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 בתיקייה ~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 חבריכם.
- 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 להתשמש 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 }
```

```
int main() {
    B b1(1), b2(2), b3(3);
    const B b4 = b1 + (b2 + b3);
    cout << smaller(b1,b2) << endl;
    return 0;
}
```

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.

<u>טיפול בשגיאות</u>: 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);
```

$$\text{identity\_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \text{identity\_3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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:

$$mat\_4 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad mat\_3 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

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+` ו `operator+=`. לדוגמא, הקוד הבא:

```
mat_6 += 2;
IntMatrix mat_7 = mat_6 + 1;
mat_7 = 1 + mat_6;
```

$$mat\_6 = \begin{pmatrix} 0 & -4 & -8 \\ -2 & -6 & -10 \end{pmatrix} \qquad \mathrm{mat\_7} = \begin{pmatrix} 1 & -3 & -7 \\ -1 & -5 & -9 \end{pmatrix}$$

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 אחיד שניתן להדפסה.

סדר האיברים המצופה במערך matrix_values - כמו בסעיף0 .

לדוגמה:

```
cout << mat_6;
```

ידפיס למסך בהתאם לפורמט שמסופק לכם בפונקציה printMatrix:

$$\begin{matrix} 0 & -4 & -8 \\ -2 & -6 & -10 \end{matrix}$$

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 וכתיבה. 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:

$$mat\_9 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad mat\_10 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

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

3.1.13. פעולות קיבוץ בוליאני

יש לממש את הפונקציות הבוליאניות all, any כפונקציות חיצוניות למחלקה IntMatrix. הפונקציה all מחזירה true אם ורק אם <u>כל</u> איברי המטריצה שהיא מקבלת שונים מאפס. הפונקציה any מחזירה true אם ורק אם קיים <u>לפחות איבר אחד</u> 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 assigment
*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 טיפוסי 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. טיפוסי 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
1	`Matrix::AccessIllegalElement`	Mtm matrix error: An attempt to access an illegal element
2	`Matrix::IllegalInitialization`	Mtm matrix error: Illegal initialization values
3	`Matrix::DimensionMismatch`	Mtm matrix error: Dimension 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<int> 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: Dimension 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:

$$\text{mat\_1} = \begin{pmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{pmatrix} \quad \text{mat\_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

<u>הערה</u>: 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 נזרקת 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);
```

$$\text{diagonal\_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \text{diagonal\_3} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

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 אונאר3.1.6

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

$$mat\_5 = (\text{"}Hello\ world\text{"} \quad \text{"}Bye\ bye\ world\text{"})$$

אזי לאחר קטע הקוד הבא

```
mat_5 += std::string( "!");
Matrix<std::string> mat_6 = mat_5 + std::string( "*");
mat_7 = std::string( "*") + mat_6;
```

ערכי המטריצות יהיו:

$$mat\_5 = (\text{"}Hello\ world!\text{"} \quad \text{"}Bye\ bye\ world!\text{"})$$
$$mat\_6 = (\text{"}Hello\ world!*\text{"} \quad \text{"}Bye\ bye\ world!*\text{"})$$
$$mat\_7 = (\text{"}*Hello\ world!*\text{"} \quad \text{"}*Bye\ bye\ world!*\text{"})$$

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

$$mat\_1 = \begin{pmatrix} \text{"1"} & \text{"3"} & \text{"5"} \\ \text{"2"} & \text{"4"} & \text{"6"} \end{pmatrix}$$

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 שליליים, 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:

$$mat\_9 = \begin{pmatrix} true & true & false \\ true & false & false \end{pmatrix}, \qquad mat\_10 = \begin{pmatrix} false & false & false \\ false & true & false \end{pmatrix}$$

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 האחיד 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 assigment
*it_begin = 17; // mat_3[0][0] is now 17
cout << *it_begin << endl; // prints out 17.
```

דהיינו, <u>המשימות שלכם בסעיף זה הן</u>:

(a) להגדיר מחלקה 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 האחיד עבור מחלקת מסוג const Matrix. כמו שהוגדר עבור IntMatrix.

