מבוא לתכנות מערכות תרגיל בית 2 – אביב 2021

תאריך פרסום: 26.5.2021

תאריך הגשה: 20.6.2021

משקל התרגיל: 12% מהציון הסופי (תקף).

מתרגלים אחראיים: מוחמד מסארוה ,יניב גופט ואיתי אלפסי.

הערות כלליות

- ההגשה היא בזוגות באתר webcourse.
- **שימו לב:** לא ינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות התרגיל, או לשאול בפורום הקורס בפיאצה. לפני שליחת שאלה – נא וודאו שהשאלה לא מופיעה כבר בפיאצה, ושהתשובה אינה ברורה בFAQ.
- **קראו מסמך זה עד סופו לפני שאתם מתחילים לממש!** יתכן שתצטרכו להתאים את המימוש שלכם לחלק עתידי בתרגיל. תכננו את המימוש שלכם לפני שאתם ניגשים לעבוד.
- חובה להתעדכן בעמוד הFAQ של התרגיל! התשובות המועברות שם הן מחייבות.
- העתקות מכל סוג תטופלנה בחומרה.
- התרגיל מורכב מחלק רטוב וחלק יבש.
- **על כל הקוד להיות תחת namespace mtm.**
- כדי לשאול שאלות לגבי התרגיל, השתמשו בפורום הקורס בpiazza.
- במהלך הdebugging, אנו ממליצים להשתמש בהוראות בdebugging cheatsheet שמופיע תחת תיקיית Debugging Workshop.

חלק א

בחלק זה של התרגיל תממשו חלק מתוך מערכת ניהול המבחנים באתר students.

1. מחלקת ExamDetails

מחלקה זו מתארת בחינה במערכת. עליכם לממש מחלקה בשם ExamDetails, אשר תספק את הממשק הבא:

1. **בנאי:** הבנאי מקבל שישה פרמטרים:
   - מס' קורס – מס' שלם. אין צורך לבצע בדיקת תקינות קלט עבור מס' קורס.
   - חודש הבחינה – נע בין 1 ל12.
   - יום הבחינה – ניתן להניח שכל חודש מכיל 30 ימים.
   - שעת הבחינה – ניתן להניח שמבחן מתחיל רק בשעות עגולות או בחצאי שעות (יכול להתחיל ב06:00 או 13:30, אבל לא ב14:15).
   - אורך הבחינה– שעות עגולות בלבד. אין הגבלה על אורך בחינה.

בהצלחה! ☺

- לינק לחדר הזום של הבחינה. **מוגדר ערך ברירת מחדל** – אם לא ניתן לינק, הוא מוגדר כמחרוזת ריקה. ניתן לייצג את הלינק באמצעות std::string.
- עליכם להגדיר שלוש חריגות: חריגה מסוג ExamDetails::InvalidDateException שתיזרק במקרה שנתוני התאריך אינם תקינים, חריגה מסוג ExamDetails::InvalidTimeException שתיזרק במקרה שנתוני השעה אינם תקינים, וחריגה מסוג ExamDetails::InvalidArgsException שתיזרק בכל סוג אחר של נתונים לא תקינים לבנאי (למשל, מס' קורס שאינו חוקי).
  אין צורך לבצע בדיקת תקינות ללינק.
  יש לבדוק את תקינות הקלט לפי סדר הארגומנטים לבנאי, ולזרוק חריגה המתאימה לארגומנט הראשון שאינו תקין במקרה של שגיאות מרובות.

2. **הורס.**
3. **בנאי העתקה ואופרטור השמה**.
4. **מתודת getLink:** מתודה זו מחזירה את הלינק לבחינה (לא מקבלת פרמטרים).
5. **מתודת setLink:** מתודה זו מקבלת לינק חדש לבחינה ומעדכנת את האובייקט בהתאם.
6. **operator-:** אופרטור שמקבל בחינה נוספת ומחזיר את הפרש הימים בין הבחינה השמאלית לבין הבחינה הימנית. האופרטור יכול להחזיר ערך שלילי.
7. **operator<:** אופרטור שמקבל בחינה נוספת ומחזיר true אם הבחינה השמאלית מתרחשת **לפני** הבחינה הימנית. עבור בחינות שמתקיימות במקביל האופרטור מחזיר false.
8. **הדפסה:** הדפסת התאריך באמצעות אופרטור הפלט. פורמט הפלט יפורט בדוגמת הקוד.

9. **בנאי סטטי:** יש להגדיר במחלקת ExamDetails מתודה סטטית בשם makeMatamExam שיוצרת ומחזירה עצם ExamDetails שמתארת את הבחינה בקורס מת"ם.
   הבחינה המוגדרת בעצם תהיה בתאריך 28.7, תתחיל ב13:00 ותימשך 3 שעות.
   הלינק המוגדר עבור הבחינה הוא "https://tinyurl.com/59hzps6m".

