פרק 4
קלט / פלט

פונקציות ב-C

- פונקציה היא קטע קוד עם שם.
- קריאה לפונקציה גורמת להפעלת קטע הקוד שלה.
- פונקציה איננה פקודת C בסיסית אלא אוסף של פקודות פשוטות יותר.
- דוגמאות:

```
sin(), printf(), exit()
```


פונקציות ב-C

- קריאה לפונקציה נעשית על ידי ציון שמה עם סוגריים.
הסוגריים מציינים לקומפיילר שהשם שכתבנו מייצג פונקציה.
- בתוך הסוגריים מסודרים הפרמטרים שהפונקציה צריכה על מנת לבצע את עבודתה (אם יש כאלה).
- כל קריאה לפונקציה מחזירה ערך מטיפוס כלשהו. ניתן להשתמש בערך ההחזרה או להתעלם ממנו.
- נרחיב על פונקציות בהמשך הקורס.

שימוש בערך ההחזרה של הפונקציה לשם השמה ל-**y**

```
y = sin(x);
printf("cos(x)=%lf", cos(x));
```

אנו בוחרים שלא להשתמש בערך ההחזרה של **printf()**

שימוש בערך ההחזרה של הפונקציה **cos()** כפרמטר לפונקציה אחרת

חתימה של פונקציה

```
return_type function_name(param1,param2,…);
```

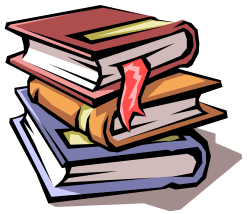
- חתימה של פונקציה מספקת תיאור כללי לגבי אופן השימוש בה.
כדי להשתמש בפונקציה, על המתכנת להכיר את החתימה שלה.
- בחתימת הפונקציה מצוינים שם הפונקציה, טיפוסי הפרמטרים שלה וטיפוס ההחזרה.

```
double sin(double x);
```

```
double pow(double a, double b);
```

הספרייה הסטנדרטית של שפת C

- מכילה אוסף פונקציות שימושיות בתחומים שונים.
- הספרייה מאורגנת במספר קבצים, כל קובץ מכיל פונקציות בתחום מסוים.
- אוסף הפונקציות בספרייה הסטנדרטית מוגדר במפרט של שפת C, ולכן זהה בכל הקומפיילרים.



קובץ הספרייה של השבוע...

ספריית הקלט/פלט של C

- על מנת להשתמש בספרייה, יש להוסיף בראש הקובץ את השורה

```
#include <stdio.h>
```

- פונקציות שנכיר היום:
  - `getchar()` ו-`putchar()`
  - `printf()` ו-`scanf()`

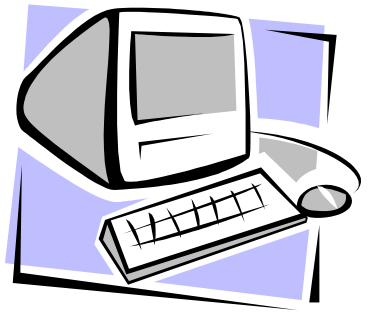
קלט/פלט ב-C

- **התקן קלט** – אובייקט שמייצר רצף ביטים, וניתן לקרוא אותם לתוך התוכנית בזה אחר זה. דוגמאות: המקלדת, קובץ נתונים, מידע שמגיע מהרשת.
- **התקן פלט** – אובייקט שיודע לקבל רצף ביטים, וניתן להזין אותם אליו בזה אחר זה. דוגמאות: המסך, קובץ ששומרים אליו נתונים, מידע שנשלח ברשת.
- תוכנה בשפת C יכולה "לדבר" עם מספר התקני קלט/פלט. ככלל יש צורך בפעולות מיוחדות על מנת ליצור קשר בין התוכנה להתקן.

