מבוא לתכנות מערכות – תרגיל בית 2 – חורף 2021-2022

תאריך פרסום: - 21/12/2021
תאריך הגשה: - 14/01/2022
משקל התרגיל: 12% מהציון הסופי (תקף)
מתרגל אחראי: מוחמד גומייד

1 הערות כלליות

- ההגשה היא בזוגות בלבד, באתר הקורס ב-Webcourse.
- שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות של התרגיל, או לשאול בפורום של הקורס בפיאצה. לפני שליחת שאלה – נא וודאו שהיא לא נענתה כבר ב-FAQ או בפורום, ושהתשובה אינה ברורה ממסמך זה, מהדוגמאות 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 שאתה ניגשים לעבוד.
- חובה להתעדכן בעמוד ה-FAQ של התרגיל.
- העתקות בתוכנה תטופלנה בחומרה.
- התרגיל מורכב מחלק רטוב וחלק יבש.

2 חלק רטוב

2.1 רקע

בתרגיל בית הזה אנחנו ננסה לדמות את שוק העבודה בעיר מסוימת. העיר מכילה אזרחים שמחולקים לשני סוגים: עובדים רגילים, שכל הזמן הם מנסים לרכוש מיומנות כדי שיכלו לעבוד במקומות עבודה בהם הם רוצים לעבוד. סוג אחר של אזרחים הוא מנהלים, הם עובדים במקומות עבודה שונים, ומנהלים את העובדים הרגילים.

2.2 מחלקת Skill

מחלקה שמייצגת מְיוּמָנוּת. לכל מְיוּמָנוּת יש מספר מזהה, שם, כמה נקודות צריך כדי לרכוש אותה (מספר שלם), בנוסף תספק הממשק הבא:

1. **בנאי:** בנאי מקבל פרמטרים הבאים: מספר מזהה למיומנות, שם של המיומנות, כמה נקודות צריך כדי לרכוש אותה, ומאתחלת את האובייקט.
2. **מתודות getId, getName, getRequiredPoints:** לא מקבלות פרמטרים, ומחזירה את המספר המזהה, השם של המְיוּמָנוּת, כמה נקודות צריך כדי לרכוש אותה של this בהתאמה.
3. הדפסת מיומנות בעזרת **אופרטור הפלט**. **פורמט ההדפסה המדויק יתואר בהמשך.**
4. השוואת מיומנויות בעזרת **האופרטורים >, <, ==, =<, => ו-=!**. בהשוואה בין שני מיומנויות, ההשוואה בין המיומנויות היא לפי מספר מזהה.
5. קידום מספר נקודות צריך כדי לרכוש אותה מְיוּמָנוּת בעזרת **האופרטור ++.** יש לתמוך ב-++ מצד ימין בלבד , ולא מצד שמאל.
6. קידום מספר נקודות צריך כדי לרכוש אותה מְיוּמָנוּת בעזרת **האופרטור =+.** אם התקבל מספר שלילי בתור פרמטר, יש לזרוק חריגת NegativePoints.
7. חיבור בין מיומנות ומספר אי-שלילי points בעזרת **האופרטור +**. הפעולה תבצע קידום מספר נקודות צריך כדי לרכוש אותה מְיוּמָנוּת שהיא קבלה. אם התקבל מספר שלילי בתור פרמטר, יש לזרוק חריגת NegativePoints.

הערות:

1. אם עוד לא למדתם על חריגות בשלב זה, מומלץ לכתוב את הקוד כאילו אין שגיאות, ולהוסיף טיפול בשגיאות בעזרת חריגות יותר מאוחר. הפתרון שתגישו בסוף חייב לטפל בשגיאות.
2. בחלק זה אסור להשתמש ב-STL.
3. עליכם להשתמש ב std::string לטיפול במחרוזות ולא ב*char.

