תרגיל 3 – slabcpp
Template class, exceptions, Move Semantics and STL

תאריך הגשה: יום חמישי, 11.9.2014 עד שעה 23:55
תאריך הגשה מאוחרת (בהפחתה של 10 נקודות): יום שישי, 12.9.2014 עד שעה 14:00

שימו לב:

- עליכם לבצע את פקודת הקומפילציה עם הדגל Wall– על מנת לוודא שתכניתכם מתקמפלת ללא אזהרות. תכנית שמתקמפלת עם אזהרות תגרור הורדת נקודות.
- עליכם לוודא שהתרגילים תקינים ושהתכניות רצות על המחשבים של בית הספר, במערכות 64-bit (כמו המחשבים שבמעבדות לוי ואקווריום וכמו השרת river), כי תרגיליכם ייבדקו על מערכות כאלה וההתנהגות עלולה להיות שונה במערכות אחרות. לפני ההגשה וודאו שהכל עובד על מחשב של בית הספר. ניתן להתחבר מרחוק באמצעות SSH לשרת river.
- עליכם לוודא שתרגילכם עובר את ה presubmission script. תרגיל שלא עובר את הסקריפט יאבד נקודות רבות (קיראו את מדיניות הקורס להגשת תרגילים).
- בתרגיל זה עליכם **להימנע לחלוטין** משימוש ב-new ו-delete. היעזרו לשם כך בספריה הסטנדרטית

משימת תכנות – מטריצה גנרית:
בתרגיל זה תממשו מחלקה גנרית של מטריצה, כלומר איברי המטריצה הם מטיפוס גנרי.
המחלקה תוכל גם לשמש כקונטיינר לכל טיפוס שהוא (כפי ש std::vector ו std::list הגנריים יכולים להכיל איברים מכל טיפוס שהוא) שמייצגים 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.

הטיפוסים הספציפיים לאיברי המטריצה:
ניתן ליצור מטריצה מכל טיפוס אשר יש לו מימוש לארבעת האופרטורים = ,*, + ו >> וכן מימוש של בנאי ברירת מחדל היוצר את איבר האפס של המחלקה.
לדוגמא, כמובן שטיפוסים פרימיטיבים כדוגמת int עומדים בקריטריונים אלו.

לכל מחלקה יש דרך שונה לחישוב פעולות החשבון, לייצוג כמחרוזת, ואיבר אפס משלה.
לכל מחלקה (שאינה מטיפוס פרימיטיבי) יש בנוסף בנאי שמקבל מחרוזת כארגומנט. בנאי זה יהיה בשימוש הדרייבר, שקורא נתוני מטריצות מהמשתמש ומייצר מטריצות, ניתן יהיה להניח שהוא מקבל קלט תקין.
בנוסף יכולות להיות פונקציות ייחודיות לכל מחלקה.

עליכם לממש את המחלקה Rational בקובץ Rational.cpp לפי הממשק הנתון בקובץ Rational.h. זוהי מחלקת מספרים רציונלים אשר תוכלו לבדוק באמצעותה את המימוש שלכם למטריצה הגנרית. טיפוס זה מייצג מספר רציונלי באמצעות שני מספרים שלמים (מטיפוס long int), המונה והמכנה של השבר. נדרוש שלכל מספר רציונלי יהיה ייצוג יחיד קאנוני (מספר רציונלי צריך תמיד להיות מיוצג בצורתו הכי מצומצמת שאפשר, המכנה תמיד יהיה שלם וחיובי), לכן ישנם כמה כללים שמפורטים בקובץ ה-header, שעליכם למלא אותם.

הייצוג המחרוזתי של מספר רציונלי (גם עבור קריאה ממחרוזת בבנאי שמקבל מחרוזת וגם עבור הפונקציה שמחזירה ייצוג מחרוזת) הוא "[numerator]/[denominator]", זאת אומרת, עם קו שבר נטוי, ללא רווחים, ללא סוגריים מרובעים וללא הגרשיים.

ממשק המטריצה:

עליכם להגדיר ולממש את המחלקה הגנרית Matrix בקובץ Matrix.hpp.
המטריצה תהיה גנרית ואבריה לא יהיה בהכרח מספרים שלמים אלא מספרים מטיפוס גנרי.
לצורך נאותות, אתם רשאים להניח שהטיפוסים יהיו בעלי מימוש לאופרטורים הנדרשים כנמצא לעיל. כמו כן אתם יכולים להניח שהטיפוסים שיהיו בשימוש לאברי המטריצה הם כאלה שסדר הפעולות האטומיות של חיבור או כפל (בשרשרת פעולות חישוב ארוכה) לא משנה.
לדוגמא: (a + b) + c == a + (b + c)

