מדריך מקוצר לתכנות מיקרו-בקרים בשפת

מאת

חברת BRK

חינוך טכנולוגי מתקדם

www.brk.co.il

© BRK.

תוכן עניינים

מבוא 3
קצת רקע על שפת C# 3
צעדים ראשונים 3
טיפוסי משתנים 4
שלמים: 4
עם נקודה עשרונית 4
בוליאני 5
מחרוזת 5
איך נדע באיזה טיפוס להשתמש? 5
פקודת if 8
לולאות 12
for 12
while 15
do while 17
continue 17
break 18
goto 18
switch 19
תכנות מתקדם 21
פונקציות / מטודות 21
מחלקות 24
מילון מושגים 24
מפת הדקים של הכרטיס 24
אזהרות 25

© BRK.

מבוא

אנו, בחברת BRK רואים את המיקרו-בקרים המתוכנתים בשפת C# כשלב הבא אליו יתקדם החינוך הטכנולוגי בארץ. הטכנולוגיות המיושנות שעדיין מלומדות בשלב זה במרבית בתי הספר ומכללות להנדסאים לא פעם גרמו מבוכה בהשוואה לטכנולוגיות המיושמות בתעשיית ההיי-טק העולמית והישראלית. מצד שני, המוצרים המקביליים מאפשרים תכנות בקרים בשפת "צעצועי-לגו" שלא מקנה ללומדיה כלים ממשיים אותם יוכל ליישם בתעשייה, אלא הפעלה מהירה של רכיבי חומרה. תכנות הבקרים בשפת C#, לעומת זאת, מקנה את המקצועיות בתכנות מונחה עצמים (Object Oriented) והשקעת זמן הפיתוח בבניית פרויקטים משמעותיים ושימוש בטכנולוגיות עכשוויות, במקום להשקיע אותו בעבודה ברמת הסיביות מול חיישנים, אוגרים פנימיים וכד'. רכישת היכולת לתכנת בשפת C# מקנה את האפשרות לתכנת לא רק מיקרו-בקרים, אלא ליצור פרויקטים המשלבים ישומי מחשב תחת windows (WinForm), אתרי אינטרנט, אפליקציות בטלפונים סלולאריים ועוד.

קצת רקע על שפת C#

C# היא שפת תכנות פשוטה וקלה ללימוד, מודרנית, מונחת עצמים ובעלת ישומים בתחומים רבים. שפה זאת פותחה ע"י Microsoft בשנת 2000 כחלק הפרוייקט ה .net ושודרגה מספר פעמים בשנות האלפיים. השפה נחשבת היום לאת השפות המובילות התחום הנדסת תוכנה ודומה מאוד לשפת C הישנה ולשפת Java.

צעדים ראשונים

בפרק זה אנו נדבר על הפקודות הבסיסיות בשפת C#. כפי שניתן יהיה לירות, מרבית הפקודות דומות לשפת C הרגילה והמוכרת. ראשית, יש לפתוח פרויקט לעבודה עם הבקר שלנו. ניתן לראות את ההסבר המפורט על כך בנספח "פתיחת פרויקט חדש" המצורף בסוף דפי הסבר אלה.

© BRK.

טיפוסי משתנים

בשפת #C (בסביבת פיתוח Visual studio 2012) קיימים טיפוסים (סוגים) שונים מהם ניתן ליצור משתנים. כל סוג או הטיפוס של כל משתנה צריך להתאים למידע אותו נרצה לאכסן בו. כפי שישנן משאיות המיועדות לאכסון והובלה של מטענים שונים: נוזל דליק, מים, מזון בקירור, מזון בהקפאה, שתיה קלה בבקבוקים ופחיות, אבנים במחצבה וכו', כך גם קיימים טיפוסים שונים עבור המשתנים. את תמציתם ניתן לראות בטבלאות הבאות:

שלמים:

טיפוס	טווח ערכים	גודל
sbyte	-128 – 127	שלם, 8 סיביות עם סימן
byte	0 – 255	שלם, 8 סיביות ללא סימן
char	U+0000 to U+ffff	תו Unicode של 16 סיביות
short	-32,768 – 32,767	שלם, 16 סיביות עם סימן
ushort	0 – 65,535	שלם, 8 סיביות ללא סימן
int	-2,147,483,648 – 2,147,483,647	שלם, 32 סיביות עם סימן
uint	0 – 4,294,967,295	שלם, 32 סיביות ללא סימן
long	-9,223,372,036,854,775,808 – 9,223,372,036,854,775,807	שלם, 64 סיביות עם סימן
ulong	0 – 18,446,744,073,709,551,615	שלם, 64 סיביות ללא סימן

עם נקודה עשרונית

טיפוס	טווח ערכים	גודל	דיוק
float	±1.5e−45 – ±3.4e38	32 סיביות	7 סיביות
double	±5.0e−324 – ±1.7e308	64 סיביות	15-16 סיביות
decimal	$(-7.9 \times 10^{28} - 7.9 \times 10^{28}) / (10^{0-28})$	128 סיביות	28-29 סיביות

© BRK.

בוליאני

טיפוס	טווח ערכים	גודל
bool	false – true	8 סיביות

מחרוזת

טיפוס	טווח ערכים	גודל
string	מחרוזת של תווים	16 סיביות * מס' תווים במחרוזת + 20 * 8 סיביות

איך נדע באיזה טיפוס להשתמש?

בכדי לקבוע איזה טיפוס מתאים למשתנה, עלינו לדעת מה נרצה לאחסן בו. במידה ונרצה לשמור את שם המשתמש, נצטרך משתנה שיכול לשמור בתוכו טקסט. עבורמספר תעודת הזהות או מספר חשבון בנק נצטרך משתמה שיכול לאגור בתוכו מספרים שלמים וכן המסכורת החודשית של איש צוות תכנס למשתנה שיכול לאגור בתוכו מספרים לא שלמים, בעלי נקודה עשרונית. לפעמים, נפעיל פונקציות (על כך נרחיב בהמשך) המחזירות לנו את התוצאה בתוך משתנה מטיפוס מסוים ולכן עלינו לשמור אותו בסוג זה של טיפוס דווקא. כך למשל אם נרצה לקרוא את מצב הלחצן הממוקם ע"ג ערכת הפיתוח של הבקר, הפונקציה שאותה נפעיל בשביל זה תחזיר לנו משתנה מטיפוס `bool` ובתוכו: `false` – במידה והלחצן לא לחוץ ו `true` המידה והלחצן לחוץ. ניתן לסכם את האמור בטבלה הבאה:

טיפוס המשתנה	סוג המידע הנשמר	דוגמאות
`String`	טקסט	שם פרטי ושם משפחה, הודעה אותה נרצה להציג למשתמש על מסך המחשב או תצוגת גביש נוזלי
`int`	מספרים שלמים	מספר תעודת זהות, מס' שניות שיש להמתין עד לביצוע פעולה מסויימת, מס' תלמידים בבית ספר
`double`	מספרים עם נקודה עשרונית	המתח האנלוגי המתקבל במבוא של הבקר, תוצאות חישובים כדוגמת חילוק.
`bool`	כן / לא	מצב של מבוא ספרתי של הבקר או מוצא ספרתי שלו, תוצאה של תנאי לוגי,

© BRK.

דוגמא 1

נגדיר משתנה מטיפוס bool (קיצור של Boolean) שיכול להכיל רק ערכים לוגיים: "1" לוגי = true או "0" לוגי = false.

לשם כך בין הסוגריים המסולסלים { } של הפונקציה הראשית ()Main המתבצעת עם הרצת התוכנית, נכתוב את השורה הבאה:

```
bool condition = false;
```

שימו לב שסביבת הפיתוח בתכנות בשפת #C עוזרת למתכנת לבחור את הפקודה הרצויה תוך שמירה על כתיבתה הנכון. עם התכלת ההקלדה של המילה bool תופיע הרשימה של הפקודות האפשרויות עם המתחילות באותיות שהוקלדו.

```
public static void Main()
{
    b
    await
    BaseEvent
    Battery
    BaudRates
    bool            struct System.Boolean
    Boolean
    break
    Button
    byte
}
```

ניתן לבחור את הפקודה הרצויה מתוך הרשימה ע"י מקשי החצים של המקלדת, או להקליק עליה עם העכבר. במידה והקלדתם מספיק אותיות והפקודה הרצויה כבר נבחרה עם השורה בצבע כחול, ניתן להקיש על המקש Tab או Enter שבמקלדת וסביבת הפיתוח תשלים עבורכם את הקלדת האותיות החסרות.

