

234124 – מבוא לתכנות מערכות

תרגיל בית 3

סמסטר אביב תש"ף

תאריך פרסום: 2 ביוני

תאריך הגשה: 2 ביולי (23:55)

משקל התרגיל: 12% מהציון הסופי (תקף)

מתרגלים אחראים: רון קנטורוביץ' ואפרת לבקוביץ'

1 הערות כלליות

- את התרגיל יש לכתוב ולהגיש בזוגות.
- שאלות בנוגע להבנת התרגיל יש לשאול בפורום הקורס ב-moodle או בשעות הקבלה של המתרגלים האחראים על התרגיל. לפני שליחה של שאלה – נא וודאו שהיא לא נענתה כבר ב-FAQ או במודל, ושהתשובה אינה ברורה ממסמך זה, מהדוגמאות שסופקו ומהבדיקות שיפורסמו עם התרגיל.
- **חובה** להתעדכן בעמוד ה-FAQ של התרגיל.
- כל חומר נלווה לתרגיל נמצא על השרת בתיקיה `~mtm/public/1920b/ex3`
- קראו את התרגיל עד סופו לפני שאתם מתחילים לממש. ייתכן שתצטרכו להתאים את המימוש שלכם לחלק עתידי בתרגיל. תכננו את המימוש שלכם ביסודיות לפני שאתם ניגשים לכתוב קוד.
- מומלץ מאוד לכתוב את הקוד בחלקים קטנים, לקמפל כל חלק בנפרד על השרת ולבדוק שהוא עובד באמצעות שימוש בטסטים (אין צורך להגיש טסטים כנ"ל). אנו מעודדים אתכם לחלוק טסטים עם חברים.
- העתקות קוד בין סטודנטים תטופלנה בחומרה. אף על פי כן, מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.
- שימו לב: **לא יינתנו דחיות במועד התרגיל, פרט למקרים מיוחדים**. תכננו את הזמן בהתאם.
- אי שמירה על Code Conventions תגרור הורדה בציון.
- **אין להשתמש במבני STL לכל אורך התרגיל, אלא אם צוין אחרת.**
- תעדו את הקוד שלכם – בפרט ובמיוחד את החלקים הפומביים של המחלקות ואת פונק' הממשק שאתם מממשים.

תרגיל זה יתפרסם ב-3 חלקים שיוגשו ביחד. החלק הראשון יתמקד בהכרה ראשונית של השפה, החלק השני יתמקד בתבניות ויכלול חלק יבש, ולבסוף החלק השלישי יתמקד בירושה, בפולימורפיזם ובחריגות. הוראות ההגשה בסוף המסמך.

הפתרונות לכל חלקי הרטוב צריכים להיכלל תחת namespace בשם mtm.

הערה: בדוגמאות לאורך החלק הרטוב נניח שבראש הקובץ מופיעות השורות

```
#include <iostream>
using std::cout;
using std::endl;
```

כאשר קוד הדוגמה נמצא בגוף פונק' (למשל main). שימו לב: אין להשתמש ב-using בקבצי header! שימוש כזה כופה את אשר מוצהר ב-using על המשתמש שעושה #include ל-header.

2 חלק יבש

שאלה 1

סעיף א

נתונה המחלקה B ופונקציית main. בקוד הבא מסומנות 3 שורות בהן קיימות שגיאות הידור – שורה 13, שורה 22 ושורה 30. לכל אחת מהשורות המסומנות:
1. הסבירו מהי השגיאה.
2. הסבירו כיצד ניתן לשנות את הקוד כך שהשגיאה תתוקן.

```
1 #include <iostream>
2 using namespace std;
3
4 class B {
5 private:
6     int n;
7 public:
8     B(int x) : n(x) {}
9     B operator +(B& b) {
10         return B(n+b.n);
11     }
12     friend ostream& operator <<(ostream &out, const B& b) {
13         out << "B: " << n;
14         return out;
15     }
16     bool operator <(const B& rhs) {
17         return n < rhs.n;
18     }
19 };
20
21 B smaller (const B& b1, const B& b2) {
22     if(b1 < b2)
23         return b1;
24     else
25         return b2;
26 }
```

```

27
28 int main() {
29     B b1(1), b2(2), b3(3);
30     const B b4 = b1 + (b2 + b3);
31     cout << smaller(b1,b2) << endl;
32     return 0;
33 }

```

סעיף ב

מה תדפיס התוכנית הבאה: (כתבו את הפלט **והסבירו את שרשרת הקריאות** שמייצרת כל שורה).

```

#include <iostream>
using namespace std;

class A {
public:
    A() {}
    A(const A& a) { cout << "A copy ctor" << endl; }
    virtual ~A() { cout << "A dtor" << endl; }
    virtual void type() const { cout << "This is A" << endl; }
};

class B: public A {
public:
    virtual ~B() { cout << "B dtor" << endl; }
    void type() const override { cout << "This is B" << endl; }
};

A f(A a) {
    a.type();
    return a;
}

const A& g(const A& a) {
    a.type();
    return a;
}

int main()
{
    A* pa = new B();
    cout << "applying function f:" << endl;
    f(*pa).type();
    cout << "applying function g:" << endl;
    g(*pa).type();
    delete pa;
    return 0;
}