לרשותכם דרייבר GenericMatrixDriver.cpp שמחבר את חלקי התרגיל ובודק את המחלקה הגנרית, באמצעות טיפוסים שונים.
בין הפונקציות שיש לממש (באופן גנרי, כמובן):

- בנאי ברירת מחדל (זאת אומרת בנאי ללא ארגומנטים). הבנאי בונה מטריצה של 1 על 1 עם איבר שערכו הוא ברירת המחדל של אותה מחלקה.
- בנאי שמקבל את מספר השורות, מספר העמודות ווקטור עם ערכי המטריצה למילוי (ראו את הקריאה לבנאי הזה מהדרייבר). יש לוודא את הקלט ולא ניתן להניח כי הוא יהיה תקין.
- בנאי העתקה (copy constructor) המקבל מטריצה אחרת.
- בנאי העברה (move constructor) המקבל מטריצה אחרת כ-rvalue.
- אופרטור השמה. יש להשתמש ב-copy-and-swap idiom לשם מימוש האופרטור על ידי שימוש ב-std::swap
- operator+ לחיבור מטריצות. אם מימדי שתי המטריצות שמבקשים לחבר לא מתאימים (תזכורת: בחיבור שתי המטריצות המחוברות צריכות להיות בדיוק באותם מימדים) על הפונקציה לאתר זאת ולזרוק exception ראו למטה.
- operator* לכפל מטריצות (המטריצה של האובייקט עליו מפעילים את הפונקציה היא המטריצה השמאלית בכפל). גם כאן יש לוודא שמימדי המטריצות מתאימים (תזכורת: בכפל מספר העמודות במטריצה השמאלית צריך להיות זהה למספר השורות במטריצה הימנית) ואם המימדים לא מתאימים, לזרוק exception.
- פונקצית שחלוף מטריצה (transpose).
- פונקצית עקבה (hasTrace) המקבלת reference לאיבר מהטיפוס הגנרי ומשימה בו את ערך העקבה של המטריצה ומחזירה ערך בוליאני: true אם המטריצה ריבועית, ו false אחרת ובמקרה כזה תשים את ערך איבר האפס באיבר שהתקבל כ reference.
- ארבעת הפונקציות הנ"ל (חיבור, כפל transpose ו hasTrace) לא משנות את האובייקט עליו הופעלה הפונקציה, אלא מייצרות אובייקט חדש שיוחזר.

בנוסף תוכלו להוסיף עוד פונקציות ציבוריות או פרטיות כרצונכם, לפי מה שנראה לכם שימושי למחלקה.

בתרגיל זה הממשק (החתימות של הפונקציות) מוכתב לכם חלקית ע"י הקריאות לפונקציות הללו מהדרייבר, אבל הוא לא מוכתב לגמרי ועדיין יש כמה החלטות שעליכם לעשות. חישבו למשל על:

- לכל פונקציה אם היא צריכה להיות מוגדרת כ const (לא משנה את האובייקט עליו היא מופעלת).
- לכל ארגומנט אם הוא צריך להיות מועבר by reference או מועתק, או אולי כדאי להשתמש במצביע.
- לערך ההחזרה, האם הוא צריך להיות מוגדר כ const, והאם הוא מועבר by reference או מועתק.
- החריגות שאתם זורקים, צריכות לרשת מ-std::exception ולהיות אינפורמטיביות. לדוגמא, אל תזרקו את המספר 2 או את המחרוזת "משהו רע קרה בגלל...".

התמחות (specialization) בנוסף לממשק הגנרי:
בנוסף למימוש הגנרי של המחלקה Matrix, עליכם להוסיף לקובץ Matrix.hpp גם מימוש ספציפי אחד: מימוש אלטרנטיבי לפונקציה trace, המחשבת את העקבה של המטריצה, עבור המקרה בו הטיפוס של האיברים הוא Rational.

ההיגיון מאחורי ההתמחות:
שימו לב שבמחלקת הטיפוס Rational, לאחר כל ביצוע פעולת חישוב אטומית של חיבור שני רציונלים, מתבצע צמצום השבר, וכדי לבצע זאת צריך לחשב את המחלק המשותף הגדול ביותר של המונה והמכנה (GCD - greatest common divisor). חישוב ה GCD אינו בזמן קבוע, אלא בזמן שתלוי במספרים עצמם ולעיתים יכול להיות מאד ממושך.

במימוש הגנרי של trace יש צורך להשתמש באופרטור + של הטיפוס הגנרי הרבה פעמים (כמספר אברי האלכסון של המטריצה). אם המימוש הגנרי של trace יופעל על מטריצה שאבריה הם מטיפוס Rational, יתבצעו הרבה חישובי ביניים של GCD, שיקחו זמן חישוב רב.

