מכללת הדסה, החוג למדעי המחשב
מבוא לתכנות מונחה עצמים והנדסת תוכנה
סמסטר א', תשפ"ג
תרגיל 1

תאריך אחרון להגשה:

הנביאים – יום א', 13/11/2022, בשעה 23:59

שטראוס גברים – יום א', 13/11/2022, בשעה 23:59

שטראוס נשים – מוצאי שבת, 12/11/2022, בשעה 23:59

מטרת התרגיל:

בתרגיל זה נתרגל את המונחים הבסיסיים בקורס, ובכללם מחלקות (classes), משתני מחלקה ופונקציות מחלקה (data members and member functions) בנאים (constructors) מצייני גישה (access modifiers: public, private), תוך התאמה לממשק (interface) שנקבע מראש.

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

תיאור כללי:

בתרגיל נממש אובייקטים פשוטים של צורות גיאומטריות. נממש 4 אובייקטים: מלבן, משולש (שווה שוקיים), מעוין וחלון. בקבצים הנתונים קיימת תכנית (פונקציית main) שמשתמשת באובייקטים האלה ומציירת אותם על המסך.

פירוט הדרישות:

נכתוב 4 מחלקות (classes):

1. Rectangle – מלבן מקביל לצירים
2. IsoscelesTriangle – משולש שווה שוקיים עם צלע מקבילה לציר ה־x
3. Diamond – מעוין המורכב משני משולשים שווי שוקיים זהים
4. Window – "חלון", כלומר מלבן המחולק פנימית לארבעה מרובעים (לא בהכרח שווים בגודלם)

המחלקות צריכות להיקרא בדיוק בשמות האלה וכל אחת צריכה להיות מוגדרת בקובץ ".h" בשם תואם. בהמשך יוגדרו הבנאים (constructors) והפונקציות הציבוריות במדויק עבור כל מחלקה. הממשק מוגדר בצורה מדויקת – **אין להשמיט או להוסיף פונקציות ציבוריות (public). החתימות של הפונקציות הציבוריות צריכות להיות בדיוק כפי שהן מופיעות בהגדרת התרגיל** (למעט מקומות שבהם מופיעים בתרגיל מספרים מפורשים מטעמי נוחות, אבל בפועל בתרגיל מצופה שתשתמשו בקבועים (const) שתגדירו). **אין לחשוף משתנים פנימיים (data members)**, כלומר המשתנים חייבים להיות private. מותר להוסיף פונקציות עזר, אבל הן חייבות להיות או private או פונקציות חיצוניות (בדרך כלל בקובץ אחר).

בכל מובן אחר, נוכל להחליט בעצמנו כיצד לממש את המחלקות (בכפוף, כמובן, לכללי התכנות הנאות). לכן 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 בקוד שכבר כתבנו ולהימנע מכפל קוד?

הערות וטיפים לגישה לתרגיל:

- כפי שהוזכר בתרגול, אין צורך להתעמק בקוד הנתון. 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 קומפילציה, השגיאות עצמן אמורות לספק מספיק מידע מה הפונקציות שאולי שכחנו לממש או שטעינו בחתימה שלהן וכדומה.
- כדאי להתחיל לממש את המחלקות לפי הסדר. בפרט, כדאי להתחיל מ־Rectangle. זו מחלקה פשוטה יותר, והיא נדרשת עבור המחלקות האחרות (למשל, כטיפוס ההחזרה מפונקציית ()getBoundingRectangle).
- נשתמש ב־#define־ים שבתחילת קובץ main.cpp כדי להגביל את התוכנית לשימוש רק במחלקות שכבר מימשנו (או שאנחנו תוך כדי עבודה עליהן). כך, לא נקבל שגיאות קומפילציה לא רלוונטיות ממחלקות שעוד לא התחלנו לעבוד עליהן, וכל מחלקה שנסיים נוכל להריץ את התוכנית ולבדוק אותה.
- למרות שבפירוט הממשק מוזכרים קודם הבנאים ורק אחר כך הפונקציות השונות, מומלץ להתחיל לממש מהפונקציות (הספציפיות עבור המחלקה). התכנון של מימוש הפונקציות האלה יעזור לנו להבין מה ה־data members שצריך לשמור במחלקה לשם המימוש שלהן. אחר כך כבר יהיה יותר ברור מה לעשות בבנאי ואיך לאתחל את ה־data members בעזרת הארגומנטים שהבנאי מקבל.
- אחד הדברים שיחסוך לנו הרבה קוד בתרגיל הזה הוא reuse. זה כולל שימוש חוזר במחלקות, שימוש חוזר בפונקציות ושימוש חוזר בבנאים (למשל, delegated c-tors). כשנבוא לממש מחלקה, נחשוב האם אפשר להיעזר באחת המחלקות האחרות למימוש

