<u>סדנת תכנות בשפת C++, מס׳ קורס 67317 - 2018</u>

<u>תרגיל 3</u>

<u>תאריך הגשת התרגיל והבוחן:</u> יום שלישי 17.1.19 עד שעה 23:55

הנחיות כלליות לקורס:

1. בכל התרגילים יש לעמוד בהנחיות הגשת התרגילים וסגנון כתיבת הקוד. שני המסמכים נמצאים באתר הקורס – הניקוד יכלול גם עמידה בדרישות אלו.
2. בכל התרגילים עליכם לכתוב קוד ברור. בכל מקרה בו הקוד שלכם אינו ברור מספיק עליכם להוסיף הערות הסבר בגוף הקוד. יש להקפיד על תיעוד (documentation) הקוד ובפרט תיעוד של כל פונקציה.
3. במידה ואתם משתמשים בעיצוב מיוחד או משהו לא שגרתי, עליכם להוסיף הערות בקוד המסבירות את העיצוב שלכם ומדוע בחרתם בו.
4. עליכם לקמפל עם הדגלים Wall -Wextra -std=c++17 -pthread -g ולוודא שהתוכנית מתקמפלת ללא אזהרות, **תכנית שמתקמפלת עם אזהרות תגרור הורדה משמעותית בציון התרגיל.**
5. עליכם לוודא שהתרגילים שלכם תקינים ועומדים בכל דרישות ההידור והריצה במחשבי בית הספר מבוססי מעבדי bit-64 (מחשבי האקוריום, לוי, השרת river). **<u>חובה להריץ את התרגיל במחשבי בית הספר לפני ההגשה</u>**. (ניתן לוודא שהמחשב עליו אתם עובדים הנו בתצורת bit-64 באמצעות הפקודה "uname -a" ווידוא כי הארכיטקטורה היא 64, למשל אם כתוב x86_64)
6. לאחר ההגשה, בדקו הפלט המתקבל בקובץ ה-PDF שנוצר מה presubmission script בזמן ההגשה. באם ישנן שגיאות, תקנו אותן על מנת שלא לאבד נקודות.
   <u>שימו לב ! תרגיל שלא יעבור את ה presubmission script ציונו ירד משמעותית</u> (הציון יתחיל מ-50, ויוכל לרדת) **<u>ולא יהיה ניתן לערער על כך.</u>**
7. בדיקת הקוד לפני ההגשה, גם על ידי קריאתו וגם על ידי כתיבת בדיקות אוטומטיות (tests) עבורו היא אחריותכם. חישבו על מקרי קצה לבדיקת הקוד.

הנחיות חשובות לכלל התרגילים בקורס C++

1. הקפידו להשתמש בפונקציות ואובייקטים של C++ (למשל new, delete, cout) על פני פונקציות של C (למשל malloc, free, printf). בפרט השתמשו במחלקה string (ב-std::string) ולא במחרוזת של (* char) C!
2. יש להשתמש בספריות סטנדרטיות של C++ ולא של C אלא אם כן הדבר הכרחי (וגם אז עליכם להוסיף הערה המסבירה את הסיבות לכך).
3. הקפידו על עקרונות Information Hiding – לדוגמא, הקפידו כי משתני המחלקות שלכם מוגדרים כמשתנים פרטיים (private).
4. הקפידו לא להעתיק by value משתנים כבדים, אלא להעבירם (היכן שניתן) by reference.
5. **<u>הקפידו מאוד</u>** על שימוש במילה השמורה const בהגדרות הפונקציות והפרמטרים שהן מקבלות.
   פונקציות שאינן משנות פרמטר מסויים – הוסיפו const לפני הגדרת הפרמטר.
   מתודות של מחלקה שאינן משנות את משתני המחלקה – הוסיפו const להגדרת המתודה.
   שימו לב: הגדרת משתנים / מחלקות ב- C++ כקבועים הוא אחד העקרונות החשובים בשפה.