```

שאלה 2

עבור תכנת ניווט חדשה, אנו מעוניינים לחשב את צריכת הדלק של מסלולי נסיעה שונים. צריכת הדלק במסלול מסוים תלויה במהירות הנסיעה ובסוג הרכב הנוסע. מסלול מורכב מאוסף של דרכים, כאשר כל דרך ממומשת ע"י המחלקה Road להלן, ומתאפיינת באורך ובמהירות הנסיעה.

```

class Road {
public:
    double length(); // km
    int speed(); // km per hour
};

```

אנו מעוניינים לממש את הפונקציה

```
double getPetrol(std::vector<Road> roads, const Car& car);
```

אשר מקבלת מסלול ורכב, ומחשבת את צריכת הדלק של הרכב במסלול הנתון.
צריכת הדלק משתנה בין רכבים שונים, ועבור אותו הרכב – נקבל צריכה שונה במהירויות שונות.

סעיף א

כתוב/י מחלקה אבסטרקטית Car, אשר מחייבת את כל המחלוקות היורשות ממנה לממש פונק' בשם getFuelConsumption המקבלת מספר שלם, המייצג מהירות (בקמ"ש), ומחזיר את צריכת הדלק (מספר ממשי, בקילומטר לליטר) באותה המהירות.

סעיף ב

ממש/י את הפונקציה getPetrol, המחשבת את סך צריכת הדלק במסלול.

3 חלק א'

בתרגיל זה תממשו מחלקות לייצוג אובייקטים אלגבריים ותוסיפו להן תמיכה בפעולות בסיסיות. בחלק הראשון תממשו מחלקה לייצוג מטריצה שאיבריה מספרים שלמים. מסופק לכם קובץ בשם Auxiliaries המכיל מחלקה בשם Dimensions. מחלקה זו מכילה זוג מספרים המייצגים מימדי מטריצות ווקטורים וכמו כן פונקציות שימושיות.

טיפול בשגיאות: מכיוון שחריגות, שהן הדרך הסטנדרטית בשפת C++ להתמודדות עם שגיאות, טרם נלמדו – יש להניח בחלק זה שהפונקציות ייקראו עם קלטים תקינים (למשל – פעולות בין זוגות מטריצות ייקראו רק בין מטריצות בגדלים מתאימים).

3.1 מחלקת IntMatrix

מחלקה זו מתארת מטריצה בסיסית עם הפונקציות המפורטות להלן. עליכם להסיק חלק מחתימותיהן על סמך דוגמאות הקוד המצורפות:

3.1.1. בנאי

מייצר מטריצה שממדיה הם לפי הארגומנט הראשון, ושכל איבריה מאותחלים לארגומנט השני. אם לא סופק ארגומנט שני כל הערכים של המטריצה יאותחלו לערך 0. לדוגמא, הקוד הבא:

```
Dimensions dims(2,4);
IntMatrix mat_1(dims, 5);
IntMatrix mat_2(dims);
```

ייצור את המטריצות:

$$\text{mat_1} = \begin{pmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{pmatrix} \quad \text{mat_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3.1.2. בנאי העתקה, הורס ואופרטור השמה

אופרטור ההשמה יפעל באופן הסטנדרטי שראיתם בקורס – כלומר יחליף את איברי מטריצת היעד כך שיהיו זהים לאיברי מטריצת המקור (ללא תלות במימדי המטריצות). לדוגמא, בהינתן המטריצה

$$mat_3 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

תוצאת ההשמה $mat_1 = mat_3$ תהיה

$$mat_1 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

בנאי ההעתקה וההורס מוגדרים באופן הסטנדרטי שראיתם בקורס.

3.1.3. יצירת מטריצת יחידה

בנוסף לבנאים הרגילים, נרצה לאפשר ליצור מטריצת יחידה ממימד נתון, באופן הבא:

```
IntMatrix identity_2 = IntMatrix::Identity(2);
```

```
IntMatrix identity_3 = IntMatrix::Identity(3);
```

$$identity_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad identity_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

תזכורת: מטריצת יחידה היא מטריצה ריבועית שכל איבריה אפסים, פרט לאחדות באלכסון הראשי.

3.1.4. תשאול מימדים

ממשו את הפונקציות `width`, `height` ו-`size` המחזירות את גובה המטריצה (מספר השורות), רוחב המטריצה (מספר העמודות), ואת מספר האיברים במטריצה, בהתאמה. למשל עבור המטריצות שהוגדרו בסעיף הקודם, נקבל:

```
cout << mat_3.height() << "x"
      << mat_3.width() << ", total= "
      << mat_3.size(); // prints "2x3, total=6"
```

3.1.5. שחלוף (transpose)

הפעולה מחזירה את המטריצה המשוחלפת, מבלי לשנות את האובייקט שעליו הופעלה. השחלוף מוגדר באופן הרגיל: עבור מטריצה A , בביצוע שחלוף (מסומן כ- A^T) האיבר $A[i][j]$ מתחלף עם האיבר $A[j][i]$. שימו לב שעבור מטריצה לא ריבועית השחלוף משנה את מימדי המטריצה.

