

תרגיל 5: C++ מתקדם

כללי

1. מועד הגשה: 26/01/2020 בשעה 23:55.
2. קראו היטב את ההוראות במסמך זה ובקוד שניתן לכם.
3. אחראי על התרגיל: **דניאל**. שאלות על התרגיל יש לשלוח למייל danytnt@campus עם הנושא: "HW5 236703". שאלות אדמיניסטרטיביות, כגון מילואים, יש לשלוח לנתן, תחת אותה כותרת.
4. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם. כמו כן הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שמימשתם כבר.
5. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
6. הגשה בזוגות בלבד.
7. בכדי להימנע מטעויות, אנא עיינו ברשימת ה-FAQ המתפרסמת באתר באופן שוטף.

מבוא

בתרגיל זה נתרגל את האפשרויות המתקדמות של C++. התרגיל מורכב משני חלקים **בלתי תלויים** המתמודדים עם נושאים שונים. בחלק הראשון נממש מנגנון החסר בשפת C++ והוא מנגנון Events בדומה למנגנון ב-C# והחלק השני יעסוק בשימוש במערכת ה-Template Meta-Programming של השפה לצורך בניית משחק, ש"ירוך" בזמן קומפילציה ויציג את התוצאה בעת הרצת התוכנית.

חלק א' – מימוש מנגנון דמוי Events ב-C++

מבוא:

בחלק זה של התרגיל נממש מנגנון המבוסס על תבנית התיכון [Observer Pattern](#). בתבנית זו אובייקט הנקרא **Subject** "מכיר" קבוצת אובייקטים אחרים שנקראים **Observers** והוא שולח **לכולם**, אחד אחרי השני, הודעות על שינויים המתרחשים בו. "שליחת ההודעות" מתבצעת ע"י קריאה לפונקציה מסוימת של ה-**Observers**.

התבנית באה להגדיר יחס תלות של רבים-אחד בין ה-**Observers** ל-**Subject**. אובייקט ה-**Subject** אינו תלוי ב-**Observers** שלו, אך הם כן. (תלויים בכך שנקראת פונקציה שלהם כאשר ה-**Subject** מחליט להודיע הודעה). בנוסף, תבנית זו תורמת **לאנקפסולציה** של מערכת – אבסטרקציה של רכיבי מערכת מרכזיים בתור **Subject** ורכיבים משתנים בתור **Observers**.

לדוגמה – ממשקי משתמש גרפיים (GUI) יוזמים טיפול מבוזר באירועים שתלויים במשתמש, כאשר המתכנת יכול להוסיף טיפול מותאם אישית להם – כך למשל באפליקציות אנדרואיד המתכנת יכול לכתוב פונקציית טיפול בנגיעה בכפתור, שתקרא ברגע שהמשתמש נוגע בו.

בתרגול של #C הכרתם/תכירו את מערכת ה-Delegates וה-Events. מערכת זו מהווה מימוש ל-Observer Pattern. (דוגמת חיישן הטמפרטורה והמזגנים)
בגרסה שלנו, נגדיר 2 מחלקות גנריות – **Observer** ו-**Subject**. הפרמטר הגנרי שלהן יהיה הטיפוס אותו מעביר ה-Subject בתור הודעה ל-Observers שלו. בנוסף המחלקה **Observer** תהיה וירטואלית טהורה – מחלקות שירשו ממנה יממשו את פונקציית הטיפול באירוע.
למעשה המחלקה **Observer** היא מעין **Interface** המצהיר על כוונת מחלקות היורשות ממנו להיות בעלות היכולת להירשם כ-Observer אצל **Subject** כלשהו.