הנחיות ספציפיות לתרגיל זה:

1. מומלץ מאוד לקרוא ולהבין כל התרגיל <u>לפני</u> שאתם מתחילים לתכנן את מימוש התרגיל. עיצוב הקוד (code design) שלכם הוא חלק מהדברים שייבדקו בתרגיל.
2. בתרגיל זה אינכם רשאים להוסיף קבצים נוספים.
3. בתרגיל זה הנכם רשאים (ואף נדרשים) להשתמש ב STL, לדוגמא vector יכול לעזור.
4. בתרגיל זה חל איסור על שימוש ב- new ו- delete.

5. עליכם להתמודד עם כישלון הקצאות זיכרון באמצעות מנגנון ה-exceptions. החריגות שאתם זורקים, צריכות לרשת מ-std::exception ולהיות אינפורמטיביות - נושא זה יועבר בתרגול הקרוב.
6. הוסיפו לתיעוד כל פונקציה איזו שגיאה היא זורקת ובאילו מצבים, תעדו גם אם השגיאה עלולה להיזרק מפונקציה מובנית או מספרייה שאתם משתמשים בה.

Matrix

בתרגיל זה תידרשו לממש מטריצה גנרית, כלומר איברי המטריצה יהיו מטיפוס גנרי T. המחלקה תוכל לשמש כקונטיינר לכל טיפוס שהוא (בדומה ל std::vector ו std::list הגנריים המסוגלים להכיל איברים מכל טיפוס שהוא) המייצג מספרים עם פעולות חשבון מוגדרות מראש, המחלקה תדע לבצע פעולות חישוב של מטריצות. עליכם לכתוב את הקובץ Matrix.hpp שיכיל את ההצהרה והמימוש של המחלקה הגנרית Matrix.
ניתן ליצור מטריצה מכל טיפוס T אשר יש לו מימוש לאופרטורים +, -, =-, =+, *, =,== ו >>, וכן מימוש של בנאי העתקה ובנאי האפס (מקבל 0 כארגומנט וייוצר את איבר האפס של המחלקה). למשל int או double.
לכל טיפוס עשויה להיות דרך שונה לחישוב פעולות החשבון, לייצוג כמחרוזת, ואיבר אפס משלו.

יישום הממשק Matrix

- בקובץ Matrix.hpp עליכם להגדיר את המחלקה Matrix, שתתאר מטריצה גנרית, שאיבריה מטיפוס כל שהוא (כפי שהוגדר לעיל).
- המימושים לפונקציות המחלקה יהיו בקובץ Matrix.hpp. ויכללו את הפונקציות הבאות:
  - בנאי ברירת מחדל (ללא ארגומנטים) - המחזיר מטריצה ממימד 1x1 המכילה את איבר ה-0.
  - בנאי המקבל את ממדי המטריצה

    Matrix(unsigned int rows, unsigned int cols)

    ומאתחל מטריצה בגודל הנתון שכל התאים בה מכילים את איברי האפס.
  - בנאי העתקה.
  - בנאי הממש את החתימה הבאה:

    Matrix(unsigned int rows, unsigned int cols, const vector<T>& cells)

    ומאתחל מטריצה בגודל הנתון המכילה את איברי הוקטור הנתון. סדר האיברים בווקטור תואם את סדר המעבר על איברי המטריצה באמצעות האיטרטור (ראו להלן). עליכם להניח של-T יש בנאי העתקה.
  - Destructor.
  - אופרטור השמה ('=') לשם ביצוע פעולת השמת מטריצה. אופרטור זה מאפשר שינוי של המטריצה המיוצגת על ידי האובייקט שמשמאל לסימן ה-'=', כך שתהייה זהה למטריצה המיוצגת על ידי האובייקט המועבר כפרמטר (מימין לסימון ה-'='). זכרו: פעולת ההשמה גורמת להיווצרות עותק זהה ובלתי תלוי.
  - אופרטור חיבור ('+') שערך ההחזרה שלו non-const לשם ביצוע פעולות חיבור מטריצות. (בונוס - על אופרטור זה לתמוך גם במימוש מקבילי - ראו בהמשך).
  - אופרטור חיסור ('-') שערך ההחזרה שלו non-const לשם ביצוע פעולות חיסור מטריצות.
  - אופרטור כפל ('*') שערך ההחזרה שלו non-const לשם ביצוע פעולות כפל מטריצות - עליכם להשתמש באלגוריתם הנאיבי של כפל מטריצות - Iterative Algorithm[1].
    (בונוס - על אופרטור זה לתמוך גם במימוש מקבילי - ראו בהמשך)
  - אופרטורי השוואה ('==' ו- '=!') לשם ביצוע פעולת השוואת מטריצות.
  - פונקציית שחלוף בשם trans. הפונקציה אינה משנה את האובייקט עליו היא הופעלה, אלא מחזירה אובייקט חדש. הפונקציה פועלת רק על מטריצות ריבועיות, וזורקת חריגה אם המטריצה אינה ריבועית.