דוגמת קוד להתנהגות המחלקה הנדרשת:

```
ExamDetails exam1 = ExamDetails::makeMatamExam();
ExamDetails exam2(104032, 7, 11, 9.5, 3);
ExamDetails exam3 = exam1;


cout << "the difference between MATAM and infi 2m is " << (exam1-exam2) << " days";


ExamDetails& closest = (exam1 < exam2) ? exam1 : exam2;
cout << "your closest exam is:" << endl << closest << endl;
```

הפלט עבור דוגמה זו הוא:

```
the difference between MATAM and infi 2m is 17 days

your closest exam is:
Course Number: 104032
Time: 11.7 at 9:30
Duration: 3:00
Zoom Link:
```

בהצלחה! ☺

דגשים למימוש:

- המחלקה Exam תמומש בקבצים examDetails.cpp ו examDetails.h.
- אתם רשאים לממש כל מתודה **פרטית** הנדרשת למימוש המחלקה אך אין לשנות או להוסיף לממשק המחלקה.
- אם עוד לא למדתם על חריגות בשלב זה, מומלץ לכתוב את הקוד כאילו אין שגיאות, ולהוסיף טיפול בשגיאות בעזרת חריגות יותר מאוחר.
- במקרה של טיפוסי נתונים פרימיטיביים ומחרוזות, ניתן להעביר ולהחזיר נתונים ממתודות by value.
- הקוד חייב להיכתב בהתאם לכללי הפיתוח הנכון שלמדנו בקורס.
- הפתרון שתגישו חייב לטפל בשגיאות.
- בהשוואה בין ערכי double, הפרש בין ערכים שקטן מ10^{-6} נחשב כמספרים שווים.
- ניתן להשתמש בכל פונקצית ספריה מתמטית הנדרשת לבדיקת תקינות הקלט, ובפרט בstd::modf.

2. מבנה נתונים גנרי – SortedList

בחלק זה אסור להשתמש בSTL.

2.1. תיאור כללי

בחלק זה של התרגיל נממש מבנה נתונים גנרי ב++C תוך שימוש בתבניות, חריגות ואיטרטורים.

הערה: כיוון שבמועד פרסום התרגיל טרם נלמדו חריגות, מומלץ בתור התחלה לממש את המבנה ללא התחשבות בשגיאות, ולהוסיף שימוש בחריגות לאחר לימוד הנושא.

איטרטורים נלמדו בהרצאה 8, ומופיעים בשקפים 43-48. חזרה נוספת על איטרטורים תועבר בתרגול 10.

מבנה הנתונים אותו נממש הוא רשימה מקושרת ממוינת. והוא צריך לקיים את הדרישות הבאות:

1. כמות האיברים בSortedList אינה חסומה.
2. הוספת איברים מתבצעת ע"י פעולת insert
3. הסרת איברים מבוצעת ע"י פעולת remove
4. **איברי הרשימה ממוינים בכל רגע נתון –** פעולות insert וremove שומרות על הסדר בין איברי הרשימה. הסדר בין האיברים מוגדר ע"י אופרטור > של הטיפוס T.

לתבנית המחלקה SortedList יהיה פרמטר יחיד: class T. פרמטר זה יהיה טיפוס האלמנטים של הSortedList.

שימו לב: עליכם להניח כמות הנחות מינימלית על הטיפוס T. לא ניתן להניח שקיים בנאי חסר ארגומנטים לT.

2.2. ממשק הSortedList

המחלקה תספק את הממשק הבא:

1. בנאי חסר פרמטרים ליצירת SortedList ריק..
   הערה: זכרו להשתמש בעיקרון RAII (Resource Acquisition is Initialization) לתכנות נכון.
2. הורס לSortedList.
3. בנאי העתקה.
4. אופרטור השמה.
5. insert – מתודה שמקבלת אלמנט חדש כפרמטר ומכניסה אותו לרשימה.
6. remove – מתודה שמקבלת **איטרטור** (יפורט בהמשך) על המבנה כפרמטר ומסירה את האלמנט אליו הוא מפנה מהרשימה.

בהצלחה! ☺

7. length – מתודה שמחזירה את מס' האלמנטים ברשימה.
8. filter – מתודה שמקבלת **פרדיקט** (פונקציה בוליאנית) על טיפוס איברי הרשימה, ומחזירה רשימה חדשה שמכילה **רק את האיברים מהרשימה המקורית שמקיימים את התנאי**.
   הדרכה: יש להשתמש עבור מתודה זו בtemplate נוסף.
9. apply – מתודה שמקבלת **פונקציה** מטיפוס איברי הרשימה לטיפוס איברי הרשימה, ומחזירה SortedList שמכילה את תוצאות הפעלת הפונקציה על כל איברי הרשימה.
   הדרכה: יש להשתמש בטיפוס template נוסף.
10. begin – מתודה שמחזירה **איטרטור** לתחילת הרשימה.
11. end – מתודה שמחזירה **איטרטור** לסוף הרשימה.