התקני הקלט והפלט הסטנדרטיים

- לכל תוכנה מוגדרים מראש התקן קלט והתקן פלט סטנדרטיים מוכנים לעבודה.
- הקלט הסטנדרטי מגיע מהמקלדת.
- הפלט הסטנדרטי נשלח למסך.
- כשמריצים את התוכנית, ניתן לשנות באופן זמני את ההתקנים הסטנדרטיים עבור אותה הרצה, באמצעות redirection.
- התוכנית אינה יודעת בזמן הרצתה לאן מקושרים ההתקנים הסטנדרטיים; זהו תפקידה של מערכת ההפעלה. לכן היא גם איננה יודעת (ואין זה מעניינה) אם שינינו אותם.

התקני הקלט והפלט הסטנדרטיים



- הקלט הסטנדרטי נקרא **stdin**
- קיצור של standard input
- הפלט הסטנדרטי נקרא **stdout**
- קיצור של standard output

Redirection - תזכורת

command line	stdin	stdout
`c:\src> ex0`	מקלדת	מסך
`c:\src> ex0 < dat.in`	dat.in	מסך
`c:\src> ex0 > dat.out`	מקלדת	dat.out
`c:\src> ex0 < dat.in > dat.out`	dat.in	dat.out

קלט/פלט ישיר

```
int getchar(void);
```

קריאת תו בודד מהקלט הסטנדרטי.

פרמטרים: אין (**void** מציין שאין פרמטרים).

ערך החזרה: קוד ASCII של התו שנקרא מהקלט.

תאור: הפונקציה קוראת את התו הבא מ-**stdin** ומחזירה אותו. אם אין יותר תווים לקרוא, היא מחזירה את הערך המיוחד **EOF** (**EOF** הוא מספר שלם המוגדר כ-**#define**, ומובטח שהוא מחוץ לטווח הייצוג של **char** ושל טבלת ASCII).

קבלת הקלט מקובץ

באמצעות redirection, ניתן לקשר את התקן הקלט הסטנדרטי **stdin** לקובץ במקום למקלדת. במצב זה, אופן הקריאה הוא כזה:

- כאשר קוראים לראשונה קלט מ-**stdin**, התוכנית מתחילה לקרוא תווים מתחילת הקובץ.
- בכל פעם נוספת שקוראים קלט מ-**stdin**, התוכנית ממשיכה לקרוא מהמקום בו עצרה בפעם הקודמת.
- במידה והתוכנית מגיעה לסוף הקובץ, בפעם הבאה שננסה לקרוא מ-**stdin** יוחזר **EOF** על מנת לסמן שאין עוד קלט לקרוא.

קבלת הקלט מהמקלדת (1)

אם לא השתמשנו ב-redirection, הקלט הסטנדרטי מגיע לתוכנית מהמקלדת. במצב זה, אופן הקריאה הוא כזה:

- בפעם הראשונה שהתוכנית זקוקה לקלט מ-**stdin**, מערכת ההפעלה עוצרת את ריצת התוכנית ומציגה למשתמש סמן מהבהב.
- המשתמש מקליד את הקלט, ולוחץ על Enter. רק לאחר הלחיצה על Enter, כל מה שהמשתמש הקליד מועבר לתוכנית, והתוכנית חוזרת לרוץ.
- כעת, התוכנית קוראת את מה שדרוש לה מתוך מה שהמשתמש הקליד. במידה והוא הקליד יותר מן הנדרש, יתר התווים שהוקלדו **נשמרים** לפעם הבאה שהתוכנית מבצעת קריאה מ-**stdin**.