לכן עליכם להוסיף מימוש ספציפי של trace במיוחד עבור מטריצה שאבריה הם מטיפוס Rational, ובמימוש זה עליכם לעקוף את חישובי הביניים הרבים של ה GCD, בכך שלא תקראו לאופרטור + של Rational עבור כל איבר נוסף שמחברים. במקום זה עליכם להשתמש בפונקציות getNumerator() ו getDenominator() של Rational, ולעשות את החישוב והחיבור של השברים מכל אברי האלכסון בעצמכם, ללא צמצומי הביניים. באופן מצטבר (אתם רשאים להניח שערכי המונה והמכנה המצטברים לא יעברו את הערך המקסימלי של long int), ורק בסוף החישוב יתבצע הצמצום (ע״י המימוש הספציפי שלכם, או ע״י הבנאי של Rational שייצר את המספר הרציונלי של התוצאה הסופית של חישוב העקבה).

כדי לתת סימן לפלט של התכנית, שאכן מופעל המימוש הספציפי של trace, עליכם להוסיף הדפסת שורת הודעה בתחילת הפונקציה המתמחה:

Performing specialized function of trace for Rational values

ראו את הפלט לדוגמה של פתרון בית הספר והשוו בעזרת diff כדי לוודא שהמחרוזת אותה אתם מדפיסים היא נכונה.

דרייבר:
לרשותכם נתון דרייבר בשם GenericMatrixDriver.cpp. הוא בודק את המימוש הגנרי שלכם ל Matrix באמצעות טיפוסים שונים ובדרך עקיפה גם בודק את המימוש שלכם למחלקה Rational.
קמפלו את התכנית לכדי קובץ ריצה GenericMatrixDriver.
הריצו את התכנית כדי לראות איך היא עובדת, איזה קלט היא מצפה לקבל מהמשתמש.

בדיקה:
לרשותכם קבצי קלט ופלטים מתאימים של פתרון בית הספר, עבור התכנית GenericMatrixDriver שקומפלה עם הדרייבר הנ״ל ועם פתרון בית הספר למחלקה הגנרית בעזרת טיפוסים שונים.
וודאו שהפלטים של תכניתכם זהים לפלטים של פתרון בית הספר.
כתבו קבצי קלט נוספים לבדיקת תרחישים נוספים. אתם יכולים להשוות את הפלטים שלכם ביניכם.
אתם יכולים להוסיף מחלקות נוספות כדוגמת Complex שראיתם בכיתה, ולנסות בעזרתה את מימוש המטריצה.

כדי לוודא שהמימוש שלכם ל Matrix אכן גנרי ותקין, אתם יכולים להגדיר מחלקות טיפוסים נוספות שיתארו מספרים מסוגים נוספים (שעליהם להוות קבוצה סגורה לחיבור ולכפל ולשמור על חוק החילוף חוק הקיבוץ וחוק הפילוג) ושיהיו להן הפונקציות הציבוריות הדרושות. כך תוכלו גם

להרחיב את הדרייבר, שידע לעבוד עם טיפוסים נוספים וליצור מטריצה גנרית עם טיפוסי-איברים נוספים.

זכרו שגם הבדיקה של התרגילים תוכל לבדוק את מחלקת Matrix שלכם עם דרייבר מורחב ועם טיפוסי איברים נוספים.

קובץ Makefile:
על קובץ ה-`Makefile` ליצור את הבינארי `GenericMatrixDriver`.
מומלץ כי תכללו גם מספר הרצות לדוגמא של הבינארי עם קלטים שונים ובדיקות לדליפת זיכרון.

הגשה:
עליכם להגיש קובץ `tar` בשם `ex3.tar` שמכיל את הקבצים הבאים:

- Makefile
- Matrix.hpp
- Rational.cpp
- Other “number” classes you implemented

דאגו לבדוק שקובץ ההגשה שלכם עובר את ה `presubmission script` ללא שגיאות וללא אזהרות. פתחו את הקובץ בתיקיה נפרדת וודאו שכל התכניות מתקמפלות ללא שגיאות וללא אזהרות.
כמו-כן, הריצו את הכלי לבדיקת ה `coding style`.

בהצלחה!

References

Move Semantics:
http://www.cprogramming.com/c++11/rvalue-references-and-move-semantics-in-c++11.html

http://stackoverflow.com/questions/3106110/what-are-move-semantics

http://msdn.microsoft.com/en-us/library/dd293665.aspx

http://blog.smartbear.com/c-plus-plus/c11-tutorial-introducing-the-move-constructor-and-the-move-assignment-operator/