המחלקה Subject (בקובץ Subject.h):

המחלקה הגנרית תקבל את T בתור הטיפוס הגנרי ותכיל בנאי חסר פרמטרים.
המחלקה תממש את המתודות הבאות:

- **void notify(const T&)** - מתודה המקבלת ארגומנט מהטיפוס הגנרי ושולח אותו כהודעה לכל ה-Observers אותם העצם מכיר. (קריאה למתודה `handleEvent()` שתוגדר בהמשך עבור המחלקה `Observer`). סדר הקריאה לכל ה-Observers יהיה לפי סדר ההוספה שלהם ל-`Subject`.
- **void addObserver(Observer<T>&)** – מתודה המקבלת עצם `Observer` ומוסיפה אותו לקבוצת האובייקטים אותם מקבל ההודעה מכיר. אם העצם כבר מוכר למקבל ההודעה יש לזרוק את החריגה `ObserverAlreadyKnownToSubject` המוגדרת בקובץ `OOP5EventExceptions.h`.
- **void removeObserver(Observer<T>&)** – מתודה המקבלת עצם `Observer` ומוחקת אותו מקבוצת האובייקטים אותם מקבל ההודעה מכיר. אם העצם אינו מוכר למקבל ההודעה יש לזרוק את החריגה `ObserverUnknownToSubject` המוגדרת בקובץ `OOP5EventExceptions.h`.
- **Subject<T>& operator+=(Observer<T>&)** – אופרטור += המקבל עצם `Observer` ומוסיף אותו לקבוצת העצמים המוכרים ל-`Subject` עליו הפעילו את האופרטור (תוך שימוש במתודת `addObserver`). יש להחזיר את עצם ה-`Subject` לאחר ההוספה.
- **Subject<T>& operator-=(Observer<T>&)** – אופרטור -= המקבל עצם `Observer` ומוחק אותו מקבוצת העצמים המוכרים ל-`Subject` עליו הפעילו את האופרטור (תוך שימוש במתודת `removeObserver`). יש להחזיר את עצם ה-`Subject` לאחר המחיקה.
- **Subject<T>& operator()(const T&)** – אופרטור () המקבל ארגומנט מהטיפוס הגנרי ושולח כהודעה לכל ה-Observers שהעצם מכיר (תוך שימוש במתודת `notify`). יש להחזיר את עצם ה-`Subject` לאחר השליחה.

המחלקה Observer (בקובץ Observer.h):

המחלקה הגנרית תקבל את T בתור הטיפוס הגנרי ותכיל בנאי חסר פרמטרים.
המחלקה תכיל את המתודה הוירטואלית הטהורה `&void handleEvent(const T` שתהיה המתודה הנקראת על ידי ה-Subject בקריאה למתודת `notify`. מתודה זו היא למעשה "מצביע הפונקציה" שמוסיפים ל-Delegate ב-C#.

דוגמת שימוש במנגנון בקובץ Part1Examples.cpp המצורף.

חלק ב' – Template Meta-Programming

מבוא:

כפי שראיתם בתרגול, מערכת ה-`Templates` ב-C++ היא **טיורינג שלמה**. למעשה, זה אומר שניתן לממש בעזרתה כל תכנית מחשב, כבר בזמן הקומפילציה. בחלק זה נדגים תכונה זו על ידי כתיבת גרסה משלנו למשחק **Rush Hour**.

מטרת חלק זה היא להכיר לעומק את מערכת התבניות ולהכיר את הכוח הרב הטמון בה. בחלק זה נשתמש בין היתר ב-`Template Specialization/Partial Specialization`, במילה השמורה `typename`, באופרטור `sizeof ...` ובמאקרו `static_assert`.

מומלץ מאוד לקרוא את כל הדרישות והטיפים לפני שמתחילים לחשוב על פתרון.



במשחק זה נתון לוח ריבועי מגודל 6x6 (בגרסה שלנו נוכל לקבוע גודל אחר) עם פתח יציאה בדופן הימנית. על הלוח מונחות מכוניות ומשאיות, כל אחת **בצבע אחר**. עליכם להזיז אותן כך שהמכונית האדומה תוכל לצאת מהלוח.

