# מבוא לתכנות מערכות, סמסטר אביב 2018

# תרגיל בית מספר 5 – C++ - המשך

תאריך פרסום: 11/06/2018

תאריך הגשה: 01/07/2018

משקל התרגיל: 5% (תקף)

מתרגל אחראי: **יורי פלדמן**

## הערות כלליות

- שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.
- לשאלות בנוגע להבנת התרגיל יש לפנות ל:
  - לשאול בפורום של הקורס במודל
  - סדנאות של אחד המתרגלים
  - לבדוק ב-FAQ של התרגיל באתר הקורס

לפני פרסום שאלה נא עברו על הודעות קודמות בפורום / FAQ – בהרבה מקרים שאלתכם כבר נענתה.

- קראו מסמך זה עד סופו לפני שאתם מתחילים לממש. תכננו את המימוש שלכם לפני שאתם ניגשים לעבוד.
- **חובה** להתעדכן בעמוד ה-FAQ של התרגיל. הכתוב שם מחייב.
- **העתקות קוד תטופלנה בחומרה**

## חלק א' – יבש

1. נתון הקוד הבא

```
#include <iostream>

class Base {
	virtual void method() {std::cout << "from Base" << std::endl;}
public:
	virtual ~Base() {method();}
	void baseMethod() {method();}
};

class A : public Base {
	void method() override {std::cout << "from A" << std::endl;}
public:
	~A() override {method();}
};

int main(void)
{
	Base* base = new A;
	base->baseMethod();
	delete base;
	return 0;
}
```

א. רשמו מה ידפיס הקוד הנ"ל והסבירו מדוע.

ב. כיצד נוכל לשנות את השורה הראשונה בפלט על ידי שינוי המחלקה Base בלבד? הציגו את הפלט החדש והסבירו את תשובתכם.

ג. נרצה להוסיף קוד למחלקות הנתונות כך ששורת היצירה של האובייקט תדפיס את השורות

from Base
from A

כיצד תעשו זאת מבלי לשנות או להוסיף פונקציות הדפסה?

2. להלן הגדרה של מחלקת חריגות ושלוש דרכים לתפיסת חריגה

```
class MyExceptionClass : public std::exception
{
	// ... some private fields
public:
	MyExceptionClass()                              {cout << "Ctor" << endl;          }
	MyExceptionClass(MyExceptionClass const&)       {cout << "Copy Ctor" << endl;    }
	MyExceptionClass& operator=(MyExceptionClass const&)
                                                        {cout << "operator =" << endl;   }
	~MyExceptionClass()                             {cout << "Dtor" << endl;         }
};

try {
	throw MyExceptionClass();
}
```

1)
```
catch (MyExceptionClass ex)
{
	// ... handling
}
```

2)
```
catch (MyExceptionClass const* ex)
{
	// ... handling
}
```

3)
```
catch (MyExceptionClass const& ex)
{
	// ... handling
}
```

איזו מהדרכים עדיפה? מדוע (ומה החסרון בדרכים האחרות)?

# חלק ב' – מערך גנרי ב- C++

ממשו בקובץ mtm_array.h תבנית עבור מחלקת מערך בשם array. על ממשק התבנית להיות בדומה למחלקה Array משקף 10-11 מהרצאה 9, שקף 25 מתרגול 9, אך עם פרמטר תבנית נוסף – גודל המערך (בדומה ל-std::array, שיוזכר בעתיד), כלומר התבנית תהיה תואמת להגדרה.

```
template <class T, int SIZE>
class Array {
      // TODO: your code here
};
```

על התבנית לתמוך (לפחות) בפעולות:

- בנאי חסר ארגומנטים, בנאי העתקה, אופרטור=
- אופרטור [] לגישה לאינדקס (שימו לב לגרסה למערך const!). **במקרה של ניסיון גישה לאינדקס לא חוקי, יש לזרוק עצם מטיפוס std::out_of_range** (נדרש #include <stdexcept>) עם טקסט כלשהו , באופן הבא

```
#include <stdexcept>
... main definition, etc
throw std::out_of_range("mtm_array::operator []: index out of range!")
```

- פעולות begin() ו-end(), שתאפשרנה מעבר על מערך קבוע או לא, וטיפוסי האיטרטור המתאימים (פירוט בהמשך).