לדוגמא, לאחר הרצת הקוד הבא:

```
IntMatrix mat_4 = mat_3.transpose();
```

ערכי איברי המטריצות יהיו:

$$mat_4 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \quad mat_3 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

הערה: שימו לב שעבור מטריצה לא ריבועית השחלוף משנה את מימדי המטריצה.

3.1.6. אופרטור חיבור

עליכם לתמוך בפעולת `operator+` המחברת את המטריצות איבר-איבר. ניתן להניח בחלק זה שנשתמש בחיבור רק עבור מטריצות מאותו מימד.

לדוגמה, תוצאת החיבור

```
IntMatrix mat_5 = mat_3 + mat_3;
```

תהיה:

$$mat_5 = \begin{pmatrix} 2 & 6 & 10 \\ 4 & 8 & 12 \end{pmatrix}$$

3.1.7. אופרטור חיסור אונארי

עליכם לספק גרסה אונארית של `operator-` עבור מטריצות.

```
IntMatrix mat_6 = -mat_5; // note the "-"mat_1
```

$$mat_6 = \begin{pmatrix} -2 & -6 & -10 \\ -4 & -8 & -12 \end{pmatrix}$$

3.1.8. אופרטור חיסור

עליכם לתמוך בפעולת `operator-` המחסירה את המטריצות איבר-איבר. ניתן להניח בחלק זה שנשתמש בחיסור רק עבור מטריצות מאותו מימד, בדומה לחיבור.

3.1.9. חיבור עם סקלר

עבור סקלר מטיפוס `int`, מתבצעת הוספה של הסקלר לכל איברי המטריצה. את פעולה זו יש לממש בשתי צורות: `operator+ 1` ו-`operator+=`. לדוגמה, הקוד הבא:

```
mat_6 += 2;
```

```
IntMatrix mat_7 = mat_6 + 1;
```

```
mat_7 = 1 + mat_6;
```

$$mat_6 = \begin{pmatrix} 0 & -4 & -8 \\ -2 & -6 & -10 \end{pmatrix} \quad mat_7 = \begin{pmatrix} 1 & -3 & -7 \\ -1 & -5 & -9 \end{pmatrix}$$

הערה: שימו לב לסימטריות של פעולת החיבור

הערה 2: על אופרטור `+=` לתמוך בשרשור פעולות, כלומר תוצאת השורה הבאה זהה לתוצאת השורה הראשונה בדוגמה:

```
(mat_6 += 1) += 1;
```

3.1.10. אופרטור פלט

עליכם לממש אופרטור `operator<<` להפניית המטריצה לערוץ פלט. לצורך כך נתונה לכם בקבצי העזר של התרגיל פונקציית עזר עם החתימה:

```
std::string printMatrix(const T* matrix_values, const Dimensions& dim);
```

הפונקציה מקבלת מערך מטיפוס `T` המכיל את כל איברי המטריצה פרוסים שורה-שורה, וכן את ממדי המטריצה המקוריים, ומחזירה מחרוזת המתארת את המטריצה בפורמט אחד שניתן להדפסה.

סדר האיברים המצופה במערך `matrix_values` - כמו בסעיף 0.

לדוגמה:

```
cout << mat_6;
```

ידפיס למסך בהתאם לפורמט שמופק לכם בפונקציה `printMatrix`:

```
0  -4  -8
-2 -6 -10
```

סדר הדפסת האיברים המצופה הוא:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

3.1.11. גישה לאיברי המטריצה - אופרטור ()

נרצה לממש אופרטור `()` (אופרטור סוגריים) באופן שיאפשר גישה לאיברי המטריצה עם שני אינדקסים, לקריאה (כתיבה. לדוגמה, `mat(1,0)`) יתייחס לאיבר הראשון בשורה השנייה של המטריצה `mat_1`.

דהיינו, בהינתן:

$$mat_1 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

עליכם לממש את הממשקים הנדרשים למחלקה `IntMatrix` על מנת לתמוך בגישה לקריאה באופן

```
cout << mat_1(1,0) << endl; // prints 2
```

וכן בגישה לכתיבה באופן:

```
mat_1(1,0) = 18;
```

```
cout << mat_1(1,0) << endl; // prints 18
```

בנוסף, עליכם לתמוך גם בגישה (לקריאה בלבד) למטריצה קבועה:

```
const mat_8(mat_1);
```

```
cout << mat_8(1,0) << endl; // prints 18
```

```
mat_8(1,0) = 19; // Compilation error
```


3.1.12. פעולות השוואה לוגיות

ממשו את פעולות ההשוואה $<$, $>$, $<=$, $>=$, $==$, $!=$ בין מטריצות לבין מספרים שלמים (int). תוצאת השוואה בין מטריצה למספר תוגדר להיות מטריצה המכילה אפסים ואחדות בלבד, בעלת ממדים זהים למטריצה המקורית, ובה האיברים המקיימים את ההשוואה במטריצה המקורית יוחלפו באחדות, ואילו שאר האיברים יוחלפו באפסים. לדוגמה, בהינתן

$$mat_1 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

תוצאת הקוד הבא:

```
mat_9 = mat_1 < 4;  
mat_10 = mat_1 == 4;
```

תהיה:

$$mat_9 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad mat_10 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

בסעיף זה אין צורך לממש את האופרטורים בצורה סימטרית, כלומר ניתן להניח שבהפעלתם המספר השלם יהיה תמיד מצד ימין של האופרטור והמטריצה תמיד מצד שמאל.

