מבוא לתכנות מערכות – תרגיל בית 3 – חורף 2020-2021

תאריך פרסום: 31/12/2020
תאריך הגשה: 24/01/2021
משקל התרגיל: 12% מהציון הסופי (תקף)
מתרגל אחראי: בר מגל

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 רקע

לאור ההצלחה המסחררת של מערכת ניהול האירועים עבור ועד מדמ"ח, הוחלט לבנות בנוסף מערכת רישום לאירועים, שתאפשר לעיין באירועים הקיימים ולרשום אורחים לאירועים הללו. עליכם לבנות את המערכת הזו בשפת C++, על סמך ההוראות המוצגות במסמך זה.

3 חלק א: מחלקת DateWrap

לכל אירוע יש תאריך, ועל כן מערכת רישום לאירועים צריכה מחלקה שתייצג תאריכים. בתיקייה `~mtm/public/2021a/ex3/provided` מסופקים לכם הקובץ date.h מתרגיל בית 1, והקובץ libdate.a שמכיל מימוש בשפת C של Date ADT שהוגדר בתרגיל בית 1. כתבו מחלקה בשם DateWrap, אשר תעטוף את Date ADT, ותספק את הממשק הבא:

1. **בנאי**: הבנאי מקבל שלושה פרמטרים: יום, חודש ושנה. אם התאריך אינו חוקי, יש לזרוק חריגת InvalidDate.
2. **מתודות day, month ו-year**: לא מקבלות פרמטרים, ומחזירות עותק של היום, חודש או שנה של this.
3. הדפסת תאריך בעזרת **אופרטור הפלט**. **פורמט ההדפסה המדויק יתואר בהמשך.**
4. השוואת תאריכים בעזרת **האופרטורים** >, <, ==, =<, => **ו-**!=. בהשוואה בין שני תאריכים, התאריך ה-"קטן" יותר מוגדר להיות התאריך המוגדר המוקדם יותר מביניהם.
5. קידום תאריך ביום אחד בעזרת **האופרטור ++**. יש לתמוך ב-++ מצד ימין בלבד, ולא מצד שמאל.
6. קידום תאריך במספר ימים בעזרת **האופרטור =+**. אם התקבל מספר שלילי בתור פרמטר, יש לזרוק חריגת NegativeDays.
7. חיבור בין תאריך ומספר אי-שלילי days בעזרת **האופרטור +**, כדי לקבל תאריך חדש המתרחש days ימים אחרי התאריך המקורי. אם התקבל מספר שלילי בתור פרמטר, יש לזרוק חריגת NegativeDays.

הערות:

- המחלקה DateWrap תמומש בקבצים date_wrap.h ו-date_wrap.cpp.

- המחלקה DateWrap צריכה לעמוד בעקרונות התכנות הנכון שנלמדו בכיתה, כולל עיקרון RAII (Resource Acquisition is Initialization, ראו תרגול 10 שקף 47).
- לשם פשטות, וכמו בתרגיל בית 1, הניחו שבכל חודש יש 30 ימים, ואין שנים מעוברות. תאריכים בעלי שנה שלילית או שווה ל-0 לא ייבדקו.
- תאריך חוקי הוא תאריך שהיום שלו הוא מספר שלם בין 1 ל-30 (כולל), והחודש שלו הוא מספר בין 1 ל-12 (כולל).
- 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.
- בחלק זה אסור להשתמש ב-STL.
- מותר להוסיף מתודות ולהרחיב את הממשק של DateWrap, למשל עם בנאי העתקה, אך בלי להפר את עקרונות התכנות הנכון.
- מסופקים לכם הקובץ date.h מתרגיל בית 1, והקובץ libdate.a שמכיל מימוש של Date בשפת C שניתן להשתמש בו ב-csl3. קבצים אלה יצורפו לפתרון שלכם בעת בדיקת התרגיל, ואין לצרף אותם להגשה. כדי להשתמש ב-Date מתוך קוד ++C, יש לעטוף את ה-include ל-date.h בתוך בלוק של "extern "C, ולקרוא לפונקציות מ-date.h כרגיל. בתהליך הקמפול צרפו את libdate.a בשלב הנכון, בעזרת דגלי L- ו--l.
- הקובץ libdate.a מתאים ל-csl3 ונבנה בעזרת gcc. כדי לעבוד על התרגיל במחשב האישי שלכם, תצטרכו לבנות את libdate.a באותו המחשב ואותו הקומפיילר בעזרתו אתם מקמפלים את התרגיל. ראו הוראות בהמשך.