רכב יכול לנוע רק בכיוון בו הוא מונח (בתמונה המכונית האדומה יכולה לזוז אופקית בלבד והמכוניות הירוקות אנכית).



קישור לגרסת אונליין למשחק: [כאן](#) (תהנו!)

בגרסה שלנו המשתמש יכתוב לתוכנית את הלוח ואת סדרת המהלכים שלו ויקמפל אותה. **בזמן הקומפילציה** היא תבדוק את הפתרון המוצע ותאפשר **בזמן ריצה** גישה למידע האם סדרת המהלכים הובילה למצב בו למכונת האדומה יש את היכולת לצאת מהלוח.

את הלוח נייצג בתור רשימה של רשימות. כל רשימה תייצג שורה, וכל כניסה בה תייצג את סוג המשבצת – ריקה או עם מכונת.

מהלך מוגדר כוקטור (Type, Direction, Amount) – העבר את המכונת מסוג Type בכיוון Direction לאורך Amount משבצות, **והמשחק** יוגדר כסדרת מהלכים + לוח.

בחלק זה, כל האינדקסים מתחילים מ-0 (שורות ועמודות)

איך מכשילים את הקומפילציה?

בחלק זה של התרגיל נטפל בשגיאות קלט (כמו מהלכים לא חוקיים, או סדרת מהלכים שלא פותרת את המשחק) על ידי הכשלת תהליך הקומפילציה.

C++11 הוסיפה לסטנדרט של השפה את המקרו `static_assert` שפעולתו זהה לפעולת המקרו `assert`, אך עובד **בזמן קומפילציה**. ניתן להשתמש במקרו בכל חלק בקוד ובעזרתו נוכל לעצור את הקומפילציה במידת הצורך.

הגדרת המקרו היא `static_assert(bool_constexpr, message)`. אם `bool_constexpr` משוער בזמן קומפילציה לערך `false` הקומפילציה תיכשל והודעה `message` תוצג. דוגמת שימוש (פשוטה אך שימושית):

```
int main(){
    static_assert(sizeof(long) == 8, "Sorry, can't use non 64bit longs!");
    // rest of code
}
```

בכל מקום בחלק זה של התרגיל בו צריך להכשיל את הקומפילציה יש להשתמש במאקרו. הודעת השגיאה לא תיבדק ולכן אפשר לבחור הודעה אקראית (לצרכי דיבוג, מומלץ לתת הודעה אינפורמטיבית!).

הערה חשובה לפני שמתחילים

ב-IDE-ים רבים (אם לא בכולם) ה-Linter (הרכיב התוכנתי שאחראי על סימון שגיאות בקוד בזמן כתיבת הקוד, לפני ניסיון קומפילציה) לא מסוגל להתמודד עם מערכת התבניות של C++ בשימוש שהוא לא סטנדרטי, מכיוון שהוא אינו יכול לסמלץ את כל המגנונים של התבניות בזמן הריצה שלו מבלי לקמפל את התכנית, ולכן במקרים רבים הם יציגו לכם שגיאות שקשורות לתבניות, גם אם הקוד מצליח לעבור קומפילציה.

לכן, על מנת למנוע בלבול ועצבים (ושעות של דיבוג מיותר), מומלץ לעבוד בחלק זה עם עורך טקסט שלא מריץ Linter כברירת מחדל (כדוגמת Atom או Visual Studio Code). כמו כן, על מנת לוודא האם הקוד שלכם מתקמפל - יש להשתמש בשרת הטכנוני, שכן הטסטים ירוצו שם.

