פרק 2
סקירת שפת C

אלגוריתם

- אלגוריתם הוא פתרון של בעיה כלשהי.
- בתיאור של אלגוריתם, מפורטים שלבי הפתרון בזה אחר זה, כאשר כל שלב הוא **פעולה שאנו יודעים לבצע** (...או לכל הפחות, אנו יודעים שקיימת האפשרות לבצע אותה).
- אנלוגיות מתחומים אחרים:
  - בישול: נניח שעלינו להכין גפילטע-פיש. זוהי בעיה שאיננו יודעים לפתור, אולם אם ניעזר **במתכון**, שבו כל שלב מתאר פעולה פשוטה במטבח, נוכל (בתקווה..) להצליח במשימה.
  - ניווט: נניח שאנו צריכים כעת להביא את הגפילטע-פיש לדודה מיידל מכפר אז"ר. גם זו בעיה קשה, ואולם אם נקבל **הוראות מפורטות** כיצד להגיע, ייתכן שנצליח בכך.

תוכנית ה-C הראשונה

התוכנית:

```
#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
```

Run!

הפלט:

```
Hello World!
```


ניתוח התוכנית

```
#include <stdio.h>

int main()
{
  printf("Hello World!
  return 0;
}
```

מציין שאנו רוצים להשתמש בפקודות שמסופקות על ידי הקובץ `stdio.h`

הקובץ `stdio.h` מספק פקודות לצורך הצגת פלט על המסך, וכן לקריאה של קלט מהמקלדת. אנו נזדקק לפקודות אלה בכל תוכנית C שנכתוב, ולכן נציין שורה זו בתחילת כל קובץ קוד.

ניתוח התוכנית

```c
#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
```

פקודה שמורה להדפיס למסך את המחרוזת "Hello World!".

את הפקודה **printf()** ננתח ביתר פירוט בהמשך הפרק.

ניתוח התוכנית

```c
#include <stdio.h>

int main()
{
  printf("Hello Worl
  return 0;
}
```

מציין את תחילת התוכנית. מכאן, בין שני הסוגריים המסולסלים **{}**, יש לכתוב את כל הפקודות של התוכנית.

הפקודות שאנו רוצים שהתוכנית תבצע נכתבות בין זוג הסוגריים המסולסלים **{** ו-**}**, בזו אחר זו לפי סדר הביצוע שלהן.

נקודות כלליות לגבי שפת C

- כל פקודה ב-C חייבת להסתיים בתו **;** (נקודה-פסיק).
- מבחינת שפת C, אין משמעות לרווחים בין הפקודות, או לירידת שורה ביניהן. שפת C מפרידה בין הפקודות אך ורק על פי התו **;** .
- **למרות זאת**, יש להתחיל כל פקודה בשורה חדשה, ולשמור על צורה כללית בהירה של הקוד על מנת שיהיה קל לקרוא אותו ולהבין את המבנה שלו.

ניתוח התוכנית

```c
#include <stdio.h>

int main()
{
  printf("Hello
  return 0;
}
```

פקודה זו מציינת שהתוכנית סיימה את ריצתה בהצלחה, וגורמת להפסקת ריצת התוכנית.

הפקודה **return 0;** מציינת שהתוכנית סיימה לרוץ ללא תקלות. היא צריכה להיות הפקודה האחרונה בכל תוכנית C, וגורמת להפסקת ריצת התוכנית.

נקודות כלליות לגבי שפת C

הנה גרסה נוספת של אותה התוכנית, שהיא חוקית לחלוטין מבחינת שפת C, אולם כאן בלבלנו את הריווח בין הפקודות. שימו לב להבדל בין שתי הגרסאות מבחינת הקריאות ובהירות הקוד (וגם מבחינת הציון..).

```
#include <stdio.h>
int
main(){
printf("Hello World!\n")
;return 0;}
```

```
#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
```


על הפקודה printf()

- הפקודה **printf()** גורמת להדפסת **מחרוזת** (=רצף של אותיות, כלומר טקסט) למסך.
- השימוש בה נעשה על ידי כתיבת המחרוזת הרצויה בין זוג מירכאות ("...").
- זוג התווים **\n** (backslash-n) מציין התחלת שורה חדשה. כאשר זוג התווים הזה מופיע במקום כלשהו במחרוזת, הוא **אינו מודפס למסך**, ובמקום זאת הוא גורם לירידת שורה במסך, כך שהדפסת יתר המחרוזת ממשיכה בתחילת השורה הבאה. שימו לב ש-**printf()** איננה יורדת שורה אלא אם מציינים במפורש שעליה לעשות כן (בעזרת הסימון **\n**).

