Chess

## Description:

Chess is a two-player board game played on a chessboard, a checkered game board (usually black and white) with 64 squares arranged in an eight-by-eight grid. Each player begins the game with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. Each of the six-piece types moves differently. The most powerful piece is the queen and the least powerful piece is the pawn. The objective is to checkmate the opponent's king by placing it under an inescapable threat of capture.

## Features:

* Two players: Human vs. Human.
* Check.
* Checkmate. (if the player has no legal moves and the king is in check).
* Protecting the king using other pieces.
* Stalemate (if the player has no legal moves).
* Promotion.
* Undo & Redo.
* Save & Load.
* Undo & Redo are also available in the loaded game.
* Starting a New game (or load) after the game has ended.

## Design:

* White squares are "."
* Black squares are "="
* White pieces are represented by "p", "r", "n", "b", "q", "k" for pawns, rooks, knights, bishops, queen and king respectively.
* Black pieces are represented by "P", "R", "N", "B", "Q", "K" for pawns, rooks, knights, bishops, queen and king respectively.
* "Dead" list on the right to show what pieces has been taken out.
* Letters on top and bottom (from A to H) & Numbers on left and right (from 1 to 8) of the board to indicate from where to where the pieces can move
* Pieces on a white square look like ".k"
* Pieces on a black square look like "=K"
* "|" & "­-" are the boarders of the playing board.

## Data types used:

* Integer Arrays
* Character Arrays
* Integer variables
* Character variables

## Important functions:

### void readLine:

Reads all user inputs until '\n' is inputted.

### void startingChessBoard:

Initiates the starting positions of the pieces on the board.

### void scanAndConvert:

Takes the input of the user and changes it to an index to be used on the pieces to know where from and where to move the piece. And checks if the input is correct or not.

### void checkValidMove:

Checks if the piece is chosen correctly from each player.

### void kingsIndex:

Sets and index for each king.

### void swapElements:

Swaps the pieces and eliminates if there is an opposite piece while swapping.

### void reverseSwapElement:

Reverses the swap and returns the eliminated piece from the last turn:

### void saveUndo & saveRedo:

Saves each turn's moves to be able to be undone, and if undone saves it to be able to be redone.

### voids (rook, knight, king, bishop, queen, pawn):

Checks if the player's moves are correct to each corresponding piece.

### void moveSet:

Chooses which piece the program should check it's move.

### int isKing (Boolean flag):

It's used to identify if the piece is king or not.

### voids fullPieceCheck(White/Black):

Checks the sent argument (piece) if it is in check or not (can be eaten in the next move or not) .

* Might be captured by (Knight, bishop, queen, rook).
* If (isKing = 1) that means the selected piece is a king and it might be captured by a opposite king or opposite pawn.
* If (isKing = 0) that means the selected place is not a king (an empty space or another piece) and it might be captured by opposite pawn. (if empty space then the pawn moves normally if not then the pawn moves in a capturing move)

### voids checkMate(White/Black)Path:

Checks the 8 directions (up, down, left, right and 4 diagonals) from the desired place to the sent arguments.

* In every index(x,y) for every path, check if any same colored piece can be in this index, by using "fullPieceCheck(Black/White). (Eg: if checkMateWhitePath then we use fullPieceCheckBlack) because if this index is in check then there is a possible move.

### voids king(White/Black)CheckMate:

Only uses this function if king in check to check if it's checkmate or not.

* Checks if the 8 indexes around the king can have the king in check if it moved to it.
* If it can't move to any of the 8 indexes, we check if any same colored piece can capture the opposite checking piece by using "fullPieceCheck(Black/White)" (if there is a piece than can capture the checking piece, then it means there is a possible move)
* If there are no possible moves, it uses "checkMate(Black/White)Path" to check there is any possible moves.
* If there is none, then the same colored piece as the function is in Checkmate.

### voids checkValid(White/Black)Stalemate:

Checks if that piece moved to any other place will the king be in check or not.

### voids king(White/Black)StaleMate:

Only uses this function if king isn't in check to check if it's stalemate or not.

* Checks if the 8 indexes around the king can have the king in check if it moved to it.
* If the king is in check in all 8 indexes. Checks if the board has any legal moves or not for the same color.
* It checks all the indexes in the board by checking if any same colored piece can capture the opposite piece by using "fullPieceCheck(Black/White)"
* If any opposite piece is found that it can be in check (it means that he has legal moves), it checks if that piece that has legal move if it moved, will the king be in check or not by using "checkValid(White/Black)Stalemate".
* If the king won't be in check then it means he has legal move and he is not in stalemate.
* If he has no legal moves, then it is Stalemate and the game ends by draw.

# User Manual:

1. All inputs should be in capital letters.
2. "U" to undo.
3. "R" to redo if undo is done.
4. "S" to save the game then you name your save file.
5. "L" to load the game.
6. To promote, you move the pawn then enter the desired piece to promote to when asked which do you want to promote:

eg. (E7E8, "Promote to:" Q)

* 1. "N" to knight.
  2. "B" to bishop.
  3. "Q" to queen.
  4. "R" to rook.
  5. "P" to stay as pawn.

1. If any other input is done you will get a "Invalid move" message.

# References:

Using FILE in save and load:

1. <https://www.tutorialspoint.com/cprogramming/c_file_io.html>
2. <https://www.youtube.com/watch?v=38I_AUMpKpQ&index=50&list=PL6gx4Cwl9DGAKIXv8Yr6nhGJ9Vlcjyymq>