מקרי השגיאה והטיפול בהם במחלקות Matrix<T>::const_iterator וMatrix<T>::iterator:

- יש לזרוק חריגה מסוג `AccessIllegalElement` אם מנסים לבצע operator* על iterator שלא מצביע לאיבר חוקי. Iterator לא יצביע לאיבר חוקי במקרה ש:
  - הiterator 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 (פונקצייה 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 חלק ג'

5.1. מבוא

על מנת ליישב את הוויכוח הנצחי על מהי השפה הטובה ביותר בקורס: C++ או Python, חברי סגל הקורס החליטו להתפצל לשתי קבוצות ולהילחם עד המוות. למרבה המזל, בגלל דרישות הריחוק החברתי של הקורונה – לא ניתן היה לעשות זאת במציאות, ולכן הוחלט לקיים את המלחמה באופן ממוחשב, בתור משחק לוח בסיסי שאותו תממשו אתם בחלק זה של התרגיל.

לצורך המשחק, יוגדרו בהמשך מספר סוגים של דמויות עם הבדלים בהתנהגויותיהן. המשחק מתקיים על לוח משבצות מלבני (אילו רק מישהו היה מממש עבורנו מבנה נתונים גנרי ומלבני), כאשר בכל משבצת נמצאת בכל רגע נתון לכל היותר דמות אחת.

מטרת המשחק פשוטה: על הדמויות משתי הקבוצות להילחם אחת נגד השנייה באמצעות פעולות תקיפה. כאשר נקודות החיים של דמות מגיעות ל־0 – הדמות יוצאת מהמשחק, וכאשר נותרות על הלוח דמויות מקבוצה אחת בלבד – נכריז שהמשחק הסתיים.

בתרגיל זה לא נממש את מהלך המשחק השלם, אלא נתמקד במימוש הפעולות בלוח המשחק – נרצה לאפשר לשחקנים להוסיף דמויות ללוח, להזיז דמויות ולתקוף דמויות באמצעות דמויות אחרות.

הערות:

- מחלקות החריגות שמוזכרות בתיאור הממשקים מפורטות בפרק 5.5.
- **בחלק זה ניתן להשתמש בכל מבני/רכיבי STL <u>שנלמדים בקורס</u>**
- בכל פעם שמוזכר מושג ה"מרחק" בין 2 משבצות, הכוונה היא למספר הצעדים ימינה/שמאלה/למעלה/למטה בין המשבצות. פורמלית, מרחק זה שווה לסכום ההפרשים בערך מוחלט של כל קואורדינטה בנפרד, כלומר, המרחק בין הנקודות $(x_1, y_1), (x_2, y_2)$ הוא:

$$|\boldsymbol{x_1} - \boldsymbol{x_2}| + |\boldsymbol{y_1} - \boldsymbol{y_2}|$$

5.2. המחלקה Character

בתרגיל זה עליכם להשתמש בירושה ופולימורפיזם על מנת לייצג את הדמויות השונות ולאחסן אותן בלוח המשחק. ממשקי המחלקות שבהן תשתמשו אינם מוגדרים במלואם בתרגיל, ועליכם לתכנן אותם בצורה נכונה, פשוטה ויעילה (ללא שכפולי קוד מיותרים) על מנת לקיים את דרישות המשחק שיפורטו בהמשך.

דרישה אחת שמוגדרת במפורש מהמחלקות בתרגיל היא שמותיהן והיררכיית הירושה ביניהן.

לצורך תרגיל זה עליכם לכתוב את המחלקה Character המתארת דמות כללית. עליכם להבטיח שלא ניתן יהיה ליצור עצמים ממחלקה זו (אלא רק ממחלקות יורשות ממנה, שיוגדרו בהמשך). על המחלקה להיות מוגדרת בקובץ משלה, Character.h.