דוגמאות ל- printf()

הפקודה:

```
printf("Hello World!");
printf("Hello World!");
```

הפלט:

```
Hello World!Hello World!
```

הפקודה:

```
printf("Hello World!\n");
printf("Hello World!");
```

הפלט:

```
Hello World!
Hello World!
```


דוגמאות ל- printf()

הפקודה:

```
printf("Hello\nWorld!");
```

הפלט:

```
Hello
World!
```

הפקודה:

```
printf("Hello\nWo");
printf("rld!");
```

הפלט:

```
Hello
World!
```

כתיבת הערות בתוכנית C

- ניתן לשלב בתוך קבצי קוד C הערות והסברים למיניהם, בצורת טקסט חופשי, לנוחות המתכנת.
- בתוך איזור שמסומן כהערה, ניתן לכתוב טקסט כרצוננו, בין אם זה פקודות C, שפה מדוברת, או סימנים. הקומפיילר של C **מתעלם לחלוטין** מכל מה שנמצא בתוך איזור של הערה.
- ניתן לסמן הערה בשתי צורות. בצורה הראשונה, הערה מתפרשת מצמד התווים **/*** הפותח את ההערה, ועד שמופיע צמד התווים ***/** הסוגר אותה (ניתן לכתוב כך הערה על פני כמה שורות). בצורה השנייה, רק מה שמופיע באותה השורה, מצמד התווים **//** ועד סוף אותה השורה, נחשב כהערה.

כתיבת הערות בתוכנית C

```
/* This is a comment. It may span several lines.
   The compiler will ignore everything here, including
   C commands and preprocessor directives such as
   #include <idontknow.h>
   this type of comment ends with a */


int x; // I'm also a comment. Everything from the
       // sign `till the end of the line will be
       // completely ignored by the C compiler.
```

הערה: שפת C "הטהורה" מכירה למעשה רק בסוג הראשון; C++ הוסיפה את הסוג השני. עם זאת, רוב הקומפיילרים של C מקבלים את שני הסוגים.

דוגמה: תיעוד של קובץ

```
/*********************************************************************
 *
 * File name: shared.h
 *
 * Author: Bill Gates
 *         01234567-8
 *  email: bill@ms
 *
 * Contents:
 *
 * - Include statements for some commonly used header files.
 * - Declarations of common methods, globals, constants and types.
 *
 * Created: 23.6.1981
 *
 *********************************************************************/
                              .......
/*********************************************************************
                        End of file shared.h
*********************************************************************/
```


דוגמה: חלוקת תוכן הקובץ

```
/*-------------------------------------------------------------------
  Include files:
-------------------------------------------------------------------*/
#include <stdio.h>


/*===================================================================
  Constants and definitions:
===================================================================*/
#define PI 3.14159
typedef enum {false, true} boolean;


/*===================================================================
                  Function implementation:
===================================================================*/
int main() {...}
```

וכעת: תוכנה לברמנים

- **וודקה תפוזים** היא תערובת טעימה ומזינה של וודקה ומיץ תפוזים.
- נתבקשנו לתכנת את התוכנית הבאה, ששואלת את המשתמש כמה כוסות וודקה וכמה כוסות תפוזים יש במשקה, ומחזירה את אחוז האלכוהול שבו.
- בוודקה יש 40% אלכוהול.

```
enter number of vodka cups: 5
enter number of orange juice cups: 3
vodka with orange juice has 0.25 part alcohol
```


נתחיל בכתיבת התוכנית

- עלינו לבקש מהמשתמש שיקליד את מספר כוסות הוודקה במשקה. נשתמש ב-**printf()** לשם כך.
- איננו רוצים שהתוכנה תרד שורה בסוף ההדפסה, כיוון שיהיה למשתמש יותר נוח להקליד את מספר הכוסות באותה השורה בה הדפסנו את הבקשה. לכן איננו מסיימים את המחרוזת עם **\n** .

```
#include <stdio.h>
int main()
{
  printf("enter number of vodka cups: ");
  return 0;
}
```