© BRK.

ניתן להגדיר את המשתנה ללא ערך התחלתי ולבצעה השמה מאוחר יותר. כך למשל:

```
bool condition;
…
condition = false;
…
```

השמת מידע לא תואם את סוג המשתנה תגרום ברוב המקרים לשגיאת קומפילציה. כך, למשל, לא ניתן להכניס מספר 2.73 לתוך משתנה וטיפוס int או מספר 512 אל תוך משתנה byte.

דוגמא 2

נראה דוגמא נוספת, בה לצד השימוש במשתמים, נבצע עלהם פעולות אריטמטיות פשוטות:

```
int a = 2;
int b = 3;
int c = a+b;
a=b+c;
```

בשורה הראשונה אנו מגדירים את המשתנה מסוג int (שיכול לקבל מספרים שלמים בלבד) בשם a וישר מכניסים לתוכו ערך 2.
בשורה שנייה מוגדר משתנה נוסף מאותו הטיפוס int בשם b ומכניסים בתוכו ערך של 3.
בשורה שלישית מוגדר משתנה מסוג int בשם c אליו מכניסים את הסכום של תוכן המשתנה a ותוכן המשתנה b.
בשורה רביעית ואחרונה מכניסים לתוך משתנה a שהוגדר כבר, את הסכום של תוכן המשתנים b ו c. לאחר ביצוע שורה זו תוכן המשתנה a יהיה 7.

© BRK.

פקודת if

לפעמים עולה צורך לבצע קטע קוד מסוים רק אם מתקיים תנאי מסוים. לשם כתיבת תוכנית מסוג זה נשתמש בפקודת if:

מילה שמורה

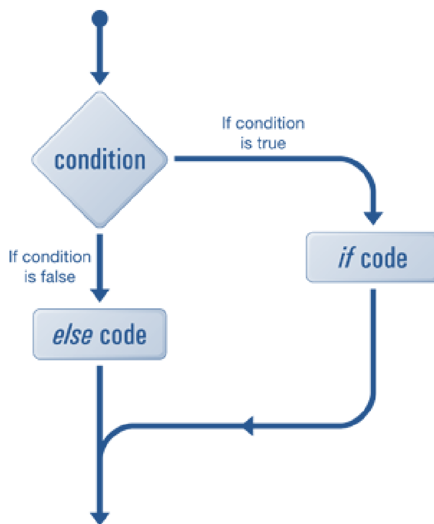
תנאי לוגי

בין הסוגריים במסולסלות רשום הקוד שיתבצע כאשר התנאי הלוגי (בדוגמא זו: a>b) מתקיים

```
if (a>b)
{
    Debug.Print("The condition is true.");
}
else
{
    Debug.Print("The condition is false.");
}
```

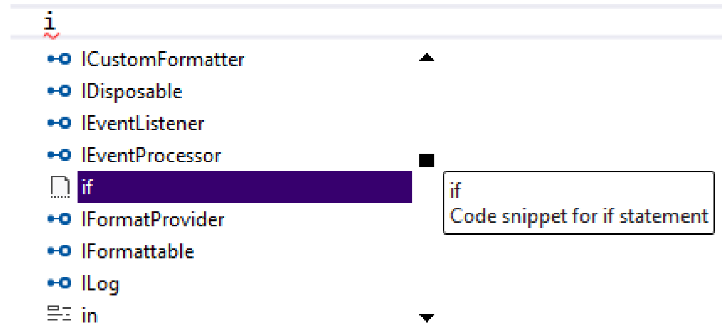
בין הסוגריים במסולסלות שלאחר המילה השמורה else, רשום הקוד שיתבצע כאשר התנאי הלוגי (בדוגמא זו: a>b) **לא** מתקיים

להבנת אופן זרימת ביצוע התוכנית, ניתן להיעזר בתרשים הבא:



© BRK.

בשפת #C בשונה משפת C קיימים הרבה כלים המקלים על המתכנת את העבודה. כך גם כאן, מספיק להתחיל להקליד את הפקודה הרצויה וסביבת הפיתוח תציע רשימת פקודות התואמות התחלת ההקלדה של המשתמש. עם הקלדת האות i בלבד, תפתח רשימה של כל הפקודות המתחילות האות זו:



ניתן לבחור מתוך הרשימה את פקודת ה if ע"י חצי המקלדת או עם הקליק של העכבר. לאחר שהפקודה הרצויה סומנה, ניתן ללחוץ על הלחצן Tab בכדי שהיא תופיע על המסך. לחיצה מידית נוספת על ה Tab תביא לפתיחת כל פקודת ה if והדגשת התנאי הלוגי אותו יש למלא ונמצא כ true בברירת מחדל:

```
if (true)
{

}
```

נחליף מילת ה true הגורמת לתנאי להתקיים תמיד, בתנאי לוגי כרצוננו.
התנאי הלוגי המופיע בתוך הסוגריים של if חייב להיות מורכב ממשתנים שהוגדרו, אופרטורים (>, <, = וכו') ומספרים. בכדי שהתנאי a>b יהיה חוקי, עלינו להגדיר לפניכן את המשתנים האלו. נגדיר ונאתחל אותם עם ערכים שונים. ניתן לעשות זאת ע"י הפקודות:

```
int a = 8;
int b = 5;
```

המשתנים a ו b לא חייבים להיות דווקא int, אלא טיפוסים נוספים כדוגמת char, long וכו'.
נציין בנפרד את הטיפוס string שלא קיים בשפת C. הטיפוס הוא למעשה כמין מחרוזת של תווים כפי שזה מוכר לנו בשפת C. ניתן להגדיר ולעבוד בקלות עם משתנים מסוג זה כאשר ישנו צורך לעבד טקסט.

© BRK.

האופרטורים החוקיים בהם ניתן להשתמש הם:

- < קטן מ...
- > גדול מ...
- <= קטן או שווה ל...
- >= גדול או שווה ל...
- == שווה ל...
- != לא שווה ל...
- || או לוגי
- && וגם לוגי
- () סדר פעולות

ישנם עוד מס' אופרטורים חוקיים כגון `is`, אך הדיון עליהם יוכל להתקיים בפרקים מתקדמים יותר.

להלן מספר דוגמאות לשימוש בתנאים לוגיים:

```
if (a/2 >= c+10)
if (a*2 > b)
if ((a > b) && (a > 20))
if (++a != b--)
```

חשוב להקפיד שבתוך הסוגריים תמיד יהיה תנאי לוגי ולא פעולה אריתמטית, למשל.

כמוכן, ניתן להגדיר משתנה מטיפוס בוליאני (`bool`) ולהכניס בו ערך `false` או `true` לפני ביצוע התנאי ולהשתמש במשתנה זה לבדו או בצירופו עם משתנים אחרים. האפשרות הזאת יכולה להיות שימושית בעבודה עם תנאים מורכבים וכן העברת נתונים בין הפונקציות השונות בתוכנית.

© BRK.

דוגמא 1

```
int a = 4, b = 5;

if (a>b)
{
    Debug.Print("a > b");
}
else
{
    Debug.Print("b >= a");
}
```

הגדרת שני משתנים a ו b והכנסת ערך התחלתי אלהם

האם תוכן המשתנה a גדול יותר מתוכן המשתנה b

אם התנאי מתקיים - הצגת הודעה על מסך המחשב "a > b"

אם התנאי לא מתקיים - הצגת הודעה על מסך המחשב "b >= a"

בנוסף לאפשרות לכתוב תנאי בודד, ניתן גם לשלב את התנאי בתוך תנאי אחר.

דוגמא 2

```
bool condition = true;
if (condition)
            {
                Debug.Print("The condition is true.");
                if (a<b)
                {
                    Debug.Print("a<b");
                }
                else
                {
                    Debug.Print("a>=b");
                }

            }
            else
            {
                Debug.Print("The condition is false.");
            }
```

ההודעה שתוצג על מסך המחשב רק אם condition = true וגם התוכן של המשתנה a קטן מתוכן של המשתנה b

ההודעה שתוצג על מסך המחשב רק אם condition = true וגם התוכן של המשתנה a גדול או שווה לתוכן של המשתנה b

© BRK.

