שפת C++ – תרגיל 3

Template class, exceptions, Move Semantics, STL and Multithreading

תאריך הגשה של התרגיל והבוחן: **יום ראשון 17.09.17 עד שעה 23:55** [1]

1. **הנחיות חשובות:**

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. עבור כל פונקציה בה אתם משתמשים, עליכם לוודא שאתם מבינים היטב מה הפונקציה עושה גם במקרי קצה (התייחסו לכך בתיעוד). ובפרט עליכם לוודא שהפונקציה הצליחה.
5. בכל התרגילים במידה ויש לכם הארכה, או שאתם מגישים באיחור. **חל איסור להגיש קובץ כלשהו בלינק הרגיל (גם אם לינק overdue טרם נפתח)**. **מי שיגיש קבצים בשני הלינקים מסתכן בהורדת ציון משמעותית.**
6. אין להגיש קבצים נוספים על אלו שתדרשו .ובפרט אין להגיש קובץ README אלא אם צוין במפורש שיש צורך בכך (לדוגמא, בתרגיל זה אין צורך להגיש).
7. עליכם לקמפל עם הדגלים Wall -Wextra -Wvla -std=c++17 -pthread -g ולוודא שהתוכנית מתקמפלת ללא אזהרות, **תכנית שמתקמפלת עם אזהרות תגרור הורדה משמעותית בציון התרגיל.** למשל, בכדי ליצור תוכנית מקובץ מקור בשם ex1.cpp יש להריץ את הפקודה:

g++ -Wextra -Wall -Wvla -std=c++17 -pthread -g ex1.cpp -o ex1

8. עליכם לוודא שהתרגילים שלכם תקינים ועומדים בכל דרישות הקימפול והריצה במחשבי בית הספר מבוססי מעבדי bit-64 (מחשבי האקוריום, לוי, השרת river). **חובה להריץ את התרגיל במחשבי בית הספר לפני ההגשה**. (ניתן לוודא שהמחשב עליו אתם עובדים הנו בתצורת bit-64 באמצעות הפקודה "uname -a" ווידוא כי הארכיטקטורה היא 64, למשל אם כתוב x86_64)
9. לאחר ההגשה, בדקו את הפלט המתקבל בקובץ ה-PDF שנוצר מהpresubmission script בזמן ההגשה. באם ישנן שגיאות, תקנו אותן על מנת שלא לאבד נקודות.
שימו לב ! תרגיל שלא יעבור את ה presubmission script ציונו ירד משמעותית (הציון יתחיל מ-50, ויוכל לרדת) **ולא יהיה ניתן לערער על כך.**
10. בדיקת הקוד לפני ההגשה, גם על ידי קריאתו וגם על ידי כתיבת בדיקות אוטומטיות (tests) עבורו היא אחריותכם. בידקו מקרי קצה.
במידה וסיפקנו לכם קבצי בדיקה לדוגמא, השימוש בהם יהיה על אחריותכם. **במהלך הבדיקה הקוד שלכם ייבדק מול קלטים נוספים לשם מתן הציון.**
11. **הגשה מתוקנת -** לאחר מועד הגשת התרגיל ירוצו הבדיקות האוטמטיות ותקבלו פירוט על הטסטים בהם נפלתם. לשם שיפור הציון יהיה ניתן להגיש שוב את התרגיל לאחר תיקוני קוד ולקבל בחזרה חלק מהנקודות - פרטים מלאים יפורסמו בפורום ואתר הקורס.

[1] ניתן להגיש באיחור של עד 24 שעות עם קנס של 10 נקודות.

2. 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 C (* char).
2. יש להשתמש בספריות סטדנרטיות 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. הקפידו מאוד על שימוש במילה השמורה const בהגדרות המתודות והפרמטרים שהן מקבלות:
המתודות 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.
6. הקפידו על השימוש ב- static, במקומות המתאימים (הן במשתנים והן במתודות)
7. הקפידו לשחרר את כל הזיכרון שאתם מקצים (השתמשו ב-valgrind כדי לבדוק שאין לכם דליפות זיכרון).
8. שימו לב שהאלגוריתמים שלכם צריכים להיות יעילים.
9. אתם רשאים (ולעתים אף נדרשים) להגדיר פונקציות נוספות לשימושכם הפנימי.

3. הנחיות ספציפיות לתרגיל זה:

1. בתרגיל זה הינכם נדרשים להשתמש ב STL, לדוגמא vector עשוי לעזור.
2. בתרגיל זה חל איסור על שימוש ב- new ו- delete.
3. עליכם להתמודד עם כישלון הקצאות זיכרו באמצעות מנגנון ה-exceptions. החריגות שאתם זורקים, צריכות לרשת מ-std::exception ולהיות אינפורמטיביות.
4. הוסיפו לתיעוד כל פונקציה איזו שגיאה היא זורקת ובאילו מצבים, תעדו גם אם השגיאה עלולה להיזרק מפונקציה מובנית או מספרייה שאתם משתמשים בה.

