# מבוא לתכנות מערכות - 234122
# תרגיל בית מספר 4 (C++)

## סמסטר אביב 2017

תאריך פרסום: 14/6/2017
תאריך הגשה: 3/7/2017
משקל התרגיל: 10% מהציון הסופי (תקף)
מתרגל אחראי לתרגיל: אמיר שגיב , amirmtm17@gmail.com

תרגיל זה מחולק לשלושה חלקים, כך שניתן להתחיל לפתור אותו לפני שנלמד כל החומר עבור התרגיל.
**חלק ראשון** – מחלקות והעמסת אופרטורים.
**חלק שני** – תבניות.
**חלק שלישי** – ירושה, פולימורפיזם ו- STL.

כל ההודעות הנוגעות בתרגיל ימצאו באתר של הקורסhttp://webcourse.cs.technion.ac.il/234122 בעמוד ה-FAQ. דף זה יכיל הבהרות וכן שאלות ותשובות נפוצות. רק הודעות דחופות תשלחנה בדואר. עליכם לעקוב אחר האתר והעדכונים שיפורסמו בו.

תרגיל בית זה נכתב בצורה המפורטת ביותר. אנא וודאו שאתם עוברים על כל הפרטים, כמו כן על ה- FAQ והקבצים המצורפים. לכל שאלה על התרגיל יש לפנות לאמיר שגיב במייל. לכל נושא אדמיניסטרטיבי (בקשת דחיה עקב מילואים, מקרים רפואיים, וכו') יש לפנות לתומר גולני.

# בהצלחה!

ESCAPE ROOM

# חלק יבש

1. א. בשאלה זו עליכם לממש אלגוריתם אשר עובד על שני אוספים גנריים. על האלגוריתם לקבל את האוספים וכן אובייקט המממש אופרטור סוגריים עגולים לבדיקת תנאי על זוגות איברים מהאוספים (האופרטור מקבל שני איברים ומחזיר true/false). האלגוריתם מחזיר רשימה (std::list) של איברים מהאוספים שמוגדרת כך:

   עברו על שני האוספים יחד. לכל זוג איברים (אחד מכל אוסף), בדקו את התנאי על הזוג.
   1. במידה והתנאי החזיר TRUE, הכניסו לרשימת התוצאה את האיבר מהאוסף השני, והמשיכו לבדוק את אותו איבר מהאוסף הראשון עם האיבר הבא מהאוסף השני.
   2. במידה והתנאי החזיר False, הכניסו לרשימת התוצאה את האיבר מהאוסף הראשון, והמשיכו לבדוק את אותו איבר מהאוסף השני עם האיבר הבא מהאוסף הראשון.

   שימו לב, שני האוספים לא בהכרח באותו האורך. במידה ועברתם על אוסף בשלמותו ונותרו איברים באוסף השני, הם יועתקו לאוסף התוצאה תוך שמירה על הסדר המקורי.

   ב. הראו והסבירו מתי וכיצד ניתן להשתמש באלגוריתם שכתבתם כדי ליצור משני אוספים רשימה אחת ממוינת.

