תרגיל 2 – slabcpp
Inheritance, STL, Operators and Copies

תאריך הגשה: יום רביעי, 3.9.2014 עד שעה 23:55
תאריך הגשה מאוחרת (בהפחתה של 10 נקודות): יום חמישי, 4.9.2014 עד שעה 23:55

הנחיות כלליות

- בתרגיל זה הינכם רשאים להשתמש ב STL, לדוגמא map ו vector יכולים לעזור. ראו גם את קבצי Map.h, Vector.h אשר מהווים typedef, ויכולים לשמש אתכם.
- במידה והנכם משתמשים בהקצאות זיכרון דינמיות, הניחו כי הן מצליחות תמיד. מקובל להתמודד עם כישלון הקצאות זיכרון ב-++C באמצעות מנגנון ה-exceptions אותו תתרגלו בתרגיל הבא.

משימת התכנות

בתרגיל זה נממש ייצוג של מטריצה ממשית ועליה פעולות אלמנטריות.
במסגרת השאלה תתרגלו העמסת אופרטורים שונים בשפה, ותתכננו את המבנה הפנימי של מחלקת ייצוג המטריצה.

המחלקה MyMatrix תייצג מטריצה ממשית מגודל כלשהו, ממשו מחלקה זו בקבצים:

- MyMatrix.cpp
- MyMatrix.h

קבצי התרגיל נמצאים באתר הקורס ובלינק:

~slabcpp/ex2/ex2_files.tar

אופרטורים ופונקציות

1. חיבור וחיסור מטריצות מגודל זהה של MxN, על ידי מימוש האופרטורים: =- , =+ , - , +
שימו לב שניתן לממש את אופרטור '=+' ע"י אופרטור '+' (וטענה דומה תקפה גם עבור אופרטור הכפל והחיסור). מימוש האופרטורים '+' ו- '=+' באופן נפרד הוא שכפול קוד, שלא לדבר על העבודה המיותרת הרבה שאתם מוסיפים לעצמכם.

2. כפל מטריצות בגדלים NxM ו-MxL, על ידי מימוש האופרטורים: =* , *
סיבוכיות: בתרגיל זה אנו נסתפק באלגוריתם הטריוויאלי לביצוע כפל ב-$O(n \cdot m \cdot l)$.

3. האופרטור == (אופרטור השוואה): נגדיר שני מטריצות כזהות אם כלל האיברים של שתיהן שווים עד כדי אפסילון, כאשר אפסילון קטן שווה ל-10^{-6}.

4. אופרטור מינוס (-) אונארי.

5. אופרטור החדרה לפלט (<<) הכותב לתוך ה ostream הנתון מחרוזת המייצגת את המטריצה בייצוג כבדוגמא הבאה:
המטריצה :

$$\begin{pmatrix} a & b \\ -c & d \end{pmatrix}$$

ייוצג כך :

a , b\n
-c , d\n

הדברים שיש לשים לב אליהם בייצוג המטריצה:

- כל האיברים מופרדים על ידי סימן פסיק מרווח, זאת אומרת שיש רווח לפני הפסיק וגם אחריו.
- בסוף כל שורה יש ירידה לשורה חדשה ללא רווח

- גם בשורה האחרונה יש ירידת שורה
- יש לכתוב את סימן האיבר

עבור מטריצת האפס (שכל אבריה אפסים), יש להחזיר את הספרה 0 ואז ירידת שורה:
0\n

6. פונקציה בשם trace המחשבת את ה-trace של המטריצה. ה-trace מוגדר בתור סכום האיברים שעל האלכסון הראשי של מטריצה ריבועית. אם המטריצה אינה ריבועית, הפונקציה תחזיר 0.

7. פונקציה בשם frobeniusNorm המחשבת את נורמת פרוביניוס של המטריצה. נורמה זו מוגדרת להיות סכום הריבועים של כל איברי המטריצה.
טיפ: לנורמה זו יש גם דרך חישוב אחרת שתוכל למנוע כתיבת קוד מיותר על ידי שימוש במכפלת מטריצות ו-trace.

<u>בנאים (constructor)</u>

עליכם לממש בנאי העתקה, אופרטור השמה ו-destructor עבור המחלקות שלכם.

1. ממשו בנאי המקבל את גודל המטריצה ומערך חד מימדי המכיל את איברי המטריצה לפי סדר עמודות ובונה את המטריצה בהתאם:
```
(double arr[], unsigned int colSize, unsigned int rowSize);
```

סדר עמודות אומר שקוראים מערך דו-מימדי עמודה אחרי עמודה ומעתיקים אותו למערך חד מימדי.

2. ממשו בנאי המקבל מחרוזת (string) המייצגת מטריצה ובונה את אובייקט המטריצה לפי המחרוזת (למבנה המחרוזת ראו את מתודת ה >>operator לעיל).
הניחו כי בבנאי זה תקבלו רק קלט תקין.

מצורף קובץ DemoMyMatrix.cpp המכיל דוגמא לשימוש במחלקה MyMatrix.
וודאו כי המחלקה שלכם מתקמפלת עם קובץ זה וכי הפלט תקין.