קבלת הקלט מהמקלדת (2)

- בפעם הבאה שהתוכנית קוראת קלט מ-**stdin**, ישנן שתי אפשרויות:
- במידה ונותרו תווים שעוד לא נקראו מהפעם הקודמת, התוכנית תשתמש בתווים אלו, ולא תציג כלל למשתמש את האפשרות להקליד קלט נוסף.
- רק במידה ולא נותרו עוד תווים מהפעם הקודמת, התוכנית תעצור את ריצתה ותאפשר למשתמש להכניס תווים נוספים. כרגיל, במידה והוא יקליד תווים עודפים הם יישמרו לפעם הבאה.
- שימו לב שכאשר הקלט מגיע מהמקלדת, הוא למעשה לעולם אינו מסתיים – ולכן לעולם לא נקבל **EOF**. עם זאת, בסביבות עבודה מסוימות קיימת האפשרות למשתמש "לשתול" סימן **EOF** מלאכותי בקלט (לרוב באמצעות לחיצה על Ctrl+Z במקלדת); פעולה זו מציינת לתוכנית שעליה להחזיר **EOF**, ממש כאילו נתקלה בסוף קובץ.

קבלת הקלט מהמקלדת (3)

שימו לב להבדל בין ההרצות (בשתיהן **stdin** הוא המקלדת):

```
enter number of vodka cups: 2
enter number of orange juice cups: 3
Your drink has 0.160000 part alcohol!
```

אין קריאה של קלט נוסף, כיוון שה-3 נותר מהפעם הקודמת

```
enter number of vodka cups: 2 3
enter number of orange juice cups:
Your drink has 0.160000 part alcohol!
```

קלט/פלט ישיר

```
int putchar(int ch);
```

כתיבת תו בודד לפלט הסטנדרטי.

פרמטרים:
ch: התו שיש לכתוב (קוד ASCII)

ערך החזרה: התו שנכתב, או **EOF** במקרה תקלה.

תאור: הפונקציה כותבת תו בודד ל-**stdout**, ומחזירה אותו. במקרה של כישלון, הפונקציה מחזירה במקום התו את הערך **EOF**.

קלט/פלט ישיר

```
#include <stdio.h>

int main()
{
  putchar('.');
  putchar('.');
  putchar('\n');
  putchar('o');
  return 0;
}
```

הפלט:

```
..
o
```


קלט/פלט ישיר

```
#include <stdio.h>
int main()
{
  int ch;
  ch = getchar();
  if (ch != EOF) {
    putchar(ch);
  }
  return 0;
}
```

EOF מציין סוף קובץ (End Of File). הוא מוגדר כ-**#define** ב-**stdio.h** למספר שלילי כלשהו

דוגמה: שיכפול קלט

mycopy.c

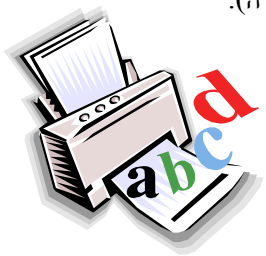
```
int main() {
  int ch;
  ch = getchar();
  while (ch != EOF) {
    putchar(ch);
    ch = getchar();
  }
  return 0;
}
```

```
c:\> mycopy < c:\mom\applepie.txt > d:\recipies\mypie.txt
```

פלט חכם

```
int printf(char* format, arg1, arg2, …)
```

כתיבה לפלט הסטנדרטי.



פרמטרים:
format: פורמט הפלט (בצורת מחרוזת עם תגי בקרה).
$\mathbf{arg_i}$: הערכים אותם יש להדפיס במקום תגי הבקרה.

ערך החזרה:
מספר התווים שנכתבו, או מספר שלילי בכישלון.

תאור: הפונקציה כותבת ל-**stdout** את המחרוזת **format**, כאשר היא מחליפה כל תג בקרה בערכו של הפרמטר המתאים לו.