2. נתון הקוד הבא:

```
#include


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


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

# חלק א' – EscapeRoomWrapper , Enigma

גם בתרגיל הזה נתעסק בחדרי בריחה. בחלק הזה של התרגיל תממשו שתי מחלקות אותן תרחיבו בחלק האחרון של התרגיל:

- מחלקה שתייצג חדר בריחה **EscapeRoomWrapper.**
- מחלקה שתייצג חידה **Enigma**.

## המחלקה EscapeRoomWrapper

בשלב זה של העבודה תיצרו מחלקה עוטפת ל-ADT שנכתב בשפת C. נתונים לכם קובץ h ומימוש של טיפוס EscapeRoom. שימו לב, בכדי שתוכלו לקמפל את הקוד הנתון יחד עם שאר קוד ה++C שלכם, תוכלו למצוא בקובץ ה-h שורה המכילה את הצירוף "extern "C. זו הדרך לשלב בין קוד C ל-++C ואתם מוזמנים לקרוא על כך.

הטיפוס מייצג חדר בריחה בעל התכונות הבאות:

לכל חדר בריחה יש שם name, זמן מקסימלי להיחלצות מהחדר בדקות maxTime (מס' שלם בין 30-90), רמה level (מס' שלם חיובי – קבוע ולא משתנה בטווח 1-10), מספר המשתתפים המקסימלי maxParticipants (מספר שלם וחיובי הגדול ממש מ-1), דירוג ממוצע של החדר avgRate ( חדר מדורג כמספר ממשי בטווח 0-5) ומספר ביקורות votes.

עליכם לממש מחלקה בשם EscapeRoomWrapper המהווה מעטפת ל-ADT הנתון לכם איתה תעבדו בהמשך העבודה. על המחלקה שלכם לשמור אובייקט מטיפוס EscapeRoom ועליה לכלול את הפונקציונליות הבאה:

א. שני בנאים:
- בנאי המקבל את שם החדר (מחרוזת), זמן מקסימלי להיחלצות, רמה, ומספר משתתפים מקסימלי. הבנאי יצור חדר חדש בעל כל הערכים המתקבלים כפרמטרים ומוזכרים לעיל, כאשר הדירוג ההתחלתי של החדר הוא 0, ומספר הביקורות גם הוא 0.
- בנאי המקבל את שם החדר (מחרוזת), ורמה בלבד. הבנאי יצור חדר חדש בעל השם המבוקש, והרמה המבוקשת. זמן ההיחלצות המקסימלי יהיה 60 דקות, ומספר המשתתפים המקסימלי יהיה 6 משתתפים. הדירוג ההתחלתי של החדר יוגדר להיות 0, ומספר הביקורות ההתחלתי גם הוא 0.

ב. תמיכה בהעתקה, הריסה, והשמה של חדרים.

ג. אופרטור שוויון (==), אופרטור שוני (=!), ואופרטורי ההשוואה >, < . ההשוואה בין החדרים תתבצע לפי החישוב הבא:

לכל שני חדרים נגדיר כי הם שווים זה לזה אם ה-power שלהם זהה ושמם זהה. ערך ה-power יחושב על ידי הנוסחא: $power = \frac{level \cdot maxTime}{maxParticipants}$ . במידה והם שונים, נכריע שחדר מסוים קשה יותר מחדר אחר אם ערך ה-power שלו גדול מערכו של החדר השני. במידה וערך ה-power של שני חדרים זהה, נבדוק אם שמם זהה, ואם כן, נאמר שהחדרים זהים ברמת הקושי שלהם.

מעתה והלאה, נגיד שחדר "קשה" יותר מחדר אחר אם הוא גדול ממנו לפי אופרטורי ההשוואה הנ"ל (באופן דומה, נגיד שחדר "קל" יותר או "שווה" לחדר אחר).

שימו לב, כי אין לאפשר שימוש באופרטורי ההשוואה => , =<.

ד. אופרטור פלט (<<), שמדפיס לערוץ את המחרוזת הבאה שמכילה את נתוני החדר (ללא ירידת שורה):

```
<name> (<maxTime>/<level>/<maxParticipants>)
```

ה. פונקציה level, שמחזירה את דרגת החדר.

ו. פונקציה rate, שמקבלת כארגומנט דירוג חדש לחדר ומעדכנת את הדירוג הממוצע של החדר בהתאם, וכך גם את מספר הביקורות.

ז. פונקצית getName המחזירה את שם החדר.

ח. פונקצית getMaxTime המחזירה את הזמן המקסימלי להיחלצות מהחדר.

ט. פונקצית getMaxParticipants המחזירה את מספר המשתתפים המקסימלי בחדר.

י. פונקצית getRate המחזירה את הדירוג הממוצע של החדר.

# המחלקה Enigma

מחלקה זו משמשת כדי לייצג חידה. כידוע לכם, חדר בריחה הוא חדר המורכב ממספר מסוים של חידות המגיעות אחת אחרי השניה, כאשר כל חידה מקדמת את הקבוצה המשחקת עוד צעד בדרך לבריחה המיוחלת. בשלב זה, המחלקה חידה מיוצגת על ידי שם החידה (name), רמת הקושי שלה (difficulty) ומספר הפריטים ממנה היא מורכבת(numOfElements).

בקובץ enigma.h המסופק לכם, מופיע enum המגדיר את רמות הקושי השונות (Difficulty).

