מבוא לתכנות מערכות תרגיל בית 2 – אביב 2021

תאריך פרסום: 26.5.2021

תאריך הגשה: 20.6.2021

משקל התרגיל: 12% מהציון הסופי (תקף).

מתרגלים אחראיים: מוחמד מסארוה ,יניב גופט ואיתי אלפסי.

הערות כלליות

- ההגשה היא בזוגות באתר הwebcourse.
- שימו לב: לא ינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות התרגיל, או לשאול בפורום הקורס בפיאצה. לפני שליחת שאלה – נא וודאו שהשאלה לא מופיעה כבר בפיאצה, ושהתשובה אינה ברורה בFAQ.
- **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! התשובות המועברות שם הן מחייבות.
- העתקות מכל סוג תטופלנה בחומרה.
- התרגיל מורכב מחלק רטוב וחלק יבש.
- **על כל הקוד להיות תחת namespace mtm.**

חלק א

בחלק זה של התרגיל תממשו חלק מתוך מערכת ניהול המבחנים באתר students.

1. מחלקת ExamDetails

מחלקה זו מתארת בחינה במערכת. עליכם לממש מחלקה בשם ExamDetails, אשר תספק את הממשק הבא:

1. **<u>בנאי:</u>** הבנאי מקבל שישה פרמטרים:
   - מס' קורס – מס' שלם
   - חודש הבחינה – נע בין 1 ל12.
   - יום הבחינה – ניתן להניח שכל חודש מכיל 30 ימים.
   - שעת הבחינה – ניתן להניח שמבחן מתחיל רק בשעות עגולות או בחצאי שעות (יכול להתחיל ב06:00 או 13:30, אבל לא ב14:15).
   - אורך הבחינה– שעות עגולות בלבד. אין הגבלה על אורך בחינה.
   - לינק לחדר הזום של הבחינה. **מוגדר ערך ברירת מחדל** – אם לא ניתן לינק, הוא מוגדר כמחרוזת ריקה. ניתן לייצג את הלינק באמצעות std::string.

   עליכם להגדיר חריגה מסוג ExamDetails::InvalidDateException ולזרוק אותה במקרה שנתוני התאריך אינם תקינים.

2. **הורס.**
3. **בנאי העתקה ואופרטור השמה**.
4. **מתודת getLink:** מתודה זו מחזירה את הלינק לבחינה (לא מקבלת פרמטרים).
5. **מתודת setLink:** מתודה זו מקבלת לינק חדש לבחינה ומעדכנת את האובייקט בהתאם.
6. **operator-:** אופרטור שמקבל בחינה נוספת ומחזיר את הפרש הימים בין הבחינה השמאלית לבין הבחינה הימנית. האופרטור יכול להחזיר ערך שלילי.
7. **operator<:** אופרטור שמקבל בחינה נוספת ומחזיר true אם הבחינה השמאלית מתרחשת **לפני** הבחינה הימנית. עבור בחינות שמתקיימות במקביל האופרטור מחזיר false.
8. **הדפסה:** הדפסת התאריך באמצעות אופרטור הפלט. פורמט הפלט יפורט בדוגמת הקוד.

9. **בנאי סטטי:** יש להגדיר במחלקת ExamDetails מתודה סטטית בשם makeMatamExam שיוצרת ומחזירה עצם ExamDetails שמתארת את הבחינה בקורס מת"ם.
הבחינה המוגדרת בעצם תהיה בתאריך 28.7, תתחיל ב13:00 ותימשך 3 שעות.
הלינק המוגדר עבור הבחינה הוא "https://tinyurl.com/59hzps6m".

דוגמת קוד להתנהגות המחלקה הנדרשת:

```
ExamDetails exam1 = ExamDetails::makeMatamExam();
ExamDetails exam2(104032, 7, 11, 9.5, 3);
ExamDetails exam3 = exam1;


cout << "the difference between MATAM and infi 2m is " << (exam1-exam2) << " days";


ExamDetails& closest = (exam1 < exam2) ? exam1 : exam2;
cout << "your closest exam is:" << endl << closest << endl;
```

הפלט עבור דוגמה זו הוא:

```
the difference between MATAM and infi 2m is 17 days

your closest exam is:
Course Number: 104032
Time: 11.7 at 9:30
Duration: 3:00
Zoom Link:
```

דגשים למימוש:

- המחלקה Exam תמומש בקבצים examDetails.cpp ו examDetails.h.
- אתם רשאים לממש כל מתודה **פרטית** הנדרשת למימוש המחלקה אך אין לשנות או להוסיף לממשק המחלקה.
- אם עוד לא למדתם על חריגות בשלב זה, מומלץ לכתוב את הקוד כאילו אין שגיאות, ולהוסיף טיפול בשגיאות בעזרת חריגות יותר מאוחר.
- במקרה של טיפוסי נתונים פרימיטיביים ומחרוזות, ניתן להעביר ולהחזיר נתונים ממתודות by value.
- הקוד חייב להיכתב בהתאם לכללי הפיתוח הנכון שלמדנו בקורס.
- הפתרון שתגישו חייב לטפל בשגיאות.

2. מבנה נתונים גנרי – SortedList

בחלק זה אסור להשתמש בSTL.

2.1. תיאור כללי

בחלק זה של התרגיל נממש מבנה נתונים גנרי ב++C תוך שימוש בתבניות, חריגות ואיטרטורים.

הערה: כיוון שבמועד פרסום התרגיל טרם נלמדו חריגות, מומלץ בתור התחלה לממש את המבנה ללא התחשבות בשגיאות, ולהוסיף שימוש בחריגות לאחר לימוד הנושא.

מבנה הנתונים אותו נממש הוא רשימה מקושרת ממוינת. והוא צריך לקיים את הדרישות הבאות:

1. כמות האיברים בSortedList אינה חסומה.
2. הוספת איברים מתבצעת ע"י פעולת insert
3. הסרת איברים מבוצעת ע"י פעולת remove
4. **איברי הרשימה ממוינים בכל רגע נתון –** פעולות insert וremove שומרות על הסדר בין איברי הרשימה. הסדר בין האיברים מוגדר ע"י אופרטור > של הטיפוס T.

לתבנית המחלקה SortedList יהיה פרמטר יחיד: class T. פרמטר זה יהיה טיפוס האלמנטים של הSortedList.

שימו לב: עליכם להניח כמות הנחות מינימלית על הטיפוס T. לא ניתן להניח שקיים בנאי חסר ארגומנטים לT.

2.2. ממשק הSortedList

המחלקה תספק את הממשק הבא:

1. בנאי חסר פרמטרים ליצירת SortedList ריק..
   הערה: זכרו להשתמש בעיקרון RAII (Resource Acquisition is Initialization) לתכנות נכון.
2. הורס לSortedList.
3. בנאי העתקה.
4. אופרטור השמה.
5. insert – מתודה שמקבלת אלמנט חדש כפרמטר ומכניסה אותו לרשימה.
6. remove – מתודה שמקבלת **איטרטור** (יפורט בהמשך) על המבנה כפרמטר ומסירה את האלמנט אליו הוא מפנה מהרשימה.
7. length – מתודה שמחזירה את מס' האלמנטים ברשימה.
8. filter – מתודה שמקבלת **פרדיקט** (פונקציה בוליאנית) על טיפוס איברי הרשימה, ומחזירה רשימה חדשה שמכילה **רק את האיברים מהרשימה המקורית שמקיימים את התנאי**.
   הדרכה: יש להשתמש עבור מתודה זו בtemplate נוסף.

9. apply – מתודה שמקבלת **פונקציה** מטיפוס איברי הרשימה לטיפוס **כלשהו**, ומחזירה SortedList שמכילה את תוצאות הפעלת הפונקציה על כל איברי הרשימה.
   הדרכה: גם כאן ניתן להשתמש בטיפוס template נוסף.
10. begin – מתודה שמחזירה **איטרטור** לתחילת הרשימה.
11. end – מתודה שמחזירה **איטרטור** לסוף הרשימה.

2.3. איטרטור קבוע עבור SortedList

בנוסף לממשק הקיים, נממש מחלקת איטרטור קבוע עבור הרשימה, אשר תוגדר בתור SortedList::const_iterator.

