EE 441 PROGRAMMING ASSIGNMENT #1: Classes, Stacks, Queues and Linked Lists

Due Date: 20.11.2020 at 23.59
Form of Delivery: You should upload a single zip file with your solution on odtuclass. The filename should be "EE441_PA1_firstname_lastname_studentID.zip". Also indicate how much time you spent for the homework (see Regulations).

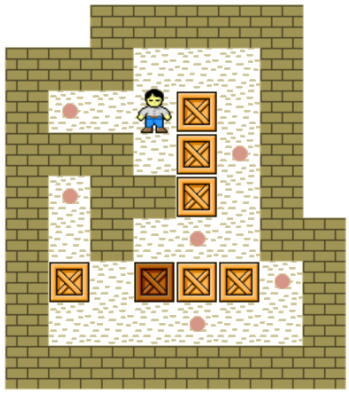


Figure 1: Sokoban Game

Sokoban, which means 'warehouse keeper' in Japanese, is a transport puzzle game. In this game, the player tries to put boxes at designated locations by pushing them around a maze. The rules of this game are[1]:

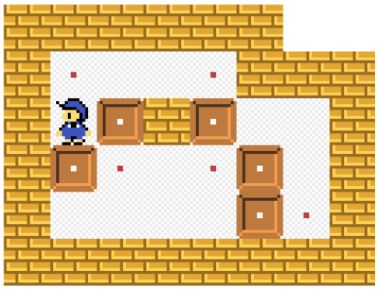
- Only one box can be pushed at a time
- Boxes cannot be pulled. Only pushing is allowed.
- The player cannot walk through boxes or walls.
- The number of the boxes is equal to the number of target locations.
- The puzzle is solved when all boxes are on the target locations.

In this homework, we are interested in the puzzles of size 6*8 blocks. For console applications, a text-based user interface can be designed. The code holds a char array for the cells, and uses different ASCII characters for different cell states, as follows:

- '@' : the player
- '.' : target location
- '+' : the player at a target location
- ' ' (space) : empty cell
- '$' : movable box
- '*' : movable box at target location
- '#' : wall

[1] https://www.sokobanonline.com/help/how-to-play

An example puzzle and the corresponding representation is given in Fig.2.



```
######
#.  .###
#@$#$  #
#$. .$ #
#    $.#
########
```

Figure 2: Initial puzzle state and its text-based representation

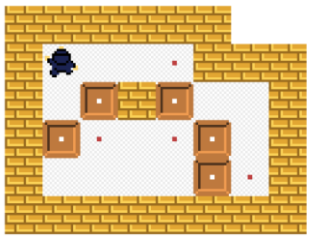
Q1. Implement a class `Sokoban` that represents a Sokoban puzzle. Your class must have at least the following methods:

- A constructor that initializes the class from a char array, a copy constructor and copy assignment operator.
- A constructor that initializes the class from a text file. **Hint:** You may use the code segment in the appendix to read the sample file into a char array.
- `move_up, move_down, move_left, move_right` methods. The return type of these methods should be `bool`. If the move operation is invalid according to the rules of Sokoban, the method returns false, otherwise true.
- `is_solved` method. This method must return true if the puzzle is solved, otherwise false.
- `print_puzzle` function, that prints the text-based representation of the current state of the puzzle.

Some examples are given as follows. Assume that we start from the state given above. In this state, the `move_left` command returns false, since the wall cannot be moved, and no change occurs in the puzzle state. Similarly, the `move_right` command returns false because there is a wall behind the box. However, the `move_up` command returns true. The new game state is given in Fig.3. Note that the player is at a target location, hence is shown as '+'.

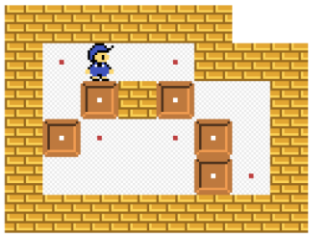
Now, the `move_right` command returns true, and the new game state is given in Fig.4.
`Then, move_down` command returns true. The new game state is given in Fig.5.
Another `move_down` also returns true. New game state is given in Fig.6.

