

# אופרטורים ביטויים לוגיים

יסודות מדעי המחשב



## ביחידה זו נלמד:

- פעולות ארכיטמטיות
- ביטויים חשבוניים
- אופרטור **++** ואופרטור **--**
- ביטויים מקוצרים
- המרות בין טיפוסים (casting)
- ביטויים לוגיים

## פעולות חשבוןות

```
static void DivideAndModulo()
{
    int x = 13, y = 5;
    double z = 13.5;

    Console.WriteLine (x / y);      2
    Console.WriteLine (z / y);      2.7
    Console.WriteLine (x % y);      3
    Console.WriteLine (z % y);      3.5
}
```

ניתן לחשב ביטויים בעזרת הפעולות הבאות :

- חיבור +
- חיסור -
- כפל \*
- חילוק /

- פעולה חילוק שני מרכיבים שלמים מחזירה את מנת החלוקה (החלק השלם של התוצאה).  
אם אחד מהרכיבים בפעולה הוא מספר ממשי – מנת החלוקה תהיה מספר ממשי.
- אסור לחלק ב-0 !
- מודולו % (פעולה חילוק המחזירה את תוצאה השארית של החלוקה).
- אסור לחלק ב-0 !

### בפעולה חשבוןות :

- אם שני המרכיבים הם מספרים שלמים – התוצאה תהיה מספר שלם.
- אם לפחות אחד מהרכיבים בפעולה הוא מספר ממשי – התוצאה תהיה מספר ממשי.

## שימוש בפעולות חשבוןות : **%** בדיקה האם מספר זוגי

- כאשר מחלקים מספר ב-2 התוצאה תמיד תהיה 0 או 1.
- אם שארית חלוקה ב-2 היא 0, המספר זוגי.
- אם שארית חלוקה ב-2 היא 1, המספר אי-זוגי.

```
static void Modulo2()
{
    Console.WriteLine (23 % 2);      1
    Console.WriteLine (24 % 2);      0
    Console.WriteLine (25 % 2);      1
    Console.WriteLine (26 % 2);      0
    Console.WriteLine (27 % 2);      1
}
```

## שימוש בפעולות חשבון: **%10** קבלת סכום אחדות

- שימוש ב-**%10** כדי לקבל את הספירה הימנית של מספר, היא סכום אחדות.

```
static void Modulo10()
{
    Console.WriteLine (367 % 10);
    Console.WriteLine (95 % 10);
    Console.WriteLine (8 % 10);

    Console.WriteLine (357 % 100);
    Console.WriteLine (95 % 100);
    Console.WriteLine (8 % 100);
}
```

7  
5  
8  
  
57  
95  
8

## שימוש בעולות חשבוניות : 10/ קיצוץ ספרת האחדות

- שימוש ב-10/ כדי לקצץ את הספרה הימנית של מספר, היא ספרת האחדות.

```
static void Modulo10()
{
    Console.WriteLine (367 / 10);
    Console.WriteLine (95 / 10);
    Console.WriteLine (8 / 10);

    Console.WriteLine (357 / 100);
    Console.WriteLine (95 / 100);
    Console.WriteLine (8 / 100);
}
```

36  
9  
0  
  
3  
0  
0

שאלה:  
מדוע ישן פעולות שנותנות/מקצתות  
את הספרה הימנית, אך אין פעולות  
דומות עברו הספרה השמאלית ?

## דוגמה : מספר המיצג שעה

```
Enter the time: 1245  
The time is 12:45
```

```
static void ShowTime()  
{  
    int theTime;  
    int hours, minutes;  
  
    Console.Write("Enter the time: ");  
    theTime = int.Parse(Console.ReadLine());  
  
    hours = theTime / 100;  
    minutes = theTime % 100;  
  
    Console.WriteLine("The time is " + hours + ":" + minutes);  
}
```

- קלטו מהמשתמש מספר המיצג שעה.  
למשל : 1245 מייצג את השעה 12 ו-45 את הדקות
- הציגו הודעה ברורה מהי השעה.
- לקבלת השעות :  
 $1245 / 100 = 12$
- לקבלת הדקות :  
 $1245 \% 100 = 45$

# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין. ↙

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); █
}
```

# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין. ↙

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); █
}
```

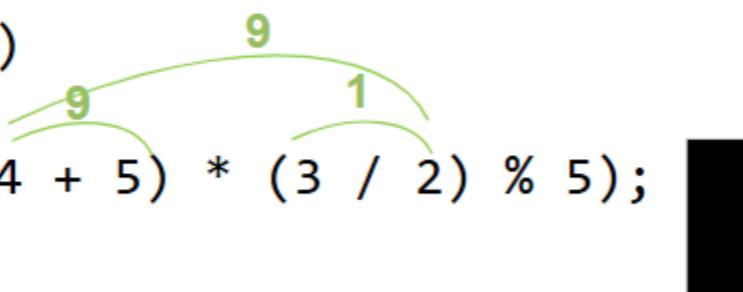
# קידמיות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- לבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); }
```



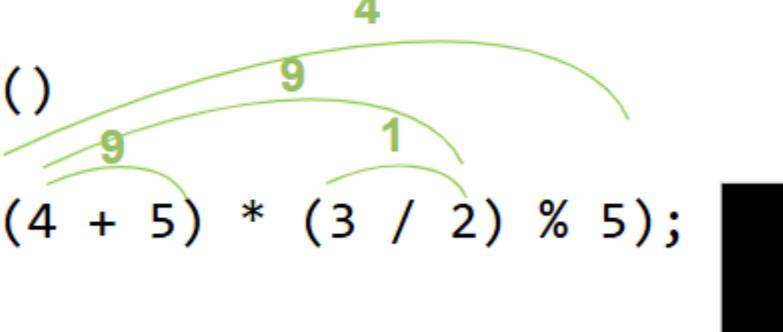
# קדמיות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- לבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); }
```



# קידימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
}
```

# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5);
}
```

# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

↳ במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5);
}
```

# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין. 

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5); 5
}
```

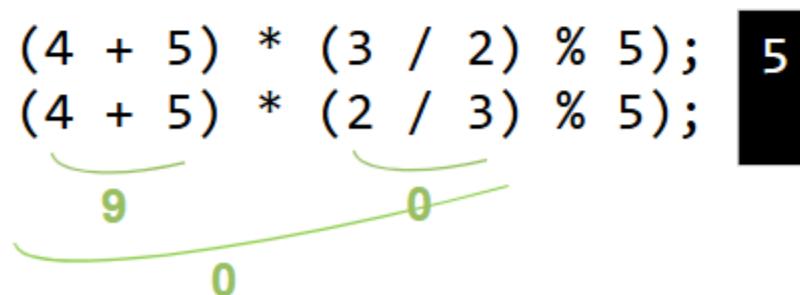
# קידמיות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- לבסוף חיבור וחיסור

↳ במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5); 5
}
```



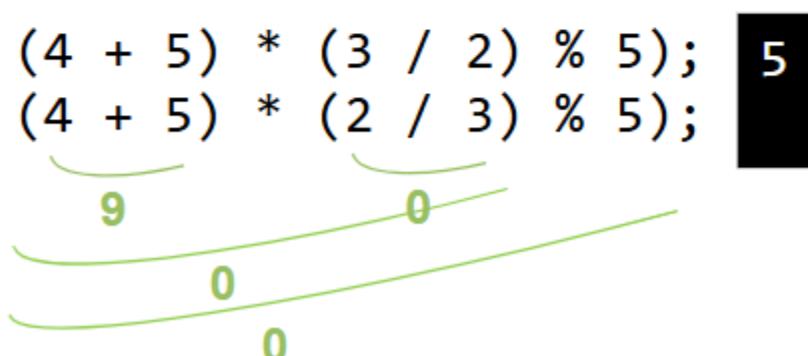
# קידימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- לבסוף חיבור וחיסור

↙ במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5);
```



