפלוס (+). לסימן הפלוס יש, אם כן, שתי משמעויות: חיבור (עבור נתונים מספריים) ושרשור (עבור נתונים מסוג מחרוזת).



### הערות

- האופרטור + מייצג שרשור רק כאשר מעורבת מחרוזת (אחת לפחות).
- לביטוי מטיפוס char יש ערך מספרי (ערך ה-Unicode). לפיכך ניסיון להשתמש בסימן + כאשר בציודו האחד ביטוי מטיפוס char ובציודו האחר ביטוי נוסף שיש לו ערך מספרי (כלומר ביטוי מטיפוס double, int או char) ייתן חיבור מספרי ולא שרשור!

לדוגמה:

```
int a = 3;
string s = "The number of brothers I have is " + a;
```

ערך המשתנה s הוא "The number of brothers I have is 3" (האופרטור + מייצג כאן את פעולת השרשור).

### רצף ערכים מטיפוסים שונים המחוברים על-ידי הסימן +

במקרה כזה, נקבע את טיפוס הביטוי כולו ואת ערכו בשלבים. נבין זאת מתוך הדוגמה הבאה:

תלמיד מדעי המחשב רצה להרכיב את המחרוזת "100worlds" על-ידי שימוש באופרטור + ובמשתנים הבאים:

```
int num = 100;
char ch = 'w';
string word="orlds";
```

הוא כתב את השורה הבאה וציפה שכתוצאה ממנה במשתנה result תהיה המחרוזת המבוקשת:

```
string result = num + ch + word;
```

האם קרה כמצופה?, נבדוק זאת:

נפעל כך כדי לגלות את הטיפוס ואת הערך של הביטוי `num + ch + word`. חישוב ערך הביטוי מתבצע על-ידי חלוקה לתת-ביטויים משמאל לימין:

1. ראשית נבצע את הפעולה `num + ch`.

מדובר בחיבור של מספר שלם ותו בעל ערך Unicode של 'w' (ערך זה הוא 119), ולכן הפלוס מייצג חיבור מספרי. נקבל:  $100 + 119 = 219$ . `100 + ch = 100 + 'w' = 219`.

2. נבצע את הפעולה האחרונה: `219 + "ords"`. סימן הפלוס מייצג כאן שרשור, מכיון שאחד האופרנדים בפעולה הוא מחרוזת. לפיכך: `219 + "ords" = "219ords"`. כלומר, ערך הביטוי כולו הוא "219ords".

ו...בכתיבה מקוצרת :

```
result = num + ch + word =  
= 100 + 'w' + word =  
= 100+ 119 + word =  
= 219 + "orlds" =  
= "219orlds"
```

בסופו של דבר התלמיד יקבל במשתנה result את המחרוזת "219orlds" ולא את המחרוזת "100worlds" כפי שציפה.

כיצד נתקן את הביטוי שכתב את התלמיד כך שערכו הסופי יהיה אכן "100worlds"?

נוסיף **סוגריים** כך שראשית תבצע הפעולה "worlds" = "orlds" + 'w'.  
הפתרון הוא, אפוא:

```
string result = num + (ch + word);
```

בדקו שאכן ברור לכם מדוע ערך המשתנה result יהיה "100worlds" לאחר אתחול זה.

**נסכם**  **בהערה:**

חישוב ביטוי המורכב מערכים בעלי טיפוסים שונים המחוברים בסימן הפלוס יבוצע על-ידי חלוקת הביטוי לתת-ביטויים משמאל לימין עד סוף הביטוי. עם זאת, שימוש בסוגריים יכול לשנות את סדר החישוב כרצונכם.

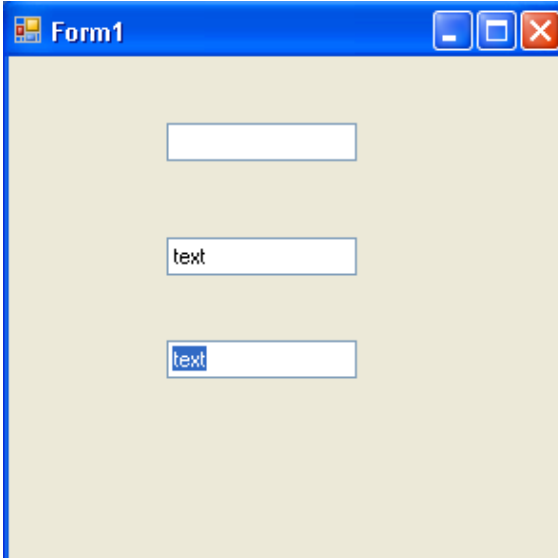
## תיבת טקסט - TextBox

הקומפוננט (המרכיב) העיקרי של Windows Forms שבו השתמשנו עד עכשיו היה ה-button (כפתור). כך עשינו במשימת הרמזור: לחצנו על כפתור אחד, וקרה דבר מסוים לכפתור אחד או לכמה כפתורים אחרים.

קעת אנו מוסיפים כלי חדש למשחק: ה-TextBox, תיבת-הטקסט. תיבת-הטקסט כשמה כן היא: היא תיבה שנועדה להכיל טקסט מסוים, כלומר אוסף כלשהו של תווים (מחרוזת).

ברור, אפוא, כמה ה-TextBox צפוי להיות שימושי: עד עכשיו התקשורת שלנו עם המשתמש באה לידי ביטוי בעיקר בכיתוב שעל הכפתורים ובכיתוב שעל התוויות, כלומר אנחנו המתכנתים העברנו למשתמש מידע; זו אמנם תקשורת, אך היא חד-צדדית ומוגבלת מאד. ה-TextBox יאפשר לנו לקבל מן המשתמש קטעי טקסט, ובכך להרחיב את התקשורת שלנו איתו. נוכל לעשות שימוש בטקסטים אלה וכיו"ב.

ה-TextBoxes נראים כך:



כפי שניתן לראות מהתמונה, ה- TextBox יכול להיות ריק (כלומר להכיל מחרוזת ריקה), יכול להכיל טקסט, והטקסט שבו יכול להיות מסומן. תיתכנה, כמובן, עוד כמה וכמה אופציות, אך הצגנו כאן את אלה הבסיסיות ביותר.

את ה- TextBox, כמו כל עצם/מרכיב/קומפוננט אחרת נגרור מן ה- Toolbox אל הטופס שלנו.

### **בהיבט התכנותי :**

לחיצה כפולה על TextBox מסוים בטופס (במקרה זה ה- Name של תיבת הטקסט הוא textBox1) תגרום להופעת פעולה שההוראות שבה יבוצעו במידה שהטקסט בתיבת-הטקסט ישונה. הפעולה תיראה כך :

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    // הוראות
}
```

לא נשתמש לעיתים קרובות בפעולה זו, אך נזכיר שניתן לבחור פעולה אחרת על-ידי לחיצה על הברק (Events) שבראש טבלת המאפיינים. לחיצה כפולה על הפעולה הרצויה תוביל לקטע הקוד הרצוי. נעיר שב- TextBox לא נשתמש בדרך-כלל בפעולות כפי שהסברנו כעת, אלא נשנה את המאפיינים שלו מתוך פעולות של מרכיבים אחרים כמו כפתורים וכו'.

ל- TextBox, כמו לכל עצם אחר, יש מאפיינים שונים בטבלת המאפיינים. במעמד זה נציין מאפיין מרכזי של ה- TextBox, והוא ה- **Text**. מאפיין זה מכיל בכל רגע נתון את הטקסט (המחרוזת) שנמצא בתיבת-הטקסט (נכללת האפשרות שבה התיבה ריקה). על מנת לדעת מהו הטקסט שמכילה התיבה נכתוב את הפקודה :

```
this.textBox1.Text
```

הפעולה מחזירה לנו את מחרוזת התווים, כלומר הטקסט, שנמצא בתיבה.

על מנת לשנות את הטקסט שבתיבה להיות למשל המחרוזת "This is C#", נכתוב את הפקודה :

```
this.textBox1.Text = "This is C#";
```

נציין מספר מאפיינים נוספים של עצם מסוג TextBox :

1. **המאפיין Enabled** : מאפיין מטיפוס bool. אם ערכו הוא true, תיבת הטקסט פעילה, ואז המשתמש יכול לשנות את הטקסט שלה, לסמן אותו וכו'. אם ערך המאפיין הוא false, תיבת הטקסט אינה פעילה, וכך לא ניתן למשתמש לעשות את הדברים שמנינו. כמו כן, צבע הרקע של התיבה ישונה באופן אוטומטי לאפור. ניתן לשנות את צבע הרקע באמצעות המאפיין הבא שזכיר.
2. **המאפיין BackColor** : צבע הרקע של תיבת הטקסט. מכיל צבע ממחלקת Color.
3. **המאפיין Visible** : מאפיין בוליאני. אם ערכו הוא true, ניתן לראות את תיבת הטקסט. אם ערכו הוא false, תיבת הטקסט תהיה בלתי-נראית.

### **במשימת המחשבון :**

נשתמש בתיבות-טקסט בעיקר כדי לקבל מידע מהמשתמש (יש לציין שניתן גם להציג מידע למשתמש באמצעות תיבות-הטקסט, אך נעשה את זה במקרים ספציפיים למדי). יש להכניס לתוך השיקולים גם את העובדה שניתן לסמן את הטקסט שבתיבת-הטקסט.

לכן, במשימה זו יהיה לנו נוח להשתמש ב- TextBoxes כדי לקבל מהמשתמש את המספרים שאותם הוא ירצה לחבר/לחסר/להכפיל.

## **Labels - שימושן במחשבון**

למדנו על תוויות בנושא משימת ההיכרות שלנו, ולמדנו על תפקידן להבנת התוכנית ביתר קלות.



## עתה נבדוק את התוויות מההיבט התכנותי :

לחיצה כפולה על Label (שה- Name שלה הוא label1) תגרום להופעת פעולה שהוראות שבה יבוצעו במידה שהתווית נלחצה. הפעולה תיראה כך :

