You code will play a game called CaptureMe.   CaptureMe is a game similar to Reversi or Go. The rules for this game are a little different from GO or Reversi so please read the rules below carefully.

the CaptureMe.cpp file should contain:

1. main() function

2. CheckLine function

3. PlacePiece function

4. InitializeBoard function

5. DisplayBoard function

You will need to develop and implement an algorithm to solve the problem. Include the common case and most special cases in the algorithm and implementation. You must implement all special cases including those special cases that involve inputs of incorrect data type.

The Game of CaptureMe is a strategy oriented board game with the following rules:

1.     The game begins on a board of N squares by N squares.   N must be at least 8 and N must be even,

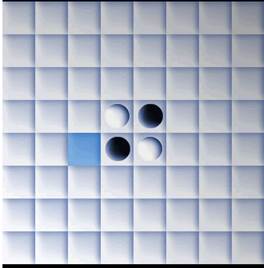
2.     The game is played by two players using circular game pieces that are placed on the squares of the board. One player uses white game pieces; the other player uses black game pieces.

3.     White moves first.

4.     The game ends when of the condition below is true

When a player plays 0 game pieces and their opponent also plays 0 game pieces

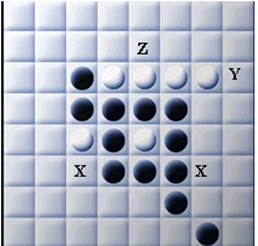
5.     The initial board will be set up as follows



6.     When a player places a game piece on the board it MUST trap at least one of their opponent’s game pieces.  To understand when a piece is trapped we must think of the board in terms of ‘lines’. A line must be straight, but may be horizontal, vertical or diagonal.

a. To trap an opponent’s piece there must be one of your pieces on each side of the piece within a particular line. For example placing a white piece on the blue square in the board above would trap one black piece between two white pieces in a horizontal line.

b.You may trap any number of your opponent’s pieces in a single move. To trap more than one of your opponent’s pieces your opponent must have more than one adjacent piece in a line.  Some examples are shown on the board below. If it is white’s move, the X’s show two possible squares where a white piece could be placed to trap two black pieces in a row along a diagonal line. If it is black’s move the Y shows where a black piece could be placed to trap 4 white pieces in a horizontal row.  In some cases it may be possible to trap opponent’s pieces along more than one line with a single move. An example of this is shown on the board below. Assume that it is black’s turn and black has chosen to place a black piece on the square indicated by the Z. This move traps 1 piece along a vertical line and 1 piece along a diagonal line.



7.     When a player makes a move they must place zero, one, or two pieces on the board.  Each piece placed on the board must trap an opponent’s game piece.

a.      If it is not possible to trap an opponent’s game piece along a horizontal line or along a vertical line (your program needs to check this and end the turn if necessary) then the player places zero game pieces on the board and ends their turn.

b.     If it is possible to trap an opponent’s game piece along a horizontal line or along a vertical line then the player must try to play their first game piece.  If the player does not trap two or more of their opponent’s pieces when they play their first game piece the player’s turn is over.

c.      If the player traps two or more of their opponent’s game pieces along a horizontal line or along a vertical line when they place their first game piece on the board then the player may choose to place a second game piece on the board.  If the second game piece is placed it MUST be placed so that it traps at least one opponent’s game pieces along a diagonal line.

8.     Immediately after a game piece has been placed on the board ALL trapped opponent’s game pieces will be “captured”.   This includes game pieces trapped in ALL possible directions (horizontally, vertically, or diagonally). When a game piece is captured the color of the game piece is changed.  In the examples above the black game pieces trapped by placing the new white game pieces on one X would become white game pieces or the white game pieces trapped by the new black game piece places on the Y would be changed to black game pieces.

**Your main program should**

***1.***     Declare an array myCaptureMeBoard[MAX\_ARRAY\_SIZE][MAX\_ARRAY\_SIZE] as a local automatic variable in the main program.  MAX\_ARRAY\_SIZE should be a declared constant with global scope and with a value 24.

