236703 – תכנות מונחה עצמים, חורף 2018-2019

תרגיל 5: Advanced C++

כללי

1. **מועד הגשה: 23/1/2019 בשעה 23:59.**
2. קראו היטב את ההוראות במסמך זה ובקוד שניתן לכם.
3. אחראי על התרגיל: **אריק**. שאלות על התרגיל יש לשלוח למייל eric.oop.course@gmail.com עם הנושא: "236703 HW5" .
4. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם. כמו כן הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שמימשתם כבר.
5. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
6. **הגשה בזוגות בלבד.**
7. **בכדי להימנע מטעויות ,אנא עיינו ברשימת ה-FAQ המתפרסמת באתר באופן שוטף.**

מבוא

בתרגיל זה נתרגל את האפשרויות המתקדמות של C++. התרגיל מורכב משני חלקים **בלתי תלויים** המתמודדים עם נושאים שונים. בחלק הראשון נבנה ספריית טיפול במטריצות ש"תרוץ" בזמן **קומפילציה** בעזרת מנגנון התבניות (בפרט Template Meta Programming) של השפה, ובחלק השני נממש מערכת Events שמיישמת את ה- Observer pattern.

חלק א' – Template Meta Programming

מבוא:

כפי שראיתם בתרגול, מערכת ה-Templates ב-C++ היא **טיורינג שלמה**. למעשה, זה אומר שניתן לממש בעזרתה כל תכנית מחשב, כבר בזמן הקומפילציה. בחלק זה נדגים תכונה זו על ידי כתיבת ספרייה מתמטית למטריצות.

מטרת חלק זה היא להכיר לעומק את מערכת התבניות ולהכיר את הכוח הרב הטמון בה. בחלק זה נשתמש בין היתר ב-**Template Specialization/Partial Specialization**, במילה השמורה **`typename`**, באופרטור **`sizeof…`** ובמאקרו **`static_assert`**.

מומלץ מאוד לקרוא את כל הדרישות והטיפים לפני שמתחילים לחשוב על פתרון.

איך מכשילים את הקומפילציה?

בחלק זה של התרגיל נטפל בשגיאות קלט (למשל חיבור 2 מטריצות מגודל לא זהה או מכפלה לא מוגדרת) על ידי הכשלת תהליך הקומפילציה.

C++11 הוסיפה לסטנדרט של השפה את המאקרו **static_assert** שפעולתו זהה לפעולת המאקרו assert, אך עובד **בזמן קומפילציה**. ניתן להשתמש במאקרו בכל חלק בקוד ובעזרתו נוכל לעצור את הקומפילציה במידת הצורך.

הגדרת המאקרו היא static_assert(bool_constexpr, message). אם bool_constexpr משוערך בזמן קומפילציה לערך **false** הקומפילציה תיכשל וההודעה message תוצג.

דוגמת שימוש (פשוטה אך שימושית):

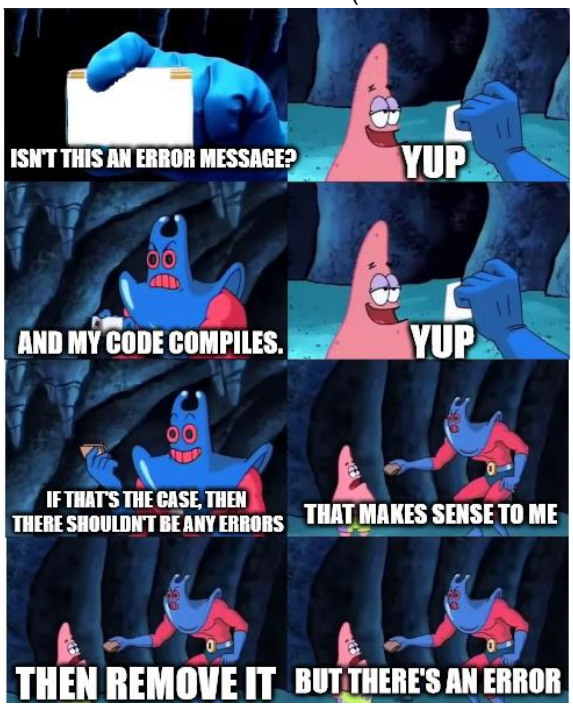
```
int main(){
    static_assert(sizeof(long) == 8, "Sorry, can't use non 64bit longs!");
    // rest of code
}
```