שלה. כשנבוא לממש פונקציה, נחשוב האם אפשר להיעזר בפונקציה אחרת של המחלקה, בפונקציות של ה־data members או להוציא קוד לפונקציית עזר (private או חיצונית) כדי להשתמש בו ממספר פונקציות דומות. כשנבוא לממש בנאי, נבדוק האם אפשר להיעזר באחד הבנאים הקיימים במחלקה כדי לממש אותו.

- כחלק מה־reuse, נרוויח גם לפעמים שחלק מהבדיקות שכתוב שצריך לבצע לא נצטרך לכתוב במפורש, כי הן כבר נבדקות בבנאי אחר או במחלקה אחרת.
- שימו לב להערות נוספות בסוף התרגיל, ולא לשכוח לממש את הפונקציות הנדרשות עבור כל המחלקות (למשל, פונקציית ()draw) שברשימה בסוף.
- בחלק מחתימות הפונקציות המוגדרות בהמשך מוזכר const בסופן. זו איננה טעות. כותבים את ה־const הזה אחרי הסוגריים של הפונקציה (הן בהצהרה והן ובמימוש). בהמשך הקורס נבין את המשמעות שלו, כרגע מספיק שנדע שהקוד הנתון לא יתקמפל אם לא נעשה כך.
- על מנת להשוות בין שני ערכי double אין להשתמש באופרטור ==. זאת מאחר והרבה פעמים שני ערכים מתקבלים מחישובים שונים והם שווים בהתכנסותם למספר הרצוי, אך בפועל כל אחד מיוצג מעט אחרת בגלל אילוצי מקום. (לדומא המספר 0.9999999 עלול להיות שווה למספר 1) לכן על מנת להשוות שני מספרים נחשב את ההפרש ביניהם, ואם ההפרש הזה קטן מאפסילון כלשהו נתייחס אליהם כשווים זה לזה. (במקרה שלנו, להגדיר את 0.5 כאפסילון הזה יספיק ברוב המקרים, כי בכל מקרה לצורך הציור המספרים מעוגלים לשלם הקרוב.)
- כדי לקבל גישה ל־()std::sqrt ועוד פונקציות שימושיות (למשל, ()std::abs יכולה להיות שימושית כאן), נעשה include ל־cmath.

תוכן הקבצים הנתונים:

1. קבצי CMakeLists.txt המתאימים להגדרת פרויקט המשתמש בשאר הקבצים הנתונים.

2. קובץ Vertex.h שבו הגדרה של struct שנקרא Vertex המייצג קודקוד במישור על פי קואורדינטות (x, y). כדי למנוע בלבול בחישובים בהמשך, אנחנו מיד מתרגמים את x ל־col (קיצור עבור column, כלומר עמודה) ואת y ל־row (שורה), כדי להבהיר מה המשמעות של כל אחד מהם מבחינת המיקום על המסך.

