CS202

Assignment 5: Miniature Chess

Description

Ah yes, chess. The game of champions, a strategic game like league except chess builds and strengthens friendships not ruin friendships, unlike league of legends or whatever the kids play these days. Chess also makes sense unlike league. Anyway, in this miniature version of chess you will only have a queen, two rooks, two bishops, and two knights. You will have the following abstract class

```
#pragma once

class chessPiece
{
public:
  chessPiece(bool);
  virtual bool move(char, int, char, int, chessPiece***) = 0;
  bool getPlayerType() const;
  virtual ~chessPiece();
private:
  bool color;
};
```

Each member performs/contains the following

- `bool color` - each piece will be colored red or black, `true` for red and `false` for black (we use red and black to celebrate red black trees since it's a fun data structure)
- `chessPiece::chessPiece(bool color)` - constructor that sets `this->color = color`
- `bool chessPiece::getPlayerType() const` - function that returns color
- `chessPiece::~chessPiece()` - destructor, when we deallocate an object from the chess board, outputs the following message `"Piece removed from board."`
- `virtual bool move` is a pure virtual function that is overridden in the derived classes

Queen Class

```
class queenType : public chessPiece
{
public:
  queenType(bool);
  bool move(char, int, char, int, chessPiece***);
  ~queenType();
};
```

Each member performs/contains the following

- `queenType::queenType(bool color)` - queen constructor that makes a call to the base class constructor
- `bool queenType::move(char startRow, int startCol, char endRow, int endCol, chessPiece*** board)` - function that verifies if coordinate [startRow, startCol] to coordinate [endRow, endCol] is a valid move and the path from [startRow, startCol] to coordinate [endRow, endCol] contains no other chess piece and coordinate [endRow, endCol] is either `nullptr` or a `chessPiece` of the opposite color, if this is the case then return `true` otherwise return `false`
- `queenType::~queenType()` - destructor, outputs `"Queen Taken."`, this occurs when we deallocate a queen object off the board in main

Rook Class

```
class rookType : public chessPiece
{
public:
  rookType(bool);
  bool move(char, int, char, int, chessPiece***);
  ~rookType();
};
```

Each member performs/contains the following

- `rookType::rookType(bool color)` - rook constructor that makes a call to the base class constructor
- `bool rookType::move(char startRow, int startCol, char endRow, int endCol, chessPiece*** board)` - function that verifies if coordinate [startRow, startCol] to coordinate [endRow, endCol] is a valid move and the path from [startRow, startCol] to coordinate [endRow, endCol] contains no other chess piece and coordinate [endRow, endCol] is either `nullptr` or a `chessPiece` of the opposite color, if this is the case then return `true` otherwise return `false`
- `rookType::~rookType()` - destructor, outputs `"Rook Taken."`, this occurs when we deallocate a queen object off the board in main

Bishop Class

```
class bishopType : public chessPiece
{
public:
  bishopType(bool);
  bool move(char, int, char, int, chessPiece***);
  ~bishopType();
};
```

Each member performs/contains the following

- `bishopType::bishopType(bool color)` - bishop constructor that makes a call to the base class constructor
- `bool bishopType::move(char startRow, int startCol, char endRow, int endCol, chessPiece*** board)` - function that verifies if coordinate [startRow, startCol] to coordinate [endRow, endCol] is a valid move and the path from [startRow, startCol] to coordinate [endRow, endCol] contains no other chess piece and coordinate [endRow, endCol] is either `nullptr` or a `chessPiece` of the opposite color, if this is the case then return `true` otherwise return `false`

- `bishopType::~bishopType()` - destructor, outputs `"Bishop Taken."`, this occurs when we deallocate a queen object off the board in main

Knight Class

```
class knightType : public chessPiece
{
public:
  knightType(bool);
  bool move(char, int, char, int, chessPiece***);
  ~knightType();
};
```

Each member performs/contains the following

- `knightType::knightType(bool color)` - knight constructor that makes a call to the base class constructor
- `bool knightType::move(char startRow, int startCol, char endRow, int endCol, chessPiece*** board)` - function that verifies if coordinate [startRow, startCol] to coordinate [endRow, endCol] is a valid move and coordinate [endRow, endCol] must be either `nullptr` or a `chessPiece` of an opposite color, if this is the case then return `true` otherwise return `false`
- `knightType::~knightType()` - destructor, outputs `"Knight Taken."`, this occurs when we deallocate a queen object off the board in main

Contents Of Main

In main, you first need to allocate a chessboard and place the pieces on the board (given to you in the provided main). Then you must implement the following algorithm

1. Output the board
2. Ask the user for starting coordinates [x_1, y_1]
3. If $board[x1][y1]$ is either null or color of the piece does not match the player color (a `bool` variable, initially set to `true` that denotes red player turn), output `"Starting coordinate is empty!"` or `"Invalid piece selected."` and go back to the previous step
4. Ask the user for ending coordinates [x_2, y_2]
5. If $board[x_1][y_1] \rightarrow move(x_1, y_1, x_2, y_2, board)$ returns `true` then move the piece at position [x_1, y_1] to [x_2, y_2], if a piece of opposite color is at [x_2, y_2] then `delete board[x2][y2]` before you move the piece at [x_1, y_1] to [x_2, y_2] and then go to the next step, if the move function returns `false` then output `"Invalid move!"` and go back to step 2
6. Alternate a `bool` variable from `true` to `false` and vice versa (used to alternate player turn)
7. If the game is over output the player that won and go to the next step, otherwise go back to step 1
8. Deallocate the chessboard pieces and end the program

Specifications

- Make sure you program is memory leak free
- Properly document your code
- Use upcasting (polymorphism) when calling move function, only use dynamic casting when you output the chess piece type to the screen

Sample Run

`An exe file is provided that shows how the program runs`

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/R7JD-MQQXok
- Link to the top image can be found at https://intelli-corp.com/test/wp-content/uploads/2016/01/chess_main.png