פורמט ההדפסה המדויק של Skill:

```
#include <iostream>
#include "Skill.h"

using std::cout;
using namespace mtm;
int main() {
    Skill s1(1, "C++", 0);
    cout << s1;
    return 0;
}
```

עבור התכנית הזאת:

```
C++
```

יתקבל הפלט הבא:

2.3 מחלקת אזרח וסוגי עובדים

2.3.1 מחלקת Citizen

מחלקה **אבסטרקטית** שמייצגת אזרח בעיר, לכל אזרח יש מספר תעודת זהות, שם פרטי ושם משפחה, שנת לידה, ותספק את הממשק הבא:

1. **בנאי**: בנאי מקבל הפרמטרים הבאים: מספר תעודת זהות, שם פרטי ושם משפחה, שנת לידה.
2. **מתודות getId, getFirstName, getLastName, getBirthYear**: לא מקבלות פרמטרים, ומחזירות השדה המבוקש של this.
3. השוואת אזרחים בעזרת **האופרטורים >, <, ==, =<, => ו-=!**. ההשוואה בין האזרחים היא לפי מספר מזהה.
4. **מתודת printShort**: מקבלת std::ostream, מדפיסה אליו תיאור קצר של האזרח, ומחזירה את ה-std::ostream שהיא קיבלה. פונקציה זו היא פונקציה וירטואלית שתדרסו בהמשך.
5. **מתודת printLong**: מקבלת std::ostream, מדפיסה אליו תיאור מפורט של האזרח, ומחזירה את ה-std::ostream שהיא קיבלה. פונקציה זו היא פונקציה וירטואלית שתדרסו בהמשך.
6. **מתודת clone**: לא מקבלת פרמטרים. מייצרת עותק חדש של this ומחזירה מצביע לעותק החדש שנוצר.

2.3.2 מחלקת Employee

מחלקת שמייצגת עובד רגיל. כל עובד הוא בפרט אזרח, יש לו שכר שהוא מקבל ממקום העבודה, ניקוד, סט מיומנויות. בנוסף, המחלקה צריכה לספק את הממשק הבא:

1. **בנאי**: הבנאי יקבל את כל המידע שמקבל הבנאי של המחלקה Citizen ,ויאתחל עובד חדש עם שכר שווה לאפס, ניקוד ששווה לאפס, וקבוצת מיומנויות ריקה.
2. **מתודות getSalary, getScore**: לא מקבלות פרמטרים, ומחזירות השדה המבוקש של this.
3. **מתודת learnSkill**: מקבלת מְיוּמָנוּת ומנסה לרשום אותה לסט המיומנויות של העובד. אם המְיוּמָנוּת כבר רשומה, יש לזרוק חריגת SkillAlreadyLearned. אם העובד לא יכול לרכוש את המְיוּמָנוּת, יש לזרוק חריגת CanNotLearnSkill.
4. **מתודת forgetSkill**: מקבלת מספר מזהה למְיוּמָנוּת ומסירה אותה מסט המיומנויות של העובד. אם המְיוּמָנוּת לא רשומה, יש לזרוק חריגת DidNotLearnSkill.
5. **מתודה hasSkill**: מקבלת מספר מזהה למְיוּמָנוּת ומחזריה True אם יש לעובד המיומנות, אחרת False.
6. **מתודה setSalary**: מתודה זאת מקבלת מספר שלם ומוסיפה אותו לשכר של עובד.
7. **מתודה setScore**: מתודה זאת מקבלת מספר שלם ומוסיפה אותו לניקוד של עובד.
8. **מתודת printShort**: מקבלת std::ostream, מדפיסה אליו תיאור קצר של העובד, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
9. **מתודת printLong**: מקבלת std::ostream, מדפיסה אליו תיאור מפורט של העובד סט מיומנויות שלו, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
10. **מתודת clone**: לא מקבלת פרמטרים. מייצרת עותק חדש של this ומחזירה מצביע לעותק החדש שנוצר. המתודה צריכה להעתיק גם את סט מיומנויות.

