

# **מכללת הדסה, החוג למדעי המחשב**

## **מבוא לתכנות מונחה עצמים והנדסת תוכנה**

### **סמסטר א', תשפ"א**

#### **תרגיל 1**

תאריך אחרון להגשה:

הנביאים – יום א', 08/11/2020, בשעה 23:59

שטרاؤ גברים – יום ג', 03/11/2020, בשעה 23:59

שטראו נשים – יום ד', 04/11/2020, בשעה 23:59

#### **מטרת התרגיל:**

בתרגיל זה נתרגל את המונחים הבסיסיים בקורס, ובכללן מחלקות (classes), משתני מחלוקת ופונקציות מחלוקת (constructors) בנאים (data members and member functions) מצינית גישה (access modifiers), תוך התאמת למשחק (interface) שנקבע מראש.

נתונים קובץ פרויקט וקבצים עם מספר פונקציות בסיסיות. נתאים את הפרויקט לשימוש עבורנו (שינוי שם הפרויקט בהתאם להנחיות ההגשה) ונוסף קבצים עם המחלקות המוגדרות בהמשך. את הקבצים הנתונים אין לשנות בשום אופן, אלא אם צוין בפירוש אחר. לעומת זאת קובצי הקוד שקיבלנו, נגייש כפי שהם (למעט השינויים המותרים) כחלק מהתרגיל.

#### **תיאור כללי:**

בתרגיל נמשח אובייקטים פשוטים של צורות גיאומטריות. נמשח 4 אובייקטים: מלבן, משולש שווה שוקיים, משולש שווה צלעות, ומסגרת. בקבצים הנתונים קיימת תכנית (פונקציית main) שמשתמשת באובייקטים האלה ומציררת אותם על המסך.

#### **פירוט הדרישות:**

נכטו 4 מחלקות (classes):

1. – מלבן מקביל לצירים – Rectangle
  2. – משולש שווה שוקיים עם בסיס מקביל לציר ה-*x* IsoscelesTriangle
  3. – משולש שווה צלעות עם צלע מקבילה לציר ה-*x* EquilateralTriangle
  4. – מסגרת, כלומר מלבן בתוך מלבן (בצורה ממורצת), שניהם מקבילים לצירים Window
- המחלקות צרכות להיקרא בדיק בשמות הalfa וכל אחת צריכה להיות מוגדרת בקובץ "א." בשם תואם. בהמשך יוגדרו הבנאים (constructors) והפונקציות הציבוריות בבדיקה עבור כל מחלוקת. המשחק מוגדר בצורה מדיקת – אין להשמיט או להוסיף פונקציות ציבוריות (public). החתימות של הפונקציות הציבוריות צרכות להיות בדיק כפי שהן מופיעות בהגדרת התרגיל (למעט מקומות שבהם מופיעים בתרגיל מספרים מפורטים מטעמי נוחות, אבל בפועל בתרגיל מצופה

שתשתמשו בקבועים (const) שתגדירו). אין לחסוף משתנים פנימיים (data members), כלומר המשתנים חייבים להיות private. מותר להוסיף פונקציות עזר, אבל הן חייבות להיות או private או פונקציות חייזניות (בדרך כלל בקובץ אחר).

בכל מובן אחר, נוכל להחליט בעצמנו כיצד למש את המחלקות (בכפוף, כמובן, לכללי התכונות הנאות). لكن נוכל (ונצטרך) להחליט בין השאר:

- איזה מידע יש להחזיק בתוך האובייקטים?
- איך לייצג מידע זה? כלומר אלו data members יהיו ומה יהיה הטיפוס (type) שלהם?
- אלו פונקציות פרטיות (private) כדאי להוסיף?
- איזה שימוש חוזר (reuse) ניתן לעשות בכך שכבר כתבנו ולהימנע מכך?