נמשיך בכתיבת התוכנית

- לאחר שנקבל את מספר כוסות הוודקה מהמשתמש, עלינו לאחסן את הנתון הזה במקום כלשהו כך שנוכל להשתמש בו בהמשך.
- כיצד מאחסנים נתונים ב-C ?

```
#include <stdio.h>
int main()
{
  printf("enter number of vodka cups: ");
  return 0;
}
```


אחסון נתונים ב-C

- הנתון הבסיסי בשפת C הוא **המספר**.
- מבחינים בין שני סוגים יסודיים של מספרים – מספרים שלמים, ושברים עשרוניים.
- נתון מספרי מאוחסן **במשתנה**. כל משתנה מסוגל לאחסן מספר יחיד.
- לכל משתנה, יש להצהיר מראש איזה טיפוס של מספר הוא יאחסן (שלם או שבר), ותכונה זו של המשתנה אינה ניתנת לשינוי במהלך התוכנית.
- במקרה שלנו, נניח שמספר כוסות הוודקה במשקה הוא שלם.

נמשיך בכתיבת התוכנית

- עלינו לאחסן שני מספרים שלמים (מס' כוסות הוודקה ומס' כוסות מיץ התפוזים), ולשם כך אנו זקוקים לשני משתנים מטיפוס מספר שלם.
- בשפת C יש להצהיר על כל המשתנים שבכוונתנו להשתמש בהם בתוכנית **בתחילת הקוד**, מייד לאחר הסוגר הפותח { .
- בהצהרה על משתנה אנו גם מציינים את סוג המשתנה, וגם נותנים לו שם שבו נשתמש על מנת להתייחס למשתנה זה במהלך התוכנית.

הצהרה על משתנים

שורות אלה מגדירות שני משתנים, בשמות **vodke_cups** ו-**orange_cups**.

זה האזור בו מצהירים על משתנים

```
#include <stdio
int main()
{
  int vodka_cups;
  int orange_cups;

  print("enter number of vodka cups: ");
  return 0;
}
```

המילה **int** מציינת שהמשתנה הוא מטיפוס מספר שלם (integer). על מנת לציין שבר עשרוני, יש להשתמש במילה **double**.

קריאת נתונים מהמשתמש

- קריאת נתון מהמשתמש נעשית באמצעות הפקודה **scanf()**.
- לפקודה **scanf()** יש לציין איזה טיפוס של נתון אנו רוצים לקרוא מהמשתמש (שלם או שבר), וכן לאיזה משתנה לכתוב את הנתון שהמשתמש מכניס.
- כאשר התוכנית מגיעה בזמן ריצתה לפקודה **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.

שימוש בפקודה scanf()

מציין את טיפוס הנתון אותו יש לקרוא. **"%d"** מציין מספר שלם, ואילו **"%lf"** מציין שבר עשרוני. יש לוודא התאמה בין הטיפוס המצוין כאן לטיפוס של המשתנה שאנו נותנים ל-**scanf()**.

שם משתנה היעד אליו יוכנס הנתון.

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

ב-**scanf()** יש לכתוב תו **&** לפני שם המשתנה.

התוכנית עד כה

```c
#include <stdio.h>
int main()
{
  int vodka_cups;
  int orange_cups;

  printf("enter number of vodka cups: ");
  scanf("%d", &vodka_cups);
  printf("enter number of orange juice cups: ");
  scanf("%d", &orange_cups);

  return 0;
}
```


חישובים ב-C

- כעת נותר לנו לחשב מהו אחוז הוודקה במשקה.
- נעשה זאת בשלושה שלבים:
  1. נחשב את סך כוסות המשקה שיש לנו.
  2. נחשב את סך האלכוהול שיש במשקה שלנו.
  3. נחלק את כמות האלכוהול הכוללת בכמות המשקה הכוללת.
- ב-C ניתן להפעיל את ארבע פעולות החשבון היסודיות (כפל, חילוק, חיבור וחיסור) על ערכים מספריים כלשהם. התוצאה תאוחסן במשתנה אותו אנו מציינים.

