234124 - מבוא לתכנות מערכות
תרגיל בית מספר 3
סמסטר אביב 2022

תאריך פרסום: 17.5.2022
תאריך הגשה: 1.6.2022 עד 23:59
מתרגל אחראי לתרגיל: עדי חריף

1 הערות כלליות

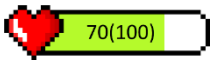
- **תרגיל זה מהווה 6% מהציון הסופי**
- התרגיל להגשה **בזוגות** בלבד.
- מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה או בסדנות. לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.
- שימו לב: לא תינתנה דחיות במועד הגשת התרגיל פרט למקרים חריגים. **תכננו את הזמן בהתאם**.
- ראו את התרגיל עד סופו לפני תחילת המימוש. חובה להתעדכן בעמוד ה-F.A.Q של התרגיל, הכתוב שם מחייב.
- העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם זאת – מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.
- קבצי התרגיל נמצאים בGitHub repository הבא: https://github.com/CS234124/ex3
- המסמך נכתב בלשון זכר מטעמי נוחות בלבד ומיועד לשני המינים.

```
 _______________________________
< Good luck and have fun coding! >
 -------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

2 תרגילי בית 2+3+4

בתרגילי בית 2+3+4 נממש משחק קלפים/תפקידים רב-משתתפים המבוסס (אך שונה) על משחק הקלפים הפופולרי Munchkin. במשחק זה תלחמו במפלצות, תדבגו באגים ותתקנו דליפות זיכרון על מנת לעלות בדרגות ולהגיש את תרגילי הבית בזמן.

3 HealthPoints



בחלק זה נממש מחלקה שתייצג לנו נקודות חיים (HP) של שחקן במשחק. נקודות החיים של שחקן הן מספר שלם בין 0 לכמות נקודות מקסימלית האפשריות לאותו שחקן.

3.1 ממשק HealthPoints

בחלק זה נגדיר את ממשק המחלקה HealthPoints.

3.1.1 בנאי

מייצר אובייקט המייצג נקודות חיים עם מספר מקסימלי של נקודות המתקבל כפרמטר. במידה ולא מסופק מספר מקסימלי, ערך ברירת המחדל הינו 100.
כמות הנקודות ההתחלתית של האובייקט שווה למספר הנקודות המקסימלי של אותו מופע.
אם מועבר לבנאי מספר שלילי או 0, יש לזרוק חריגה מסוג HealthPoints::InvalidArgument.

```
HealthPoints healthPoints1; /* has 100 points out of 100 */
HealthPoints healthPoints(150); /* has 150 points out of 150 */

try {
        HealthPoints healthPoints3(-100);
}
catch (HealthPoints::InvalidArgument& e) {
        /* some code... */
}
```

3.1.2 אופרטורים אריתמטיים

נגדיר אופרטור חיבור ואופרטור חיסור. ניתן לחבר ולחסר מופעים של HealthPoints עם מספרים שלמים מסוג int. המספר אותו נחבר או נחסר מהאובייקט יתווסף או יוחסר מכמות הנקודות הנוכחית של האובייקט. בכל מקרה כמות הנקודות הנוכחית לא תעלה על המקסימלית ולא תרד מתחת ל-0. לא ניתן לחבר או לחסר שני אובייקטים אחד עם השני.
בנוסף לחיבור וחיסור (+, -) נגדיר גם אופרטורים += ו- -= הפועלים באותו אופן ומבצעים השמה לאובייקט עליו פועלים.

```
healthPoints1 -= 20; /* now has 80 points out of 100 */
healthPoints1 +=100; /* now has 100 points out of 100 */
healthPoints1 -= 150; /* now has 0 points out of 100 */
healthPoints2 = healthPoints2 - 160; /* now has 0 points out of 150 */
healthPoints2 = 160 + healthPoints1; /* now has 100 out of 100 */
```

3.1.3 אופרטורים בוליאניים

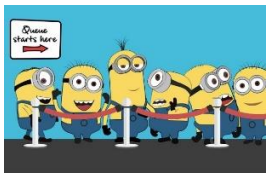
לשם השוואה בין מופעים של המחלקה נגדיר עבורה את האופרטורים == ו- =!. שני מופעים נחשבים שווים זה לזה אם ורק אם כמות נקודות החיים הנוכחית שלהם שווה. בנוסף נגדיר אופרטורי יחס עבור המחלקה: =< ,> ,=> ,<. כמו אופרטורי ההשוואה, אופרטורים אלו משווים בין הכמות הנקודות הנוכחית של שני אובייקטים.