2.     Prompt for and read the size of the board (***numRowsInBoard*).**Use the following prompt

***Enter the number of squares along each edge of the board***

***a)***     Check the  board size entered by the user.

The correct range for board size is***8 <= numRowsInBoard <= 24,  numRowsInBoard*** must be even***.***If the board size is not valid one of the following error messages should be printed

***ERROR:  Board size too large***

***ERROR:  Board size too small***

***ERROR:  Board size odd***

***ERROR:  Board size is not an integer***

***b)***     If the user enters an invalid value or a value that is out of range, tell the user what legal values are by printing the message

***8 <= number of squares <= 24***

and reprompt for and reread a new value. Read the value a maximum of 3 times, if the value is still  invalid or out of range print the message below and terminate the program.

***ERROR:  Too many errors entering the size of the board***

***3.***     Initialize the game board using the***InitializeBoard***function.

a)     Declare the array to hold the game board  in the main program

b)     Do not initialize the array in the main program

***c)***      If the board does not initialize correctly (***InitializeBoard*** returned false) print the error message below and then terminate the main program

***ERROR:  Could not initialize the game board***

***4.***     Print the initial board to the screen using the ***DisplayBoard***funtion

***5.***     Check to see that the board displayed correctly (***DisplayBoard*** returned true)

***a)***     If the board does not display correctly print and error message and continue

***ERROR:  Could not print the game board***

6.     For each turn

a)     Check if the player can play their first piece by calling function IsMove.   If funtion IsMove returns false then there is no available move.

b)     If there is not possible square on which the player can place their first game piece (your program should check this) print one of the following messages.

***White is unable to move***

***Black is unable to move***

**c)**      If the player can place their first game piece on the board print one of the following single line message to the screen

***White takes a turn***

***Black takes a turn***

d)     Give the player three attempts to select a legal square to place their first game piece on.  If the player does not select a legal square in three tries he forfeits his turn. If the player forfeits their turn print one of the following messages

***White has forfeited a turn***

***Black has forfeited a tur***n

e)     For each attempt at placing the first game piece on the game board

                                     i.     If three attempts to make a legal move have already failed forfeit the turn.(print one of the messages in c) then go to step xi

                                   ii.     Request the number of the square on which the player wishes to place his piece. The number of the square will be displayed on the gameboard. For example the initial gameboard for an 8x8 board would be like that shown below. The square number of the square is shown for each empty square. W is shown for each white piece. B is shown for each black piece.

***0   1   2   3   4   5   6   7***

***8   9  10  11  12  13  14  15***

***16  17  18  19  20  21  22  23***

***24  25  26   W   B  29  30  31***

***32  33  34   B   W  37  38  39***

***40  41  42  43  44  45  46  47***

***48  49  50  51  52  53  54  55***

***56  57  58  59  60  61  62  63***

When requesting the number use the prompt

***Enter the square number of the square you want to put your first piece on***

                                 iii.     If the chosen square is not on the board print the error message below and continue to the next attempt at a legal mover (this attempt was not legal)

***ERROR:  that square is not on the board.***

***Try again***

Go back to step i.

                                 iv.     If the chosen square  is already occupied,  print an error message below and continue to the next attempt at a legal move (this attempt was not legal)

***ERROR:  that square is already occupied***

***Try again***

Go back to step i

                                   v.     Attempt to place the first piece by calling function ***PlacePiece***. If ***PlacePiece***returns false go back to step i.  (this attempt was not legal)

f)      If the first piece captured two or more of the opponents pieces, ask the player if they wish to play a second piece.  If the player is not eligible to play a second piece or does not wish to play a second piece then go to step v.  Check if there is a legal move (using IsMove).  If there is no legal move go to step 7, Otherwise.

                                          i.     Request (Request a maximum of 3 times) the number of the square on which to place the second piece using the prompt

***Enter the square number of the square you want to put your second piece on***

                                        ii.     If the chosen square is not on the board print the error message below and continue to the next attempt at a legal move (this attempt was not legal)

***ERROR:  that square is not on the board.***

***Try again***

                               Go back to step i.