חישובים ב-C

- ב-C ישנן חמש פעולות חשבון יסודיות:

חיבור	+
חיסור	-
כפל	*
חילוק	/
שארית	%

- ניתן להפעיל פעולות אלה הן על מספרים מפורשים, והן על משתנים שמכילים מספרים, כרצוננו. שפת C מחשבת את הפעולות הללו על פי סדרן המתמטי (כלומר כפל לפני חיבור).

חישובים ב-C

- פעולת החילוק היא פעולה מיוחדת בין חמש הפעולות, כיוון שיש לה **שתי גרסאות**:
- אם אחד או שני המספרים בפעולת החילוק הם שברים, החילוק מתבצע באופן הרגיל.
- אם שני המספרים הם שלמים, מתבצע **חילוק בשלמים**. במקרה זה פעולת החילוק מחזירה מספר שלם, שהוא החלק השלם של תוצאת החילוק, וכל מה שלאחר הנקודה העשרונית – מושמט.
- לדוגמה:

```
7/3.2 = 2.1875
7/3   = 2
```

חישובים ב-C

- פעולת השארית (מודולו) אף היא פעולה מיוחדת. פעולה זו מוגדרת על **שלמים בלבד**.
- פעולת המודולו מחזירה את השארית של פעולת החילוק בין שני המספרים.
- כאשר **x** ו-**y** שלמים, מתקיים (למה?):

```
(x/y)*y + (x%y) = x
```

- דוגמאות:

```
7/3 = 2        7%3 = 1
9/3 = 3        9%3 = 0
```


חישובים ב-C

- לצורך חישוב סך כוסות המשקה, נוסיף בתחילת התוכנית הצהרה על משתנה נוסף מטיפוס שלם שייקרא **cups_tot** ויאחסן את התוצאה.
- בזאת אנו ערוכים לביצוע פעולת החישוב הראשונה!

```
#include <stdio.h>
int main()
{
  int vodka_cups;
  int orange_cups;
  int cups_tot;

  printf("enter number of vodka cups: ");
  scanf("%d", &vodka_cups);
  printf("enter number of orange juice cups: ");
  scanf("%d", &orange_cups);
  cups_tot = vodka_cups + orange_cups;

  return 0;
}
```


חישובים ב-C

- עלינו לחשב כמה אלכוהול יש **סך הכל** במשקה. אלכוהול יש רק בוודקה (ולא במיץ תפוזים), לכן סך האלכוהול במשקה מתקבל פשוט ע"י החישוב – (סך כוסות וודקה) x (0.4)
- התוצאה אינה שלמה, ועלינו לאחסן אותה במשתנה מטיפוס **שבר עשרוני**. טיפוס זה נקרא טיפוס **double** ב-C.

```
#include <stdio.h>
int main()
{
  int vodka_cups;
  int orange_cups;
  int cups_tot;
  double alc_tot;

  printf("enter number of vodka cups: ");
  scanf("%d", &vodka_cups);
  printf("enter number of orange juice cups: ");
  scanf("%d", &orange_cups);
  cups_tot = vodka_cups + orange_cups;
  alc_tot = vodka_cups * 0.4;

  return 0;
}
```


חישובים ב-C

- לסיום, נחשב את אחוז האלכוהול במשקה, על ידי חלוקה של סה"כ האלכוהול במספר כוסות המשקה.

```
#include <stdio.h>
int main()
{
  int vodka_cups;
  int orange_cups;
  int cups_tot;
  double alc_tot;
  double alc_fraction;

  printf("enter number of vodka cups: ");
  scanf("%d", &vodka_cups);
  printf("enter number of orange juice cups: ");
  scanf("%d", &orange_cups);
  cups_tot = vodka_cups + orange_cups;
  alc_tot = vodka_cups * 0.4;
  alc_fraction = alc_tot / cups_tot;

  return 0;
}
```

הדפסת נתונים מספריים

- נותר לנו להציג את התוצאה למשתמש!
- פקודת **printf()** שהשתמשנו בה עד כה להדפסת מחרוזות למסך משמשת גם להדפסת נתונים מספריים למסך.
- על מנת להדפיס נתון מספרי כלשהו בעזרת **printf()**, יש לציין זאת בתוך המחרוזת עצמה, במקום בו אנו רוצים שהנתון יוצג, תוך ציון הטיפוס של הנתון. לאחר מכן יש לספק את הנתון עצמו (כמספר מפורש או כמשתנה שמכיל את המספר), כאשר טיפוס הנתון מתאים לטיפוס שציינו.