## הערות וטיפים לגישה לתרגילים:

- כפי שהוזכר בתרגול, אין צורך להתעמק בכך הנטען. יש בו שימוש ביכולות רבות שעוד לא הכרנו (אבל נכיר בהמשך הקורס), וחבל לבזבז על כך את הזמן. אם נבנה את המחלקות לפי ההנחיות, לא אמורה להיות בעיה. אם יש בעיית קומפליציה, השגיאות עצמן אמורים לספק מספיק מידע מה הפונקציות שאלוי שכתבנו למש או שטעינו בחתימה שלהן וכדומה.
- כדאי להתחיל למש את המחלקות לפי הסדר. בפרט, כדאי להתחיל מ-`Rectangle`. זו מחלוקת פשוטה יותר, והיא נדרשת עבור המחלקות האחרות (למשל, כתיפוס ההחזרה `mpenalty()`).
- השתמש ב-`#define`-ים שבתחילה קובץ `main.cpp` כדי להגביל את התוכנית לשימוש רק במחלקות שכבר מישנו (או שאנו נור כדי לעבוד עליהם). אך, לא קיבל שגיאות קומפליציה לא רלוונטיות מחלוקת שעוד לא תחלמו לעבוד עליהם, וכל מחלוקת שנשים נוכל להריץ את התוכנית ולבדוק אותה.
- למרות שבפירות המשק מוזכרים קודם הבנאים ורק אחר כך הפונקציות השונות, מומלץ להתחיל למש מהפונקציות (הסתפויות עבור המחלוקת). התכנון של שימוש הפונקציות האלה יעזר לנו להבין מה `data members` שצריך לשמר בחלוקת לשם השימוש שלהן. אחר כך כבר יהיה יותר ברור מה לעשות בבנאי ואיך לאותל את `data members` בעזרה הארגומנטים שהבנאי מקבל.
- אחד הדברים שייחסור לנו הרבה בתרגיל זה הוא `reuse`. זה כולל שימוש חוזר במחלקות, שימוש חוזר בפונקציות ושימוש חוזר בבנאים (למשל, `c-tors` `delegated`). כשבנואו למש מחלוקת, נחשוב האם אפשר להיעזר באחת המחלקות האחרות למימוש שלה. כשבנואו למש פונקציה, נחשוב האם אפשר להיעזר בפונקציה אחרת של המחלוקת, בפונקציות של `data members` או להוציא קוד לפונקציית עזר (`private` או חייזנית) כדי להשתמש בו ממש פונקציות דומות. כשבנואו למש בנהי, נבדוק האם אפשר להיעזר באחד הבנאים הקיימים בחלוקת כדי למש אותו.

- חלק מה-*reuse*, נרוויח גם לפעמים חלק מהבדיקות שכתוב שציר לבעץ לא נצטרך לכתוב במפורש, כי הן כבר נבדקוות במבנה אחר או במחלקה אחרת.
- שימושו לב להעירות נוספות בסוף התרגיל, ולא לשכוח למש את הפונקציות הנדרשות עבור כל המחלקות (למשל, פונקציית (*draw*) שברשימה בסוף).