                                      iii.     If the chosen square  is already occupied,  print an error message below and continue to the next attempt at a legal move (this attempt was not legal

***ERROR:  that square is already occupied***

***Try again***

                               Go back to step i

                                      iv.     Attempt to make a move by calling function ***PlacePiece***. If ***PlacePiece***returns false go back to step i.  (this attempt was not legal)

                                        v.     If ***PlacePiece*** returns true, print the board to the screen using the***DisplayBoard***funtion

***1.***  Check to see that the board displayed correctly (***DisplayBoard***returned true)***.***If the board does not display correctly print and error message and continue

***7.***     Go to step 6 and let the opponent take a turn

***8.***     Terminate the program if the maximum number of turns have been taken (N2)  or if both players are unable to move.   When the game finishes print the messsages

***GAME OVER***

***Please enter any character  to close the game window***

Then wait till the player enters a character, then terminate the program.

**Your function *InitializeBoard* should**

1.     Have the prototype

***bool InitializeBoard(int  CaptureMeBoard[MAX\_ARRAY\_SIZE][MAX\_ARRAY\_SIZE],  int numRowsInBoard );***

2.     ***InitializeBoard***  will fail (return false)  if the numRowsInBoard is out of range or if an unallocated dynamic array is passed to the function.

3.     The meanings of the values that will be placed into the CaptureMeBoard when it is initialized  are

a)     0 in an element of***CaptureMeBoard*** indicates there is no game piece on the square

b)     1 in an element of ***CaptureMeBoard*** indicates there is a white game piece on the square

c)     2 in an element of ***CaptureMeBoard*** indicates there is a black game piece on the square

4.     Begin by initializing the CaptureMeBoard to be an empty board

5.     Place the first four tiles on the center 4 squares of the board. (see the example in the example of the rules at the start of this lab)

6.     For example after initialization a 8X8  or 10X10 board would have the following values in the array ***CaptureMeBoard***

***0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0***

***0   0   0   1   2   0   0   0***

***0   0   0   2   1   0   0   0***

***0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   1   2   0   0   0   0***

***0   0   0   0   2   1   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

***0   0   0   0   0   0   0   0   0   0***

**Your function *DisplayBoard* should**

1.     Have the prototype

***bool DisplayBoard( int CaptureMeBoard[MAX\_ARRAY\_SIZE][MAX\_ARRAY\_SIZE],***

***int numRowsInBoard);***

2.     The meanings of the values tin the CaptureMeBoard are

a)     0 in an element of***CaptureMeBoard*** indicates there is no game piece on the square

b)     1 in an element of ***CaptureMeBoard*** indicates there is a white game piece on the square

c)     2 in an element of ***CaptureMeBoard*** indicates there is a black game piece on the square

3.     Step through the ***CaptureMeBoard***array

a)     For the nonzero elements of the ***CaptureMeBoard***

                                i.     Print a B if a there is a black game piece on the square (CaptureMeBoard element = 2)

                              ii.     Print a W if there is a white game piece on the square (CaptureMeBoard element =1)

b)     For zero elements (empty squares) of CaptureMeBoard print the square number

                                i.     An example of the square numbers for a 8x8 board is given above (in the discussion of the main program), here is an additional example for a 10x10 board

***0   1   2   3   4   5   6   7   8   9***

***10  11  12  13  14  15  16  17  18  19***

***20  21  22  23  24  25  26  27  28  29***

***30  31  32  33  34  35  36  37  38  39***

***40  41  42  43  44  45  46  47  48  49***

***50  51  52  53  54  55  56  57  58  59***

***60  61  62  63  64  65  66  67  68  69***

***70  71  72  73  74  75  76  77  78  79***

***80  81  82  83  84  85  86  87  88  89***

***90  91  92  93  94  95  96  97  98  99***

c)     Each number or character should be printed in a field 4 characters wide

**The provided function *CheckLine***

1.     Has the prototype

***bool CheckLine( int CaptureMeBoard[MAX\_ARRAY\_SIZE][MAX\_ARRAY\_SIZE], int numRowsInBoard,***

***int player, int squarePlayed, int xDirection, int yDirection, int& captured );***

2.     Begins at a square indicated by the ***squarePlayed*** (the place the player is trying to put his piece this turn).  The value of ***squarePlayed*** will be the square number as displayed in the***DisplayBoard*** function. The value of ***squarePlayed*** should be translated to the location in the ***CaptureMeBoard***array (the indices x and y in CaptureMeBoard[x][y] ).