לולאות

הגדרה: לולאה היא קטע קוד שמתבצע מספר פעמים.

בשפת #C ישנן מספר פקודה בעזרתן ניתן ליצור לולאות מסוגים שונים.

for

זהי פקודה בעזרתה ניתן ליצור לולאה. נשתמש בה ע"פ רוב כאשר מספר החזרות על קטע קוד רצוי ידוע מראש. למשל: בכדי להדפיס את שמכם על המסך 10 פעמים.

תחביר הפקודה הוא:

```
for (int i = 0; i < 10; i++)
        {
          Debug.Print("Hello");
        }
```

שימו לב כי בתוך הסוגריים העגולים ישנם 3 שדות המופרדים בניהם עם ; (ולא עם ,):

תנאיי התחלה: בשדה זה של הפקודה נרשום את כל התנאים (במידה וקיימים) שמהם אנו רוצים להתחיל את הלולאה – תנאיי התחלה. הקוד הרשום בשדה זה יתבצע רק פעם אחת בכניסה אל הלולאה. בדוגמא שלנו הגדרנו משתנה בשם i מסוג int והכנסנו בו ערך 0.

תנאיי סיום: בשדה זה של הפקודה עלינו להגדיר את התנאי או התנאים בהם תסתיים ביצוע הלולאה והתוכנית תמשיך לרוץ ולבצעה את הפקודות הבאות. בדוגמא שלנו, כל עוד i<10 (תוכנו של המשתנה בשם i קטן מ 10), הלולאה תחזור על עצמה שוב, אך כאשר תנאי זה יופר (למשל, בתוך המשתנה i יהיה מספר 11) הלולאה תסתיים והתוכנית תבצעה את הפקודות הבאות. המחשב עובר לבדיקת תנאי היציאה מהלולאה מיד אחרי ביצוע הקוד בשדה של תנאיי ההתחלה וכן בסיום כל לולאה.

קידום: בשדה זה של הפקודה, נרשום את הקוד שיתבצע בסיום כל לולאה. בדוגמא שלנו, נוסיף 1 לתוכן של המשתנה בשם i (++i). לאחר ביצוע קוד זה, המחשב יבדוק האם התנאי בו יש לבצע את הלולאה שוב (תנאי סיום), עדיין מתקיים.

© BRK.

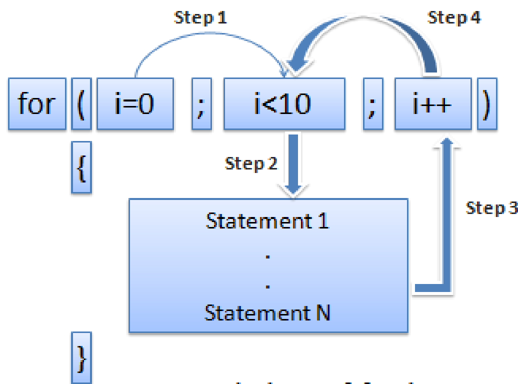
במידת הצורך, ניתן להשאיר אחד או יותר משלושת השדות הנ"ל של הלולאה for ריקים.

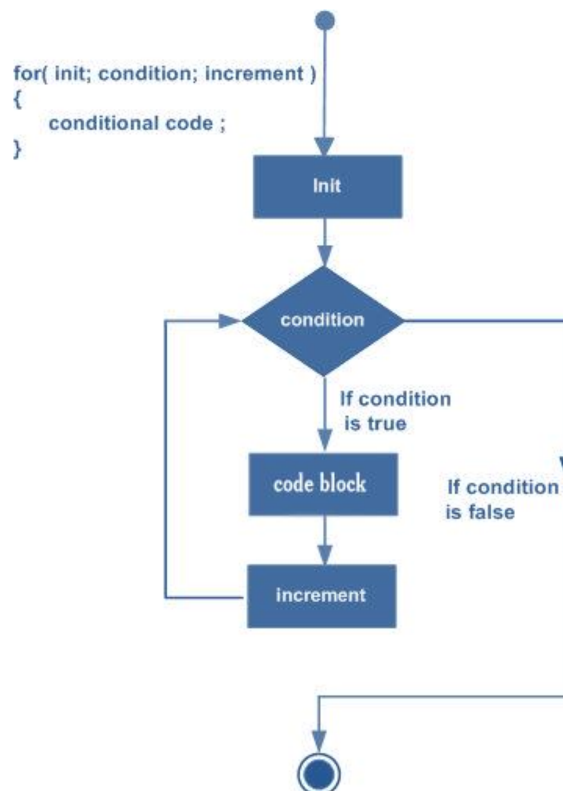
שימו לב: בסוף השורה של for אין ; !

כמו בפקודה if, גם כאן, הקטע קוד שיתבצע בלולאה הוא המוקף בסוגריים מסולסלים { }.

גם כאן, במידה ומדובר בפקודה בודדת, לא חייבים לשים סורגים אלה, אך כמתכון למניעת שגיאות מומלץ לשים סוגריים אלה גם במצב זה.

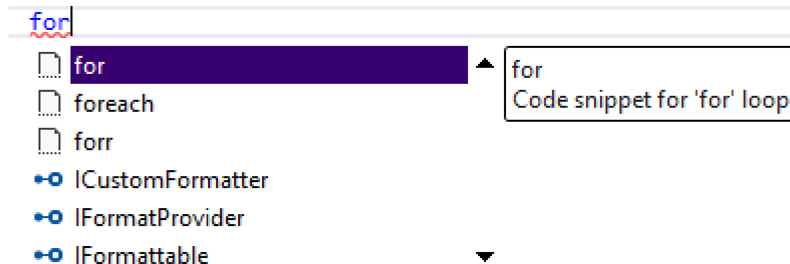
להסבר וויזואלי של לולאה זאת ניתן להיעזר בתרשימים הבאים:





© BRK.

כפי שהדגמנו זאת עם השימוש ב if, גם כאשר נתחיל להזין את הפקודה for בסביבת הפיתוח visual studio והיא תציעה רשימה של השלמות אפשריות. אם נקליד את הפקודה for עד שהיא תבחר מתוך הרשימה הנפתחת ונלחץ על לחצן ה Tab שבמקלדת, היא תושלם ותופיע על המסך.



בשלב זה, אם נקיש פעם נוספת על הלחצן Tab, סביבת הפיתוח תשלים את כל כתיבת התחביר של for בסיוס הבא:

```
for (int i = 0; i < length; i++)
{

}
```

מיד לאחר הופעת תבנית ה for ניתן לשנות את שם האינדקס למשל ל j או כל שם אחר, ללחוץ על לחצן ה Tab במקלדת ושם זה יופיע בשלושת המקומות בהם היה רשום i לפניכן. לאחר הלחיצה על מקש ה Tab ניתן לשנות את השדה length (כרגע זה שם משתנה שלא קיים בתוכנית שלנו) למספר כלשהו, למשל 5. נוסיף פקודה כלשהיא בלולאה ונקבל:

```
for (int j = 0; j < 5; j++)
    {
     Debug.Print("It's working well");
    }
```

כפי שהתכן ושמתם לב, בסביבת הפיתוח visual studio קיים קיצור נוסף forr. אם נקליד אותו ונלחץ פעמיים על מקש ה Tab שבמקלדת, נראה שנקבל תבנית של לולאת for בה האינדקס הולך וקטן (בשונה מהתבנית הרגילה בה האינדקס הולך וגדל עם ביצוע הלולאות). תבנית זאת יכולה להיות שימושית במקרים מסוימים וכדאי להכיר אותה.