פורמט ההדפסה המדויק של Employee:

עבור התכנית הזאת:

```
#include <iostream>
#include "Employee.h"

using namespace mtm;
using std::cout;
using std::endl;

int main() {
    Employee e1(1, "John", "Williams", 2002);
    Skill s1(1,"C++",0);
    Skill s2(2, "Java", 0);
    e1.learnSkill(s1);
    e1.learnSkill(s2);
    cout << "Short_Print" << endl;
    e1.printShort(cout);
    cout << "Long Print" << endl;
    e1.printLong(cout);
    return 0;
}
```

```
Short_Print
John Williams
Salary: 0 Score: 0
Long Print
John Williams
id - 1 birth_year - 2002
Salary: 0 Score: 0 Skills:
C++
Java
```

יתקבל הפלט הבא:

2.3.3 מחלקת Manager

מחלקת שמייצגת מנהל. כל מנהל הוא בפרט אזרח, יש לו שכר שהוא מקבל ממקום העבודה, וקבוצת העובדים שהוא מנהל. בנוסף, המחלקה צריכה לספק את הממשק הבא:

1. **בנאי**: הבנאי יקבל את כל המידע שמקבל הבנאי של המחלקה Citizen, ויאתחל מנהל חדש עם שכר שווה לאפס, וקבוצת העובדים ריקה.
2. **מתודה getSalary**: לא מקבלות פרמטרים, ומחזירות השדה המבוקש של this.
3. **מתודת addEmployee**: מקבלת מצביע לעובד ומנסה לרשום אותה לקובצת העובדים של המנהל. אם העובד כבר בקבוצה, יש לזרוק חריגת EmployeeAlreadyHired.
4. **מתודת removeEmployee**: מקבלת מספר מזהה של עובד ומסירה אותו מקובצת העובדים של המנהל. אם העובד אינו בקבוצה, יש לזרוק חריגת EmployeeIsNotHired.
5. **מתודה setSalary**: מתודה זאת מקבלת מספר שלם ומוסיפה אותו לשכר של המנהל.
6. **מתודת printShort**: מקבלת std::ostream, מדפיסה אליו תיאור קצר של המנהל, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
7. **מתודת printLong**: מקבלת std::ostream, מדפיסה אליו תיאור מפורט של המנהל, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
8. **מתודת clone**: לא מקבלת פרמטרים. מייצרת עותק חדש של this ומחזירה מצביע לעותק החדש שנוצר.

פורמט ההדפסה המדויק של Manager:

עבור התכנית הזאת:

```
#include <iostream>
#include "Employee.h"
#include "Manager.h"

using namespace mtm;
using std::cout;
using std::endl;

int main() {
    Employee e1(1, "John", "Williams", 2002);
    Employee e2(2, "Alex", "Martinez", 2000);
    Manager m1(1,"Robert", "stark", 1980);
    m1.addEmployee(&e1);
    m1.addEmployee(&e2);
    cout << "Short_Print" << endl;
    m1.printShort(cout);
    cout << "Long Print" << endl;
    m1.printLong(cout);
    return 0;
}
```

יתקבל הפלט הבא:

```
Short_Print
Robert stark
Salary: 0
Long Print
Robert stark
id - 1 birth_year - 1980
Salary: 0
Employees:
John Williams
Salary: 0 Score: 0
Alex Martinez
Salary: 0 Score: 0
```

2.3.4 מחלקת Workplace

מחלקה שמייצגת מקום עבודה שמכיל מנהלים (לכל מנהל יש עובדים שלו). המחלקה צריכה לספק את הממשק הבא:

1. **בנאי**: מקבל את מספר מזהה ,שם מקום העבודה, השכר שמקבלים כל העובדים בחברה, והשכר שמקבלים כל המנהלים בחברה, ומאתחל את מקום העבודה עם קבוצת מנהלים ריקה.
2. **מתודות getId, getName, getWorkersSalary, getManagersSalary**: לא מקבלות פרמטרים, ומחזירות השדה המבוקש של this.
3. **מתודת hireEmployee**: היא פונקציה גנרית שמייצגת תנאי לקבלת העובד (שהוא Function object) ,מצביע לעובד, והמספר המזהה של המנהל. אם העובד שהיא מקבלת מקיים את התנאי, והמנהל נמצא במקום העבודה, המתודה מוסיפה אותו לחברה תחת בקבוצה של המנהל שהיא מקבלת. המתודה זורקת:
   a. EmployeeNotSelected במקרה שהעובד לא מקיים התנאי.
   b. ManagerIsNotHired במקרה שהמנהל אינו קיים במקום העבודה.
   c. EmployeeAlreadyHired במקרה שכבר עובד העובד הזה בקבוצה של המנהל שקבלה.
4. מתודה **hireManager:** מקבלת מצביע למנהל ומוסיפה מנהל למקום העבודה. אם המנהל כבר קיים לזרוק ManagerAlreadyHired, אם המנהל עובד במקום עבודה אחר אז הפונקציה זורקת CanNotHireManager.
5. **מתודת fireEmployee**: מקבלת מספר מזהה לעובד, והמספר המזהה של המנהל. המתודה מסירה אותו מהקבוצה של המנהל שהיא מקבלת. המתודה זורקת חריגת EmployeeIsNotHired במקרה שהעובד לא קיים בקבוצה של המנהל, ManagerIsNotHired במקרה שהמנהל אינו קיים במקום העבודה.
6. מתודה **fireManager:** מקבלת מספר מזהה למנהל ומסירה אותו והקבוצה שלו ממקום העבודה. אם המנהל אינו קיים במקום העבודה יש לזרוק ManagerIsNotHired.
7. הדפסת מקום עבודה בעזרת **אופרטור הפלט**. **פורמט ההדפסה המדויק יתואר בהמשך.**

