



מבוא למדעי המחשב

תירגול 3: לולאות, קלט, וטיפוסים



תוכנייה

- לולאת while
- קלט
- טיפוסים משתנים
- המרת טיפוסים
- טיפוס char





לולאת while



לולאת while

- קטע קוד מתבצע שוב ושוב כל עוד תנאי מתקיים

```
int number = 40;
```

```
while (number > 0)
```

```
{
```

תנאי

```
    printf("%d remain\n", number);
```

```
    number = number - 1;
```

```
}
```

גוף הלולאה

לולאת while

תרגיל 1: הדפיסו כמה פעמים ניתן לחלק מספר ב 2 לפני שהתוצאה קטנה או שווה ל 1

לולאת while

```
int number = 128, counter = 0;
```

```
while (number > 1)
```

תנאי

```
{
```

```
    number /= 2;
```

```
    counter++;
```

גוף הלולאה

```
}
```

```
printf("log2 (%d)=%d\n", number,  
       counter);
```

מה הבעיה?

לולאת while

```
int original_number = 128, counter=0;  
int number = original_number;  
while(number>1)
```

```
{
```

תנאי

```
    number/=2;
```

```
    counter++;
```

```
}
```

גוף הלולאה

```
printf("log2 (%d)=%d\n", original_number,  
      counter);
```



קלט



מבוא למדעי המחשב מ' - תירגול 3

scanf - קלט

- הפונקציה scanf קוראת ערך לתוך משתנה.
- קריאה פשוטה ל-scanf מתבצעת כך:

```
int x;  
scanf("%d", &x);
```

- שורה זאת קוראת ערך לתוך המשתנה x.
- בדומה ל-printf ניתן לקרוא יותר ממשתנה אחד:

```
scanf("%d%d", &a, &b);
```



שימו לב לסימן ה-**&** לפני כל שם משתנה!
סימן זה הוא חובה! את הסיבה לכך נבין בתרגולים מתקדמים



scanf - קלט

- לפקודה `scanf()` יש לציין איזה טיפוס של נתון אנו רוצים לקרוא מהמשתמש: שלם `%d` או שבר `%lf` (שימו לב להבדל מ-`printf`).
- כאשר התוכנית מגיעה בזמן ריצה לפקודת `scanf()`, היא עוצרת ומציגה סמן מהבהב למשתמש.
- בשלב זה התוכנית מאפשרת למשתמש להקליד את הנתון, וכאשר הוא לוחץ על `Enter` הנתון שהוא הקליד מועבר לתוכנית. נתון זה מוכנס למשתנה שצוין, ואז התוכנית ממשיכה בריצתה.

קלט/פלט

- תרגיל 2: כתבו תוכנית הקולטת 2 מספרים מהמשתמש ומדפיסה את סכומם

```
#include <stdio.h>

int main()
{
    printf("Please enter numbers:\n");
    int x=88,y=19;
    scanf("%d%d", &x, &y);
    printf("The sum is: %d",x+y);
    return 0;
}
```

פלט:

Please enter numbers:

קלט:

10 2

פלט:

The sum is: 12

קלט/פלט - הרחבה

- כישלון בקריאת הקלט + בדיקת קלט
- Input buffer
- EOF



כשלון של scanf

```
int main() {  
    int a=-1;  
    scanf("%d", &a);  
    printf("a=%d", a);  
    return 0;  
}
```

- מה תדפיס התוכנית הבאה?

קלט:

ten 10↵

- תשובה: a=-1
- התוכנית לא הצליחה לקרוא את a, כי הכנסנו משהו שהוא לא מספר!

ערך החזרה של `scanf`

- איך נגלה אם קריאת הקלט הצליחה?
- תשובה: ע"י בדיקת ערך ההחזרה של `scanf`.
- ערך ההחזרה של `scanf` הוא מספר שלם המייצג את מספר הדגלים (%) ש-`scanf` הצליחה "לאכול".
- כלומר, מספר המשתנים ש `scanf` הצליחה לקלוט.

