School of Electrical and Computer Engineering
CptS223: Advanced Data Structures C/C++
Spring 2018
Homework #5 (Programming Assignment)
Due: Wednesday 03/07/2018 @ 11:59pm

Instructions: All source code must be in C++. You are required to use the Unix environment. You submission should accompany a report that explains your work. Your code will be tested on EECS servers.

For this programming assignment, you have the option of either working as a TEAM if you prefer, or as an individual. A TEAM should contain at most 2 students including yourself. Grading will NOT differentiate between the two - i.e., if you are working as a team of two people, then the same grade will be applied to both of you.

If working as a team, both of you should submit the same package. However, you will need to clearly mention team members names in your report. If working as a team, also remember to include both your last names in the zip file of your submission.

In the rest of this document, the term "YOU" is used either to refer to you (if you are working individually) or your team (if you work as a team).

The final material for submission should be entirely written by you. If you decide to consult with others or refer materials online, you MUST give due credits to these sources (people, books, webpages, etc.) by listing them in the report that accompanies your submission. Note that no points will be deducted for referencing these sources. However, your discussion/consultation should be limited to the initial design level. Sharing or even showing your source code/assignment solution verbiage to anyone else in the class, or direct reproduction of source code/verbiage from online resources, will all be considered plagiarism, and will therefore be awarded ZERO points and subject to the WSU Academic Dishonesty policy. (Reproducing from the Weiss textbook is an exception to this rule, and such reproduction is encouraged wherever possible.)
Grading will be based on correctness, coding style, implementation efficiency, exception handling, source code documentation, and the written report.

All assignments files (code and report) should be zipped/tarred into one archive folder (named after your last name), and submitted through WSU Blackboard by the submission deadline.

Note that submissions through any other means for example by emailing your package to the instructor/TA/TAs will be discarded and will NOT be graded. So please follow the above instructions carefully.

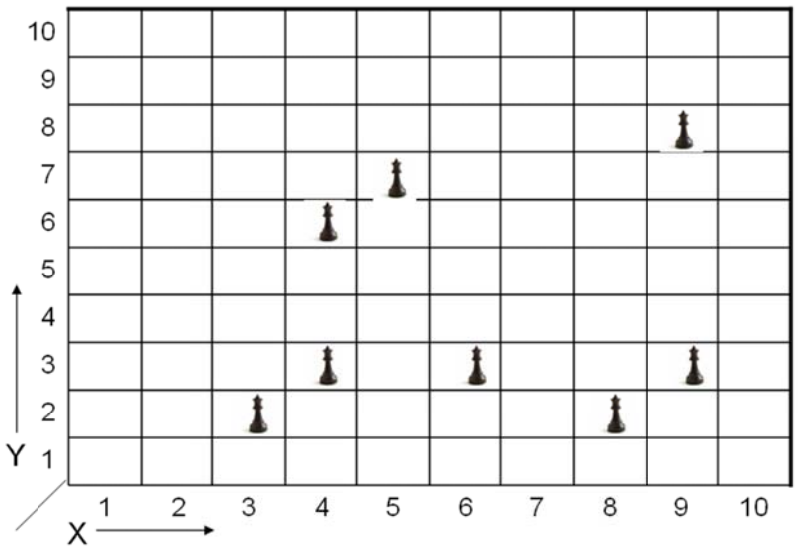
Late submission policy: A late penalty of 10% will be assessed for late submissions within the next 24 hours (i.e., until midnight of the next day after the submission deadline). After this one-time late submission, no submissions will be accepted.

(1) ASSIGNMENT:

For this assignment, you will be implementing a simple board game, as per the specifications stated below, using any (or more) of the data structures we have discussed in class so far (e.g., list, vector, balanced binary search trees (STL: set, map, multiset, multimap)) as appropriate.

Specifications:

A "board" is defined as a $m \times m$ square matrix, with each cell identified using a unique (x,y) coordinate. An example 10 × 10 board is shown in the figure below. In a game, a board has '*n*' players, where n ≤ m always. The figure shows an example for m=10 and n=8.



Your task is to implement the following specifications and functionalities:

- Write a Board class that implements a board of size $m \times m$ with n players. The value of m is user specified (during class construction). At the start of the game, all the $m \times m$ cells of the board are empty (i.e., n=0). As the game progresses '*n*' is allowed to change. Read the rest of the problem carefully and understand the requirements before deciding on how to implement this class.

- Write a Player class that implements each player. Each player object will have a unique, positive integer ID, and at any given time the player will occupy a unique (x,y) position on the board. Care must be taken that the value of (x,y) position is always within bounds of the board at all times. No two players are allowed to share the same cell.

- Implement an Insert method in the Board class that will allow you to add a new player to the board. The method should take as input the player ID to be inserted, along with a desired (x,y) position into

which it is to be initially placed. For successful insertion, (i) a player with the same ID should not already exist on the board; AND (ii) the specified insertion position on board should be currently unoccupied. A third condition would be to ensure the total number of players with this insertion (i.e., n) should not exceed m. If successful, the method should update n and return true. If insertion fails, the code should display an appropriate error message and return false without changing anything.