לשם ההתחלה, נגדיר מס' מבני עזר בשביל מימוש המשחק (כל מבנה מוגדר בקובץ המתאים):

List: (קובץ List.h)

המבנה הראשון שנגדיר הוא רשימה מקושרת של איברים גנריים. הרשימה תקבל טיפוסים בתור איבריה (ולא ערכים) ותכיל את השדות הבאים:

- head – איבר הראש של הרשימה
- next – טיפוס של רשימה מקושרת המכילה את שאר איברי הרשימה
- size – שדה מטיפוס int שיכיל את גודל הרשימה

דוגמת שימוש ברשימה: (הניחו שהטיפוס Int<N> הוגדר במקום אחר)

```
typedef List<Int<1>, Int<2>, Int<3>> list;
list::head; // = Int<1>
typedef typename list::next listTail; // = List<Int<2>, Int<3>>
list::size; // = 3
```

בנוסף להגדרת הרשימה עצמה נגדיר מספר מבני עזר לרשימה:

- **PrependList** – מבנה שיקבל **טיפוס ורשימה** (בסדר הזה) ויכיל את הטיפוס **list** שיהיה רשימה המכילה את הטיפוס שהועבר, ואחריו כל איברי הרשימה.

דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
typedef typename PrependList<Int<4>, list>::list newList; // = List<Int<4>,
Int<1>, Int<2>, Int<3>>
```

- **GetAtIndex** – מבנה שיקבל **מספר שלם N ורשימה** (בסדר הזה) ויכיל את הטיפוס **value** שיהיה האיבר שנמצא באינדקס ה-N ברשימה. (האינדקסים מתחילים מ-0). ניתן להניח שלא יועברו אינדקסים שחורגים מגודל הרשימה או אינדקסים שליליים.

דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
GetAtIndex<0, list>::value; // = Int<1>
GetAtIndex<2, list>::value; // = Int<3>
```

- **SetAtIndex** – מבנה שיקבל **מספר שלם N, טיפוס ורשימה** (בסדר הזה) ויכיל את הטיפוס **list** שיהיה רשימה הזוהר לרשימה שהועברה, פרט לאיבר ה-N שהוחלף באיבר שהועבר. (האינדקסים מתחילים מ-0)

ניתן להניח שלא יועברו אינדקסים שחורגים מגודל הרשימה או אינדקסים שליליים.

דוגמת שימוש במבנה:

```
typedef List<Int<1>, Int<2>, Int<3>> list;
typedef typename SetAtIndex<0, Int<5>, list>::list listA; // = List<Int<5>,
Int<2>, Int<3>>
typedef typename SetAtIndex<2, Int<7>, list>::list listA; // = List<Int<1>,
Int<2>, Int<7>>
```

רמז למימוש: חישוב כיצד ניתן להשתמש ב-PrependList.

Conditional: (קובץ Utilities.h)

תחת סעיף זה נגדיר 2 מבנים – Conditional ו-ConditionalInteger:

שני המבנים יהיו למעשה מימוש בזמן קומפילציה למבנה הבקרה if-else.

המבנים יקבלו ביטוי בוליאני, ו-2 תוצאות, אחת עבור ביטוי בוליאני false ואחת עבור ביטוי בוליאני true. המבנים יכילו את השדה/הטיפוס value שיהיה התוצאה המתאימה.

חתימות המבנים יהיו:

- `Conditional<bool, typename, typename>`: המבנה המתאים לטיפוסים (ולא לערכים מספריים). הטיפוס הראשון יהיה טיפוס התוצאה (value) אם הפרמטר הראשון משוערך ל-true והשני יהיה טיפוס התוצאה אם הפרמטר הראשון משוערך ל-false.
- `ConditionalInteger<bool, int, int>`: מבנה זה זהה בהגדרתו למבנה Conditional, אך ערכי התוצאה יהיו מספרים שלמים ולא טיפוסים.

דוגמת שימוש במבנים (הניחו שהטיפוס `Int<N>` הוגדר במקום אחר):

```
int val = ConditionalInteger<(0 != 1), 0, 1>::value; // = 0
val = ConditionalInteger<(0 == 1), 0, 1>::value; // = 1

typedef typename Conditional<(0 != 1), Int<0>, Int<1>>::value test1; // = Int<0>
typedef typename Conditional<(0 == 1), Int<0>, Int<1>>::value test2; // = Int<1>
```

אחרי החימום, נעבור למימוש רכיבי המשחק העיקריים.