ערך החזרה של scanf

- אם הקריאה של דגל כלשהו נכשלה, לא יהיה ניסיון להמשיך ולקרוא את שאר הדגלים.
- דוגמאות לשימוש בערך ההחזרה של scanf:

```
int res = scanf("%d", &x);
```

שמירת הערך

```
if (scanf("%d", &x)==1)
```

השוואת הערך לקבוע 1

scanf – בדיקת ערך החזרה

```
int main() {
    int a=-1,b=-1,c=-1;
    if(scanf("%d", &a)!=1) {
        printf("Error in a\n");
        return 0;
    }
    if(scanf("%d%d", &b, &c)!=2) {
        printf("Error in b or c\n");
        return 0;
    }
    printf("a=%d,b=%d,c=%d",
           a,b,c);
    return 0;
}
```

קלט:

10 twenty 30 ←

פלט

Error in b or c

בדיקת קלט

- כאשר אתם משתמשים ב-`scanf` בתרגילי הבית, עליכם לוודא שהיא הצליחה. ניתן לעשות זאת באמצעות בדיקת ערך ההחזרה שלה:
 - למשל אם היא הייתה אמורה לקלוט 2 מספרים:

```
if (scanf ("%d%d", &d, &n) != 2) {  
    printf ("Error");  
    ...  
}
```

חוצץ הקלט (Input Buffer)

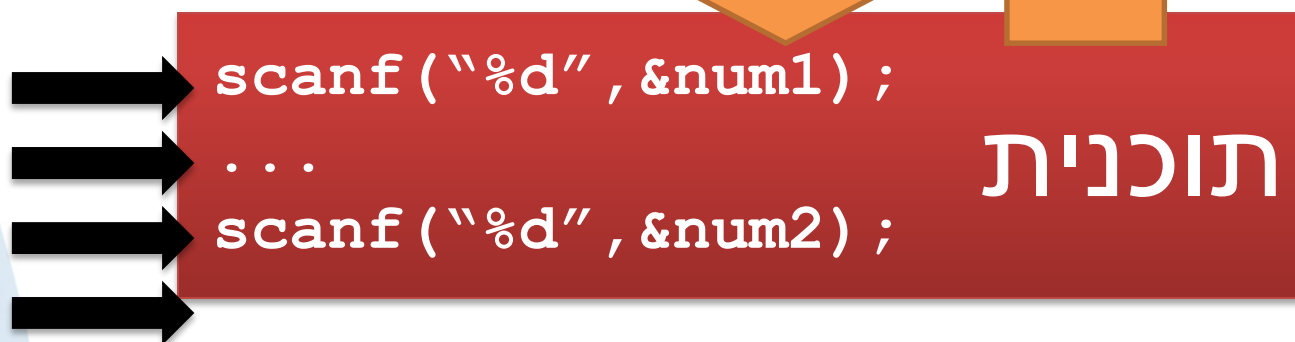
- מטרתנו כעת היא להבין כיצד עובדת קליטת קלט בעזרת `scanf`. לצורך כך, עלינו להכיר מושג חדש: **חוצץ הקלט**.
- חוצץ הקלט הוא אזור זיכרון המתוחזק ע"י מערכת ההפעלה. תפקידו הוא לשמור את קלט המשתמש עד אשר הוא נקרא ע"י התוכנית (או עד שהתוכנית מסתיימת).
- מערכת ההפעלה תפנה למשתמש (תחכה לקלט) רק כאשר החוצץ ריק. **בתחילת התוכנית החוצץ תמיד ריק.**

מה קורה אם נקליד שני מספרים?