   מותר להוסיף לקובץ הזה פונקציות עזר לטיפול ב־Vertex (למשל, לשם השוואה בין שני אובייקטים מסוג Vertex, כדי שנוכל להשתמש בה במספר מקומות). כדי להוסיף פונקציות כאלה, נוסיף את ההצהרה על הפונקציה לקובץ הזה, 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 ה־col האפשריים בתרגיל הם המספרים בין 0 ל־MAX_COL (כולל MAX_COL) וערכי ה־row האפשריים בתרגיל הם המספרים בין 0

ל־MAX_ROW (שוב, כולל המספר הזה). אם מתקבלים בבנאים שנגדיר בהמשך נתונים החורגים מהתחומים הללו, צריך להתעלם מהנתון הבעייתי ולהשתמש במקומו בערכי ברירת מחדל המוגדרים בהמשך לכל מחלקה. ערכי ברירת המחדל הללו נמצאים, כמובן, בתחום החוקי, ואין צורך לבדוק אותם שוב. **נשים לב שלא להתייחס בקוד שלנו למספרים הספציפיים המופיעים בקובץ אלא לקבועים הנ"ל**. נזכיר שוב, הקובץ macros.h שנצרף בהגשת התרגיל חייב להיות זהה לזה שקיבלנו.

4. קובץ שנקרא main.cpp, המכיל תכנית המקבלת נתונים מהמשתמש ועל פיהם יוצרת אובייקטים מהמחלקות שנממש, מנהלת לוח פנימי שעליו יצוירו הצורות ולבסוף מדפיסה אותו בחלון המסוף (Terminal, Console). למרות שצריך להגיש את הקובץ הזה כפי שקיבלנו אותו, מומלץ לנסות את המחלקות שלנו גם עם פונקציית main משלנו, עם בדיקות נוספות לפונקציות נוספות ובנאים נוספים שאינם נבדקים על ידי ה־main הנתון. בכל מקרה, להגשה נצרף את הקובץ המקורי. הקובץ הזה הוא לא בהכרח ה־tester שאתו התרגיל ייבדק אבל מן הסתם הוא דומה לו. התוכנית נועדה להמחיש ולהדגים את צורת השימוש באובייקטים שאתם צריכים לממש. בתחילת התוכנית קיימים define־ים: RECTANGLE, DIAMOND, ISOSCELESTRIANGLE, ו־WINDOW הקובעים אילו אובייקטים לבדוק, כפי שהוסבר בתרגול.
5. צמד קבצים בשמות Board.h ו־Board.cpp המכיל את פונקציות הלוח הרלוונטיות. נצטרך להוסיף include לקובץ Board.h במחלקות שלנו כדי שנוכל לממש את פונקציית ה־()draw שצריך לממש בכל מחלקה, על ידי שימוש בפונקציית ()drawLine המוגדרת עבור Board.
6. צמד קבצים בשמות Utilities.h ו־Utilities.cpp שישמשו אותנו לפונקציות עזר כלליות למימוש פונקציות הלוח.

פירוט הממשק:

כפי שהזכרנו לעיל, נממש 4 מחלקות. כעת נפרט את חתימות הפונקציות הציבוריות שיש לספק עבור כל מחלקה.

עבור Rectangle:

Rectangle (const Vertex& bottomLeft, const Vertex& topRight) – בנאי המקבל שני קדקודים התוחמים את המלבן. כפי שהשמות מרמזים, הראשון מציין את הקודקוד השמאלי־תחתון והשני – את הקדקוד הימני־עליון. כפי שהוזכר לעיל, יש לוודא שערכי ה־X של שני הקצוות נמצאים בתחום [MAX_COL,0] ושערכי ה־Y נמצאים בתחום [MAX_ROW,0]. אם אפילו **אחד מהפרמטרים** לא עונה על הקריטריון הזה עליכם לבנות Rectangle שקדקודיו כנ"ל הם (20,10) ו־(30,20). כמו כן יש לבדוק האם הקדקוד השמאלי־תחתון אכן שמאלי־תחתון. נדגיש כי ייתכן שהקדקודים יתלכדו באחת הקואורדינטות או יותר (מלבן "מנוון" נחשב חוקי), אבל לא ייתכן

שהקודקוד השמאלי יהיה מימין לימני או שהעליון מתחת לתחתון. גם עבור הבדיקה הזו, אם היא נכשלת, ניצור את המלבן לפי ברירת המחדל כנ"ל.

Rectangle (const Vertex vertices[2]) – אותו דבר בדיוק כמו הבנאי הראשון, כולל הבדיקות וברירת המחדל במקרה שהן נכשלות, אלא שמקבלים את הקדקודים במערך (vertices[0] הוא הקדקוד השמאלי־תחתון ו־vertices[1] הוא הימני־עליון).