כל תא בלוח המשחק ייוצג בעזרת 3 מאפיינים – סוג התא, אורך המכונית הנמצאת עליו וכיוונה. (עבור תא ריק אורך המכונית וכיוונה ייבחרו שרירותית)

כך למשל 2 תאים עליהם נמצאת מכונית אדומה (שסימונה במשחק X) מאורך 2 וכיוון תנועה מעלה יתוארו כ-(UP, 2) ותא ריק יתואר כ-(EMPTY, LEFT, 0). (הכיוון והגודל של התא הריק נבחרו שרירותית)

CellType: (קובץ CellType.h)

על מנת לייצג סוגי תאים נגדיר את ה-enum CellType.

ערכי ה-enum יהיו EMPTY ושאר הערכים בהתאם למפתח בתמונה:



(להעתקה מהירה ונוחה: X, A, B, C, D, E, F, G, H, I, J, K, O, P, Q, R)

Direction: (קובץ Direction.h)

על מנת לייצג כיוונים במשחק נגדיר את ה-enum Direction.

ערכי ה-enum יהיו:

- UP = 0 •
- DOWN = 1 •
- LEFT = 2 •
- RIGHT = 3 •

השימוש ב-Direction יהיה בתאי הלוח (כיווני המכוניות המונחות על תאים) ובמהלכי המשחק.

בפועל, מכיוון שנשתמש ב-enum גם בשביל הגדרת מהלכים, בהם צריך 4 כיוונים של תנועה, בשימוש בו **בהגדרת תאים בלוח** הכיוונים UP ו-DOWN שקולים לתנועה אנכית והכיוונים LEFT ו-RIGHT שקולים לתנועה אופקית.

BoardCell: (קובץ BoardCell.h)

כעת נוכל להגדיר את המבנה שייצג תא בלוח.

המבנה **BoardCell** יקבל את סוג התא, כיוון המכונית עליו ואת אורך המכונית **בסדר זה** (שימו לב שמכונית אחת יכולה להיות ממוקמת על כמה תאים צמודים).

המבנה יכיל את השדות הבאים:

- **type** – יכיל את סוג התא
- **direction** – יכיל את כיוון המכונית שנמצאת על התא
- **length** – יכיל את גודל המכונית שנמצאת על התא

GameBoard: (קובץ GameBoard.h)

מבנה זה ייצג את לוח המשחק.

המבנה יקבל רשימה של רשימות של **BoardCell**.

המבנה יכיל את השדות הבאים:

- **board** – יכיל את הרשימה הראשית.
- **width** – יכיל את הרוחב של הלוח (כמות העמודות)
- **length** – יכיל את האורך של הלוח (כמות השורות)

ניתן להניח שתמיד יועברו רשימות תקינות למבנה.

תנועת מכוניות – MoveVehicle והמבנה Move: (קובץ MoveVehicle.h)

בשביל לממש את מנגנון הזזת המכוניות על הלוח נממש מספר מבנים.

:Move

מבנה זה ייצג מהלך משחק. כפי שהוגדר, מהלך הוא וקטור (Type, Direction, Amount).

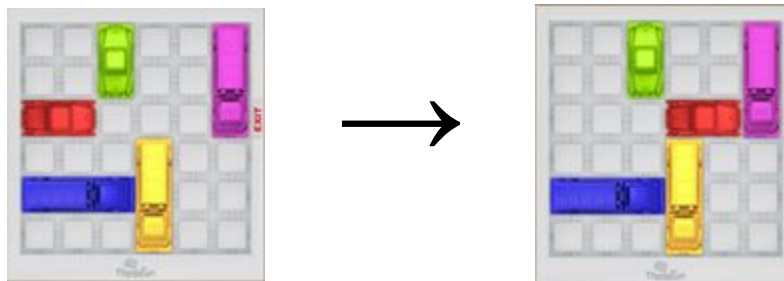
המבנה יקבל את סוג המכונית שרוצים להזיז, כיוון התנועה המבוקש ואת כמות המשבצות שיש להזיז אותה בכיוון (בסדר הזה).