3.     The values o***f xDirection*** and ***yDirection*** will indicate which direction to check for possible ‘trapping’ of opponents pieces.  If the piece is being placed at location (x,y) in the board (row x, column y)  then we can check in any one of eight directions. If***xDirection=1***and ***yDirection=0*** you are checking along a horizontal line toward the right. If ***xDirection = 0***  ***yDirection =*** -1 you are checking along a vertical line to the bottom.  If***xDirection*** = -1 and ***yDirection***-=1 then your are checking along a diagonal line upwards and to the left. Etc.

4.     The value of ***player***is 1 if this is a white player’s turn, and 2 if this is a black player’s turn.

5.     If the first square in the selected direction is empty, return false

6.     If the first square in the selected direction contains one of the player’s pieces, return false

7.     If the first square int the selected direction contains one of the ooponent’s pieces then we must consider addition squares further along the line in the same direction. For each successive square along the line in the direction being considered

a. if the square is empty return false.

b. If the square contains an opponent’s piece continue to the next square along the line

c. If you reach the edge of the board without finding one of the player’s pieces return false

d. If the square contains one of the player’s pieces then

                                          i.     Change each of the trapped opponent’s pieces to player’s pieces by changing the value in the CaptureMeBoard array

                                        ii.     Increment the value of captured for each piece that was captured (argument captured)

                                      iii.     Return true

**Your function *PlacePiece* should**

*1.*    Have the prototype

***bool PlacePiece(int CaptureMeBoard[MAX\_ARRAY\_SIZE][MAX\_ARRAY\_SIZE], int numRowsInBoard,  int player, int squarePlayed, int whichPiece, int& numberCaptured );***

2.     Begin at a square indicated by the ***squarePlayed*** (the place the player is trying to put a piece this turn).  The value of ***squarePlayed*** will be the square number as displayed in the***DisplayBoard*** function. The value of ***squarePlayed*** should be translated to the location in the ***CaptureMeBoard***.

3.     If whichPiece indicates the first piece is being played, then  make four calls to ***CheckLine***for horizontal and vertical lines.

a)     If any of the calls to CheckLine returns true then the chosen square is a square into which the piece may be placed.

                                          i.     Place the piece on the board

                                        ii.     Determine how many of the opponent’s game pieces will be captured.  It is possible the trap opponent’s pieces in more than one direction when placing a single piece on the board. To determine the number of pieces captured you must consider all directions (including diagonal directions).  The number of captured pieces is recorded in numberCaptured and sent back to the calling program.   The number captured includes all pieces captured in all directions.

                                      iii.     Change the colour of all the trapped pieces  (this is actually done in CheckLine)

                                      iv.     Print a line of text to the screen that says either

***White has placed the first piece***

***Black has placed the first piece***

                                        v.     return true

b)     If none of the the four calls to ***CheckLine*** for horizontal and vertical lines returns true then the move is not a valid move

                                          i.     Print a line of text to the screen that says

***Illegal move***

                                        ii.     Return false

4.     If whichPiece indicates the second piece is being played then make four calls to***CheckLine*** for diagonal lines

a)     If any of the calls to CheckLine returns true then the chosen square is a square into which the piece may be placed.

                                          i.     Place the piece on the board

                                        ii.     Change the colour of all the trapped pieces  (this is actually done in CheckLine)

                                      iii.     Print a line of text to the screen that says either

***White has placed the second piece***

***Black has placed the second piece***

                                      iv.     Return true

b)     If none of the the four calls to ***CheckLine*** for diagonal lines returns true then the move is not a valid move

                                          i.     Print a line of text to the screen that says

***Illegal move***

                                        ii.     Return false

Use dynamic allocation of the arraymyCaptureMeBoard[] in the main function. Main program should dynamically allocate the array after the size of the board has be successfully read (instead of statically allocating the array using a declaration)

***bool InitializeBoard****(int\*\* CaptureMeBoard,  int numRowsInBoard );*

***bool PlacePiece****(int\*\* CaptureMeBoard, int numRowsInBoard,  int player, int squarePlayed,* *int whichPiece, int& numberCaptured);*

***bool DisplayBoard****( int\*\* CaptureMeBoard,  int numRowsInBoard);*

***bool CheckLine****( int\*\* CaptureMeBoard, int numRowsInBoard, int player, int squarePlayed, int xDirection, int yDirection, int& captured)*