## תוכן הקבצים הנדרשים:

1. קבצי `CMakeLists.txt` המתאימים להגדרת פרויקט המשמש בשאר הקבצים הנדרשים.
2. קובץ `Vertex.h` שבו הגדרה של `struct` שנקרא `Vertex` המיציג קודקוד במשור עלי פי קואורדינטות (`y`, `x`). כדי למנוע הבלבול בחישובים בהמשך, אנחנו מיד מתרגמים את `x` ל-`col` (קיצור עבור `column`, `column` עמודה) ואת `y` ל-`row` (שורה), כדי להבהיר מה המשמעות של כל אחד מהם מבחינת המיקום על המסך.  
אפשר להוסיף לקובץ זהה פונקציות עזר לטיפול ב-`Vertex` (למשל, לשם השוואה בין שני אובייקטים מסווג `Vertex`, כדי שנוכל להשתמש בה במספר מקומות). כדי להוסיף פונקציות כאלה, נוסיף את ההצהרה על הפונקציה לקובץ זהה, `Vertex.h`, ואת המימוש נשים בקובץ `Vertex.cpp`, כפי שלמדנו על חלוקה לקבצים.
3. קובץ שנקרא `macros.h` ובו מוגדרים הקבועים `MAX_COL` ו-`MAX_ROW`. אלה ערכי ה-`col` וה-`row` המקסימליים שניתן לקבל. ערכי `col` האפשריים בתרגיל הם המספרים בין 0 ל-`MAX_COL` (כולל `MAX_COL`) וערך `row` האפשריים בתרגיל הם המספרים בין 0 ל-`MAX_ROW` (שוב, כולל המספר הזה). אם מתקבלים במבנה שנגדי בהמשך נתונים החורגים מהתחומים הללו, צריך להתעלם מהנתון הבועתי ולהשתמש במקומו בערך בירית החוקן, ואין צורך לבדוק אותו שוב. **נשים לב שלא להתייחס בקוד שלנו למספרים הספציפיים המופיעים בקובץ אלא לקבועים הנ"ל.** נזכיר שוב, הקובץ `macros.h` שנוצר בהגשת התרגיל חייב להיות זהה לזו שקיבלנו.
4. קובץ שנקרא `main.cpp`, המכיל תוכנית המקבלת נתונים מהמשמש ועל פיהם יוצרת אובייקטים מהמחלקות שנמצש, מנהלת לוח פנימי שעליו יצירו הצורות ולבסוף מדפסה אותו בחלון המסוף (`Terminal`, `Console`). למרות שציר להגיש את הקובץ זהה כפי שקיבלנו אותו, מומלץ לנטות את המחלקות שלנו גם עם פונקציית `main` משלהן, עם בדיקות נוספות לפונקציות נוספות ובנאים נוספים שאינם נבדקים על ידי ה-`main` הנוכחי. בכל מקרה, יבדק אבל מן הסתם הוא דומה לו. התוכנית נעודה להמבחן ולהציג את צורת השימוש באובייקטים שאתם צריכים למש. בתחילת התוכנית קיימים `#define`-ים: `RECTANGLE`, `WINDOW`, `ISOSCELESTRIANGLE`, `EQUILATERALTRIANGLE` ועודיקטים לבודק, כפי שהסביר בתרגול.

5. צמד קבצים בשמות `h` ו-`cpp` `Board.h` ו-`Board.cpp` המכיל את פונקציות הלוח הרכלוונטיות. נctruck להויסיף `include` לקובץ `h` במחלקות שלנו כדי שנוכל למש את פונקציית `draw` (`drawLine` המוגדרת עבורה `Board`).
6. צמד קבצים בשמות `h` ו-`cpp` `Utilities.h` ו-`Utilities.cpp` שיישמשו אותנו לפונקציות עזר כלליות למש פונקציות הלוח.

## **פירוט המשך:**

כפי שהזכירנו לעיל, נמש 4 מחלקות. כתת נפרט את חתימות הפונקציות הציבוריות שיש לספק עבור כל מחלקה.

### **עבור Rectangle:**

Constructor `Rectangle (const Vertex& bottomLeft, const Vertex& topRight)` – בנאי המקבל שני קדקודים התוחמים את המלבן. כפי שהשמות מרמזים, הראשון מצין את הקודקוד השמאלי-תחתון והשני – את הקודקוד הימני-עליון. כפי שהוזכר לעיל, יש לוודא שערכי ה-X של שני הקצוות נמצאים בתחום `[0,MAX_COL]` ושערכי ה-Y נמצאים בתחום `[0,ROW_MAX]`. אם אפילו אחד מהפרמטרים לא עונה על הקритריון זהה עליום לבנות `Rectangle` שקדקודיו כנ"ל הם `(0,10)` ו-`(30,20)`. כמו כן יש לבדוק האם הקודקוד השמאלי-תחתון אכן שמאלי-תחתון. נציג כי ניתן שהקדקודים יתלבדו באחת הקואורדינטות או יותר (מלבן "מנון" נחשב חוקן), אבל לא ניתן שהקודקוד השמאלי יהיה מימין לימי או שהעלון מתחת לתחתון. גם עבור הבדיקה הזאת, אם היא נכשלה, ניצור את המלבן לפי בירית המחדל כנ"ל.