המבנה יכיל את השדות הבאים:

- **type** – יכיל את סוג המכונית שרוצים להזיז
- **direction** – יכיל את כיוון התנועה המבוקש
- **amount** – יכיל את כמות המשבצות שיש להזיז את המכונית בכיוון

בזמן יצירת המבנה יש לוודא כי סוג התא המועבר שונה מ-EMPTY (כי לא ניתן להזיז EMPTY). אם מועבר סוג תא של משבצת ריקה יש להכשיל את הקומפילציה.

כדוגמת שימוש במבנה, המעבר הבא:



מתואר על ידי `Move<X, RIGHT, 3>`.

תהליך הקומפילציה אמור להיכשל עבור קטע הקוד הבא:

```
int amount = Move<EMPTY, UP, 1>::amount;
```

ואילו קטע הקוד הבא אמור לעבור קומפילציה בהצלחה:

```
int amount = Move<X, RIGHT, 1>::amount;
```

:MoveVehicle

המבנה המורכב והעיקרי במשחק. מבנה זה יקבל לוח (מטיפוס `GameBoard`), אינדקס שורה `R`, אינדקס עמודה `C`, כיוון `D` ומספר שלם `A` (בסדר הזה) ויחזיק טיפוס יחיד `board` (של `GameBoard`) שיהיה הלוח, **אחרי הזזת המכונית** שנמצאת על התא בשורה `R` ועמודה `C`, בכיוון `D`, משבצות, כאשר האינדקסים $(0,0)$ מציינים את הפינה השמאלית עליונה בלוח. יש לשים לב לכך שיש להזיז את **כל** המכונית, ולא רק את התא שמועבר.

לדוגמה, בהינתן המבנה `MoveVehicle<gameBoard, 2, 1, RIGHT, 3>` והטיפוס `gameBoard` שמייצג את הלוח הבא:



הטיפוס `board` של המבנה צריך לייצג את הלוח:



הערה: עבור הלוח בדוגמה לעיל המבנה `MoveVehicle<gameboard, 2, 0, RIGHT, 3>` מוביל לאותה תוצאה!

ניתן להניח שלא ייבדקו מהלכים שמוציאים מכונית מהלוח.

במקרים הבאים יש להכשיל את הקומפילציה:

- מועברות קואורדינטות (R,C) שלא נמצאות על הלוח
- מועברות קואורדינטות של משבצת ריקה (`type = EMPTY`)
- מועברות קואורדינטות של מכונית שלא יכולה לנוע בכיוון שהועבר
- המהלך יגרום למכונית לעבור בתא בו יש מכונית אחרת (מכונית יכולה לזוז רק על תאים מסוג `EMPTY`)

רמזים למימוש:

- הפרידו בטיפול בין תנועה אופקית לבין תנועה אנכית.
- השתמשו במבנה `Transpose` שמקבל מטריצה (רשימה של רשימות) ומחזיק טיפוס `transposed` שהוא המטריצה המשוחלפת. המבנה נמצא בקובץ המצורף `TransposeList.h`, חישבו כיצד לנצל את המבנה בשביל לחסוך קוד.
- ממשו מבנה עזר שמזיז מכונית **משבצת אחת בכל פעם**. נצלו את העובדה שכל תאי הלוח שמציינים את אותה מכונית זהים מבחינת שדות.

מבנה המשחק `CheckSolution` והמבנה `CheckWin`: (קובץ `RushHour.h`)

לסיום המשחק נגדיר 2 מבנים אחרונים: `CheckSolution` ו-`CheckWin`.