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

Rectangle (const Vertex& start, double width, double height) – בנאי המקבל קודקוד שמאלי־תחתון, רוחב וגובה ובונה על פיהם מלבן. גם כאן צריך לבדוק אם הקודקודים (זה שהתקבל כפרמטר וזה שמחושב מתוך הנתונים) נמצאים בטווח שהוגדר למעלה ואם לא – לפעול באותה צורה כמו בבנאים הקודמים. בנוסף, הרוחב והגובה צריכים להיות לא שליליים.

Vertex getBottomLeft() const – מחזירה את הקודקוד השמאלי־תחתון של המלבן.

Vertex getTopRight() const – מחזירה את הקודקוד הימני־עליון של המלבן.

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

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

עבור IsosscelesTriangle:

IsosscelesTriangle (const Vertex vertices[3]) – בניית משולש שווה שוקיים משלושה קודקודים. גם כאן, צריך לבדוק אם כל שלושת הקודקודים נמצאים בתחום החוקי. בנוסף, צריך לוודא שאכן קיבלנו משולש כנדרש, כלומר שצלע הבסיס שלו מקבילה לציר ה־X ושתי הצלעות האחרות שוות זו לזו. אם לא – צריך ליצור משולש ששלושת קודקודיו בנקודות (20,20), (25,25), (30,20).
ניתן להניח שאנו מקבלים את הקודקודים משמאל לימין (על פי ערכי ה־X שלהם), כלומר הקודקוד השמאלי, המרכזי והימני.

IsosscelesTriangle (const Vertex& left, const Vertex& right, double height) – בניית משולש מקודקודי הבסיס וגובה המשולש. בהינתן ערכים חיוביים לארגומנט הגובה יפנה המשולש כלפי מעלה 🔺 , ואילו בהינתן ערכים שליליים יפנה המשולש כלפי מטה 🔻 . גם כאן, צריך לבדוק אם כל שלושת הקודקודים נמצאים בתחום החוקי. בנוסף, צריך לוודא שאכן קיבלנו משולש כנדרש, כלומר שיש לו צלע מקבילה לציר ה-X ושהוא שווה שוקיים. אם לא – צריך ליצור משולש ברירת מחדל כפי שראינו בבנאי הקודם.

שימו לב שאם הנחתם בבנאי הקודם את הסדר משמאל לימין, עליכם לוודא גם בבנאי הזה שאתם שומרים על הסדר הזה (כלומר, הקודקוד שקיבלתם יהיה השני מבין השלושה).

Vertex getVertex (int index) const – פונקציה המחזירה את הקדקוד ה-index (כאשר האינדקס הוא בין 0 ל-2, כרגיל במערכים) מבין שלושת קודקודי המשולש. (בהמשך הקורס נלמד איך לממש זאת כאופרטור [ ] כך שיהיה אפשר לגשת אליהם כמו במערך רגיל: Triangle[0], Triangle[1], Triangle[2].) סדר הקודקודים הוא לפי שיעורי ה־X שלהם (כלומר, בסדר שבו קיבלתם אותם, אם הנחתם לעיל תקינות...)

double getBaseLength() const – מחזירה את אורך צלע הבסיס של המשולש.

double getLegLength() const – מחזירה את אורך השוק של המשולש.

double getHeight() const – מחזירה את הגובה של המשולש מהבסיס לקודקוד שבו מחוברים השוקיים השווים, כלומר הגובה בציר ה־Y.

עבור Diamond:

Diamond(const Vertex vertices[4]) – בניית מעוין שאלה הקודקודים שלו, הקודקודים ינתנו על פי כיוון השעון, מתחיל מהקודקוד השמאלי. צריך לבדוק שהקודקודים חוקיים (לא חורגים מהלוח) ושהם יוצרים מעוין חוקי, זאת אומרת שהקודקוד התחתי יש לו את אותו שיעור x כמו הקודקוד העליון וכן שהקודקוד השמאלי יש לו את אותו שיעור y כמו לקודקוד הימני. בנוסף נוודא שיש לנו מעויין ולא דלתון. אם הערכים לא חוקיים, נבנה מעוין שקדקודיו הם (20,20), (25,25), (30,20) (25,15).