5.3. דמויות

במשחק זה מוגדרים כמה סוגים שונים של דמויות, אשר נבדלות אחת מהשנייה בהתנהגות שלהן (בפעולות התקיפה שניתן לבצע איתן בלוח המשחק). בתרגיל זה לא ניתנת דרישה מפורשת על ממשק המחלקות הללו, ועליכם לתכנן אותן בצורה נבונה, תוך שמירה על עקרונות הירושה והפולימורפיזם, על מנת שהמימוש שלכם יהיה נכון ופשוט.

קיימים מספר פרמטרים המוגדרים עבור כל הדמויות:

- נקודות חיים (health) – כאשר ערך זה מגיע ל־0 הדמות נחשבת כ"מתה" ויש להוציאה מהמשחק.
- תחמושת (ammo) – הדמויות השונות זקוקות לתחמושת על מנת לבצע פעולות תקיפה (ייתכן שדמויות שונות יצרכו תחמושת לפי כללים שונים).
- טווח תקיפה (range) – ערך זה קובע אילו מיקומים דמות מסוימת מסוגלת לתקוף מהמקום שבו היא נמצאת באותו הרגע.
- עוצמה (power) – ערך זה קובע את הנזק (בנקודות חיים) שנגרם לדמות יריבה כאשר תוקפים אותה.

הדרישות היחידות על ממשק הדמויות הן:

- עבור כל דמות תוגדר מחלקה עם שם הדמות, ובקובץ בעל אותו השם. למשל, עבור הדמות Medic עליכם להגדיר מחלקה בשם Medic בקבצים בשם Medic.h ו־Medic.cpp.
- על כל מחלקות הדמויות לרשת (public) מהמחלקה Character.

5.3.1. מחלקת Soldier

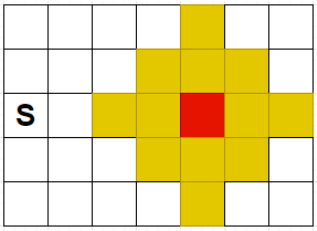
- דמות זו מסוגלת לזוז למרחק של עד 3 משבצות בכל פעולת תזוזה.
- בפעולת טעינה, לדמות זו נוספות 3 נקודות תחמושת.
- מתקפה של דמות זו עולה יחידת תחמושת (ammo) אחת.
- הדמות יכולה לתקוף בקו ישר בלבד (משבצות באותה שורה או באותה עמודה של הלוח), משבצות הנמצאות לכל היותר במרחק השווה לטווח (range) של הדמות.
- הדמות יכולה לתקוף משבצות ללא תלות בתוכן שלהן (גם אם המשבצת ריקה או מכילה דמות מאותה קבוצה).
- כאשר הדמות תוקפת משבצת מסוימת, ליריבים הנמצאים באותה משבצת נגרם נזק השווה לערך ה־power של התוקף, וליריבים שאינם במשבצת היעד אך נמצאים במרחק לכל היותר $\lceil range/3 \rceil$ (שליש מ-range מעוגל לשלם הקרוב ביותר מלמעלה) ממנה נגרם נזק השווה לערך $\lceil power\ /\ 2 \rceil$ של התוקף. לתוקף עצמו, וכן לדמויות מקבוצת התוקף – לא נגרם נזק בתקיפה.

לדוגמה:

דמות Soldier הנמצאת בשורה 2 (מלמעלה) ועמודה 0 של הלוח הבא, והיא בעלת range=4 ו־power=10. הדמות מסוגלת לתקוף את המשבצת הנמצאת בשורה 2 ועמודה 4 (מסומנת באדום בשרטוט) מכיוון שהמרחק בינה לבין הדמות אינו עולה על range.

