Project: **Chess using C++** Name: **Mamta Shah**

Compilation Instructions:

To run the program use the Makefile to compile by following the steps below:

- Open Makefile and ***change the PATH variable value*** to the path of the folder in which all the files, namely Board.cpp, Board.h, Piece.cpp and Piece.h are present.
- Execute "***make all***" or "***make***" command using the command line to automatically compile the program named as "chess".
- Execute "***./chess***" command on the command line to start executing the program.

Playing Instructions:

- The program displays the menu and at each step asks for user input. The user input is an integer based on the action he/she wants to perform. The mapping of user input to the action performed is as shown below:
    - 0: Get the next move of the player white or black based on whose turn it is.
    - 1: Reset the board to its original state.
    - 2: Print the current state of the board.
    - 3: Print this menu of options.
    - 4: Seamless play. (Refer section Enhancements)
- The black pieces are shown with dark blue color and white pieces are shown using white color on the board printed.
- After entering 0, a user will be prompted to input the move for either black or white player based on whose turn it is. The input should be 4 integers in the following manner, ***"<current_row_index><space><current_column_index><space><destination_row_index><space><destination_column_index><enter>"***. For example, to move a white pawn from position (7, 2) on the grid to (5, 2), we need to input **7 2 5 2**.
- The symbols printed on the board with printBoard function mean the following:
  R--> Rook, K --> Knight, B-->Bishop, Q-->Queen, K*-->King, P-->Pawn

Files:

There are 4 files, namely ***Board.cpp, Board.h, Piece.cpp and Piece.h***.

- **Board.cpp:** Contains the main function used to simulate the game. Contains the Board class and its related functions such as resetBoard, printBoard and makeMove. It's makeMove function is highly dependent on the Piece class which identifies the validity of move for particular type of piece such as king, pawn, queen, etc…

- **Piece.cpp**: Contains skeleton of each piece on the board in class Piece. All the other classes are derived from Piece such as King, Queen, Bishop, Knight, Rook and Pawn each of which implements their own valid move identification function.

Enhancements:

It seemed very cumbersome to play by giving options to print, reset and make a move on the board continuously and hence a new option was incorporated to help play seamlessly. If a user chooses option 4 then the program goes to an continuous loop of make move option choice. This option will print the board after every move and continuously ask for the next move from the respective player.

Environment Tested on:

- The program was compiled and tested using g++ version information shown below:
    - Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/c++/4.2.1
    - Apple clang version 11.0.3 (clang-1103.0.32.59)
    - Target: x86_64-apple-darwin19.5.0
    - Thread model: posix
    - InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin

Bonus Questions:

1. **If we were to deploy this app into production (on an iOS app or a webapp), can you describe how you would configure the development and production flow?**
   I would configure the workflow as below:
    - Come up with a ***design document*** for the product addressing the target audience, clearly mentioning functionalities, limitations of the app and test conditions. Also, identify the best technology to use in order to reduce work and make it adaptable to most platforms.
    - Get the design document reviewed by the team and finalize things and division of work for development, peer review and exhaustive testing.
    - Break list of functionalities into ***versions of development/release***. Example, tournaments of chess, random pairing of players etc... can be added in later

versions however the minimal chess playing functionality of play between two players would be bare minimum.

- Adopting ***iterative testing*** that is ***unit testing*** for each module, that is developed modules need to be tested by developers each day and need to be submitted for ***peer review of code***.
- Full functional, scalability and negative testing.
- Internal release to team for beta testing.
- Release for general use.

2. **What kind of testing would you perform to validate the app before release to customers or demoing?**
   I would recommend an exhaustive list of types of moves for each piece to be tested before the release, for example, testing that each rook is able to move left, right, up and down on the board. I would also recommend all types of negative test cases for each piece, for example, rook can not move over a piece on its way to the destination cell.
   If possible I would like to conduct an in-house beta testing amongst the team members so as to not only validate the functionality but also get a feedback on the usability, visual appeal and any other feedback and comments which could be used to enhance the app.