```
enum Difficulty {...};
```

על המחלקה לכלול:

א. בנאי היוצר חידה חדשה בהינתן שמה, רמת הקושי שלה ומספר הפריטים מהם היא מורכבת.

ב. תמיכה בהעתקה והשמה של חידות.

ג. אופרטור שוויון (==), אופרטור שוני (=!), ואופרטורי השוואה >, <. ההשוואה בין חידות תתבצע לפי השוואת רמות הקושי של החידות ושמן. כלומר, שתי חידות בעלות אותה רמת קושי ואותו שם, ייחשבו שוות. עבור כל שתי חידות, חידה בעלת רמת קושי גבוהה יותר תיחשב גדולה יותר, וחידה בעלת רמת קושי קטנה יותר, תיחשב קטנה יותר.
שימו לב, עליכם למנוע שימוש באופרטורי ההשוואה =<, =>.

ד. מתודה בשם areEqualyComplex, המקבלת חידה אחרת כארגומנט ובודקת האם החידות בעלות דרגת קושי זהה **וגם** שתיהן מורכבות ממספר זהה של פריטים.

ה. מתודה בשם getDifficulty, המחזירה את רמת הקושי של החידה.

ו. מתודה בשם getName, המחזירה את שמו של החדר.

ז. אופרטור פלט (<<), שמדפיס לערוץ את המחרוזת הבאה שמכילה את נתוני החידה:

```
<name> (<Difficulty>) <number of items>
```

# דרישות

**נתונים לכם:**

- קובץ escapeRoom.h ובו הגדרת ממשק עבור הטיפוס escapeRoom . שימו לב כי זהו טיפוס נתונים שנכתב בשפת C. מצורף לכם גם המימוש בקובץ C מתאים.
- קובץ enigma.h ובו הגדרת המחלקה enigma כפי שמתוארת בחלק זה.
- קובץ escapeRoomWrapper.h ובו הגדרת המחלקה EscapeRoomWrapper כפי שמתוארת בחלק זה.
- קובץ exceptions.h ובו הגדרת חריגות שעליכם לזרוק במקומות המוגדרים בקובץ ה-h או בהוראות.

**נדרש מכם בחלק זה:**

- להוסיף שדות ופונקציות לחלק ה- private ב- escapeRoomWrapper.h ו- enigma.h כרצונכם.
- לממש את הפונקציות של שתי המחלקות escapeRoomWrapper ו- enigma בקבצים חדשים escapeRoomWrapper.cpp ו- enigma.cpp.
- לכתוב בדיקות יחידה לכל אחת משתי המחלקות (ראו "בדיקות יחידה" בהמשך).

# חלק ב' – רשימה מקושרת גנרית

בחלק זה נגדיר מבנה נתונים גנרי מסוג רשימה מקושרת גנרית, אותו תממשו בקובץ list.h ( יש ליצור קובץ זה).
שימו לב, כיוון שהמחלקה צריכה להיות גנרית, המנשק והמימוש צריכים להופיע **שניהם** בקובץ זה.

**מחלקות פנימיות :**

בנוסף למחלקה List<T>, עליכם לכתוב מחלקה של איטרטור לרשימה בשם Iterator אשר תאפשר מעבר על הרשומות במבנה.
איטרטור יכול להצביע לאיבר במבנה או להצביע לסוף המבנה (להבדיל מהאיבר האחרון במבנה, הצבעה לסוף המבנה היא אינדיקציה לכך שהאיטרטור כבר לא מצביע על אף איבר במבנה).

על האיטרטורים לתמוך בפעולות הבאות:

- קידום: ע"י אופרטור ++.
- נסיגה: ע"י אופרטור --.
  שימו לב כי עליכם לממש את שתי הגרסאות של האופרטורים. חישבו מהו ערך ההחזרה המתאים עבור כל גרסא של האופרטור. במידה והאיטרטור מצביע לסוף המבנה, הפעלת האופרטור אינה מוגדרת.
- Dereference: ע"י האופרטור האונארי *.
  החזרת האיבר עליו מצביע האיטרטור. במידה והאיטרטור מצביע לסוף המבנה, יש לזרוק את החריגה ListExceptions::ElementNotFound המופיעה בקובץ Exceptions.h.
- השוואה: אופרטור == ואופרטור =! .
  שני איטרטורים שווים אם הם מצביעים לאותו איבר ברשימה, או אם שניהם מצביעים לסוף הרשימה
- בנוסף, על האיטרטור לתמוך בהעתקות ובהשמות של עצמו.

**פעולות על הרשימה:**

הרשימה צריכה לתמוך בפעולות הבאות:

- Iteraror<T> begin() - חזרת איטרטור לתחילת הרשימה. אם הרשימה ריקה, יוחזר איטרטור לסוף הרשימה.
- Iterator<T> end() - החזרת איטרטור לסוף הרשימה. כאמור, האיטרטור לא מצביע לאף איבר ברשימה.
- Void insert(const T& data, Iterator<T> iterator) – הוספת איבר חדש לרשימה בעל הערך data. אם האיטרטור מצביע לסוף הרשימה, אז האיבר החדש יוכנס כאיבר אחרון לרשימה. בכל מקרה, האיבר החדש יוכנס לפני האיבר אליו מצביע האיטרטור. אם האיטרטור הוא של רשימה אחרת, עליכם לזרוק את החריגה: ListExceptions::ElementNotFound.