שימוש בפקודה printf()

הפקודה **printf()** מדפיסה את המחרוזת כרגיל עד לנקודה זו. כאשר היא נתקלת בסימון המיוחד **"%d"** היא אינה מדפיסה אותו, ובמקום זאת מחליפה אותו בערך שציינו אחרי הפסיק. כמו ב-**scanf()**, גם כאן **"%d"** מציין מספר שלם ואילו **"%lf"** מציין שבר עשרוני. לאחר הדפסת הנתון, **printf()** ממשיכה להדפיס את יתר המחרוזת כרגיל.

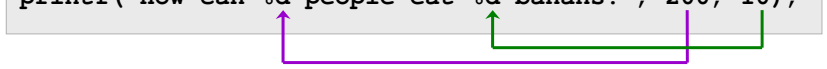
הנתון שברצוננו להדפיס

```
printf("You have %d cups total", cups_tot);
```


עוד קצת על printf()

- ניתן לתת ל-**printf()** יותר מערך אחד להדפסה.
- לכל ערך, נציין במקום המתאים במחרוזת של **printf()** את הטיפוס של הערך שברצוננו להדפיס.
- לאחר המחרוזת יש להביא רשימה (מופרדת על ידי פסיקים) של כל הערכים שברצוננו להדפיס, בהתאם לסדר הופעתם במחרוזת.

```
printf("How can 2 people eat 300 bananas?");
printf("How can %d people eat %d banans?", 200, 10);
```



הדפסת התוצאה

- לאחר החישוב, אנו יכולים להדפיס את התוצאה, שהיא מטיפוס **double** ולכן עלינו להשתמש ב-**%lf**.
- שימו לב ש-C מתעלמת מכך שעברנו שורה באמצע הפקודה.
- המקומות היחידים בקוד בהם איננו יכולים להוסיף רווחים או לעבור שורות כרצוננו, הם באמצע שם של משתנה או פקודה, ובאמצע מחרוזת.

```
#include <stdio.h>
int main()
{
  int vodka_cups;
  int orange_cups;
  int cups_tot;
  double alc_tot;
  double alc_fraction;

  printf("enter number of vodka
  scanf("%d", &vodka_cups);
  printf("enter number of orange
  scanf("%d", &orange_cups);
  cups_tot = vodka_cups + orange
  alc_tot = vodka_cups * 0.4;
  alc_fraction = alc_tot / cups_
  printf("Your drink has %lf part alcohol!\n",
         alc_fraction);

  return 0;
}
```

Run!

שימוש ב-#define

- בקוד התוכנית אנו עושים לעיתים קרובות שימוש במספרים מפורשים לצורך חישובים שונים.
- נהוג ומומלץ להגדיר את כל המספרים המפורשים הללו בתחילת התוכנית, באמצעות הפקודה **#define**.
- בפקודה **#define** משתמשים בראש התוכנית, באזור של פקודות ה-**#include** (בדרך כלל מייד לאחריהן).
- לדוגמה, בתוכנית שלנו מופיע מספר מפורש בשורה :

```
alc_tot = vodka_cups * 0.4;
```


שימוש ב-#define

במקום כתיבה מפורשת של המספר 0.4 בקוד, נעדיף להגדיר מספר זה בתחילת התוכנית **כקבוע**, באמצעות **#define** :

```
#include <stdio.h>
#define VODKA_ALC_PERCENT 0.4

int main()
{
  int vodka_cups;
  ...
  cups_tot = vodka_cups + orange_cups;
  alc_tot = vodka_cups * VODKA_ALC_PERCENT;
  ...
}
```


#define לעומת משתנים

- מנגנון הפעולה של **#define** הוא פשוט מאוד: בכל מקום שבו אנו כותבים את שם הקבוע שהגדרנו, הקומפיילר מתייחס לכך **כאילו הקלדנו את המספר הזה מפורשות** בתוך הקוד.
- חשוב להדגיש שבתהליך זה **לא מעורב משתנה כלשהו** ולא מוקצה כל זיכרון. שם הקבוע אינו מייצג משתנה והתהליך כולו מכאני לחלוטין – בכל מקום בו מופיע שמו של הקבוע, הקומפיילר פשוט מחליף אותו במספר המתאים ממש כאילו אנו הקלדנו אותו שם מלכתחילה.
- לפיכך, לא ניתן לשנות את ערכו של קבוע **#define** או לבצע השמה לתוכו, כיוון שהוא אינו מאוחסן כלל בזיכרון.

