Lab 04 - Inheritance & Polymorphism

Instructions:

- **The game of chess consists of six distinct pieces, namely pawn, knight, rook, bishop, queen, and king. Each piece has a unique way of moving. The pawn can move one space forward vertically but can move two spaces forward vertically on its first move. The knight can move in an L-shape [either two spaces vertically, and then, one space horizontally; or one space vertically, and then two spaces horizontally]. The rook can move horizontally or vertically in any number of spaces. The bishop can move diagonally any number of spaces. The queen can move like the rook and the bishop. And the king can move one space in any direction. Your objective is to define chess piece classes that inherit an abstract class for a playable chess piece.**
- **Your submissions must be submitted to the GitHub repository in the Lab04 directory.**
- **Make sure to include the header file `"Position.h"` to your header files. It contains the class *Position* that contains private int fields to hold horizontal and vertical coordinates, public special member functions, a public overloaded constructor that takes two int parameters that assign values to the horizontal and vertical coordinates respectively, public getter and setter methods named `GetX()`, `GetY()`, `SetX()`, and `SetY()`; the setter methods only perform an assignment if the parameter is between 0 and 7 inclusively. And a public `ToString()` method and a friend ostream operator.**
- **Cheating of any kind is prohibited and will not be tolerated.**
- **Violating and/or failing to follow any of the rules will result in an automatic zero (0) for the lab.**

Grading

Task	Name	Maximum Points	Points Earned
1		2	
2		1.5	
3		1.5	
Total		5	

Task 1

Create a header file named `"AbstractPiece.h"` that defines an abstract class named *ChessPiece* within the namespace *LB4* that must contain

- a protected *Position* field named *location*.
- a protected bool field named *team*.
- a private int field named *moves*.
- a protected void method named `IncrementMoves()` that takes no parameters and increments *moves* by 1.
- a public default constructor that assigns the position (0,0), 0, and false to the fields *location*, *moves*, and *team* respectively.
- a public overloaded constructor that takes a constant *Position* reference parameter and a bool parameter and assigns each parameter to their respective fields, and 0 to *moves*.
- a public copy constructor.
- a public assignment operator.
- a public empty destructor.
- a public constant getter method for *location* named `GetPosition()` whose return is a constant *Position* reference.
- a public pure virtual bool method named `MoveTo()` that takes a constant *Position* reference parameter.
- a public virtual string constant method named `ToString()` that takes no parameters. It returns a string in the format

$$\begin{cases} \texttt{White}\ x & \text{if } \textit{team} = \texttt{true} \\ \texttt{Black}\ x & \text{if } \textit{team} = \texttt{false} \end{cases}$$

  where x is the value of *location*.
- a friend overloaded ostream operator that returns its output in the same format as `ToString()`.

Task 2

Create header files named `"Knight.h"`, `"Queen.h"` and `"King.h"` that define the classes *Knight*, *Queen*, and *King* respectively such that each class is within the namespace *LB4* and publicly inherit *ChessPiece*. Furthermore, each class must contain

- a public overloaded constructor that takes a constant *Position* reference parameter and a bool parameter and assigns each parameter to their respective fields, and 0 to *moves*.
- a public copy constructor.
- a public assignment operator.
- a public overridden `MoveTo()` method that assigns the parameter to *location*, increment *moves* and returns true if the parameter represents a valid move from *location* for the respective piece; otherwise, it just returns false.
- a public overridden `ToString()` method that returns a string in the format

$$\begin{cases} \texttt{White}\ n\ x & \text{if } \textit{team} = \texttt{true} \\ \texttt{Black}\ n\ x & \text{if } \textit{team} = \texttt{false} \end{cases}$$

  where x is the value of *location*, and n is `"Knight"`, `"Queen"`, and `"King"` for the classes *Knight*, *Queen* and *King* respectively.

Task 3

Create header files named `"Pawn.h"`, `"Rook.h"` and `"Bishop.h"` that define the classes *Pawn*, *Rook*, and *Bishop* respectively such that each class is within the namespace *LB4* and publicly inherit *ChessPiece*. Furthermore, each class must contain

- a public overloaded constructor that takes a constant *Position* reference parameter and a bool parameter and assigns each parameter to their respective fields, and 0 to *moves*.
- a public copy constructor.
- a public assignment operator.
- a public overridden `MoveTo()` method that assigns the parameter to *location*, increment *moves* and returns true if the parameter represents a valid move from *location* for the respective piece; otherwise, it just returns false. Only the distance of the positions matters for the pawn.
- a public overridden `ToString()` method that returns a string in the format

$$\begin{cases} \texttt{White}\ n\ x & \text{if } \textit{team} = \texttt{true} \\ \texttt{Black}\ n\ x & \text{if } \textit{team} = \texttt{false} \end{cases}$$

  where x is the value of *location*, and n is `"Pawn"`, `"Rook"`, and `"Bishop"` for the classes *Pawn*, *Rook* and *Bishop* respectively.

Extra Credit

Create a header file named `"Extra.h"` define an interface named *Comparable* that must contain

- a public pure virtual int constant method named `Value()` that takes no parameters.
- a public pure virtual string constant method named `ToString()` that takes no paramaters.
- a friend overloaded ostream operator that returns an output in the same format as `ToString()`.

And a class named *Box* that publicly inherits *Comparable* and must contain

- a private int field named *length*.
- a private int field named *width*.
- public special member functions that make the default values of both fields 1.
- a public constant getter method for each field.
- a public setter method for each field that only assigns the parameter to the field if the parameter is positive.
- a public overridden `Value()` method that returns the product of the fields.
- a public overridden `ToString()` method that returns a string in the format

  $$\text{lne}[x,y]$$

  where x and y are the values of the fields *width* and *length* respectively.

(2 points)