3.1.13. פעולות קיבוץ בוליאני

יש לממש את הפונקציות הבוליאניות all, any כפונקציות חיצוניות למחלקה IntMatrix. הפונקציה all מחזירה true אם ורק אם כל איברי המטריצה שהיא מקבלת שונים מאפס. הפונקציה any מחזירה true אם ורק אם קיים לפחות איבר אחד במטריצה ששונה מאפס.

לדוגמה, עבור המטריצות מהסעיף הקודם נקבל שהקוד הבא:

```
cout << all(mat_1) << all(mat_9) << all(mat_10) << endl;  
cout << any(mat_1) << any(mat_9) << any(mat_10) << endl;  
cout << any(mat_1 > 7);
```

ידפיס:

```
true false false  
true true true  
false
```

3.1.14 מעבר סדרתי (איטרציה) על איברי המטריצה

(1) רקע

מבני נתונים רבים בC++ מספקים ממשק אחיד למעבר סדרתי על איבריהם, באופן שמסתיר מהמשתמש את פרטי המימוש של המבנה וכך מונע שינויים בלתי רצויים ופגיעה בשלמות המבנה.

כדי לתמוך בממשק האיטרציה האחיד, על מחלקת מבנה הנתונים (במקרה שלנו – מחלקת המטריצה) לספק מתודה בשם `begin()`, שמחזירה "איטרטור" לאיבר הראשון של המבנה, ומתודה בשם `end()`, שמחזירה "איטרטור" המציין שהמעבר על המבנה הסתיים, קרי – איטרטור המתקבל על ידי קידום איטרטור שמצביע לאיבר האחרון של המבנה (תחת סדר האיטרציה) במקום אחד. האיטרטור עצמו ממומש כמחלקה נפרדת בעלת גישה לשדות מבנה הנתונים, ועל אובייקטים ממחלקה זו ניתן לבצע פעולת קריאת/כתיבת ערך ופעולת קידום.

(2) מחלקת `IntMatrix::iterator`

עליכם לספק תמיכה בממשק האיטרציה האחיד עבור מחלקת `IntMatrix`. ערכי החזרה של המתודות `begin()` ו-`end()` יהיה עצם מטיפוס `IntMatrix::iterator`, שעליכם להגדיר כך שיתמוך באופן השימוש הבא:

$$mat_3 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

// 1. Begin and end:

```
IntMatrix::iterator it_begin = mat_3.begin(); // it_begin is a new iterator pointing to first element  
// (mat_3[0][0], which is equal to 1)
```

```
IntMatrix::iterator it_end = mat_3.end(); // it_end points to one past last element
```

// 2. dereference (operator*) returns the element pointed to by the iterator

```
cout << *it_begin << endl; // prints out 1
```

// 2a. dereference supports element assignment

```
*it_begin = 17; // mat_3[0][0] is now 17
```

```
cout << *it_begin << endl; // prints out 17.
```

// 3. operator++ advances iterator to next element. Advancing past the end() iterator

// is undefined

```
it_begin++; // iterator now points to the 2nd element (mat_3[0][1], which is equal to 3)
```

```
cout << *it_begin << endl; // prints out 3
```

// 4. Comparison operator==

```
cout << it_begin == it_end << endl; // prints false
```

דהיינו, המשימות שלכם בסעיף זה הן:

(a) להגדיר מחלקה `IntMatrix::iterator` (דהיינו, תת-מחלקה של `IntMatrix`), שתומכת בפעולות הבאות:

1.1. בנאי העתקה, **הורס ואופרטור השמה**

1.2. אופרטור `*`

1.3. אופרטור `++` שמאלי וגם אופרטור `++` ימני.

1.4. אופרטור `==` ואופרטור `!=`

(b) להוסיף מתודות `begin()` ו-`end()` מתאימות לממשק של מחלקת ה-`IntMatrix`. לאחר מימוש האיטרטור, ניתן יהיה לעבור בלולאה על איברי המטריצה באופן הבא:

```
for (IntMatrix::iterator it = mat_3.begin(); it != mat_3.end(); ++it)
{
    cout << *it << endl;
}
```

3.1.14.3 מחלקת `IntMatrix::const_iterator`

שימו לב שהאיטרטור שהגדרנו עד כה לא יעבוד עם מטריצה קבועה. לאור זאת, יש להגדיר בנוסף גם מחלקה בשם `IntMatrix::const_iterator`, שתאפשר איטרציה על מטריצה קבועה, יחד עם גרסאות קבועות לפעולות `begin` ו-`end`. כלומר – כאשר פעולות אלו נקראות על מטריצות `const`, האובייקט שיוחזר יהיה ממחלקת `const_iterator`. בניגוד ל-`iterator`, מחלקת `const_iterator` תאפשר גישה לקריאה בלבד לאיברי המבנה. השימוש באיטרטור זה זהה לשימוש באיטרטור הרגיל, אלא שעל `const_iterator` למנוע שינוי של הערך אליו הוא מצביע.