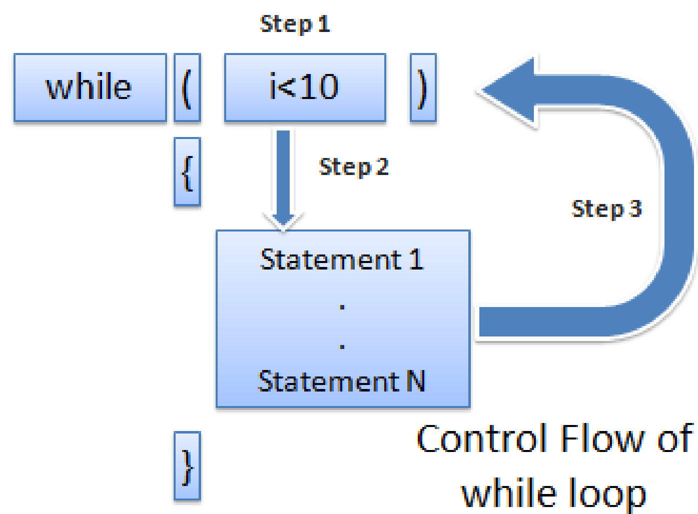
© BRK.

לפני המעבר ללולאה הבאה, יש לציין כי בשפת C# (בשונה משפת C הרגילה) קיימת לולאה הדומה ל for, אך שונה ממנו במקצת והיא foreach. השימוש בלולאה זאת נעשה בקריאה מתוך אוסף כלשהו כדוגמת מערך וכדו'. השימוש בה נוח, אך מאפשר קריאת נתונים בלבד מבלי היכולת לכתוב אותם לתוך האוסף (המערך). המעוניינים, יכולים לתרגל עבודה עם לולאה זאת באופן עצמאי או להיעזר באינטרנט.

while

בנוסף ללולאת for עליה דיברנו, בשפת C# קיימת גם לולאת while. לולאה זאת, בדומה ל for חוזרת על קטע קוד מסוים מספר פעמים, אלא שנוח יותר להשתמש בה במקרים בהם אין אנו יודעים מראש את מספר החזרות שנצטרך לבצע. למשל, התוכנה יכולה לבקש מהמשתמש להקיש על ספרה כלשהי במקלדת וכל עוד המשתמש מקיש על תווים אחרים, כדוגמת אותיות וסימנים שונים, לחזור על הבקשה להקיש דווקא ספרה.

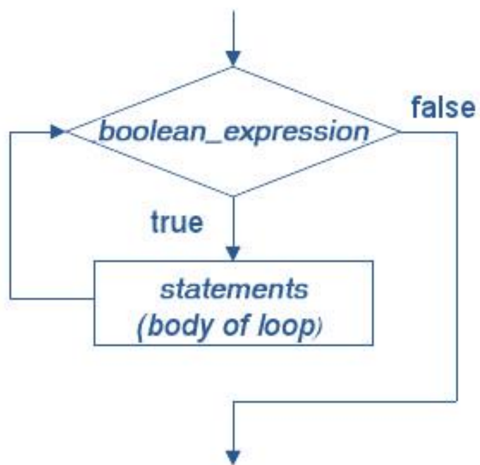
את התחביר ובקרת הזרימה של הלולאה ניתן לראות באיור הבא:



שימו לב שגם במקרה זה אין ; בסוף השורה של while !

© BRK.

תרשים הזרימה של הלולאה היינו:

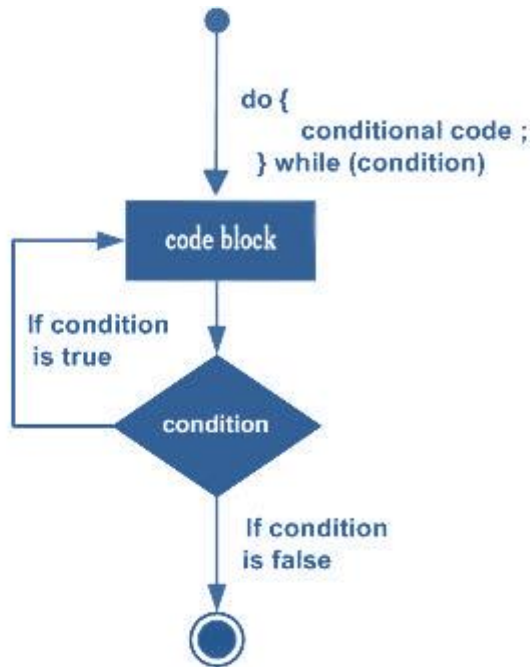


שימוש נפוץ נוסף של הלולאה הוא יצירת לולאה אינסופית. ניתן לעשות זאת גם ע"י לולאת for, אך נוח יותר ליישם אותה דווקא עם while. הלולאה האינסופית יכולה להיות שימושית למשל בסיום התוכנית. אין להשאיר את התוכנית לרוץ הלאה מאחר והתוצאה של זה היינה בלתי צפויה, אלא להכניסה ללולאה אינסופית שלא מבצעת כלום.

```
while (true);
```


© BRK.

do while



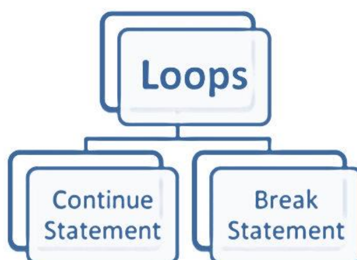
נוסף על הלולאה while בה קודם כל אנו בודקים האם התנאי מתקיים ורק לפי התוצאה מחליטים האם לבצעה את הקטע קוד, קיימת לולאה דומה נוספת שקודם מבצעת את הקוד פעם אחת ולאחר מכן בודקת את התנאי בו צריך לחזור על הקוד פעם נוספת. לולאה זאת נקראת while do. תחביר הפקודה ותרשים הזרימה שלה נתונים באיור הבא:

גם ע"י לולאה זאת ניתן לממש את הלולאה האינסופית, אלא שהשימוש בה לצורך זה נפוץ פחות:

```
do
{

} while (true);
```

continue



משפט זה משמש בלולאות לשם מעבר לתחילת הלולאה והמשך ביצוע התוכנית משם. בכדי להמחיש את השימוש במשפט זה נתבונן בדוגמא הבאה המדפיסה בחלונית ה output את המספרים הזוגיים בלבד בין 0 ו 100:

```
for (int i = 0; i < 100; i++)
{
     if (i%2 != 0)
     {
       continue;
     }
     Debug.Print(i.ToString());
}
```


© BRK.

ההוראה continue תתבצע רק כאשר המספר i מתחלק ב 2 עם שארית (כלומר לא מתחלק ב 2 ללא שארית). הוראה זאת תגרום למעבר לראש הלולאה וקידום האינדקס i ב 1 (i++). השורה השולחת נתונים לחלונית ה output לא תתבצע במקרה זה.

break

משפט זה מבצע סיום הלולאה ויציאה מידית ממנה ללא כל תנאי. כך בדוגמא הבאה המדפיסה בחלונית ה output את כל המספרים מ 0 עד 100, נעשה שימוש בלולאה אינסופית, כאשר תנאי היציאה ממנה הוא שתוכן המשתנה a יהיה גדול מ 100:

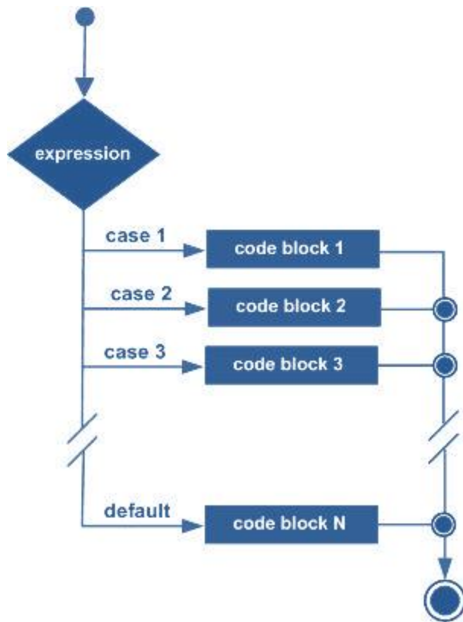
```
int a = 0;
while(true)
{
    if (a > 100)
    {
        break;
    }
    Debug.Print(a.ToString());
    a++;
}
```

goto

בשפת C#, כמו גם בשפות שקדמו לה, ישנו משפט goto המורה לעבור למקום אחר בתוכנית המסומן בתווית ששמה מופיע אחרי ה goto ולהמשיך את ביצוע התוכנית משם. לדוגמא, הפקודה goto drive; תחפש בתוכנה את התווית בשם drive ותמשיך את ביצוע התוכנית ממנה. נציין, כי אחרי שם התווית יש לרשום את התו : בכדי שהמהדר יתייחס למילה שרשומה לפני ה : כשם התווית. באופן כללי, מומלץ להימנע ככל האפשר מהשימוש במשפט ה goto מאחר והם שוברים את רצף ביצוע התוכנית ויכולים להוות מקור לתקלות.