- Implement a Remove method in the Board class that will allow you to remove a player from the board. The method should take as input the player ID to be removed, and should return true upon successful removal and false otherwise (i.e., player not found). The value of n should also be accordingly updated. Note that upon successful removal, the corresponding cell on the board should become available for newer insertions.

- Implement a Find method in the Board class that is given a player ID and returns true if the player is found and false otherwise.

- Implement a MoveTo method in the Board class that takes as input a player ID and a destination (x2,y2) cell position. The method should first locate the player, and move the player from its current position, say (x1,y1), to the new position (x2,y2) only if:
  - (x2,y2) is within bounds, AND
  - the movement from (x1,y1) to (x2,y2) is always along the same row or same column or same diagonal in any direction (i.e., top-left -- right-bottom or top-right -- left-bottom).

  Additional action is necessary if the destination cell (x2,y2) already has some other player, say Y. Then this function should first remove Y from the board before placing this player (ID) in its new position. Upon a successful move, the method should return true. If any player was removed in the process, the method should print a message to indicate which player was removed. If the move fails, the code should display an error message stating the reason of failure and return false.

  Note: Any player(s) on the board along the path of moving from (x1,y1) to (x2,y2) is/are left unaffected by this move.

- Implement a **PrintByID** method in the Board class that prints all the player IDs along with their (x,y) positions, in the increasing order of their IDs. Again, the print should *not* display any unoccupied positions.

Design:
It is expected that you use balanced binary search trees for this project (STL set, map, multiset, multimap). You are also, in addition, free to pick other STL containers as appropriate. Your choice should be based on what is expected to give the "best" performance (both in terms of time and memory) assuming the following:

- Although we will be testing for smaller values of n and m, your design should be optimized for the general use-case where the value of n is expected to be much smaller than the value of m. (For example, n=100, m=1,000,000). This simply implies that your code should not store the whole m*m board because it could run out of memory. Alternatively, think of storing only the occupied positions and the players.

- You can assume that the number of players sharing the same x-value is going to be a small constant.

- You can assume that for a given game, the value of m never changes. Of course the value of n could change dynamically as players are inserted and removed as the game progresses.

In trying to meet the above guidelines, you are permitted to maintain multiple data structures and/or copies of the same data in different containers as you deem necessary for design and efficiency. You are free to define new member functions not specified above to enhance code reuse and modularity.

Testing:
For testing, use the *TestBoard.cpp* code which is provided as part of this assignment. This code is for the example board shown above. The code is also available at the following address:

http://epsl.eecs.wsu.edu/wp-content/uploads/2018/02/TestBoard.CPP_.txt

Save the output of your testing as a text file and submit it as part of your report.

(2) REPORT:

In a separate document (Word or PDF), compile the following sections:

A: Problem Statement
In a couple of sentences state the goal(s) of this exercise.

B: Experimental Setup
In this section, specify the machine architectural details where the testing was conducted. Also mention whether you used Windows or Unix or Mac OSX for your testing.

C: Algorithm Design (limit to 2 pages)
Provide a design document for your code with the following information. State what data container(s) you used to support the different functions and why; what is the run-time complexity of each of the functions in your code; and what is the memory complexity for your implementation as a whole. Figures and illustrations can be used to show the design.

D: Test Results
Save the output of your testing using the *TestBoard.cpp* and attach it as a *text* file.

(3) GRADING:

This assignment is mainly about designing efficient algorithms and implementations for the methods specified in the Board class. There is not really an extensive empirical component that involves timing and such but you need to pay attention to design in a manner that is expected to keep the runtimes really low for even large input cases. So for grading this PA, we will primarily look at how well you have designed your algorithm behind the implementation, how well you have implemented the code, and whether your code tests correctly. Therefore the points during grading will be distributed as follows (The whole assignment is worth a total of 100 points).

ALGORITHM DESIGN, CODING & TESTING (65 pts):

- (25 pts): Is the algorithm design efficient? That is, does the algorithm deploy the appropriate kind of data structures and use them effectively in a way that is expected to deliver the best possible performance both in terms of runtime and memory?
- (20 pts): Is the code implemented in an efficient way? i.e., are there parts in the code that appear redundant, or implemented in ways that can be easily improved? Does the code conform to good coding practices of Objected Oriented programming?
- (10 pts): Is the code documented well and generally easy to read (with helpful comments and pointers)?
- (10 pts): Does the code compile and run successfully on a couple of test cases? Also does the output for the sample test case provided look right?

REPORT (35 pts):

- (25 pts): Is the algorithm described well in the report? Are all important details effectively presented, or does the TA have to dig up some critical information (such as which STL classes were used to implement the Board class or how the MoveTo method was implemented using different data structures) from the code?
- (10 pts): Is the basis of the choice of different data structures well justified/supported with the presentation of the expected time and memory complexities?

Obviously to come up with the above evaluation, the TAs are going to both read your report and run your code.