HW 9: OOP Design.

1. Play tic-tac-toe using the attached code for tic-tac-toe game program – test all possible outcomes: win, loss, tie.

2. Decide how to split the tic-tac-toe game program (written in a procedural way) into classes, including Game, Board, AbstractPlayer, Computer, Human.

3. Demonstrate these classes via UML diagram, showing inheritance (is-a) and aggregation (has-a) relationship between these classes, containing member variables and functions.

4. Assign the existing variables and functions to the appropriate classes, get rid of unnecessary parameters and return values (you don't need to pass and return member variables of the class). Change the functions *humanMove* and *computerMove* to *move*; change the functions *humanPiece* and *opponent* to *selectPiece*; functions *move* and *selectPiece* should be pure virtual in AbstractPlayer class and defined in each derived class). Define the new functions if needed, such as constructors.

5. Split the program into these classes, modify the functions declarations and definitions. Split the program into header files, implementation files, and the application file. (Look at blackjack.cpp as an example of using various classes in a game program.)

6. Compile and run your modified program, play the tic-tac-toe game testing all possible outcomes: win, loss, tie.

7. Modify functions askYesNo and askNumber in tic-tac-toe program: if inputted values are not (y/n) in askYesNo and not an integer or an integer outside of low-high range in askNumber, then the program should output the corresponding incorrect input message and ask for input again (in all the cases of incorrect input). To implement this feature, declare classes NotY/N, NonDigit, OutOfRange; make askYesNo and askNumber input each value as string, throw exceptions for each case of incorrect input and catch these exceptions outputting corresponding message. In askNumber check each character with !isdigit(c) (before throwing an exception) and convert string s to integer temp = atoi(s.c_str()). (Look at histogram.cpp as an example of using exceptions for a foolproof I/O).

8. Test inputting all kinds of correct and incorrect answers.

9. Submit UML diagram, all the source files, and output of all the tests.