© BRK.

switch



פקודה זאת, כשמה, מבצעת מיתוג בין האפשרויות הקיימות. למשל, אם קיימים רק מספר מצומצם של אפשרויות ערך משתנה מסוים, כמו קומות בבקר מעלית, נוח להשתמש בפקודה זאת בכדי להגדיר מה על הבקר לעשות בכל מקרה נתון. בכדי למנוע מצבים בלתי צפויים, ניתן ומומלץ (אך לא חובה) להגדיר את מצב ברירת המחדל של הפקודה, כלומר המצב אליו תעבור התוכנה במקרה שבמשתנה המדובר קיים תוכן חורג שלא תואם אף אחד מהמקרים האפשריים. ניתן להיעזר בתרשים זרימה שמשמאל בכדי להבין את אופן ביצוע הפקודה.

תחביר הפקודה:

```
switch (שם משתנה המכיל את המידע)
        {
            case אפשרות 1 של תוכן המשתנה :
                    קטע קוד שיתבצע
                    break;
            case אפשרות 2 של תוכן המשתנה :
                    קטע קוד שיתבצע
                    break;
            …
            default:
                    קטע קוד שיתבצע
                break;
        }
```


© BRK.

דוגמא:

```
switch (name)
            {
                case "Avi":
                            Debug.Print("You are menager");
                            break;
                case "Itzik":
                            Debug.Print("You are super user");
                            break;
                case "Kobi":
                            Debug.Print("You are registred user");
                            break;
                default:
                            Debug.Print("You are gest");
                    break;
            }
```

ניתן גם לשלב caseים ו/או לקבוע מעברים בניהם:

```
switch (str)
        {
            case "1":
            case "small":
                cost += 25;
                break;
            case "2":
            case "medium":
                cost += 25;
                goto case "1";
            case "3":
            case "large":
                cost += 50;
                goto case "1";
            default:
                Debug.Print("Invalid selection.");
                break;
        }
```


BRK ©.

תכנות מתקדם

לאחר שעברנו על מושגי יסוד בשפת #C, בפרק זה אנו נעסוק בנושאים מתקדמים יותר המבוססים על הנלמד בפרק הקודם.

פונקציות / מטודות

בספרות שונה אתם יכולים לפגוש את השם פונקציה תחת שמות נרדפים שלה כדוגמת:

שגרה (באנגלית **Subroutine**)

פרוצדורה (באנגלית **Procedure**)

שיטה (באנגלית **Method**)

בשפת #C מקובל להשתמש דווקא במושג המעוברת האחרון – מטודה.

לא נעמוד כאן על ההבדלים הדקים שישנן בין מושגים אלו ונתייחס אליהם כשמות נרדפים לפונקציה.

הגדרתית:

פונקציה / מטודה, היא רצף של פקודות המאוגדות יחדיו, במטרה לבצע מטלה מוגדרת.

כך, לדוגמא, במקום לכתוב שורות ארוכות של קוד בתוך ה ()Main המגדיר את הנסיעה קדימה של רובוט מסוים, נעשה זאת במטודה המופיעה במקום אחר בתוכנית זו או אף בקובץ אחר ובתוך ה()Main נרשום רק את הקריאה למטודה זו. צורת כתיבת קוד המשתמשת המטודות עבור כל פעולה נדרשת, מקלה מאוד על הבנת הקוד שנכתב, מאפשרת שינוי עתידי קל וחלוקה של התוכנית הכוללת למספר מתכנתים כך שכל אחד יכתוב מספר מטודות שיחוברו יחד בתוכנית.

מתודה יכולה לקבל ערך או מספר ערכים מהמפעיל שלה, כלומר מהחלק בתוכנה בה קראו למטודה זאת להתחיל לבצע את התוכנית שלה. יחד עם זאת, יכולות להיות מטודות שלא מקבלות ערך מהמפעיל כלל. כך באותה הדוגמא של רובוט שנוסע ישר ניתן לכתוב מטודה המפעילה את הנעת הרובוט קדימה מבלי לקבל נתון כלשהו. במצב זה הרובוט, למשל, ינוע קדמה עד שתופעל מתודה אחרת.

מתודות יכולות גם להחזיר ערך או ערכים ויכולות גם לא לעשות זאת – תלוי בצורך. נרצה שמתודה המבצעת פעולה מתמטית על המספרים שהועברו אליה, תחזיר את ערך התוצאה שהתקבלה, אך מתודה המניעה את הרובוט יכולה גם שלא להחזיר שום ערך למפעיל.

© BRK.

דוגמא 1

```
private static int max(int num1, int
num2)

{

   int result;


   if (num1 > num2)

      result = num1;

   else

      result = num2;


   return result;

}
```

השורה הראשונה בה מגדירים את המטודה נקראת signature – חתימת המטודה. בשורה זו אנו רואים את שם המטודה: max. היא מקבלת 2 ערכים מסוג int בשם num1 ו num2 ומחזירה ערך מסוג int למפעיל שלה. (את המשמעות של private ו static נסביר בפרק הבא המדבר על מחלקות).

בתוך גוף הפונקציה (בין הסוגריים המסולסלים – { }) ישנו הגדרת משתנה מסוג int בשם result. משתנה זה יהיה מוכר (ניתן לעשות בו שימוש) רק בתוך מטודה זו. במידה וננסה להשתמש במשתנה זה מחוץ למטודה – נקבל שגיאת קומפילציה.

התנאי בגוף הפונקציה בודק אם המספר הראשון שהועבר אליו גדול מהמספר השני. במידה וכן – מכניס את המספר הראשון לתוך משתנה פנימי result ובמידה ותנאי זה לא מתקיים – מכניס את המספר השני לתוך המשתנה result.

לבסוף, בכל מצב של התנאי עליו דיברנו, הפונקציה תחזיר למפעיל שלה את התוכן של המשתנה result. יוצא שהערך המוחזר הוא המספר הגבוהה מבין השתיים שהתקבלו.

© BRK.

דוגמא 2

```
private int Subtract(int firstNumber, int secondNumber)
{
    int answer;

    answer = firstNumber - secondNumber;

    return answer;
}
```

דוגמא 3

```
static int CalculatePlayerScore()
{
    int livesLeft = 2;
    int tanksDestroyed = 17;
    int coptersDestroyed = 4;
    int JetDestroyed = 1;

    if(livesLeft == 0)
    {
        return 0;
    }

    return tanksDestroyed * 10 +
           coptersDestroyed * 100 +
           JetDestroyed * 1000;
}
```

בדוגמא זו אנו רואים שימוש במטודה לחישוב ניקוד שחקן במשחק מסוים. מטודה CalculatePlayerScore לא מקבלת ערך מהמפעיל שלה ולכן הסוגריים שאחרי השם שלה ריקות. המטודה מחזירה מספר שלם למפעיל שלה ולכן לפני שמה מופיע int. ארבעת השורות הראשונות שבגוף המטודה מגדירות ארבעה משתנים פנימיים ומאתחלים אותם עם ערכים. בתוכנית אמתית ערכים אלו יוספקו באופן אחר, אך מאחר וטרם דיברנו על מחלקות ואובייקטים – נסתפק באופן זה.

© BRK.

בשורה 5 מתבצעת בדיקה האם השחקן נהרג במשחק, כלומר מספר החיים שנותרו לו שווה לאפס. ע"פ התוכנית הנתונה, במצב זה כל הניקוד של השחקן נמחק כי המטודה מחזירה ערך 0. במידה והשחקן לא נהרג במשחק, המטודה תזכה אותו ב 10 נקודות על כל טנק שהושמד, 100 נקודות על כל מסוק שהושמד ו 1000 על כל מטוס. תוצאת חישוב הניקוד תוחזר למפעיל המטודה.

מחלקות

Privet / public

מילון מושגים

https://en.wikibooks.org/wiki/C_Sharp_Programming

מפת הדקים של הכרטיס

© BRK.