בכל מקום בחלק זה של התרגיל בו צריך להכשיל את הקומפילציה יש להשתמש במאקרו. **הודעת השגיאה לא תיבדק ולכן אפשר לבחור הודעה אקראית (לצרכי דיבוג, מומלץ לתת הודעה אינפורמטיבית!)**

הערה חשובה לפני שמתחילים

ב-IDE-ים רבים (אם לא בכולם) ה-Linter (הרכיב התוכנתי שאחראי על סימון שגיאות בקוד בזמן כתיבת הקוד, לפני ניסיון קומפילציה) לא מסוגל להתמודד עם מערכת התבניות של C++ בשימוש שהוא לא סטנדרטי, מכיוון שהוא אינו יכול לסמלץ את כל המנגנונים של התבניות בזמן הריצה שלו מבלי לקמפל את התכנית. לכן במקרים רבים הם יציגו לכם שגיאות שקשורות לתבניות, **גם אם הקוד מצליח לעבור קומפילציה**.

לכן, על מנת למנוע בלבול ועצבים (ושעות של דיבוג מיותר), מומלץ לעבוד בחלק זה עם עורך טקסט שלא מריץ Linter כברירת מחדל (כדוגמת Atom או Visual Studio Code).



לשם ההתחלה, נגדיר מס' מבני עזר בשביל מימוש המערכת (**כולם בקובץ Utilities.h**):

List:

המבנה הראשון שנגדיר הוא רשימה מקושרת של **איברים גנריים (רשימה טמפלייטית)**.
הרשימה תקבל טיפוסים בתור איבריה (ולא ערכים) ותכיל את השדות הבאים:

- **head** – איבר הראש של הרשימה
- **next** – טיפוס של רשימה מקושרת המכילה את שאר איברי הרשימה
- **size** – שדה מטיפוס int שיכיל את גודל הרשימה

דוגמת שימוש ברשימה: (הניחו שהטיפוס Int<N> הוגדר במקום אחר)

```
typedef List<Int<1>, Int<2>, Int<3>> list;
list::head; // = Int<1>
typedef typename list::next listTail; // = List<Int<2>, Int<3>>
list::size; // = 3
```

שימו לב: next צריך להיות מוגדר לכל רשימה לא ריקה (לפחות באורך אחד). רשימה ריקה לא צריכה להגדיר next.

בנוסף להגדרת הרשימה עצמה נגדיר מספר מבני עזר לרשימה:

- **PrependList** – מבנה שיקבל **טיפוס** ו**רשימה** (בסדר הזה) ויכיל את הטיפוס **list** שיהיה רשימה המכילה את הטיפוס שהועבר, ואחריו כל איברי הרשימה.
  דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
typedef typename PrependList<Int<4>, list>::list newList; // = List< Int<4>, Int<1>, Int<2>, Int<3>>
```

- **ListGet** – מבנה שיקבל **מספר שלם N** ו**רשימה** (בסדר הזה) ויכיל את הטיפוס **value** שיהיה האיבר שנמצא באינדקס ה-Nי ברשימה. (האינדקסים מתחילים מ-0)
  ניתן להניח שלא יועברו אינדקסים שחורגים מגודל הרשימה או אינדקסים שליליים.
  דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
ListGet<0, list>::value; // = Int<1>
ListGet<2, list>::value; // = Int<3>
```

- **ListSet** – מבנה שיקבל **מספר שלם N**, **טיפוס** ו**רשימה** (בסדר הזה) ויכיל את הטיפוס **list** שיהיה רשימה הזהה לרשימה שהועברה, פרט לאיבר ה-Nי שהוחלף באיבר שהועבר. (האינדקסים מתחילים מ-0)
  ניתן להניח שלא יועברו אינדקסים שחורגים מגודל הרשימה או אינדקסים שליליים.

דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
typedef typename ListSet<0, Int<5>, list>::list listA; // = List<Int<5>, Int<2>, Int<3>>
typedef typename ListSet<2, Int<7>, list>::list listA; // = List<Int<1>, Int<2>, Int<7>>
```

רמז למימוש: חישבו כיצד ניתן להשתמש ב-PrependList.

הגדרה: מטריצה היא רשימה של רשימות של מספרים שלמים.

מכיוון שהרשימה שהגדרנו מקבלת **טיפוסים** ולא **ערכים** נוסיף מבנה עזר נוסף, שיהווה **Wrapper** לערך מספרי:

:Int<N>

המבנה יקבל ערך מספרי $\mathbf{N} \in \mathbb{Z}$ ויחזיק שדה **value** שיהיה שווה לערך N.

דוגמת שימוש:

```
typedef typename Int<5> wrappedFive;
int a = wrappedFive::value; // = 5
```

כעת, נוכל לייצג את המטריצה $\begin{pmatrix} 1 & -2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ באמצעות הקוד הבא:

```
typedef List<
                List< Int<1>, Int<-2>, Int<0> >,
                List< Int<0>, Int< 1>, Int<0> >,
                List< Int<0>, Int< 0>, Int<1> >
            > matrix;
```

לאחר שהגדרנו את המבנים לעיל, נעבור למימוש המערכת עצמה.

הערה חשובה: הגישה לממשק של הספרייה תתבצע דרך קובץ ה-Header **MatrixOperations.h** בלבד! יש לוודא שהוספתם include לכל קובץ שהגדרתם בחלק זה של התרגיל (כולל $Utilities.h$), כמו כן ניתן להפריד את הפעולות שיוגדרו בהמשך לקבצים ככל העולה על רוחכם, אך יש לוודא שכולם נגישים מ-MatrixOperations.h.

:Add

המבנה מקבל 2 מטריצות (= רשימות של רשימות, כפי שהוגדר) ומחזיק את **הטיפוס result** שיהיה המטריצה המתקבל כתוצאה **מחיבור** (סטנדרטי מעל $\mathbb{N}$) 2 המטריצות שהועברו.

ניתן להניח שגדלי המטריצות שיועברו גדולים ממש מ-0.

דוגמת שימוש במבנה:

```
typedef List<
                List< Int<1>, Int<2>, Int<0> >,
                List< Int<0>, Int<1>, Int<0> >,
                List< Int<0>, Int<0>, Int<5> >
            > matrix1;

typedef List<
                List< Int<7>, Int<6>, Int<0> >,
                List< Int<0>, Int<7>, Int<0> >,
                List< Int<8>, Int<0>, Int<3> >
            > matrix2;

typedef typename Add<matrix1, matrix2>::result matrix3; // = List<
                                                                List< Int<8>, Int<8>, Int<0> >,
                                                                List< Int<0>, Int<8>, Int<0> >,
                                                                List< Int<8>, Int<0>, Int<8> >
                                                            >
```

במקרים בהם תוצאת החיבור לא מוגדרת (המטריצות מגודל שונה) יש להכשיל את הקומפילציה!

Multiply:

המבנה מקבל 2 מטריצות (= רשימות של רשימות, כפי שהוגדר לעיל) ומחזיק את **הטיפוס** **result** שיהיה המטריצה המתקבל כתוצאה **מכפל** (סטנדרטי מעל $\mathbb{Z}$) המטריצה הראשונה במטריצה השנייה.

ניתן להניח שגדלי המטריצות שיועברו גדולים ממש מ-0.

דוגמת שימוש במבנה:

```
typedef List<
                List< Int<1>, Int<2> >,
                List< Int<0>, Int<1> >
            > matrix1;

typedef List<
                List< Int<0>, Int<7> >,
                List< Int<8>, Int<0> >
            > matrix2;