- Void insert(const T& data) – הוספת איבר חדש לרשימה בעל הערך data לסוף הרשימה.
- Void remove (Iterarot<T> iterator) – הסרת האיבר אליו מצביע האיטרטור מהרשימה. במידה והרשימה ריקה, או במידה והאיטרטור הוא של רשימה אחרת, יש לזרוק את החריגה ListExceptions::ElementNotFound.
- Iterator<T> find(const Predicate& predicate) – חיפוש איבר ברשימה. Predicate הוא טיפוס של Function Object, כאשר הפונקציה היא פונקציה המקבלת איבר ומחזירה ערך אמת. אם קיים איבר ברשימה המקיים את התנאי predicate, יש להחזיר איטרטור עליו. במידה וכמה איברים מקיימים, יש להחזיר את הראשון המקיים את התנאי. במידה ולא קיים איבר כזה, יש להחזיר איטרטור לסוף הרשימה.
- Void sort(const Compare& compare)– מיון הרשימה. Compare הוא טיפוס של Function Object, כאשר הפונקציה היא פונקציה המקבלת שני איברים, ומחזירה ערך אמת המציין האם האיבר הראשון

קטן מהאיבר השני. על פעולת המיון למיין את הרשימה כך שלכל זוג איברים עוקבים o1,o2 ברשימה הממוינת מתקיים: comparer(o1,o2)=true.

- Int getSize() – החזרת מספר האיברים ברשימה.

**הערות נוספות:**

1. איטרטור לרשימה הופך להיות **לא מוגדר** ברגע שמתבצעת פעולה המשנה את הרשימה. הניחו שניסיון לגשת לרשימה עם איטרטור שאינו מוגדר הוא באג של המשתמש ולכן ההתנהגות של הרשימה במקרה כזה אינה מוגדרת.
2. הימנעו מדרישות מיותרות על הטיפוסים הגנריים, כלומר אל תניחו הנחות כלשהן על אילו פעולות מוגדרות עבור הטיפוסים הללו.
3. שימו לב לניהול נכון של זיכרון של הרשימה. על הרשימה ליצור **עותק** של איברים המוכנסים אליה.

# חלק ג' – הרחבת המחלקות והוספת המחלקה Company

כעת, נרחיב את המחלקות אותן מימשנו בחלק א' על ידי שימוש במבנה vector ובמבנה set (קבוצה) הגנרי הנתונים לנו בספריית ה-STL.

## המחלקה Enigma

כרגע המחלקה Enigma מורכבת משם החידה, רמת הקושי שלה ומספר האביזרים בה.
עליכם להוסיף מבנה נתונים מסוג קבוצה שישמור בתוכו את שמות האביזרים (מחרוזות) המעורבים בחידה.

- Enigma(const std::string& name, const Difficulty& difficulty, const int& numOfElements, const set<string>& elements)
  עליכם לעדכן את הבנאי לקבל גם קבוצה של אביזרים (מחרוזות) לוודא כי יש התאמה בין מספר האביזרים בקבוצה למספר שהתקבל כפרמטר. שימו לב:
  - במידה ואין התאמה בין גודל האוסף לפרמטר המייצג את הגודל יש לזרוק חריגה: EnigmaIllegalSizeParamException.
- Enigma(const std::string& name, const Difficulty& difficulty)
  עליכם להוסיף בנאי שיקבל כפרמטרים את שם החידה ואת רמת הקושי שלה, ויאתחל את מספר האביזרים להיות 0.
- עליכם להוסיף מתודה void addElement(const string& element) שתקבל כפרמטר מחרוזת המייצגת אביזר, ותוסיף אותו לחידה המבוקשת.