ליריבים במשבצת היעד נגרם נזק של 10 נקודות. ליריבים שאינם נמצאים במשבצת היעד, אך נמצאים במרחק לכל היותר $[4/3] = 2$ מהיעד, נגרם נזק של 5 נקודות (המשבצות הרלוונטיות מסומנות בצהוב). שימו לב שהמרחק נמדד כאן במספר הצעדים ימינה/שמאלה/למעלה/למטה (ללא אלכסונים) שצריך לבצע כדי לזוז בין המשבצות.

לדמויות מהקבוצה של ה־Soldier, כמו גם ל־Soldier עצמו, לא נגרם נזק, אפילו אילו הן היו בטווח הפגיעה הצהוב או האדום.

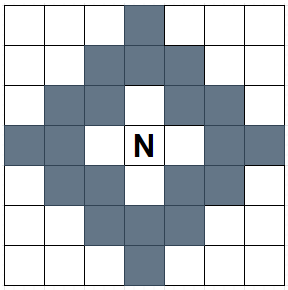


5.3.2. מחלקת Medic

- דמות זו מסוגלת לזוז למרחק של עד 5 משבצות בכל פעולת תזוזה.
- בפעולת טעינה, לדמות זו נוספות 5 נקודות תחמושת.
- דמות זו יכולה לתקוף משבצת המכילה יריב, או משבצת המכילה דמות מאותה קבוצה של התוקף, אך לא משבצת ריקה. כמו כן על המשבצת להיות במרחק לכל היותר range מהתוקף (משבצת היעד אינה חייבת להיות באותה שורה או באותה עמודה).
- במקרה שבמשבצת היעד יש יריב – המתקפה עולה יחידת תחמושת אחת, וגורמת ליריב נזק השווה לערך ה־power של התוקף.
- במקרה שבמשבצת היעד יש דמות מאותה קבוצה של התוקף – המתקפה אינה עולה יחידות תחמושת, והיא מוסיפה לדמות שבמשבצת היעד נקודות חיים בערך השווה ל־power של התוקף.
- דמות זו אינה יכולה לתקוף את עצמה.

5.3.3. מחלקת Sniper

- דמות זו מסוגלת לזוז למרחק של עד 4 משבצות בכל פעולת תזוזה.
- בפעולת טעינה, לדמות זו נוספות 2 נקודות תחמושת.
- מתקפה של דמות זו עולה יחידת תחמושת אחת.
- דמות זו יכולה לתקוף משבצות המכילות יריב בלבד, ונמצאות לכל היותר במרחק range, ולכל הפחות במרחק $\lceil range / 2 \rceil$ מהצלף (משבצת היעד אינה חייבת להיות באותה שורה או באותה עמודה). לדוגמה, בשרטוט שלהלן, דמות Sniper הנמצאת בשורה 3 ובעמודה 3 בעלת range=3 יכולה לתקוף יריבים הנמצאים במשבצות האפורות, שכן משבצות אלה נמצאות לכל היותר במרחק 3 ולכל הפחות במרחק $\lceil 3/2 \rceil = 2$.



- מתקפה של דמות זו גורמת ליריב לנזק השווה בערכו לערך ה־power של התוקף, אך כל מתקפה שלישית של דמות זו גורמת לנזק כפול (השווה בערכו ל־2 * power של התוקף). רק התקפות מוצלחות (כאלה שגרמו נזק ליריב) נספרות.

לדוגמה, עבור Sniper עם power=3, ערכי הנזק של רצף תקיפות ייראו כך:

3, 3, **6**, 3, 3, **6**, 3, 3, **6**, ...

5.4. המחלקה Game

מחלקה זו מתארת את המשחק כולו, ולה ממשק שיוגדר כעת – אין לשנות את הממשק הנתון או להוסיף אליו פעולות. (אך מותר להוסיף למחלקה זו ולכל המחלקות פונקציות עזר כל עוד הן private).