תגי הבקרה הבסיסיים

תו	תאור	דוגמה
`%d`	מספר שלם (מטיפוס **int**)	`-234114`
`%u`	מספר שלם ללא סימן (מטיפוס **unsigned int**)	`234114`
`%f`	מספר ממשי עשרוני (מטיפוס **float**)	`234.114000`
`%e`	מספר ממשי עשרוני (מטיפוס **float**) – בייצוג מדעי	`2.341140e+02`
`%c`	תו בודד	`A`
`%s`	מחרוזת	`Intro to CS`

הרחבת אוסף תגי הבקרה

- לצורך קלט/פלט של הטיפוסים השונים, צריך תגי בקרה נוספים.
- תגי הבקרה מתקבלים מתגי הבקרה הבסיסיים על ידי הוספת אות לפניהם, על פי הטבלה הבאה:

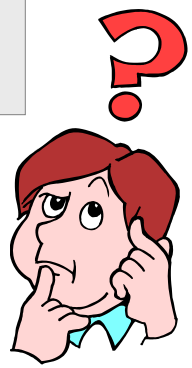
h	עבור טיפוס שלם: משנה לטיפוס מסוג **short**
l	עבור טיפוס שלם: משנה לטיפוס מסוג **long** עבור טיפוס ממשי: משנה לטיפוס **double**
L	עבור טיפוס ממשי: משנה לטיפוס **long double**

סיכום: תגי בקרה עבור טיפוסים מספריים

short	`%hd`	**unsigned short**	`%hu`
int	`%d`	**unsigned int**	`%u`
long	`%ld`	**unsigned long**	`%lu`
float (ייצוג רגיל)	`%f`	**float** (ייצוג מדעי)	`%e`
double (ייצוג רגיל)	`%lf`	**double** (ייצוג מדעי)	`%le`
long double (ייצוג רגיל)	`%Lf`	**long double** (ייצוג מדעי)	`%Le`

מה תדפיס הפקודה הבאה?

```
printf("  %d\n%s %c %lf\n  %d",
       50, "---------",
       247, 50/3.0, 3);
```



מה תדפיס הפקודה הבאה?

```
printf("  %d\n%s %c %lf\n  %d",
       50, "---------",
       247, 50/3.0, 3);
```

הפלט:

```
  50
------ ≈ 16.666667
  3
```


תגי בקרה עם פרמטרים

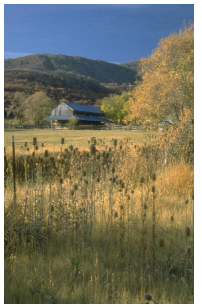
- הפרמטרים של תג בקרה משמשים לשליטה על אופן הצגת הנתון המתאים לו.
- הם מופיעים מייד לאחר סימן ה-% של התג, ולפני התווים המציינים את הטיפוס.
- הפרמטרים מובאים בסדר הבא:

% <סמן מקדים> <רוחב שדה> <.דיוק> <תג טיפוס>

תג הטיפוס – תו או זוג תווים המציינים את סוג הנתון (אחד מאלו המופיעים בשקף סיכום תגי הבקרה).

רוחב השדה

כל נתון מודפס כרצף של תווים בעל אורך מסוים.
רוחב השדה – מספר התאים המוקצים להדפסת הנתון.



```
printf("%d %lf %c %s",
       3500, 3.141, 'A',
       "hello world");
```

3500	3.141000	A	hello world
4	8	1	11

קביעת רוחב השדה

- רוחב השדה נכתב בתג הבקרה כמספר טבעי.
- מספר זה מציין רוחב שדה מינימאלי: כלומר, אם הנתון קצר ממנו, מוסיפים רווחים למילוי יתר השדה. אם הנתון ארוך ממנו, השדה מוגדל על מנת לאפשר את הדפסתו.
- בתוך השדה, הנתון מוצמד לימין, דהיינו מוסיפים רווחים משמאל לנתון על מנת למלא את השדה.
- אם הנתון הוא מספר, ונוצר צורך להוסיף משמאלו רווחים לשם מילוי השדה, אזי ניתן לגרום להחלפת הרווחים באפסים על ידי כתיבת 0 לפני רוחב השדה.