typedef typename Multiply<matrix1, matrix2>::result matrix3; // = List<
                                                                List< Int<16>, Int<7> >,
                                                                List< Int<8>, Int<0> >
                                                              >
```

תזכורת מאלגברה א' (או אלגוריתמים נומריים או אלגברה 1מ או כל קורס אחר הכולל מטריצות):
מכפלת מטריצות A ו-B $(A * B)$ כאשר A היא מטריצה מגודל $n \times m$ ו-B היא מטריצה מגודל $m \times l$ מוגדרת להיות מטריצה C מגודל $n \times p$ כאשר איברי C, $c_{i,j}$ מוגדרים להיות:

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} * b_{k,j}$$

במקרים בהם המכפלה לא מוגדרת (גובה המטריצה הראשונה שונה מאורך המטריצה השנייה) <u>יש להכשיל את הקומפילציה</u>!

רמז למימוש:

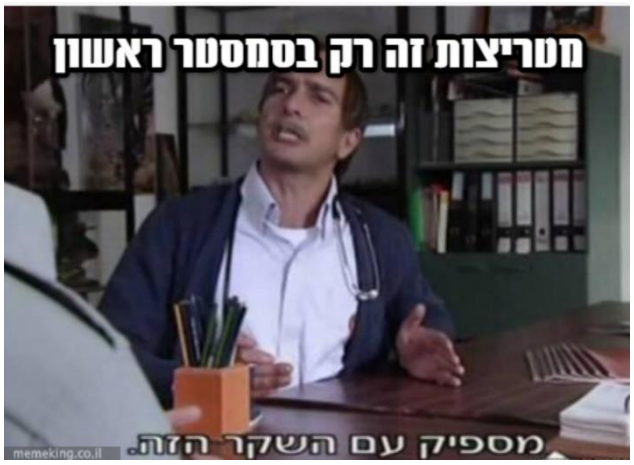
השתמשו במבנה **Transpose** שמקבל מטריצה ומחזיק טיפוס **matrix** שהוא המטריצה המשוחלפת. המבנה נמצא בקובץ המצורף **Transpose.h**.

עבור 2 הפעולות מומלץ מאוד לממש 2 מבני עזר MatrixGet, MatrixSet שמקבלים (כפרמטר גנרי) אינדקס שורה, אינדקס עמודה ומטריצה וממשים את הגרסה הדו מימדית של ListSet ו-ListGet. מבנים אלו אינם חובה ולכן לא ייבדקו.

<u>חזרו על התרגול של תבניות ב-C++ והבינו את הנושא היטב. עברו על המבנים המצורפים לתרגיל (בקובץ Transpose.h), הבינו איך (ולמה) הם עובדים. הבנה טובה של הנושא תעזור מאוד במימוש חלק זה של התרגיל.</u>

הערות והנחות:

- בכל מקום בו כתוב מבנה הכוונה היא ל-struct.
- עבור Add ו-Multiply, ניתן להניח שכל השורות של מטריצה מסוימת הן בעלות האורך.
- **ניתן ואף רצוי להוסיף שדות, מבנים וטיפוסים נוספים.**
- מומלץ להקפיד על חלוקה לקבצים שונים לפי קטגוריות. יש לוודא כי קובץ ה-Header הראשי (**MatrixOperations.h**) יכלול include לכל קובץ Header שיצרתם. חלק מהבדיקות בחלק זה של התרגיל יכילו include יחיד והוא ל-$MatrixOperations.h$.



חלק ב' – מימוש מנגנון דמוי Events ב-C++

מבוא:

בחלק זה של התרגיל נממש מנגנון המבוסס על תבנית התיכון Observer Pattern. בתבנית זו אובייקט הנקרא **Subject** "מכיר" קבוצת אובייקטים אחרים שנקראים **Observers** והוא שולח **לכולם**, אחד אחרי השני, הודעות על שינויים המתרחשים בו. "שליחת ההודעות" מתבצעת ע"י קריאה לפונקציה מסוימת של ה-**Observers**.

התבנית באה להגדיר יחס תלות של רבים-אחד בין ה-**Observers** ל-**Subject**. אובייקט ה-**Subject** תלוי ב-**Observers** שלו, אך הם לא. (תלויים ברמת שימוש בקוד). בנוסף, תבנית זו תורמת **לאנקפסולציה** של מערכת – אבסטרקציה של רכיבי מערכת מרכזיים בתור Subject ורכיבים משתנים בתור Observers.

לדוגמה – ממשקי משתמש גרפיים (GUI) יוזמים טיפול מבוזר באירועים שתלויים במשתמש, כאשר המתכנת יכול להוסיף טיפול מותאם אישית להם – כך למשל באפליקציות אנדרואיד המתכנת יכול לכתוב פונקציית טיפול בנגיעה בכפתור, שתקרא ברגע שהמשתמש נוגע בו.

בתרגול של #C תכירו את מערכת ה-Delegates וה-Events. מערכת זו מהווה מימוש ל-Observer Pattern, אך שימו לב כי תרגיל זה עומד בפני עצמו ואינו מצריך ידע מוקדם.