דוגמה לשימוש במחלקת DateWrap:

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

int main() {
    DateWrap date(30, 11, 2020);
    cout << date << endl; // output: "30/11/2020"
    cout << date + 4 << endl; // output: "4/12/2020"
    cout << 3 + date << endl; // output: "3/12/2020"
    date++;
    cout << date << endl; // output: "1/12/2020"
    date += 7;
    cout << date << endl; // output: "8/12/2020"
    cout << (date > DateWrap(29, 11, 2020)) << endl; // output: "1"
    cout << (date <= DateWrap(29, 11, 2020)) << endl; // output: "0"
    cout << (date == DateWrap(30, 11, 2020)) << endl; // output: "0"
    date += (-3); // throw exception NegativeDays
    date = date + (-3); // throw exception NegativeDays
    return 0;
}
```

דוגמה לשימוש ב-extern “C”:

```
// assuming "some_c_module.o" is an object file compiled from a C source
// assuming "some_c_module.h" is the matching C header file
// assuming "some_c_module.h" contains a function named `doFoo`

// using an extern "C" block prevents the C++ compiler from performing
// name-mangling inside the block. Result: function names are compatible
// with the C compiler.

extern "C" {
    #include "some_c_module.h"
}

void printFoo(int x) {
    std::cout << doFoo(x) << std::endl;
}
```

3.1 שימוש ב-Date שכתבתם בעצמכם בתרגיל בית 1

בעת כתיבת התרגיל, ייתכן שתרצו לקמפל את date.c שכתבתם בעצמכם בתרגיל בית 1, במקום להשתמש ב-libdate.a שאנו מספקים. למשל, אם תרצו לדבג את הפתרון במחשב האישי שלכם (לא על השרת), תצטרכו לקמפל את date.c על המחשב האישי שלכם.

3.1.1 קמפול בעזרת Make

בהנחה שכתבתם Makefile בתרגיל בית 1 שמגדיר את המקרואים CC ו-CFLAGS (ראו תרגול 5), מספיק שתוסיפו כלל ריק עבור libdate(date.o), כלומר כלל שלא מכיל פקודות, ולאחר מכן תוכלו ליצור את libdate.a בעזרת הפקודה make libdate.a.

```
libdate.a(date.o): date.o
```

התוכנית make תדע אוטומטית איך לבנות את date.o ו-libdate.a על סמך הערכים של CC ו-CFLAGS, בעזרת הפעלת התוכנית ar.

אפשרות אחרת, שלא תלויה ב-CC ו-CFLAGS, היא להוסיף את הכלל הבא:

```
libdate.a: date.o
      ar rv libdate.a date.o
```

גם האפשרות הזו תגרום ל-make להכיר את הפקודה make libdate.a.

3.1.2 קמפול בעזרת CMake

על מנת לקמפל את libdate.a בעזרת CMake, הוסיפו לקובץ CMakeLists.txt את הפקודה הבאה:

```
add_library(date date.c date.h)
```

כעת תוכלו ליצור את libdate.a בעזרת בניית ה-target בשם date.

4 חלק ב: סוגי אירועים ואוספי אירועים

במערכת רישום האירועים יהיו קיימים אירועים מכל מיני סוגים: אירועים פתוחים לקהל הרחב, אירועים סגורים לפי רשימת מוזמנים, וכו'. לכל סוג אירוע תהיה מחלקה מתאימה, וכל מחלקה כזו תירש ממחלקת אב משותפת בשם BaseEvent. עליכם להגדיר ולממש את כל המחלקות המתוארות בסעיף "סוגי אירועים", כולל המחלקה BaseEvent.

בנוסף, ישנם אירועים החוזרים על עצמם, וישנם אירועים חד-פעמיים. כדי לייצג זאת במערכת, נגדיר מחלקות עבור "אוספי אירועים" אשר יירשו ממחלקת אב משותפת בשם EventContainer. עליכם להגדיר ולממש את כל המחלקות המתוארות בסעיף "אוספי אירועים", כולל המחלקה EventContainer.

הערות:

- המשתתפים באירועים הם כולם סטודנטים, ואנחנו נייצג סטודנט ע"י מספר בלבד. כלומר, מבחינתנו בתרגיל הזה סטודנט הוא מספר.
- נגדיר סטודנט חוקי כמספר חיובי בין 1 ל-1234567890 (כולל 1 וכולל 1234567890).
- כל מתודה שמקבלת סטודנט כפרמטר, צריכה לזרוק חריגת InvalidStudent אם היא קיבלה סטודנט לא חוקי.
- לכל המחלקות מותר להוסיף מתודות בנוסף למתודות הנדרשות, לפי הצורך ותוך הקפדה על עקרונות התכנות הנכון שנלמדו בכיתה.
- שימו לב, בניגוד לתרגיל בית 1, בתרגיל הזה איננו מספקים לכם קבצי header (למעט date.h). במקום זאת, עליכם לכתוב גם את קבצי ה-header בעצמכם ולהגיש אותם כחלק מההגשה.
- בחלק זה אסור להשתמש ב-STL.

4.1 סוגי אירועים

4.1.1 מחלקת BaseEvent

מחלקת אב **אבסטרקטית** משותפת עבור סוגי אירועים. לכל אירוע יש תאריך (DateWrap), שם (מחרוזת) ורשימת משתתפים. בנוסף, המחלקה צריכה לספק את הממשק הבא:

1. **בנאי**: הבנאי יקבל תאריך ושם, ויאתחל BaseEvent חדש עם התאריך והשם שהתקבלו, ועם רשימת משתתפים ריקה.
2. **מתודת registerParticipant**: מקבלת סטודנט ומנסה לרשום אותו לרשימת המשתתפים. אם הסטודנט כבר רשום, יש לזרוק חריגת AlreadyRegistered. אם לסטודנט אסור להירשם לאירוע, יש לזרוק חריגת RegistrationBlocked.
3. **מתודת unregisterParticipant**: מקבלת סטודנט ומסירה את הרישום שלו מרשימת המשתתפים. אם הסטודנט לא רשום, יש לזרוק חריגת NotRegistered.
4. **מתודת printShort**: מקבלת std::ostream, מדפיסה אליו תיאור קצר של האירוע, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
5. **מתודת printLong**: מקבלת std::ostream, מדפיסה אליו תיאור מפורט של האירוע כולל רשימת משתתפים, ומחזירה את ה-std::ostream שהיא קיבלה. **פורמט ההדפסה המדויק יתואר בהמשך.**
6. **מתודת clone**: לא מקבלת פרמטרים. מעתיקה את this ומחזירה מצביע לעותק החדש. המתודה צריכה להעתיק גם את רשימת המשתתפים.

הערות:

- המחלקה BaseEvent לא מוסיפה איסורים על סטודנטים להירשם, לכן לא מצפים ממנה לזרוק RegistrationBlocked בעצמה. מחלקות יורשות רשאיות להוסיף איסורים, והן אלה שצריכות לזרוק RegistrationBlocked במקרה הצורך – זו הדרך שלהן לסמן שסטודנט אינו רשאי להירשם.
- להזכירכם, מבחינתנו בתרגיל הזה סטודנט הוא מספר, וכל פונקציה שמקבלת סטודנט כפרמטר צריכה לזרוק חריגה אם הסטודנט אינו חוקי.

4.1.2 מחלקת OpenEvent

מחלקה המייצגת אירוע פתוח לקהל הרחב. עליה לספק את הממשק הבא:

1. **בנאי**: כמו BaseEvent.
2. **מתודת registerParticipant**: כמו BaseEvent. כל סטודנט רשאי להירשם לכל אירוע מטיפוס OpenEvent.
3. **מתודת unregisterParticipant**: כמו BaseEvent.

4. **מתודת printShort**: כמו BaseEvent.
5. **מתודת printLong**: כמו BaseEvent.