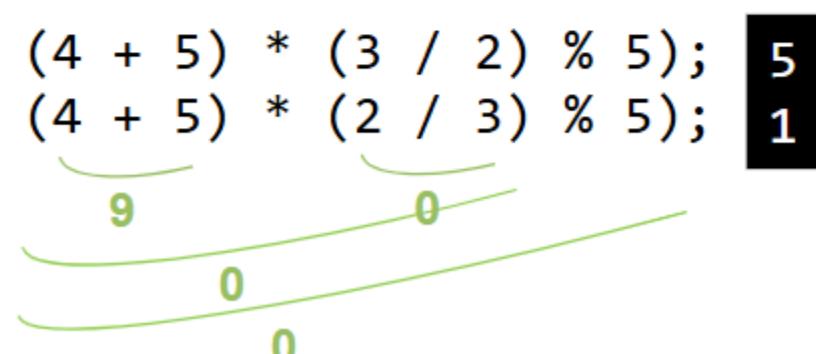
# קדימות הפעולות

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- לבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5);
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5);
}
```



# קדמיות הפעולות

סדר הפעולות :

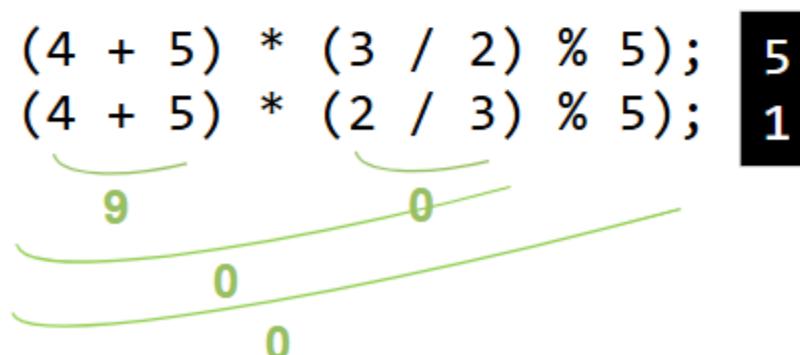
- רמה 1 : ()
- רמה 2 : +, -
- רמה 3 : \*, /
- רמה 4 : -, =
- רמה 5 : =

כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ כפל, חילוק ומודולו
- ולבסוף חיבור וחיסור

במידה ויש כמה פעולות באותו רמת עדיפות החישוב יבוצע משמאל לימין.

```
static void OperatorsOrder()
{
    Console.WriteLine(1 + (4 + 5) * (3 / 2) % 5); 5
    Console.WriteLine(1 + (4 + 5) * (2 / 3) % 5); 1
}
```



## ביטוי חשבוני : שימוש

- ניתן להציג ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7+n = \text{sum}$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמור את ערכו בתחום משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());
}

}
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7+8 = sum$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמר את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());
}
}
```

```
Please enter 2 numbers:  
3  
5
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7+8 = \text{sum}$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמור את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו

```
Please enter 2 numbers:  
3  
5
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7+8 = \text{sum}$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמר את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());
    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו

```
Please enter 2 numbers:
3
5
The sum is 8
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7 + n = \text{sum}$ .
- ניתן להשתמש בביטוי חשבוני לשירות בתוכנית, או לשמור את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);

    Console.WriteLine ("The sum is " + (n1 + n2));
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו

שימוש בביטוי ישירות  
בהוראת הדפסה.

```
Please enter 2 numbers:  
3  
5  
The sum is 8
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7+8 = \text{sum}$ .
- ניתן להשתמש בביטוי חשבוני לשירות בתוכנית, או לשמר את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);

    Console.WriteLine ("The sum is " + (n1 + n2));
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו

שימוש בביטוי לשירות  
בהוראת הדפסה.

```
Please enter 2 numbers:
3
5
The sum is 8
The sum is 8
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7 + a = \text{sum}$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמור את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);

    Console.WriteLine ("The sum is " + (n1 + n2));
    Console.WriteLine ("The sum is " + n1 + n2);
}
```

השמה של תוצאת הביטוי  
במשתנה, לפני הדפסתו

שימוש בביטוי ישירות  
בהוראת הדפסה.

```
Please enter 2 numbers:  
3  
5  
The sum is 8  
The sum is 8
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7 + n = \text{sum}$ .
- ניתן להשתמש בביטוי חשבוני לשירות בתוכנית, או לשמור את ערכו בתחום משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);

    Console.WriteLine ("The sum is " + (n1 + n2));
    Console.WriteLine ("The sum is " + n1 + n2);
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו  
שימוש בביטוי ישירות  
בhorאת הדפסה.

```
Please enter 2 numbers:
3
5
The sum is 8
The sum is 8
The sum is 35
```

## ביטוי חשבוני : שימוש

- ניתן להגדיר ביטוי חשבוני על ידי מספרים, משתנים ואופרטורים. לדוגמה:  $7 + a = \text{sum}$ .
- ניתן להשתמש בביטוי החשבוני לשירות בתוכנית, או לשמור את ערכו בתוך משתנה.

```
static void UsingOperator()
{
    int n1, n2, sum;
    Console.WriteLine ("Please enter 2 numbers: ");
    n1 = int.Parse (Console.ReadLine());
    n2 = int.Parse (Console.ReadLine());

    sum = n1 + n2;
    Console.WriteLine ("The sum is " + sum);

    Console.WriteLine ("The sum is " + (n1 + n2));
    Console.WriteLine ("The sum is " + n1 + n2);
}
```

השמה של תוצאה הביטוי  
במשתנה, לפני הדפסתו

שימוש בביטוי ישירות  
בהוראת הדפסה.

```
Please enter 2 numbers:  
3  
5  
The sum is 8  
The sum is 8  
The sum is 35
```

- כאשר משרשרים מחרוזת עם מספר באמצעות האופרטור **+**,  
החיבור הוא למעשה שרשור של שתי מחרוזות.
- אם רוצים לשרשר את תוצאה החיבור של שני מספרים  
יש לעטוף את הפעולה החשבונית בסוגרים.

## ביטויים מקוצרים : אופרטור **++**, אופרטור **--**

- כדי להגדיל/להקטין משתנה ב-1 :

**x = x - 1;**

**x = x + 1;**

- קיים תחביר מיוחד ומקוצר עבור מקרים אלו :

**++** עבור הגדלת משתנה ב-1: **++x** או **x++**

**--** עבור הקטנת משתנה ב-1: **--x** או **x--**

- לומר:

**x=x+1;**     $\equiv$     **x++;**     $\equiv$     **++x;**

**x=x-1;**     $\equiv$     **x--;**     $\equiv$     **--x;**

## הגדלה/הקטנה ב-1 : ההבדל בין $X++$ ל- $+X$

- כאשר משתמשים באופרטור  $++$  לצורך השמה למשתנה אחר יש הבדלים בין תוצאות הביטויים:

ערך של  $x$  לא משתנה בעקבות פקודת זו,  $y$  משתנה  
ערך של  $x$  גדל ב-1 בעקבות פקודת זו,  $y$  משתנה  $\left\{ \begin{array}{l} y = x++ \\ y = ++x \end{array} \right.$

## הגדלה/הקטנה ב-1 : ההבדל בין $X++$ ל- $++X$

- כאשר משתמשים באופרטור  $++$  לצורך השמה למשתנה אחר יש הבדלים בין תוצאות הביטויים:

```
static void PrefixPostfix()
{
    int x, y;

    x = 4;
    y = x++;
    Console.WriteLine("x=" + x + " y=" + y);

}
```

ערך של  $x$  לא משתנה בעקבות פקודה זו,  $y$  משתנה  $\left\{ \begin{array}{l} y = x++; \\ y = ++x; \end{array} \right.$

ערך של  $x$  גדל ב-1 בעקבות פקודה זו,  $y$  משתנה

## הגדלה/הקטנה ב-1 : ההבדל בין $X++$ ל- $++X$

- כasher משתמשים באופרטור  $++$  למשתנה אחר יש הבדלים בין תוצאות הביטויים:

```
static void PrefixPostfix()
{
    int x, y;

    x = 4;
    y = x++;
    Console.WriteLine("x=" + x + " y=" + y);
    x=5 y=4
}
```

ערך של  $x$  לא משתנה בעקבות פקודת זו,  $y$  משתנה  
ערך של  $x$  גדל ב-1 בעקבות פקודת זו,  $y$  משתנה

$\left\{ \begin{array}{l} y = x++ \\ y = ++x \end{array} \right.$

## הגדלה/הקטנה ב-1 : ההבדל בין **X++** ל- **++X**

- כאשר משתמשים באופרטור **++** לצורך השמה למשתנה אחר יש הבדלים בין תוצאות הביטויים:

```
static void PrefixPostfix()
{
    int x, y;
    x = 4;
    y = x++;
    Console.WriteLine("x=" + x + " y=" + y);
    x = 4;
    y = ++x;
    Console.WriteLine("x=" + x + " y=" + y);
}
```

ערך של **x** לא משתנה בעקבות פקודה זו, **y** משתנה ←  $y = x+1;$

ערך של **x** גדל ב-1 בעקבות פקודה זו, **y** משתנה ←  $\begin{cases} y = x++ \\ y = ++x; \end{cases}$

- אם האופרטור **++** מופיע מימין למשתנה אז לתוך **y** נכנס הערך המקורי של **x** ורק אז ערך של **x** גדל

## הגדלה/הקטנה ב-1 : ההבדל בין $X++$ ל- $X+$

- כasher משתמשים באופרטור  $++$  למשתנה אחר יש הבדלים בין תוצאות הביטויים:

```
static void PrefixPostfix()
{
    int x, y;
    x = 4;
    y = x++;
    Console.WriteLine("x=" + x + " y=" + y);

    x = 4;
    y = ++x;
    Console.WriteLine("x=" + x + " y=" + y);
}
```

ערך של  $x$  לא משתנה בעקבות פקודה זו,  $y$  משתנה ← ערך של  $x$  גדל ב-1 בעקבות פקודה זו,  $y$  משתנה  $\left\{ \begin{array}{l} y = x++ \\ y = ++x; \end{array} \right.$

אם האופרטור  $++$  מופיע מימין למשתנה אז לתוך  $y$  נכנס הערך המקורי של  $x$  ורק אז ערך של  $x$  גדל

$x=5$	$y=4$
$x=5$	$y=5$

# הגדלה/הקטנה ב-1 : ההבדל בין $X++$ ל- $-X$

- כasher משתמשים באופרטור  $++$  לצורך השמה למשתנה אחר יש הבדלים בין תוצאות הביטויים:

```
static void PrefixPostfix()
{
    int x, y;

    x = 4;
    y = x++;
    Console.WriteLine("x=" + x + " y=" + y);

    x = 4;
    y = ++x;
    Console.WriteLine("x=" + x + " y=" + y);
}
```

ערך של  $x$  לא משתנה בעקבות פקודה זו,  $y$  משתנה ← ערך של  $x$  גדל ב-1 בעקבות פקודה זו,  $y$  משתנה  $\left\{ \begin{array}{l} y = x++ \\ y = ++x; \end{array} \right.$

- אם האופרטור  $++$  מופיע מימין למשתנה אז לתוך  $y$  נכנס הערך המקורי של  $x$  ורק אז ערך של  $x$  גדל
- אם האופרטור  $++$  מופיע משמאלו למשתנה אז קודם ערך של  $x$  גדל ואז ערכו החדש נכנס לתוך  $y$

החוקים עבור האופרטור  $--$  זהים

## ביטויים מקוצרים : אופרטור השמה

- כדי להוסיף ערך כלשהו לממשנה ניתן להשתמש בכתיבה מקוצר:  
 $y = y + 3;$     $\equiv$     $y += 3;$

```
static void ShortOperators()
{
    int x;

    x = 3;
    x = x + 2;

}
```

## ביטויים מקוצרים : אופרטור השמה

- כדי להוסיף ערך כלשהו למשתנה ניתן להשתמש בכתיבה מקוצר:

**y = y+3;    ≡    y += 3;**

```
static void ShortOperators()
{
    int x;

    x = 3;
    x = x + 2;
    Console.WriteLine(x);

}
```

## ביטויים מקוצרים : אופרטור השמה

- כדי להוסיף ערך כלשהו למשתנה ניתן להשתמש בכתיבה מקוצר:  
 $y = y + 3;$   $\equiv$   $y += 3;$

```
static void ShortOperators()
{
    int x;

    x = 3;
    x = x + 2;
    Console.WriteLine(x);

    x = 3;
    x += 2;
}
```

## ביטויים מקוצרים : אופרטור השמה

- כדי להוסיף ערך כלשהו למשתנה ניתן להשתמש בכתיבה מקוצר:  
 $y = y + 3;$   $\equiv$   $y += 3;$

```
static void ShortOperators()
{
    int x;

    x = 3;
    x = x + 2;
    Console.WriteLine(x);

    x = 3;           5
    x += 2;          5
    Console.WriteLine(x);
}
```

## ביטויים מקוצרים : אופרטור השמה

- כדי להוסיף ערך כלשהו למשנה ניתן להשתמש בכתיבה מקוצר:  
 $y = y + 3;$     $\equiv$     $y += 3;$   
ניתן להשתמש בכתיבה מקוצרת זו עבור כל האופרטורים שלמדנו ( $+$  -  $*$  /  $%$ ).

```
static void ShortOperators()
{
    int x;
    x = 3;
    x = x + 2;
    Console.WriteLine(x);

    x = 3;
    x += 2;
    Console.WriteLine(x);
}
```

$$x *= y + 1; \left\{ \begin{array}{l} \equiv x = x * (y + 1); \\ \neq x = x * y + 1; \end{array} \right.$$

5  
5

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

}
```

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
```

}

int n1	0	1000
int n2	0	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
}
```

int n1	0	1000
int n2	0	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
}
```

int n1	2	1000
int n2	0	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
}
```

int n1	2	1000
int n2	0	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
}
```

int n1	3	1000
int n2	0	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
}
```

int n1	3	1000
int n2	3	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--;
}
```

int n1	3	1000
int n2	3	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
```

int n1	3	1000
int n2	3	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
```

int n1	3	1000
int n2	9	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
```

int n1	2	1000
int n2	9	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
}
```

int n1	2	1000
int n2	9	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
}
```

int n1	1	1000
int n2	9	1004
int n3	0	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;
    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
}
```

int n1	1	1000
int n2	9	1004
int n3	9	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
}
```

int n1	1	1000
int n2	8	1004
int n3	9	1008
int n4	0	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;

}
```

int n1	1	1000
int n2	8	1004
int n3	9	1008
int n4	0	1012

## ביטויים מקודרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;

}
```

int n1	1	1000
int n2	8	1004
int n3	9	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;

}
```

int n1	2	1000
int n2	8	1004
int n3	9	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;

}
```

int n1	3	1000
int n2	8	1004
int n3	9	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
}
```

int n1	3	1000
int n2	8	1004
int n3	9	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
}
```

int n1	3	1000
int n2	8	1004
int n3	6	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
}
```

int n1	4	1000
int n2	8	1004
int n3	6	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
}
```

int n1	5	1000
int n2	8	1004
int n3	6	1008
int n4	2	1012

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
    n3 = n4 = n1 - 1;
}
```

int n1	5	1000
int n2	8	1004
int n3	6	1008
int n4	2	1012

האופרטור **++** מימין תמיד פועל בסוף,  
גם אם הוא בסוגרים !

## ביטויים מקודרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
    n3 = n4 = n1 - 1;
}
```

int n1	5	1000
int n2	8	1004
int n3	6	1008
int n4	4	1012

האופרטור `++` מימין תמיד פועל בסוף,  
גם אם הוא בסוגרים !

## ביטויים מקוצרים : דוגמה

```
static void Example1()
{
    int n1, n2, n3, n4;

    n1 = 2;
    n2 = ++n1;
    n2 *= n1--; // n2 = n2 * n1--
    n3 = --n1 * n2--;
    n4 = n1++ + n1++;
    n3 = (n1++) + (n1++);
    n3 = n4 = n1 - 1;
}
```

int n1	5	1000
int n2	8	1004
int n3	4	1008
int n4	4	1012

האופרטור **++** מימין תמיד פועל בסוף,  
גם אם הוא בסוגרים !

