בס"ד

תרגיל בית 5 (שפת ++C) - Templates

תאריך הגשה: 22.5.16

הנחיות חשובות:

- בתרגיל זה מותר לכם להשתמש בקבצי הקוד שלכם בספריות הסטנדרטיות הבאות בלבד:

```
#include <cassert>
#include <vector>
#include <cstdlib>
#include <cmath>
```

- בתרגיל זה אסור להשתמש ב new ו- delete או בהקצאה דינמית אחרת באופן ישיר.
- שימו לב שלקבצים מצורפים קבצי קוד וההוראות בקבצים הללו מחייבות!

הנחיות חשובות לכלל התרגילים מעתה והלאה בקורס:

- התרגילים הם לעבודה ביחידים. מותר להתייעץ אך ורק בעל פה, אסור בתכלית האיסור שחומר כתוב/מודפס/אלקטרוני יעבור בין אנשים. בנוסף, על חלק מהתרגילים תיבחנו פרונטלית ועליכם להבין כל דבר בקוד!
- **הימנעו ממספרי קסם**: מספרים ממש שמופיעים באמצע הקוד בלי משמעות מיוחדת (לדוגמא נניח שמספר הרשומות בתרגיל אחר הוא מקסימום 50 ואז בכל מקום בקוד כתוב 50. לעומת זאת, 0 לתחילת מערך לא נחשב מספר קסם - הפעילו הגיון בריא) והשתמשו במקום זאת בפקודות מאקרו (#define או אם כבר למדתם על כך ב const).
- אין להשתמש ב-variable length arrays , וכן במשתנים סטאטיים ו / או גלובליים. כל התרגילים בקורס צריכים להתקמפל ולרוץ באתר c9.io (האתר מריץ מערכת הפעלה אובונטו) עם שורת הקמפול:

```
gcc -Wall -Wvla -Werror -g  …
```

ושורת הקמפול עבור תרגילי ++C:

```
g++ -Wall -Wvla -Werror -g -D_GLIBCXX_DEBUG -std=c++11 …
```

או במידה ומצורף Makefile עם ה Makefile המצורף.

- יש להקפיד על סגנון תכנות טוב כמו שלמדתם. לדוגמא, להימנע מחזרות קוד (לכתוב פונקציות שצריך), שמות משתנים עם משמעות, בהירות הקוד, תיעוד הקוד, להקפיד להשתמש בקבועים שצריך ולא במספרי קסם וכו'.
- <u>אופן כתיבת ההערות</u>: בכל תחילת קובץ הצהרות (h. או hpp.) יש לכתוב את תפקיד הקוד שבקובץ.כמו-כן לפני כל פונקציה ומתודה ומשתנה מחלקה/מבנה יש לכתוב הערה על תפקידם.יש להוסיף הערות במימוש לפי הצורך.

בקורס זה במידה וניתנו לכם הוראות מפורטות לגבי פונקציה בתרגיל מותר לכתוב ראה בהגדרת התרגיל במקום לעשות copy&paste.

- הקפידו להשתמש בפונקציות של ++C (למשל new, delete, cout) על פני פונקציות של C (למשל malloc, free, printf). בפרט השתמשו במחלקה string ולא במחרוזת של C (כלומר, *char).
- יש להשתמש בספריות סטנדרטיות של ++C ולא של C אלא אם כן זה הכרחי.
- הקפידו על עקרונות Information hiding. לדוגמא, הקפידו להגדיר משתני מחלקות כ private.
- הקפידו לא להעביר אובייקטים גדולים by value אלא by reference או by const reference.
- הקפידו מאד על const correctness. כלומר, על שימוש במילה השמורה const בצורה נכונה.
- שאלות על התרגיל יש לשאול בפורום המתאים במודל בלבד!

בס"ד

התרגיל:

משימת תכנות – מטריצה גנרית:

בתרגיל זה נממש מטריצה גנרית, היכולה להחזיק איברים מכל טיפוס שהוא, בדומה למבני הנתונים std::vector ו-std::list הגנריים.
על האובייקט של המטריצה יהיה אפשר לבצע פעולות חשבוניות שונות, כפי שיפורט בהמשך.

ממשק מחלקת המטריצה:

את מחלקת Matrix<T> יש להגדיר ולממש בקובץ Matrix.hpp.