10 9 ←

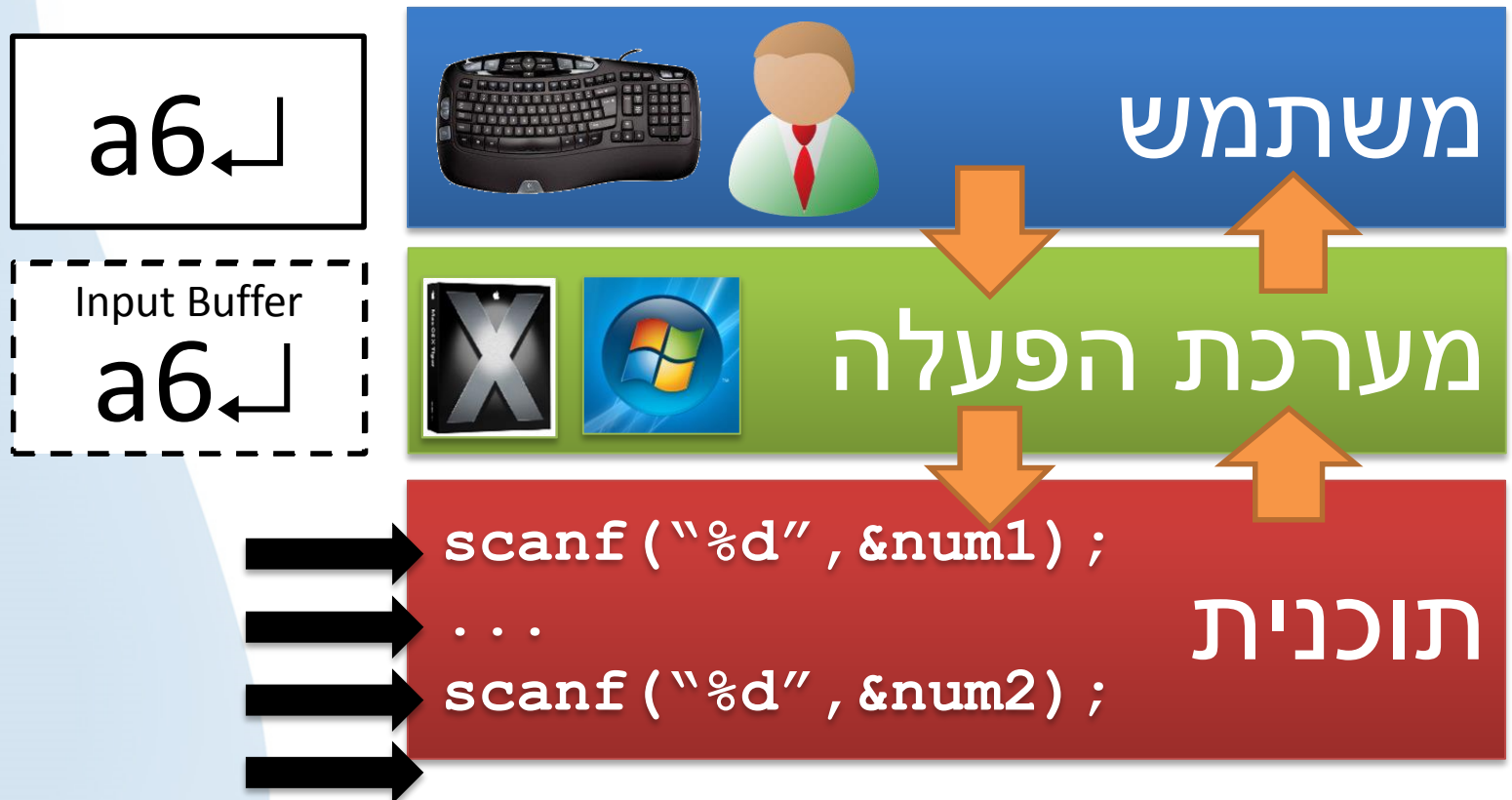


Input Buffer
10 9 ←



קלט שגוי

כאשר הנתון הבא בתור אינו מתאים לטיפוס המבוקש
הקריאה נכשלת (התוכן של המשתנה נשאר כשהיה והנתון נשאר בחוצץ)



התקני הקלט והפלט הסטנדרטיים

- באופן רגיל
 - קלט סטנדרטי (scanf): מגיע מהמקלדת
 - פלט סטנדרטי (printf): מגיע למסך
- עם redirection (כמו שנלמד בתרגול 1)
 - `hw0q1.exe < input.txt > output.txt`
 - קלט סטנדרטי (scanf): מהקובץ `input.txt`
 - פלט סטנדרטי (printf): לקובץ `output.txt`.