## המרה בין טיפוסים (casting)

- יתכן ונרצה ליחס ביטוי המכיל משתנים מטיפוסים שונים, ולבן צריך לדעת איך להתייחס לפעולות על טיפוסים שונים.

```
static void TypesConversion1()
{
    int x = 5;
    double d = 2.5;
    Console.WriteLine(x + d);
}
```

## המרה בין טיפוסים (casting)

- יתכן ונרצה לחשב ביטוי המכיל משתנים מטיפוסים שונים, וכן נדרש לדעת איך להתייחס לפעולות על טיפוסים שונים.

```
static void TypesConversion1()
{
    int x = 5;
    double d = 2.5;
    Console.WriteLine(x + d); 7.5
}
```

# המרה בין טיפוסים (casting)

- יתכן ונרצה ליחס ביטוי המכיל משתנים מטיפוסים שונים, וכן צריך לדעת איך להתייחס לפעולות על טיפוסים שונים.

```
static void TypesConversion1()
{
    int x = 5;
    double d = 2.5;
    Console.WriteLine(x + d); 7.5
}
```

- קיימת היררכיה של המרות אוטומטיות כך שמשתנים מטיפוסים בעלי עדיפות נמוכה יותר בביטוי מומרים לטיפוס בעל העדיפות הגבוהה ביותר:

❖ **double**  
❖ **int**  
❖ **char**

- היררכיה היא לפי גודל הטיפוס, כך שלא נאבד מידע.
- במידה ורוצים לבצע המרה מטיפוס בעל עדיפות גבוהה לנמוכה יותר, יש לבצע המרה מפורשת.

# המרה בין טיפוסים (casting) : דוגמה 1

```
static void TypesConversion2()
{
    double res, d1 = 5.2;
    int n1 = 4;
    res = n1 + d1;
}
```

- ניתן לכתוב ביטוי המכיל טיפוסים שונים, למשל:

בדוגמה זו מתבצעת המרה של המשתנה `n1` מ-`int` ל-`double`, ורק אז מבוצעות פעולות החיבור וההשמה.

- נשים לב כי המשתנה `n1` בזיכרון אינו הופך ל-`double`, שכן ההמרה נעשית באופן זמני בלבד, אך ורק לצורך חישוב הביטוי.

## חוק המרות:

- \* נחשב את הביטוי בצד ימין ע"י המרת כל המשתנים לטיפוס בעל העדיפות הגבוהה ביותר,
- \* ולבסוף נבצע המרה של התוצאה לטיפוס שמשמאלי.

## המרה בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7;

    d0 = d1 + n1 / n2;

}
```

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0

    d0 = d1 + n1 / n2;

}
```

- ראשית מחושב הביטוי  $d0 = d1 + n1 / n2$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - מאחר ושני מרכיביו הם מטיפוס **int** (מספר שלם) לא מתבצעת המרת.
  - תוצאה ביטוי זה היא 0.

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
    0
    d0 = d1 + n1 / n2;
}
```

- ראשית מחושב הביטוי  $d0 = d1 + n1 / n2$  כי לפעולות החילוק יש עדיפות על פני פעולה החיבור.
  - מאחר ושני מרכיביו הם מטיפוס `int` (מספר שלם) לא מתבצעת המרתם.
  - תוצאה ביטוי זה היא 0.

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
    d0 = d1 + n1 / n2;
}
```

- ראשית מחושב הביטוי  $d0 = d1 + n1 / n2$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - אחר ושני מרכיביו הם מטיפוס `int` (מספרשלם) לא מתבצעת המרתם.
  - תוצאה ביטוי זה היא 0.
- הביטוי המחושב כתת הוא חיבור בין `double` ל-`int`.

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
                        0.0
    d0 = d1 + n1 / n2;
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפועלות החילוק יש עדיפות על פני פעולה החיבור.
  - אחר ושני מרכיביו הם מטיפוס **int** (מספר שלם) לא מתבצעת המרתם.
  - תוצאה ביטוי זה היא 0.
- הביטוי המחושב כעת הוא חיבור בין **double** ל-**int**.
  - לכן ה-**int** מומר ל-**double** (0.0).

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
    0.0
    d0 = d1 + n1 / n2;
    Console.WriteLine(d0);
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפועלות החילוק יש עדיפות על פני פעולה החיבור.
  - אחר ושני מרכיביו הם מטיפוס `int` (מספר שלם) לא מתבצעת המרת.
  - תוצאת ביטוי זה היא 0.
- הביטוי המחשב כתע הוא חיבור בין `double` ל-`int`.
  - לכן ה-`int` מומר ל-`double` (0.0).
  - והתוצאה הנכנסת למשתנה `d0` היא 7.0.

## המראת בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
    0.0
    d0 = d1 + n1 / n2;
    Console.WriteLine(d0); 7
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפעולות החילוק יש עדיפות על פני פעולה החיבור.
  - מאחר ושני מרכיביו הם מטיפוס `int` (מספר שלם) לא מתבצעת המרת.
  - תוצאה ביטוי זה היא 0.
- הביטוי המחושב כעת הוא חיבור בין `double` ל-`int`.
  - لكن ה-`int` מומר ל-`double` (0.0).
  - והתוצאה הנכנסת למשתנה `d0` היא 7.0.

## המרות בין טיפוסים : דוגמה 2

```
static void TypesConversion2()
{
    int n1 = 5, n2 = 8;
    double d0, d1 = 7; // d1 is actually 7.0
                        0.0
    d0 = d1 + n1 / n2;
    Console.WriteLine(d0); 7
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפועלות החילוק יש עדיפות על פני פעולת החיבור.
  - אחר ושני מרכיביו הם מטיפוס **int** (מספר שלם) לא מתבצעת המרתה.
  - תוצאה ביטוי זה היא 0.
- הביטוי המחושב כעת הוא חיבור בין **double** ל-**int**.
  - לכן ה-**int** מומר ל-**double** (0.0).
  - והתוצאה הנכונת למשתנה **d0** היא 7.0.
- בהדפסה יודפס 7 (ולא 7.0) כי כך מוצג מספר עשרוני שחלק העשורי שלו הוא 0 (כלומר שערךו מספר שלם).

## המרות בין טיפוסים : דוגמה 3

```
static void TypesConversion3()
{
    int n1 = 5;
    double d0, d1 = 7, d2 = 8.0;

    d0 = d1 + n1 / d2;
}
```

- ראשית מחושב הביטוי  $d0 = d1 + n1 / d2$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - הפעם 1<sup>st</sup> מומר ל-**double**.
  - פעולה החילוק בין שני ה-**double** מחזירה 0.625.

## המרות בין טיפוסים : דוגמה 3

```
static void TypesConversion3()
{
    int n1 = 5;
    double d0, d1 = 7, d2 = 8.0;
    0.625
    d0 = d1 + n1 / d2;
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - הפעם 1 מומר ל-double.
  - פעולת החילוק בין שני ה-double מוציאה 0.625.

## המרות בין טיפוסים : דוגמה 3

```
static void TypesConversion3()
{
    int n1 = 5;
    double d0, d1 = 7, d2 = 8.0;
    0.625
    d0 = d1 + n1 / d2;
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - הפעם 1 הוא מומר ל-`double`.
  - פעולה החילוק בין שני ה-`double` מוחזירה 0.625.
- הביטוי המוחושב כעת הוא חיבור בין `double` ל-`double` בין `double` וכן אין המרות נוספות.

## המרות בין טיפוסים : דוגמה 3

```
static void TypesConversion3()
{
    int n1 = 5;
    double d0, d1 = 7, d2 = 8.0;
    0.625
    d0 = d1 + n1 / d2;
}
```

- ראשית מחושב הביטוי  $d1/n1$  כי לפעולת החילוק יש עדיפות על פני פעולה החיבור.
  - הפעם 1<sup>st</sup> מומר ל-**double**.
  - פעולה החילוק בין שני ה-**double** ממחזירה 0.625.
- הביטוי המוחושב כעת הוא חיבור בין **double** ל-**double** בין **double** ולכן אין המרות נוספות.
  - התוצאה היא 7.625.

## המרה בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5;
}
```

## המרה בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
}
```

## המרות בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority

    n = (int)(d1 / d2 + 5);
}
```

## הבדון בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
```

```
n = (int)(d1 / d2 + 5);
```

- ראשית מחושב הביטוי  $d1/d2 + 5$  שתוצאתו היא 4.5 .

## המרה בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
}
```

$\frac{4.5}{d1 / d2} + 5$   
n = (int)(d1 / d2 + 5);

- ראשית מחושב הביטוי  $d1 / d2$  שנותצאותו היא 4.5.
- הביטוי המוחשב כעת הוא חיבור בין **int** ל-**double**:
  - ראשית ה-**int** מומר ל-**double**.

