מבוא לתכנות מערכות

תרגיל בית מספר 4 – basic C++

סמסטר אביב 2018

תאריך פרסום: 27 במאי 2018

תאריך הגשה: 10 ביוני 2018

משקל התרגיל: 5% מהציון הסופי (תקף)

מתרגלת אחראית: **לינוי בוכניק**

1 הערות כלליות

- **שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.**
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות של אחד מהמתרגלים, או לשאול בפורום של הקורס במודל. לשאלה לגבי הניסוח אפשר לפנות ללינוי במייל. לפני שליחת שאלה - נא וודאו שהיא לא נענתה כבר בFAQ - או במודל ,ושהתשובה אינה ברורה ממסמך זה, מהדוגמא ומהבדיקות שפורסמו עם התרגיל
- **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, הכתוב שם מחייב
- **העתקות בתוכנה תטופלנה בחומרה**
- **מומלץ מאוד מאוד לכתוב את הקוד בחלקים קטנים, לקמפל כל חלק בנפרד על השרת, ולבדוק שהוא עובד באמצעות שימוש בטסטים. קראו את סעיף 5– דגשים ורמזים – לפני תחילת העבודה על הקוד!**

חלק יבש 2

לפניכם מופיע קוד למבנה נתונים מחסנית. הקוד מכיל שגיאות קומפילציה ושגיאות קונבנציה וקריאות (עיקבו אחר העקרונות שנלמדו בכיתה). מצאו לפחות שמונה שגיאות בקוד וכתבו את הקוד המתוקן.
דגש: שימו לב שמכיוון שטרם למדנו בכיתה טיפול בשגיאות בC++, חוסר הטיפול בשגיאות הוא אינו חלק משמונה השגיאות שעליכם למצוא.
עבור כל שגיאה הסבירו בקצרה לגבי מהותה.

```
class Stack {
    const int size;
    char** const data;
    int currSize;
public:
    Stack(int size): size(size),data(new char*[size]),currSize(0){}
    ~Stack() {
        delete(data);
    }

    void push(char* stringToAdd) {
        data[currSize++] = stringToAdd;
    }
    char* get() {
        return data[currSize];
    }
    void pop() {
        free(data[--currSize]);
    }

    void pop(int numOfElemToPop) {
        for (int i = 0; i < numOfElemToPop; i++) {
            pop();
        }
    }
    void pop(double numOfElemToPop) {
        for (int i = 0; i < numOfElemToPop; i++) {
            pop();
        }
    }

    friend void changeStackSize(int newSize,Stack stack) {
        stack.size = newSize;
        data_temp = new char*[newSize];
        for (int i = 0; i < stack.currSize; i++) {
            data_temp[i] = stack.data[i];
        }
        delete[] data;
        data = data_temp;
    }
```

```
    bool operator==(Stack& stack) {
        if (currSize != stack.currSize) {
            return false;
        }
        for (int i = 0; i < stack.currSize; i++) {
            if (data[i] != stack.data[i]) {
                return false;
            }
        }
        return true;
    }
};
```

3 חלק רטוב

בתרגיל זה, נבנה ממשק למשחק מלחמה רב משתתפים הנקרא **Call Of Matam Duties**. בחלק זה של התרגיל תממשו שלוש מחלקות:

-מחלקה שתייצג שחקן - Player
-מחלקה שתייצג כלי נשק Weapon
-מחלקה שתייצג משחק - Game

מחלקת שחקן – Player :

לכל שחקן יש:

1. שם: name – משתנה מטיפוס *char.
2. הדרגה שבה הוא נמצא: level – משתנה מטיפוס int.
3. מספר החיים שנותרו לו: life –משתנה מטיפוס int.
4. הכוח שלו: strength – משתנה מטיפוס int.
5. כלי נשק שבו ישתמש – weapon.
6. מיקום של השחקן – int מספר שלם המצביע על משבצת המשחק שהשחקן נמצא בה. לדוגמא אם הערך הוא 5, השחקן נמצא במשבצת מספר 5.
7. על המחלקה לתמוך בפעולות הבאות:

א. בנאי –

$$\boldsymbol{Player(const\ char * name, const\ Weapon\&\ weapon)}$$

הבנאי יקבל: שם שחקן ויאתחל את name להיות שם השחקן, כלי נשק- weapon ויאתחל את weapon כמו כן יאתחל ב-1 את השדות $life, strength, level$ ואת מיקום השחקן יאתחל ב-0.