התוכנית איננה יודעת בזמן הרצתה
לאן מקושרים הקלט והפלט הסטנדרטיים.
זהו תפקידה של מערכת ההפעלה.

קריאה מקובץ והקבוע EOF

- כאשר קוראים מקובץ, הקלט יכול להגמר (לא יכול לקרות בקלט ממקלדת).
- במקרה כזה, scanf מחזירה את הקבוע EOF (End Of File).

```
if (scanf ("%d%d", &d, &n) == EOF) {  
    printf("No input");  
    ...  
}
```

- כאשר הקלט מגיע מהמקלדת, ניתן להכניס סימן EOF מלאכותי בקלט באמצעות לחיצה על Ctrl+Z במקלדת (Ctrl+D במאכ). **שימו לב!** יש ללחוץ על Ctrl+Z בשורה חדשה במסך על מנת שיקלט EOF.



טיפול משותנים



טיפוסי משתנים

- אם משתנה הוא רצף סיביות בזיכרון, אזי טיפוס המשתנה היא הדרך **לפרש** את הסיביות

- לדוגמא:



עבור unsigned int
מתפרש כ
"4,294,967,295"

עבור int
מתפרש כ "1-"

משתנים וטיפוסי משתנים

- ישנם טיפוסים רבים בשפת C:
 - שלמים: char, short, int, long, long long
 - חיוביים: unsigned char, unsigned short, etc.
 - שברים: float, double, long double
- טיפוסים שונים תופסים כמות זיכרון שונה.

משתנים וטיפוסי משתנים

Type	min	max	precision	Size in memory*
int	-2147483648	2147483647	1	4
unsigned	0	4294967295	1	4
double	$-1.79e10^{308}$	$1.79e10^{308}$	$2.2 \cdot 10^{-308}$ (depend on number)	8
long long	-2^{63}	$2^{63}-1$	1	8
char	-128	127	1	1

הגודל המדויק תלוי בחומרה ובמערכת ההפעלה
הדיוק של שברים תלוי בערכם המדויק (ככול שהערך יותר גדול,
הדיוק קטן)

משתנים וטיפוסים

תרגיל 2: קלטו 2 מספרים שלמים base ו exp , וחשבו base^{exp} (חזקה)

התוכנית צריכה לטפל בתוצאות עד 10^{15}

משתנים וטיפוסים

```
long long pow=1;
int base, exp;
scanf("%d%d", &base, &exp);
while( exp>0 ) {
    pow*=base;
    exp--;
}
printf("pow=%lld\n",pow);
```

משתנים וטיפוסים

```
long long pow=1;
int base, exp;
scanf("%d%d", &base, &exp);
while( exp>0 ) {
    pow*=base;
    exp--;
}
printf("pow=%lld\n",pow);
```

תוצאה כאשר pow מטיפוס long long:

```
10 10
pow=1000000000000
```

תוצאה כאשר pow מטיפוס int:

```
10 10
pow=1410065408
```



המרה - Casting



המרת משתנים אוטומטית

- בשפת C ישנן פעולות רבות המערבות שני איברים:

`a > b` `a += b` `a = b` `a * b` `a + b`

- אם האיברים אינם מאותו הטיפוס, מתבצעת המרה אוטומטית.

- עבור פעולות השמה, הערך בצדו הימני של האופרטור מומר לטיפוסו של המשתנה שלתוכו כותבים.

- לכל יתר האופרטורים, הטיפוס עם תחום הייצוג הקטן יותר מומר לזה עם תחום הייצוג הגדול יותר, לפי הסדר

הבא:

<code>char</code>	→	<code>short</code>	→	<code>int</code>	→	<code>long</code>	→
<code>float</code>	→	<code>double</code>	→	<code>long double</code>			

המרה אוטומטית – סוף הסיפור?

- המרה אוטומטית אינה מתבצעת כשמשתמשים ב- `printf()`.