הקוד הבא יתקמפל ויעבוד באותו אופן:

```
for (IntMatrix::const_iterator it = mat_3.begin(); it != mat_3.end(); ++it)
{
    cout << *it << endl;
}
```

בעוד שהקוד הבא לא יתקמפל, עקב ניסיון שינוי הערך המוצבע דרך האיטרטור הקבוע.

```
for (IntMatrix::const_iterator it = mat_3.begin(); it != mat_3.end(); ++it)
{
    *it = 1;
}
```

הערות:

- בתקן C++ 11 שאנו עובדים איתו, ניתן להשתמש גם בכתיבה המקוצרת לאותה פעולה:

```
for (int element : mat_3)
{
    cout << element << endl;
}
```

- שימו לב להבדל ביחס לממשק האיטרציה מתרגיל בית 1: כל קריאה ל- `IntMatrix::begin()` מחזירה עצם חדש, דהיינו ייתכנו מספר איטרטורים המצביעים בו-זמנית ובאופן בלתי תלוי זה בזה לאיברי המטריצה.
- סדר האיברים במטריצה לצורך האיטרציה מוגדר כמקודם, לפי החוקיות בסעיף 0.
- אין לשנות את ממשקי האיטרטורים ואין להוסיף להם פונקציות או בנאים נוספים – יש לממש את הממשקים המוגדרים במסמך זה בלבד.

4 חלק ב'

4.1. מבוא

בחלק זה עליכם לממש מחלקה גנרית בשם `Matrix`, שתתמוך בממשק הדומה ל- `IntMatrix` מחלק א', עבור טיפוס איברים גנרי (שנסמנו ב-T). כמו בחלק הקודם, עליכם להסיק את חתימות הממשק על סמך דוגמאות הקוד, אך בחלק זה בשונה מחלק א' יש להשתמש בתבניות ולטפל בשגיאות בעזרת מנגנון החריגות של C++.

הערות:

- תעדו את כל הקוד שאתם כותבים ובמיוחד את פונקציות הממשק שאתם ממשים. **רשמו בתיעוד של כל פונק' ממשק מה ההנחות על טיפוס האיברים בפונק' זו.** שימו לב: לא כל פונק' הממשק תהיינה מוגדרות עבור כל טיפוס איברים, אלא רק פונק' שההנחות שלהן על טיפוס האיברים מתקיימות. מכיוון שרק פונק' תבנית שבשימוש עוברות הידור (conditional instantiation), זה לא יפגע ביכולת להגדיר מטריצות עם טיפוס איברים אלה.
- רוב הפעולות בממשק המחלקה הגנרית `Matrix` מוגדרות בדומה לחלק א', עם זאת – בחלק מהמקרים יש שינויים וכן תוספת של חריגות. קראו בעיון את ההוראות!
- עבור חלק זה מסופק לכם קובץ `Auxiliaries` מעודכן.

4.2. ממשק המחלקה Matrix

4.2.1. טיפוס חריגות

ממשו את 3 המחלקות המתוארות להלן שישמשו לחריגות. לכל מחלקה יש לממש פונקציה בשם `what()` אשר מחזירה מחרוזת המתארת את השגיאה, בהתאם לפירוט הבא (ראו דוגמאות שימוש מטה)

שם המחלקה	מחרוזת תיאור
Matrix::AccessIllegalElement	Mtm matrix error: An attempt to access an illegal element
Matrix::IllegalInitialization	Mtm matrix error: Illegal initialization values
Matrix::DimensionMismatch	Mtm matrix error: Dimensions mismatch: (<mat1_height>,<mat1_width>) (<mat2_height>,<mat2_width>)"

לדוגמה, עבור קטע הקוד הבא (שמנסה לאתחל מטריצה בגובה 0):

```
try {
    Dimensions dim(0,5);
    Matrix<char*> mat(dim);
} catch (const mtm::matrix<int>::IllegalInitialization & e){
    cout<< e.what() <<endl;
}
```

הפלט יהיה:

Mtm matrix error: Illegal initialization values

דוגמא נוספת (שמנסה לחבר מטריצות בגדלים שונים):

```
try {
    Dimensions dim(2,5);
    Matrix<int> mat_1(dim);
    Matrix<int> mat_2 = Matrix<int>::Diagonal(2,1);
    Matrix<int> mat_3 = mat_1+mat_2
} catch (const mtm::matrix<int>::DimensionMismatch & e) {
    cout<< e.what() <<endl;
}
```

יודפס:

Mtm matrix error: Dimensions mismatch: (2,5) (2,2)

שימו לב לסדר הדפסת מימדי המטריצות: תחילה יודפס מימד המשתנה השמאלי ולאחר מכן הימני.