Q2. Implement a mixed `StackQueue` template class. Your class must have a statically allocated member that stores elements of the template parameter type. In addition, the class must have the methods `push_front`, `pop_front` and `pop_rear` that store a new element at the front of the storage, removes and returns the element at the front of the storage and at the rear of the storage, respectively. Your implementation should be able to store at least 100 elements.



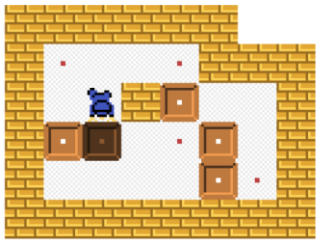
```
######
#+  .###
# $#$  #
#$. .$ #
#    $.#
########
```

Figure 3: Puzzle state after 'move_up' command



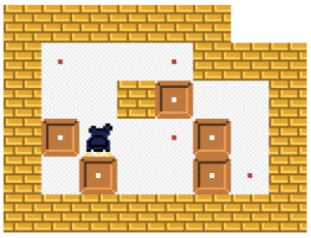
```
######
#.@ .###
# $#$  #
#$. .$ #
#    $.#
########
```

Figure 4: Puzzle state after 'move_right' command



```
######
#.  .###
# @#$  #
#$* .$ #
#    $.#
########
```

Figure 5: Puzzle state after 'move_down' command



```
######
#.  .###
#  #$  #
#$+ .$ #
# $  $.#
########
```

Figure 6: Puzzle state after 'move_down' command

Q3. In your main function, instantiate a `Sokoban` class and a `StackQueue` class with the template argument `Sokoban`. You can use the puzzle in Q1 as the initial state. Write a code to respond to key press events as follows:

- w, a, s, d keys should move the player to up, left, down and right, respectively. If the move is valid, the puzzle state is stored in the stack-queue, and printed to the console.
- z key undoes the last valid move. The last puzzle state is also removed from the queue-stack.
- r key replays all the moves starting from the initial state. All puzzle states are to be printed to the console in order.

Q4. Implement a doubly-linked list class. Replace the storage element in your `StackQueue` class with this doubly-linked list. **Use a separate file for this purpose, do not delete your first implementation**. Now, your stack-queue class can hold an unlimited number of elements. Check that your code in Q3 works fine with this new stack-queue implementation.

Regulations:

- You should insert comments to your source code at appropriate places without including any unnecessary detail. **Comments WILL be graded.**
- Submit the whole Code::Blocks project folder in a zip file. The file name should be "EE441_PA1_firstname_lastname_studentID.zip"
- Indicate how much time you spent for the homework in a comment at the top of your main.cpp
- Late submissions are welcome, but penalized according to the following policy:
  - 1 day late submission: HW will be evaluated out of 70 points
  - 2 days late submission: HW will be evaluated out of 50 points
  - 3 days late submission: HW will be evaluated out of 30 points
  - 4 or more days late submission: HW will not be evaluated

APPENDIX: LOAD A CHAR ARRAY FROM FILE

```
/* file stream library  */
/* required for file IO */
#include <iostream>
#include <fstream>
using namespace std;


/* code segment to read the      */
/* sample file into a char array */

/* file path                     */
/* you can use relative path if  */
/* the file is in project folder */
string path = "sample_puzzle.txt";

/* contents of the file will be  */
/* stored in this array          */
char data[6][8];
/* newline character will be     */
/* read in dummy                 */
char dummy;

/* input file stream */
ifstream file;
file.open(path);

for(int i=0; i<6; ++i){
  for(int j=0; j<8; ++j){
    /* read a character like cin */
    /* noskipws is required for  */
    /* reading whitespaces       */
    file >> noskipws >> data[i][j];
  }

  /* read the newline character  */
  /* at the end of each line     */
  file >> noskipws >> dummy;
}

file.close();
```