ב. הורס.

$$\boldsymbol{\sim Player()}$$

ג. אופרטור הדפסה, שידפיס שחקן בפורמט הבא:

{player name: name, weapon: weapon}

ד. מתודה nextLevel – המקדמת את ה level ב-1, הפונקציה אינה מקבלת ואינה מחזירה דבר.

$$\boldsymbol{void\ nextLevel()}$$

ה. מתודה isPlayer המקבלת מחרוזת מחזירה true אם זה שם השחקן ו-false אחרת.

$$\boldsymbol{bool\ isPlayer(const\ char * playerName)\ const}$$

ו. מתודה makeStep – המקדמת את השחקן בצעד אחד (למשבצת הבאה במשחק) – מגדילה את המיקום ב-1, המתודה אינה מקבלת ואינה מחזירה דבר.

$$\boldsymbol{void\ makeStep()}$$

ז. מתודה addLife המעלה את מספר החיים של השחקן ב-1 , הפונקציה אינה מקבלת ואינה מחזירה דבר.

$$\boldsymbol{void\ addLife()}$$

ח. מתודה addStrength המקבלת ערך ומוסיפה אותו לערך strength , הפונקציה אינה מחזירה דבר.

$$\boldsymbol{void\ addStrength(int\ strengthToAdd)}$$

ט. מתודה isAlive המחזירה true אם כל הערכים – level,life,strength גדולים מ-0 ו-false אחרת. (כלומר מספיק שערך אחד יהיה שווה ל-0 והשחקן מת)

$$\boldsymbol{bool\ isAlive()\ const}$$

י. מתודה weaponIsWeak(int weaponMinStrength) המחזירה true אם הערך (value) של הנשק של השחקן קטן מ- weaponMinStrength ו-false אחרת.

$$\boldsymbol{bool\ \mathbf{weaponIsWeak(int\ weaponMinStrength)}\ const}$$

יא. אופרטורי השוואה (<, >) כאשר ההשוואה תעשה לפי השם בסדר לקסיקוגרפי.

יב. מתודה fight –
(שימו לב שסעיף זה מתבסס על מחלקת weapon המפורטת בהמשך.)
המתודה תדמה לחימה בין 2 שחקנים באופן הבא:
השחקן עם כלי הנשק הטוב יותר יתקיף (כלי הנשק הטוב ביותר הוא כלי הנשק המקיים ש$weapon1 > weapon2$), כאשר תקיפה מוגדרת באופן הבא:
אם כלי הנשק תוקף את ה- level של השחקן ומוריד לו 3 נק', אז לשחקן שהותקף ירדו 3 נק מה-level.
באופן דומה גם עבור כלי נשק התוקפים את ה-life ואת ה-strength. שימו לב שאם לאחר ההורדה הערך הפך לשלילי, יש לאפס אותו. (כלומר כלי הנשק לא יכול להוריד את הערך לפחות מ-0)
שימו לב ששחקנים יכולים להלחם רק במידה ושניהם נמצאים באותה משבצת.
המתודה מחזירה false עם המתקפה לא הצליחה (הדבר קורה אם 2 השחקנים לא באותה משבצת או אם לשניהם יש כלי נשק עם עוצמה זהה) אחרת, מחזירה true .

$$\boldsymbol{bool}\ \mathbf{fight}(\mathbf{Player}\&\ \mathbf{player})$$

מחלקת Weapon:

לכל כלי נשק יש:

1. שם: name – ערך מסוג char*.
2. במה הוא פוגע: target המקבל אחד מ- 3 הערכים: level, life or strength
השתמשו בenum הבא:

```
enum Target
{
        LEVEL,
        STRENGTH,
        LIFE
};
```

3. כמה נזק הוא עושה- hitStrength – ערך מסוג int.
4. על המחלקה לתמוך בפעולות הבאות:

א. בנאי - בנאי יקבל את שם הנשק (כמחרוזת), target, ו-hitStrength , ויאתחל את השדות בהתאם:

$$\boldsymbol{Weapon(const\ char * name, Target\ target, int\ hit\_strength)}$$

ב. הורס:

$$\boldsymbol{\sim Weapon()}$$

ג. מתודה getTarget המחזירה במה כלי הנשק פוגע.

$$\boldsymbol{Target\ getTarget()\ const}$$

ד. מתודה getHitStrength המחזירה כמה נק' כלי הנשק מוריד.