[1] https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm#Iterative_algorithm

- פונקציה בשם isSquareMatrix שמחזירה true אם המטריצה ריבועית.
- מימוש אופרטור '<<' לשם הדפסת המטריצה עם אובייקט ostream באופן הבא: כל שורת ערכים מודפסת בשורה נפרדת, ו tab מפריד בין הערכים (גם לאחר הערך האחרון בשורה). ראו את הפלט לדוגמה של פתרון בית הספר והשוו בעזרת diff כדי לוודא שהמחרוזת אותה אתם מדפיסים היא נכונה.
- אופרטור() המקבל כפרמטרים (unsigned int,unsigned int) ומחזיר את הערך בתא [row,col]. יש לממש גרסת const ו-non-const לאופרטור זה (חישבו מה צריך להיות ערך ההחזרה בכל אחד מהמקרים).
- איטרטורים: עליכם לממש את הפונקציות begin ו- end כך שהן יחזירו ערכים המתאימים לאיטרטור העובר על כל המטריצה. על איטרטור זה **להיות const (כלומר ההוא איננו מאפשר לשנות את איברי המטריצה) ועליו לממש** את ה concept של Bidirectional Iterator (חפשו באינטרנט את הממשק התואם לסטנדרט c++14). כלומר, יש לממש את המתודות הבאות:
  - מתודה בשם ()begin המחזירה איטרטור על כל תאי המטריצה לפי הסדר הרגיל, המתחיל בתחילת המטריצה:

(0,0)->(0,1)->...->(0,col-1)->(1,0)->....->(row-1,col-1)

  - מתודה בשם ()end המחזירה איטרטור המצביע לסוף המטריצה כמקובל בסטנדרט.
- פונקצית ()rows ופונקציית ()cols, המחזירות בהתאם את מספר השורות והעמודות במטריצה.
- <u>בנוסף תוכלו להוסיף עוד פונקציות ציבוריות או פרטיות כרצונכם, לפי מה שנראה לכם שימושי למחלקה.</u>

- <u>התמחות (specialization).</u>
  - בנוסף למימוש הגנרי של המחלקה Matrix, עליכם להוסיף לקובץ Matrix.hpp גם מימוש ספציפי אחד: מימוש אלטרנטיבי לפונקציה trans, המחשבת את הצמוד[2] של המטריצה (הכוונה למטריצה הצמודה הרמיטית), עבור המקרה בו הטיפוס של האיברים הוא Complex . (ישנו קובץ נתון למחלקה זו)

<u>טיפים והנחיות:</u>

- <u>מימוש המחלקת מטריצות:</u>
  - המימוש הפנימי של המטריצה ישפיע על מימוש הדרישות לעיל, ובחירה נכונה יכולה לחסוך לכם לא מעט עבודה.
  - כל הפונקציות המתאימות מבצעות את הפעולה המתמטית המקבילה להגדרתם.
  - בכל מקרה בו לא ניתן לבצע את הפעולה עליכם לזרוק חריגה עם הסבר מתאים (למשל הכפלת מטריצות מממדים לא תואמים). אין חובה שההסבר יתאם במדויק לפתרון בית הספר, אבל עליכם לספק הסבר אינפורמטיבי והגיוני כלשהו באמצעות מנגנון החריגות.
  - באופרטור '=' מתבצע עדכון האובייקט השמאלי. חישבו היטב מה קורה מבחינת הזיכרון כאשר המטריצה משנה את ממדיה עקב פעולה זו.
  - הממשק המדויק (החתימות של הפונקציות) לא מוכתב לכם ונתון להחלטתכם, אבל ברוב המקרים ישנה דרך עיקרית אחת שהיא הטובה ביותר להגדרת הפונקציה. חישבו למשל על:
    - האם על הפונקציה להיות מוגדרת כ const.
    - האם הארגומנט צריך להיות מועבר by reference או by value, או אולי כדאי להשתמש במצביע. האם הארגומנט צריך להיות מוגדר כ const?
    - האם ערך ההחזרה צריך להיות מוגדר כ const, והאם הוא מועבר by reference או by value.