הממשק יכלול **לפחות** את המתודות הבאות:

1. **Default Constructor** – בנאי ללא פרמטרים, הבונה מטריצה של 1x1.
2. **Matrix(size_t, size_t, std::vector<T>)** - בנאי המקבל כפרמטר את מספר השורות, מספר העמודות, ווקטור עם ערכי המטריצה למילוי. (ראו את הקריאה לבנאי הזה מהטסטר).
3. **Destructor**.
4. **Copy Constructor**, בנאי העתקה, הבונה מטריצה אחת כהעתק של מטריצה אחרת נתונה.
5. **אופרטור השמה**.
6. **operator+** הממש חיבור של שתי מטריצות.
7. ***operator** לכפל מטריצות. המטריצה של האובייקט עליו מפעילים את הפונקציה היא
8. המטריצה השמאלית בכפל.
9. ***operator** לכפל המטריצה בסקלר, כשהסקלר מצד ימין (כל איבר במטריצה מוכפל בסקלר).
10. **transpose()** – פונקציית שחלוף מטריצה.
11. **hasTrace(T&)** – פונקציית עקבה, המקבלת איבר גנרי ומשימה בו את ערך העקבה (אם קיים) של המטריצה. ערך עקבה קיים רק אם המטריצה ריבועית, לכן הפונקציה תחזיר ערך בוליאני: true אם המטריצה ריבועית, false אחרת. במידה והמטריצה לא ריבועית, הערך שיושם בפרמטר הוא ערך איבר האפס.
12. **getColNum** – מחזירה את מספר הטורים.
13. **getRowNum** – מחזירה את מספר השורות.
14. **isSquareMatrix** – פונקציה בוליאנית המחזירה true רק אם המטריצה ריבועית (מספר השורות שווה למספר העמודות).
15. **operator==** אשר מחזיר true אם שתי המטריצות שוות בגודלן ובאיבריהן.
16. **operator!=** מחזיר true אם המטריצות אינן זהות.

בנוסף לממשק הבסיסי הזה, אתם יכולים להוסיף עוד מתודות כרצונכם.
מה שמופיע בטסט הוא חלקי ביותר, ועליכם להגיש ממשק שלם ומדויק, גם מבחינת שימוש ב-const, טיפוסים מוחזרים וכד'. בתרגיל זה הממשק (החתימות של הפונקציות) מוכתב לכם חלקית ע"י חלק מקריאות לפונקציות הללו מהטסט, אבל הוא לא מוכתב לגמרי ועדיין יש כמה החלטות שעליכם לעשות.

אם יש מתודה שיש לה מימוש שניתן על ידי הקומפיילר והוא מספק אתכם, **אין להגדיר אותה מחדש**.

בדקו שהתוכנית שלכם **בטוחה ונכונה מבחינה מתמטית**.

תכונות המחלקה T:

האיברים של המטריצה יכולים להיות מכל טיפוס, אך עליהם לתמוך בממשק הבא כדי שיוכלו להיות איברים במטריצה Matrix<T> שהוגדרה לעיל:

1. תמיכה באופרטורים של חיבור, חיסור והשמה.
2. default constructor היוצר את איבר האפס (אובייקט מאופס, כל סוג אובייקט והגדרת האיפוס שלו).
   עבור טיפוסים בסיסיים ראו את הקישור הבא כדי לדמות התנהגות שדומה להפעלת constructor.

כמובן שהמחלקה יכולה להכיל פונקציות נוספות לשימושה, אך מחלקת Matrix<T> מניחה של-T יש לפחות את המתודות הללו.

מחלקת Rational – כדוגמא לטיפוס של איבר במטריצה

עליכם לממש את המחלקה Rational לפי הממשק הנתון בו בקובץ Rational.hpp המצורף לתרגיל.
זוהי מחלקה המייצגת מספרים רציונליים.
בעזרת מחלקה זו תוכלו לבדוק את המימוש שלכם ל-Matrix<T>.

המחלקה מחזיקה שני מספרים מטיפוס long int שיהוו מונה ומכנה של השבר.
נדרוש שלכל מספר רציונלי יהיה ייצוג יחיד קנוני (כלומר מיוצג בצורה המצומצמת ביותר כשהמונה והמכנה שלמים). יש כמה כללים שמפורטים בקובץ ה-header.
המחלקה צריכה להמיר בין מספר רציונלי ובין מחרוזת, **לשני הכיוונים**, כשהייצוג כמחרוזת הוא "nominator/denominator" (בלי ה-"" ובלי רווחים).

טסטרים:

בקובץ zip יש כרגיל קבצי Test שצריך להשלים ולהגיש.

בדיקת פרמטרים של פונקציות בעזרת myassert:

בתרגיל זה עליכם לבדוק את הפרמטרים של הפונקציות בעזרת המקרו **myassert** המוגדר בקובץ myassert.hpp. שימו לב שמותר לכם להשתמש ב assert רגיל לבדיקת לוגיקה של התוכנית שלכם אך אתם מחויבים להשתמש ב myassert על מנת לבדוק פרמטרים של פונקציות, ואנו נשנה את המקרו על מנת לבדוק את התוכנית שלכם.

פרטים שעליכם לבדוק:

- המכנה של שבר הוא אף פעם לא אפס.
- ממדי מטריצות בחיבור וכפל הם נכונים.
- מספר השורות והעמודות במטריצה צריך להיות גדול או שווה לאחד.

