**Project report name**

**The Chess Game**

**Group members**

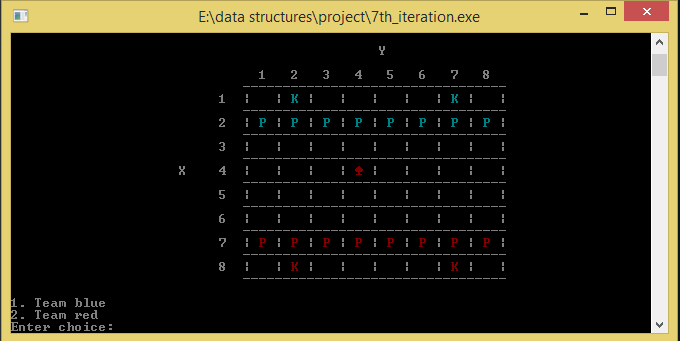
**Muhammad Ferdeen Bakht (20K-0219)**

**Muhammad Ayaz (20K-1044)**

**Hamza Ahmed (20K-1097)**

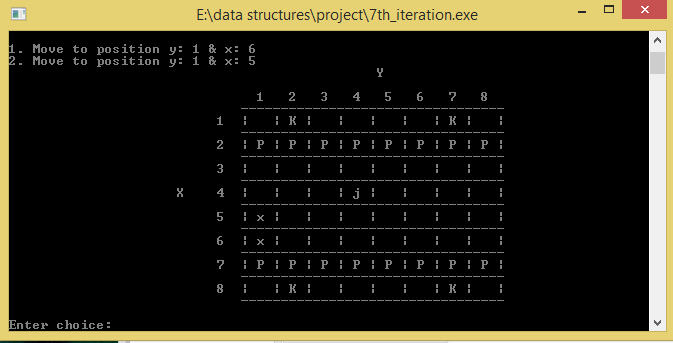
**Introduction**

The chess is one of the most common indoor game. It includes pieces which have specific moves. The pieces that exist are king, pawn, knight, bishop, queen and rook. Each piece has its own moves which are different from each other. There are two teams and the job of both teams is to protect the king. If the king has no legal moves it is called slate-mate and if it is attacked and also has no legal moves it is called check-mate in both ways the game ends. If you checkmate you win the game and if you stalemate the game draws. The basic interface looks like this:



**Working**

We created a 8x8 2d array to represent the board of the chess. Then we represented each piece of the chess with a specific number for e.g. knight has a number 1(-1 for opposing team) and pawn has number 2 and then we converted it into P and K. We have implemented specific conditions which the specific piece can move or cannot move. The moves are also displayed and are represented by letter x.



**Applications**

If asked in terms of its use this simple command based interface can be used to create mobile game or windows game or an online website game.

**Algorithms used**

The greatest essence of this program is backtracking of king which was purely created by our creativity. This backtracking algorithm of the king never existed before. The function solve() also uses recursion to update the values. Complete binary tree is also used as it removes values in the same order as pushed. The list of the types of data structures or algorithms we used:

-Backtracking ( used in solve\_king () function )

-Linked List

-Stack

-Queue

-Complete Binary tree

-Recursion ( used in solve() function )