2.3. איטרטור קבוע עבור SortedList

בנוסף לממשק הקיים, נממש מחלקת איטרטור קבוע עבור הרשימה, אשר תוגדר בתור SortedList::const_iterator.

על האיטרטור לספק את הממשק הבא:

1. בנאי העתקה ואופרטור השמה – מקבלים כפרמטר איטרטור אחר ומבצעים פעולה סטנדרטית.
   אסור לאפשר למשתמש גישה לבנאי הסטנדרטי של המחלקה.
2. הורס לאיטרטור.
3. ++operator – מימוש לאופרטור ++ שמקדם את האיטרטור לאיבר הבא ברשימה.
   אם האיטרטור מצביע לסוף הרשימה, **יש לזרוק חריגה** מטיפוס std::out_of_range.
4. ==operator – מימוש לאופרטור == שמקבל איטרטור נוסף ובודק האם שניהם שווים.
5. *operator - מימוש לאופרטור * שיחזיר **const reference** לאיבר אליו האיטרטור מפנה.

דגשים למימוש:

- אנחנו ממליצים להתחיל ממימוש של SortedList כרשימה **לא** גנרית ללא filter, apply.
  לאחר מכן לממש גנריות, ולבסוף לממש את filter וapply.
- את המימוש עליכם לכתוב בקובץ SortedList.h.
- יש לממש את המבנה כך שיהיה **לא תלוי** במשתמש מבחינת זיכרון. כלומר, עליו ליצור עותקים משלו לכל האובייקטים שהוא מקבל מהמשתמש.
- אתם רשאים להוסיף מתודות פרטיות ומחלקות עזר כרצונכם, תוך שמירה על עקרונות התכנות הנכון. אין לשנות את ממשק הרשימה או האיטרטור.
- הרשימה יכולה להכיל ערכים שווים מבחינת operator<. הדרישה היחידה היא סידור לפי הסדר שמגדיר האופרטור.

בהצלחה! ☺

דוגמת קוד להמחשת הממשק:

```
using std::string;
string getLen(string str)
{
    return std::to_string(str.length());
}

template<class T>
void printList(SortedList<T> list) {
    for (auto it = list.begin(); !(it == list.end()); ++it) {
        cout << *it << endl;
    }
    cout << endl;
}

int main()
{
    SortedList<string> lst1 = SortedList<string>();
    lst1.insert("Charlie");
    lst1.insert("Bob");
    lst1.insert("Alice");
    lst1.insert("Donald");
    printList(lst1);

    SortedList<string> lst2 = lst1;
    lst2 = lst2.apply(getLen);
    printList(lst2);

    SortedList<string>::const_iterator it = lst2.begin();
    cout << *it << endl << endl;
    ++it;
    lst2.remove(it);
    printList(lst2);
    return 0;
```

הפלט המתקבל עבור דוגמה זו הוא:

```
Alice
Bob
Charlie
Donald

3
5
6
7

3

3
6
7
```

בהצלחה! ☺

3. שאלות יבשות על SortedList

עליכם לענות על השאלות הבאות ולהגישן כקובץ PDF בשם dry.pdf שיצורף להגשה הסופית.

1. במבנה הSortedList הגנרי, מה הדרישות **ההכרחיות** שעל הטיפוס T לקיים? הסבירו.

2. נניח כי היינו רוצים לממש איטרטור non-const עבור SortedList. כלומר, עבור איטרטור זה האופרטור * היה מחזיר T&. איזו בעיה עלולה להיווצר במימוש?

3. במתודה filter, המימוש כולל שימוש בtemplate נוסף כדי להעביר את הפרדיקט. אילו שתי דרכים שונות קיימות בשפת C++ למימוש והעברת הפרדיקט? מה ההבדל ביניהן? מדוע אנחנו לא צריכים לספק שני מימושים שונים כדי לתמוך בשתי הדרכים הללו?

בהצלחה! ☺

4. חלק ב'

4.1 מבוא

על מנת ליישב את הוויכוח הנצחי על מהו ענף הספורט של האנשים החזקים ביותר: Crossfitters או Powerlifters, 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, 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 4.5.
- אין קשר בין חלק א' לחלק ב' ולא מצופה מכם להשתמש במימוש של sortedList בחלק זה.
- **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:

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

4.2 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.

בהצלחה! ☺

4.3 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.

4.3.1 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.