```
bool comparisonResult;
HealthPoints healthPoints1(100);
HealthPoints healthPoints2 = 100; /* 100 points out of 100 */
comparisonResult = (healthPoints1 == healthPoints2); /* returns true */
healthPoints2(150); /* has 150 points out of 150 */
comparisonResult = (healthPoints1 == healthPoints2); /* returns false */
healthPoints2 -= 50; /* now has 100 points out of 150 */
comparisonResult = (100 == healthPoints2); /* returns true */
comparisonResult = (healthPoints1 < healthPoints2); /* returns false */
healthPoints1 -= 50; /* now has 50 points out of 100 */
comparisonResult = (healthPoints1 < healthPoints2); /* returns true */
```

3.1.4 אופרטור הדפסה

על מנת להדפיס מופעים של המחלקה בצורה נוחה, נגדיר עבורה אופרטור הדפסה ל-ostream, בדומה למה שראינו בהרצאות ובתרגולים. פורמט ההדפסה הינו (<currentValue>(<maxValue>.

```
std::cout << healthPoints1 << ", " << healthPoints2;
/* output is "50(100), 100(150)" */
```



4 Queue

בחלק זה תדרשו לממש מבנה נתונים שנקרא תור (Queue) אשר יוכל לשמש אותנו בהמשך כדי לנהל חפיסת קלפים, ואת תור שחקני המשחק.

4.1 הסבר על המבנה נתונים

תור הוא מבנה נתונים בו הנתונים שאנו מכניסים מוצאים עפ"י סדרם לפי עקרון FIFO – First In First Out. האיבר הראשון שנכנס לתור הוא האיבר הראשון שיוצא מהתור, האיבר השני שנכנס הוא השני שיוצא, וכך הלאה.
לשם מימוש המשחק נרצה להשתמש בתור שלנו למספר שימושים שונים, בהם נחזיק איברים מטיפוסים שונים (Player, Card) ולכן נממש תור גנרי.

4.2 ממשק של QUEUE

בחלק זה נגדיר את הפעולות בהם הממשק של התור יתמוך. פעולות אלו צריכות לתמוך בכל דוגמאות הקוד המופיעות במסמך זה.

4.2.1 pushBack

הפעולה pushBack מכניסה איבר חדש לסוף התור. שימו לב שנישמר עותק של האיבר החדש.

```
Queue<int> queue1;
queue1.pushBack(1);
queue1.pushBack(2);
```

4.2.2 front

הפעולה front מחזירה לנו את האיבר הראשון בתור.

```
int front1 = queue1.front(); /* front1 equals 1 as its the first value */
queue1.front() = 3; /* queue1 content is: 3, 2 (left to right) */
front1 = queue1.front(); /* front1 now equals 3 */
```

4.2.3 popFront

הפעולה popFront מוציאה את האיבר הראשון מהתור.

```
queue1.popFront();
```

שימו לב – המתודה אינה מחזירה את ערכו של האיבר שהוצא.

4.2.4 size

הפעולה size מחזירה את כמות האיברים שנמצאים כרגע בתור.

```
int size1 = queue1.size();
```

אם התור ריק המתודה תחזיר 0.

4.2.5 filter

הפעולה filter מקבלת תנאי מסויים ומסננת את איברי התור על פיו. התנאי (פרדיקט) מובא ע"י אובייקט עם אופרטור סוגריים (מצביע לפונקציה או Function Object) המקבל איבר ומחזיר ערך בוליאני עבורו. הפונקציה filter תחזיר תור חדש בו נמצאים כל איברי התור המקורי עבורם מתקבל true בהפעלת הפרדיקט.

```
static bool isEven(int n)
{
        return (n % 2) == 0;
}

int main()
{
        /* some code... */
        Queue<int> queue3;
        for (int i = 1; i <= 10; i++) {
                queue3.pushBack(i);
        }
        Queue<int> queue4 = filter(queue3, isEven);
        /* now queue4 contains: 2, 4, 6, 8, 10 (left to right) */
}
```

שימו לב – בשונה ממה שראיתם בהרצאה עבור המחלקה Set בהרצאה, הפונקציה filter אינה מתודה של המחלקה Queue אלא פונקציה חיצונית.

4.2.6 Transform

הפעולה transform עוברת על כל איברי התור ומשנה אותם (In-place) ע"י אופרציה נתונה. האופרציה המבוצעת על כל אחד מהאיברים נתונה על ידי אובייקט עם אופרטור סוגריים (מצביע לפונקציה או Function Object) המקבל איבר מהתור ומשנה אותו. הפעולה transform משנה את התור שקיבלה בכך שתפעיל על כל אחד מהאיברים את האופרציה.