בגרסה שלנו, נגדיר 2 מחלקות גנריות – **Observer** ו-**Subject**. הפרמטר הגנרי שלהן יהיה הטיפוס אותו מעביר ה-Subject בתור הודעה ל-Observers שלו. בנוסף, המחלקה Observer תהיה וירטואלית טהורה – מחלקות שירשו ממנה יממשו את פונקציית הטיפול באירוע.

למעשה המחלקה Observer היא מעין Interface המצהיר על כוונת מחלקות היורשות ממנו להיות בעלות היכולת להירשם כ-Observer אצל Subject כלשהו.

המחלקה Subject (בקובץ Subject.h):

המחלקה הגנרית תקבל את T בתור הטיפוס הגנרי ותכיל בנאי חסר פרמטרים.

המחלקה תממש את המתודות הבאות:

- **`void notify(const T&)`** – מתודה המקבלת ארגומנט מהטיפוס הגנרי ושולח אותו כהודעה **לכל** ה-Observers אותם העצם מכיר. (קריאה למתודה `handleEvent()` שתוגדר בהמשך עבור המחלקה Observer)

  סדר הקריאה לכל ה-Observers יהיה לפי סדר ההוספה שלהם ל-Subject.
- **`void addObserver(Observer<T>&)`** – מתודה המקבלת עצם Observer ומוסיפה אותו לקבוצת האובייקטים אותם מקבל ההודעה מכיר. אם העצם כבר מוכר למקבל ההודעה יש לזרוק את החריגה **`ObserverAlreadyKnownToSubject`** המוגדרת בקובץ OOP5EventExceptions.h.
- **`void removeObserver(Observer<T>&)`** – מתודה המקבלת עצם Observer ומוחקת אותו מקבוצת האובייקטים אותם מקבל ההודעה מכיר. אם העצם אינו מוכר למקבל ההודעה יש לזרוק את החריגה **`ObserverUnknownToSubject`** המוגדרת בקובץ OOP5EventExceptions.h.
- **`Subject<T>& operator+=(Observer<T>&)`** – אופרטור += המקבל עצם Observer ומוסיף אותו לקבוצת העצמים המוכרים ל-Subject עליו הפעילו את האופרטור (תוך שימוש במתודת addObserver). יש להחזיר את עצם ה-Subject **לאחר** ההוספה.
- **`Subject<T>& operator-=(Observer<T>&)`** – אופרטור -= המקבל עצם Observer ומוחק אותו מקבוצת העצמים המוכרים ל-Subject עליו הפעילו את האופרטור (תוך שימוש במתודת removeObserver). יש להחזיר את עצם ה-Subject **לאחר** המחיקה.
- **`Subject<T>& operator()(const T&)`** – אופרטור () המקבל ארגומנט מהטיפוס הגנרי ושולח כהודעה לכל ה-Observers שהעצם מכיר (תוך שימוש במתודת notify). יש להחזיר את עצם ה-Subject **לאחר** השליחה.

המחלקה Observer (בקובץ Observer.h):

המחלקה הגנרית תקבל את T בתור הטיפוס הגנרי ותכיל בנאי חסר פרמטרים. **המחלקה תכיל את המתודה הוירטואלית הטהורה `void handleEvent(const T&)` שתהיה המתודה הנקראת על ידי ה-Subject בקריאה למתודת `notify`. מתודה זו היא למעשה "מצביע הפונקציה" שמוסיפים ל-Delegate ב-#C.**

דוגמת שימוש במנגנון בקובץ Part1Examples.cpp המצורף.

הערות לשני החלקים של התרגיל:

- בדיקת התרגיל תכלול הן את הפונקציונליות של המערכת כולה (אינטגרציה של כל הרכיבים) והן את הפונקציונליות של כל רכיב בפני עצמו (בדיקות יחידה). ודאו שכל רכיב עומד בדרישות התרגיל! לשם כך, מומלץ מאוד לכתוב ולהריץ בדיקות!
- עליכם לדאוג לשחרר את כל האובייקטים שהקצתם ב-2 חלקי התרגיל! בחלק ב' של התרגיל על המשתמש לדאוג לשחרור עצמים שהוא ייצר בתוך הפונקציות האנונימיות שהוא מספק. עליכם לדאוג להקצאות שלכם בלבד.
- הקוד יקומפל בעזרת הפקודה:

g++ -std=c++11 *.cpp

יש להריץ/לקמפל את העבודה כולה על שרת ה-csl2, שכן העבודה שלכם תבדק עליו.

באופן ברירת מחדל, על שרתי ה csl2, מותקנת גרסה ישנה של gcc אשר אינה תומכת ב C++11. על מנת לעדכן את הקומפיילר שלכם ב- csl2 / t2 באופן הבא:

```
1. gcc --version # If GCC is 4.7 or above, stop here.
2. bash
3. . /usr/local/gcc4.7/setup.sh
4. cd ~
5. echo . /usr/local/gcc4.7/setup.sh >> .bashrc # This makes the change permanent.
```

הוראות הגשה

- **בקשות לדחייה, מכל סיבה שהיא, יש לשלוח למתרגל האחראי על הקורס (נתן) במייל בלבד תחת הכותרת HW5 236703. שימו לב שבקורס יש מדיניות איחורים, כלומר ניתן להגיש באיחור גם בלי אישור דחייה – פרטים באתר הקורס תחת General info.**
- הגשת התרגיל תתבצע אלקטרונית בלבד (יש לשמור את אישור השליחה!)
- יש להגיש קובץ בשם `OOP5_<ID1>_<ID2>.zip` המכיל:
  - קובץ בשם readme.txt המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים בפורמט הבא:

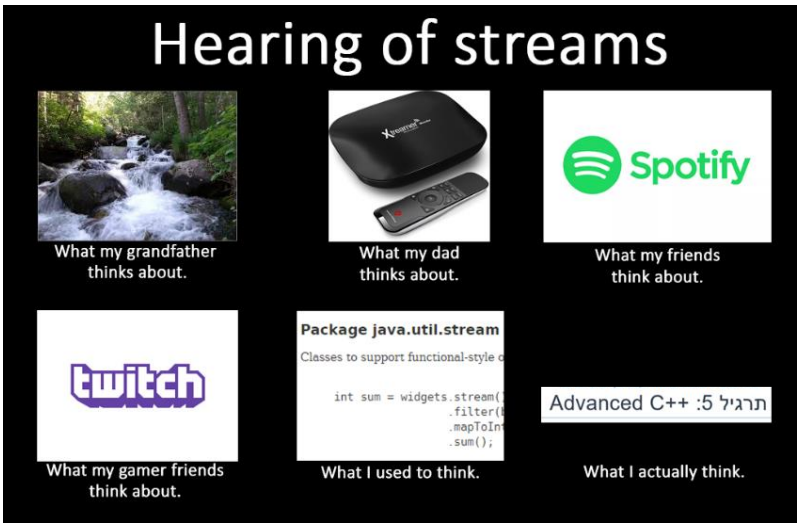
    Name1 id1 email1
    Name2 id2 email2
  - תיקייה בשם part1 שתכיל את כל הקבצים **שמימשתם** בחלק הראשון של התרגיל + הקובץ **TransposeList.h** שסופק לכם. (**לא** כולל Part1Examples.cpp!)
  - תיקייה בשם part2 שתכיל את כל הקבצים **שמימשתם** בחלק השני של התרגיל + הקובץ **OOP5EventException.h** שסופק לכם. (**לא** כולל Part2Examples.cpp!)

```
OOP5_123456789_112233445.zip/
    |_ readme.txt
    |_ part1/
            |_ Utilities.h
            |_ MatrixOperations.h
            |_ TransposeList.h
            |_ additional files…

    |_ part2/
```

```
|_ OOP5EventException.h
|_ Subject.h
|_ Observer.h
|_ additional files…
```

- נקודות יורדו למי שלא יעמוד בדרישות ההגשה (rar במקום zip, קבצים מיותרים נוספים, readme בעל שם לא נכון וכו')



בהצלחה!