Method `const Vertex vertices[2]` – אותו דבר בדיקת כמו הבניי הראשון, כולל הבדיקות ובירית המחדל במקרה שהוא נכשלות, אלא שמקבלים את הקדקודים במערך `[0]` הוא `vertices` והו הקודקוד השמאלי-תחתון ו-`[1]` הוא הימני-עליון.

Method `Rectangle (double x0, double y0, double x1, double y1)` – בונה `Rectangle` שבו הקודקוד השמאלי-תחתון הוא הנקודה `(y0, x0)` והקדקוד הימני-עליון הוא `(y1, x1)`. כמובן שגם פה צריך לעשות את אותן בדיקות כמו בבניים הקודמים ובירית המחדל כנ"ל.

Method `const Vertex& center, double width, double height` – בנאי המקבל את מרכז המלבן (קדקוד), רוחב וגובה ובונה על פיהם מלבן. גם כאן צריך לבדוק אם הקודקודים שתחשבו מטור הנתונים נמצאים בטוויה שהוגדר למעלה ואם לא – לפעול באותה צורה כמו בבניים הקודמים. בנוסף, הרוחב והגובה צריכים להיות לא שליליים.

Method `Vertex getBottomLeft() const` – מחזירה את הקודקוד השמאלי-תחתון של המלבן.  
Method `Vertex getTopRight() const` – מחזירה את הקודקוד הימני-עליון של המלבן.

– מחזירה את הרוחב של המלבן (אורך הצלע המקבילה לציר ה- $x$ ).  
double getWidth() const

– מחזירה את הגובה של המלבן (אורך הצלע המקבילה לציר ה- $y$ ).  
double getHeight() const

## עבור IsoscelesTriangle :

(const Vertex vertices[3]) – בניית משולש משלושה קודקודים. גם כאן, צריך לבדוק אם כל שלושת הקודקודים נמצאים בתחום החוק. בנוסף, צריך לוודא שני הקודקודים הראשונים ייצרים צלע המקבילה לציר ה- $x$ , והוא בסיס המשולש, וההמשולש שווה שוקיים (שתי הצלעות האחרות באורך זהה). אם אחד מהתנאים לא מתקיים – צריך ליצור משולש שלושת קודקודיו בנקודות (20,20), (30,20), (25,20+sqrt(75)).

(const Vertex& v0, const Vertex& v1, double height) – בניית משולש שני קודקודים (שאמורים לתאר את הבסיס של המשולש) וגובהה. לוודא שהקודקודים (אליה שקיבלו נזה שנחשב לפני הגובה) חוקיים ושகודקודים שקיבלו אכן מתארים צלע המקבילה לציר ה- $x$ . אם לא – ניצור משולש ברירת מחדל כפי שהוא במבנה הקודם. נשים לב שהגובה המתkeletal יכול להיות שלילי, ומהמשמעות היא שהקודקוד השלישי יהיה " מתחת" לבסיס, נמוך יותר, ככלומר המשולש יפנה כלפי מטה.

(int index) – פונקציה המחזירה את הקודקוד index (כאשר האינדקס הוא בין 0 ל-2, כרגע במערכות) מבין שלושת קודודי המשולש. (במהמשך הקורס נלמד איך למשת צאת כאופרטור [ ] כך שייהי אפשר לגשת אליהם כמו במערך רגיל, כלומר אם  $t$  הוא אובייקט מסוג IsoscelesTriangle, יוכל לגשת אליהם כ- $t[0], t[1], t[2]$ ). סדר הקודקודים יהיה לפי הסדר בו הם התקבלו במבנה (עבור הבנייה שמקבל רק שני קודקודים, השניים הראשונים הם אלה שהתקבלו במבנה, ולפי הסדר שבו התקבלו, והשלישי הוא זה שנחשב לפני הגובה שהתקבל).