5.4.1. enums

עבור מחלקה זו מוגדרים טיפוסי ה־enum הבאים, שישמשו אתכם להמשך:

```
enum Team { CPP, PYTHON };
enum CharacterType {SOLDIER, MEDIC, SNIPER};
typedef int units_t;
```

5.4.2. Auxiliaries

מסופק לכם בנוסף לתרגיל זה קובץ עם הגדרות עזר, כולל ההגדרות הנ"ל ומחלקה בשם GridPoint המתארת משבצת בלוח המשחק באמצעות קואורדינטות דו־מימדיות (כאשר המספור מתחיל מ־0, כמו בגישה לאינדקסים של מטריצה).

5.4.3. בנאי

```
Game(int height, int width);
```

הפעולה יוצרת לוח משחק חדש וריק במימדים נתונים.

- במקרה שבו אחד מהפרמטרים הוא 0 או מספר שלילי – יש לזרוק חריגת IllegalArgument.

5.4.4. הורס

```
~Game();
```

על ההורס לשחרר את כל הזיכרון שהיה בשימוש לצורך המשחק.

5.4.5. בנאי העתקה ואופרטור השמה

```
Game(const Game& other);
Game& operator=(const Game& other);
```

על בנאי ההעתקה ועל אופרטור ההשמה להעתיק את תוכן המשחק כולו: את הדמויות, התכונות של כל אחת (אותה כמות נקודות חיים, אותה כמות תחמושת וכו') ואת מיקומיהן על לוח המשחק.

בעת שימוש בבנאי העתקה או באופרטור ההשמה בין שני משחקים – עליכם לדאוג שהמשחקים לאחר הפעולה יהיו בלתי תלויים לחלוטין, כלומר מאותה נקודה והלאה פעולות שיבוצעו על משחק אחד לא ישפיעו בשום אופן על המשחק האחר.

הנחיה: כללו במחלקת Character מתודת clone() אבסטרקטית כפי שראיתם בהרצאות, והשתמשו בה על מנת לממש בצורה פשוטה את הפונקציות בסעיף זה (הרצאה 10, שקפים 34-35).

5.4.6. הוספת דמות

5.4.6.1.הוספת דמות קיימת

```
void addCharacter(const GridPoint& coordinates, std::shared_ptr<Character> character);
```

הפעולה מקבלת מצביע (חכם) לדמות, ומוסיפה דמות זו ללוח המשחק במשבצת שהקואורדינטות שלה בלוח המשחק נתונות על ידי coordinates. אין צורך לשכפל את הדמות, אלא ניתן להשתמש באובייקט עצמו שהועבר לפונקצייה ולשמור אותו בתוך מחלקת המשחק.

הערות:

- הוסיפו את השורה `#include <memory>` על מנת להשתמש ב-std::shared_ptr
- ככלל, אין להשתמש בפקודות using בקבצי ממשק!

מקרי השגיאה והטיפול בהם:

- אם המשבצת הנתונה אינה בתחומי לוח המשחק – יש לזרוג את החריגה IllegalCell.
- אם במשבצת הנתונה כבר קיימת דמות – יש לזרוק את החריגה CellOccupied.

5.4.6.2.יצירת דמות חדשה

הגדירו במחלקה Game מתודה סטטית בעלת החתימה הבאה:

```
static std::shared_ptr<Character> makeCharacter(CharacterType type, Team team,
                units_t health, units_t ammo, units_t range, units_t power);
```

מתודה זו מקבלת פרמטרים ליצירת דמות, מקצה את הדמות הדרושה, ומחזירה מצביע (חכם) לאובייקט המתאר את דמות שיוצרה, מהמחלקה המתאימה ובעלת הנתונים המתאימים.

הדמות שנוצרת היא בעלת health נקודות חיים, ammo נקודות תחמושת, טווח תקיפה של range יחידות ועוצמת תקיפה של power יחידות.

סוג/מחלקת הדמות שיש ליצור הוא בהתאם ל־type, והקבוצה שאליה הדמות שייכת היא בהתאם ל־team.