יתרונות השימוש ב-#define

- **והרי סיפור:**
- דני וציפרון ראו שניהם את הפוטנציאל העצום הטמון בתוכנת הברמנים, והחליטו להרחיב אותה. כל אחד מהם, בנפרד, פיתח תוכנה המחשבת את כמות האלכוהול במספר רב של משקאות.
- שניהם הניחו 40% אלכוהול בוודקה.
- **דני** הגדיר בראש התוכנית קבוע לכל סוג של אלכוהול, והשתמש בו לאורך הקוד. הקוד שלו מתחיל כך:

```
#define  VODKA_ALC_PCT    0.4
#define  WHISKEY_ALC_PCT  0.4
#define  BEER_ALC_PCT     0.05
#define  KAHLUA_ALC_PCT   0.26
…
```

- **ציפרון** כתב בכל חישוב הנוגע לוודקה את המספר 0.4 מפורשות.

יתרונות השימוש ב-#define

- דני וציפרון נסעו שניהם ל-BarCode לנסות ולמכור את התוכנה. להפתעתם, הם גילו שמשתמשים שם בוודקה Obsolete, שהיא בעלת 32% אלכוהול בלבד.
- **דני** חזר לביתו ושינה את ההגדרה של **VODKA_ALC_PCT** מ-0.4 ל-0.32. שמח וטוב לב חזר עם התוכנה המעודכנת, מכר אותה ועשה ממון רב. הוא היום מנכ"ל *Soft-Drink, Inc.*
- **ציפרון** מצא עצמו עובר לילה שלם על אלפי שורות קוד בחיפוש אחר הערך 0.4. בנוסף, הוא נאלץ לברר בכל שורה למה הכוונה – האם לאחוז האלכוהול בוודקה, ברנדי או וויסקי. ציפרון הרים ידיים ב-5:00 לפנות בוקר. כיום הוא עובד ב-BarCode כמאבטח.

יתרונות השימוש ב-#define

- גם אם אנו משתמשים במספר כלשהו רק במקום אחד בקוד, כדאי מאוד להגדיר אותו כ-**#define**. שלוש סיבות טובות לכך:

1. ייתכן שבעתיד נשתמש בקבוע זה בעוד מקום שקשה לנו עדיין לחזות. מעקרון העצלנות נובע שבשלב זה נשאיר את הקוד בכל זאת עם מספרים מפורשים (כי הם רק שניים..), ומכאן המצב רק ידרדר.
2. אולי בעתיד נרצה לשנות ערך זה. מדוע לחפש אותו בכל הקוד, כשאפשר לבודד את כל הקבועים ברשימה נוחה בראשית התוכנית?
3. קריאות הקוד: שימוש במלים משמעותיות במקום מספרים סתמיים.

- מסקנה – עקרון הערכים הקבועים: כל מספר שמופיע מפורשות בקוד (פרט אולי ל-0 או 1), צריך כנראה להיות מוגדר כקבוע.

התוכנית המלאה (1)

ניתן להצהיר על מספר משתנים באותה השורה על ידי הפרדת שמותיהם בפסיקים.

```
#include <stdio.h>
#define VODKA_ALC_PERCENT 0.4

int main()
{
  int vodka_cups, orange_cups, cups_tot;
  double alc_tot, alc_fraction;

  printf("Enter number of vodka cups: ");
  scanf("%d", &vodka_cups);
  …
```


התוכנית המלאה (2)

```
  …
  printf("Enter number of orange juice cups: ");
  scanf("%d", &orange_cups);

  cups_tot = vodka_cups + orange_cups;
  alc_tot = vodka_cups * VODKA_ALC_PERCENT;
  alc_fraction = alc_tot / cups_tot;

  printf("Your drink has %lf part alcohol!",
         alc_fraction);

  return 0;
}
```