```
private void label1_Click(object sender, EventArgs e)
{
    // הוראות
}
```

נציין שבדומה ל-TextBox, גם ב-Label לא נשתמש בדרך-כלל בפעולות כפי שהסברנו כעת, אלא נשנה את המאפיינים של ה-Label הרצוי בטופס מתוך פעולות של מרכיבים אחרים כמו כפתורים וכו'. לעצם מסוג Label, כמו לכל עצם אחר, יש מאפיינים שונים בטבלת המאפיינים. גם במקרה זה יש לציין את המאפיין Text. המשמעות והשימוש במאפיין זה נעשה בדיוק כפי שעשינו ב-TextBox (למעט, כמובן, שינויים קלים הנובעים מכך שאנו מתעסקים ב-Label ולא ב-TextBox). נציין גם את המאפיין Visible : אם ערכו true, התווית תהיה גלויה למשתמש ; ואם ערכו false, התווית תוחבא מעיני המשתמש.

### במשימת המחשבון :

נשתמש בתוויות בעיקר כדי להציג מידע למשתמש. במקרה זה התכונה שתיכנס לשיקולינו היא שלא ניתן לסמן את הטקסט שבתווית. תוויות נוחות מאד, אפוא, כדי להנחות את המשתמשים בטופס שלנו. במשימה זו יהיה לנו קל להיעזר ב-Labels כדי לסמן למשתמשים את המקומות שעליהם להכניס מידע (במקרה שלנו- המספרים שבביטוי המתמטי) וכן את המקומות שבהם יש יצפו לקבל מידע (במקרה שלנו- התשובה לתרגיל).

לדוגמה, נשים תווית מעל לתיבת-הטקסט שנועדה להכיל את התוצאה, ונכתוב בה : "Result".

## המאפיין ReadOnly

המאפיין ReadOnly הוא מאפיין השייך לעצם מסוג TextBox (אך לא רק לו). זה הוא מאפיין מטיפוס בוליאני, כלומר ערכו יכול להיות true או false בלבד. נדבר על מאפיין זה בהקשר של TextBox כעת. אם ערכו הוא false, אז ניתן למשתמש לשנות את הטקסט של תיבת-הטקסט. אם ערכו הוא true, אז לא ניתן למשתמש לעשות זאת.

על פי ברירת המחדל, ערך המאפיין ReadOnly הוא false. ואכן, אנו יודעים שאם נשים עצם מסוג TextBox בטופס שלנו מבלי לשנות את ערכי המאפיינים שלו, המשתמש יוכל לשנות את הטקסט שבו בזמן ריצת התוכנית. ראוי לציין שבאופן אוטומטי המערכת משנה את צבע הרקע של תיבת-הטקסט עם שינוי ערך המאפיין ReadOnly מ-true ל-false. צבע התיבה משתנה מברירת המחדל (לבן) לצבע אפור (ברירת המחדל לצבע הרקע של הטופס). שימו לב, ניתנת האפשרות להחליף את צבע התיבה, למרות שינוי ערך המאפיין ReadOnly, לכל צבע שתמצאו. שינוי הצבע יעשה על-ידי שינוי ערך המאפיין BackColor.

כמו-כן נציין כי ערך המאפיין ReadOnly (ככל מאפיין אחר) יכול להשתנות בזמן ריצה. למשל, אם יש לנו כפתור ותיבת-טקסט בטופס (שה- Names שלהם הם button1 ו- textBox1 בהתאמה), ולכפתור יש את הפעולה הבאה :

```
private void button1_Click(object sender, EventArgs e)
{
    this.textBox1.ReadOnly = true;
}
```

}

אז לחיצה על הכפתור תשנה את ערך המאפיין ReadOnly של תיבת-הטקסט ל-true.

### **במשימת המחשבון :**

אנו רוצים מקום שבו תוצג למשתמש תוצאת הביטוי המתמטי שהוא מנסה לחשב. חשוב לנו שלמשתמש תהיה היכולת לסמן את התוצאה (במקרה שהוא ירצה להעתיק אותה למקום אחר). אולם עלינו לדאוג שלמשתמש לא תהיה היכולת לשנות את התוצאה, כלומר שהוא לא יוכל לכתוב דבר שונה במקום שבו הצגנו לו את התוצאה. ממבט ראשון, נראה שאולי תווית תתאים לכך, מפני שלא ניתן למשתמש לשנות באופן ישיר את הטקסט שבה. אולם אין היא מתאימה מפני שלא ניתן לסמן את הטקסט שבתווית. המרכיב שבו נשתמש יהיה תיבת-טקסט, מפני שבה ניתן לסמן את הטקסט. אך, כאמור, עלינו לדאוג שלא תינתן למשתמש האפשרות לשנות את הכתוב (התוצאה). איך נעשה זאת? נשנה את המאפיין ReadOnly של ה-TextBox שלנו ל-true.

### **מדוע נשתמש במאפיין ReadOnly ולא במאפיין Enabled שכבר למדנו ואנו מכירים?**

נצביע על נקודות הדמיון ועל נקודות השוני בין שני המאפיינים :

נקודת דמיון : לא ניתן למשתמש לשנות את הטקסט שבתיבת הטקסט.

נקודות שוני : ReadOnly מאפשר למשתמש לסמן את הטקסט שבתיבה, ואילו Enabled לא מאפשר זאת.

במחשבון אנו דורשים שהמשתמש לא יוכל לשנות את התוצאה, ושהוא יוכל לסמן אותה. לכן ReadOnly הוא המאפיין המתאים לצורך זה, ובו נשתמש.

## **קלט (Input) והמרתו לטיפוס שונה**



**קלט** - נתונים המגיעים אל תכנית שלנו מן המשתמש שלה.

במקרה שלנו, האמצעי העיקרי לקבלת מידע מהמשתמש בצורה של טקסט הוא כאמור ה-TextBox. מסיבה זו בחרנו במרכיב זה כדי לקבל מהמשתמש את המספרים שעליהם ירצה לבצע פעולה כלשהי.

כפי שצינו קודם לכן, בכל רגע נתון רצף התווים שבתיבת-טקסט (המחרוזת) שמור במאפיין Text של התיבה. רצף זה יכול להכיל כל תו שהוא, וכך כל הרצפים הבאים חוקיים: "48 is a number", "48", "is a number".

באופן טבעי, לא ניתן לבצע פעולות חשבון על טקסטים, כלומר אין משמעות מתמטית לביטוי "48"+"is a number", ואף לא לביטוי "84"+"48".

נדרשת, אם כן, פעולת תרגום/המרה כלשהי של טקסטים למספרים שלמים; ובאמת למחלקת המספרים השלמים, מחלקת int, קיימת פעולת תרגום/המרה של טקסטים (מחרוזות) למספרים שלמים. שם הפעולה הוא Parse.

נניח, לשם הדיון, שקיימת לנו textBox1 ושערך המאפיין Text שלה הוא טקסט שתוכנו מספר שלם כלשהו (נקבע למשל שערך ה-Text הוא "4").

אז, אם נרצה לתרגם/להמיר את הטקסט "4" למספר 4, ולהכניס את זה למשתנה מסוג int, נעשה זאת כך :

```
int number;  
number = int.Parse(this.textBox1.Text);
```

ניתן, כמובן, לאחד את שתי השורות הללו לכדי שורה אחת:

```
int number = int.Parse(this.textBox1.Text);
```

המשמעות היא שאנו מבקשים ממחלקת int לתרגם את ערך המאפיין Text של textBox1 לטיפוס int, ולהכניס את הערך המתורגם למשתנה number (שהוא מטיפוס int).



### שימו לב:

- ניתן לפעול בדיוק באותה דרך כדי להמיר טקסטים (מחרוזות) לטיפוסים אחרים (למשל double).
- חשוב מאד להבחין שהנחנו לפני התרגום שערך המאפיין Text של תיבת-הטקסט הוא טקסט שתוכנו מספר שלם כלשהו. אם ערך ה- Text היה איזשהו רצף תווים שהתוכן שלו אינו מספר שלם (למשל: " 48 is a number"), היינו מקבלים שגיאת ריצה בזמן התרגום של הטקסט למספר שלם.

### במשימת המחשבון:

אין כל ספק שהקלט יהווה חלק משמעותי מהמשימה הנוכחית. המשתמש יזין לתוכנית שלנו מספרים בתור קלט, ואנחנו נצטרך להמיר את הקלט הזה מטקסט (מחרוזות) ל-int כדי שנוכל לבצע את החישובים המתמטיים הנדרשים.

## המרות בין טיפוסים משתנים

לעיתים קרובות בתוכניות מחשב אנו זקוקים להכניס ערכים של משתנה מטיפוס אחד למשתנה מטיפוס אחר. על מנת לעשות זאת יש צורך בפקודת המרה מטיפוס אחד לאחר. להמרות יש כללים מוגדרים, ולכל המרה יש פקודה מתאימה ב-C#.

לפנינו 3 מבני ההמרה:

1. קבוע/משתנה/ביטוי\_להמרה (טיפוס\_רצוי) = שם\_משתנה ;
2. קבוע/משתנה/ביטוי\_להמרה) .Parse(טיפוס\_רצוי) = שם\_משתנה ;
3. קבוע/משתנה/ביטוי\_להמרה) .ToString() = שם\_משתנה ;

להלן טבלה המרכזת את כללי ההמרה (שימוש בכל סוג המרה) ואת פקודות ההמרה ב- C#:

<u>string</u>	<u>double</u>	<u>int</u>	טיפוס מקור / טיפוס רצוי
<p>ניתן לבצע המרה של טיפוס string ל- int על ידי פקודת ההמרה Parse.</p> <p><u>לדוגמה:</u></p> <pre>string a = "17"; int b = int.Parse(a);</pre> <p>מכך נובע ש- b = 17.</p> <p><u>שים לב!</u></p> <p>ההמרה תוכל להתבצע רק אם התוכן של המחרוזת הוא מספר שלם.</p>	<p>ניתן לבצע המרה של טיפוס double לטיפוס int ע"י ציון טיפוס המטרה בסוגריים, לפני המשתנה.</p> <p>במקרה זה יושם במשתנה מטיפוס int החלק השלם של ערך המשתנה הממשי. (צמצום של ערך מספרי)</p> <p><u>לדוגמה:</u></p> <pre>double x = 4.4; int y = (int)x;</pre> <p>מכך נובע ש- y = 4.</p>	<p><b>אין צורך בהמרה!</b></p> <p>תמיד ניתן לבצע השמה של ביטוי מטיפוס מסוים למשתנה מאותו הטיפוס.</p>	<u>int</u>
<p>ניתן לבצע המרה של טיפוס string לטיפוס double ע"י פקודת ההמרה Parse.</p> <p><u>לדוגמה:</u></p> <pre>string a = "12.1234"; double b = double.Parse(a);</pre> <p>מכך נובע ש- b = 12.1234.</p> <p><u>שים לב!</u></p> <p>נוכל לבצע את ההמרה רק אם התוכן של המחרוזת הוא מספר ממשי.</p>	<p><b>אין צורך בהמרה!</b></p> <p>תמיד ניתן לבצע השמה של ביטוי מטיפוס מסוים למשתנה מאותו הטיפוס.</p>	<p><b>אין צורך בהמרה!</b></p> <p>ניתן לבצע השמה ישירה ורגילה של טיפוס int לטיפוס double. (הרחבה של ערך מספרי)</p> <p><u>לדוגמה:</u></p> <pre>int x = 1; double y; y = x;</pre> <p>מכך נובע ש- y = 1.0 (מספר ממשי).</p>	<u>double</u>
<p><b>אין צורך בהמרה!</b></p> <p>תמיד ניתן לבצע השמה של ביטוי מטיפוס מסוים למשתנה מאותו הטיפוס.</p>	<p>ניתן להמיר double לטיפוס string ע"י הפעולה ToString.</p> <p><u>לדוגמה:</u></p> <pre>double a = 1.051; string b = a.ToString();</pre> <p>מכך נובע ש- b = "1.051"</p>	<p>ניתן להמיר int לטיפוס string ע"י הפעולה ToString.</p> <p><u>לדוגמה:</u></p> <pre>int a = 14; string b; b = a.ToString();</pre> <p>מכך נובע ש- b = "14"</p>	<u>string</u>

## דוגמה למשימה של המרה מ-string לטיפוס מספרי ולהיפך

**המשימה:** הגדר כפתור שתוכן הטקסט שלו יהיה מספר כלשהו (למשל: 14). לאחר לחיצה על הכפתור המספר שעל הכפתור יגדל ב-3.

נציג עתה דוגמה להמרה מטיפוס string לטיפוס int ולהיפך.

### שלבים לביצוע התוכנית:

1. נגרוך כפתור לטופס.
2. נגדיר את הערכים הראשוניים של המאפיינים לכפתור: נכתוב 14 במאפיין Text (שנו אתם את ערך המאפיין Name של הכפתור לשם משמעותי לפי הכללים).
3. **כתיבת הקוד** (פירוט מלא ראו בתוכנית המלאה הרשומה למטה)
  - א. נלחץ פעמיים על הכפתור, ואז יופיע לנו קובץ Form1.cs, בו נכתוב את הקוד.
    - ב. נגדיר משתנה x מסוג int במחלקה.
    - ג. מוצאים את הפעולה של button1, ובתוכה נכתוב את ההוראות הנדרשות למילוי המשימה.
    - ד. נמיר ל-int את הטקסט של הכפתור שהוא מסוג string (יכולנו להמיר רק מכיוון שתוכן הטקסט הוא מספר שלם). נוסיף לערך המומר 3 כנדרש, ונכניס את הערך החדש למשתנה x.

```
x = int.Parse(button1.Text) + 3;
```

ה. ניקח את הערך של x, נמיר אותו ל-string ונכניס אותו למאפיין Text של הכפתור:

```
button1.Text = x.ToString();
```

ההמרה לטיפוס string היא הכרחית משום שהטיפוס של המאפיין Text הוא string.

### קוד תוכנית המחשב במלואו:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        int x;
        private void button1_Click(object sender, EventArgs e)
        {
            x = int.Parse(button1.Text) + 3;
            button1.Text = x.ToString();
        }
    }
}
```

## try ו-catch (מיועד לתלמידים מתקדמים) <

כפי שאמרנו קודם לכן, המרת טקסט (מחרוזת) לכל טיפוס שהוא (במקרה שלנו- מספר שלם) תתבצע באופן תקין אם ורק אם התוכן של הטקסט מתאים לטיפוס המבוקש.

למשל, אם הטקסט הוא "15", אז ניתן לתרגם אותו ל-int; אך אם הטקסט הוא "fifteen" או "num15", ניסיון התרגום ייכשל.

ניתן דוגמה נוספת: את הטקסט "false" ניתן לתרגם ל-bool; ובטקסט "truefalse" או "true4" – התרגום ייכשל.

### מה קורה כשהתרגום נכשל?

בזמן שאנו כותבים את הקוד, המחשב והתוכנית – אין בכוחם לדעת שהטקסט שננסה לתרגם אינו "כשיר" לתרגום. כך, נגלה את השגיאה רק בזמן הריצה של התוכנית ולא בזמן כתיבתה. אם באמת תתגלה שגיאה שכזו, התכנית תיעצר ותופיע הודעת שגיאה. בקצרה: תתרחש כאן שגיאת ריצה. נשאלת השאלה - כיצד נעקוף אותה? התשובה טמונה בנושא הסעיף הזה: try ו-catch.

try ו-catch הן שתי מילים בשפה, והן מילים שמורות. כל אחת מהן פותחת בלוק.

## try

אנו נחליט, בהתאם לשיקולינו, אם ומתי להשתמש ב-try.

יש להבין שאנו המתכנתים שמים לב שהקוד שכתבנו יכול להוביל לשגיאת ריצה, אך אם לא תתרחש שגיאה כזו, הרי שבוצע בדיוק מה שרצינו.

לדוגמה: אם המשימה הייתה לתרגם טקסט (מחרוזת) למספר שלם אז היינו כותבים את פעולת התרגום. אנו יודעים שאם המחרוזת תהיה לא "כשירה" לתרגום, תופיע שגיאת ריצה. אבל אם שורת התרגום תרוץ כהלכה, משימתנו בוצעה.

אם כן, השימוש ב-try ובבלוק הקוד שמגיע אחריו מהווה מעין הצהרה: אנו, המתכנתים, מודיעים למחשב שבבלוק שייגיע אחרי המילה השמורה try עשויה להתרחש שגיאת ריצה. אנו בעצם מורים למחשב לנסות (ומכאן המילה try) לבצע את הקוד שבבלוק במלואו. אם הניסיון יצלח, הרי שהמשימה בוצעה כשורה.

סיכום ביניים: המחשב ינסה לבצע את הקוד שבבלוק ה-try. אם הקוד יעבוד וירוצ במלואו ללא שגיאות- מה טוב. אם לא, נלמד בהמשך כיצד להתמודד עם השגיאה בעצמנו במקום לתת למחשב לעצור את התוכנית.

נדגים את השימוש ב-try בדוגמה הבאה.

דוגמה: נניח שקיים טופס ובו כפתור, תיבת-טקסט ותווית.

המשימה:

1. המשתמש צפוי להכניס מספר שלם a לתיבת-הטקסט.
2. כשהכפתור יילחץ, נשים בתווית את המספר a+1.
3. אם המשתמש יכניס טקסט שתוכנו אינו מספר שלם וילחץ על הכפתור, נרשום בתווית את ההודעה: "Invalid input" (כלומר: "קלט לא תקין").

פתרון: הקוד שנכתוב יהיה כמובן קוד הפעולה של הלחיצה על הכפתור.

אנו מזהים שעשויה להתרחש שגיאת ריצה אם המשתמש יכניס לתיבת-הטקסט רצף תווים שתוכנו אינו מספר שלם (ואז התרגום ל-int ייכשל). מהסיבה הזו אנו משתמשים במנגנון של try ו-catch. כעת אנו כותבים את בלוק ה-try. **לצורך כתיבתו בלבד** אנו מניחים שהמשתמש הכניס טקסט "כשיר" לתרגום לתיבת-הטקסט. אם כן, קוד ה-try ייראה כך:

```
try
{
    int a = int.Parse(this.textBox1.Text);
    int b = a + 1;
    this.label1.Text = b.ToString();
}
```

הסבר לקוד: בשורה הראשונה תרגמנו את הטקסט שב-TextBox שלנו למספר שלם, והשמנו את הערך המתורגם למשתנה a (מטיפוס int); התרגום הצליח בגלל ההנחה שלנו שהטקסט היה מתאים לתרגום. בשורה השנייה, לשם נוחות ובהירות, השמנו את הערך a+1 למשתנה חדש b (גם מטיפוס int). בשורה השלישית המרנו את הערך של b למחרוזת והשמנו את הערך החדש למאפיין Text של ה-Label שלנו. ההמרה למחרוזת חיונית משום שהטיפוס של המאפיין Text הוא מחרוזת (string).

עד כאן לא חידשנו דבר למעט המילה try עצמה. הקוד שכתבנו נכתב מתוך הנחות שהוא ירוץ במלואו, ואת ההנחה הזו יכולנו לעשות גם קודם לכן, ללא השימוש במילה try ובבלוק שלה. נזכיר שהמטרה של try ו-catch הוא לטפל דווקא במקרים שבהם כן יש שגיאות, וזה הכוח שהמרכיב החדש הזה של השפה מקנה לנו המתכנתים. כעת נעבור לטיפול בשגיאות זמן הריצה.