הנחיות נוספות

- אתם יכולים להוסיף מחלקות נוספות כדוגמת Complex שראיתם בכיתה ולבדוק באמצעותה את המטריצה שמימשתם.
- זכרו שהבדיקה של התרגילים תבדוק את מחלקת Matrix שלכם עם טיפוסי איברים נוספים.
- חובה כרגיל להגיש גוגל טסט של התוכנית
- בדיקת הקוד לפני ההגשה, גם על ידי קריאתו וגם על ידי כתיבת בדיקות אוטומטיות (כמו שכתוב בסעיף הקודם, חובה להגיש גוגל טסט) היא אחריותכם. חישבו על מקרי קצה, חלק מהציון ניתן על עמידה בבדיקות אוטומטיות.

הערה: מצורף קובץ Makefile שיש כרגיל לעמוד בדרישות שלו!

הנחיות הגשה

עליכם להגיש קובץ ששמו מספר ת.ז. שלכם כמו שהיא מופיעה באתר המודל נקודה zip. לדוגמא, אם מספר ת.ז. שלי הוא 12345678 אז שם הקובץ יהיה:
12345678.zip
הקובץ יכיל את הקבצים הבאים בלבד (שנו לתעודת הזהות שלכם את ID ב Makefile והשתמשו ב- `make zipfile` וב- `make checkzipfile` על מנת ליצור את קובץ הזיפ ולבדוק אותו):

- Matrix.hpp
- Rational.hpp
- MatrixTest.cpp
- RationalTest.cpp

בהצלחה!!!

בס"ד

שאלות תיאורטיות

על שאלות כאלו וכדוגמתן תצטרכו לענות בבחינה הפרונטלית (בנוסף לשאלות על הקוד שהגשתם).
שימו לב: אין צורך להגיש תשובות לשאלות הללו

שאלה 1

מתכנת קיבל ספרייה הממשת מערך של מספרים. יש לו את קובץ ה header ואת הקובץ הבינארי שמממש אותו אך אין לו גישה לקוד עצמו. להלן חלק מקובץ ה header:

```
class DoubleArray {
public:
      // This method returns the sum of the doubles in the DoubleArray
      // Running time: O(N), where:
      // N is the numbers of doubles in the DoubleArray
      double sum() const;

      // returns a unique id for each DoubleArray. Starts from zero and
      // advanced by one each time.
      // 0,1,... for the first, second, ... created DoubleArray
      unsigned int getId() const;
      // more code
      ...
};
```

המתכנת יצר מערך המכיל M מערכים כאלו (DoubleArray) כ"א מהם מכיל בדיוק N מספרים.
לכל אחד מהמערכים הללו יש מספר ייחודי בין 0 ל-(M-1) שניתן לגשת אליו דרך המתודה getId.
הוא כתב גם פונקציית השוואה, שמשווה בין 2 מערכים כאלו, לפי הסכום שלהם.
להלן הקוד הרלוונטי:

```
bool DoubleArrayComparatorBySum(const DoubleArray& a, const DoubleArray& b) {
      return (a.sum()<b.sum());
}

#include <algorithm>
int main() {
      // Creates M DoubleArray objects
      DoubleArray arr[M];
      // code that adds N Double’s to each DoubleArray
      ...
       std::sort(arr,  arr + M,  DoubleArrayComparatorBySum;)
       // other code
       ...
}
```

כאשר הוא ביצע בדיקת זמן ריצה הוא קיבל שזמן הריצה הוא בממוצע `O(M*log(M)*N )` ,
וזאת מכיוון שבכל פעם שמתבצעת השוואה בין שני מערכים, יש צורך במעבר על כל ה-N איברים על מנת לסכום אותם.

כתוב קוד שיוריד את זמן הריצה של המיון ל `O(M*(log(M)+N ))`.

הערות:

- לא ניתן לשנות את הקוד של המחלקה DoubleArray מכיוון שהוא לא ברשותך.
- אסור לרשת מהמחלקה (כי זה מצריך שינויים במקומות רבים בקוד אח"כ).
- יש להשתמש ב sort של המחלקה הסטנדרטית ולא לממש בעצמכם.
- אפשר להניח שלאחר שנוצרו M המערכים בשורה הראשונה של ה-main לא נוצרים עוד מערכים כאלו ולא נהרסים מערכים כאלו עד אשר יוצאים מה main.
  דהיינו, התבצעה specialization של swap ל DoubleArray ולכן ההחלפה מבוצעת ב O(1) ללא כל קריאה ל copy ctor, ולכן לכל אחד מהמערכים לאורך כל התוכנית יש מספר ייחודי בין ל (M-1).

שאלה 2

האופרטור [ ] מקבל בקוד הבא שתי הגדרות חדשות. הנח ששתי הגרסאות הללו מוגדרות במחלקת template<class T> class Array.
מה ההבדל בין שתי הפונקציות?
מה נאבד אם לא נרשום גם את הפונקציה השנייה במחלקה?

```
T& operator[ ](int index);
const T& operator[ ] (int index) const;
```

שאלה 3

מדוע מגדירים את כל הפונקציות שהן template בקובץ ה-hpp ביחד עם ההצהרה? האם חייבים את כל הפונקציות לממש ב-hpp או שאפשר חלק לממש ב-cpp? מדוע?

שאלה 4

מנה 2 חסרונות עיקריים של שימוש ב-template.