על האיטרטור לספק את הממשק הבא:

1. בנאי העתקה ואופרטור השמה – מקבלים כפרמטר איטרטור אחר ומבצעים פעולה סטנדרטית.
   אסור לאפשר למשתמש גישה לבנאי הסטנדרטי של המחלקה.
2. הורס לאיטרטור.
3. operator++ – מימוש לאופרטור ++ שמקדם את האיטרטור לאיבר הבא ברשימה.
   אם האיטרטור מצביע לסוף הרשימה, **יש לזרוק חריגה** מטיפוס std::out_of_range.
4. operator== – מימוש לאופרטור == שמקבל איטרטור נוסף ובודק האם שניהם שווים.
5. operator* - מימוש לאופרטור * שיחזיר **const reference** לאיבר אליו האיטרטור מפנה.

דגשים למימוש:

- אנחנו ממליצים להתחיל ממימוש של SortedList כרשימה **לא** גנרית ללא filter, apply.
  לאחר מכן לממש גנריות, ולבסוף לממש את filter וapply.
- את המימוש עליכם לכתוב בקובץ SortedList.h.
- יש לממש את המבנה כך שיהיה **לא תלוי** במשתמש מבחינת זיכרון. כלומר, עליו ליצור עותקים משלו לכל האובייקטים שהוא מקבל מהמשתמש.
- אתם רשאים להוסיף מתודות פרטיות ומחלקות עזר כרצונכם, תוך שמירה על עקרונות התכנות הנכון. אין לשנות את ממשק הרשימה או האיטרטור.
- הרשימה יכולה להכיל ערכים שווים מבחינת operator<. הדרישה היחידה היא סידור לפי הסדר שמגדיר האופרטור.

דוגמת קוד להמחשת הממשק:

```
using std::string;
string getLen(string str)
{
    return std::to_string(str.length());
}

template<class T>
void printList(SortedList<T> list) {
    for (auto it = list.begin(); !(it == list.end()); ++it) {
        cout << *it << endl;
    }
    cout << endl;
}

int main()
{
    SortedList<string> lst1 = SortedList<string>();
    lst1.insert("Charlie");
    lst1.insert("Bob");
    lst1.insert("Alice");
    lst1.insert("Donald");
    printList(lst1);

    SortedList<string> lst2 = lst1;
    lst2 = lst2.apply(getLen);
    printList(lst2);

    SortedList<string>::const_iterator it = lst2.begin();
    cout << *it << endl << endl;
    ++it;
    lst2.remove(it);
    printList(lst2);
    return 0;
```

הפלט המתקבל עבור דוגמה זו הוא:

```
Alice
Bob
Charlie
Donald

3
5
6
7

3

3
6
7
```

3. שאלות יבשות על SortedList

עליכם לענות על השאלות הבאות ולהגישן כקובץ PDF בשם dry.pdf שיצורף להגשה הסופית.

1. במבנה הSortedList הגנרי, מה הדרישות **ההכרחיות** שעל הטיפוס T לקיים? הסבירו.

2. נניח כי היינו רוצים לממש איטרטור non-const עבור SortedList. כלומר, עבור איטרטור זה האופרטור * היה מחזיר &T. איזו בעיה עלולה להיווצר במימוש?

3. במתודה filter, המימוש כולל שימוש בtemplate נוסף כדי להעביר את הפרדיקט. אילו שתי דרכים שונות קיימות בשפת C++ למימוש והעברת הפרדיקט? מה ההבדל ביניהן? מדוע אנחנו לא צריכים לספק שני מימושים שונים כדי לתמוך בשתי הדרכים הללו?

הוראות הידור, קישור ובדיקה

התיקייה התיקייה `~mtm/public/2021b/ex2/provided` מכילה קבצי עזר להידור ובדיקת התרגיל, ומומלץ להעתיק אותה לתיקיית התרגיל הפרטית שלכם. התרגיל ייבדק על על השרת csl3, ועליו יבוצע ההידור בעזרת הפקודות הבאות.

הקובץ part_a_test.cpp מכיל קובץ בדיקה בסיסי.

על הקוד של חלק א לעבור הידור בעזרת הפקודה הבאה, כאשר היא מורצת מתוך התיקייה עם קבצי המקור:
שימו לב שכדי להריץ את הטסט של הסגל, יש לשים אותו באותה התיקייה עם קבצי המקור.

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

בהמשך תפורסם גם תוכנית finalCheck. באמצעותה תוכלו לבדוק את ההגשה שלכם. השתמשו ב-finalCheck כדי לוודא שההגשה שלכם עוברת הידור בהצלחה. התוכנית שלכם **חייבת** לעבור את finalCheck לפני ההגשה.

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

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

בהצלחה! ☺