[2] https://en.wikipedia.org/wiki/Conjugate_transpose

- חישבו איך האופרטור שאתם מממשים פועל על טיפוסים מובנים בשפה ונסו להתחקות אחרי זה במימוש שלכם עבור המטריצה.
- כמו כן, עליכם לוודא שהממשק שלכם תואם את הדרייבר GenericMatrixDriver.cpp שמסופק לכם (ואת הדרייבר ParallelChecker.cpp במידה ומימשתם את הבונוס).

עבודה עם eigen

- תיאור כללי:
  - ספריית Eigen היא ספריה מתמטית אשר מסוגלת לבצע מספר רב של חישובים מתמטיים - ביניהם פעולות על מטריצות
  - אמליץ לכם תחילה להסתכל על הAPI שלה כאן

המדידות - timeChecker.cpp

- הקובץ TimeChecker.cpp:
  - על קובץ זה להכיל רק מתודת main המקבלת ארגומנט יחיד - מספר בין 1 ל 500.
  - במתודת main עליכם לעשות שימוש בספריית eigen על מנת לבצע חישוב של כפל מטריצות בגודל שיתקבל מן הארגומנטים. (ניתן להניח כי גודל זה תקין)
  - לפעולות הכפל והחיבור בeigen ביצעו override ולכן תוכלו להשתמש בהן כפי שאתם מכירים.
  - על מנת למדוד זמן יש בקובץ ParallelChecker.cpp מימוש פשוט. ניתן להעתיקו לקובץ שלכם.
  - אופן הדפסת הפלט יהיה בפורמט הבא:
    - size <n> \n
    - eigen mult <num1>\n
    - eigen add <num2>\n
    - matlib mult <num3>\n
    - matlib add <num4>\n
    - matlibP mult <num5>\n
    - matlibP add <num6>\n

  - כאשר n הינו גודל המטריצה והמספרים 1,2,3… הם הזמן אשר התקבל מן הפונקציות מדידה.
- eigen:
  - המטריצה צריכה להיות רנדומית ותתקבל ע"י :
    - MatrixXd m = MatrixXd::Random(n,n);
  - בהינתן הגודל נרצה להגריל 2 מטריצות רנדומיות ריבועית בגודל n על n ולאחר מכן למדוד את הזמן שלוקח לספרייה לכפול ולחבר את המטריצות. (שני זמנים נפרדים) (ללא הזמן ליצור את המטריצות)

- הספרייה שלכם:
  - עליכם ליצור מטריצת אחדות בגודל nxn
  - תוכלו ליצור את המטריצה שלכם באיזה דרך שתרצו.
  - מדידת הזמן תיעשה על פעולת הכפל והחיבור בנפרד - ללא זמן היצירה.
  - באם מימשתם את הבונוס יש לצרף חלק מדידה נוסף בו הזמן שלכם לבונוס.

- סכמו את הערך המודפס עבור כל אחת מהפעולות בטבלה הבאה, בקובץ הREADME.

	50x50		500x500	
	+	*	+	*
מקבילי (אם יש)				
טורי				
ספרייה				

- ענו בקובץ ה README על השאלות הבאות:
  - ענו על השאלות הבאות עבור 2 המצבים הבאים : טורי וספריה (ומקבילי במידה וביצעתם את הבונוס)
  - באיזה מצב התכנית רצה מהר יותר? ממה נובע הבדל זה?האם יש הבדל בין 2 הסטים של הנתונים?
  - האם יש הבדל בין פעולות החיבור, אל מול הכפל? ממה נובע הבדל זה?