– מחזירה את אורך הבסיס (הצלע המקבילה לציר ה- $x$ ).  
double getBaseLength() const

– מחזירה את אורך השוק במשולש.  
double getLegLength() const

– מחזירה את האורך של גובה המשולש, ככלומר אורך הקטע היוצא מהקודקוד המחבר את השוקיים (קודקוד הראש) ומאונך לבסיס (למעשה, הגובה לציר ה- $y$ ). נשים לב שהגדירה, כ舍םברים על אורך הוא תמיד לא-שלילי (בניגוד להגדרה שהגדכנו במבנה, שייתכן גובה שלילי, כדי לאפשר בניית מושלמים הפוכים).

## עבור EquilateralTriangle :

(const Vertex vertices[3]) – בניית משולש שווה צלעות משלושה קודקודים. כמובן שגם צריך לבדוק אם כל שלושת הקוד הודים נמצאים בתחום החוק. בנוסף, צריך לוודא שני הקודקודים הראשונים ייצרים צלע המקבילה לציר ה- $x$  ושלכל צלעות

המשולש שווה זו לו. אם לא – צריך ליצור מושלש שלושת קודקודיו בנקודות: (20,20), (20,20), (30,20).  
.(25,20+sqrt(75))

מודדרת באותו אופן כמו הפונקציה בשם זהה במחלקה Vertex getVertex (int index) const .IsoscelesTriangle

– מחזירה את אורך הצלע במשולש. double getLength() const

## עבור Window:

Window (const Rectangle& outer, const Rectangle& inner) – בניית מסגרת משני מלבים, חיצוני ופנימי. המלבנים שקיבלו הם בהכרח מלבים חוקיים (לא ניתן לבנות מלבן לא חוקי, הבנים שהגדרנו לאמורים לאפשר זאת). מה שנוטר לבדוק הוא שם ממורכבים (נקודות המרכז של שניהם זהה) והפנימי אכן פנימי (הוא יכול להתלכד עם החיצוני, מסגרת "מנוגנת", אבל לא להיות מחוץ לו, לא גדול יותר). אם אחת הבדיקות נכשלה, ניצור את המסגרת משני מלבים שקודקודיהם (20,10) ו-(30,20) (כלומר, שני המלבנים יתלכדו).

Window (const Rectangle& outer, double verticalThickness, double horizontalThickness) – המלבן הנתון מגדר את הגבול החיצוני של המסגרת, ושני הפרמטרים הנוספים מגדרים את עובי המסגרת בגובה (vertical) וברוחב (horizontal), כלומר המרחק בין המלבן החיצוני לפנימי. כאן נצטרך לבדוק שנתוני העובי חוקיים, כלומר הם לא יכולים להיות יותר ממחצית מהגובה או הרוחב (בהתקאה) של המלבן החיצוני, כי אז אי אפשר לבנות את המלבן הפנימי. גם כאן, במקרה של בעיה, ניצור את צורת ברירת המחדל.

– מחזירה את הקודקוד השמאלי-תחתון של המסגרת. Vertex getBottomLeft() const

– מחזירה את הקודקוד הימני-עליון של המסגרת. Vertex getTopRight() const

double getVerticalThickness() const – מחזירה את עובי המסגרת בגובה (ה מרחק בין הצלע העליונה של המלבן החיצוני לצלע העליונה של המלבן הפנימי, וכן בין הצלעות התחתונות של המלבנים).

double getHorizontalThickness() const – מחזירה את עובי המסגרת ברוחב (ה מרחק בין הצלע השמאלית של המלבן החיצוני לצלע השמאלית של המלבן הפנימי, וכן בין הצלעות הימניות של המלבנים).