6. **מתודת clone**: כמו BaseEvent.

4.1.3 מחלקת ClosedEvent

מחלקה המייצגת אירוע סגור לפי רשימת מוזמנים. לכל אירוע מטיפוס ClosedEvent תהיה רשימת מוזמנים, שזוהי רשימת הסטודנטים הרשאים להירשם לאירוע. על המחלקה לספק את הממשק הבא:

1. **בנאי**: הבנאי יקבל תאריך ושם, ויאתחל ClosedEvent חדש עם התאריך והשם שהתקבלו, ועם רשימת מוזמנים ריקה.
2. **מתודת addInvitee**: (עם שני e בסוף) מקבלת סטודנט ומוסיפה אותו לרשימת המוזמנים. אם הסטודנט כבר נמצא ברשימת המוזמנים, יש לזרוק חריגת AlreadyInvited.
3. **מתודת registerParticipant**: כמו BaseEvent. סטודנט רשאי להירשם לאירוע מטיפוס ClosedEvent רק אם הוא נמצא ברשימת המוזמנים של אותו אירוע.
4. **מתודת unregisterParticipant**: כמו BaseEvent.
5. **מתודת printShort**: כמו BaseEvent.
6. **מתודת printLong**: כמו BaseEvent.
7. **מתודת clone**: כמו BaseEvent. המתודה צריכה להעתיק גם את רשימת המוזמנים.

4.1.4 מחלקת CustomEvent<CanRegister>

תבנית מחלקה (template) המייצגת אירועים בעלי תנאי הרשמה מהטיפוס הגנרי CanRegister. תנאי הרשמה הוא פונקציה או Function Object, אשר מקבלת סטודנט ומחזירה true אם הסטודנט רשאי להירשם לאירוע ו-false אם אסור לו. על המחלקה לספק את הממשק הבא:

1. **בנאי**: הבנאי יקבל תאריך, שם ותנאי הרשמה, ויאתחל CustomEvent חדש עם התאריך והשם שהתקבלו, ועם תנאי ההרשמה שסופק.
2. **מתודת registerParticipant**: כמו BaseEvent. סטודנט רשאי להירשם לאירוע מטיפוס CustomEvent רק אם הפעלת תנאי ההרשמה על הסטודנט הזה מחזירה true.
3. **מתודת unregisterParticipant**: כמו BaseEvent.
4. **מתודת printShort**: כמו BaseEvent.
5. **מתודת printLong**: כמו BaseEvent.
6. **מתודת clone**: כמו BaseEvent. המתודה צריכה להעתיק גם את תנאי ההרשמה.

4.2 אוספי אירועים

4.2.1 מחלקת EventContainer

מחלקת אב **אבסטרקטית** משותפת עבור אוספי אירועים. המחלקה צריכה לספק את הממשק הבא:

1. **בנאי חסר פרמטרים**: מאתחל את EventContainer כך שלא יכיל אף אירוע.
2. **מתודת add**: מקבלת רפרנס ל-BaseEvent. אם אוסף האירועים תומך בהוספת אירועים, המתודה מוסיפה עותק שלו לאוסף האירועים. אחרת, המתודה זורקת חריגת NotSupported.
3. **מתודת begin**: מתודה חסרת פרמטרים, אשר מחזירה איטרטור מטיפוס EventContainer::EventIterator שמצביע לאירוע הראשון השמור בתוך אוסף האירועים.
4. **מתודת end**: מתודה חסרת פרמטרים, אשר מחזירה איטרטור מטיפוס EventContainer::EventIterator שמעיד על סיום איטרציה על האירועים הרשומים בתוך אוסף האירועים ("איטרטור למקום האחד אחרי האיבר האחרון באוסף").
5. האירועים ב-EventContainer צריכים להיות מסודרים מהמוקדם ביותר (לפי התאריך) למאוחר ביותר. כלומר, סדר האיטרציה צריך להיות זהה לסדר התאריכים של האירועים. אם יש שני אירועים בעלי אותו תאריך, אז האירוע בעל השם הקטן יותר בסדר לקסיקוגרפי צריך להיות הראשון מבין שניהם.

אתם חופשיים להגדיר את הטיפוס EventContainer::EventIterator כראות עיניכם, אך הוא צריך לתמוך בבנאים ובאופרטורים הבאים:

1. **בנאי העתקה ואופרטור השמה**: איטרטור המטרה (this) יצביע לאותו אוסף אירועים ואותו איבר באוסף האירועים כמו האיטרטור המקורי.
2. **אופרטור ++ מצד שמאל**: ביצוע iter++ יקדם את האיטרטור לאיבר הבא באוסף האירועים.
3. **אופרטור dereference**: ביצוע iter* יחזיר אובייקט מטיפוס **רפרנס** ל-BaseEvent.
4. **אופרטור השוואה ואי-שוויון**: ביצוע iter1 == iter2 או iter1 != iter2 יחזיר אמת או שקר בהתאם להאם האיטרטורים מצביעים על אותו איבר.

דוגמת שימוש:

```
// assume `ec` in an EventContainer with 2 events
EventContainer& ec = ...;
EventContainer::EventIterator it = ec.begin();
EventContainer::EventIterator it_end = ec.end();

BaseEvent& ev = *it; // `ev` is the first event stored in `ec`
Ev.printShort(); // print short description of 1st event
(*it).printShort(); // same output as previous line

++it;
(*it).printShort(); // print short description of 2nd event

std::cout << (it == it_end) << std::endl; // print "0"
std::cout << (it != it_end) << std::endl; // print "1"

++it;
std::cout << (it == it_end) << std::endl; // print "1"
std::cout << (it != it_end) << std::endl; // print "0"
```

4.2.2 מחלקת Festival

אוסף אירועים המכיל מספר אירועים המתרחשים כולם באותו יום. לכל האירועים יש רשימות משתתפים נפרדות. על מחלקת Festival לספק את הממשק הבא:

1. **בנאי**: בנאי המקבל תאריך, ומתאחל פסטיבל ריק (חסר אירועים) המתרחש באותו התאריך.
2. **מתודת add**: מקבלת רפרנס ל-BaseEvent ומוסיפה עותק שלו לפסטיבל. אם התאריך של האירוע שונה מהתאריך של הפסטיבל, יש לזרוק חריגת DateMismatch ולא להוסיף את האירוע לפסטיבל.
3. **מתודות begin ו-end**: כמו EventContainer.

4.2.3 מחלקת RecurringEvent<EventType>

תבנית מחלקה (template) המייצגת אוסף אירועים המכיל אירוע שחוזר על עצמו מספר פעמים, בהפרש ימים קבוע בין כל מופע. אירוע שחוזר על עצמו ייוצג ע"י מספר אירועים שונים בעלי שם זהה אבל תאריכים נפרדים ורשימות משתתפים נפרדות. על מחלקת RecurringEvent<EventType> לספק את הממשק הבא:

1. **בנאי**: בנאי המקבל תאריך first_date, שם, מספר מופעים num_occurrences, ומספר ימי הפרש בין מופעים interval_days. הבנאי מאתחל num_occurrences אירועים חדשים מטיפוס EventType, כאשר האירוע הראשון מתרחש בתאריך first_date ובין כל שני אירועים עוקבים יש הפרש של interval_days ימים. הבנאי שומר את האירועים האלה בתוך ה-RecurringEvent החדש. אם num_occurrences או interval_days אינם חיוביים, יש לזרוק חריגת InvalidNumber או InvalidInterval, בהתאמה.
2. **מתודת add**: מקבלת רפרנס ל-BaseEvent וזורקת חריגת NotSupported.
3. **מתודות begin ו-end**: כמו EventContainer.

4.2.4 מחלקת OneTimeEvent<EventType>

תבנית מחלקה (template) המייצגת אוסף אירועים המכיל אירוע יחיד. עליה לספק את הממשק הבא:

1. **בנאי**: בנאי המקבל תאריך ושם, יוצר אירוע חדש מטיפוס EventType בעל התאריך והשם הנ"ל, ושומר את האירוע הזה בתוך ה-OneTimeEvent החדש.
2. **מתודת add**: מקבלת רפרנס ל-BaseEvent וזורקת חריגת NotSupported.
3. **מתודות begin ו-end**: כמו EventContainer.

