Compiling Instructions:
-A Makefile is attached.

Running Program with Optional Parameters from the Command Line:
-b boardFile = boardFile is a text file containing the board representation in the following format. 1=Red, 2=Black, 3=Red King, 4=Black King. An empty space can be represented by any character besides a space and a period. Periods delimit the 8 rows, and there are 4 columns per row.

> Note the orientation of the board. I have seen boards both ways, with either a black or red square in the top-left.
> Also, the file is parsed from row 1 to row 8, meaning the first 4 characters in the file represent the row 1, the bottom row of the board.

An example file would contain the following text: 1111.1111.1111.----.----.2222.2222.2222

-p types = types is a two digit number. The first digit is the type of Player 1, the red player, and the second digit is the type of Player 2, the black player. A 0 represents the computer, and a 1 represents a human. The default is “10”.

-h hnumber = hnumber is a two digit number. The first digit is the heuristic number used by the red player, and the second digit is for the black player. I have made it default to heuristic #2, which is more complex, and have left in one other basic heuristic (number 1) to demonstrate this optional feature.

Setting Up:
There are two prompts, before starting the game. First, the user chooses who moves first. Typically, red should move first in a standard game of checkers. Seconds, the user chooses a time limit for the program's moves. This time-limit applies to all players which have a type specified as “computer.”

Moving:
All legal moves are listed by “#: [origin]->[dest]”. If it is a jump, there is a “^” character. Simply type in the # preceding, then move. Board locations are described using row.col notation.
For multiple-jump moves: if there is an option on a later jump, you will be prompted for that choice after picking the first stage of the jump.

Draws:
If, the computer decides that neither person has an advantage or the ability to win, it will offer a draw. This decision is made if all pieces are kings, the players are equal in material, each has three pieces or less, and no jumps are available to the next player. A user can offer a draw by typing “draw” at move-choice prompt. In a game against a computer, this is unnecessary, because if a computer is willing to accept a draw, it will offer it to you as well.

Implementation:
To organize the code, I use multiple classes. These include Game, Player, Board, Move, Square.
Game - As Game.play() loops through the turns, it first finds legal moves. it checks what player type is present for the current turn. If it is a human, it outputs the list of moves, and prompts the user for a choice. It then makes the move, checking to make sure that the game has not ended.

Move – In checkers, the moves are a bit interesting. Each move contains an “origin” and “destination” square. If the move is a jump, then there is also a “jumped” square, which will be

cleared when the move is made. In addition, it also must allow for further jumps. Therefore, a move contains a vector of pointers to further moves, called nextJumps, as well as a single pointer to a move called moveNextChosen.

Board – This is the largest of all of the classes. It is not connected to the player at all, and simply cares about which color is moving. It has the following functionalities:

- Get all legal moves for a particular color, returning a vector of move pointers
- Make a move, changing the board state
- Print the board to standard output
- Get the best move from a vector of moves, using iterative deepening mini-max search with alpha-beta pruning
- Use a heuristic function to evaluate a given board state, to allow a cutoff at a particular depth for mini-max

The heuristic function takes into account the following characteristics of the board. The weights of each of these factors were tweaked to optimize the playing ability. In certain aspects, the heuristic used near the end of the game, is different than the one used in the beginning and middle:

- Material Advantage
- Progress of non-kings down the board
- Number of pieces on the board. A winning player wants to minimize, and a losing player wants to maximize.
- Distance from all opponent's pieces. A winning player wants to minimize, and a losing player wants to maximize.
- Various bonuses, including position on the sides, the two middle positions of the board, a full back row, and possession of the turn.