<u>קבצים נתונים:</u>

- <u>מימוש טיפוס Complex</u>
  - מימשנו עבורכם את המחלקה Complex בקובץ Complex.cpp לפי הממשק הנתון בקובץ Complex.h. זוהי מחלקת מספרים מורכבים אשר תוכלו לבדוק באמצעותה את המימוש שלכם למטריצה הגנרית.
  - <u>אין להגיש קבצים אלו.</u>

- <u>הדרייבר GenericMatrixDriver</u>
  - לרשותכם דרייבר בשם GenericMatrixDriver.cpp שבודק **<u>באופן בסיסי</u>** את המחלקה הגנרית שלכם. אתם מוזמנים להשתמש בו ולשנותו כרצונכם.
  - אתם מוזמנים להדר ולהריץ אותו ביחד עם הספריה שלכם.
  - <u>אין להגיש קובץ זה</u>

- <u>הקובץ ParallelChecker.cpp:</u>
  - משמש לבדיקת הבונוס ומדידת זמנים.
  - על מנת לצרף מדידת זמנים ניתן לקחת את הקוד המוכן מהקובץ הנ"ל ולבצע בו שימוש.
  - <u>אין להגיש קובץ זה</u>

<u>בונוס! 10 נקודות - תכנות מקבילי:</u>

- <u>רקע:</u>
  - עליכם לספק מימוש מקבילי לאופרטורים + ו- *.
  - עיבוד מקבילי הוא עיבוד בו זמנית של מטלה מסוימת על ידי מספר מעבדים או מספר ליבות, כאשר היא מפוצלת בהתאם, כדי להגיע לתוצאות מהר יותר משיטה של עיבוד טורי. הרעיון מבוסס על העובדה שניתן בדרך כלל לפצל את תהליך הפתרון של בעיה כלשהי למספר מטלות קטנות יותר, שאותן ניתן לבצע בו-זמנית, עם מידה מסוימת של תיאום. בתרגיל זה נתרגל מעט תכנות מקבילי ונטעם על קצה המזלג את היתרונות והחסרונות שלו.
- <u>מימוש:</u>

- עליכם להגדיר במחלקה Matrix מתודה סטטית בשם setParallel המקבלת ארגומנט מסוג bool. במידה והארגומנט הוא true, לאחר הקריאה למתודה עם הערך **true**, פעולות הכפל/חיבור של מטריצות יתבצעו באמצעות תכנות מקבילי. קריאה למתודה עם ארגומנט false תחזיר את המחלקה להתנהגות הדיפולטיבית שלה, בה יתבצעו פעולות כפל/חיבור באופן טורי (כרגיל).
- בעקבות כל קריאה למתודה שמשנה את התנהגות המחלקה עליכם להדפיס את ההודעה הבאה:

Generic Matrix mode changed to (Parallel|non-Parallel) mode.

ההודעה תודפס רק אם היה שינוי בהתנהגות המחלקה יחסית למצב הנוכחי.

- הנחיות והקלות במימוש:
  - על מנת להקל עליכם ובכדי שהביצועים שלכם יהיו דומים, אתם נדרשים לשמור על הכללים הבאים:
    - המתודות אותן אתם נדרשים לממש גם במצב מקבילי, הן אופרטורי הכפל (*) והחיבור (+) בלבד.
    - על מנת למנוע הסתבכויות מיותרות, עליכם להשתמש באלגוריתם הנאיבי של כפל מטריצות - Iterative Algorithm[3], גם למימוש הטורי (רגיל) וגם למימוש המקבילי.
    - בכל הפעולות הנ"ל על התכנות המקבילי להתבצע עבור כל שורה במטריצת התוצאה במקביל (כלומר, שימוש ב thread נפרד לכל חישוב שורה במטריצת תוצאה).
- בדיקת נכונות הבונוס:
  - בחלק זה של התרגיל אנו נשווה את התנהגות של התכנות המקבילי אל התכנות הטורי, ונוודא שאכן יש כאן עבודה מקבילית. אין להגיש חלק זה
  - לשם כך מסופקים לכם:
    - קובץ בשם ParallelChecker.cpp
    - 2 קבצי קלט המכילים נתונים באמצעותם תתבצע ההשוואה:

(קובץ גדול ואנו מציעים לקרוא אותו ישירות מהנתיב המסופק big.txt) ~labcpp/www/ex3/sets/big.txt

~labcpp/www/ex3/sets/small.txt

    - הוראות שימוש:
      - בצעו קומפילציה ו-linkage עם הדגלים 'O-':

```
g++ -std=c++11 -Wextra -Wall -pthread -Wvla -O –DNDEBUG  ParallelChecker.cpp
Complex.cpp -o ParallelChecker
```

הדגל O- אומר לקומפיילר לבצע אופטימיזציות על הקוד כך שירוץ מהר יותר.
בדוגמא זו ביצענו את הקומפילציה וה-linkage בשורה אחת, הדבר אינו הכרחי (אך מפשט את החיים במקרה זה).