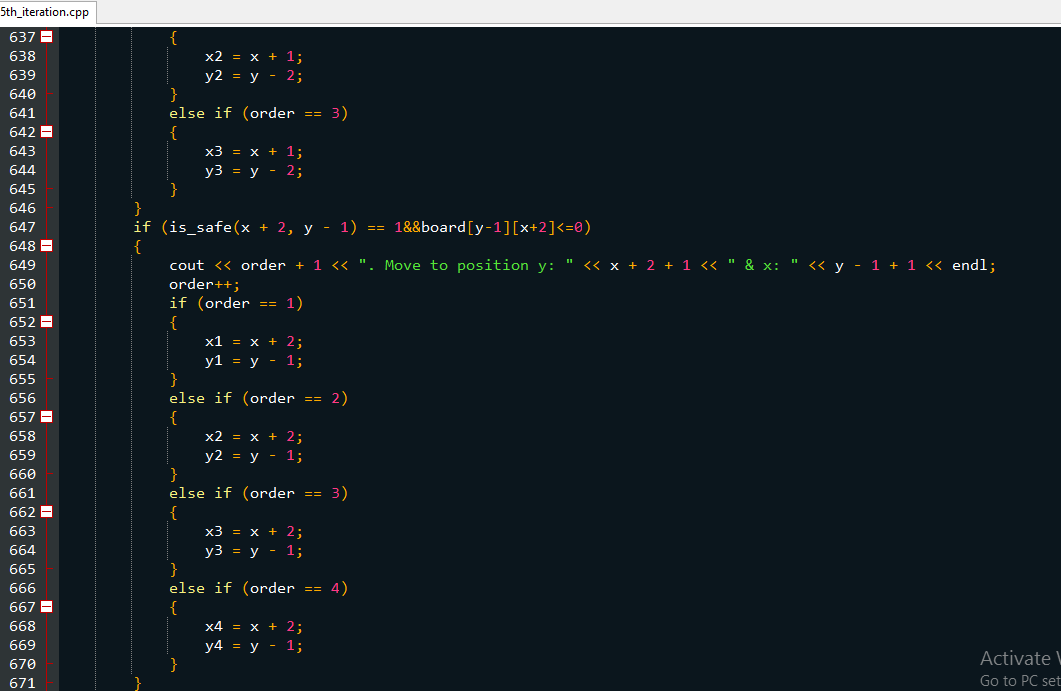
**Challenges and Discussions**

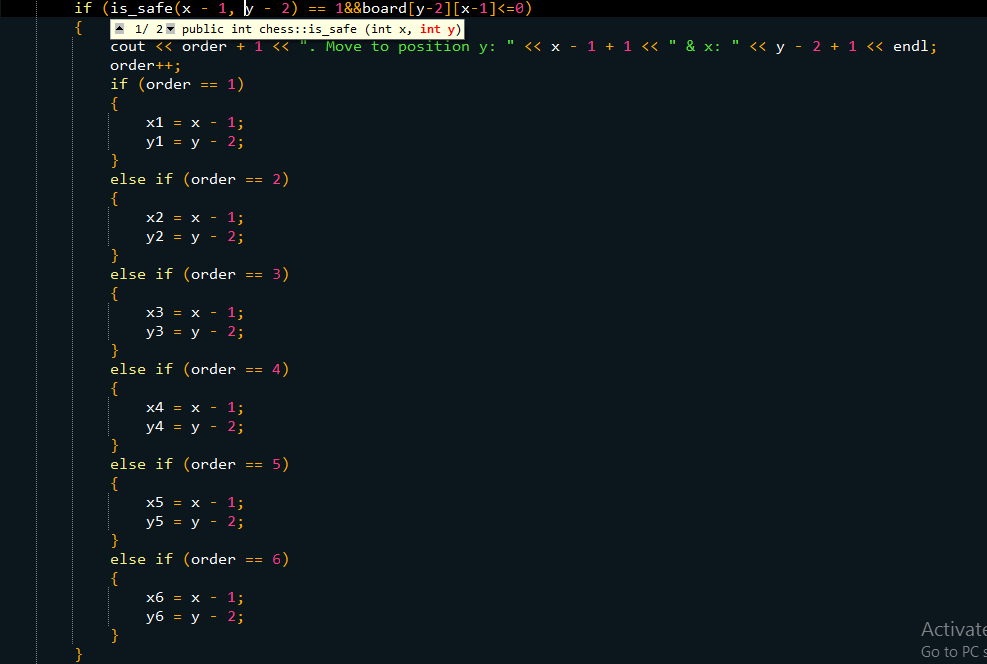
The advantage of using data structures/algorithms was that the size of code decreased and it became more readable.

The previous iteration of code shows lengthy code with large amount of if-else statements in the function solve\_knight(). If we speak mathematically with every if statement to maintain the order the if-else statements increases n times. After we made use of DS there was no need to put if-else statements inside the if statement.

**Before using data structures/algorithms**

****





**After using data structures and algorithms**