<u>רמת דיוק</u>

- בעת כתיבת מספרים בפונקצית ההחדרה לפלט יש להדפיס בדיוק של 5 ספרות אחרי הנקודה העשרונית (כלומר לעגל עד הספרה החמישית אחרי הנקודה העשרונית).
אם ישנם אפסים בסוף, אין להדפיס אותם, ויש לחתוך את זנב המספר.
לדוגמא המספר 123.456789 יודפס כ 123.45679, והמספר 123.4000000 יודפס כ 123.4.
אנו ממליצים לכם להשתמש לצורך העניין במתודה ()stringstream.precision (ראו קישור למטה).
בשאר השיטות השתמשו בכל הייצוג של הערכים.
- בשל העובדה כי ייצוג המקדמים נעשה ב- double ומגבלות דיוק במחשב ייתכנו הבדלי פלט בין פתרונות שונים, הקפידו להריץ את הקוד במחשבי 64 ביט של CSE כדי לקבל תוצאות כמו בקבצי הבדיקה.
- שני מספרים שהפרשם קטן מ-10^{-6} יחשבו כשווים.

<u>ייצוג המטריצה</u>

כעת נתמודד עם השאלה כיצד תיוצג המטריצה במחלקה MyMatrix.
לאופן הייצוג של המטריצה יש השלכה הן על נוחות העבודה והן על סיבוכיות זמן הריצה והזיכרון אותו תצרוך המחלקה MyMatrix.

לצורך תרגיל זה, נסווג את המטריצות ל-2 סוגים.

- מטריצות רגילות: מטריצה רגילה היא כזו אשר רוב אבריה שונים מ- 0.
  לדוגמא: המטריצה $\begin{pmatrix} 4 & -45 \\ 3.45 & 2.9 \end{pmatrix}$ היא מטריצה רגילה.

- מטריצות "דלילות" (sparse): מטריצה דלילה היא מטריצה אשר יותר ממחצית אבריה הינם 0.
  לדוגמא: המטריצה $\begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix}$ היא מטריצה דלילה

במידה וחצי מהאיברים בדיוק הינם אפסים נגדיר את המטריצה כמטריצה רגילה.

לסוג המטריצה יש השלכה על אופן כדאיות ייצוגה במחלקה.

מטריצה רגילה כדאי לייצג על ידי מערך או ווקטור המכילים את איברי המטריצה.
נשים לב כי שיטה זו הינה נוחה למניפולציות אלגבריות. לדוגמא, כאשר אנו מעוניינים ליישם חיבור מטריצות נעבור על מערך האיברים ונחבר איבר-איבר לקבלת מטריצת התוצאה.

בניגוד למטריצות רגילות, שיטה זו אינה מתאימה לייצוג מטריצות דלילות בשל העובדה שהיא בזבזנית בזיכרון וגורמת לעבודה מיותרת.
מטריצות דלילות נהוג לייצג על ידי רשימה מקושרת של שלשות (a,m,n) כאשר a מייצג את ערך האיבר במטריצה ו-m,n את השורה והעמודה של אותו איבר.
לדוגמא, המטריצה דלעיל תיוצג על ידי השלשה (4,1,1)
שיטה זו הינה יעילה יותר וחסכונית בזיכרון עבור מטריצות דלילות כיוון שאיננו שומרים מידע על האיברים שערכם אפס וחוסכים מעבר על איברי ה- 0 בפעולות כגון כפל.

כאמור כל שיטה הינה יעילה לסוג מטריצה אחר. שימו לב כי מימוש אלגוריתמי הפעולות כחיבור, חיסור, כפל השוואה וכו', שונה לחלוטין בין שיטות הייצוג השונות כיוון שאופן ייצוג המטריצה שונה.

<u>מטרתכם</u>:

- המחלקה MyMatrix "תחליט" בזמן ריצה, לפי סוג המטריצה המיוצגת (על פי מספר המקדמים), באיזו שיטה לייצג את המטריצה באופן פנימי.

- שמרו את ייצוג המטריצה הרגילה בעזרת מערך או vector של STL (חד או דו מימדי).
  לרשותכם ה-Vector.h header בו יש שתי מחלקות לשימושכם:
  Vector שהוא typedef של vector של STL לערכים מסוג double.
  Vector2D שהוא וקטור של וקטורים של double.
  למטה תמצאו לינק לתיאור מחלקות אלו.

- הנכם רשאים לבחור כיצד לשמור את ייצוג רשימת השלשות במטריצה הדלילה. הנכם יכולים להשתמש לרשותכם ה-Map.h header בו יש שתי מחלקות לשימושכם:
  Pair שהוא typedef של std::pair לערכים של unsigned int ומייצג זוג של מספרים שלמים חיוביים.
  Map שהיא typedef של map של ה-STL. זהו מיפוי בין מפתח מטיפוס Pair (זוג של unsigned int) לערכים מטיפוס double.

  ראו דוגמא לשימוש בקובץ MapDriver.cpp.

- זכרו כי המשתמש במחלקה MyMatrix פועל מול ממשק (interface) יחיד.
  אופן הייצוג הפנימי של המטריצה צריך להיות "שקוף" למשתמש.