      - הריצו את התכנית שנוצרה על 2 הקבצים של הנתונים וודאו שהיא פועלת בצורה תקינה וכי אתם מבינים כיצד להשתמש בה.

חומר עזר

1. את פתרון הבית ספר ניתן למצוא ב:

~labcpp/www/ex3/GenericMatrixDriver

2. את קבצי התרגיל ניתן למצוא ב:

~labcpp/www/ex3/ex3_files.tar

[3] https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm#Iterative_algorithm

בדקו את תכניתכם וודאו שהפלטים שלכם זהים לאלה של פתרון בית הספר. אתם יכולים לייצר קבצי קלט רבים נוספים כדי לבדוק מקרים נוספים, ולהשוות את הפלט של התכנית שלכם עם פלטים של תלמידים אחרים, או עם הפלט שנוצר כשאתם נותנים את הקלט הזה לקובץ הריצה של פתרון בית הספר.

3. ביצוע overloading ב-C++:

http://www.cprogramming.com/tutorial/operator_overloading.html
http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B
http://www.cplusplus.com/reference/set/set/operators/

הגשה

1. עליכם להגיש קובץ tar בשם ex3.tar מכיל את כל הקבצים הנדרשים לקימפול התוכנית שלכם ואת הקבצים הבאים:
   1.1. Matrix.hpp
   1.2. TimeChecker.cpp
   1.3. קובץ Makefile התומך בפקודות הבאות:
      1.3.1. הרצת make ללא פרמטרים, תייצר את timeChecker **ותריץ אותו. עם 500 כקלט**
      1.3.2. make Matrix- יצירת Matrix.hpp.gch.
      1.3.3. make clean - ניקוי כל הקבצים שנוצרו באמצעות פקודות ה-makefile.
   1.4. README עם התשובות והמדידת זמנים של העיבוד המקבילי והספרייה.
   1.5. extension.pdf - רק במקרה שההגשה היא הגשה באיחור.

אין להגיש את הקבצים הבאים:
GenericMatrixDriver.cpp, ParallelChecker.cpp, Complex.h, Complex.cpp

2. שימו לב! על אף שאתם יכולים להוסיף קבצים נוספים כרצונכם, הימנעו מהוספת קבצים לא רלוונטים (גם בכדי להקל על הבודקים, וגם בכדי שציונכם לא יפגע מכך). אנו נוריד נקודות למי שיגיש קבצים לא רלוונטים.
3. ניתן ליצור קובץ tar כדרוש על ידי הפקודה:

```
tar cvf <tar name> <files>
```

4. לפני ההגשה, פתחו את הקובץ ex3.tar בתיקיה נפרדת וודאו שהקבצים מתקמפלים ללא שגיאות וללא אזהרות. וודאו שההגשה שלכם עוברת את ה-presubmission script ללא שגיאות או אזהרות.

```
~labcpp/www/ex3/presubmit_ex3
```

5. אתם יכולים להריץ בעצמכם בדיקה אוטומטית עבור סגנון קידוד בעזרת הפקודה:

```
~labcpp/www/codingStyleCheck <code file or directory>
```

כאשר <directory or file> מוחלף בשם הקובץ אותו אתם רוצים לבדוק או תיקייה שיבדקו כל הקבצים הנמצאים בה (שימו לב שבדיקה אוטומטית זו הינה רק חלק מבדיקות ה codingStyle)

6. דאגו לבדוק לאחר ההגשה את קובץ הפלט (submission.pdf) וודאו שההגשה שלכם עוברת את ה-presubmission script ללא שגיאות או אזהרות.

בהצלחה!