- עליכם להוסיף מתודה void removeElement(const string& element) שתקבל כפרמטר מחרוזת המייצגת אביזר ותסיר אותו מהקבוצה (במידה וקיים). שימו לב:
  - במידה ולא נמצא איבר מתאים להסרה, תיזרק החריגה: EnigmaNoElementsException

## המחלקה EscapeRoomWrapper

נרחיב את המחלקה המייצגת את חדר הבריחה. כידוע, כל חדר מכיל מספר חידות הבאות אחת אחרי השניה ופתירתן מובילה לפיצוח החדר.

- עליכם להוסיף למחלקה מבנה נתונים מסוג וקטור שישמור בתוכו אובייקטים מסוג חידה (Enigma).
- עליכם לממש מתודה void addEnigma(const Enigma& enigma) שתקבל כפרמטר חידה חדשה ותוסיף אותה אל החדר. החידה תתווסף לסוף וקטור החידות.
- עליכם לממש מתודה void removeEnigma(const Enigma& enigma) שתקבל כפרמטר חידה ותסיר אותה מוקטור החידות של החדר( אם קיימת).
  - במידה ולא קיימות חידות בחדר, תיזרק החריגה EscapeRoomNoEnigmasException
  - במידה והחידה המבוקשת לא קיימת, תיזרק החריגה EscapeRoomEnigmaNotFoundException.
- עליכם להוסיף מתודה בשם Enigma getHardestEnigma() שמחזירה את החידה הקשה ביותר בחדר. אם קיימות מספר חידות בדרגת קושי מקסימלית זהה, תוחזר הראשונה מבניהן בוקטור.
  - במידה ולא קיימות חידות בחדר, תיזרק החריגה EscapeRoomNoEnigmasException
- עליכם להוסיף מתודה vector<Enigma>& getAllEnigmas() המחזירה את אוסף החידות הקיימות בחדר.

כעת, נוסיף לחדרי הבריחה תכונה נוספת שתבדיל לנו בין סוגים של חדרים. כידוע, ברחבי העולם ניתן למצוא סוגים שונים של חדרי בריחה ונביא זאת לידי ביטוי גם במערכת שלנו!
עליכם לבנות מחלקות חדשות שיירשו מהמחלקה EscapeRoom ויביאו לידי ביטוי סגנונות של חדרי בריחה:

- המחלקה ScaryRoom תמומש בקובצים ScaryRoom.h ו-ScaryRoom.cpp, ותייצג חדר בריחה המתאפיין בקונספט של אימה ופחד. בנוסף לתכונות שלו עקב היותו חדר בריחה, חדרים מסוג זה מתאפיינים גם בהגבלת גיל, כלומר גיל כניסה מינימלי (int), ומספר החידות המפחידות בחדר.
  a. עליכם להוסיף למחלקה בנאי המקבל את כל הפרמטרים הדרושים לבניית חדר בריחה בתוספת פרמטר של הגבלת גיל ומספר החידות המפחידות:
  ScaryRoom(char* name, const int& escapeTime, const int& level, const int& maxParticipants, const int& ageLimit, const int& numOfScaryEnigmas)
  b. עליכם להוסיף מתודה void setNewAgeLimit(const int& limit) המקבלת כפרמטר הגבלת גיל חדשה ומעדכנת את הגבלת הגיל לחדר המבוקש.
  c. עליכם להוסיף מתודה void incNumberOfScaryEnigmas() המגדילה את מונה החידות המפחידות של החדר ב-1.
  d. עליכם להוסיף מתודה int getAgeLimit() המחזירה את הגבלת הגיל של החדר.
  e. עליכם לממש אופרטור פלט (<<), שמדפיס לערוץ את המחרוזת הבאה שמכילה את נתוני החדר (ללא ירידת שורה):