$$\boldsymbol{int\ getHitStrength()\ const}$$

ה. המתודה getValue המחזירה את הערך של הנשק שיוגדר באופן הבא:
אם כלי הנשק פוגע ב- level, תחזיר ערך פגיעה - $1 * \text{hit\_strength}$
אם כלי הנשק פוגע ב-strength, תחזיר ערך פגיעה - 2* hit_strength
אם כלי הנשק פוגע ב-life, תחזיר ערך פגיעה - 3* hit_strength

$$\boldsymbol{int\ getValue()\ const}$$

ו. אופרטור שוויון (==), אופרטור אי שוויון (=!) ואופרטורי השוואה (<, >) כאשר ההשוואה תעשה לפי הvalue – לדוגמא $weapon1 < weapon2$ אם
$weapon1.getValue() < weapon2.getValue()$

ז. אופרטור הדפסה, שידפיס כלי נשק בפורמט הבא:

$$\{weapon\ name: name, weapon\ value: weapon.getValue()\}$$

מחלקת GAME :

המחלקה מכילה:

1. מספר השחקנים המקסימלי במשחק - maxPlayers
2. מערך של מצביעים לשחקנים
3. על המחלקה לתמוך בפעולות הבאות:

א. בנאי המאתחל מערך בגודל maxPlayers.

$$\boldsymbol{Game(int\ maxPlayer)}$$

ב. הורס:

$$\boldsymbol{\sim Game()}$$

ג. מתודה addPlayer המקבלת את כל הפרמטרים הנחוצים ליצירת שחקן חדש עם נשק חדש.

$$\boldsymbol{GameStatus\ addPlayer(const\ char*\ playerName,}$$
$$\boldsymbol{const\ char*\ weaponName, Target\ target, int\ hit\_strength)}$$

שגיאות:

אם קיים שחקן עם שם זהה – NAME_ALREADY_EXISTS

אם אין יותר מקום במערך – GAME_FULL

במידה והפעולה הצליחה החזירו - SUCCESS

ד. מתודה nextLevel – המקבלת string – שם של שחקן ומקדמת את לו את הlevel.

$$\boldsymbol{GameStatus\ \ nextLevel(const\ char*playerName)}$$

שגיאות:

אם לא קיים שחקן בשם זה: NAME_DOES_NOT_EXIST.

ה. מתודה makeStep – המקבלת string – שם של שחקן ומקדמת אותו בצעד.

$$\boldsymbol{GameStatus\ \ makeStep(const\ char*playerName)}$$

שגיאות:

אם לא קיים שחקן בשם זה: NAME_DOES_NOT_EXIST.

במידה והפעולה הצליחה החזירו - SUCSSES

ו. מתודה addLife – המקבלת string – שם של שחקן ומעלה לו את החיים ב1.

$$\boldsymbol{GameStatus\ \ addLife(const\ char*playerName)}$$

שגיאות:

אם לא קיים שחקן בשם זה: NAME_DOES_NOT_EXIST.

במידה והפעולה הצליחה החזירו - SUCCSES

ז. מתודה addStrength – המקבלת string – שם של שחקן וערך StrenghToAdd, ומעלה לו את הstrength בStrenghToAdd .

$$\boldsymbol{GameStatus\ \ \mathbf{addStrength}\ (const\ char*playerName, int\ strengthToAdd)}$$

שגיאות:

אם לא קיים שחקן בשם זה: NAME_DOES_NOT_EXIST.

אם התקבל X שלילי –INVALID_PARAM .

במידה והפעולה הצליחה החזירו - SUCCSES

ח. מתודה removeAllPlayersWithWeakWeapon(int weaponStrength) המקבלת מספר מסויים ומסירה את כל השחקנים שכלי הנשק שלהם חלש יותר מהמספר הנ"ל. מחזירה true אם הוסרו שחקנים ו-false אחרת.

$$\boldsymbol{bool\ \mathbf{removeAllPlayersWithWeakWeapon(int\ weaponStrength)}}$$

ט. מתודת fight המקבלת שתי מחרוזות המייצגות שמות של 2 שחקנים ומפעילה את מתודת fight עליהם.
אם בסוף הקרב אחד השחקנים לא בחיים (מתודת isAlive מחזירה false), יש להסירו מהרשימה.
$GameStatus\ fight(const\ char * playerName1, const\ char * playerName2)$
שגיאות:
אם לא קיים שחקן בשם זה: NAME_DOES_NOT_EXIST .
אם הקרב נכשל (fight החזיר false) החזירו FIGHT_FAILED
במידה והפעולה הצליחה החזירו - SUCCSES

י. אופרטור הדפסה, שידפיס את כל השחקנים שבמשחק בפורמט הבא ממויינים ע"פ השמות בסדר לקסיקוגרפי עולה :

player 0: player0,
player 1: player1,
player 2: player2,..