## catch

אנו נשתמש במילה השמורה catch ובבלוק שלאחריה מיד לאחר סיום בלוק ה-try. בלוק ה-catch הוא שיטפל בשגיאות זמן ריצה אם אכן קרתה כזו בבלוק ה-try.

ברור שאם הגענו לבלוק ה-catch, ארעה שגיאת ריצה קודם לכן בבלוק ה-try. אם כן, ארעה שגיאה כזו, התכנית לא נעצרה, ואנחנו קיבלנו את הכוח לטפל בשגיאה. זה אך טבעי, לכן, שנקבל איזושהי דרך ממש להשתמש בשגיאה ולקבל מידע אודותיה.

כדי להקל עלינו את הטיפול בשגיאות, נכתבה מחלקה שנועדה בדיוק לכך, וזו המחלקה Exception.

## Exception

מחלקה זו כשמה כן היא- נועדה לייצג יוצאי דופן, חריגות ושגיאות בתוכנית. עצמים ממחלקה זו הינם שגיאות שונות, והם מספקים מידע על עצמם באמצעות המאפיינים שלהם. מאפיין חשוב, שבו נעשה שימוש בהמשך, הוא המאפיין Message שמחזיר מחרוזת שמייצגת את השגיאה.

ובכן, איך נקבל את המידע אודות השגיאה שהתרחשה בבלוק ה- try והובילה ל- catch?  
התשובה היא שיש סיבה שכותבי השפה בחרו דווקא במילה catch, והסיבה היא שמתאפשר לנו "לתפוס" את השגיאה שארעה. "נתפוס" את השגיאה על-ידי כך שנוסיף מיד לאחר המילה catch את הצירוף (כולל הסוגריים):  
(Exception ex)

כך תיראה השורה שלפני בלוק ה-catch:

```
catch (Exception ex)
```

נעת כל המידע שנרצה לקבל על השגיאה שהתרחשה נמצא במשתנה ex שהוא מסוג Exception.

 **שימו לב:**

אין הכרח להשתמש דווקא בשם ex: ניתן להשתמש בכל שם אחר כל עוד הוא חוקי (לפי הכללים לשם משתנה) ולא השתמשנו בו קודם לכן בקוד. באותה נשימה יש לציין שהשימוש במילים catch ו-Exception הוא כן מחייב, כמו גם הסוגריים.

לאחר כותרת ה-catch יבוא בלוק ה-catch שבו נעשה את כל העולה על רוחנו לטיפול בשגיאה שארעה. אנחנו, ואנחנו בלבד, נהיה אלה שיקבעו מה לעשות לאור השגיאה.

נמשיך לפתור את הדוגמה שבה נקבע שאם המשתמש הכניס מחרוזת שאינה מתאימה לתרגום, עלינו לשים בתווית את ההודעה "Invalid input".

אם כן, על קוד ה-catch להיראות כך:

```
catch (Exception ex)
{
    this.label1.Text = "Invalid input";
}
```

 **שימו לב:**

בבלוק ה-catch לא התייחסנו כלל לשגיאה עצמה (שנמצאת במשתנה ex). המטרה שלנו הייתה לעשות משהו לאור שגיאת ריצה שצפינו מראש, ולא דווקא להשתמש בשגיאה עצמה. כפי שעשינו בדוגמה הזו, פשוט כתבנו הודעת שגיאה בעצמנו בתווית ללא כל התייחסות למהות השגיאה ex. אין כל מניע בשפה לפעול בדרך שפעלנו. למרות זאת, בחלק מהמקרים כן נרצה להשתמש בשגיאה עצמה, וכך נעשה בהמשך.

נאחד את קטעי הקוד לכדי פתרון מלא של הדוגמה:

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int a = int.Parse(this.textBox1.Text);
        int b = a + 1;
        this.label1.Text = b.ToString();
    }
    catch (Exception ex)
    {
        this.label1.Text = "Invalid input";
    }
}
```



## ספק שגיאות - ErrorProvider

לאחר שלמדנו חומר תכנותי שאינו קשור באופן ישיר ל- WindowsFormsApplications אלא ל- C# באופן כללי, נעבור כעת לרכיב נוסף בטפסים: ErrorProvider (ספק שגיאות), שמטרתו הן:

- לאפשר לנו לסמן באופן ויזואלי שבמקום מסוים ארעה שגיאה.
- לאפשר לנו להציג מידע (בצורת טקסט) לגבי אופי השגיאה.

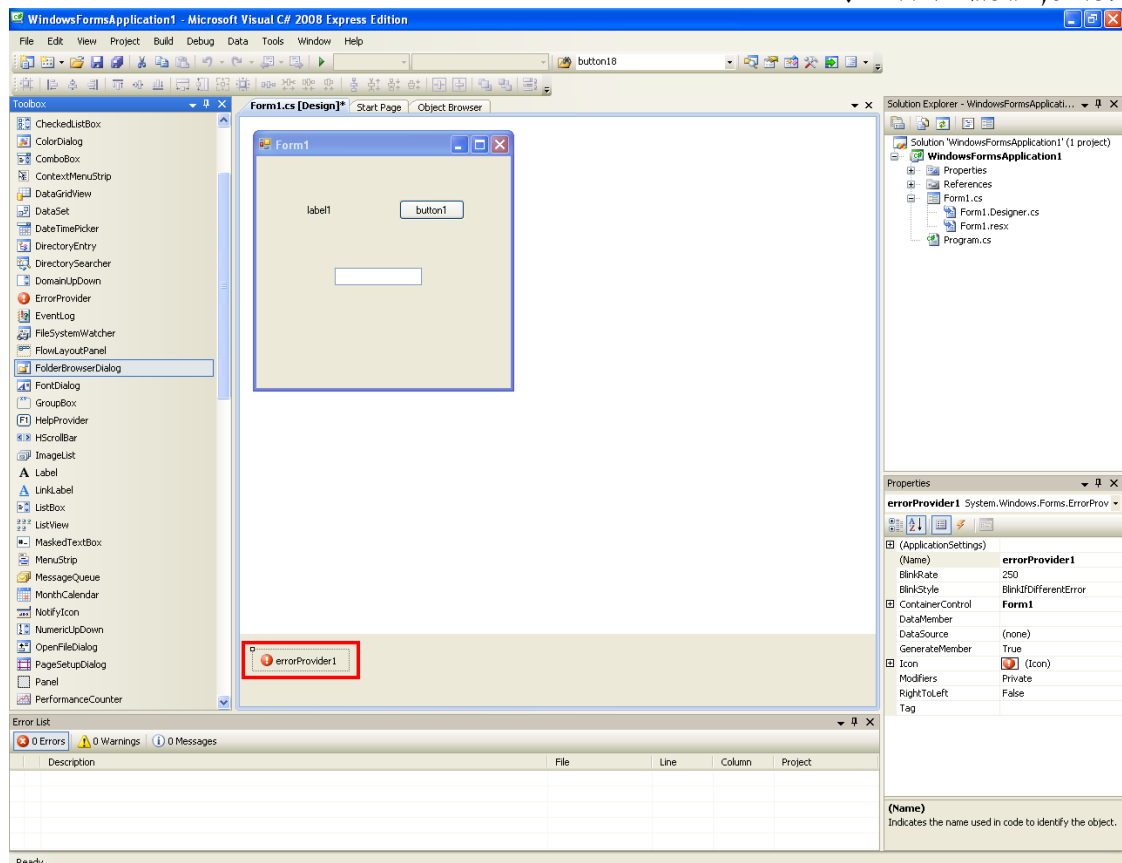
החידוש של ה- ErrorProvider הוא העובדה שהוא בולט מבחינה חזותית. קודם לכן כתבנו הודעת שגיאה משלנו ("Invalid input") בתוך תווית (למשל), ואין ספק שזה לא בולט ושההודעה הייתה עשויה לחמוק מעינינו. ה- ErrorProvider יפתור לנו בדיוק את הבעיה הזו.

את ה- ErrorProvider לא נוכל לגרור למקום מדויק בטופס, ויש לכך שתי סיבות עיקריות:

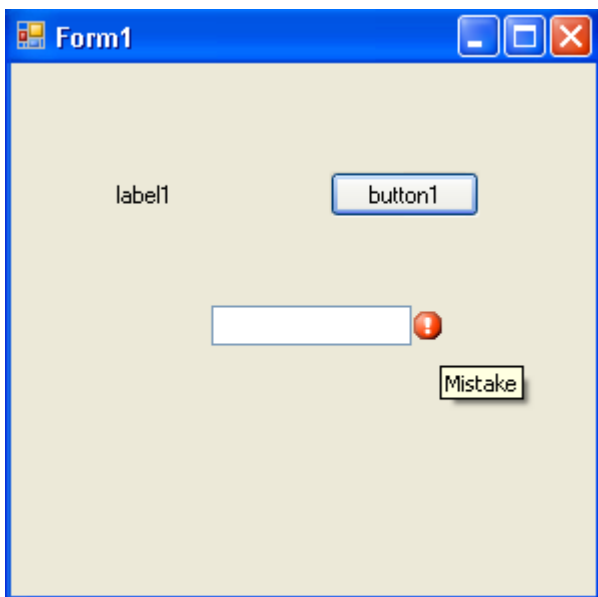
- בעת עיצוב הטופס אין אנו יודעים היכן תתרחש שגיאה. לא יהיה זה נכון, אם כן, להציב את ה- ErrorProvider במקום ספציפי מראש.
- יתכן שתתרחשנה למעלה משגיאה אחת. אם באמת היה עלינו לגרור את ה- ErrorProvider למקומות ספציפיים בטופס עבור כל שגיאה אפשרית, במקרה של שגיאות רבות העבודה הייתה נהפכת למסורבלת וללא-נוחה.

הפתרון הוא גרירת ה- ErrorProvider פעם אחת בלבד לטופס. הוא יהיה המקור שלנו לסימון כל שגיאה. נצטרך לספק לו, עבור כל שגיאה, את הרכיב בו ארעה השגיאה ואת המידע שנרצה שיוצג.

כשנגרור את ה- ErrorProvider לטופס שלנו, הדבר ייראה כך (המיקום שבו מופיע ה- ErrorProvider, מוחץ לטופס, מוסגר באדום !):



כעת, נניח שזיהינו שבתבית-הטקסט שבטופס יש בעיה. אם נשתמש ב- errorProvider כדי לסמן שבתביה יש שגיאה ונחליט שאנו רוצים שיוצג המידע "Mistake" לגבי השגיאה, הדבר יבוא לידי ביטוי כך בטופס הגמור:



בעת מעבר העכבר על סמל השגיאה



כשהעכבר אינו על סמל השגיאה

### בהיבט התכנותי :

מבנה השורה בקוד שתגרום להופעת סימן השגיאה עם הטקסט הרצוי ליד (קומפוננט) רכיב מסוים הוא (עבור עצם מסוג ErrorProvider שה- Name שלו הוא errorProvider1):

**this.errorProvider1.SetError(this.שבו\_השגיאה\_שם\_הרכיב\_שבו\_השגיאה, (מחרוזת\_תיאור\_השגיאה, שם\_הרכיב\_שבו\_השגיאה);**

למשל, השורה שגרמה לסימן השגיאה שבתמונות לעיל היא:

```
this.errorProvider1.SetError(this.textBox1, "Mistake");
```

### היכן נמקם את השורה הזו בקוד?

ראשית, עלינו להבין שמבחינת השפה מותר לנו למקם את השורה הזו בכל מקום שנרצה. אין הכרח למקם אותה באף מקום שהוא. אנחנו נחליט בעצמנו איפה להציב את השורה. אך עם זאת ניזכר שכבר למדנו דבר אחד או שניים לגבי שגיאות בכלל, ולגבי try ו-catch בפרט. כפי שאנו יודעים, אם הגענו לבלוק ה-catch, הרי שארעה שגיאה. לפיכך, קטע זה בקוד יהיה מקום "אסטרטגי" וטבעי למיקום השורה. בלוק ה-catch מקנה לנו בונוס נוסף: השגיאה בידינו! "תפסנו" אותה במשתנה (בשם ex לדוגמה). כך, המאפיין Message (של ex) - שמחזיר מחרוזת המייצגת את השגיאה - יוכל לשמש אותנו לצורך המידע שנציג אודות השגיאה שהתרחשה.

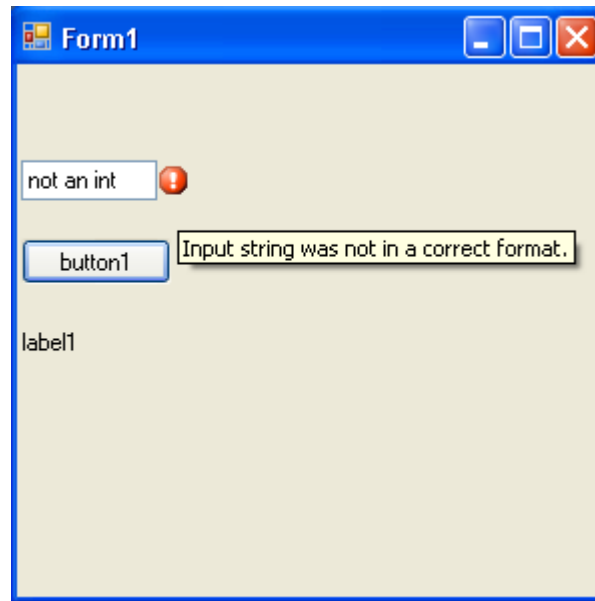
למשל, נחזור לדוגמה שבה השתמשנו בסעיף "try ו-catch". השינוי היחיד הוא שבמקום הופעת הטקסט "Invalid input" בתווית, יופיע סמל השגיאה של ה- ErrorProvider שלנו עם המידע מהמאפיין Message של ה- Exception שתפסנו.

קוד הפתרון ייראה כך:

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int a = int.Parse(this.textBox1.Text);
        int b = a + 1;
        this.label1.Text = b.ToString();
    }
    catch (Exception ex)
    {
        this.errorProvider1.SetError(this.textBox1, ex.Message);
    }
}
```

}

אם נכניס טקסט שתוכנו אינו מספר שלם ונלחץ על הכפתור, הטופס שיוצג יהיה:



### הסבר

הכנסנו את הטקסט "not an int" לתוך תיבת-הטקסט ולחצנו על הכפתור. הלחיצה גרמה לכך שניסינו לתרגם את ערך המחרוזת למספר שלם. התרגום, כמובן, כשל מפני שתוכן המחרוזת לא היה מספר שלם, מה שהוביל אותנו לבלוק ה-catch, שבו זימנו את ה- errorProvider לסמן את תיבת-הטקסט כמקום שבו יש שגיאה. המידע שסיפקנו על השגיאה הוא המידע מה- Exception שתפסנו בבלוק ה-catch (שקרה בבלוק ה-try). מידע זה מופיע כאשר עוברים עם העכבר על סימן השגיאה.

### כיצד מעלימים את סימן השגיאה?

לצורך ההסבר נשתמש, פעם נוספת, בדוגמה מהסעיף על "try ו-catch" שהרחבנו למעלה. נניח שהמשתמש באמת הכניס טקסט שאינו בר-תרגום ל-int ולחץ על הכפתור. הודעת השגיאה הופיעה, והמשתמש שינה את הטקסט שבתיבה לטקסט "48" שהוא אכן בר-תרגום. הכפתור נלחץ פעם נוספת. הפעם בלוק ה-try עובר ללא תקלות, והטקסט שבתווית מעודכן להיות "49". לכאורה נראה שהכל עבד היטב; האמנם? שאלה: מה קרה לסימן השגיאה שהופיע בהתחלה? תשובה: שום דבר. הוא נותר בדיוק כפי שהיה. אף על פי שהיינו רוצים שהוא ייעלם (המשתמש תיקן את הטעות!), הוא נשאר. עלינו למצוא דרך להעלים את סימן השגיאה, והדרך לעשות זאת היא כתיבת השורה (עבור עצם מסוג ErrorProvider ש-Name שלו הוא errorProvider1):

```
this.errorProvider1.SetError(this.שם_הרכיב_שבו_ארעה_השגיאה, "");
```

משמעות השורה - אם היה קיים סימן שגיאה ליד הרכיב המדובר, מחק אותו; אם לא, אל תעשה דבר. מה שאנו עושים כאן הוא בקשה להופעת סימן השגיאה ליד הקומפוננט המדובר, כשהמידע על השגיאה נתון על-ידי אוסף תווים ריק (מחרוזת ריקה). סימן השגיאה לא יופיע, ואף ייעלם אם היה קיים, מפני שהמידע על השגיאה הוא למעשה כלום.

נשים את השורה שמוחקת את סימן השגיאה בתוך בלוק ה-try. כך, אם לא הופיע עד עכשיו סימן שגיאה, שום דבר לא יקרה; אם היה קודם סימן שגיאה, הוא ייעלם; ואם תהיה שגיאה בבלוק ה-try (במקרה שלנו- כישלון בתרגום), אז נעבור לבלוק ה-catch שיפעיל מחדש את סימן השגיאה מה- ErrorProvider.

הקוד הסופי של פתרון הדוגמה עם שימוש ב-errorProvider הוא, אפוא :

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int a = int.Parse(this.textBox1.Text);
        int b = a + 1;
        this.label1.Text = b.ToString();
        this.errorProvider1.SetError(this.textBox1, "");
    }
    catch (Exception ex)
    {
        this.errorProvider1.SetError(this.textBox1, ex.Message);
    }
}
```

## MessageBox

למדנו עד כה שתי דרכים להודיע למשתמש שקרתה שגיאה בתוכנית :

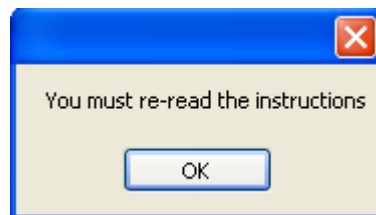
1. כתיבת הודעת שגיאה משל עצמנו בתוך תווית (או כל רכיב אחר של השפה שיכול להציג מידע).
2. שימוש ב- ErrorProvider על כל תכונותיו.

הסכמנו קודם לכן שהדרך הראשונה אינה בולטת מספיק, ושזו השנייה כן בולטת למדי. בסעיף זה נלמד עוד דרך אחת ודי.

ה- MessageBox (תיבת הודעה) היא מחלקה ב- Windows Forms שכל תכליתה היא להציג למשתמש הודעות באופן כזה שהוא לא יוכל בשום דרך להתעלם מהן. זוהי עליית הדרגה האחרונה שנבצע בסולם המובלטות של ההודעות שנציג למשתמש.

חייבים לשים לב שה- MessageBox אינה מרכיב נוסף בטפסים שלנו: לא נגרור שום דבר מה- Toolbox אל הטופס. השימוש ב- MessageBox יעשה מתוך הקוד באופן טהור ומוחלט.

נניח שנרצה להציג למשתמש את ההודעה: "You must re-read the instructions". הודעה העושה שימוש ב- MessageBox תיראה כך :



והיא תופיע במרכז המסך.

כאמור, לא ניתן להעלים עין מהודעה כזו. עד שנלחץ על כפתור ה-OK או על האיקס האדום שבכותרת החלון, לא ניתן יהיה ללחוץ על שום דבר בטופס שלנו או לעשות בו כל שימוש אחר.

### בהיבט התכנותי :

הצגת הודעה למשתמש תוך שימוש במחלקת MessageBox היא פשוטה למדי. כל שעלינו לעשות הוא להשתמש בפעולה Show של המחלקה, ולספק לה מחרוזת כזו או אחרת. תוכן המחרוזת הוא הטקסט שיוצג בהודעה. אם כן, מבנה השורה שתגרום להופעת הודעה כמו בתמונה לעיל הוא זה :

**MessageBox.Show(מחרוזת המתארת את ההודעה);**

לדוגמה, השורה שגרמה להופעת ההודעה שבתמונה היא :