ghפורמט ההדפסה המדויק של Workplace:

עבור התכנית הזאת:

```
#include <iostream>
#include "Workplace.h"

using namespace mtm;
using std::cout;
using std::endl;

class Condition{
public:
    bool operator()(Employee* emp){
        return emp->getId()>0;
    }
};

int main() {
    Workplace Meta(1,"Meta", 10000, 20000);
    Employee* e1 = new Employee(1, "John", "Williams", 2002);
    Employee* e2 = new Employee(2, "Alex", "Martinez", 2000);
    Manager* m1 = new Manager(1,"Robert", "stark", 1980);
    Meta.hireManager(m1);
    Condition condition;
    Meta.hireEmployee(condition,e1,m1->getId());
    Meta.hireEmployee(condition,e2,m1->getId());
    cout << Meta;
    Meta.fireManager(m1->getId());
    cout << Meta;
    return 0;
}
```

יתקבל הפלט הבא:

```
Workplace name - Meta Groups:
Manager Robert stark
id - 1 birth_year - 1980
Salary: 20000
Employees:
John Williams
Salary: 10000 Score: 0
Alex Martinez
Salary: 10000 Score: 0
Workplace name - Meta
```

2.3.5 מחלקת Faculty

מחלקה גנרית המייצגת פקולטה שבה עובדים לומדים המיומנויות אם הם מקיימים תנאי הקבלה. על מחלקה לספק את הממשק הבא:

1. **בנאי**: בנאי המקבל מספר מזהה לפקולטה ,תנאי קבלה לפקולטה (שהוא Function Object שמקבל עובד ומחזיר True במקרה שהוא מקיים התנאי),המיומנות שהיא מלמדת, וכמה נקודות הלימודים בפקולטה מוסיף לניקוד של העובד. ומאתחל אובייקט מסוג פקולטה.
2. **מתודות getSkill, getId, getAddedPoints**: לא מקבלות פרמטרים, ומחזירות השדה המבוקש של this.
3. **מתודת teach**: מקבלת עובד ובודקת אם הוא מקיים את תנאי הקבלה של הפקולטה, אם כן, מלמדת אותו את המיומנות ומגדילה את הניקוד שלו. המתודה זורקת חריגת EmployeeNotAccepted במקרה שאינו מקיים התנאי.

2.4 חלק ד: מחלקה City

המחלקה הזאת תכיל את כל האזרחים, פקולטות ומקומות עבודה שיש בתוך העיר, המחלקה הזאת תספק הממשק הבא:

1. **בנאי:** הבנאי מקבל את השם של העיר ומאתחלת את העיר להיות ריק.
2. **מתודת addEmployee**: מקבלת את כל הפרטים שדרושים כדי לאתחל עובד ומוסיפה אותו לעיר. אם כבר נמצא בעיר, אז יש לזרוק חריגת CitizenAlreadyExists.
3. **מתודת addManager**: מקבלת את כל הפרטים שדרושים כדי לאתחל מנהל ומוסיפה אותו לעיר. אם כבר נמצא בעיר, אז יש לזרוק חריגת CitizenAlreadyExists.
4. **מתודת addFaculty**: מקבלת את כל הפרטים שדרושים כדי לאתחל פקולטה ומוסיפה אותו לעיר. אם כבר נמצא בעיר, אז יש לזרוק חריגת FacultyAlreadyExists.
5. **מתודה createWorkplace** מקבלת את כל הפרטים שדרושים כדי לאתחל מקום עבודה ומוסיפה אותו לעיר. אם יש עוד מקום עבודה עם אותו מספר מזהה אז יש לזרוק WorkplaceAlreadyExists.
6. **מתודת teachAtFaculty**: מקבלת מספר מזהה של עובד ומספר מזהה של פקולטה, ומנסה ללמד העובד את המיומנות שמלמדת הפקולטה. אם העובד אני קיים, המתודה זורקת EmployeeDoesNotExist, אם הפקולטה אינה קיימת, המתודה זורקת FacultyDoesNotExist.
7. **מתודת hireEmployeeAtWorkplace**: היא פונקציה גנרית שמייצגת תנאי לקבלת העובד (שהוא Function object) ,מספר מזהה של עובד ,מספר מזהה של מנהל והמספר המזהה של מקום העובדה. ומנסה להעביד העובד במקום העבודה בקבוצה של המנהל. אם העובד אינו קיים בעיר היא זורקת EmployeeDoesNotExist, אם המנהל אינו קיים בעיר זורקת ManagerDoesNotExist, אם מקום העבודה אינו קיים זורקת WorkplaceDoesNotExist.
8. מתודה **hireManagerAtWorkplace:** מקבלת מספר מזהה של מנהל והמספר המזהה של מקום העובדה. ומנסה להעביד המנהל במקום העבודה. אם המנהל אינו קיים בעיר זורקת ManagerDoesNotExist, אם מקום העבודה אינו קיים זורקת WorkplaceDoesNotExist.
9. **מתודת fireEmployeeAtWorkplace**: מקבלת מספר מזהה לעובד, המספר המזהה של המנהל והמספר המזהה של מקום העובדה. המתודה מסירה אותו מהקבוצה של המנהל שהיא מקבלת. אם העובד אינו קיים בעיר היא זורקת EmployeeDoesNotExist, אם המנהל אינו קיים בעיר זורקת ManagerDoesNotExist, אם מקום העבודה אינו קיים זורקת WorkplaceDoesNotExist.
10. מתודה **fireManagerAtWorkplace:** מקבלת מספר מזהה למנהל והמספר המזהה של מקום העובדה, ומסירה אותו ממקום העבודה. אם המנהל אינו קיים בעיר זורקת ManagerDoesNotExist, אם מקום העבודה אינו קיים זורקת WorkplaceDoesNotExist.
11. **מתודה getAllAboveSalary:** מקבלת std::ostream, ושכר מסיום. מדפיסה תיאור קצר לכל האזרחים שיש להם שכר גדול שווה מהשכר שקבלה (מסודרים בסדר עולה לפי המספר המזהה), ומחזירה את מספרם.

12. **מתודה isWorkingInTheSameWorkplace:** מקבלת שני מספרים מזהים לעובדים, וחזירה true אם הם עובדים באותו מקום עבודה, אחרת מחזירה false. אם לפחות אחד העובדים אינו קיים בעיר היא זורקת EmployeeDoesNotExist.
13. **מתודה printAllEmployeesWithSkill:** מקבלת std::ostream, מדפיסה אליו תיאור קצר של כל העובדים שיש להם מיומנות שהיא מקבלת.

דוגמא לפלט של הפונקציות נמצא בטסט שקבלתם.

הדרכה לשימוש ב Function Object במחלקה:

הפקולטה צריכה לשמור את תנאי הקבלה של העובד. מכיוון שלכל פקולטה יש תנאי, ואנחנו צריכים לשמור את הפקולטות בתוך container מסוים בתוך המחלקה City, אנחנו צריכים להשתמש בפולימורפיזם כדי לפתור הבעיה. יש כמה דרכים לפתור את זה, וכאן נציג אחת הדרכים האפשריים:

בקובץ הH של המחלקה Faculty, נוסיף את המחלקה הזאת:

```
class Condition{
public:
    virtual bool operator()(Employee* employee) = 0;
};
```

המחלקה הזאת היא מחלקה אבסטרקטית שכל המחלקות האחרות שמייצגות תנאי קבלה ירשו ממנה ויצרכו לממש את אופרטור ההפעלה.