`CheckWin`:

מבנה זה יקבל את הלוח המשחק (כ-`GameBoard`) ויבדוק אם המשחק פתור. המשחק מוגדר כפתור אמ"מ המכונית האדומה יכולה לנוע בחופשיות (במהלך חוקי) לעבר פתח היציאה. (כלומר הדרך ריקה ממכוניות)

כך לדוגמה הלוח הבא פתור:



והלוח הבא **לא** פתור:



המבנה יכיל את השדה הבוליאני `result` שיהיה שווה ל-`true` אם המשחק פתור ול-`false` אחרת.

CheckSolution:

השימוש במשחק יתבצע דרך מבנה זה.

המבנה מקבל את לוח המשחק (כ-GameBoard) וסדרת מהלכים (כרשימה של Move) (בסדר הזה) ויכיל שדה בוליאני result שערכו יהיה true אם סדרת המהלכים פותרת את המשחק ו-false אחרת.

שימו לב, יש לבצע את כל המהלכים ורק אז לבדוק אם המשחק פתור (בעזרת CheckWin)

הערה: שימו לב שהמבנה Move מקבל את סוג התא ולא את מיקומו והמבנה MoveVehicle מקבל את מיקום המכונית. **מומלץ לממש מבנה עזר** שמקבל את הלוח ואת סוג התא ומוצא את מיקום המכונית על הלוח. (בשביל נוחות – קבעו שהמבנה מחזיר את התא הראשון או האחרון של המכונית על הלוח, כאשר התא הראשון יהיה מוגדר להיות התא השמאלי-עליון ביותר, לדוגמה). **מבנה זה אינו חובה ולא ייבדק.**

דוגמת שימוש במשחק:

```
typedef GameBoard<List< List< BoardCell<EMPTY, UP, 0>, BoardCell<EMPTY, UP, 0>>,
                    List< BoardCell<X, RIGHT, 1>, BoardCell<A, UP, 1>>,
                    List< BoardCell<EMPTY, UP, 0>, BoardCell<EMPTY, UP, 0>>>> gameBoard;
typedef List<Move<A, UP, 1>> moves;
typedef CheckSolution<gameBoard, moves> gameState;

static_assert(gameState::result, "Fail!"); // gameState::result should be true
```

הלוח בדוגמה הזו הוא הלוח:



הקוד לעיל אמור לעבור קומפילציה בהצלחה.

יש לדאוג לכך שמהקובץ `RushHour.h` תהיה גישה **לכל** פונקציונליות המוגדרת בחלק זה (כלומר יש לעשות לכל קובץ שמימשתם + הקבצים שהוגדרו בתרגיל).

בקובץ `Part2Examples.cpp` מצורפות דוגמאות שימוש במבנים ובמשחק, עברו עליהן וודאו כי הפתרון שלכם עובר קומפילציה!

לנוחיותכם, מצורף הקובץ `Printer.h` בו מוגדר הטיפוס `Printer` שמכיל את הפונקציה `print`. הפונקציה מקבלת ערוץ הדפסה (למשל `std::cout`) ומדפיסה אליו את תוכן המבנה שהטיפוס מקבל כפרמטר גנרי. דוגמת שימוש ב-`Printer`:

```
typedef GameBoard<List< List< BoardCell<EMPTY, UP, 0>, BoardCell<EMPTY, UP, 0>>,
                    List< BoardCell<X, RIGHT, 1>, BoardCell<A, UP, 1>>,
                    List< BoardCell<EMPTY, UP, 0>, BoardCell<EMPTY, UP, 0>>>> gameBoard;
Printer<gameBoard>::print(std::cout);
```

ניתן להעביר ל-`Printer` את הטיפוסים `List`, `BoardCell` ו-`GameBoard`.

חזרו על התרגול של תבניות ב-C++ והבינו את הנושא היטב. עברו על המבנים המצורפים לתרגיל (Printer ו-Transpose), הבינו איך (ולמה) הם עובדים. הבנה טובה של הנושא תעזור מאוד במימוש התרגיל.