```
MessageBox.Show("You must re-read the instructions");
```

ניתן דוגמה נוספת. נניח שאנחנו משתמשים במנגנון ה-try ו-catch. אנחנו כעת נמצאים בתוך בלוק ה-catch, והשגיאה "נתפסה" במשתנה ex. מה שנרצה לעשות בבלוק ה-catch הוא להציג את תוכן השגיאה שארעה, כפי שזו מופיעה במאפיין Message של ex (שטיפוסו הוא מחרוזת), בהודעה מסוג MessageBox. בלוק ה-catch ייראה, אם כך, כדלהלן:

```
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```



### הערה

ניתן לתת לפעולה Show פרמטרים שונים. למשל ניתן להוסיף עוד פרמטר מטיפוס מחרוזת שיהווה את הכותרת של ההודעה (שתופיע בקו הכחול בתמונה למעלה); או ניתן לשנות את טיב הכפתורים שיופיע בהודעה (למשל Yes ו- No במקום ה-OK). אף על פי זאת, אנחנו נשתמש בפעולה Show, על פי רוב, כפי שהשתמשנו בה בהסברים שבסעיף זה. נרחיב על כך בנספחי הספר.

### "סוף דבר" על שגיאות

נסקור כיצד הדיון בשגיאות התפתח: ראשית נגענו בקלט מהמשתמש ובהמרתו לטיפוס רצוי. שם גילינו שאנו צפויים להיתקל בשגיאות, וכך נולד הצורך לטפל בהן. צורך זה בא על סיפוקו בהכרת הכלי התכנותי שב-try ו-catch. בנוסף לכך, סיפקנו עוד שתי דרכים ויזואליות ובולטות להצגת הודעת שגיאה למשתמש: ה-ErrorProvider וה-MessageBox. כעת ברור כשמש שאנו חמושים בידע רב אודות שגיאות ואודות הדרכים לטיפול בהן.

### סיום ראשוני של משימת המחשבון

משלמדנו על תיבות-טקסט, על תוויות, על המאפיין ReadOnly, על קליטת קלט מהמשתמש והמרתו לטיפוס שונה ועל כל הקשור לשגיאות, אנו מצוידים בכל הכלים להתמודדות עם משימת המחשבון בהצלחה.



נסו כעת לבנות את המחשבון שלכם בעזרת כל שלמדתם! בהצלחה!



## ג. משימת מחשבון (2#) -

### כולל double ו-תנאי מקונן:

במחשבון שהזכרנו מקודם, אנו צריכים לעשות פעולת חילוק בין שני משתנים שהם מספרים שלמים מסוג int. המספרים נלקחים מתיבות-הטקסט על-ידי המרת המחרוזות (string) שבמאפייני ה-Text ל- int.

#### פעולת החילוק

אנו יודעים שערך הביטוי  $6/4$  הוא 1.5. למרות זאת, אם נכתוב ב- C# את ביטוי זה, נקבל את הערך 1. זהו חילוק מיוחד שנקרא "חילוק שלמים" או "חילוק int-ים". כדי לדעת ערכים של ביטויים מהסוג  $a/b$  (כאשר a ו-b הם משתנים מטיפוס int) נשאל את השאלה: כמה פעמים נכנס b בשלמותו ב-a? בדוגמה של  $6/4$  ברור ש-4 נכנס פעם אחת בלבד בשלמותו ב-6, וזו הסיבה שהתוצאה היא 1.

יש בידינו במשימת המחשבון שני מספרים שלמים שנרצה לחלק. חילוק שלמים לא מתאים לנו במקרה זה מפני שאנו רוצים לקבל את הדיוק המקסימאלי במחשבון (נרצה למשל כן להציג את התשובה 1.5 אם המשתמש יכניס את המספרים 6 ו-4 ויבחר בפעולת החילוק). נצטרך, איכשהו, לעקוף את פעולת חילוק ה- int-ים ולהשתמש ב"חילוק ממשיים" (כך נקראת פעולת החילוק המדויקת יותר).

כדי לבצע חילוק ממשיים נצטרך שאחד (או יותר) מהגורמים בביטוי יהיו מטיפוס double, שמייצג מספר ממשי. באופן טבעי, גם התוצאה תהיה מטיפוס זה. נזכר כי אפילו אם התוצאה היא מטיפוס שלם (למשל  $8/4=2$ ) הוא מספר שלם), הדבר לא ישנה כי הטיפוס double יכול גם להכיל ערכים שלמים ללא כל בעיות בתוכנה.

#### טיפוס המשתנים double



### double

double הוא טיפוס המאפשר למשתנה לאחסן ערך של מספר ממשי (כלומר מספר בעל ספרות לאחר הנקודה העשרונית) לדוגמה: 2.5, 7.0, 5.556. רמת הדיוק של הערכים שמשתנה מסוג double יכול להכיל היא עד 15-16 ספרות אחרי הנקודה העשרונית.

הגדרת משתנה double ב- C#: נגדיר משתנה באחת משתי הדרכים:

1. הגדרת המשתנה ללא אתחול. כך עדיין אין בו ערך ממש. אם נרצה שיהיה בו ערך, נצטרך להשים את ערך זה אל המשתנה. לדוגמה: `double b;`

2. הגדרת המשתנה ואתחולו. לדוגמה: `double b=54.7;`

השמת ערך חדש למשתנה שהוגדר ואתחל תפעל כאן כרגיל- הערך החדש יהיה ערך מפורש מטיפוס double, משתנה מטיפוס double, או ביטוי שטיפוסו הוא double.



### שימו לב

קבוצת המספרים השלמים היא חלקית לקבוצת המספרים הממשיים, שכן כל מספר שלם (לדוגמה, המספר 5) ניתן לכתוב כמספר ממשי (בדוגמתנו, המספר 5.0). משום כך ניתן להשים ערך שלם למשתנה ממשי - **כי לא נאבד מידע על המספר השלם**.

לעומת זאת, קבוצת המספרים הממשיים אינה חלקית לקבוצת המספרים השלמים, שכן לא ניתן לכתוב כל מספר ממשי כמספר שלם (למשל: המספר 5.5). אם היינו רוצים להכניס את המספר הממשי 5.5 למשתנה שלם, היינו צריכים לוותר על החלק שאחרי הנקודה העשרונית וכך לקבל את המספר השלם 5. שפת C# לא מאפשרת את איבוד המידע הזה. לכן, לא ניתן להשים ערך ממשי למשתנה שלם - **כי נאבד מידע על המספר הממשי**.

## פעולת החילוק, המשך

לשם קבלת תוצאת החלוקה, עומדות לרשותנו, שתי אפשרויות:

1. המרת המחרוזות שבתיבות-הטקסט שבהן מכניס המשתמש נתונים ל- double מראש (ולא ל-int).
2. המרה מ- int ל- double בזמן החישוב ואך ורק לצורך החישוב (כפי שנדגים למטה).

במקרה של המרת אחד המשתנים מ- int ל- double בזמן החישוב, הקוד ייראה:

```
double c = (double)a / b;
```

כאשר a ו- b הם המשתנים (מטיפוס int) שבהם שמורים המספרים שהכניס המשתמש, ו- c הוא המשתנה שבו נשמור את תוצאת החישוב.

כמו שאמרנו, מספיק להמיר רק את אחד המספרים ל- double (ההמרה היא רק לצורך החישוב), וכך עשינו.

במקרה שתרגמנו מראש את ערכי תיבות-הטקסט לערכים ממשיים (אפשרות 1), נשתמש כרגיל בסימן החילוק (/).  
(. התוצאה תהיה מדויקת משום שהגורמים הם מספרים ממשיים.

### שימו לב

**אין לחלק באפס!!! הדבר יגרום לבח"מ (ביטוי חסר משמעות) -> שגורר שגיאת ריצה ב-C#!**

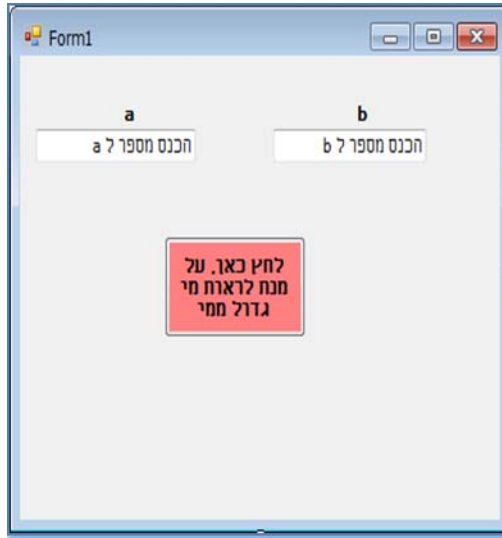
## תנאי מקונן

נזכור כי בכל בלוק נוכל להכניס אוסף פעולות כרצוננו. לכן, כאשר ישנה פקודת תנאי, ניתן בבלוק שלה להוסיף פקודת תנאי נוספת. הדבר נקרא בשם "תנאי מקונן".