## פונקציות שיהיו בכל אחת מהמחלקות:

void draw (Board& board) const – מצירת את הצורה על הלוח שהתקבל כפרמטר. כדי שהוזכר, השתמש בפונקציה () של מחלקת Board כדי לצייר את הצורה שלמו על הלוח. לדוגמה, אם החלטנו שחלוקת מציר הצלחה צריך למתווך קו בין שני קודקודים נתוניים, 1 ו-2, בתווך

הfonקציה ()draw נכתבת את השורה: board.drawLine(v1, v2); נפעיל את הפונקציה ()draw על האובייקט board שקיבלנו כפרמטר, ונעביר לה כפרמטרים את הקודקודים הרצויים).

const Rectangle getBoundingClientRect() – מחזירה את המלבן המקביל לציריהם (כפי שהגדכנו את Rectangle לעיל) החוסם את הצורה, כלומר המלבן הקטן ביותר שמקביל לציריהם ומכל את המלבן, המשולש שווה השוקיים, המשולש שווה הצלעות או המסגרת.

const double getArea() – מחזירה את שטח הצורה הכלול (במשולש שווה שוקיים או צלעות ניתן לחשב את הגובה בקלות, וממילא קל לחשב את השטח; במסגרת, נחשב את שטח המלבן החיצוני פחות שטח המלבן הפנימי).

const double getPerimeter() – מחזירה את היקף הצורה (במסגרת, נחשב את סכום היקפי שני המלבנים).

const Vertex getCenter() –מחזירה את מרכז הצורה (שמוגדר כאן כממוצע הקודקודים, גם במשולש שווה שוקיים).

bool scale (double factor) – מגדילה או מקטינה את מרחק כל הקודקודים ממרכז הצורה לפי הפקטור המועבר ביחס למרכז הצורה כפי שהוגדרה למטה, ומחזירה true. אם הצורה חורגת מהגבולות שהוגדרו (אחד הקודקודים יהיה עם ערכים החורגים ממה שהוגדר לעיל) – להשאר את הצורה ללא שינוי ולהחזיר false. באופן דומה, אם הפקטור אינו מספר חיובי, לא משנים כלום ומהציגים false.

לשם הפשטות, אנחנו מחשבים מרחק ב- $\sqrt{L}$ , כלומר מחשבים את המרחק בציר ה- $x$  ואת המרחק בציר ה- $y$  כל אחד בפני עצמו, ואת המרחק זהה מכפילים בפקטור הנutan, לא מרחק אוקלידי.

## הערות חשובות:

- על מנת להשוות בין שני ערכי double אין להשתמש באופרטור ==. זאת מאחר ורבה פעמים שני ערכים מתקבלים מчисובים שונים והם שוים בהتنסותם למספר הרצוי, אך בפועל כל אחד מיוצג מעט אחרת בגלל אילוצי מקום. (לדוגמא המספר 0.99999999 עלול להיות שווה למספר 1) לכן על מנת להשוות שני מספרים علينا להשתמש בחישוב ההפרש ולקבוע אפסיילון כלשהו כך שם ההפרש הוא קטן ממנה, יוכrazו המספרים כשיווים.
- בחלק מהfonקציות לעיל מוזכר const בסוףן. זו אינה טעות. כתובים את ה-const הזה אחרי הסוגרים של הפונקציה (הן בהצהרה והן ובימוש). בהמשך הקורס נבין את המשמעות שלו, כרגע מספיק שנדע שהקובד הנutan לא יתكمפל אם לא נעשה כך.
- כדי לקבל גישה ל-(`std::sqrt`) ועוד פונקציות שימושיות (למשל, (`std::abs`)) יכולה להיות שימושית כאן), נעשה include ל-`cmath`.

## קובץ ה-README:

יש לכלול קובץ README שיקרא README.doc, README.docx או README.txt (ולא בשם אחר). הקובץ יכול להכתב בעברית ובלבד שיכיל את הסעיפים הנדרשים.