## המרהות בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
}
```

$n = (\text{int})(\underline{\underline{d1 / d2}} \underline{\underline{+ 5}});$

- ראשית מחושב הביטוי  $d1 / d2$  שתווצאתו היא 4.5 .
- הביטוי המחשב כעת הוא חיבור בין  $\text{int}$  ל- $\text{double}$  .
  - ראשית ה- $\text{int}$  מומר ל- $\text{double}$ .

## המרה בין טיפוסים : דוגמה 4

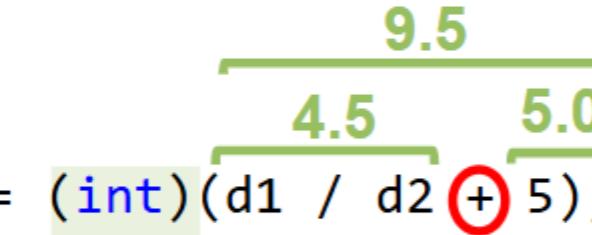
```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
    n = (int)(d1 / d2 + 5);
}
```

$$\begin{array}{r} 9.5 \\ \hline 4.5 & 5.0 \\ \hline \end{array}$$

- ראשית מחושב הביטוי  $d1/d2$  שנותצאותו היא 4.5 .
- הביטוי המחשב כתעט הוא חיבור בין **int** ל-**double** .
  - ראשית ה-**int** מומר ל-**double** .
  - תוצאה של הפעולה החיבור היא 9.5 .

## המרות בין טיפוסים : דוגמה 4

```
static void TypesConversions4()
{
    int n;
    double d1 = 9, d2 = 2.0;
    n = d1 / d2 + 5; // doesn't compile since R-Value is with bigger priority
    n = (int)(d1 / d2 + 5);
}
```



- ראשית מחושב הביטוי  $d1/d2 + 5$  שתווצאתו היא 4.5.
- הביטוי המחושב כתע הוא חיבור בין **int** ל-**double**:
  - ראשית ה-**int** מומר ל-**double**.
  - תוצאת פעולה החיבור היא 9.5.
- ערך הביטוי הוא **double** ולכן המרת מפורשת ל-**int** והtoutput תהיה 9.

## המרה בין טיפוסים : דוגמה 5

```
static void TypesConversions5()
{
    int x;
    char ch = 'a';

    x = ch + 3;
}
```

## המרה בין טיפוסים : דוגמה 5

```
static void TypesConversions5()
{
    int x;
    char ch = 'a';
    97
    x = ch + 3;
}
```

- מתריצה המרה מ-**char** ל-**int** لكن תוצאה הביטוי היא ערך ה-ASCII של התו 'a'.  
 $100 \leftarrow 3 + (97)$

## המרה בין טיפוסים : דוגמה 5

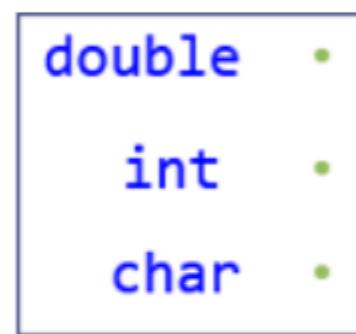
```
static void TypesConversions5()
{
    int x;
    char ch = 'a';
    97
    x = ch + 3;
}
```

- ממבצע המרה מ-**char** ל-**int** لكن תוצאה הביטוי היא ערך ה-ASCII של התו '.'  
 $100 \leftarrow 3 + (97)$

Decimal	Char
96	'`
97	'a'
98	'b'
99	'c'
100	'd'
101	'e'
102	'f'
103	'g'
104	'h'
105	'i'
106	'j'
107	'k'
108	'l'
109	'm'
110	'n'
111	'o'
112	'p'
113	'q'
114	'r'
115	's'
116	't'
117	'u'
118	'v'
119	'w'
120	'x'
121	'y'
122	'z'
123	'{'
124	' '
125	'}'
126	'"
127	

## המרה מכוונת

המרה מכוונת ממירה מטיפוס בעל עדיפות גדולה לטיפוס בעל עדיפות נמוכה כדי שלא נקבל שגיאת קומpileציה בגין איבוד מידע.



- התחבריר:  
**<type><expression>**
- דוגמה:  
**5 ← (int)(2+3.2)**
- לאופרטור **casting** יש את הקידימות הגבוהה ביותר.  
בביטוי **y/x(double)**:
  - מתבצעת המרה של **x** ל-**double**
  - ורק אז מתבצעת פעולה החילוק

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;

    d1 = n1 / n2;

}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
        0
    d1 = n1 / n2;           // d1 = 0.0
}
}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0

    d1 = (double)n1 / n2;

}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0
    5.0
    d1 = (double)n1 / n2;
}

}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0

    5.0   6.0
    d1 = (double)n1 / n2;  // d1 = 0.833333
}

}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0

    5.0   6.0
    d1 = (double)n1 / n2; // d1 = 0.833333

    d1 = (int)(n2 / 4.23);
}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0
    5.0   6.0
    d1 = (double)n1 / n2;  // d1 = 0.833333
    6.0
    d1 = (int)(n2 / 4.23);
}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0
    5.0   6.0
    d1 = (double)n1 / n2;  // d1 = 0.833333
    1.x
    6.0
    d1 = (int)(n2 / 4.23);
}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0

    5.0   6.0
    d1 = (double)n1 / n2;  // d1 = 0.833333
    1
    1.x
    6.0
    d1 = (int)(n2 / 4.23);
}
```

## המרה מכוונת : דוגמה 1

```
static void ForcedCasting1()
{
    int n1 = 5, n2 = 6;
    double d1;
    0
    d1 = n1 / n2;           // d1 = 0.0

    5.0   6.0
    d1 = (double)n1 / n2;  // d1 = 0.833333
    1
    1.x
    6.0
    d1 = (int)(n2 / 4.23); // d1 = 1.0
}
```

המרה מכוונת ל-**int**  
תעלג את המספר לפני מטה  
כלומר, **תקצוץ** את המספר

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;

    n1 = ch++;

}
```

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
}


```

## הمرة מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;

}
```

## הمرة מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;           // n1=99
}

}
```

## הمرة מכונה : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;           // n1=99

    ch = (char)(n1 + 2);

}
```

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch='b'
    n1 = ch + 1;           // n1=99
    101
    ch = (char)(n1 + 2); // ch='e'
}


```

## הمرة מכונה 2 : דוגמה

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;           // n1=99
    101
    ch = (char)(n1 + 2); // ch=e'

    99   97
    n1 = (int)((n1 - 'a' + 3) * d1);
}


```

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;           // n1=99
    101
    ch = (char)(n1 + 2); // ch=e'
```

$$\frac{5}{\underline{\underline{99 \quad 97}}}$$

```
n1 = (int)((n1 - 'a' + 3) * d1);
```

```
}
```

## הمرة מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;         // n1=99
    101
    ch = (char)(n1 + 2); // ch='e'
```

$$\begin{array}{c} 12.5 \\ \hline 5 \rightarrow 5.0 \\ \hline 99 \quad 97 \end{array}$$

```
n1 = (int)((n1 - 'a' + 3) * d1);
```

```
}
```

## הمرة מכונה 2 : דוגמה

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch=b'
    n1 = ch + 1;           // n1=99

    101
    ch = (char)(n1 + 2); // ch='e'

    12
    12.5
    5 → 5.0
    99 97

    n1 = (int((n1 - 'a' + 3) * d1));
}


```

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++;           // n1=98, ch='b'
    n1 = ch + 1;           // n1=99

    101
    ch = (char)(n1 + 2);   // ch='e'

    12
    12.5
    5 → 5.0
    99 97

    n1 = (int)((n1 - 'a' + 3) * d1); // n1=12
}


```

## הمرة מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    97
    n1 = ch++; // n1=98, ch=b'
    n1 = ch + 1; // n1=99

    101
    ch = (char)(n1 + 2); // ch=e'

    12
    12.5
    5 → 5.0
    99 97
    n1 = (int)((n1 - 'a' + 3) * d1); // n1=12

    n2 *= (int)d1; // n2 = n2 * (int)d1
}
```