### משימה לדוגמה עם תנאי מקונן

**המשימה:** עצב טופס שבו יש להכניס שני מספרים לשתי תיבות-טקסט שונות. כמו כן בטופס יהיה כפתור שלחיצה עליו תגרום להודעת MessageBox להופיע. בהודעה ייכתב שהמספר הראשון גדול מהשני, שהמספר הראשון קטן מהמספר השני או שהמספרים שווים זה לזה (בהתאם למקרה).

## טופס העבודה יראה כך :



### שלבים לביצוע המשימה :

1. נגרוור שתי תיבות-טקסט וכפתור לטופס, ונעצבם לפי העיצוב שלעיל (שנו את המאפיין Name של כל אחד מהעצמים – אנחנו לא נעשה זאת כאן).
  2. נגדיר שני משתנים a ו-b מסוג int במחלקה.
  3. נלחץ פעמיים על הכפתור, להגדרת הפעולה שהוראותיה יתבצעו במידה שהכפתור נלחץ. נכתוב את ההוראות הבאות:
- א. המרת הטקסטים של שתי תיבות-הטקסט לטיפוס int והשמת הערכים המומרים למשתנים a ו-b בהתאמה:

```
a = int.Parse(textBox1.Text);  
b = int.Parse(textBox2.Text);
```

- ב. נפעיל את התנאי המקונן באופן הבא :
- אם a גדול מ-b, כתוב בהודעה על כך. אחרת (תנאי נוסף): אם a שווה ל-b כתוב את הדבר; אחרת כתוב ש-a קטן מ-b.

```
if (a > b)  
{  
    MessageBox.Show("a גדול מ-b");  
}  
else  
{  
    if (a == b)  
    {  
        MessageBox.Show("המספרים שווים");  
    }  
    else  
    {  
        MessageBox.Show("a קטן מ-b");  
    }  
}
```



```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    int a;
    int b;
    private void button1_Click(object sender, EventArgs
    {
        a = int.Parse(textBox1.Text);
        b = int.Parse(textBox2.Text);
        if (a > b)
        {
            MessageBox.Show("a גדול מ-b");
        }
        else
        {
            if (a == b)
            {
                MessageBox.Show("שווים");
            }
            else
            {
                MessageBox.Show("a קטן מ-b");
            }
        }
    }
}
}
```

### מה מתבצע בתוכנית?

ננסה להבין את התוכנית. מה שלמעשה קורה הוא שאנו מקבלים שני מספרים מהמשתמש בתוכנית. לאחר מכן אנו נכנסים לתנאי הראשון ובו נבדק האם a גדול מהמשתנה b. אם הדבר מתקיים, נכנסים לבלוק התנאי ומבצעים אותו, ואז ממשיכים בתוכנית (ללא כניסה ל- else). אם התנאי הראשון אינו מתקיים נכנס אל האפשרות בלוק ה- else ומבצעים את ההוראות ששם. נראה כי גם בבלוק זה קיים תנאי (תנאי מקונן). התנאי בודק האם המשתנים שלנו שווים. אם הם אכן שווים, מבצעים את ההוראות שבבלוק ה- if ואז ממשיכים בתוכנית (ללא כניסה ל- else). אם התנאי אינו מתקיים, התוכנית עוברת אל בלוק ה- else השני. אם הגענו לבלוק זה, הרי ש- a קטן מ- b, ואכן מודפסת הודעה מתאימה. כך נבדקו כל האפשרויות השונות – באמצעות תנאי מקונן.

### שימוש נוסף וחשוב במחשבון שלנו:

במחשבון אנו נדרשים לתנאי מקונן גם במקרה של חילוק. בחלק החילוק במחשבון ציינו כי אין לחלק באפס מפני שהדבר אינו אפשרי וגורם לשגיאה ריצה. שאלה: כיצד התנאי המקונן בא לידי ביטוי כאן? תשובה: בשני התנאים הבאים-

1. האם נבחרה פעולת החילוק?
2. האם המחלק שווה לאפס (אם כן, מוצגת הודעה שגיאה; אם לא, יבוצע החילוק).

עתה נסו להוסיף כפתור נוסף למחשבון ולבצע את פעולת החילוק, תוך בדיקה שלא חילקנו בטעות באפס!





## ד. משימת מחשבון (#2), כולל פעולות:

כפי שוודאי שמתם לב, קיימים חלקים המופיעים גם בפעולת הכפתור של החיבור, גם בפעולת הכפתור של החיסור וגם בפעולת הכפתור של הכפל. זוהי **כפילות בקוד** והיא מעידה על תכנות לא נכון. ננסה להסביר מדוע באמצעות הדוגמה הבאה:

דמיינו שאתם עובדים בחברת היי-טק ושאתם כותבים אלפי שורות קוד. נניח שקיים חלק בקוד שחוזר על עצמו פעמים רבות ושמטרתו היא לחשב את הממוצע של שני משתנים. כעת, תארו לעצמכם שהבוס שלכם ניגש אליכם ומבקש שיבוצע חישוב הממוצע של שלושה משתנים במקום שניים. מה שתצטרכו לעשות הוא לשנות את הקוד בכל מקום שבו חישוב הממוצע מופיע. מיותר להסביר שעבודה זו תהיה ארוכה ומסורבלת אם לא בלתי-נסבלת.

מסיבה זו נרצה להימנע מכפילות בקוד בכל מקום שבו הדבר יתאפשר. נזדקק לכלי שיאפשר:

1. לכתוב את הקוד פעם אחת ובמקום אחד.
  2. לזמן את הפעולה מתי שנרצה, כלומר להשתמש בקוד שכבר כתבנו בעת הצורך.
- פעולות הן בדיוק הכלי שמאפשר לעשות את זה.

שימו לב שזו לא הפעם הראשונה שאנחנו נתקלים במושג "פעולות". דיברנו כבר על פעולות של כפתורים, תוויות וכו'. לא היה כתוב קוד שונה עבור כל לחיצה על הכפתור (למשל), אלא בכל לחיצה בוצע אותו קוד קטע, כלומר משתמשים כאן בפעולה.



**פעולה** - בלוק של קוד המכיל סדרת הוראות. מיד לפני הבלוק צריכה להופיע כותרת/חתימה לפעולה.

בעת כתיבת הקוד מצהירים על פעולה באמצעות כתיבת כותרת/חתימת הפעולה. הכותרת כוללת:

1. הרשאת גישה
2. טיפוס החזרה
3. שם הפעולה
4. פרמטרים

### הרשאת גישה

- לא ניכנס לכל "תורת" הרשאות-הגישה, משום שהדבר חורג מחומר הלימוד שבספר זה. עם זאת, ניתן הסבר קצר. ב-C# קיימות מספר הרשאות גישה; הפופולאריות ביותר הן private ו-public (ואלו הן מילים שמורות):
- public (פומבי, ציבורי) - משמעות מתן הרשאת הגישה public לפעולה היא שהגישה לפעולה אינה מוגבלת.
  - private (פרטי) - משמעות מתן הרשאת הגישה private לפעולה היא שהגישה לפעולה מוגבלת רק למחלקה שבה נכתבה הפעולה.

אנחנו נשתמש, במסגרת הלימוד שלנו, בהרשאת הגישה private.

### טיפוס החזרה

יש שני סוגים של פעולות:

#### 1. פעולות המחזירות ערך

אחת מהתכונות של הפעולות הוא שבכוון להחזיר ערך מטיפוס מסוים לבחירת המתכנת. הערך מוחזר למקום שממנו זומנה הפעולה. טיפוס החזרה יכול להיות כל אחד מהטיפוסים שאנו מכירים: int, bool, double, char, string וכדומה. הטיפוס הנבחר הוא שצריך להופיע במקום המדובר בחתימת הפעולה.

#### 2. פעולות שאינן מחזירות ערך

אם המתכנת בוחר לכתוב פעולה שאינה מחזירה ערך למקום ממנו זומנה, עליו לכתוב במקום שנועד ל"טיפוס ההחזרה" בחתימת הפעולה את המילה השמורה **void**.

### שם הפעולה

על המתכנת לבחור שם הגיוני ומשמעותי לפעולות שהוא כותב. נהוג לכנות את הפעולות בשמות שמתארים בקצרה את מטרותיהן. כך התוכניות נעשות ברורות, קריאות ומובנות. שימוש בשמות כמו Method1 או X וכיו"ב הוא אמנם תקין מבחינת השפה, אך אינו מקובל.

מוסכם, ב-C#, להתחיל שם פעולה באות גדולה. המשך המילה יהיה באותיות קטנות. אם תתחיל מילה נוספת, גם היא תפתח באות גדולה ותמשיך באותיות קטנות, וכך הלאה. מוסכמה נוספת היא שמספרים יכולים להופיע בשם הפעולה, אם כי לא בתחילתו. מקובל אף לקצר מילים, אולם עד גבול הטעם הטוב (לדוגמה, ניתן לקצר את המילה Average ל-Avg ואת המילה Number ל-num וכיו').

דוגמאות: השמות Avg, SumNums, Calculate3Digits הם חוקיים, ואילו השמות 3NumsAvg, intInput, writenumbres, oUtPut אינם חוקיים.

### פרמטרים

אם דואגים לכך בכותרת הפעולה, ניתן לדרוש מהמשתמש לשלוח פרמטרים בעת זימון הפעולה. הפעולה משתמשת בערכי הפרמטרים האלה, ובאופן זה הפעולה מבצעת דברים שונים בהתאם לערכי הפרמטרים המתקבלים. כדי לעשות זאת יש לציין בכותרת הפעולה, מיד לאחר שמה ובתוך סוגריים, את טיפוס הפרמטרים ואת שמותיהם. שמות הפרמטרים יכולים להיות ככל העולה על רוחנו, ללא קשר בכלל למקום שממנו זומנה הפעולה. אם יש למעלה מפרמטר אחד, יש להפריד אותם בפסיקים. ניתן גם לכתוב פעולה שלא מקבלת פרמטרים בכלל, ואז הסוגריים של הפרמטרים יהיו ריקים.