<u>הדרכה למימוש הממשק לאיטרציה על המערך:</u>

יש לממש מתודות begin() ו-end() עם החתימות הבאות

```
iterator begin();
const_iterator begin() const;
iterator end();
const_iterator end() const;
```

כאשר iterator ו-const_iterator מוגדרים כטיפוסים (מחלקה או typedef) בתוך mtm::array, והממשק הנדרש עבורם הוא (לפחות) פעולות קידום (x++, ++x), השוואה (==, !=), אופרטור dereference (*x) ואופרטור -> (בדומה לשקפים 23-29 בהרצאה 9). הטיפוסים iterator ו-const_iterator ייבדלו ע"י החתימה של פעולות הגישה לאיבר המערך: אופרטור * ואופרטור -> יחזירו ייחוס ומצביע שהוא const עבור const_iterator, וניתן לעריכה עבור iterator. ראו דוגמא בקובץ mtm_array.h המסופק עם התרגיל.

תוספות:

- לנוחיותכם, הממשקים הנ"ל הוגדרו חלקית בקובץ mtm_array.h.
- **במקרה של ניסיון גישה לאיבר מחוץ למערך, יש לזרוק std::out_of_range כפי שפורט מעלה.**

הממשק שהגדרנו בסעיף זה ייאפשר מעבר על איברי המערך ע"י:

```
using std::string;

Array <string, 10> arr;
for (Array<string, 10>::iterator it=arr.begin(); it != arr.end(); ++it)
      // ... do something

const Array <string, 5> arr2;
for (Array<string, 5>::const_iterator it=arr.begin(); it != arr.end(); ++it)
      // ... do something not modifying arr2
```

# חלק ג' – הרחבה לתרגיל הקודם

בעקבות הצלחת המשחק CoMD (Call of Matam Duties), הוחלט על פיתוח גרסה חדשה (בשם World of Homework, WoH), שתספק תשתית להרחבות עתידיות נוספות, תוך שימוש בכלים מתקדמים של שפת ++C. כדי לשמור על תאימות אחורה הוחלט להשאיר את ממשקי המחלקות מהתרגיל הקודם בעינם, פרט לשימוש ב-std::string בכל מקום בו מופיע *char, וכן שימוש במנגנון החריגות לטיפול (בחלק מה-) שגיאות (יתואר להלן).

1. <u>טיפול בחריגות</u>

מסופק לכם קובץ mtm_exceptions.h המגדיר את טיפוסי החריגות בתרגיל. עבור טיפוסים המתאימים לקודי שגיאה מ-GameStatus (שם טיפוס החריגה מתאים לשם הקוד ב-enum), חריגה צריכה להיזרק במצבים בהם הוחזר הקוד המתאים. עבור טיפוסי החריגות החדשים – מופיע תיאור להלן.

**<u>עריכה</u>: ראו הבהרה ופירוט הנושא בעמוד הבא.**

2. <u>הרחבה לטיפול בסוגי שחקנים שונים</u>

בגרסה החדשה של המשחק, נרצה להבחין בין 3 סוגי דמויות עבור Player: Wizard, Warrior, Troll (זאת כהכנה להרחבה לסוגי דמויות נוספים). סוגי הדמויות יהיו שונים בהתנהגות וביכולות שלהם.

**תוספת / הבהרה: יש <u>למנוע</u> יצירת עצמים מטיפוס Player**, בדרכים שנלמדו בשיעורים.

לוחם Warrior יהיה דומה ל-Player מ-CoMD (גרסת המשחק הישנה), עם ההבדל שהוא יכול להיות רכוב או לא (פרמטר bool לבנאי). עבור לוחם רכוב, הצעד יהיה של 5 יחידות (makeStep תוסיף 5 למיקום) במקום 1 עבור לוחם רגלי. בנוסף, לוחם יכול להשתמש בנשק שפוגע בכח (strength) או בחיים (life), אך לא ברמה (level).

קוסם Wizard יקבל פרמטר נוסף לבנאי והוא טווח range (שלם). קוסם יוכל לתקוף שחקן אחר ממרחק שהוא קטן או שווה לטווח שלו (המרחק הוא ההפרש בערך מוחלט בין מיקומי הדמויות). מנגד, קוסם לא יוכל לתקוף דמות שנמצאת באותו תא שטח איתו. בנוסף, קוסם יכול להשתמש בנשק שפוגע בכח (strength) או ברמה (level), אך לא בחיים (life).

טרול Troll מיוחד בכך שהוא יכול לשאת כל נשק, בהיותו יצור ענק הוא רץ 2 תאי שטח לכל הפעלה של makeStep, ובנוסף בכל הפעלה של makeStep הוא נרפא ביחידת חיים אחת (הפעלה בודדת של addLife). עם זאת, ערך ה-life שלו לא יכול לעלות על ערך מקסימלי שניתן כפרמטר לבנאי (הפעלות נוספות של addLife לא משנות).