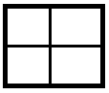
Diamond(const IsoscelesTriangle & lower) – בניית מעוין שזה יהיה המשולש התחתון שלו. המשולש העליון הוא תמונת המראה שלו, כלומר הצלע המקבילה לציר ה־x משותפת והקודקודים מרוחקים אחד מהשני. כעת נצטרך לבדוק האם אכן ניתן לבנות את המשולש העליון (או שהוא חורג מהלוח) אבל אין טעם לבדוק את ההתאמה ביניהם. גם כאן, במקרה של בעיה, ניצור את צורת ברירת המחדל.

Vertex getVertex (int index) const – פונקציה המחזירה את הקדקוד ה-index (כאשר האינדקס הוא בין 0 ל-3, כרגיל במערכים) מבין ארבעת קודקודי המעוין. סדר הקודקודים הוא מהנקודה השמאלית בכיוון השעון. (כלומר, בסדר שבו קיבלתם אותם, אם הנחתם לעיל תקינות...)

double getWidth() const – מחזירה את רוחב המעוין.

double getHeight() const – מחזירה את גובה המעוין.

עבור Window:

Window(const Rectangle& rectangle, const Vertex& point) – בניית חלון ממלבן ונקודה בתוכו. המלבן הוא מסגרת החלון. הנקודה מציינת את המיקום בחלון שמשמש לחלוקת החלון לארבעה חלקים, כך שנקבל צורה בסגנון  (החלקים לא בהכרח שווים). המלבן שקיבלנו הוא בהכרח מלבן חוקי (לא ניתן לבנות מלבן לא חוקי, הבנאים שהגדרנו לא אמורים לאפשר זאת). מה שנותר לבדוק הוא שהנקודה לא נמצאת מחוץ למלבן הזה. נדגיש כי ייתכן שהנקודה מתלכדת עם אחת הצלעות או אפילו אחד הקודקודים. במקרה כזה, כשנצייר את החלון, אחד מהקווים הפנימיים או שניהם לא ייראו, וזה בסדר. במקרה של בעיה בחוקיות, ניצור את החלון ממלבן שקודקודיו (20,10) ו־(30,20) ומחולק בנקודה (25,15) (בדיוק במרכז).

Window(const Vertex& start, double width, double height, const Vertex& point) – בניית חלון ממלבן המוגדר בעזרת הקודקוד start, הרוחב והגובה (כפי שהוגדר בבנאי המתאים לעיל בהגדרת המלבן) ונקודה בתוכו (כפי שהוגדר בבנאי הקודם). ברור שצריך שהמלבן יהיה חוקי (כפי הכללים עבור מלבן) ושהנקודה תהיה בתוכו (כפי שהוגדר בבנאי הקודם). גם כאן, במקרה כישלון, ניצור את צורת ברירת המחדל.

Vertex getBottomLeft() const – מחזירה את הקודקוד השמאלי־תחתון של החלון.

Vertex getTopRight() const – מחזירה את הקודקוד הימני־עליון של החלון.

Vertex getPoint() const – מחזירה את נקודת החלוקה של החלון.

פונקציות שיהיו בכל אחת מהמחלקות:

void draw (Board& board) const – מציירת את הצורה על הלוח שהתקבל כפרמטר. כפי שהוזכר, נשתמש בפונקציה drawLine() של מחלקת Board כדי לצייר את הצורה שלנו על הלוח. לדוגמה, אם החלטנו שכחלק מציור הצורה צריך למתוח קו בין שני קודקודים נתונים, v1 ו־v2, בתוך הפונקציה draw() נכתוב את השורה: board.drawLine(v1, v2); (כלומר, נפעיל את הפונקציה drawLine() על האובייקט board שקיבלנו כפרמטר, ונעביר לה כפרמטרים את הקודקודים הרצויים). שימו לב, במעויין, אפשר ואפילו רצוי, שיופיע קו אופקי בין שתי הנקודות הימנית והשמאלית.