- שימו לב כי לאחר פעולות מסוימות כגון חיבור וחיסור, ייתכן כי מטריצה תשתנה מרגילה לדלילה או

להיפך – אם כך קורה, עליכם לשנות גם את ייצוגה הפנימי. עליכם להחליט לאחר אילו פעולות תבחרו לבדוק (ובמקרה הצורך לעדכן) את ייצוג המטריצה. הסבירו **בקצרה** בתיעוד המחלקה MyMatrix את האופן שבו בחרתם לבדוק זאת ואת המניעים לכך. כמו כן דאגו לתעד בכל פונקציה האם הבדיקה נעשית בה או לא ובאילו תנאים.

- מימוש שני הייצוגים באותה מחלקה וקריאה לקטעי קוד מתאימים באמצעות תנאים ומשפטי switch רבים נחשב סגנון גרוע כיוון שהוא מסורבל לתחזוקה והרחבה. שיטה נכונה יותר תהיה מימוש כל שיטת ייצוג במחלקה נפרדת.
עיצוב אפשרי: בניית 2 מחלקות SparseMatrix ו-RegMatrix, כאשר המחלקה MyMatrix תכיל אותן ותבצע החלפה ביניהן בזמן ריצה.

תבנית העיצוב Strategy היא דוגמא לדרך יפה ופשוטה למימוש "נקי" של הייצוגים והחלפה ביניהם בזמן ריצה.

כדי שתוכלו להשתמש בתבנית העיצוב ביעילות, צירפנו עבורכם דוגמא לקוד המשתמש בתבנית כזו. בקבצי התרגיל תמצאו ספרייה בשם strategy_example בתוכה קבצי קוד המדגימים שימוש בתבנית העיצוב וכן קובץ הרצה המראה את תוצאת הרצת ה-Interface החיצוני.

מעבר לדוגמא שניתנה לכם, ברשת קיימות הרבה דוגמאות ומאמרים למימוש תבנית עיצוב זו, בתור התחלה הינכם יכולים להעזר בקישורים המובאים בסוף התרגיל.

הרעיון: תכניתכם תכלול שתי מחלקות הממשות ממשק הכולל את הפעולות על מטריצה (שתי המחלקות יורשות ממחלקה יחידה). המחלקה MyMatrix תחליט בזמן ריצה באיזו מחלקה ספציפית להשתמש.

הערה: מימוש אופרטורים עבור המחלקה MyMatrix, בעזרת הפיכת אובייקט מ RegMatrix ל SparseMatrix (או הפוך), חישוב האופרטור, ואז המרה חזרה הינו מימוש מאד לא יעיל, ולא נכון מבחינת כתיבת תכנית ב ++C כך שיש להימנע מכך.
רמז: כדי לעבור על כל המקדמים של מטריצה ניתן להשתמש באיטרטור, כך אין תלות במימוש הפנימי של המטריצה.

- גודל המטריצה המקסימלית האפשרית הוא $M \times N < 10^8$

הסבירו בתיעוד באופן מפורט כיצד מימשתם את ייצוג המטריצות, ומבנה המחלקות של תכניתכם.

בדיקה

על הקוד שלכם להתקמפל (ללא שגיאות וללא- warnings) עם קובץ הדוגמא DemoMyMatrix.cpp. וההרצה צריכה להסתיים ללא הודעות שגיאה.
בדקו גם שאין לכם דליפות זכרון ע"י קימפול בעזרת הדגל g– והרצת valgrind.

הוראות הגשה

עליכם להגיש את הקובץ ex2.tar הכולל את הקבצים הבאים:

- MyMatrix.cpp MyMatrix.h וקבצים נוספים כפי הצורך.
- קובץ Makefile התומך באופציות הבאות:
  - all – קימפול כל קבצי המקור לקבצי אובייקט.
  - demo – יצירת קובץ ריצה בשם DemoMyMatrix מהקובץ DemoMyMatrix.cpp.
  - הרצת ברירת המחדל (הרצת Make ללא פרמטרים) המבצעת את שני הסעיפים הקודמים.

- clean – ניקוי כל קבצי ההרצה והאובייקטים שנוצרו על ידי ה-make.

- השתמשו בסקריפט ~slabcpp/public/ex2/presubmit_ex2
על מנת לוודא תקינות בסיסית של הקובץ אותו אתם עומדים להגיש.

<u>קישורים</u>

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/

המחלקה vector של STL:
http://www.cplusplus.com/reference/stl/vector/

המחלקה map של STL:
http://www.cplusplus.com/reference/stl/map/

תבנית העיצוב Strategy:
http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/strategy.html
http://sourcemaking.com/design_patterns/strategy/cpp/1
http://r3dux.org/2011/07/an-example-implementation-of-the-strategy-design-pattern-in-c/
http://en.wikipedia.org/wiki/Strategy_pattern

דיוק בפלט בעזרת stringstream.precision():
http://www.cplusplus.com/reference/iostream/stringstream/

נורמת פרוביניוס
http://mathworld.wolfram.com/FrobeniusNorm.html

בהצלחה!