## המרה מכוונת : דוגמה 2

```
static void ForcedCasting2()
{
    char ch = 'a';
    int n1, n2 = 7;
    double d1 = 2.5;
    n1 = 97                         // n1=98, ch=b'
    n1 = ch + 1;                      // n1=99

    ch = (char)(n1 + 2);            // ch=e'

    12
    12.5
    5 → 5.0
    99 97

    n1 = (int((n1 - 'a' + 3) * d1)); // n1=12

    n2 *= (intd1);      // n2 = n2 * (int)d1 → n2 = 7*2 = 14
}
```

## אופרטורי יחס ושוון

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ו吐צאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש
קטן מ-	<	$3 < 5$ ✓ $5 < 3$ ✗
	=<	
קטן או שווה ל-	=<	$4 <= 4$ ✓ $3 <= 5$ ✓
גדול מ-	>	$3 > 5$ ✗ $5 > 3$ ✓
	=>	
גדול או שווה ל-	=>	$4 >= 4$ ✓ $3 >= 5$ ✗
שווה ל-	==	$4 == 4$ ✓ $4 == 5$ ✗
שונה מ- (לא שווה ל-)	!=	$4 != 4$ ✗ $4 != 5$ ✓

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;
}

```

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאהו היא: `false` או `true`



שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	קטן מ-
	=<	4 <= 4 ✓ 3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓	גדול או שווה ל-
	=>	4 >= 4 ✓ 3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗	שונה מ- (לא שווה ל-)
	!=	4 != 4 ✗ 4 != 5 ✓	

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;
}

```

int: x	3	1000
int: y	5	1004
bool: res		1008



## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאהו היא: `false` או `true`

שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	
קטן או שווה ל-	<=	4 <= 4 ✓ 3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓	
גדול או שווה ל-	=>	4 >= 4 ✓ 3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗	
שונה מ-	!=	4 != 4 ✗ 4 != 5 ✓	(לא שווה ל-)

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
}

```



int: x	<b>3</b>	1000
int: y	<b>5</b>	1004
bool: res		1008

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאהו היא: `false` או `true`

שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	
קטן או שווה ל-	=<	4 <= 4 ✓ 3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓	
גדול או שווה ל-	=>	4 >= 4 ✓ 3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗	
שונה מ-	!=	4 != 4 ✗ 4 != 5 ✓	(לא שווה ל-)

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
}

```



int: x	3	1000
int: y	5	1004
bool: res	true	1008

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ו吐וצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש	שם האופרטור
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	קטן או שווה ל-
גדול מ-	=	4 <= 4 ✓ 3 <= 5 ✓	
גדול או שווה ל-	>	3 > 5 ✗ 5 > 3 ✓	שווה ל-
שווה ל-	=	4 >= 4 ✓ 3 >= 5 ✗	
שונה מ-	!=	4 == 4 ✓ 4 == 5 ✗	(לא שווה ל-)
(לא שווה ל-)	!=	4 != 4 ✗ 4 != 5 ✓	

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);
}

```

int: x	3	1000
int: y	5	1004
bool: res	true	1008



## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: false או true

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
קטן או שווה ל-	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
גדול או שווה ל-	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
שונה מ- (לא שווה ל-)	!=	4 != 4 ✗ 4 != 5 ✓

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);
}

```

int: x	3	1000
int: y	5	1004
bool: res	true	1008

True

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאהו היא: `false` או `true`

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓
		5 < 3 ✗
קטן או שווה ל-	<=	4 <= 4 ✓
		3 <= 5 ✓
גדול מ-	>	3 > 5 ✗
		5 > 3 ✓
גדול או שווה ל-	>=	4 >= 4 ✓
		3 >= 5 ✗
שווה ל-	==	4 == 4 ✓
		4 == 5 ✗
שונה מ- <b>(לא שווה ל-)</b>	!=	4 != 4 ✗
		4 != 5 ✓

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);
}

```

int: x	3	1000
int: y	5	1004
bool: res	true	1008

True

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאהו היא: false או true

שם האופרטור	סימן	שימוש	קטן מ-
$3 < 5$	✓	<	
$5 < 3$	✗		
קטן או שווה ל-	=		
$4 <= 4$	✓		
$3 <= 5$	✓		
גדול מ-	>		
$3 > 5$	✗	>	
$5 > 3$	✓		
גדול או שווה ל-	=>		
$4 >= 4$	✓		
$3 >= 5$	✗		
שווה ל-	==		
$4 == 4$	✓		
$4 == 5$	✗		
שונה מ-	!=		
$4 != 4$	✗		
$4 != 5$	✓		
(לא שווה ל-)			

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);
}

```

int: x	<b>3</b>	1000
int: y	<b>5</b>	1004
bool: res	<b>true</b>	1008

True  
True

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ו吐וצאתו היא: false או true

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ- <b>(לא שווה ל-)</b>		

## אופרטורי יחס ושוין

```
static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
}
```

int: x	3	1000
int: y	5	1004
bool: res	true	1008

True  
True

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאה היא: false או true

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ-		(לא שווה ל-)

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
}

```

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ו吐וצאתו היא: `false` או `true`

<code>int: x</code>	<b>3</b>	1000
<code>int: y</code>	<b>5</b>	1004
<code>bool: res</code>	<b>false</b>	1008

True  
True

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	<=	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
(לא שווה ל-)		

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
}

```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

```

True
True
False

```

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים  
ותוצאה היא: false או true

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
<b>(לא שווה ל-)</b>		שונה מ-

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);
}

```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

True  
True  
False

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: false או true

שם האופרטור	סימן	שימוש	הו מ-
<	3 < 5	✓	קטן מ-
<=	5 < 3	✗	קטן או שווה ל-
>	4 <= 4	✓	גדול מ-
>=	3 <= 5	✓	גדול או שווה ל-
==	3 > 5	✗	שווה ל-
!=	5 > 3	✓	שונה מ-
	4 == 4	✓	(לא שווה ל-)
	4 == 5	✗	
	4 != 4	✗	
	4 != 5	✓	

## אופרטורי יחס ושוין

```
static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);
}
```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

```
True  
True  
False  
False
```

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ-		
(לא שווה ל-)		

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);

}

```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

True  
True  
False  
False

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	
קטן או שווה ל-	<=	4 <= 4 ✓ 3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓	
גדול או שווה ל-	=>	4 >= 4 ✓ 3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗	
שונה מ- (לא שווה ל-)	!=	4 != 4 ✗ 4 != 5 ✓	

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);
}

```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

True  
True  
False  
False  
False

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓	קטן מ-
	✗	5 < 3 ✗	
קטן או שווה ל-	≤	4 <= 4 ✓	קטן או שווה ל-
	✓	3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗	גדול מ-
	✓	5 > 3 ✓	
גדול או שווה ל-	≥	4 >= 4 ✓	גדול או שווה ל-
	✗	3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓	שווה ל-
	✗	4 == 5 ✗	
שונה מ- (לא שווה ל-)	!=	4 != 4 ✗	שונה מ- (לא שווה ל-)
	✓	4 != 5 ✓	

## אופרטורי יחס ושוין

```
static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);

    res = x != y;
}
```

int: x	3	1000
int: y	5	1004
bool: res	false	1008

True  
True  
False  
False  
False

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	<=	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ-		
(לא שווה ל-)		

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);

    res = x != y;
}

```

int: x	<b>3</b>	1000
int: y	<b>5</b>	1004
bool: res	<b>true</b>	1008

True  
True  
False  
False  
False

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: **false** או **true**

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=<	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=>	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ- <b>(לא שווה ל-)</b>		

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);

    res = x != y;
    Console.WriteLine (res);
}

```

int: x	<b>3</b>	1000
int: y	<b>5</b>	1004
bool: res	<b>true</b>	1008

```

True
True
False
False
False

```

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗
	=	4 <= 4 ✓ 3 <= 5 ✓
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓
	=	4 >= 4 ✓ 3 >= 5 ✗
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗
	!=	4 != 4 ✗ 4 != 5 ✓
שונה מ-		(לא שווה ל-)