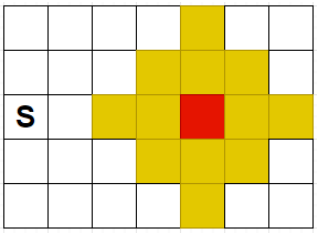
בהצלחה! ☺

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 $\lceil 4/3 \rceil = 2$ 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.



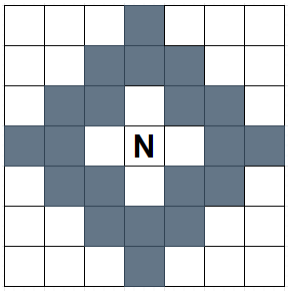
4.3.2 מחלקת Medic

- 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.

בהצלחה! ☺

4.3.3 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**, ...

בהצלחה! ☺

4.4 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).

4.4.1 enums

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

```
enum Team { POWERLIFTERS, CROSSFITTERS };
enum CharacterType {SOLDIER, MEDIC, SNIPER};
typedef int units_t;
```

4.4.2 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 המספר מתחיל מ־0, כמו בגישה לאינדקסים של מטריצה).

4.4.3 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.

4.4.4 הורס

```
~Game();
```

על ההורס לשחרר את כל הזיכרון שהיה בשימוש לצורך המשחק.

4.4.5 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).

בהצלחה! ☺

4.4.6 הוספת דמות

4.4.6.1 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.

4.4.6.2 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).

בהצלחה! ☺

4.4.7 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.

4.4.8 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.

בהצלחה! ☺

4.4.9 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.

4.4.10 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 powerlifters ואת התו 's' אם קבוצת הדמות היא Crossfitters.
3. עבור משבצת המכילה דמות Medic, התא יכיל את התו 'M' אם קבוצת הדמות היא powerlifters ואת התו 'm' אם קבוצת הדמות היא Crossfitters.
4. עבור משבצת המכילה דמות Sniper, התא יכיל את התו 'N' אם קבוצת הדמות היא powerlifters ואת התו 'n' אם קבוצת הדמות היא Crossfitters.

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

4.4.11 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 powerlifters או קבוצת Crossfitters), 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.

בהצלחה! ☺

4.5 חריגות

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

הנחיה:

בחלק זה יש להגדיר את כל החריגות בקובץ בשם Exceptions.h (ואם צריך – לממש מתודות שלהן בחלק Exceptions.cpp).

4.5.1 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"

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.

בהצלחה! ☺

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 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.

6. הידור, קישור ובדיקה

התיקייה ~mtm/public/2021b/ex2/provided מכילה קבצי עזר להידור ובדיקת התרגיל, ומומלץ להעתיק אותה לתיקיית התרגיל הפרטית שלכם. התרגיל ייבדק על השרת csl3, ועליו יבוצע ההידור בעזרת הפקודות הבאות.

הקובץ part_a_test.cpp מכיל קובץ בדיקה בסיסי.

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

```
➢ g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG partB/*.cpp
  ~mtm/public/2021b/ex2/provided/part_b/Auxiliaries.cpp -IpartB -I
  ~mtm/public/2021b/ex2/provided/part_b
  ~mtm/public/2021b/ex2/provided/part_b/<test_file>.cpp -o
  <exec_file>
```

בצורה דומה גם חלק א' (עם התיקיה partA).

שימו לב כי הפקודה מורצת מתוך התיקיה הראשית אשר מכילה את התיקיות partA, partB בהם נמצאים הקבצים הרלוונטיים לכל חלק.

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.

בהצלחה! ☺

- **-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.
- **partB/*.cpp** קבצי הפתרון שלכם. **בחלק ב' יש להשמיט אם לא הוספתם קבצי cpp תחת התיקייה partB.**
- **-lpartB** - המטרה של חלק זה היא לכלול את התקייה partB (או partA) במסלול החיפוש של קבצי ההידור. תיקייה זו תכיל את קבצי ההידור שתגדירו בתרגיל.
- **<test_file>.cpp** הוא קובץ שמכיל פונק' main.
- **-I ~mtm/public/2021b/ex2** 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.

7. הגשה

יש להגיש אלקטרונית בלבד דרך אתר הקורס, בזוגות בלבד.

ההגשה הינה בתיקיית zip, כאשר הקוד לכל חלק של התרגיל נמצא בתת־תיקייה נפרדת:

```
subbmission.zip/
├─ dry.pdf
├─ partA/
│  ├─ examDetails.{cpp,h}
│  ├─ sortedList.h
│  ├─ ... possibly other files
├─ partB/
   ├─ Character.{cpp,h}
   ├─ Game.{cpp,h}
   ├─ Medic.{cpp,h}
   ├─ ... possibly other files
   ├─ Sniper.{cpp,h}
   ├─ Soldier.{cpp,h}
```

הקבצים שמגישים בחלקים הרטובים צריכים להיקרא כמו שתואר במסמך, 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.

בהצלחה! ☺