נשמור ה Function Object כמצביע למחלקת האב (Condition) בתוך המחלקה Faculty, ונשתמש בו כדי להחליט האם לקבל את העובד במתודה teach.

דוגמא לשימוש במחלקה Faculty:

```
#include <iostream>
#include "Faculty.h"
using namespace mtm;
using std::cout;
using std::endl;

class FacultyCondition1: public Condition{
    bool operator()(Employee* employee) override{
        cout << "Condition 1 Called" << endl;
        return employee->getId() > 0;
    }
};

class FacultyCondition2: public Condition{
    bool operator()(Employee* employee) override{
        cout << "Condition 2 Called" << endl;
        return employee->getId() > 3;
    }
};

int main() {
    vector<Faculty<Condition>> Faculties;
    FacultyCondition1 fc1;
    FacultyCondition2 fc2;
    Skill skill1(1,"Programming with c++",0);
    Skill skill2(2,"Programming with c",10);
    Faculty<Condition> faculty1(1,skill1,10,&fc1);
    Faculty<Condition> faculty2(3,skill2,10,&fc2);
    Employee e1(10, "John", "Williams", 2002);
    Employee e2(20, "Alex", "Martinez", 2000);
    Faculties.push_back(faculty1);
    Faculties.push_back(faculty2);
    for(Faculty<Condition> faculty:Faculties){
        faculty.teach(&e1);
        faculty.teach(&e2);
    }
    return 0;
}
```

אם נריץ, יתקבל הפלט הבא:

```
Condition 1 Called
Condition 1 Called
Condition 2 Called
Condition 2 Called
```

ואז אנחנו יכולים לראות שנקראים התנאים המתאימים לפי הפקולטה.

3 דרישות והערות נוספות

- כל הקוד בפתרון שלכם חייב להיות תחת namespace בשם mtm.
- יש לשמור על Code Conventions בתרגיל.
- כל החריגות שהוגדרו במסמך חייבות לרשת ממחלקת mtm::Exception, ומחלקת mtm::Excpetion בעצמה חייבת לרשת מ-std::excpetion. את כל החריגות יש לממש בקובץ בשם exceptions.h.
- בחלק 2.2 אסור להשתמש ב-STL. בשאר החלקים מותר להשתמש ב-STL.
- מותר להוסיף מתודות חדשות למחלקות המתוארות במסמך, ומותר ליצור פונקציות ומחלקות חדשות שלא מתוארות במסמך.
- על מנת שהטסטים יצליחו לבצע #include למחלקות הנדרשות בתרגיל, ההגדרות של המחלקות צריכות להיות זמינות בקובץ אחד בשם solution:
  - מחלקת Skill: בקובץ Skill.h
  - כל החריגות: בקובץ exceptions.h
  - מחלקת Citizen: בקובץ Citizen.h
  - מחלקת Employee: בקובץ Employee.h
  - מחלקת Manager: בקובץ Manager.h
  - מחלקת Workplace: בקובץ Workplace.h
  - מחלקת Faculty: בקובץ Faculty.h
  - מחלקת City: בקובץ City.h

4 הידור, קישור ובדיקה

התיקייה ~mtm/public/2122a/ex2/provided מכילה קבצי עזר לבדיקת התרגיל, ומומלץ להעתיק אותו לתיקיית התרגיל הפרטית שלכם. התרגיל ייבדק על השרת csl3, ועליו לעבור הידור בעזרת הפקודה הבאה.

עבור חלק א:

```
g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -o prog -
Isolution -Iprovided provided/city_test.cpp solution/*.cpp
```

פירוט תפקידי חלק מהדגלים בפקודות ההידור:

- **-std=c++11** שימוש בתקן C++11 של שפת C++.
- **-o prog** קובץ הפלט ייקרא prog.
- **-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 ושגיאות.
- **-DNDEBUG** מוסיף את השורה #define NDEBUG בתחילת כל יחידת קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו assert לא יופעל בריצת התוכנית.
- **-Isolution** פקודות `#include "foo.h"` יחפשו את הקובץ foo.h גם בתיקייה solution.