אזהרות

- הדקי I/O ספרתיים של הרכיב בנויים לעבוד במתח של 3.3V ועמידים גם במתח של 5V.
- שים לב:

מתח המרה מרבי אותו ניתן לספק בכניסה אנלוגית של רכיב הוא 3.6V בלבד.
הספקת מתח חורג ממגבלה זו תשרוף את הכניסה האנלוגית בה הוא סופק!

- כל כניסה אנלוגית של הרכיב בעלת התנגדות של 50kΩ
- מתח מרבי אותו ניתן לקבל ביציאה אנלוגית של הרכיב היינו 3.3V
- צריכת זרם של הרכיב

Symbol	Ratings	Max.	Unit
I_{VDD}	Total current into V_{DD} power lines (source)[1]	240	mA
I_{VSS}	Total current out of V_{SS} ground lines (sink)[1]	240	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current source by any I/Os and control pin	25	
$I_{INJ(PIN)}$ [2]	Injected current on five-volt tolerant I/O[3]	–5/+0	
	Injected current on any other pin[4]	±5	
$\Sigma I_{INJ(PIN)}$[4]	Total injected current (sum of all I/O and control pins)[5]	±25	

© BRK.

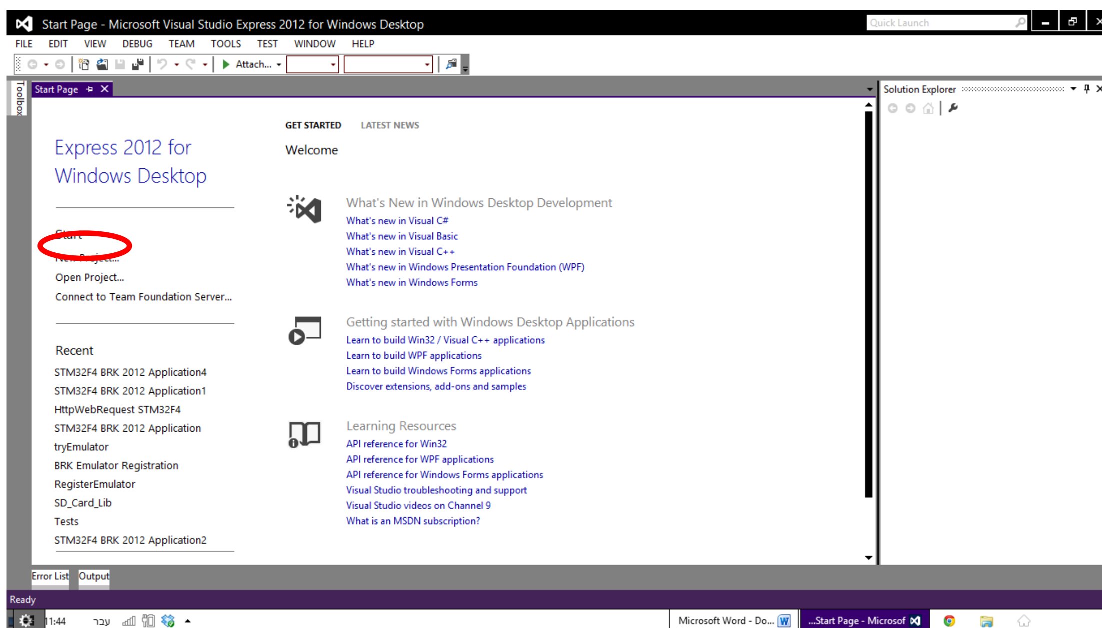
נספחים

© BRK.

נספח יצירת פרויקט חדש ב Visual Studio 2012

1. חברו את כרטיס הבקר למחשב ע"י **<u>שני</u>** חיבורי USB.
2. וודאו שנדלקו לפחות 2 לדים אדומים ולד ירוק אחד ליד חיבורי ה USB.
   a. במידה והלדים לא דולקים, בדקו אם חיברתם טוב את החיבורים ואם הבעיה לא נפתרת יש לבקש עזרה מהמורה.

3. על שולחן העבודה שבמחשב עליו אתם עובדים, תלחצו פעמיים עם העכבר על קיצור הדרך של Microsoft Visual Studio Express 2012 for Windows Desktop.
   במידה וקיצור דרך זה לא קיים שם, ניתן למצוא את התוכנה בתוך רשימת כל התוכניות המותקנות במחשב.
4. יפתח החלון המצולם באיור הבא:

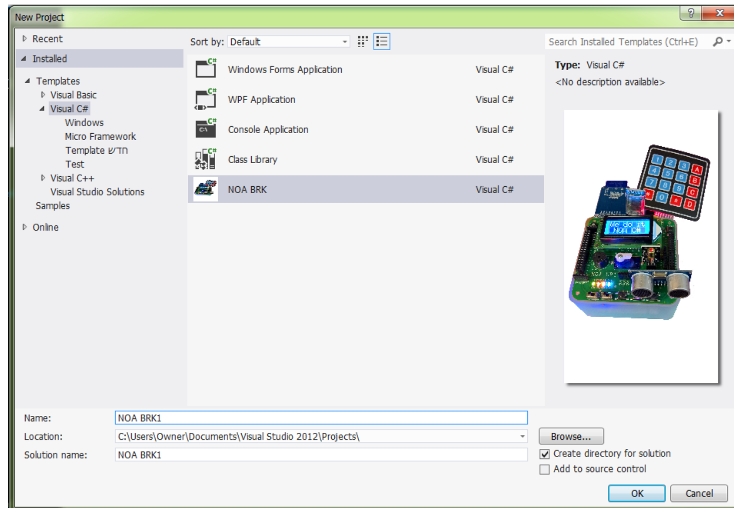


5. בלשונית Start Page שנפתחה בחלקו השמאלי של החלון, יש לבחור עם העכבר את האפשרות New Project.

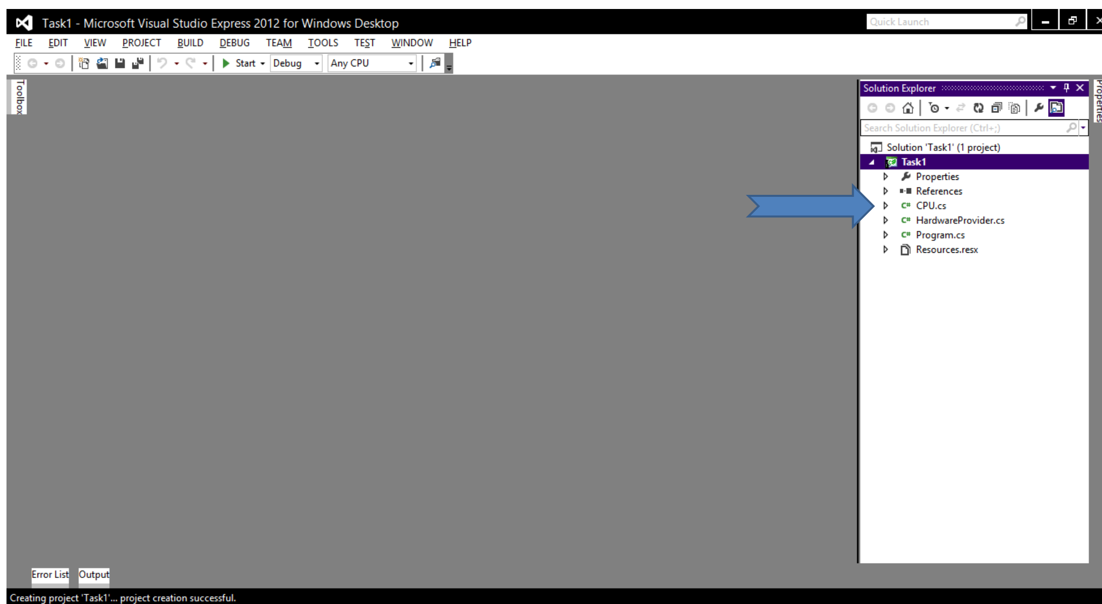
הערה: במידה ולשונית Start Page לא נפתחה, יש לבחור באפשרות New Project מתוך התפריט File

© BRK.