4. Matrix:

1. <u>רקע</u>
   a. מחלקה גנרית של מטריצה, כלומר איברי המטריצה הם מטיפוס גנרי. המחלקה תוכל גם 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.
   b. 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.
2. <u>יישום הממשק Matrix</u>
   a. בקובץ Matrix.hpp עליכם להגדיר את המחלקה Matrix, שתתאר מטריצה גנרית, שאיבריה מטיפוס כל שהוא (כפי שהוגדר לעיל).
   b. 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 מימדי המטריצה

```
Matrix(std::size_t rows, std::size_t cols)
```

ומאתחל מטריצה בגודל הנתון שכל התאים בה מכילים את איברי האפס.

   - בנאי העתקה.
   - בנאי העברה (move constructor)

- בנאי הממש את החתימה הבאה:

```
Matrix(std::size_t rows, std::size_t cols, const vector<T>& cells)
```

מאתחל 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 ('+') לשם ביצוע פעולות חיבור מטריצות. על אופרטור זה לתמוך גם במימוש מקבילי (ראו בהמשך).
- אופרטור חיסור ('-') לשם ביצוע פעולות חיסור מטריצות. שימו לב - אינכם נדרשים לממש אופרטור זה גם באופן מקבילי (אבל אתם רשאים לעשות זאת).
- אופרטור כפל ('*') לשם ביצוע פעולות כפל מטריצות - עליכם להשתמש באלגוריתם הנאיבי של כפל מטריצות - Iterative Algorithm[2]. על אופרטור זה לתמוך גם במימוש מקבילי (ראו בהמשך).
- אופרטורי השוואה ('==' ו- '!=') לשם ביצוע פעולת השוואת מטריצות.
- פונקצית שחלוף בשם trans. הפונקציה אינה משנה את האובייקט עליו היא הופעלה, אלא מחזירה אובייקט חדש.
- פונקצייה בשם isSquareMatrix שמחזירה true אאם המטריצה ריבועית.
- מימוש אופרטור '<<' לשם הדפסת המטריצה עם אובייקט ostream באופן הבא: כל שורת ערכים מודפסת בשורה נפרדת, ו tab מפריד בין הערכים. ראו את הפלט לדוגמה של פתרון בית הספר והשוו בעזרת diff כדי לוודא שהמחרוזת אותה אתם מדפיסים היא נכונה.
- אופרטור() המקבל כפרמטרים (std::size_t, std::size_t) ומחזיר את הערך בתא [row,col]. יש לממש גרסאת 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 כל מה ש-BidirectionalIterator דורש. (חפשו באינטרנט את הממשק התואם לסטנדרט c++17). 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.
- בנוסף תוכלו להוסיף עוד פונקציות ציבוריות או פרטיות כרצונכם, לפי מה שנראה לכם שימושי למחלקה.

---

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

c. טיפים והנחיות:

- 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 ממימדים לא תואמים). אין חובה שההסבר יתאם במדויק לפתרון בית הספר, אבל עליכם לספק הסבר אינפורמטיבי והגיוני כלשהו.
- 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.
  - חישבו איך האופרטור שאתם מממשים פועל על טיפוסים מובנים בשפה ונסו להתחקות אחרי זה במימוש שלכם עבור המטריצה.
- בכל מקרה, **עליכם לוודא שהממשק שלכם תואם את הדרייברים:** ParllelChecker.cpp, GenericMatrixDriver_cpp שמסופקים לכם.

3. מימוש טיפוס Complex

a. 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.

b. **אין להגיש קבצים אלו.**

4. <u>התמחות (specialization).</u>

a. בנוסף למימוש הגנרי של המחלקה Matrix, עליכם להוסיף לקובץ Matrix.hpp גם מימוש ספציפי אחד: **מימוש אלטרנטיבי לפונקציה trans**, המחשבת את הצמוד[3] של המטריצה (הכוונה למטריצה צמודה הרמטית ולא קלאסית), עבור המקרה בו הטיפוס של האיברים הוא Complex.

5. <u>תיכנות מקבילי:</u>

a. בתרגיל זה עליכם לספק מימוש מקבילי לאופרטורים + ו- *.

b. רקע - 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.

c. עליכם להגדיר במחלקה Matrix מתודה סטטית בשם setParallel המקבלת ארגומנט מסוג bool. במידה והארגומנט הוא true, לאחר הקריאה למתודה עם הערך **true**, <u>פעולות הכפל/חיבור</u> של כל המטריצות יתבצעו באמצעות תכנות מקבילי. קריאה למתודה עם ארגומנט false תחזיר את המחלקה להתנהגות ברירת המחדל שלה, בה יתבצעו פעולות כפל/חיבור באופן טורי (כרגיל).

d. בעקבות כל קריאה למתודה שמשנה את התנהגות המחלקה עליכם להדפיס את ההודעה הבאה:

"Generic Matrix mode changed to (parallel|non-parallel) mode.\n"

---

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

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

- על מנת להקל עליכם ובכדי שהביצועים שלכם יהיו דומים, אתם נדרשים לשמור על הכללים הבאים:
- המתודות אותן אתם נדרשים לממש גם במוד מקבילי, הן אופרטורי הכפל (*) והחיבור (+) בלבד.
- על מנת למנוע הסתבכויות מיותרות, עליכם להשתמש באלגוריתם הנאיבי של כפל מטריצות - Iterative Algorithm[4], 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).