```
Scary Room: <name> (<maxTime>/<level>/<maxParticipants>/<ageLimit>)
```

- המחלקה kidsRoom תמומש בקבצים KidsRoom.h ו-KidsRoom.cpp , ותייצג חדר בריחה המתאים לילדים בלבד. חדרים מסוג זה מתאפיינים בנוסף לתכונותיהם כחדרי בריחה גם מאפיין של הגבלת גיל, כלומר, גיל כניסה מקסימלי (int).
  a. עליכם להוסיף למחלקה בנאי המקבל את כל הפרמטרים הדרושים לבניית חדר בריחה בתוספת פרמטר של הגבלת גיל:
  KidsRoom(char* name, const int& escapeTime, const int& level, const int& maxParticipants, const int& ageLimit)

b. עליכם לממש מתודה void setNewAgeLimit(const int& limit) המקבלת כפרמטר הגבלת גיל חדשה ומעדכנת את הגבלת הגיל לחדר המבוקש.
c. עליכם להוסיף מתודה int getAgeLimit() המחזירה את הגבלת הגיל של החדר.
d. עליכם לממש אופרטור פלט (<<), שמדפיס לערוץ את המחרוזת הבאה שמכילה את נתוני החדר (ללא ירידת שורה):

```
Kids Room: <name> (<maxTime>/<level>/<maxParticipants>/<ageLimit>)
```

## המחלקה Company

כעת, נממש מחלקה חדשה בשם Company המייצג חברה של חדרי בריחה.
חברה מאופיינת על ידי שמה (מחרוזת) ועל ידי מספר הטלפון שלה (מחרוזת).

- כל חברה מכילה אוסף השומר את חדרי הבריחה השייכים לה.
- למחלקה קיים בנאי המקבל כפרמטרים את שם החברה ומספר הטלפון שלה ומאתחל את הפרטים הדרושים.
- ניתן להוסיף חדר בסיסי חדש לחברה על ידי המתודה createRoom המקבלת את כל הפרמטרים הדרושים ליצירת חדר. המתודה תיצור את החדר ותוסיף אותו לאוסף החדרים של החברה.
  - במידה ויצירת החדר נכשלה תיזרק החריגה : CompanyMemoryProblemException
- ניתן להוסיף חדר מפחיד חדש לחברה על ידי המתודה createScaryRoom על ידי קבלת כל הנתונים הדרושים ליצירת חדר מפחיד, והמתודה תיצור את החדר ותוסיף אותו למבנה הנתונים אותו היא מנהלת.
  - במידה ויצירת החדר נכשלה תיזרק החריגה : CompanyMemoryProblemException
- ניתן להוסיף חדר ילדים חדש לחברה על ידי המתודה createKidsRoom על ידי קבלת כל הנתונים הדרושים ליצירת חדר ילדים, והמתודה תיצור את החדר ותוסיף אותו למבנה הנתונים אותו היא מנהלת.
  - במידה ויצירת החדר נכשלה תיזרק החריגה : CompanyMemoryProblemException
- קיימת מתודה בשם getAllRooms המחזירה את אוסף החדרים בבעלות החברה.
- החברה יכולה להרוס חדרים שבבעלותה, כלומר להסיר חדר מקבוצת החדרים שבבעלותה על ידי המתודה removeRoom.
  - במידה והחדר לא קיים, תיזרק החרגיה CompanyRoomNotFoundException
- החברה יכולה להוסיף חידות חדר מסוים על ידי המתודה addEnigma המקבלת כפרמטר את החדר אליו היא רוצה להוסיף חידה ואת החידה המבוקשת.
  - במידה והחדר לא קיים, תיזרק החריגה CompanyRoomNotFoundException
- החברה יכולה גם להסיר חידות מסוימות מחדר מסוים על ידי המתודה removeEnigma המקבלת כפרמטר את החדר ממנו היא רוצה להסיר את החידה ואת החידה המבוקשת.
  - במידה והחדר לא קיים, תיזרק החריגה CompanyRoomNotFoundException
  - במידה ואין חידות בחדר, תיזרק החריגה CompanyRoomHasNoEnigmasException
  - במידה והחידה לא קיימת בחדר, תיזרק החריגה CompanyRoomEnigmaNotFoundException
- החברה יכולה להוסיף אביזר מסוים לחידה על ידי המתודה addItem המקבלת כפרמטר את החדר בו נמצאת החידה אליה נרצה להוסיף אביזר, את החידה ואת האביזר המבוקש.
  - במידה והחדר לא קיים, תיזרק החריגה CompanyRoomNotFoundException
  - במידה והחידה לא קיימת בחדר, תיזרק החריגה CompanyRoomEnigmaNotFoundException
- החברה יכולה להסיר אביזר מחידה מסוימת על ידי המתודה removeItem המקבלת כפרמטר את החדר בו נמצאת החידה ממנה נרצה להסיר אביזר, את החידה ואת האביזר המבוקש.
  - במידה והחדר לא קיים, תיזרק החריגה CompanyRoomNotFoundException

- במידה ואין אביזרים לחידה בחדר, תיזרק החריגה CompanyEnigmaHasNoElementsException
- במידה והאביזר לא קיים בחידה בחדר, תיזרק החריגה CompanyRoomEnigmaElementNotFoundException

- קיימת מתודה בשם getAllRoomsByType המקבלת כפרמטר סוג חדר ומחזירה אוסף המכיל את כל החדרים מהטיפוס המבוקש (אוסף ריק אם אין חדרים כאלו).
- קיימת מתודה בשם getRoomByName המקבלת כפרמטר שם של חדר ומחזירה עותק של החדר בעל השם הנ"ל. (הניחו כי יש אחד כזה).
  - במידה ולא קיים חדר בעל השם המבוקש, תיזרק החריגה CompanyRoomNotFoundException
- עליכם לממש אופרטור פלט (<<), שמדפיס לערוץ את המחרוזת הבאה שמכילה את נתוני החדר (ללא ירידת שורה):