הערות והנחות:

- בכל מקום בו כתוב מבנה הכוונה היא ל-`struct`.
- פתח היציאה מהלוח תמיד יהיה בשורה של המכונת האדומה, ותמיד בצד הימני של הלוח (המכונת האדומה יכולה לנוע רק בתנועה אופקית).
- ניתן ואף רצוי להוסיף שדות, מבנים וטיפוסים נוספים.
- מומלץ להקפיד על חלוקה לקבצים שונים לפי קטגוריות. יש לוודא כי קובץ ה-Header הראשי (`RushHour.h`) יכלול `include` לכל קובץ Header שיצרתם. חלק מהבדיקות בחלק זה של התרגיל יכלול `include` יחיד והוא ל-`RushHour.h`.

הערות לשני החלקים של התרגיל:

- בדיקת התרגיל תכלול הן את הפונקציונליות של המערכת כולה (אינטגרציה של כל הרכיבים) והן את הפונקציונליות של כל רכיב בפני עצמו (בדיקות יחידה). ודאו שכל רכיב עומד בדרישות התרגיל! לשם כך, מומלץ מאוד לכתוב ולהריץ בדיקות!
- עליכם לדאוג לשחרר את כל האובייקטים שהקצתם ב-2 חלקי התרגיל!
- הקוד יקומפל בעזרת הפקודה:

`g++ -std=c++11 *.cpp`

יש להריץ/לקמפל את העבודה כולה על שרת ה-csl3, שכן העבודה שלכם תבדק עליו.

על שרתי ה-csl3, לפעמים מותקנת גרסה ישנה של gcc אשר אינה תומכת ב C++11. על מנת לעדכן את הקומפיילר שלכם פעלו באופן הבא:

1. `gcc --version` # If GCC is 4.7 or above, stop here.
2. `bash`
3. `./usr/local/gcc4.7/setup.sh`
4. `cd ~`
`echo ./usr/local/gcc4.7/setup.sh >> .bashrc` # This makes the change permanent

הוראות הגשה

- בקשות לדחייה, מכל סיבה שהיא, יש לשלוח למתרגל האחראי על הקורס (נתן) במייל בלבד תחת הכותרת HW5 236703. שימו לב שבקורס יש מדיניות איחורים, כלומר ניתן להגיש באיחור גם בלי אישור דחייה – פרטים באתר הקורס תחת General info.
- הגשת התרגיל תתבצע אלקטרונית בלבד (יש לשמור את אישור השליחה!).
- יש להגיש קובץ בשם `OOP5_<ID1>_<ID2>.zip` המכיל:
 - קובץ בשם `readme.txt` המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים בפורמט הבא:

Name1 id1 email1

Name2 id2 email2

- תיקייה בשם `part1` שתכיל את כל הקבצים **שמימשתם** בחלק הראשון של התרגיל + הקובץ `OOP5EventException.h` שסופק לכם. (לא כולל `Part1Examples.cpp`)
- תיקייה בשם `part2` שתכיל את כל הקבצים **שמימשתם** בחלק השני של התרגיל + הקובץ `TransposeList.h` שסופק לכם. (לא כולל `Part2Examples.cpp`)

```
OOP5_123456789_112233445.zip/
|_ readme.txt
|_ part1/
|   |_ OOP5EventException.h
|   |_ Subject.h
|   |_ Observer.h
|   |_ additional files...

|_ part2/
|   |_ BoardCell.h
|   |_ CellType.h
|   |_ Direction.h
|   |_ GameBoard.h
|   |_ MoveVehicle.h
|   |_ RushHour.h
|   |_ TransposeList.h
|   |_ additional files...
```

- נקודות יורדו למי שלא יעמוד בדרישות ההגשה (`rar` במקום `zip`, קבצים מיותרים נוספים, `readme` בעל שם לא נכון וכו')

בהצלחה!