Rectangle getBoundingRectangle() const – מחזירה את המלבן המקביל לצירים (כפי שהגדרנו את Rectangle לעיל) החוסם את הצורה, כלומר המלבן הקטן ביותר שמקביל לצירים ומכיל את המלבן, המשולש שווה השוקיים, המעוין או החלון.

double getArea() const – מחזירה את שטח הצורה הכולל (במשולש כבר נדרשנו לחשב את הגובה, וממילא קל לחשב את השטח; במעוין, כמובן שהכוונה לסכום שטח המשולשים המרכיבים אותו).

double getPerimeter() const – מחזירה את היקף הצורה.

Vertex getCenter() const – מחזירה את מרכז הצורה (שמוגדר כאן כממוצע הקודקודים).

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

שימו לב שהחישוב די פשוט כשמחשבים אותו רכיב-רכיב, כלומר מחשבים את המרחק (בין הקודקוד למרכז הצורה) בציר ה־x ואת המרחק בציר ה־y כל אחד בפני עצמו, את המרחק הזה מכפילים בפקטור הנתון ואת הקודקוד ממקמים מחדש לפי המרחק החדש ביחס למרכז הצורה. לדוגמה, אם יש לנו צורה שיש לה קודקוד במיקום (5,6) והמרכז שלה במיקום (7,9) וקיבלנו את הערך 2 כפקטור, נחשב את המרחק ב־x ונקבל 2, אחרי הכפלה בפקטור נקבל 4, ולכן רכיב ה־x של הקודקוד החדש יהיה 3. באופן דומה, עבור רכיב ה־y נקבל שהמרחק הוא 3, אחרי ההכפלה נקבל 6, והתוצאה היא 3. לסיכום, הקודקוד החדש יהיה (3,3).

קובץ ה־README:

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

קובץ זה יכיל 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 תפקיד הקובץ.
5. מבני נתונים עיקריים ותפקידיהם.
6. אלגוריתמים הראויים לציון.
7. באגים ידועים.
8. הערות אחרות.

יש לתמצת ככל שניתן אך לא לוותר על אף חלק. אם אין מה להגיד בנושא מסוים יש להשאיר את הכותרת ומתחתיו פסקה ריקה. **נכתוב ב־README כל דבר שרצוי שהבודק ידע כשהוא בודק את התרגיל.**

אופן ההגשה:

הקובץ להגשה: ניתן ליצור בקלות קובץ zip המותאם להגדרות ההגשה המפורטות להלן ישירות מתוך VS, כפי המוסבר תחת הכותרת "יצירת קובץ ZIP להגשה או לגיבוי" בקובץ "הנחיות לשימוש ב־Visual Studio 2022". אנא השתמשו בדרך זו (אחרי שהגדרתם כראוי את שמות הצוות ב־MY_AUTHORS) וכך תקבלו אוטומטית קובץ zip המותאם להוראות, בלי טעויות שיגררו אחר כך בעיות בבדיקה.

באופן כללי, הדרישה היא ליצור קובץ zip בשם exN-firstname_lastname.zip (או במקרה של הגשה בזוג – exN-firstname1_lastname1-firstname2_lastname2.zip), כשהקובץ כולל את כל קובצי הפרויקט, למעט תיקיות out ו־vs. כל הקבצים יהיו בתוך תיקייה ראשית אחת.

את הקובץ יש להעלות ל־Moodle של הקורס למשימה המתאימה. בכל מקרה, **רק אחד** מהמגישים יגיש את הקובץ ולא שניהם.

הגשה חוזרת: אם מסיבה כלשהי החלטתם להגיש הגשה חוזרת יש לוודא ששם הקובץ זהה לחלוטין לשם הקובץ המקורי. אחרת, אין הבודק אחראי לבדוק את הקובץ האחרון שיוגש.

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

מספר הערות:

1. נשים לב לשם הקובץ שאכן יכלול את שמות המגישים.
2. נשים לב לשלוח את תיקיית הפרוייקט כולה, לא רק את קובצי הקוד שהוספנו. תרגיל שלא יכלול את כל הקבצים הנדרשים, לא יתקבל וידרוש הגשה חוזרת (עם כללי האיחור הרגילים).

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

<u>בהצלחה!</u>