נשתמש בפולימורפיזם כדי לשלב את סוגי הדמויות השונים (ועתידיים) בתשתית הקיימת. נרצה ליצור מחלקה חדשה לכל אחד מהסוגים, היורשת מ-Player (שקלו להפוך חלק הכרחי משדות Player ל-protected). בנוסף, עבור כל אחת מהמחלקות החדשות, נצטרך להוסיף פונקציית ממשק למחלקה Game שמוסיפה שחקן מהסוג המבוקש. לבסוף, נעדכן את כללי ההתקפה כך ~~שאם רק אחד הצדדים יכול להילחם (לדוג' בגלל טווח), אזי הקרב מצליח והוא המתקיף~~ **עדכון/הקלה:** תנאי הטווח <u>יתווסף</u> לתנאי על יחס הנשקים. דהיינו כדי ששחקן יוכל לתקוף, על השחקן המותקף להיות בטווח שלו (ולא בתא הנוכחי עבור קוסם, כן בתא הנוכחי לשתי הדמויות האחרות), ובנוסף – שהנשק שלו יהיה חזק ממש מהנשק של הדמות המותקפת. . <u>הדרכה</u>: דרך אפשרית היא ליצור במחלקה player פונק' distance סטטית עם גישה protected, וכן פונק' canAttack וירטואלית, הקובעת האם this יכול להיות המתקיף מול שחקן אחר. במחלקה Wizard ניתן להחליף (override) את canAttack במימוש תוך שימוש ב-distance.

הטבלה הבאה מסכמת את המחלקות החדשות והממשקים שמתווספים עבורן (זאת בנוסף לממשקים אחרים שעליכם להוסיף על פי הצורך).

| שם המחלקה | בנאי | ממשק Game |
|---|---|---|
| Wizard | `Wizard(string const& name, Weapon const& weapon, int range);`<br>אם range<0, זורק InvalidParam<br>אם לא יכול להשתמש בנשק, זורק IllegalWeapon | `void addWizard(string const& playerName, string const& weaponName, Target target, int hitStrength, int range);` |
| Troll | `Troll(string const& name, Weapon const& weapon, int maxLife);`<br>אם maxLife ≤ 0, זורק InvalidParam | `void addTroll(string const& playerName, string const& weaponName, Target target, int hitStrength, int maxLife);` |
| Warrior | `Warrior(string const& name, Weapon const& weapon, bool rider);`<br>אם לא יכול להשתמש בנשק, זורק IllegalWeapon | `void addWarrior(string const& playerName, string const& weaponName, Target target, int hitStrength, bool rider);`<br>כדי לשמר תאימות אחורה, על פונק' addPlayer הנוכחית ליצור Warrior עם rider=false (ב-C++14 ניתן יהיה לסמן פונקציה כ-[[deprecated]]). |

3. <u>הרחבה לטיפול בסוגי שאילתות שונות בהסרת שחקנים</u>

נרצה להרחיב את הממשק של removeAllPlayersWithWeakWeapon לתמוך בכל קריטריון אחר המוגדר ע"י המשתמש, המועבר בצורה של function object, כך שאופרטור סוגריים () המופעל על Player מחזיר true לשחקן שיש להסירו מהמשחק, ו-false אחרת.

נממש זאת ע"י תוספת לממשק של Game של תבנית פונק' שמקבלת function object שחתימתה

```
template <class FCN>
bool removePlayersIf(FCN& fcn);
```

המחלקה תפעיל את operator() של fcn על כל אחד מהשחקנים, ותסיר את אלו עבורם התקבל true. <u>שימו לב</u>: על fcn "לעבוד" על עצמים מטיפוס Player const& (כיצד ניתן להבטיח זאת?).
בנוסף, על המימוש של פונק' הממשק הקיימת removeAllPlayersWithWeakWeapon לעשות שימוש בממשק החדש.

## <u>הבהרה לממשקים וטיפול בחריגות:</u>

מטרתנו לשמור על הממשק מהתרגיל הקודם (תאימות אחורה) - ז"א שכל פונק' ממשק של Game, Player או Weapon שהוגדרה בתרגיל הקודם שומרת על החתימה שלה ללא כל שינוי - **<u>אין לשנות חתימות אלו</u>*** (*פרט למעבר ל-std::string, שהינו שקוף מבחינת המשתמש (למה?) ).