בהמשך תפורסם גם תוכנית finalCheck באמצעותה תוכלו לבדוק את ההגשה שלכם. השתמשו ב-finalCheck כדי לוודא שההגשה שלכם עוברת הידור בהצלחה. אם ההגשה שלכם לא עוברת הידור ב-finalCheck, אז בסבירות גבוהה היא לא תעבור הידור גם בעת בדיקת התרגיל.

התרגיל ייבדק בעזרת טסטים אוטומטיים, וכן ייבדקו שגיאות זיכרון (דליפות, גישה לזיכרון לא מאותחל, וכו').
השתמשו בתוכנית valgrind כדי לגלות שגיאות זיכרון בתוכנית שלכם, כפי שביצעתם בתרגיל בית 1.

פתרון מלא הוא פתרון שממש את כל המחלקות המפורטות במסמך, ללא שגיאות זיכרון, ובהתאם לעקרונות התכנות הנכון שנלמדו בכיתה.

5 חלק יבש

5.1 סעיף א

```
template <class T>
std::vector<T> slice(std::vector<T> vec, int start, int step, int stop);
```

כתבו פונקציה בשם slice בעלת החתימה המופיעה מעלה. הפונקציה מחזירה וקטור חדש המכיל את כל הערכים מ-vec החל מאינדקס start (כולל) ועד אינדקס stop (לא כולל), בקפיצות של step. לדוגמה:

```
// this syntax initializes a vector with values a,b,c,d,e
std::vector<char> vec1 {'a', 'b', 'c', 'd', 'e'};
// returns vector with values a,c
std::vector<char> vec_sliced = slice(vec1, 0, 2, 4);
// returns vector with values b,c,d,e
std::vector<char> vec_sliced = slice(vec1, 1, 1, 5);
```

הערות:

- אם start קטן מ-0 או גדול או שווה לגודל הוקטור, זרקו חריגת BadInput.
- אם stop קטן מ-0 או גדול מגודל הוקטור, זרקו חריגת BadInput.
- אם step קטן או שווה ל-0, זרקו חריגת BadInput.
- אם start גדול או שווה ל-stop, אז הוקטור המוחזר יהיה ריק (אלא אם כן הדרישות האחרות מחייבות זריקת חריגה).

5.2 סעיף ב

נתון קטע הקוד הבא:

```
class A {
public:
  std::vector<int*> values;
  void add(int x) { values.push_back(new int(x)); }
};

int main() {
  A a, sliced;
  a.add(0); a.add(1); a.add(2); a.add(3); a.add(4); a.add(5);
  sliced.values = slice(a.values, 1, 1, 4);
  *(sliced.values[0]) = 800;
  std::cout << *(a.values[1]) << std::endl;
  return 0;
}
```

כתבו גרסה חדשה למחלקה A, על מנת שהרצת ה-main תענה על הדרישות הבאות:

1. הפונקציה תדפיס למסך את המספר 800.
2. לא יתרחשו דליפות זיכרון, גישה לזיכרון לא מאותחל, שחרור כפול או שגיאות זיכרון אחרות.

הערות:

- הניחו שהפונקציה slice קיימת, ושהיא פועלת לפי המתואר בסעיף א'.
- מותר לשנות אך ורק את המחלקה A. אסור לשנות את ה-main.
- אין צורך לציין include-ים.

6 הגשה

- ההגשה היא בזוגות בלבד, באתר הקורס ב-Webcourse.
- יש להגיש את הפתרון שלכם כקובץ zip אשר מכיל את התיקיה solutoin בלבד, ובתיקיה הזאת יש לשים את קבצי ה-h וה-cpp.
- את הפתרון לחלק היבש יש לכתוב בקובץ בשם dry.cpp, ויש לצרף אתו בתיקייה הראשית של ההגשה.
- אין לצרף קבצים מחוץ לתיקיה solution, למעט הקובץ dry.cpp.
- ניתן להגיש מספר פעמים, רק ההגשה האחרונה נחשבת.

בהצלחה!!!