מקרי השגיאה והטיפול בהם:

- אם ערך הפרמטר health הוא 0, או אם ערך כלשהו הוא שלילי – יש לזרוק חריגת IllegalArgument (שימו לב שעבור ammo ו-power ערך 0 הוא חוקי).

5.4.7. תנועת דמות

```
void move(const GridPoint & src_coordinates,  const GridPoint & dst_coordinates);
```

הפעולה משנה את מיקום הדמות שנמצאת במשבצת src_coordinates אל המשבצת dst_coordinates.

מקרי השגיאה והטיפול בהם:

- אם אחת מהמשבצות אינה בתחומי לוח המשחק – יש לזרוק את החריגה IllegalCell.
- אם במשבצת המקור לא קיימת דמות – יש לזרוק את החריגה .CellEmpty
- אם משבצת היעד מחוץ לטווח התזוזה של הדמות (בהתאם לחוקי הדמויות) – יש לזרוק את החריגה MoveTooFar.
- אם במשבצת היעד כבר קיימת דמות – יש לזרוק את החריגה CellOccupied.

5.4.8. תקיפה

```
void attack(const GridPoint & src_coordinates, const GridPoint & dst_coordinates);
```

הפעולה גורמת לדמות שבמשבצת הנתונה על ידי src_coordinates לתקוף את הדמות שבמשבצת הנתונה על ידי dst_coordinates.

אם משבצת היעד מתאימה לתקיפה – מתבצעת תקיפה. הדמות במשבצת היעד תושפע מההתקפה בהתאם לסוג הדמות התוקפת. כמו כן בהתאם לסוג ההתקפה, ייתכן שדמויות נוספות יושפעו ממנה. (הכוונה ב"מתאימה לתקיפה" היא לפי חוקי הדמות – למשל, sniper מסוגל לתקוף רק משבצות המכילות דמות יריב, ו־medic מסוגל לתקוף דמויות מקבוצתו, אך לא את עצמו, וכו').

אם כתוצאה מהתקיפה נקודות החיים של אחת הדמויות מגיעה ל־0, יש להוציא דמות זו מלוח המשחק.

מקרי השגיאה והטיפול בהם:

- אם אחת מהמשבצות אינה בתחומי לוח המשחק – יש לזרוק את החריגה IllegalCell.
- אם במשבצת המקור לא קיימת דמות – יש לזרוק את החריגה CellEmpty.
- אם משבצת היעד של התקיפה נמצאת מחוץ לטווח הדמות התוקפת (בהתאם לחוקי הדמות) – יש לזרוק את החריגה OutOfRange.
- אם לדמות במשבצת המקור אין מספיק תחמושת לבצע את התקיפה – יש לזרוק את החריגה OutOfAmmo.
- אם לא ניתן לבצע את התקיפה מסיבות שאינן נכללות בחריגות האחרות (בהתאם לחוקי הדמות התוקפת, למשל ניסיון תקיפה של משבצת ריקה על ידי Sniper או Medic) – יש לזרוק את החריגה IllegalTarget.

5.4.9. טעינת נשק

```
void reload(const GridPoint & coordinates);
```

הפעולה מוסיפה לדמות הנמצאת במשבצת coordinates תחמושת, כמוגדר לפי סוג הדמות.
מקרי השגיאה והטיפול בהם:

- אם המשבצת הנתונה אינה בתחומי לוח המשחק – יש לזרוג את החריגה IllegalCell.
- אם במשבצת הנתונה אין דמות – יש לזרוק חריגת CellEmpty.