רוחב השדה

```
printf("%27s\n", "Here are some numbers:");
printf("%10d%10d%10d\n", 1, 2, 3);
printf("%10lf%10lf%10lf\n", 1.5, 2.5, 3.5);
```

```
     Here are some numbers:
          1         2         3
1.500000  2.500000  3.500000
```


דיוק ההצגה

- הדיוק ניתן כמספר טבעי שלפניו יש נקודה.
- במקרה של מספר ממשי, הדיוק מציין את מספר הספרות אחרי הנקודה שיש להציג.
- במקרה של מחרוזת, הדיוק מציין את מספר התווים המקסימלי שאנו רוצים להציג מהמחרוזת.

דיוק 8	→	דיוק 5	→	דיוק 2
2.71828183	→	2.71828	→	2.72
vacation	→	vacat	→	va

דיוק ההצגה

```
printf("%.20s \n",
       "sorry for the interruption");
printf("%.50s ok?\n",
       "sorry for the interruption");
printf("%.3lf %lf %.8lf \n",
       2.0/3, 2.0/3, 2.0/3);
```

```
sorry for the interr
sorry for the interruption ok?
0.667 0.666667 0.66666667
```

דוגמאות משולבות

```
printf("%10.3lf!\n", 7/3.0);
printf("%10.3lf!\n", 1.5);
printf("%010.3lf!\n", 1.5);
```

```
     2.333!
     1.500!
000001.500!
```

```
printf("%3.4lf\n", 1.0);
printf("%.2lf\n", 3.14159);
printf("%.4lf\n", 3.14159);
```

```
1.0000
3.14
3.1416
```


סמנים מקדימים

- סמנים מקדימים (במידה ועושים בהם שימוש) נכתבים ראשונים, מייד לאחר סימן ה-% של התג.
- אין חשיבות לסדר של הסמנים המקדימים (כאשר משתמשים ביותר מסמן מקדים אחד).
- ישנם שלושה סמנים מקדימים אפשריים:

סימן	תאור	דוגמה
-	מצמיד את הנתון לשמאל השדה	234114 2
+	מוסיף סימן '+' לפני מספרים אי שליליים	+234114
רווח	מוסיף רווח לפני מספרים אי שליליים	234114

דוגמאות משולבות

- דוגמאות נוספות עם מספרים:

```
printf("%09.3lf!\n", 1.5);
printf("%-9.3lf!\n", 1.5);
printf("%07.3lf!\n", 1.5);
printf("%01.3lf!\n", 1.5);
```

```
00001.500!
1.500    !
001.500!
1.500!
```

רוחב השדה מוגדל כיוון שהוא אינו מספיק להצגת הנתון בדיוק שדרשנו

```
printf("%d\n", -2000);
printf("%+d\n", 2000);
printf("% d\n", 2000);
```

```
-2000
+2000
 2000
```


דוגמאות משולבות

- דוגמאות נוספות עם מחרוזות:

```
printf("%10.5s!\n",  "supercalifragalistic");
printf("%-10.5s!\n", "supercalifragalistic");
printf("%.5s!\n",    "supercalifragalistic");
printf("%10s!\n",    "supercalifragalistic");
```

```
     super!
super     !
super!
supercalifragalistic!
```

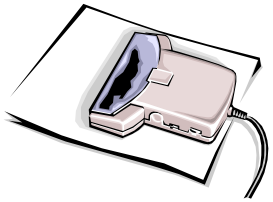
קלט חכם

```
int scanf(char* format, arg1, arg2, …)
```

קריאה מהקלט הסטנדרטי.

<u>פרמטרים</u>:
format: פורמט הקלט (בצורת מחרוזת עם תגי בקרה).
$\mathbf{arg_i}$: המשתנים לתוכם יש לקרוא את הקלט (עם סימן **&**).

<u>ערך החזרה</u>:
בהצלחה – מספר תגי הבקרה שהופיעו במחרוזת הבקרה, או מספר קטן מכך בכישלון (למעשה, מספר תגי הבקרה שנקראו, או EOF במידה ולא היה אפשר כלל לבצע קלט).



<u>תאור</u>: הפונקציה קוראת מ-**stdin** את המחרוזת **format**, כאשר כל ערך שמתאים לתג בקרה נכתב למשתנה המצוין.