שימו לב שבדוגמא המצורפת יש את פורמט ההדפסה המדוייק.

השגיאות האפשריות במחלקה:
INVALID_PARAM
NAME_ALREADY_EXISTS
GAME_FULL
NAME_DOES_NOT_EXIST
FIGHT_FAILED
SUCCESS

הערות נוספות:

1. **במחלקות הנ"ל אין התייחסות לבנאי העתקה והשמה. באחריותכם להחליט אם יש צורך להוסיף אותם. יירדו נקודות על אי מימוש או שימוש לא תקין. אל תשכחו לציין במידה והשתמשתם בבנאים הדיפולטיבים.**

4 הגשה

הגשה יבשה:

יש להגיש לתא הקורס את פתרון החלק היבש של התרגיל – מודפס משני צדי הדף .
אין להגיש את הקוד שכתבתם בחלק הרטוב.

הגשה רטובה :

את ההגשה הרטובה יש לבצע דרך אתר הקורס, תחת Assignments 4 → Exercise → Electronic submission

הקפידו על הדברים הבאים:

- יש להגיש את קבצי הקוד מכווצים לקובץ zip (ולא לפורמט אחר) כאשר כל הקבצים עבור הפתרון מופיעים בתיקיית השורש בתוך קובץ הzip. הקבצים שעליכם להגיש: Game.cpp, Game.h, Player.cpp, Player.h, Weapon.cpp, Weapn.h
- אין להגיש אף קובץ מלבד קבצי h וקבצי cpp אשר כתבתם.
- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית:
  - שימרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף .
  - שימרו עותק של התרגיל על חשבון ה csl2שלכם לפני ההגשה האלקטרונית ואל תשנו אותו לאחריה (שינוי הקובץ יגרור שינוי חתימת העדכון האחרון)
  - כל אמצעי אחר לא יחשב הוכחה לקיום הקוד לפני ההגשה .
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.

הידור קישור ובדיקה:

השתמשו הפקודת הקומפילציה הבא:

```
>g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG *.cpp  -o [program name]
```

משמעות הפקודה:

- **-std=c++11** שימוש בסטנדרט החדש של $C + 11$
- **-o [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
- **-DNDEBUG** מוסיף את השורה #define NDEBUG בתחילת כל יחידת קומפיציה. בפועל מתג זה יגרום לכך שהמאקרו nassert לא יפריע ולא יופעל בריצת התוכנית.

5 דגשים ורמזים

- הקפידו על כללי הזהב הבאים כדי להקל על העבודה
  - הקוד תמיד מתקמפל - בכל שלב של העבודה, ניתן לקמפל את הקוד ללא שגיאות או להגיע למצב זה על ידי מספר קטן של שינויים
  - כל הקוד שנכתב נבדק - ודאו בכל הרצה שכל הבדיקות עוברות. הקפידו להיות במצב שבו על ידי לחיצת כפתור ניתן להריץ בדיקה אוטומטית לכל הקוד שכבר נכתב. דבר זה יחסוך זמן דיבוג יקר. אם לא נתונות לכם בדיקות, כתבו אותן בזמן כתיבת הקוד עצמו.
- יש להקפיד על תכנון נכון של התוכנית וכתיבה נכונה בcpp
- קוד מסורבל ללא צורך אמיתי ייאבד נקודות
- אתם רשאים להניח כי לא קיימות שגיאות נוספות מלבד אלו שפורטו
- מומלץ להשתמש במאקרו assert המוגדר בקובץ assert.h כפי שהוסבר בתרגולים
  - במקרה של הסתבכות עם באג קשה: בודדו את הבאג ושחזרו אותו בעזרת קבצי בדיקה (main) קטנים ככל הניתן. ניתן לעשות זאת על ידי comment לחלק מהשורות בקוד ובדיקה אם הבאג מתרחש, והמשך בהתאם. שיטה זו פותרת כמעט כל באג אפשרי, וברוב המקרים ביעילות .

- ניתן להשתמש בדיבאגר או בהדפסות זמניות כדי לבדוק את מצב התכנית בכל שלב. אנו ממליצים להפנים ולהשתמש לפחות באחת משתי האפשרויות האלה על מנת לדבג את הקוד בהצלחה.

עבודה נעימה!