נראה כמה דוגמאות לכותרות של פעולות:

<u>כותרת הפעולה</u>	<u>פרמטרים</u>		<u>טיפוס החזרה</u>
	<u>שם הפרמטר</u>	<u>טיפוס הפרמטר</u>	
<code>private void FirstMethod ()</code>	-	-	-
<code>private void SecondMethod(int firstParam)</code>	firstParam	int	-
<code>private void ThirdMethod (bool flag, int num)</code>	flag num	bool int	-
<code>private int FourthMethod ()</code>	-	-	int
<code>private int FifthMethod(int max)</code>	max	int	int
<code>private bool SixthMethod(int a, bool b)</code>	a b	int bool	bool

### הערה

בטבלה זו לא הקפדנו לתת שמות משמעותיים לפעולות ולמשתנים מפני שהם תלושים מהקשר ומשום שהפעולות חסרות תוכן.

אנא ודאו שכותרות הפעולות שבטבלה לעיל מובנות לכן בטרם תמשיכו לקרוא.

לאחר כותרת של פעולה יבוא בלוק שבו נכתוב הוראות כרצוננו. הוראות אלה הן שתבוצענה אם הפעולה תזומן ממקום כלשהו בקוד. אין שום הכרח לגבי תוכן הבלוק להוציא את המקרה שבו הפעולה מחזירה ערך.

### החזרת ערך

בפעולות המחזירות ערך נצטרך לכתוב בשלב מסוים בבלוק הפעולה הוראת החזרה. מבנה הוראת החזרה הוא:

```
return קבוע/משתנה/ביטוי_מהטיפוס_המוחזר;
```

לדוגמה: אם ערך החזרה של פעולה הוא bool, ויש בידנו משתנה מטיפוס bool ששמו הוא b, נוכל לרשום:

```
return b;
```



### שימו לב

בדרך כלל נמקם את הוראת החזרה בסיום הפעולה. הסיבה לכך היא שהחזרה עוצרת את ביצוע הפעולה. הערך שאנו מחזירים נשלח למקום שממנו זומנה הפעולה, וקטע הקוד שבו נמצא הזימון ממשיך להתבצע. אפילו אם יש קוד נוסף בפעולה לאחר הוראת החזרה, המחשב לא יחזור לבצע את ההמשך.

## דוגמאות לפעולות מלאות

1. פעולה שלא מקבלת דבר ומחליפה את הטקסטים של שתי תיבות טקסט שבטופס (הפעולה לא מחזירה דבר):

```
private void ChangeTexts()
{
    string firstText = this.textBox1.Text;
    this.textBox1.Text = this.textBox2.Text;
    this.textBox2.Text = firstText;
}
```

2. פעולה המקבלת מספר ושמה את ריבועו בתוך המאפיין Text של ה- TextBox שבטופס (הפעולה לא מחזירה דבר):

```
private void SquareInTextBox(int number)
{
    int square = number * number;
    this.textBox1.Text = square.ToString();
}
```

או פשוט:

```
private void SquareInTextBox(int number)
{
    this.textBox1.Text = (number * number).ToString();
}
```

הערה: ההמרה ל-string נחוצה מכיון שהטיפוס של המאפיין Text הוא String (מחרוזת).

3. פעולה המקבלת פרמטר בוליאני ומשתמשת ב- MessageBox כדי להדפיס את המילה "Success" אם ערך הפרמטר הוא true (הפעולה לא מחזירה דבר):

```
private void WriteSuccessIfTrue(bool isTrue)
{
    if (isTrue)
        MessageBox.Show("Success");
}
```

4. פעולה המקבלת 2 מספרים שלמים ומחזירה את מכפלתם:

```
private int Multiply(int num1, int num2)
{
    int result = num1 * num2;
    return result;
}
```

או פשוט:

```
private int Multiply(int num1, int num2)
{
    return (num1*num2);
}
```

5. פעולה המקבלת מספר ומחזירה true אם הוא קטן מ-10 או false בכל מקרה אחר :

```
private bool IsSmallerThan10(int number)
{
    bool result;
    if (number < 10)
        result = true;
    else
        result = false;
    return result;
}
```

או פשוט :

```
private bool IsSmallerThan10(int number)
{
    return (number < 10);
}
```

### זימון פעולות

כדי לזמן פעולה בשלב מסוים בקוד כל שנצטרך לעשות הוא לכתוב את שם הפעולה ולאחריה סוגריים שבהם יופיעו הערכים שנעביר לפעולה. כמות הערכים, הסדר שלהם והטיפוסים שלהם חייבים להתאים באופן מושלם להגדרה שבחתימת הפעולה. אם הפעולה אינה מקבלת פרמטרים כלל, הסוגריים יותרו ריקים. נגדים את הזימון של את כל 5 הפעולות שכתבנו בדוגמה לעיל. במקרה שהפעולה מחזירה ערך, נשמור אותו במשתנה מטיפוס מתאים. לצורך דוגמאות הזימון, הגדרנו בשתי השורות הראשונות כמה משתנים. להלן הדוגמאות :

```
int num = 2, result4;
bool b = false, result5;
ChangeTexts(); // first method
SquareInTextBox(2); // second method
WriteSuccessIfTrue(b); // third method
result4 = Multiply(num, 5); // fourth method
result5 = IsSmallerThan10(num); // fifth method
```

### הערה על פעולות



ניתן לכתוב שתי פעולות שונות בעלות אותו שם בדיוק בתנאי שהפרמטרים של הפעולות שונים זה מזה בדרך כלשהי: במספרם, בסדרם או בטיפוסיהם. מנגנון שבו מעמיסים מספר פעולות על שם אחד נקרא, באופן לא מפתיע, העמסה (באנגלית: overloading).

### במשימת המחשבוני

קרוב לוודאי שקטע הקוד שחוזר על עצמו אצלכם הוא זה שבו ממירים את המחרוזות שבתוכנית-הטקסט ל-int (חלק מבלוק ה-try), כמו גם בלוק ה-catch במלואו (הטיפול בשגיאת ההמרה). שני אלה הם דברים הנעשים באופן זהה עבור כל פעולה חשבונית במחשבון. לכן הגיוני שנכתוב אותם בפעולה אחת. בפעולה זו הערכים שבתוכנית-הטקסט יומרו, אם ניתן, ל-int, ואם לא – תוצג הודעת שגיאה מתאימה. בפעולות החיבור, החיסור והכפל יעשה החישוב המתמטי המתאים עם הערכים שהומרו קודם לכן בפעולה.

### סיים משימת המחשבון



קעת אתם יכולים לשפר את הקוד שמאחורי המחשבון שלכם. אל תמהרו לעשות זאת ואל תקלו ראש - המשימה לא פשוטה מאד. שימו לב לנקודות הבאות :

- בתוך הפעולה שנכתוב נצטרך לשמור את הערכים המומרים ל-int במשתנים מסוימים. משתנה שיוגדר בבלוק הפעולה לא יתאים משום שתחום "חייו" נגמר בסוף הפעולה, ואז לא נוכל לבצע את הפעולות המתמטיות המתאימות עבור כפתורי החיבור, החיסור והכפל. המידע שהשגנו יאבד. נפתור את הבעיה באמצעות משתני מחלקה שעליהם למדנו במשימת הרמזור.

- השימוש ב-ErrorProvider מצריך, כמו שאנו יודעים, ציון של הרכיב אליו נרצה שיוצמד סימן השגיאה. אם נשים בבלוק try אחד את שתי פקודות התרגום ל-int ותתרחש שגיאה באחת ההמרות, לא נוכל לדעת בבלוק ה-catch באיזו תיבת טקסט הטקסט לא היה בר-תרגום. נציע כאן פתרון אחד לבעיה- שימוש במנגנון ה-try ו-catch **פעמיים**: פעם אחת עבור כל תיבת טקסט.

- נניח שבפעם הראשונה המשתמש הכניס את הטקסטים "2" ו-"4" לתיבות הטקסט ולחץ על הכפתור של פעולת החיבור. הכול עבד היטב ובתיבת התוצאה הופיעה המחרוזת "6". לאחר מכן המשתנה שינה הטקסט "2" לטקסט "not at int", שהוא כמובן אינו בר תרגום. הוא לחץ על הכפתור "כפל". בבלוק ה-try התגלתה שגיאת ההמרה, שנתפסה בבלוק ה-catch. בבלוק ה-catch הצגנו הודעת שגיאה, וכך נגמרה הפעולה שכתבנו לטיפול בקלט מהמשתמש. המחשב חזר לפעולת הכפל (שבו זומנה פעולת הקלט). כעת ההוראות הן לכפול את שני משתני המחלקה (שבהם אמורים להישמר הערכים המומרים מתיבות הטקסט של המספרים) ולשים את התוצאה בתיבת התוצאה. הדבר הבא שיקרה הוא שבתיבת התוצאה נראה את הטקסט "8". מדוע? מפני שערכי משתני המחלקה לא השתנו ונותרו 2 ו-4 עוד מהפעלת כפתור החיבור.

נשאלת השאלה - כיצד נפתור את הבעיה הזו? התשובה היא שנאלץ להשתמש במשתנה מחלקה נוסף שיגיד לנו אם הגענו לאחד מבלוקי ה-catch (כלומר ארעה שגיאה). אם לא הגענו לשם, נבצע הכל כרגיל. אם כן, נצטרך לשנות את הטקסט שבתיבת התוצאה לטקסט "", כלומר מחרוזת ריקה. במילים אחרות, אנו מתנים את ביצוע הפעולה החשבונית בשאלה האם הגענו בבדיקה האחרונה לאחד מבלוקי ה-catch.



**נסו כעת לבנות את המחשבון שלכם בעזרת כל שלמדתם, כולל פעולות! בהצלחה!**