5.4.10. אופרטור פלט

מסופקת לכם בקבצי העזר הפונקציה:

```
std::ostream& printGameBoard(std::ostream& os,  const char* begin,
                             const char* end, unsigned int width) const;
```

המשמשת להדפסת לוח המשחק בפורמט אחיד. עליכם לספק לפונקציה הזו ערוץ פלט, את רוחב לוח המשחק, וכן מצביעים להתחלה ו(אחד אחרי ה)סוף של **מערך תווים** המייצג את לוח המשחק. על מערך התווים להכיל את לוח המשחק לפי שורות, כאשר תוכן כל תא במערך מוגדר באופן הבא:

1. עבור משבצת ריקה בלוח המשחק – התא המתאים במערך יכיל את התו רווח (' ').
2. עבור משבצת המכילה דמות Soldier, התא יכיל את התו 'S' אם קבוצת הדמות היא cpp ואת התו 's' אם קבוצת הדמות היא python.
3. עבור משבצת המכילה דמות Medic, התא יכיל את התו 'M' אם קבוצת הדמות היא cpp ואת התו 'm' אם קבוצת הדמות היא python.
4. עבור משבצת המכילה דמות Sniper, התא יכיל את התו 'N' אם קבוצת הדמות היא cpp ואת התו 'n' אם קבוצת הדמות היא python.

עליכם לממש עבור המחלקה Game את האופרטור operator<< שמדפיס את לוח המשחק בהתאם לפורמט המוגדר על ידי printGameBoard.

5.4.11. בדיקת ניצחון

```
bool isOver(Team* winningTeam=NULL) const;
```

הפעולה בודקת האם במצב הלוח הנוכחי קיימת קבוצה מנצחת.

- במקרה שעל לוח המשחק נותרו דמויות מקבוצה אחת בלבד (קבוצת CPP או קבוצת PYTHON), הפעולה מחזירה true . כמו כן, אם winningTeam אינו NULL, הפונקצייה כותבת את זהות הקבוצה המנצחת למצביע זה.
- אם על הלוח לא קיימות דמויות מאף קבוצה, או אם קיימות על הלוח דמויות משתי הקבוצות – הפעולה מחזירה false ולא משנה את ערכו של winningTeam.
- ניתן תמיד להעביר לפונקצייה nullptr בתור הפרמטר winningTeam, במקרה זה היא מחזירה אם המשחק הסתיים או לא, אך לא כותבת דבר למצביע.

5.5. חריגות

עליכם להגדיר מחלקות עבור כל החריגות המוגדרות במסמך זה. על כל מחלקות החריגות להתאים בשמותיהן למצוין במסמך, ועליהן לרשת ממחלקה בשם mtm::Game::Exception, שבתורה יורשת מהמחלקה mtm::Exception, שבתורה יורשת מהמחלקה הסטנדרטית std::exception.

בנוסף – את החריגות שלכם מחלק ב' של התרגיל הגדירו כעת (אם לא עשיתם זאת בחלק הקודם) כמחלקות היורשות מ־mtm::Exception.

סדר החריגות המוגדר במסמך הוא:

1. IllegalArgument – למקרים שבהם המשתמש מנסה לבצע פעולה עם פרמטר שאינו חוקי.
2. IllegalCell – למקרים שבהם משבצת יעד של פעולה כלשהי אינה נמצאת בתחומי לוח המשחק.
3. CellEmpty – למקרים שבהם מתבצע ניסיון לפעולה ממשבצת מקור ריקה (למשל תזוזה ממשבצת מקור ריקה). שימו לב שתקיפה של משבצת יעד ריקה אפשרית בחלק מהמקרים (למשל הדמות Soldier מסוגלת לעשות זאת).
4. MoveTooFar – למקרים שבהם מתבצע ניסיון להזיז דמות למשבצת שנמצאת מחוץ לטווח התזוזה שלה.
5. CellOccupied – למקרים שבהם מתבצע ניסיון להוסיף דמות למשבצת שכבר מכילה דמות.
6. OutOfRange – למקרים שבהם דמות מנסה לתקוף משבצת שנמצאת מחוץ לטווח התקיפה שלה (טווח תקיפה של דמויות עלול להיות מוגדר באופן שונה בהתאם לסוג הדמות.
7. OutOfAmmo – למקרים שבהם לדמות אין מספיק תחמושת לביצוע תקיפה (לא בכל תקיפה נדרשת תחמושת, והדבר תלוי בסוג הדמות).
8. IllegalTarget – למקרים שבהם דמות אינה מסוגלת לתקוף משבצת יעד כלשהי, מסיבות שאינן טווח או תחמושת (בהתאם לחוקי הדמות).