מחרוזת הבקרה

מחרוזת הבקרה יכולה להכיל:

- תגי בקרה – זהים לאלו של **printf()**, אך ללא כל פרמטרים. במקרה זה הערך שנקרא מהקלט מומר על פי תג הבקרה, ונרשם למשתנה המתאים.
- רווחים – מציינים אפשרות שיופיעו רווחים בקלט.
- תווים רגילים – צריכים להתאים בדיוק לתווים המופיעים בקלט. התווים שנקראים אינם נכתבים למשתנים כלשהם.

דוגמאות (1)

```
scanf("%d", &i);
```

```
> 1324
i = 1324
```

```
scanf("%lf", &f);
```

```
> 3.1415
f=3.1415
> 3141.5e-3
f=3.1415
```

דוגמאות (2)

```
scanf("%c%c", &c1, &c2);
```

```
> abcd                    > a b c d
c1='a' c2='b'             c1='a' c2=' '
```

```
scanf("%c %c", &c1, &c2);
```

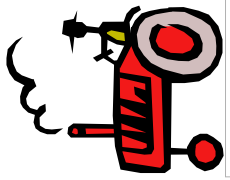
```
> abcd                    > a b c d
c1='a' c2='b'             c1='a' c2='b'
```

דוגמאות (3)

```
scanf("%d%c", &i, &c);
```

```
> 3415abcd
i=3415 c='a'
> 3415 abcd
i=3415 c=' '
```

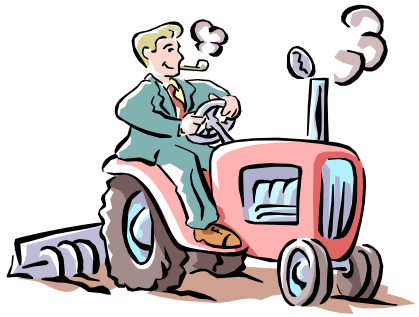
```
scanf("%d %c", &i, &c);
```



```
> 3415abcd
i=3415 c='a'
> 3415 abcd
i=3415 c='a'
```


דוגמאות (4)

```
scanf("%d%c%d", &i, &c, &j);
```



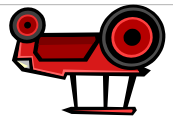
```
> 215.37
i=215 c='.' j=37
> 215 37
i=215 c=' ' j=37
```


דוגמאות (5)

```
scanf("vodka=%d", &i);
```

```
> vodka=3
i=3
```

```
> vodka = 3
*error*
```



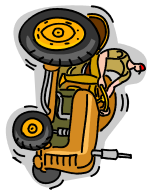
```
scanf("vodka = %d", &i);
```

```
> vodka=3
i=3
```

```
> vodka = 3
i=3
```


דוגמאות (6)

```
scanf("vodka = %d orange = %d", &v, &o);
```



```
> vodka=3 orange=5
> vodka = 3orange =5
> vodka = 3
  orange = 5
```

OK

```
> orange = 3 vodka = 5
```

X

דוגמאות (7)

```
scanf("vodka = %d orange = %d", &v, &o);
```

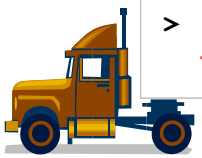
```
> vodka = 3 orange = 5
  v=3   o=5
> vodka = 3 apple = 5
  v=3   o=unchanged
> orange = 5 vodka = 3
  v=unchanged   o=unchanged
> <Ctrl-Z>
  v=unchanged   o=unchanged
```

ערך החזרה: 2

ערך החזרה: 1

ערך החזרה: 0

ערך החזרה: **EOF**



וידוא הצלחה של פעולות קלט

- שלב קבלת הקלט בתוכנית הינו בדרך כלל רגיש ובעל אפשרויות רבות להיכשל, ולכן תהליך וידוא הקלט בתוכנית חשוב מאוד.
- בדיקת הצלחה של פעולת קלט מורכבת למעשה משני חלקים:

א. <u>וידוא הצלחת פונקצית הקלט</u>: עלינו לוודא שפונקצית הקלט סיימה ללא תקלות, והצליחה לקרוא את כל הקלט הדרוש.