```
using namespace mtm;
typedef EventContainer::EventIterator Iter;

void printEventsShort(EventContainer& events) {
  for (Iter iter = events.begin(); iter != events.end(); ++iter) {
    BaseEvent& event = *iter;
    event.printShort(std::cout);
  }
}

int main() {
  Festival festival(DateWrap(21, 10, 2020));
  festival.add(OpenEvent(DateWrap(21, 10, 2020), "Performance 1"));
  ClosedEvent closed(DateWrap(21, 10, 2020), "Performance 2");
  closed.addInvitee(1);
  closed.addInvitee(500);
  festival.add(closed);
  printEventsShort(festival);

  RecurringEvent<OpenEvent> recurring(
      DateWrap(21, 10, 2020), "Wednesday Noon", 13, 7);
  printEventsShort(recurring);

  OneTimeEvent<OpenEvent> one_time(
      DateWrap(21, 10, 2020), "Start of Semester"));
  printEventsShort(one_time);

  return 0;
}
```

5 חלק ג: לוח אירועים

המערכת תכיל לוח אירועים אחד, ותאפשר הוספת אירועים ללוח וכן רישום סטודנטים לאירועים הקיימים בלוח האירועים. כתבו את המחלקה Schedule אשר תייצג לוח אירועים, ומספקת את הממשק הבא:

1. **בנאי**: הבנאי לא מקבל פרמטרים, ומאתחל לוח אירועים ריק.
2. **מתודת addEvents**: מקבלת רפרנס ל-EventContainer ומוסיפה עותקים של כל האירועים באוסף ללוח האירועים. אם לפחות אחד מהאירועים ב-EventContainer כבר נמצא בלוח, אז יש לזרוק חריגת EventAlreadyExists, ואין להוסיף אף אחד מהאירועים ב-EventContainer ללוח האירועים.
3. **מתודת registerToEvent**: מקבלת תאריך, שם וסטודנט, ומנסה להוסיף את הסטודנט לאירוע בעל התאריך והשם הנתונים. אם הסטודנט כבר רשום יש לזרוק חריגת AlreadyRegistered, ואם

לסטודנט אסור להירשם יש לזרוק חריגת RegistrationBlocked. אם האירוע לא קיים, יש לזרוק חריגת EventDoesNotExist.

4. **מתודת unregisterFromEvent**: מקבלת תאריך, שם וסטודנט, ומנסה להסיר את הסטודנט מהאירוע בעל התאריך והשם הנתונים. אם הסטודנט לא רשום יש לזרוק חריגת NotRegistered. אם האירוע לא קיים, יש לזרוק חריגת EventDoesNotExist.
5. **מתודת printAllEvents**: מדפיסה למסך את התיאור הקצר של כל אירוע, כך שכל אירוע מוצע בשורה משלו. **פורמט ההדפסה המדויק יתואר בהמשך.**
6. **מתודת printMonthEvents**: מקבלת חודש (מספר בין 1 ל-12) ושנה (מספר שלם) ומדפיסה למסך את התיאור הקצר של כל אירוע המתרחש בחודש הנתון בשנה הנתונה. **פורמט ההדפסה המדויק יתואר בהמשך.**
7. **מתודת printSomeEvents**: מקבלת פרדיקט (predicate) ומשתנה בוליאני אופציונלי בשם verbose, ומדפיסה למסך את התיאור של כל אירוע שהפרדיקט החזיר עבורו true. אם verbose הוא true אז יודפס התיאור הארוך של כל אירוע, ואם הוא false יודפס התיאור הקצר של כל אירוע. בהקשר הזה, פרדיקט הוא פונקציה (או function object) שמקבלת פרמטר אחד מטיפוס const BaseEvent& ומחזירה true או false. ערך ברירת המחדל של הפרמטר verbose הוא false. **פורמט ההדפסה המדויק יתואר בהמשך.** רמז: מחלקה יכולה להכיל תבניות (templates) בתוכה.
8. **מתודת printEventDetails**: מקבלת שם ותאריך של אירוע, ומדפיסה למסך את התיאור המפורט של האירוע הנתון, כך שיש שורה ריקה בסוף של כל תיאור. אם האירוע לא קיים, יש לזרוק חריגת EventDoesNotExist. **פורמט ההדפסה המדויק יתואר בהמשך.**

הערות:

- אין מעקב או רישום גלובליים של סטודנטים בלוח האירועים. כל אירוע מנהל רישום סטודנטים משל עצמו, ואין שום צורך ב-"רישום מקדים" כזה או אחר במערכת לוח האירועים.
- בחלק זה מותר להשתמש ב-STL.

6 פורמטי פלט

6.1 פורמט הדפסה עבור אופרטור הפלט של DateWrap

כאשר מדפיסים אוביקטט DateWrap בעזרת אופרטור הפלט, יש להדפיס על פי הפורמט הנ"ל, ללא ירידת שורה:

```
<day>/<month>/<year>
```

בהדפסה, אין לרפד את היום, החודש או השנה באפסים מצד שמאל. דוגמאות להדפסות נכונות:

```
30/12/2020
5/9/872
```

דוגמאות להדפסות שגויות:

```
2020/12/30
12/30/2020
05/09/0872
```

6.2 פורמט הדפסה עבור מתודת printShort

לכל אירוע, פורמט ההדפסה עבור printShort הוא שורה המתחילה בשם האירוע, אחריו רווח בודד, ואז תאריך האירוע. הפורמט כולל את תו ירידת השורה בסוף השורה:

```
<Event Name> <day>/<month>/<year>
```

לדוגמה:

```
Boxing Day 26/12/2020
```

לדוגמאות נוספות, ראו בקבצי הטסטים המסופקים.

6.3 פורמט הדפסה עבור מתודת printLong

הפורמט זהה לפורמט עבור printShort, אבל לאחר השורה של התיאור הקצר מופיעים כל הסטודנטים הרשומים לאירוע, סטודנט אחד בכל שורה, כולל תו ירידת שורה בסוף השורה של הסטודנט האחרון. הסטודנטים צריכים להיות מסודרים בסדר עולה:

```
<Event Name> <day>/<month>/<year>
<student1>
<student2>
...
<studentN>
```

לדוגמה:

```
Boxing Day 26/12/2020
1122334
1234567
7654321
```

לדוגמאות נוספות, ראו בקבצי הטסטים המסופקים.

6.4 פורמט הדפסה עבור מתודת printAllEvents

בלוח האירועים, פורמט ההדפסה של printAllEvents הוא שלכל אירוע מופיע התיאור הקצר שלו, ואחריו שורה ריקה. האירועים צריכים להיות מודפסים בסדר עולה לפי התאריך שלהם. אם שני אירועים הם בעלי תאריך זהה, אז האירוע בעל השם הקטן יותר בסדר לקסיקוגרפי יודפס קודם:

```
<First Event Name> <day>/<month>/<year>

<Second Event Name> <day>/<month>/<year>

...

<Last Event Name> <day>/<month>/<year>
```

לדוגמה:

```
Some Event That Recurs Twice 1/12/2020

Boxing Day 26/12/2020

Some Event That Recurs Twice 1/1/2021
```

לדוגמאות נוספות, ראו בקבצי הטסטים המסופקים.

6.5 פורמט הדפסה עבור מתודת printMonthEvents

פורמט ההדפסה של printMonthEvents הוא זהה ל-printAllEvents. ההבדל בין המתודות הוא איזה אירועים צריך להדפיס.

6.6 פורמט הדפסה עבור מתודת printSomeEvents

פורמט ההדפסה של printSomeEvents הוא זהה ל-printAllEvents, אלא אם הועבר הפרמטר verbose = true. ההבדל בין המתודות הוא איזה אירועים צריך להדפיס.

אם הועבר הפרמטר verbose = true, אז הפורמט הוא זהה פרט לכך שצריך להוסיף מתחת לכל שם ותאריך של אירוע את רשימת המשתתפים באותו אירוע. **ראו דוגמה בקבצי הטסטים המסופקים.**

6.7 פורמט הדפסה עבור מתודת printEventDetails

בהינתן האירוע המבוקש, יש להדפיס בשורה הראשונה את שם האירוע והתאריך שלו. לאחר מכן יש להדפיס את כל הסטודנטים הרשומים לאירוע, סטודנט אחד בכל שורה, ואחרי הסטודנט האחרון יש להדפיס שורה

ריקה. הסטודנטים צריכים להיות מסודרים בסדר עולה:

```
<Event Name> <day>/<month>/<year>
<student1>
<student2>
...
<studentN>
```

לדוגמאות, ראו בקבצי הטסטים המסופקים.

7 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, כולל חריגות הרלוונטיות לחלקים ב' ו-ג', יש לממש בקובץ בשם exceptions.h בתיקיית partA.
- כדי ללמוד עוד על "C" extern, ניתן לקרוא את הדף language_linkage באתר cppreference.com.
- בחלק א' ו-ב' אסור להשתמש ב-STL. בחלק ג' 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 בקבצים הבאים:
  - מחלקת DateWrap: בקובץ date_wrap.h בתיקיית partA
  - כל החריגות: בקובץ exceptions.h בתיקיית partA (גם חריגות של חלקים ב' ו-ג')
  - מחלקת BaseEvent: בקובץ base_event.h בתיקיית partB
  - מחלקת OpenEvent: בקובץ open_event.h בתיקיית partB
  - מחלקת ClosedEvent: בקובץ closed_event.h בתיקיית partB
  - מחלקת CustomEvent: בקובץ custom_event.h בתיקיית partB
  - מחלקת EventContainer: בקובץ event_container.h בתיקיית partB
  - מחלקת Festival: בקובץ festival.h בתיקיית partB
  - מחלקת RecurringEvent: בקובץ recurring_event.h בתיקיית partB
  - מחלקת OneTimeEvent: בקובץ one_time_event.h בתיקיית partB
  - מחלקת Schedule: בקובץ schedule.h בתיקיית partC

8 הידור, קישור ובדיקה

התיקייה ~mtm/public/2021a/ex3/provided מכילה קבצי עזר להידור ובדיקת התרגיל, ומומלץ להעתיק אותה לתיקיית התרגיל הפרטית שלכם. התרגיל ייבדק על השרת csl3, ועליו לעבור הידור בעזרת הפקודות הבאות.

עבור חלק א:

```
g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -o progA -IpartA
-Iprovided provided/test_partA.cpp partA/*.cpp -Lprovided -ldate
```

עבור חלק ב:

```
g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -o progC -IpartA
-IpartB -Iprovided provided/test_partB.cpp partA/*.cpp  partB/*.cpp -
Lprovided -ldate
```

עבור חלק ג:

```
g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -o progB -IpartA
-IpartB -IpartC -Iprovided provided/test_partC.cpp partA/*.cpp
partB/*.cpp partC/*.cpp -Lprovided -ldate
```

פירוט תפקידי חלק מהדגלים בפקודות ההידור:

- **-std=c++11** שימוש בתקן C++11 של שפת C++.
- **-o progA** קובץ הפלט ייקרא progA.
- **-IpartB** פקודות "foo.h" #include יחפשו את הקובץ foo.h גם בתיקייה partB.
- **-Lprovided** ארגומנטי -lfoo יחפשו את הקובץ libfoo.a גם בתיקייה provided.
- **-ldate** הוסף את libdate.a (אשר מכיל קבצי o) בשלב הקישור. הקובץ libdate.a מכיל קובץ o עם מימוש של Date.

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

9 חלק יבש

9.1 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).

9.2 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.

10 הגשה

- ההגשה היא בזוגות בלבד, באתר הקורס ב-Webcourse.
- יש להגיש את הפתרון שלכם כקובץ zip אשר מכיל את התיקיות partA, partB ו-partC בלבד, ובתיקיות האלו יש לשים את קבצי ה-h וה-cpp של כל חלק.
- כל חלק משתמש בחלק הקודם, לכן אל תשכפלו קבצים בין התיקיות השונות. לדוגמה, בהגשה שלכם צריך להיות רק קובץ date_wrap.cpp אחד, בתיקיה partA. אל תעתיקו את הקובץ הזה לתיקיות partB ו-partC.
- את הפתרון לחלק היבש יש לכתוב בקובץ בשם dry.cpp, ויש לצרף אתו בתיקייה הראשית של ההגשה.
- אין לצרף קבצים מחוץ לתיקיות partA, partB ו-partC, למעט הקובץ dry.cpp.
- אין לצרף לפתרון את הקבצים המסופקים לכם בתיקיית provided. אנחנו נוסיף את התיקייה הזו בעצמנו לכל הגשה בעת בדיקת התרגיל, ע"י העברת ארגומנט I- וארגומנט L- מתאימים לפקודות ההידור.
- ניתן להגיש מספר פעמים, רק ההגשה האחרונה נחשבת.

בהצלחה!!!