```
static void setFortyTwo(int& n)
{
        n = 42;
}

int main()
{
        /* some code... */
        Queue<int> queue5;
        for (int i = 1; i <= 5; i++) {
                queue5.pushBack(i);
        }
        transform(queue5, setFortyTwo);
        /* now queue5 contains: 42, 42, 42, 42, 42 */
        /* some more code... */
}
```

שימו לב – האופרציות ש-transform מקבלת משנות את האיבר שמקבלות כארגומנט ואינן מחזירות ערך.

4.2.7 איטרטורים

נגדיר איטרטור בכדי לבצע איטרציה על איברי התור. האיטרטור יתמוך בממשק מינימלי כפי שראיתם בהרצאה בעזרת האופרטורים: ++, =! ו-*. כמו כן נגדיר לתור פעולות begin ו end המחזירות איטרטורים לתחילת התור ולסופו בהתאמה.

שימו לב - האיטרטור המוחזר מ-end אינו מצביע לאיבר האחרון בתור אלא מצביע "אחרי האיבר האחרון". למשל במקרה של תור ריק, אין איבר אחרון שהתור יכול להצביע עליו. לאיטרטור המוחזר לא ניתן לבצע Dereference.

כל שינוי של התור (דחיפה\הוצאה וכו'...) הופכת את כל האיטרטורים הקיימים של אותו תור ללא חוקיים. שימוש באיטרטור לא חוקי גורר התנהגות לא מוגדרת ולא יבדקו מקרים מסוג זה בבדיקות האוטומטיות של התרגיל.

```
for (Queue<int>::Iterator i = queue5.begin(); i != queue5.end(); ++i) {
        /* some code... */
}
```

```
const Queue<int> constQueue = queue5;
for (Queue<int>::ConstIterator i = constQueue.begin(); i != constQueue.end(); ++i) {
        /* some code... */
}
```

4.3 חריגות

על מנת להתמודד עם שגיאות נרצה להשתמש בחריגות. בתור שלנו נגדיר שני סוגי חריגות עבור שגיאות אפשריות של התור והשימוש בו.

4.3.1 חריגת תור ריק

חריגה זו תיזרק בכל פעם שנבצע פעולה שלא יכולה להתבצע על תור ריק.

```
Queue<int> queue6;
try {
        int val = queue6.front();
}
catch (Queue<int>::EmptyQueue e) {
        /* some code... */
}
```

שימו לב – ניתן להפעיל את transform ואת filter על תור ריק, אין צורך לזרוק חריגה במקרה זה.

4.3.2 חריגת פעולה לא חוקית על איטרטור

חריגה זו תיזרק בכל פעם שנבצע פעולה לא חוקית על איטרטור המצביע לסוף התור.

```
Queue<int>::Iterator endIterator = queue6.end();
try {
        ++endIterator;
}
catch (Queue<int>::Iterator::InvalidOperation& e) {
        /* some code... */
}
```

4.3.3 חריגות בהקצאת זכרון

בעת שגיאות בהקצאת זכרון, יש לדאוג שתזרק חריגה מסוג std::bad_alloc.

4.4 הערות על הממשק

- על הממשק יתמוך בתור קבוע (const) ולא קבוע.
- מסופק עבורכם קובץ בדיקה בשם QueueExampleTest.cpp.
- ממשק התור צריך לתמוך בכל דוגמאות השימוש המובאות במסמך ובקבצי הבדיקה.

4.5 מימוש של Queue

- **אין** להשתמש ב-STL לשם מימוש התור.
- על המימוש לעמוד בקונבנציות הקוד המפורטות באתר הקורס.
- מותר להגדיר פונקציות עזר לשם מימוש התור ללא הגבלה, כל עוד הן לא משנות את ממשק התור.
- **אין** להגדיר מחלקות או מתודות חברות (friend) אם אין בכך צורך.
- במימוש הפנימי של התור ניתן ואף כדאי להשתמש באלמנטים של הממשק אותו אתם נדרשים לממש (למשל איטרטורים).

5 חלק יבש

בחלק זה מספר שאלות הנוגעות לממשק התור ולשימושים בו. עליכם לענות על השאלות בקובץ הנקרא dry.pdf.

א. על מנת לאפשר איטרציה על תור קבוע (const), הממשק מאפשר לנו שימוש ב-ConstIterator בנוסף לאיטרטור הרגיל של התור. מדוע לא ניתן להסתפק בלהגדיר את פעולות האיטרטור הרגיל כ-const?

ב. באילו מהפונקציות בממשק התור קיימות הנחות על הטיפוס הטמפלייטי? עבור כל אחת מהפונקציות הללו פרטו את הנחות.

ג. סטודנט בקורס מבוא לתכנות מערכות שכח מהאזהרות שקיבל בתרגול ומימש את המחלקה Queue בקובץ cpp במקום בקובץ h. מהי השגיאה שיקבל כאשר ינסה לקמפל את התרגיל ובאיזה משלבי הקומפילציה היא מתרחשת?

ד. סטודנטית בקורס מבוא לתכנות מערכות סיימה לפתור את תרגיל בית 3, והחליטה להשתמש במימוש התור מהתרגיל לפרוייקט צד שהיא מפתחת בשעות הפנאי. במימוש פרוייקט הצד הסטודנטית נדרשה לסנן תור של מספרים שלמים, כך שישארו בתור רק מספרים המתחלקים במספר כלשהו **שאינו ידוע בזמן קומפילציה** אלא רק בזמן ריצה. הסבירו כיצד ניתן לממש את הפונקציונליות הדרושה בעזרת הפונקציה filter.

6 הגשת התרגיל

6.1 הידור קישור ובדיקה

החלק הרטוב בתרגיל ייבדק על שרת csl3 ועליו לעבור הידור בעזרת הפקודה הבאה:

```
> g++ --std=c++11 -o ex3_test -Wall -pedantic-errors -Werror -DNDEBUG *.cpp
```

לנוחיותכם, מסופקת לכם תוכנית "בדיקה עצמית" בשם finalCheck, התוכנית בודקת שקובץ ההגשה, קובץ ה-zip בנוי נכון ומריצה את הטסטים שסופקו כפי שירוצו על ידי הבודק האוטומטי. הפעלת התוכנית ע"י הפקודה:

```
~mtm/public/2122b/ex3/finalCheck <submission>.zip
```

הקפידו להריץ את הבדיקה על קובץ ההגשה. אם אתה משנים אותו, הקפידו להריץ את הבדיקה שוב.

6.2 ולגרינד ודליפות זיכרון

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. על כן עליכם להשתמש ב-valgrind שמתחקה אחר ריצת התכנית שלכם, ובודקת האם ישנם משאבים שלא שוחררו. הדרך לבדוק אם יש לכם דליפות בתכנית היא באמצעות שתי הפעולות הבאות (שימו לב שחייב להיות main, כי מדובר בהרצה ספציפית):

1. קימפול של התרגיל בעזרת השורה לעיל עם הדגל g-.
2. הרצת השורה הבאה:

   valgrind --leak-check=full ./ex3_test

   כאשר ex3_test הוא שם קובץ ההרצה.

6.3 בדיקת התרגיל

החלק היבש של התרגיל יבדק ידנית. הקוד שתכתבו בחלקים 3 ו-4 יעבור בדיקה רטובה ובנוסף יעבור בדיקה יבשה של הקוד על מנת להעריך את איכות הקוד (שכפולי קוד, קוד מבולגן, קוד לא ברור, שימוש בטכניקות תכנות "רעות"). בדיקה רטובה כוללת את הידור התכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח בבדיקה שכזו, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות ללא דליפות זיכרון.

6.4 הגשה

את ההגשה יש לבצע דרך אתר הקורס, תחת

Assignments -> HW3 -> Electronic Submit.

הקפידו על הדברים הבאים:

```
submission.zip
├── Queue.h
├── HealthPoints.h
├── dry.pdf
└── ...possibly other files
```

- יש להגיש את קבצי הקוד יחד עם קובץ תשובות לחלק היבש מכווצים לקובץ zip (לא פורמט אחר).
- **אין להגיש אף קובץ מלבד פתרון לחלק היבש, קבצי h וקבצי cpp אשר כתבתם בעצמכם.**
- את הפתרון לחלק היבש ניתן יש להגיש כקובץ pdf בלבד.
- הקבצים אשר מסופקים לכם יצורפו על ידנו במהלך הבדיקה, וניתן להניח כי הם יימצאו בתיקייה הראשית.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.
- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית שימרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף. כמו כן שימרו עותק של התרגיל על חשבון ה-Google Drive/Github/CSL3 שלכם לפני ההגשה האלקטרונית ואל תשנו אותו לאחריה (שינוי הקובץ יגרור שינוי חתימת העדכון האחרון). כל א מצעי אחר לא יחשב הוכחה לקיום הקוד לפני ההגשה.

בהצלחה!