משפטי תנאי

- עד עתה, כתבנו פקודות שהתבצעו כולן בזו אחר זו, בלא שום אפשרות לשנות את התנהגות התוכנית בזמן ריצתה.
- בשפת C אנו יכולים לכתוב פקודה שתתבצע **רק במצבים מסוימים**, באמצעות שימוש בפקודה **`if`**.
- קטע הקוד הבא בוחן את ערכו של המשתנה **`x`**, ורק אם הוא קטן מאפס מודפסת למסך השורה **`"negative :("`**

```
if (x < 0) {
   printf("negative :(");
}
```


משפטי תנאי

- בחברת ההימורים הפופולארית *C-ha'mazal* החליטו לממש מערכת אבטחה מתקדמת, שתמנע מצעירים להיכנס לאתר. הקוד הבא מדגים את המערכת, שנפוצה כיום באתרי אינטרנט רבים:

```
double age;
printf("Enter your age: ");
scanf("%lf", &age);

if (age < 18) {
  printf("You are too young!\n");
  return 0;
}
```

קטע הקוד בין הסוגריים המסולסלים מבוצע רק אם התנאי מתקיים

ביצוע מותנה – if-else

- קיימת האפשרות לציין בנוסף גם פקודות שיבוצעו במקרה שהתנאי אינו מתקיים.
- קטע הקוד השייך ל-**`if`** מבוצע **רק** אם הגיל גדול או שווה ל-18. אם הוא קטן מ-18, מבוצע **במקום** קטע הקוד השייך ל-**`else`**.

```
if (age >= 18) {
  printf("Welcome to C-hamazal.com! \n");
  printf("<< How much money do you want ");
  printf("to lose today?™ >> \n");
}
else {
  printf("You are too young!\n");
  return 0;
}
```


ביצוע מותנה – if-else

Run!

```
#include <stdio.h>
int main()
{
  double age;
  printf("enter your age: ");
  scanf("%lf", &age);

  if (age >= 18) {
    printf("Welcome to C-hamazal.com! \n");
    printf("<< How much money do you want ");
    printf("to lose today?™ >> \n");
  }
  else {
    printf("You are too young!\n");
    return 0;
  }
  …  /* Allow access and show the website */
}
```

לולאת while

- עד עתה, אתר האינטרנט היה בלתי חביב למדי לצעירים... האם לא יהיה זה הוגן לתת להם הזדמנות נוספת לענות על השאלה?
- נשים לב שבתוכניות שכתבנו עד כה, בוצעה כל פקודה **לכל היותר פעם אחת**.
- מה שאנו זקוקים לו זה מנגנון שמאפשר לבצע אותן פקודות שוב ושוב: אנו זקוקים ל**לולאה**.

```
while (x < 0) {
  printf("x is still negative :(\n");
  x = x + 1;
}
```

גוף הלולאה

לולאת while - מבנה

- כאשר התוכנית מגיעה לפקודת ה-**while**, היא בודקת את התנאי המופיע.
- אם התנאי מתקיים, מתבצע הקוד שבגוף הלולאה פעם אחת.
- כעת התוכנית **חוזרת לפקודת ה-while** ובודקת שוב את התנאי. אם התנאי מתקיים היא חוזרת על הקוד שבגוף הלולאה.
- התוכנית ממשיכה כך, עד שהתנאי מפסיק להתקיים, ואז התוכנית ממשיכה בקוד שמופיע לאחר הסוגר המסולסל **}**.

לולאות while - מעקב

- נעקוב אחר ביצוע הלולאה בקוד הבא:

```
int x = -2;
while (x < 0) {
   printf("x is still negative.\n");
   x = x + 1;
}
printf("x is no longer negative.\n");
```

x

0

מה יקרה אם נשמיט שורה זו ??

- כמה פעמים יתבצע גוף הלולאה עבור **x=-7** ? **x=7** ?

עוד לולאת while

- הנה הגרסה הידידותית יותר של אתר האינטרנט.
- מה קורה עתה אם המשתמש מקליד גיל צעיר מ-18 ?

```
double age;
printf("enter your age: ");
scanf("%lf", &age);

while (age < 18) {
  printf("Invalid age, please try again: ");
  scanf("%lf", &age);
}
printf("Welcome to C-hamazal.com! \n");
printf("<< How much money do you want ");
printf("to lose today?™ >> \n");
```

Run!