## Gebze Technical University Department of Computer Engineering CSE 241/505

## Object Oriented Programming Fall 2019

## Homework # 4 N-Puzzle Continued Due date Nov 11<sup>th</sup> 2019

In this homework, you will make additions to your N-Puzzle program. We will write an exhaustive search algorithm that will find the best solution even if it takes too long to do so.

Your main class will be named NPuzzle. It will have the following public functions, most of which

are the same as HW3 (changes are marked with bold letters)

<b>Fuction Name</b>	Explanation
print	Prints the current configuration on the screen by sending it to cout
printReport	Prints a report about how many moves have been done since reset and
	if the solution is found
readFromFile	Reads the current configuration from the file given as function
	parameter. The file format is defined as in HW2. <b>The vector of Board</b>
	objects is resized to 1. See below for details.
writeToFile	Writes the current configuration to the file given as function
	parameter.
shuffle	Makes N random moves to shuffle the board. N is given as a function
	parameter. The vector of Board objects is resized to 1. See below
	for details.
reset	Resets the current configuration to the solution. <b>The vector of Board</b>
	objects is resized to 0.
setsize	Sets the board size to given values. The values are given as parameters
	and they can be at most 9x9. After setting the size, the boards should
	be reset.
moveRandom	Makes a valid random move. Move is recorded in vector of Board
	objects. The vector of Board objects is resized to 1. See below for
	details.
move	Makes a move according to the given char parameter. If the
	parameters is 'L' then, the blank tiles moves left,, etc, as defined in
	HW1. Move is recorded in vector of Board objects. The vector of
	Board objects is resized to 1. See below for details.
solvePuzzle	Runs the algorithm described below to solve the problem in
	optimal number of steps.
Operator>>	Prints the current configuration on the screen by sending it to
	ostream object
Operator<<	Reads the current configuration from the istream object. The file
	format is defined as in HW2.

Your **NPuzzle** class defines and uses a private inner class named **Board**, which represents the board configuration using a private **2D Vector**. The class also has two other data members

- The last move that it has performed
- The number of moves make on this board from the beginning.

Your **NPuzzle** class does not use any other data to represent the board. This class defines the following functions, most of which are the same as HW3 (changes are marked with bold letters)

<b>Fuction Name</b>	Explanation
print	Prints the board on the screen by sending it to cout
readFromFile	Reads the board from the file given as function parameter. The file
	format is defined as in HW2.
writeToFile	Writes the board to the file given as function parameter
reset	Resets the board to the solution.
setSize	Sets the board size to given values. The values are given as parameters
	and they can be at most 9x9. After setting the size, the boards should
	be reset.
move	Makes a move according to the given char parameter. If the parameter
	is 'L' then the blank tiles moves left,, etc, as defined in HW1.
isSolved	Returns true if the board is a solution
Operator()	Takes two indexes and returns the corresponding cell content.
	Terminates program if the indexes are not valid.
Operator==	Two boards are equal, if the boards are the same. This operator
	does not consider last move or the number of steps
NumberOfBoards	Returns the number of Board objects created so far.
lastMove	Returns the last move, if there is no last move, returns 'S'
numberOfMoves	Returns the number of steps (moves) this board made

Your program will use objects of **NPuzzle** to perform what we did previously in HW3. Your command line options and your user interface is very similar. The following table is repeated here just for convenience.

Input	Action
V	Solves the problem from the current
	configuration using the new algorithm.
T	Prints a report about how many moves have been
	done and if the solution is found
Е	Asks a file name and saves the current board
	configuration as a loadable shape file.
О	Asks a file name and loads the current board
	configuration from the shape file.
L	moves the empty cell left if there is room
R	moves the empty cell right if there is room
U	moves the empty cell up if there is room
D	moves the empty cell down if there is room
S	Shuffle- takes the board to the final solution, and
	makes size*size random moves to shuffle the board.

Your new algorithm follows these steps

- 1. Your **NPuzzle** class keeps a vector of **Board** objects. Initially this vector contains a single object that contains either the board from a file or shuffled board. The number of steps of this board is 0.
- 2. Take the first element from this vector,
- 3. Apply all possible moves to this board and **push\_back** each result back to the vector. Before doing the push back, you should check if the same board is already in the vector. If one of the pushed objects is the solution, then your solution is found.
- 4. Take another element from the vector in order and go to step 3.

Note that the above algorithm can stop in two cases: solution is found or there are no elements to take from the vector. We expect that if there is a solution, we will find it. In order to demonstrate the algorithm better, assume the initial 3x3 board vector in step 1 contains

```
1 2 3
4 5
7 8 6
```

Which will be represented as { ((1234b5789), S, 0)}, where S is the last move, 0 is the number of moves. Step 2 takes this board and pushes back 4 new boards to the vector

```
\{((1234b5786),S,0),((12345b786),R,1),((123b45786),L,1),((1b3425786),U,1),((1234857b6),D,1)\}
```

Note that bold letters indicate the vector elements that are not chosen from the vector yet. Since, we did not see the solution, we will pick the next element from the vector and go to step 3, which yields

```
{((1234b5786),S,0), ((12345b786),R,1), ((123b45786),L,1), ((1b3425786),U,1), ((1234857b6),D,1), ((12b453786),U,2), ((12345678b),D,2)}
```

We see that one of the newly added elements is the solution. We terminate the algorithm. From the solution, we can go back to the original board by applying the inverse moves U and L. We see that the solution needs just two moves.

## Notes:

- Do not use any C++ features that we did not learn during the lectures. Use all the OOPL rules we learned in the class.
- Do not forget to indent your code and provide comments.
- Check the validity of the user input.
- <u>Test your programs very carefully at least with 10 different runs and submit your result files for each.</u>
- You should submit your work to the moodle page and follow all the submission rules that will be posted.