- במקרה זה יש לבצע המרה מכוונת.

- ומה לגבי התוכנית הבאה?

```
int    num1 = 100000,  
       num2 = 500000;  
  
double product = num1 * num2;
```

מדוע זה עלול
לא לעבוד?

המרה מכוונת (casting)

- המרת ערך כלשהו לטיפוס אחר נעשית על ידי כתיבת שם הטיפוס החדש בסוגריים, לפני הערך עצמו. למשל:

```
int x = 2;  
printf("%lf", (double)x );  
double d = (double)3 / 2;
```


מחזיר את ערכו של
x כטיפוס **double**

- פעולת ההמרה לוקחת את הערך הנתון, ומחזירה **עותק שלו** מהטיפוס החדש (היא אינה משנה את הערך המקורי!).
- בדוגמה למעלה, **x** עצמו איננו משתנה כתוצאה מהפעולה.

דוגמאות ל-Casting


```
int cake_num = 5, children = 3;  
double cake_per_child;
```

כמות העוגה
שיש לכל ילד:



```
cake_per_child =  
(double) cake_num /  
children ;
```

```
cake_per_child =  
cake_num /  
(double) children ;
```



```
cake_per_child =  
(double) (cake_num / children) ;
```

מה תהיה
התוצאה?

טיפוסי משתנים - סיכום

- טיפוסים שונים

- `int`: מספרים שלמים, למשל: -99, 0, 1,435 וכו'.

- `double`: שברים, למשל: -324.3, 0.0, 1.2, 1.0 וכו'.

- ועוד המון: `char`, `bool`, `long long`, `long`, `unsigned`, `float` ועוד

- המרה בין טיפוסים

- מפורשת (`double`), או אוטומטית

- קלט ופלט: `%d`, `%f`, `%lf`



טיפוס char



לפענח char קצת אחרת...

- משתנה מסוג char יכול לשמור אחד מ-256 ערכים שונים (בית אחד).
- ישנה טבלה סטנדרטית (ASCII) להמרה בין 256 ערכים מספריים, לבין 256 תווים שונים.
 - למשל, $A = 65$, $B = 66$ וכו'.
 - כתיבת התו בין גרשיים נותנת את הערך המספרי שלו, לדוגמה: 'A'
- שם הטיפוס char, הוא קיצור ל-character ורומז על שימוש זה.

טבלת ASCII: ממספרים לתווים

רווח הוא מס' 32

(ישנם תווים כגון lf, cr, eof שאינם ניתנים להדפסה ומייצגים תזוזות של הסמן או הגעה לסוף קלט וכד')

000 (nul)	016 ► (dle)	032 sp	048 0	064 @	080 P	096 `	112 p
001 ☺ (soh)	017 ◀ (dc1)	033 !	049 1	065 A	081 Q	097 a	113 q
002 ☹ (stx)	018 ⬆ (dc2)	034 " ' "	050 2	066 B	082 R	098 b	114 r
003 ♥ (etx)	019 ⬇ (dc3)	035 #	051 3	067 C	083 S	099 c	115 s
004 ♦ (eot)	020 ⌘ (dc4)	036 \$	052 4	068 D	084 T	100 d	116 t
005 ♣ (enq)	021 Ⓢ (nak)	037 %	053 5	069 E	085 U	101 e	117 u
006 ♠ (ack)	022 — (syn)	038 &	054 6	070 F	086 V	102 f	118 v
007 • (bel)	023 ⬇ (etb)	039 ' ' ' "	055 7	071 G	087 W	103 g	119 w
008 ▣ (bs)	024 ⬆ (can)	040 (056 8	072 H	088 X	104 h	120 x
009 (tab)	025 ⬇ (em)	041)	057 9	073 I	089 Y	105 i	121 y
010 (lf)	026 (eof)	042 *	058 :	074 J	090 Z	106 j	122 z
011 ♂ (vt)	027 ← (esc)	043 +	059 ;	075 K	091 [107 k	123 {
012 ♣ (np)	028 L (fs)	044 ,	060 <	076 L	092 \	108 l	124
013 (cr)	029 ↔ (gs)	045 -	061 =	077 M	093]	109 m	125 }
014 ♪ (so)	030 ▲ (rs)	046 .	062 >	078 N	094 ^	110 n	126 ~
015 ☼ (si)	031 ▼ (us)	047 /	063 ?	079 O	095 _	111 o	127 ▯