ב. <u>וידוא נכונות לוגית של הקלט</u>: עלינו לוודא שהקלט שנקרא מתאים מבחינת תוכנו למה שאנו מצפים.

- לשם ההדגמה, נכתוב תוכנית קצרה שקוראת מהמשתמש שבר פשוט מהצורה a/b, ומחשבת את ערכו.

וידוא הצלחה של פעולות קלט

- נתחיל מכתיבת התוכנית עצמה, ללא כל בדיקות קלט:

```
int main()
{
  int a, b, c;

  printf("Enter a fraction: ");

  scanf("%d ", &a);
  c = getchar();
  scanf(" %d", &b);

  printf("Fraction value = %lf\n",
         a/(double)b);
  return 0;
}
```


וידוא הצלחת פונקציות קלט

- על מנת לבדוק ש-**scanf()** הצליחה, עלינו לבדוק את ערך ההחזרה שלה. פונקציה זו מצליחה אך ורק כאשר ערך ההחזרה שלה זהה למספר תגי הבקרה שהופיעו במחרוזת הבקרה; במקרה של כישלון, יוחזר מספר קטן יותר, או לחילופין **EOF**, שהוא כזכור מספר שלילי.
- לפיכך, בדיקת ההצלחה של פעולת הקלט הראשונה יכולה להראות כך:

מכסה גם את המקרה של החזרת 0, וגם של החזרת **EOF**

```
if (scanf("%d ", &a) < 1) {
  printf("Error reading fraction\n");
  return 0;
}
```

וידוא הצלחת פונקציות קלט

- על מנת לבדוק ש-**getchar()** הצליחה, עלינו לבדוק שלא הוחזר על ידה ערך **EOF**. זכרו שערך זה אינו ערך ASCII חוקי, והוא אינו ניתן לייצוג באמצעות משתנה **char** (טבלת ASCII מכסה בדיוק את טווח הייצוג של **char**, ולכן הכרחי להגדיר את **EOF** מחוץ לטווח זה). לפיכך, אנו קוראים את ערך ההחזרה של הפונקציה לתוך משתנה מטיפוס **int**, שמסוגל לייצג גם את כל טבלת ASCII וגם את הערך **EOF**.

```
c = getchar();
if (c == EOF) {
  printf("Error reading fraction\n");
  return 0;
}
```


וידוא נכונות לוגית של הקלט

- בהנחה שפעולת הקלט הצליחה, עדיין עלינו לבדוק שהקלט שקראנו הוא חוקי!
- במקרה שלנו, יש לבדוק שני דברים: (א) שהתו **c** שנקרא הוא אמנם תו **'/'**, ו-(ב) שהערך **b** איננו 0. שימו לב שבשני מקרים אלו פונקציות הקלט עצמן יסתיימו בהצלחה, ולכן השגיאה אינה בתהליך קבלת הקלט – אלא בתוכן הקלט ממש.
- עבור **b** נקבל את הבדיקה הבאה:

```
if (b==0) {
  printf("Error: division by zero\n");
  return 0;
}
```


הקוד המלא (1)

```
printf("Enter a fraction: ");

if (scanf("%d ", &a) < 1) {
  printf("Error reading fraction\n");
  return 0;
}

c = getchar();
if (c == EOF) {
  printf("Error reading fraction\n");
  return 0;
}
if (c != '/') {
  printf("Error reading fraction\n", c);
  return 0;
}
```

Run!

הקוד המלא (2)

```
…
if (scanf(" %d", &b) < 1) {
  printf("Error reading fraction\n");
  return 0;
}

if (b==0) {
  printf("Error: division by zero\n");
  return 0;
}

printf("Fraction value = %lf\n",
       (double)a/b);

return 0;
```