4.2.2. בנאי

ממשק הבנאי דומה לסעיף 3.1.1 - הארגומנט הראשון מגדיר את מימדי המטריצה, והארגומנט השני הוא הערך אליו יאותחלו כל איברי המטריצה (עצם מטיפוס T). אם לא סופק ארגומנט שני כל הערכים של המטריצה יאותחלו לערך שמוחזר מבנאי ה-default (חסר הארגומנטים) של T. לדוגמה, הקוד הבא:

```
Dimensions dims(2,4);
Matrix<int> mat_1(dims, 5);
Matrix<int> mat_2(dims); //The elements value is equal to int()
```

ייצור את המטריצות:

$$\text{mat_1} = \begin{pmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{pmatrix} \quad \text{mat_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

הערה: עבור טיפוסים בסיסיים של שפת C, הבנאי חסר הפרמטרים שלהם מייצר משתנה שערכו אפס. למשל, הסינטקסט int() קורא ל"בנאי" של int ומחזיר int שערכו 0. בשל כך, בדוגמה למעלה המטריצה mat_2 מאותחלת לאפסים.

מקרי שגיאה והטיפול בהם:

- תיזרק חריגה מסוג `IllegalInitialization` אם המימדים הם לא מספר חיובי.
 - תיזרק חריגה מסוג `std::bad_alloc` במקרה של כישלון בהקצאת זיכרון.
- (שימו לב: חריגה זו כבר נזרקה ע"י אופרטור new. אין צורך לתפוס אותה ולזרוק מחדש).

4.2.3. בנאי העתקה, הורס ואופרטור השמה

יפעלו בדומה לסעיף 3.1.2. לדוגמה:

```
Matrix<int> mat_3(mat_2);
```

הערה: לאחר שימוש בהורס או אופרטור השמה כל האיטרטורים שהוקצו לאותה מטריצה הופכים להיות לא שמישים, וזה לא **באחריותכם** לטפל בזה.

4.2.4. יצירת מטריצה אלכסונית

נרצה לתמוך ביצירה של מטריצה אלכסונית, כאשר מימד המטריצה מועבר כארגומנט ראשון. איברי המטריצה שאינם באלכסון יאותחלו להיות הערך שמוחזר מהבנאי חסר הפרמטרים של T, ואיברי האלכסון יאותחלו להיות הערך שמועבר כארגומנט שני. לדוגמה:

```
Matrix<int> diagonal_2 = Matrix<int>::Diagonal(2,1);
```

```
Matrix<short> diagonal_3 = Matrix<short>::Diagonal (3,2);
```

$$\text{diagonal_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{diagonal_3} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

מקרי השגיאה והטיפול בהם:

- יש לזרוק חריגה מסוג `IllegalInitialization` אם המימד הוא לא מספר חיובי.
- יש לזרוק חריגה מסוג `std::bad_alloc` במקרה של כישלון בהקצאת זיכרון.

4.2.5. תשאול מימדים

הממשקים יוגדרו באופן זהה לסעיף 3.1.4:

```
cout << diagonal_2.height() << "x"
      << diagonal_2.width() << ", total= "
      << diagonal_2.size(); // prints "2x2, total=4"
```

4.2.6. שחלוף (transpose)

בדומה לסעיף 3.1.5. דוגמת שימוש עבור mat_3 מסוג Matrix<int>:

```
Matrix<int> mat_4 = mat_3.transpose();
```

4.2.7. אופרטור חיבור

דוגמת שימוש בצורה זהה לסעיף ל- 3.1.6:

```
Matrix<int> mat_5 = mat_3 + mat_3; 3.1.6
```

4.2.8. אופרטור חיסור וחיסור אונארי 3.1.6

בדומה לסעיפים 3.1.4, 3.1.8

באופרטור חיבור וחיסור יש לזרוק את החריגות הבאות :

- חריגה מסוג `DimensionMismatch` אם מימדי המטריצות אינם תואמים.

4.2.9. חיבור עם אובייקט מסוג T

עבור אובייקט מטיפוס T, חיבור של האובייקט עם מטריצה יחבר את אותו אובייקט לכל איברי המטריצה. את הפעולה הזו יש לממש בשתי צורות: `+operator` ו- `+operator`.

לדוגמה, נניח ש- mat_5 מטיפוס `matrix<std::string>` מוגדרת

```
mat_5 = ("Hello world" "Bye bye world")
```

אזי לאחר קטע הקוד הבא

```
mat_5 += "!";
```

```
Matrix<std::string> mat_6 = mat_5 + " * ";
```

```
mat_7 = " * " + mat_6;
```

ערכי המטריצות יהיו:

```
mat_5 = ("Hello world!" "Bye bye world!")
mat_6 = ("Hello world! * " "Bye bye world! * ")
mat_7 = (" * Hello world! * " " * Bye bye world! * ")
```

4.2.10. אופרטור פלט

עליכם לממש אופרטור `operator<<` שמדפיס את המטריצה לערוץ פלט. לצורך כך נתונה לכם בקבצי העזר של התרגיל פונקציית עזר עם החתימה:

```
template <class ITERATOR_T>
std::ostream& printMatrix(std::ostream& os, ITERATOR_T begin,
                        ITERATOR_T end, unsigned int width);
```

הפונקציה מקבלת ערוץ פלט, איטרטורים לתחילת וסוף המטריצה (כפי שיוגדרו בסעיף 0) ומספר העמודות במטריצה, ורושמת את המטריצה לערוץ הפלט בפורמט אחיד.