**כמו כן** נצטרך לשמור על <u>ההתנהגות</u> של הפונקציות מתרגיל הקודם (מבחינת המשתמש). ז"א שפונקציות ממשק מהתרגיל הקודם צריכות להחזיר בדיוק את אותם ערכים באותם מצבים (ולא לזרוק חריגות), וכן נוסף <u>קוד שגיאה</u> חדש ILLEGAL_WEAPON שיש להחזיר במצבים המתאימים.

לעומת זאת הפונקציות החדשות שנוספות, לא נתונות תחת מגבלות אלו, והממשק שלהן שונה (ערך חזרה void, לכן עליהן לזרוק חריגות במקרה שגיאה).

**לסיכום: אין לשנות חתימות של פונקציות ממשק מהתרגיל הקודם. אין לזרוק חריגות מפונקציות ממשק של Game מהתרגיל הקודם** (פרט ל-Game::fight, פירוט בהמשך). יש לעבוד באופן פנימי עם חריגות (לטובת נוחות, אחידות ומניעת שכפול קוד). יש לתפוס חריגות אלו בתוך פונקציות הממשק הישן ולהחזיר את הערכים המתאימים.

ספציפית הפונקציה fight, מכיוון שהתנהגותה בלאו הכי משתנה בגלל הכללים שנוספו, מתעדכן גם הממשק שלה לתאום את הממשק החדש, ז"א שבמקרה של שגיאה היא זורקת חריגה. עם זאת, המצב FIGHT_FAILED

אינו מצב שגיאה, אלא חלק ממהלך משחק תקין (למה?). מהסיבה הזו אין לזרוק במצב זה חריגה, אלא יש להחזיר סטטוס. וכך הערכים ש fight-עלולה להחזיר נותרים רק SUCCESS או FIGHT_FAILED.

## הערות ותוספות לחלק ג':

- **על הקוד לתמוך בממשק התרגיל הקודם** (עד כדי השינויים שצויינו), כולל כלל הפונקציונליות שמומשה (הדפסה וכו').
- אין להשתמש ב-*char. השתמשו ב-std::string.
- אין להקצות מערכים (עם []new). השתמשו בטיפוס המערך Array שהגדרנו, או מבני נתונים מ-STL לנוחיותכם. **עדכון/הבהרה:** כיוון שלא ניתן להשתמש ב-Array כפי שהוגדר כאשר הגודל לא ידוע בזמן קומפילציה (מדוע?), ייתכן (אם אתם לא משתמשים ב-STL) ועליכם לממש טיפוס עבורו זה אפשרי (בדומה להרצאה). ניתן לממש טיפוס זה בחלק ב' (ואז לרשת את Array ממנו). מימוש כמחלקה נפרדת בחלק ג' יתקבל גם כן. שימו לב שמרבית הקוד צפוי להיות משותף בין המחלקות (כולל מימוש האיטרטורים), וניתן לצמצם זאת ע"י ירושה כנ"ל.
- בחלק זה ניתן ומומלץ להשתמש ב-STL, גם בטיפוסים שלא נלמדו.
- השתמשו ב-typedef על מנת לקצר שמות טיפוסים ארוכים ולאפשר החלפה פשוטה של טיפוסים / מבני נתונים.
- יש תמיד לסמן מתודות וירטואליות מועמסות ב-override. יש תמיד לסמן ב-default= מתודות ברירת מחדל שאתם משתמשים בהן (חוץ מהמקרה הברור שמימוש המחלקה ריק). (הנחיות אלו הן בגדר תזכורת ותקפות לכל התרגיל).

# 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➔HW 5➔Electronic Submit

הקפידו על ההנחיות הבאות:

- 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, Weapon.h, Wizard.cpp, Wizard.h, Troll.cpp, Troll.h, Warrior.cpp, Warrior.h, mtm_array.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 (**תמונת מסך**). עדיף לשלוח גם לשותף.
  - שימרו עותק של התרגיל על חשבון ה-csl2 שלכם 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 בפקודת הקומפילציה הבאה:

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

**משמעות הפקודה:**

- **-std=c++11** שימוש בתקן השפה 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 קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו assert לא יופעל בריצת התוכנית.

אם המהדר בשרת לא מזהה את הדגל -std=c++11, הריצו את הפקודה

```
. /usr/local/gcc4.7/setup.sh
```

# 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.

- ראו הרצאה 5 בנושא.

- יש להקפיד על תכנון נכון של התוכנית וכתיבה נכונה ב cpp
- קוד מסורבל ללא צורך אמיתי ייאבד נקודות
- מומלץ להשתמש במאקרו assert המוגדר בקובץ cassert 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!**