קובץ זה יכול לכל הפחות:

1. כותרת.
2. פרטי הסטודנט: שם מלא כפי שהוא מופיע ברשימות המכללה, ת"ז.
3. הסבר כללי של התרגיל.
4. רשימה של הקבצים שיצרנו, עם הסבר קצר (לרוב לא יותר משורה או שתיים) לגבי תפקידם.
5. מבני נתונים עיקריים ותפקידיהם.
6. אלגוריתמים הרואים לציון.
- 7.אגדים ידועים.
8. הערות אחרות.

יש לתמצת כל שנייתן אך לא יותר על אף חלק. אם אין מה להגיד בנושא מסוים יש להשאיר את הכותרת ומתחתיו פסקה ריקה. כתוב ב-README כל דבר שרצוי שהבודק ידע כשהוא בודק את התרגיל.

## אוף ההגשה:

הקובץ להגשה: יש לדוחס כל קובץ הקשור לתרגום, למעט מה שיצין להלן, לקובץ שמו firstname\_lastname.zip, כאשר N הוא מספר התרגום ו-exN firstname\_lastname.zip המלא (לדוגמא אלברט איינשטיין יgive כרך הראשון ex1\_albert\_einstein.zip). במקרה של גישה בזוג, שם הקובץ יהיה לפי התבנית exN\_firstname1\_lastname1\_firstname2\_lastname2.zip, עם שמות המציגים בהתאם (ללא רווחים; למשל, גם בשמות עצם יש להחליף רווחים בכו תחתי, כפי המודגם לעיל). כמו כן, במקרה של גישה בזוג, רק אחד מהציגים יgive את הקובץ ולא שנייהם.

לפני דחיסת תיקיית הפרויקט שלכם יש למחוק את הפריטים הבאים:

- תיוקיה בשם out, אם קיימת
- תיוקיה בשם sv.

שתי התיקיות האלה נמצאות בתיקייה הראשית (זו שנחננו פותחים בעזרת VS). התיקייה sv. לפעמים מסוימת, אבל אם תפתחו את קובץ ה-koz שיצרתם, בודאי תוכלו למצוא אותה ולמחוק אותה.

זכרו שגם את הקבצים שננתנו לכם צריכים לצרף, והם צריכים להיות ללא שינוי, כפי שקיבלתם אותם.

ככלל אצבע, אם קובץ ה-.kōz שוקל יותר ממ"ב אחד או שניים, כנראה שלא מוחקתם חלק מהקבצים הבינאריים המוזכרים.

וודאו כי קובץ ה-.kōz מכיל תקיה ראשית אחת, ורק בתוכה יהיו כל הקבצים ותתי התקיות של הפרויקט.

את הקובץ יש להעלות ל-Moodle של הקורס למשימה המתאימה.

**הגשה חוזרת:** אם מסיבה כלשהי סטודנט מחייב להגיש הגשה חוזרת יש לוודא שם הקובץ זהה לחנותן לשם הקובץ המקורי. אחרת, אין לבדוק אחראי לבדוק את הקובץ האחרון שיוגש.

כל שינוי ממה שמוגדר פה לגבי צורת ההגשה ומבנה ה-README עלול לגרום הורדת נקודות בציון.

מספר הערות:

1. נשים לב לשם הקובץ שכן יכולות שמות המגייסים.
2. נשים לב לשלוח את תיikit הפרויקט כולה, לא רק את קובצי הקוד שהוספנו. תרגיל שלא יכול את כל הקבצים הנדרשים, לא יתקבל וידרוש הגשה חוזרת (עם כללי האיחור הרגילים).

המלצת כללית: אחרי הכנת הקובץ להגשה, נעתיק אותו לתיקייה חדשה, נחלץ את הקבצים שבתוכו ונבדוק אם ניתן לפתח את התקיה הזו ולקמפל את הקוד. הרבה טעויות של שכחת קבצים יכולות להימנע על ידי בדיקה זו.

**בהצלחה!**