רק חצי מהטבלה מופיע כאן. כדי לראות את הטבלה המלאה חפשו

Extended ASCII Table

תכונות הטבלה

- הערכים המתאימים לתווים '0' ... '9' הם רציפים, ולפי סדר הספרות.
- הערכים המתאימים לתווים 'a' . . 'z' רציפים, ולפי סדר הא"ב הרגיל.
- כך גם הערכים המתאימים לתווים 'A' ... 'Z'.

תווים

תרגיל 3 א': כתבו תכנית המקבלת כקלט תו בודד ומדפיסה אותו ואת ערך ה-ASCII שלו.

```
char c = 0;  
scanf("%c", &c);  
printf("numeric value=%d, character=%c\n", c, c);
```

הרווח כאן חשוב- פירושו שיש להתעלם מרווחים וממעברי שורה המופיעים בתחילת המשפט (התו שיכנס ל-c הוא התו הראשון שאינו רווח)

תווים

תרגיל 3 ב': כתבו תכנית הקולטת רצף תווים עד שהתקבל התו '.', ומדפיסה כל תו ואת ערך ה-ASCII שלו (כולל הנקודה).

```
char c = 0;
scanf(" %c", &c);
while (c != '.')
{
    printf("numeric value=%d, character=%c\n", c,c);
    scanf(" %c", &c);
}
printf("numeric value=%d, character=%c\n", c,c);
```

```
abcd.
numeric value=97, character=a
numeric value=98, character=b
numeric value=99, character=c
numeric value=100, character=d
numeric value=46, character=.
```

הפלט יופיע רק לאחר הקשת ENTER.

תרגיל 4: כתבו תכנית המקבלת כקלט אות קטנה בשפה האנגלית ומדפיסה אותה כאות גדולה.

תרגיל 4: כתבו תכנית המקבלת כקלט אות קטנה בשפה האנגלית ומדפיסה אותה כאות גדולה.

```
char c;  
scanf(" %c", &c);  
c = (c - 'a') + 'A';  
printf("%c", c);
```

תווים

תרגיל 5: כתבו תכנית המקבלת כקלט אות קטנה בשפה האנגלית ומדפיסה את האות העוקבת לה (העוקבת של z היא a).

תווים

תרגיל 5: כתבו תכנית המקבלת כקלט אות קטנה בשפה האנגלית ומדפיסה את האות העוקבת לה (העוקבת של z היא a).

לפני
main

```
#define NUM_LETTERS ('z' - 'a' + 1)
```

קטע קוד
מתוך
הפונקציה
main

```
char c;  
scanf(" %c", &c);  
c = (c - 'a');  
c = (c + 1) % NUM_LETTERS;  
c += 'a';  
printf("%c", c);
```

תרגיל 6: כתבו מחשבון – תכנית המקבלת מספר, פעולת חשבון (חיבור/חיסור/כפל/חילוק) ומספר, מחשבת ומדפיסה את ערך הביטוי.

```
char c = 0; int op1 = 0, op2 = 0, res = 0;
scanf("%d %c %d", &op1, &c, &op2);
switch(c) {
    case '+': res = op1+op2; break;
    case '-': res = op1-op2; break;
    case '*': res = op1*op2; break;
    case '/': res = op1/op2; break;
    default: printf("error: unknown operation");
             return 1;
}
printf("%d%c%d=%d\n", op1, c, op2, res);
```

תווים - סיכום

- הטיפוס char מכיל מספר בין 0 ל 255
- מקביל לתווים ע"י טבלת ASCII
- קבועי תווים ('A')
- הדפסה באמצעות %d, %c

טיפול משותנים - תרגילים

מהו ערכו וטיפוסו של הביטוי?