4.2.11. גישה לאיברי המטריצה - אופרטור ()

נרצה לממש אופרטור `()`, בדומה לסעיף 3.1.11. לדוגמה, עבור `mat_1` מטריצה מסוג `Matrix<std::string>` שמוגדרת ע"י

$$mat_1 = \begin{pmatrix} "1" & "3" & "5" \\ "2" & "4" & "6" \end{pmatrix}$$

עליכם לממש את הממשקים הנדרשים למחלקה הגנרית `Matrix` על מנת לתמוך בגישה לקריאה כך:

```
cout << mat_1(1,0) << endl; // prints 2
```

וכן בגישה לכתיבה כך:

```
mat_1(1,0) = "18";
```

```
cout << mat_1(1,0) << endl; // prints 18
```

בנוסף, עליכם לתמוך גם בגישה (לקריאה בלבד) למטריצה קבועה:

```
const matrix<std::string> mat_8 = mat_1;
```

```
cout << mat_8(1,0) << endl; // prints 18
```

```
mat_8(1,0) = 19; // Compilation error
```

מקרי השגיאה והטיפול בהם:

- יש לזרוק חריגה מסוג `AccessIllegalElement` אם ערכי האינדקסים **שליילים**, או חורגים מגבולות המטריצה.

4.2.12. פעולות השוואה לוגיות

ממשו את פעולות ההשוואה `<`, `>`, `<=`, `>=`, `==`, `!=` בין אובייקט מטיפוס `Matrix<T>` לבין אובייקט מסוג `T`. תוצאת השוואה בין מטריצה לאובייקט תוגדר להיות מטריצה שאיבריה בוליאניים (דהיינו הטיפוס `Matrix<bool>`), בעלת ממדים זהים למטריצה המקורית, ובה האיברים המקיימים את ההשוואה במטריצה המקורית יוחלפו בערך `true`, ואילו שאר האיברים יוחלפו בערך `false`.

לדוגמה:

$$mat_1 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

תוצאת הקוד הבא:

```
mat_9 = mat_1 < 4;
```



```
mat_10 = mat_1 == 4;
```

תהיה:

$$mat_9 = \begin{pmatrix} true & true & false \\ true & false & false \end{pmatrix}, \quad mat_{10} = \begin{pmatrix} false & false & false \\ false & true & false \end{pmatrix}$$

גם בסעיף זה כמו ב-3.1.12 בסעיף זה אין צורך לממש את האופרטורים בצורה סימטרית, כלומר ניתן להניח שבהפעלתם הערך הסקלרי יהיה תמיד מצד ימין של האופרטור והמטריצה תמיד מצד שמאל.

4.2.13. פעולות קיבוץ בוליאני

ממשו את הפונקציות הבוליאניות all, any בדומה לסעיף 3.1.13. הפונקציה all מחזירה true אם ורק אם כל איברי המטריצה שהיא מקבלת הם true כערכי אמת (כלומר, כשהם מומרים ל-bool). הפונקציה any מחזירה true אם ורק אם קיים לפחות איבר אחד במטריצה שערכו true כערך אמת.

דוגמת שימוש:

```
cout << all(mat_1) <<endl ;  
cout << any(mat_1 > 7) <<endl;  
cout << all(mat_10) <<endl;
```

יודפס:

```
true  
false  
false
```

4.2.14. מעבר סדרתי (איטרציה) על איברי המטריצה

(1) מחלקת `Matrix<T>::iterator`

עליכם לספק תמיכה בממשק האיטרציה האחד עבור המחלקה הגנרית `Matrix`.

לדוגמה, עבור `mat_3` מטריצה מסוג `Matrix<int>` כלהלן

$$mat_3 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

עליכם לתמוך בשימוש הבא:

```
// 1. Begin and end:
IntMatrix<int>::iterator it_begin = mat_3.begin(); // it_begin is a new iterator pointing to first
element
// (mat_3[0][0], which is equal to 1)
IntMatrix<int>::iterator it_end = mat_3.end(); // it_end points to one past last element

// 2. dereference (operator*) returns the element pointed to by the iterator
cout << *it_begin << endl; // prints out 1
// 2a. dereference supports element assignment
*it_begin = 17; // mat_3[0][0] is now 17
cout << *it_begin << endl; // prints out 17.
```

דהיינו, המשימות שלכם בסעיף זה הן:

(a) להגדיר מחלקת `Matrix<T>::iterator` שתומכת באותן פעולות כמו בסעיף 3.1.14.2.

(b) להוסיף מתודות `begin` ו-`end` מתאימות לממשק של מחלקת ה-`Matrix`. לאחר מימוש האיטרטור, ניתן יהיה לעבור בלולאה על איברי המטריצה באופן הבא:

```
for (Matrix<int>::iterator it = mat_3.begin(); it != mat_3.end(); ++it)
{
    cout << *it << endl;
}
```

(2) מחלקת `Matrix<T>::const_iterator`

באותו האופן עליכם לספק תמיכה בממשק האיטרציה האחד עבור מחלקת מסוג `const Matrix`. כמו שהוגדר עבור `IntMatrix`.