```

static void BooleanOperators()
{
    int x = 3, y = 5;
    bool res;

    res = x < y;
    Console.WriteLine (res);

    res = x <= y;
    Console.WriteLine (res);

    Console.WriteLine (x > y);
    Console.WriteLine (x >= y);

    res = x == y;
    Console.WriteLine (res);

    res = x != y;
    Console.WriteLine (res);
}

```

int: x	3	1000
int: y	5	1004
bool: res	true	1008

```

True
True
False
False
False
True

```

## אופרטורי יחס ושוין

- ביטוי לוגי הינו דרך לבטא יחס בין 2 ביטויים ותוצאתו היא: `false` או `true`

שם האופרטור	סימן	שימוש	
קטן מ-	<	3 < 5 ✓ 5 < 3 ✗	
	<=	4 <= 4 ✓ 3 <= 5 ✓	
גדול מ-	>	3 > 5 ✗ 5 > 3 ✓	
	>=	4 >= 4 ✓ 3 >= 5 ✗	
שווה ל-	==	4 == 4 ✓ 4 == 5 ✗	
	!=	4 != 4 ✗ 4 != 5 ✓	
(לא שווה ל-)			

## אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :  
יחזיר **true** אם שני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי יורכב מתוצאות של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :  
يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
}
```



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי יורכב מתוצאתם של כמה ביטויים

- **האופרטור & נקרא and ומשמעותו "וגם"** :  
יחזיר **true** אם שני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
}
```

True



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי יורכב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :  
יחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
    res = x > y && y < z;
    Console.WriteLine(res);
}
```

True



## אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :  
יחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
    res = x > y && y < z;
    Console.WriteLine(res);
}
```

True  
False



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם":  
יחזיר **true** אם שני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**
- האופרטור **||** נקרא **or** ומשמעותו "או":  
יחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
    res = x > y && y < z;
    Console.WriteLine(res);
}
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True  
False



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :  
ឱץיך **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**
- האופרטור **||** נקרא **or** ומשמעותו "או" :  
ឱץיך **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
{
    int x = 4, y = 7, z = 9;
    bool res;

    res = x < y && y < z;
    Console.WriteLine(res);
    res = x > y && y < z;
    Console.WriteLine(res);
}
```

אם התוצאה ברורה ווגורה  
הביטוי השני לא יבדק !

True  
False



# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True  
False



האם הביצים משוקולד  
ואו הביצה משוקולד

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתואמתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True  
False



האם הביצים משוקולד ✓  
או הביצה משוקולד

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי יורכב מביטויים של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);
```

אם התוצאה ברורה ויגורה  
הביטוי השני לא יבדק !

True  
False  
True



האם הביצים משוקולד ✓  
או הביצה משוקולד

```
}
```

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);  
    Console.WriteLine(z < y || x < y);
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True  
False  
True



אם הביצים משוקולד  
או הביצה משוקולד

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);  
    Console.WriteLine(z < y || x < y);
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True
False
True
True



האם הביצים משוקולד  
ואו הביצה משוקולד

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

יֵחַזֵּר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);  
    Console.WriteLine(z < y || x < y);  
    Console.WriteLine(y < x || z < y);
```

True
False
True
True
False



האם הביצים משוקולד ✓  
או הביצה משוקולד

# אופרטורים לוגיים

לפעמים נרצה שערך של ביטוי ירכיב מתוצאתם של כמה ביטויים

- האופרטור **&&** נקרא **and** ומשמעותו "וגם" :

يُחזיר **true** אם שני הביטויים המרכיבים אותו החזירו **true**, אחרת יחזיר **false**

- האופרטור **||** נקרא **or** ומשמעותו "או" :

يُחזיר **true** אם לפחות אחד משני הביטויים המרכיבים אותו החזיר **true**, אחרת יחזיר **false**

```
static void LogicalOperators()
```

```
{
```

```
    int x = 4, y = 7, z = 9;  
    bool res;
```

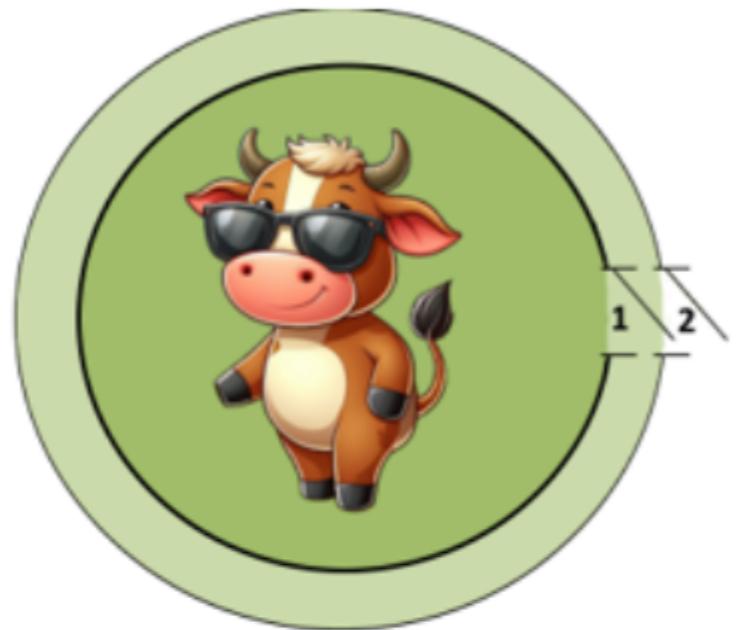
```
    res = x < y && y < z;  
    Console.WriteLine(res);  
    res = x > y && y < z;  
    Console.WriteLine(res);  
    Console.WriteLine(x < y || y < z);  
    Console.WriteLine(z < y || x < y);  
    Console.WriteLine(y < x || z < y);
```

אם התוצאה ברורה וסgorה  
הביטוי השני לא יבדק !

True
False
True
True
False

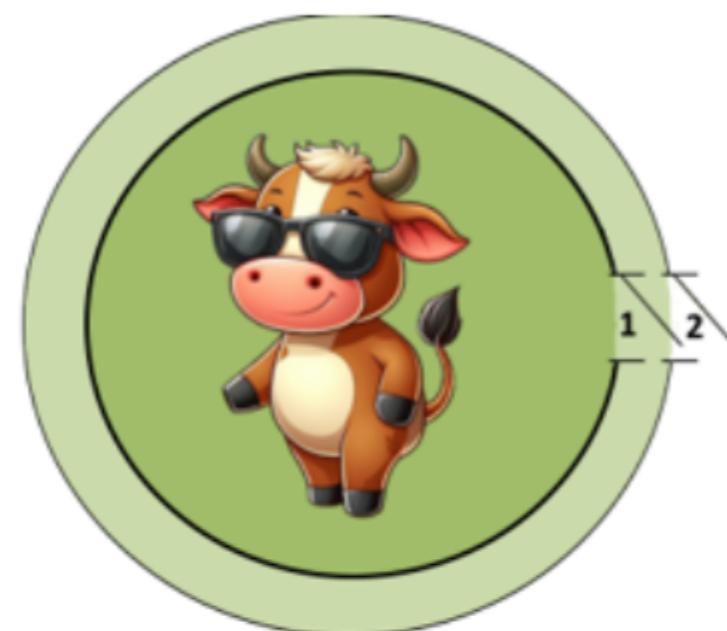
## אופרטורים לוגיים

- כדי שהשור יוכל לצאת הוא צריך לעבור דרך שער 1 **ווגם** (&) דרך שער 2 אם שער 1 סגור, השור לא יצא, ואין צורך לבדוק את תנאי 2!  
**לכן - תנאי 2 כלל לא יבדק.**



## אופרטורים לוגיים

- כדי שהשור יוכל לצאת הוא צריך לעבור דרך שער 1 **ווגם** (&) דרך שער 2 אם שער 1 סגור, השור לא יצא, ואין צורך לבדוק את תנאי 2!  
לכן - **תנאי 2 כלל לא יבדק.**
- כדי שהשור יוכל לצאת הוא צריך לעבור דרך שער 1 **או** (||) דרך שער 2 אם שער 1 פתוח, השור יצא, ואין צורך לבדוק את תנאי 2!  
לכן - **תנאי 2 כלל לא יבדק.**