ביטוי	טיפוס	ערך
5 - 4	int	1
1 / 2	int	0
1 / 2.	double	0.5
(double)1 / 2	double	0.5
(double)(1 / 2)	double	0.0
'A' + 3	int	'D' (= 68 , ONLY if ASCII)
'Z' - 1	int	'Y' (= 89, ONLY if ASCII)
'Z' - 'A' + 'a'	int	'z'
'E' + 4*('b' - 'c')	int	'A'

מהו ערכו וטיפוסו של הביטוי?

[כאשר יש יותר מביטוי אחד, התייחסו לביטוי הצהוב]

הניחו שהמשתנים הבאים הוגדרו:

```
char c;  
int i
```

ביטוי	טיפוס	ערך	תופעות לוואי
<pre>c = 'G'; i = 3; (double)i/(c-'E')</pre>			
<pre>i = 5.7;</pre>			
<pre>i = 5.2;</pre>			
<pre>(double)i</pre>			

מהו ערכו וטיפוסו של הביטוי?

[כאשר יש יותר מביטוי אחד, התייחסו לביטוי הצהוב]

הניחו שהמשתנים הבאים הוגדרו:

```
char c;  
int i
```

ביטוי	טיפוס	ערך	תופעות לוואי
<pre>c = 'G'; i = 3; (double)i/(c-'E')</pre>	double	1.5	<pre>c ← 'G' i ← 3</pre>
<pre>i = 5.7;</pre>	int	5	<pre>i ← 5</pre>
<pre>i = 5.2;</pre>	int	5	<pre>i ← 5</pre>
<pre>(double)i</pre>	double	i	אין ! בפרט, i לא השתנה!



אופרטורים



אופרטורים

- אופרטור הוא פעולה של C, המקבלת ערך יחיד או זוג ערכים, ומחזירה ערך כלשהו.
- אופרטור אונארי (unary operator) מקבל ערך יחיד.
- אופרטור בינארי (binary operator) מקבל זוג ערכים.
- ישנם מספר אופרטורים המשנים את הערכים שהם מקבלים. השפעה זו נקראת תוצאת לוואי (side-effect) של האופרטור.

אופרטורים בינאריים

- פעולות חשבון:

$a+b$	$a-b$	$a*b$	a/b	$a\%b$
-------	-------	-------	-------	--------

- השוואות:

$a==b$	$a!=b$	$a<b$	$a>b$	$a<=b$	$a>=b$
--------	--------	-------	-------	--------	--------

$a \ \&\& \ b$	$a \ \ b$
----------------	--------------

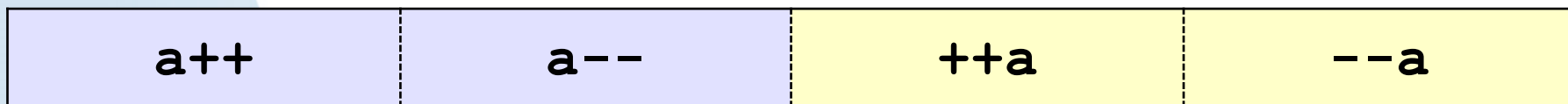
- פעולות לוגיות:

$a=b$	$a+=b$	$a-=b$	$a*=b$	$a/=b$
-------	--------	--------	--------	--------

- השמות:

אופרטורים אונאריים

- מינוס מתמטי: $-a$
- casting: `(double) num`
- אופרטורי קידום:



דוגמאות

```
int x = 5, y = 0;
```

x = 5 , y = 0

```
y = -x + x ;
```

x = 6 , y = 13

```
y = ++x + 7 ;
```

```
y = x++ + 3;
```

x = 7 , y = 9

```
y = ++x + x++ ;
```

**מה לדעתכם
יקרה פה?**

דוגמאות

```
int x=0, y=5;
while (x++<y) {
    printf("%d ", x);
}
```

מה יודפס?

```
int x=0, y=5;
while (++x<y) {
    printf("%d ", x);
}
```

ומה עכשיו?