6. השוואת עיבוד מקבילי וטורי:

a. בחלק זה של התרגיל אנו נשווה את התנהגות של התכנות המקבילי אל התכנות הטורי, וננסה לזהות חלק מהיתרונות והחסרונות שלהם.

b. לשם כך מסופקים לכם:
- קובץ בשם ParllelChecker.cpp
- 2 קבצי קלט המכילים נתונים באמצעותם תתבצע ההשוואה:

```
~labcpp/www/sets/big.txt
~labcpp/www/sets/small.txt
```

(big.txt קובץ גדול ואנו מציעים לקרוא אותו ישירות מהנתיב המסופק)

c. אנו נשתמש ב- ParllelChecker על מנת להשוות את זמן הריצה במצב העיבוד הטורי אל מול מצב העיבוד המקבילי, עבור כל אחד מקבצי הקלט.

d. אנו נעשה את ההשוואה הזו על 2 קבצי הנתונים וננסה להבין האם יש הבדלים בניהם ומהם הגורמים לכך.

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

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

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.

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

	small		big	
	+	*	+	*
מקבילי				
טורי				

g. ענו בקובץ ה README על השאלות הבאות:
- האם יש הבדל בין 2 הסטים של הנתונים? באיזה מצב התכנית רצה מהר יותר? ממה נובע הבדל זה?
- האם יש הבדל בין פעולות החיבור והחיסור, אל מול הכפל? ממה נובע הבדל זה?

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

7. הדרייבר GenericMatrixDriver
   a. לרשותכם דרייבר בשם GenericMatrixDriver.cpp שבודק **באופן בסיסי** את המחלקה הגנרית שלכם. אתם מוזמנים להשתמש בו ולשנותו כרצונכם.
   b. אתם מוזמנים לקמפל ולהריץ אותו ביחד עם הספריה שלכם.
   c. **אין להגיש קובץ זה.**
8.

5. חומר עזר:

1. את פתרון הבית ספר ניתן למצוא ב:

~plabcpp/www/ex3/schoolSol.tar

2. את הקבצים המסופקים לצורך התרגיל ניתן למצוא ב:

~plabcpp/www/ex3/ex2_files.tar

3. קבצי בדיקה לדוגמא ניתן למצוא ב:

~plabcpp/www/ex2/tests_examples.tar

4. ביצוע overloading ב-C++:
5. ניתן לבצע redirection של הקלט/פלט באמצעות המתודות rdbuf() של cout/cin.
לדוגמא, קטע הקוד הבא מבצע redirection ל-standard input מקובץ קלט:

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/

6. הגשה:

1. עליכם להגיש קובץ tar בשם ex3.tar המכיל את כל הקבצים הנדרשים לקימפול התוכנית שלכם ואת הקבצים הבאים:
   - README עם התשובות לשאלות של העיבוד המקבילי.
   - קובץ Makefile התומך לפחות בפקודות הבאות:
     - make Matrix - יצירת Matrix.hpp.gch.
     - make clean - ניקוי כל הקבצים שנוצרו באמצעות פקודות ה-makefile.
     - הרצת make ללא פרמטרים תייצר קובץ ריצה בשם GenericMatrixDriver באמצעות הדרייבר שסופק לכם והמימוש שלכם.
   - extension.pdf - רק במקרה שההגשה היא הגשה מאושרת באיחור (מכיל את האישורים הרלוונטים להארכה).
   - שימו לב! - אל אף שאתם יכולים להוסיף קבצים נוספים כרצונכם, המנעו מהוספת קבצים לא רלוונטים (גם בכדי להקל על הבודקים, וגם בכדי שציונכם לא יפגע מכך).

אין להגיש את הקבצים הבאים:

Complex.h, Complex.cpp, GenericMatrixDriver.cpp, ParallelChecker.cpp

2. לפני ההגשה, פתחו את הקובץ ex3.tar בתיקיה נפרדת וודאו שהקבצים מתקמפלים ללא שגיאות וללא אזהרות.
3. מומלץ מאוד גם להריץ בדיקות אוטומטיות וטסטרים שכתבתם על הקוד אותו אתם עומדים להגיש. בנוסף, אתם יכולים להריץ בעצמכם בדיקה אוטומטית עבור סגנון קידוד בעזרת הפקודה:

~plabcpp/www/codingStyleCheck <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)

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

~plabcpp/www/ex3/presubmit_ex3

בהצלחה!