במקרים שבהם אתם נדרשים לזרוק חריגה אך קיימות מספר חריגות שמתאימות למצב – יש לזרוק את החריגה שמופיעה ראשונה בסדר הנ"ל.

5.5.1. הודעות שגיאה

עליכם לדרוס את המתודה ()what של std::exception כך שבקריאה למתודה זו עבור אחת המחלקות שהוגדרו בסעיף זה תוחזר המחרוזת:

"A game related error has occurred: <Exception class>"

כאשר עבור כל חריגה "<Exception class>" מוחלפת בשם מחלקת החריגה. למשל, עבור החריגה IllegalArgument על המחרוזת המלאה להיות:

"A game related error has occurred: IllegalArgument"

עבור החריגות שהוגדרו בחלק ב', עליכם לדאוג שבקריאה ל־what() יוחזרו המחרוזות שהוגדרו עבור אותן חריגות בחלק ב', אך על חריגות אלה לרשת מ־mtm::Exception כאמור.

הערה: כאשר אתם דורסים את המתודה ()what, יש להוסיף בסוף החתימה שלה את המילה noexcept. הדבר נוגע לנושא שאינו מכוסה בקורס, אך הקומפיילר דורש זאת על מנת לקמפל את הקוד. מילה זו מצהירה שהמתודה לא זורקת חריגות, והדבר נדרש מכיוון שהמתודה ()what במחלקת האב מוגדרת גם היא כך.

6 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 שתיהפעולות 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.

7 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 -DNDEBUG partA/*.cpp
  ~mtm/public/1920b/ex3/Auxiliaries.cpp -I ~mtm/public/1920b/ex3/
  <test_file>.cpp -o [program name]
```

בצורה דומה גם חלק ב' ו-ג' (עם התיקיות partB ו- partC בהתאמה).

פירוט תפקיד כל דגל בפקודת ההידור:

- **-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.
- **-DNDEBUG** מוסיף את השורה #define NDEBUG בתחילת כל יחידת קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו assert לא יופעל בריצת התוכנית.
- **partA/*.cpp** קבצי הפתרון שלכם. **בחלק ב' יש להשמיט אם לא הוספתם קבצי cpp תחת התיקייה partB.**
- **`<test_file>.cpp`** הוא קובץ שמכיל פונק' main.
- **-I ~mtm/public/1920b/ex3** היא הכוונה לכלול את התיקייה במסלול החיפוש של קבצי ההידור (זוהי התיקייה שמכילה את הקובץ Auxiliaries.h). שימו לב – במחשב שלכם מיקום הקבצים יהיה אחר.

8 הגשה

יש להגיש אלקטרונית בלבד דרך אתר הקורס, בזוגות בלבד.

```
submission.zip/
├── dry.pdf
├── partA
│   ├── IntMatrix.cpp
│   ├── IntMatrix.h
│   └── ... possibly other files
├── partB
│   ├── Matrix.h
│   └── ... possibly other files
└── partC
    ├── Character.{cpp,h}
    ├── Game.{cpp,h}
    ├── Medic.{cpp,h}
    ├── ... possibly other files
    ├── Sniper.{cpp,h}
    └── Soldier.{cpp,h}
```

ההגשה הינה בתיקיית zip, כאשר הקוד לכל חלק של התרגיל נמצא בתת־תיקייה נפרדת. מבנה תיקיית הzip שיש להגיש מוצג בתמונה משמאל.

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.