6. יפתח החלון הבא:

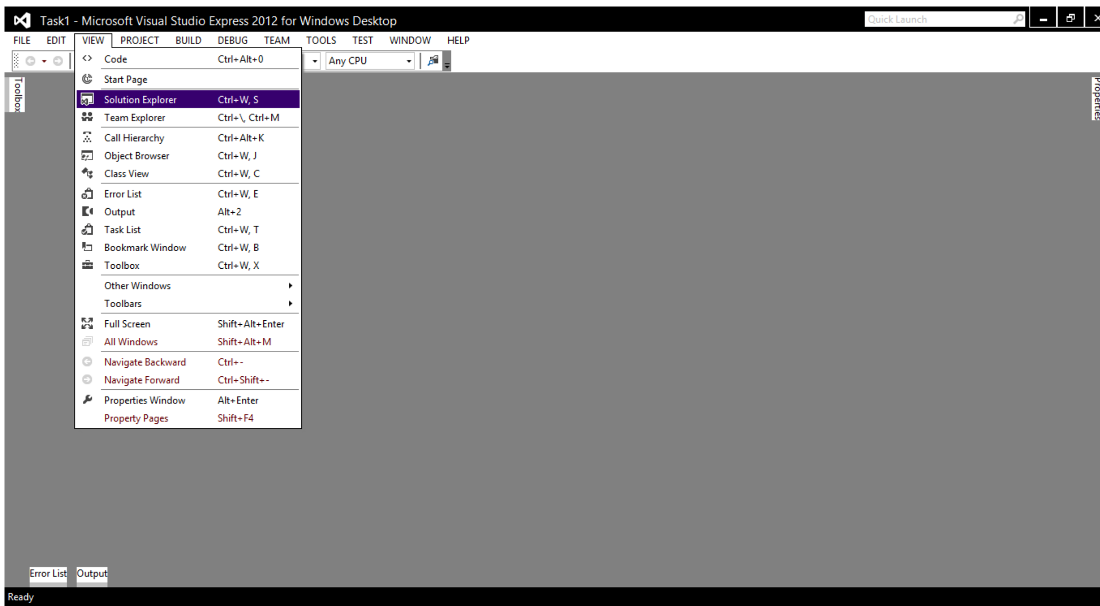


7. בחלקו השמאלי של החלון שיפתח יש לפתוח את התפריט Visual C# ולסמן את האפשרות NOA BRK כמתואר באיור.
8. בשדה Name שבחלקו התחתון של החלון יש לתת שם לתוכנית (הפרויקט). הקפידו לתת שמות הגיוניים לתוכניות המבטאות את מה שהיא תבצעה. יש לתת שמות באנגלית בלבד.
9. בשדה Location שבחלקו התחתון של החלון הכניסו את המיקום במחשב בו ישמר הפרויקט אותה אתם יוצרים כרגע. ניתן להיעזר בלחצן Browse למציאת המיקום הרצוי.
10. וודאו כי קיים סימן V בתיבת Create directory for solution
11. לחצו על הלחצן OK ליצירת הפרויקט.
12. יפתח החלון הבא:

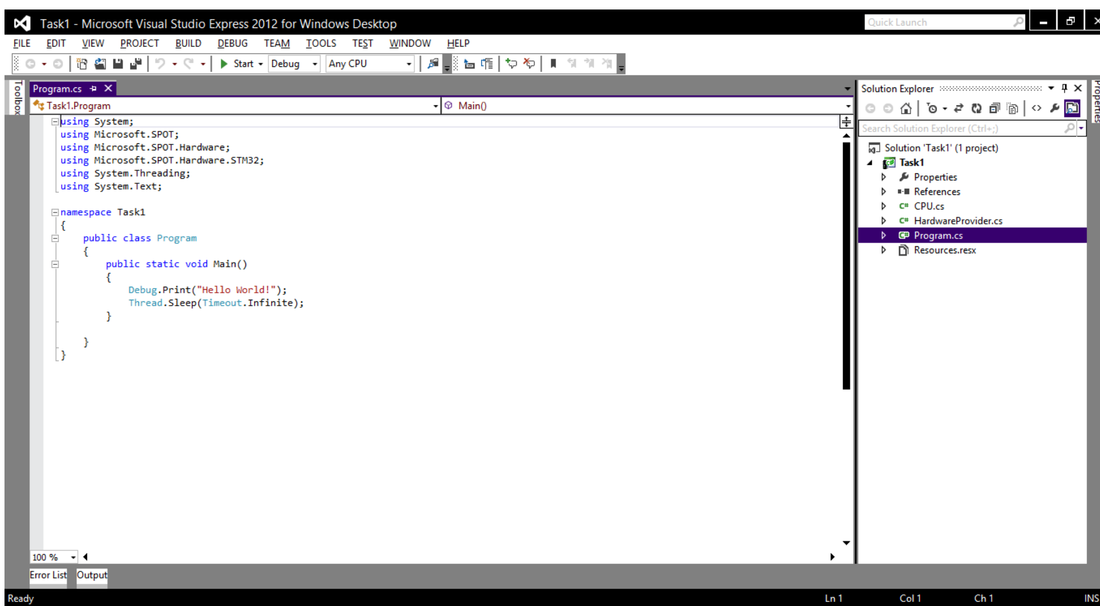


13. הקישו הקשה כפולה על הקובץ Program.cs במופיע בחלונית Solution שבחלקו הימני של המסך. במידה והחלונית Solution לא מופיעה על המסך, ניתן לבחור אפשרות הצגה שלה מתוך תפריט View כמתואר באיור הבא:

© BRK.

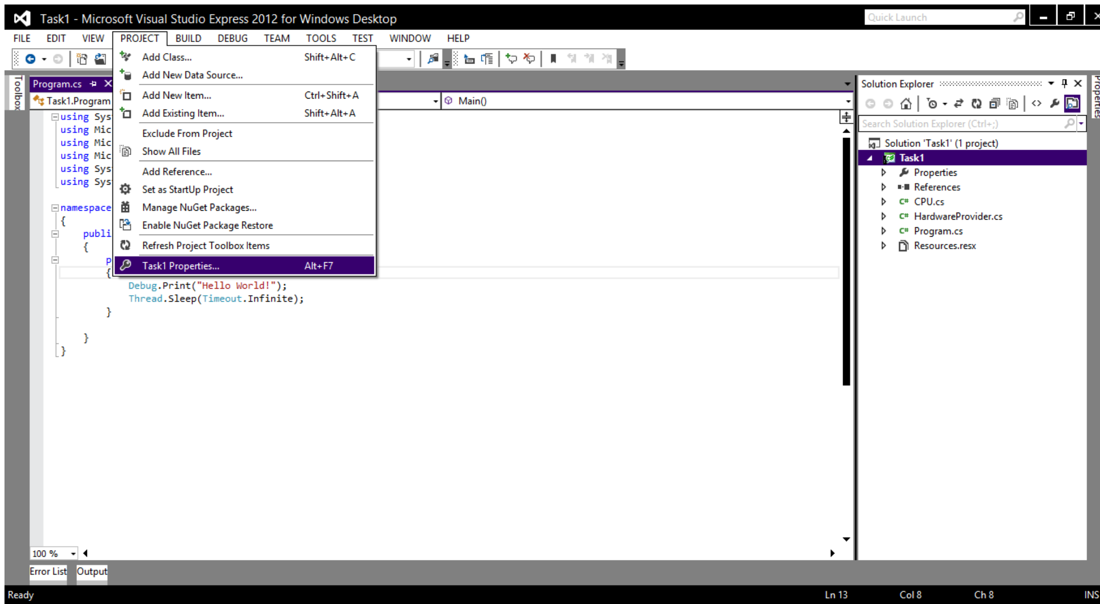


14. אחרי הלחיצה הכפולה עם העכבר על Program.cs תפתח החלונית כמתואר באיור הבא:

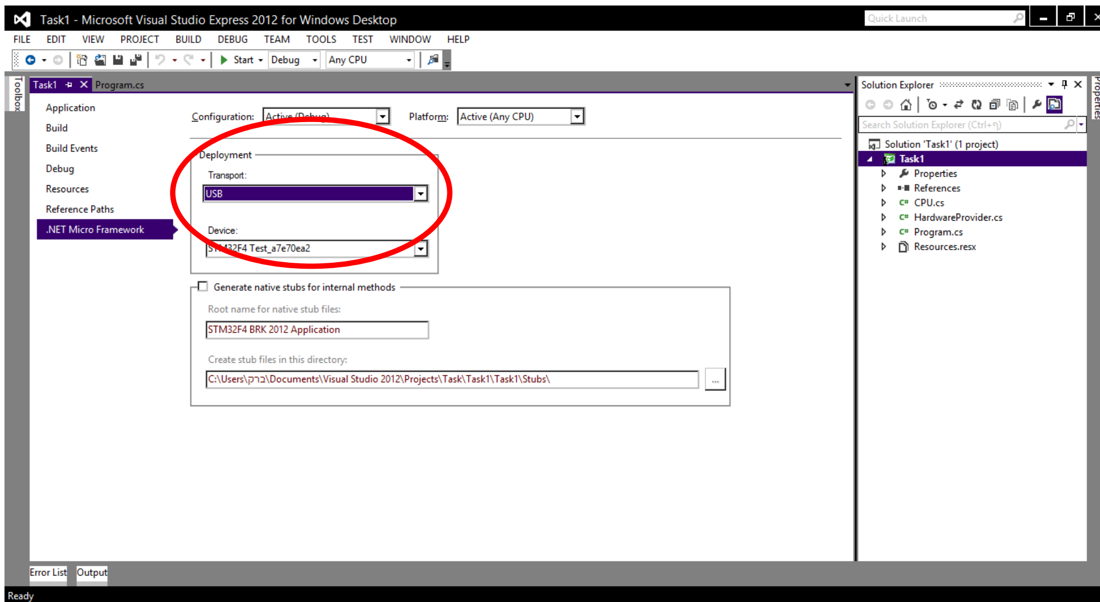


15. לפני צריבת הפרויקט בתוך הבקר, נבדוק את הגדרת הערכה במאפייני הפרויקט. לשם כך נבחר את מאפייני הפרויקט (Properties) בתוך תפריט ה Project. ניתן גם להגיע למאפייני הפרויקט ע"י הקשה בו זמנית על הלחצנים Alt ו F7. ניתן להיעזר באיור הבא:

© BRK.



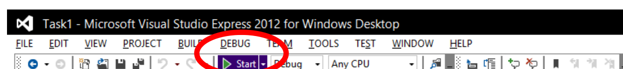
16. יפתח החלון של מאפייני הפרויקט



17. וודאו שבלשונית .Net MicroFramework ההגדרות של Deployment הם כבאיור.

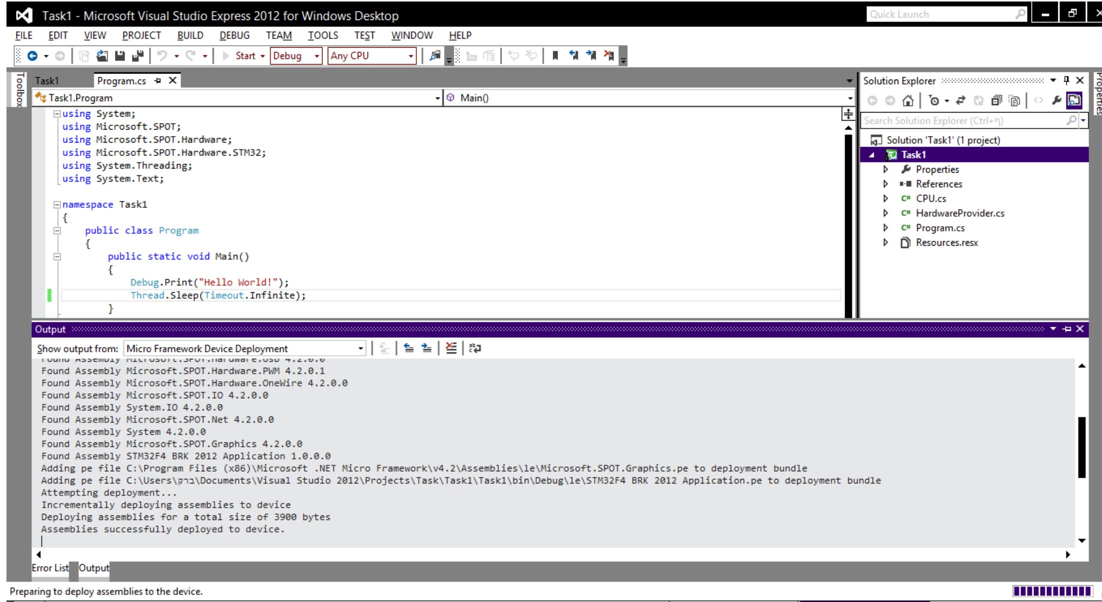
אם לאחר שבחרתם USB בשדה ה Transport אין את ה STM32F4 ברשימת ההתקנים שבשדה Device, וודאו ששני החיבורים של USB מחוברים לכרטיס פיתוח ולמחשב (על הכרטיס דולקים 2 לדים אדומים ולד ירוק אחד).

18. נריץ את הפרויקט ונצרוב אותו לבקר ע"י לחיצה על Start בשורת הפקודות

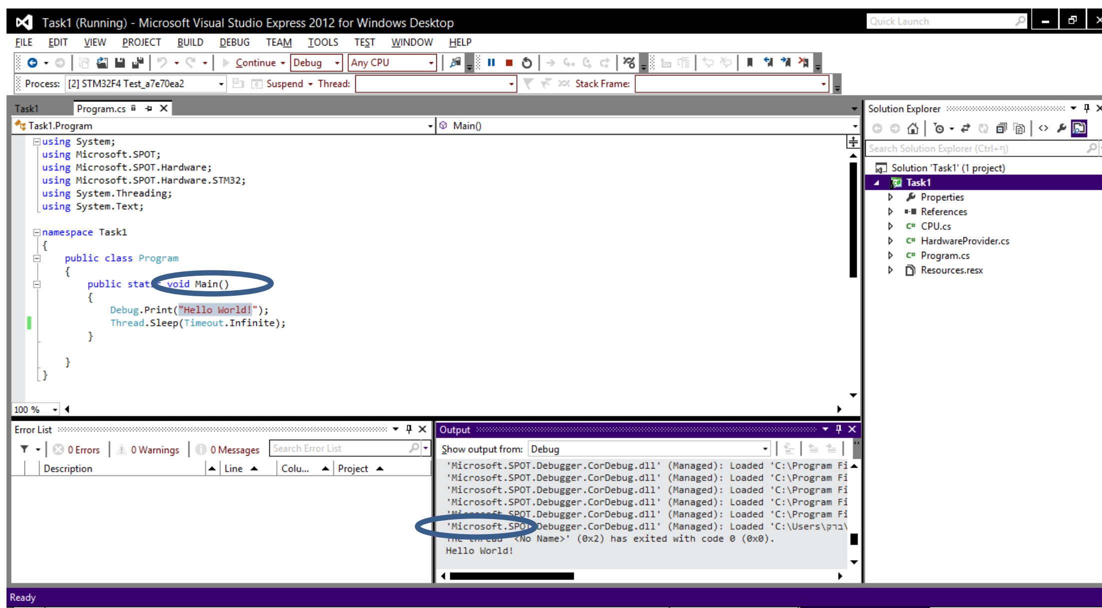


©BRK.

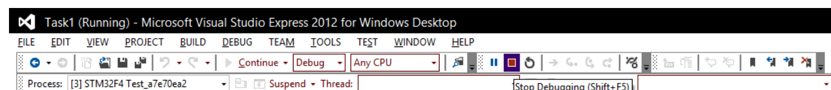
19. בחלקו התחתון של החלון תפתח חלונית Output המהווה פלט של המערכת



20. בסיום התהליך יופיע הכיתוב Hello world שקיבלנו רשום בתוכנית



21. בסיום ההרצה, בכדי לערוך שינויים בתוכנית, יש לעצור את ההרצה ע"י הקשה על Stop Debbuging בשורת הפקודות, כמתואר באיור



.BRK ©

22. נציין, כי ניתן לצרוב את הבקר גם ע"י הלחיצה על לחצן F5.
23. אפשרות נוספת היא לצרוב את הבקר מבלי אפשרות ה debugging (חיפוש ותיקון שגיאות בזמן הרצת התוכנית). צריבה זאת מהירה יותר וניתן לבצעה אותה ע"י הקשה על שני הלחצנים יחד: Ctrl + F5.

© BRK.