מקרי השגיאה והטיפול בהם במחלקות `Matrix<T>::iterator` ו-`Matrix<T>::const_iterator`:

- יש לזרוק חריגה מסוג `AccessIllegalElement` אם מנסים לבצע `*operator` על `iterator` שלא מצביע לאיבר חוקי. `Iterator` לא יצביע לאיבר חוקי במקרה ש:
 - `iterator` הוחזר מפונקציה `end()`.
 - איטרטור קודם מעבר לגבולות המטריצה

לדוגמה:

```
try {
    cout << *mat_3.end();
} catch (mtm::matrix<int>::AccessIllegalElement const& e) {
    cout<< e.what() <<endl;
}
```

יודפס:

Mtm matrix error: an attempt to access an illegal element

שימו לב שהפעלת הורס או אופרטור השמה על המטריצה תגרום לאי נכונות של האיטרטור ואין זה באחריות המתכנת לדאוג למקרים אלה.

4.2.15. הפונק' apply

נרצה לממש למחלקת Matrix פונקציית ממשק בשם apply. הפונקציה apply תקבל כפרמטר אובייקט עם אופרטור סוגריים (פונקציה של C או function object), ותפעיל אותו איבר-איבר על כל אברי המטריצה, תוך החזרת מטריצה חדשה.

לדוגמה, עבור ה-functor הבא:

```
class SignedSquareRoot {
public:
    int operator()(int val){
        return val >= 0 ? sqrt(val) : -sqrt(-val);
    }
};
```

נוכל לקרוא ל-apply עבור mat_3 כך:

```
Matrix<int> mat_11 = mat_3.apply(SignedSquareRoot());
```

5 איתור דליפות זיכרון באמצעות valgrind

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. כדי לוודא זאת, תוכלו להשתמש בכלי שאתם מכירים מתרגיל בית 1 בשם `valgrind` שמתחקה אחר ריצת התכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך להשתמש בכלי על מנת לבדוק האם יש לכם דליפות בתכנית היא באמצעות שתיהפועולות הבאות:

1. קימפול של השורה בחלק הבא עם הדגל `-g`.
2. הרצת השורה הבאה:

```
➤ Valgrind --leak-check=full ./[program name]
```

כאשר `[program name]` זה שם קובץ ההרצה (לא מגישים את קובץ ההרצה לכן תוכלו לתת לו שם כרצונכם). הפלט ש `valgrind`-מפיק אמור לתת לכם, במידה ויש לכם דליפות (והידרתם את התוכנית עם `-g`), את שרשרת הקריאות שהתבצעו שגרמו לדליפה. אתם אמורים באמצעות ניפוי שגיאות להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התכנית. בנוסף, `valgrind` מראה דברים נוספים כמו קריאה לא חוקית (שלא גררה `segmentation fault` – גם שגיאות אלו עליכם להבין מהיכן מגיעות ולתקן). תוכלו למצוא תיעוד של דגלים נוספים שימושיים של הכלי ע"י `man valgrind`, או לחפש באינטרנט.

6 הידור, קישור ובדיקה

כל חלק של התרגיל יעבור הידור בנפרד. לדוגמא חלק א' יעבור הידור על ידי הפקודה הבאה

```
➤ g++ -std=c++11 -Wall -Werror -pedantic-errors -DDEBUG partA/*.cpp  
-o [program name]
```

בצורה דומה גם חלק ב'.

פירוט תפקיד כל דגל בפקודת הקמפול:

- **-std=c++11** שימוש בתקן השפה `c++11`.
- **[program name]** הגדרת שם הקובץ המהודר.
- **-Wall** דווח על כל האזהרות.
- **-pedantic-errors** דווח על סגנון קוד שאינו עומד בתקן הנבחן כשגיאות.
- **-Werror** התייחס לאזהרות כאל שגיאות – משמעות דגל זה שהקוד חייב לעבור הידור ללא אזהרות ושגיאות.
- **-DDEBUG** מוסיף את השורה `#define NDEBUG` בתחילת כל יחידת קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו `assert` לא יופעל בריצת התוכנית.

7 הגשה

יש לבצע הגשה אלקטרונית בלבד דרך אתר הקורס, בזוגות בלבד.

ההגשה הינה בתיקיית zip, כאשר הקוד לכל חלק של התרגיל נמצא בתת-תיקייה נפרדת (עבור 2 החלקים הראשונים והחלק האחרון שיתפרסם בהמשך). מבנה תיקיית הקוד שיש להגיש מוצג בתמונה ליד. הקבצים שמגישים בחלקים הרטובים צריכים להיקרא כמו שמות הקבצים בתמונה, והחלק היבש צריך להיות מוגש בפורמט pdf ולהיקרא dry.pdf. אסור להוסיף צילומי מסך, או צילומים בכללי לקובץ pdf שמוגש בחלק היבש, יש להקליד את התשובות.

על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית:

- שימרו את תמונת המסך עם קוד האישור שמתקבלת בהגשה.
- שימרו עותק של התרגיל על השרת csl3 לפני ההגשה האלקטרונית ואל תשנו אותה, כך שחתימת העדכון האחרונה תהיה לפני מועד ההגשה.

ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.