# אופרטורים לוגיים אופרטור השילילה !

- **האופרטור ! נקרא not ומשמעותו שלילה :**

- יחזיר `true` אם ערך הביטוי המקורי `false`
- יחזיר `false` אם ערך הביטוי המקורי `true`

- **נשתמש באופרטור ! כאשר נרצה לקבל את שלילתו של ביטוי מסוים.**



אהובים מלפפון חמוץ

# אופרטורים לוגיים אופרטור השיליה !

- **האופרטור ! נקרא not ומשמעותו שלילה :**
  - יחזיר `true` אם ערך הביטוי המקורי `false`
  - יחזיר `false` אם ערך הביטוי המקורי `true`
- **נשתמש באופרטור ! כאשר נרצה לקבל את שלילתו של ביטוי מסוים.**



# אופרטורים לוגיים אופרטור השילילה !

- **האופרטור ! נקרא not ומשמעותו שלילה :**

- יחזיר **true** אם ערך הביטוי המקורי **false**
- יחזיר **false** אם ערך הביטוי המקורי **true**

- **נשתמש באופרטור !** כאשר נרצה לקבל את שלילתו של ביטוי מסוים.

```
static void OperatorNot()
{
    int x = 4, y = 5;
    Console.WriteLine(x > y); 
}
```

# אופרטורים לוגיים אופרטור השילילה !

- **האופרטור ! נקרא not ומשמעותו שלילה :**

- יחזיר `true` אם ערך הביטוי המקורי `false`
- יחזיר `false` אם ערך הביטוי המקורי `true`

- **נשתמש באופרטור ! כאשר נרצה לקבל את שלילתו של ביטוי מסוים.**

```
static void OperatorNot()
{
    int x = 4, y = 5;

    Console.WriteLine(x > y);      False
}
```

# אופרטורים לוגיים אופרטור השילילה !

- **האופרטור ! נקרא not ומשמעותו שלילה :**
  - יחזיר `true` אם ערך הביטוי המקורי `false`
  - יחזיר `false` אם ערך הביטוי המקורי `true`
- **נשתמש באופרטור ! כאשר נרצה לקבל את שלילתו של ביטוי מסוים.**

```
static void OperatorNot()
{
    int x = 4, y = 5;

    Console.WriteLine(x > y);      False
    Console.WriteLine(!(x > y));
}
```

# אופרטורים לוגיים אופרטור השיליה !

- האופרטור ! נקרא **not** ומשמעותו שיליה :
  - יחזיר **true** אם ערך הביטוי המקורי **false**
  - יחזיר **false** אם ערך הביטוי המקורי **true**
- השתמש באופרטור ! כאשר נרצה לקבל את שיליתו של ביטוי מסוים.

```
static void OperatorNot()
{
    int x = 4, y = 5;

    Console.WriteLine(x > y);      False
    Console.WriteLine(!(x > y));   True
}
```

## סיכום אופרטורים לוגיים : טבלתאמת

• בביטוי מורכב **&&** ו**כל** הביטויים המרכיבים אותו צריכים להיות **true** כדי שההתוצאה תהיה **true**

בביטוי מורכב **||** לפחות אחד מהביטויים המרכיבים אותו צריך להיות **true** כדי שההתוצאה תהיה **true**

a	b	a&&b	a  b	!a
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

- **אופרטור בינארי:** עובד על 2 ביטויים, יוצר ביטוי לוגי מורכב לדוגמא: **&&** , **||**
- **אופרטור אונארי:** עובד על ביטוי אחד לדוגמא: **!**

# סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
}
```



קידימות האופרטורים			
!			
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
}
```

## סדר קידימות האופרטורים

### קידימות האופרטורים

!

<    <=    >    >=

==    !=

&&

||

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
```

## סדר קידימות האופרטורים

### קידימות האופרטורים

!

< <= > >=

== !=

&&

||

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני כלול לא לבדוק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
}
```

True

קידימות האופרטורים
!
<    <=    >    >=
==    !=
&&

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרת בדיקת התנאי הראשון  
התנאי השני כלל לא נבדק

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
}
```

## סדר קידימות האופרטורים

True

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
**התנאי השני כלל לא נבדק**

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
}
```

## סדר קידימות האופרטורים

True

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
}
```

## סדר קידימות האופרטורים

True  
False

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
}
```

True  
False

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
}
```

True  
False

קידימות האופרטורים
!
<    <=    >    >=
==    !=
&&

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
}
```

True  
False

קידימות האופרטורים			
!			
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
}
```

True  
False  
True

### קידימות האופרטורים

!

< <= > >=

== !=

&&

||

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
}
```

True  
False  
True

קידימות האופרטורים			
!			
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
}
```

True  
False  
True

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או ||):**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני **בלל לא נבדק**

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
}
```

True  
False  
True  
True

### קידימות האופרטורים

!

< <= > >=

== !=

&&

||

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
}
```

True  
False  
True  
True

### קידימות האופרטורים

!

< <= > >=

== !=

&&

||

## סדר קדימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
}
```

True  
False  
True  
True

### קדימות האופרטורים

!

<    <=    >    >=

==    !=

&&

||

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
}
```

True  
False  
True  
True

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
	&&		

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
}
```

True  
False  
True  
True

קידימות האופרטורים			
	!		
<	<=	>	>=
==	!=		
&&			

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
}
```

True  
False  
True  
True  
True

### קידימות האופרטורים

!
<    <=    >    >=
==    !=
&&

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני **בלל לא נבדק**

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
    Console.WriteLine (false == 4 < 1 && 5 > 20);
}
```

True  
False  
True  
True  
True

### קידימות האופרטורים

!
<    <=    >    >=
==    !=
&&

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
הтенאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine ((0 < 4 == true && 5 > 2));
    Console.WriteLine (false == 4 < 1 && 5 > 20);
}
```

True  
False  
True  
True  
True

קידימות האופרטורים	
	!
<	<=
>	>=
==	!=
&&	

**בביטוי מורכב (&& או ||) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine (0 < 4 == true && 5 > 2);
    Console.WriteLine (false == 4 < 1 && 5 > 20);
}
```

True  
False  
True  
True  
True

קידימות האופרטורים
!
<   <=   >   >=
==   !=
&&

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרת בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine ((0 < 4 == true && 5 > 2));
    Console.WriteLine (false == 4 < 1 && 5 > 20);
}
```

True  
False  
True  
True  
True

### קידימות האופרטורים

!
<    <=    >    >=
==    !=
&&

**בביטוי מורכב (&& או || ) :**  
אם התוצאה ברורה אחרי בדיקת התנאי הראשון  
התנאי השני בכלל לא נבדק

## סדר קידימות האופרטורים

```
static void OperatorsPrecending()
{
    Console.WriteLine ((false && true) || true);
    Console.WriteLine (false && (true || true));
    Console.WriteLine (false && true || true);
    Console.WriteLine (false || true && true);
    Console.WriteLine ((0 < 4 == true && 5 > 2));
    Console.WriteLine (false == 4 < 1 && 5 > 20);
}
```

True  
False  
True  
True  
True  
False

### קידימות האופרטורים

!

< <= > >=

== !=

&&

||

# סדר קידמיות האופרטורים

## סדר הפעולות:

- רמה 1 : ()
- רמה 2 : !
- רמה 3 : <, >, <=, >=
- רמה 4 : ==, !=
- רמה 5 : &
- רמה 6 : ||

## כמו במתמטיקה :

- קודם מבצעים את החישוב שנמצא בסוגרים
- אח"כ שלילה
- אח"כ השוואה: (גדול / קטן / שווה / לא שווה)
- אח"כ ביטויים מורכבים: וגם ( && ) ואחריו או ( || )  
תזכורת: בביטויים מורכבים התנאי השני יבדק רק ע"פ הצורך.

במידה שיש כמה פעולות באותה רמת עדיפות החישוב יבוצע משמאל לימין. ↙

## ביחידה זו למדנו:

- פעולות אРИתמטיות
- ביטויים חשבוניים
- אופרטור **++** ואופרטור **--**
- ביטויים מקוצרים
- המרת בין טיפוסים (casting)
- ביטויים לוגיים