```
<CompanyName> : <PhoneNumber>\n
-<RoomName>
-<RoomName> ….
```

## דרישות

**נתונים לכם:**

- קובץ company.h ובו הגדרת המחלקה Company.
- קובץ exceptions.h ובו חריגות שהפונקציות שהגדרנו עלולות לזרוק.

**נדרש מכם בחלק זה:**

- להשלים את הגדרת המחלקה Company ב- company.h, ולממשה בקובץ חדש שייקרא company.cpp. (הכוונה בהשלמה הגדרה של מחלקה היא להוסיף מתודות ושדות כרצונכם בכל רמות הגישה (private/protected/public).
- להרחיב את המחלקות Enigma , EscapeRoomWrapper (הגדרה ומימוש) כרצונכם.
- להגדיר ולממש כל מחלקה אחרת שאתם צריכים.
- לכתוב בדיקות יחידה לכל אחת מהמחלקות שאתם כותבים (בפרט, להרחיב את בדיקת היחידה שכתבתם בחלק א').

# הנחיות

## חריגות

- כל אחת מהפונקציות המוגדרות בקבצים שאנחנו סיפקנו עלולה לזרוק חריגות. כל החריגות מוגדרות בקובץ exceptions.h, ומעל להגדרת כל פונקציה, מצוין אילו חריגות היא עלולה לזרוק.
- יש להתעלם מחריגות מסוג std::bad_alloc.

## הקפדה על טיב הקוד

- יש לשמור על הסתרה ומיעוט שכפולי קוד כנלמד בקורס.
- יש להקפיד על const correctness.
- יש להשתמש בהעמסת אופרטורים כאשר הדבר מתאים.
- המימוש חייב לציית לכללי כתיבת הקוד המופיעים באתר תחת Course Material, Code Conventions. אי עמידה בכללים אלו תגרור הורדת נקודות.

## בדיקות יחידה

עליכם לכתוב בדיקות יחידה עבור כל אחת מהמחלקות אותן תכתבו בחלק הרטוב. את בדיקות היחידה יש לשים בתוך תיקייה בשם test. יש להגיש קבצים אלו ללא פונקציית main.

מספר דגשים:

- הקפידו לבדוק גם את התנהגות הקוד במצבי שגיאות, ולא רק עבור פרמטרים וקלט מוצלח.
- הקפידו על איכות הקוד גם בבדיקות – הבדיקות צריכות להיות קצרות, מחולקות לפונקציות וקריאות, כך שבמקרה כישלון יהיה קל למצוא את הסיבה לבאג.
- **מומלץ לחלק את הבדיקות כך שלכל פונקציה בכל מחלקה קיימת פונקציה הבודקת אותה.**
- השתמשו בקובץ mtm_test.h כדי לבדוק את הקוד שלכם.

המלצות:

- כתיבת בדיקות לוקחת זמן אך מקלה משמעותית על תחזוקת קוד בהמשך ועל צמצום הזמן הדרוש לדיבוג התכנית. מומלץ לכתוב את הבדיקות בד בבד עם הקוד. למשל לפני שאתם כותבים פונקציה מסוימת, כתבו את הפונקציה הבודקת אותה. את כתיבת הבדיקות כדאי לעשות כאשר חושבים כיצד ניתן "להכשיל" את הקוד ובכך לוודא את איכות הקוד במקרה קצה. כתיבת כל הבדיקות לאחר סיום התרגיל תהיה מעיקה משמעותית ותתבצע אחרי כל השלבים שבהן הבדיקה יכלה להועיל בפועל.
- אל תתחילו לדבג לפני שיש לכם בדיקות יחידה.

## הגשה

יש להגיש קובץ zip בעל הפורמט הבא:

- תיקיית tests שבה נמצאים הטסטים אשר כתבתם.
- כל קבצי הקוד עבור התרגיל (h ו- cpp) בתיקייה הראשית של ה- zip (כלומר, לא בתוך אף תיקייה).

**כמו כן אין להגיש כל קובץ בו מופיעה הפונקציה main.**

## הידור, קישור, ובדיקה

המערכת צריכה לעבוד על Stud/CSL2. על מנת לעבוד עם C++11 יש להתקין גרסה חדשה יותר של G++. הוראות מפורטות על הורדת הקומפיילר ויצירת פרוייקט עבור C++11 באקליפס מופיעות ב-FAQ.

השתמשו בפקודת הקומפילציה הבאה:

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

התרגיל ייבדק בדיקה יבשה ובדיקה רטובה.
הבדיקה היבשה כוללת מעבר על הקוד ובודקת את איכות הקוד ועמידה בדרישות במימוש.
הבדיקה הרטובה תהיה אוטומטית ותכלול בדיקות אוטומטיות לוידוא נכונות הקוד.

**עליכם לבדוק שהקוד שלכם פועל כצפוי! בדיקת הקוד מהווה חלק מהתרגיל! תרגיל אשר אינו עובר בדיקות יקבל 0 בבדיקה הרטובה, לא יהיו הנחות בנושא זה.**

- וודאו את נכונות התכנית שלכם **במקרים כלליים ומקרי קצה**.
- מומלץ לחשוב על הבדיקות **כבר בזמן כתיבת הקוד**.
- נסו ליצור **מס' רב של בדיקות קצרות** ולא בדיקה אחת גדולה שכישלון בה אינו מסגיר את התקלה.
- ודאו את נכונות הקוד לפני ההגשה ו**גם לאחר שינויים קטנים** ולכאורה לא משמעותיים!

- **שימו לב לדליפות זיכרון** – הן ייבדקו בצורה אוטומטית! אפשר להשתמש ב-valgrind כדי לוודא שאין לכם דליפות זיכרון. Valgrind הוא כלי המותקן ב-CSL2 ומיועד, בין היתר, למציאת דליפות זיכרון.

**שימו לב: כל קבצי הקוד (למעט הטסטים) צריכים להימצא בתיקיית הבסיס של קובץ ה-zip המוגש!**