/\*Headers for program functionality\*/

// Iostream (stream input and output)

// limits (flushing the buffer)

// vector (vector data structure)

// algoritham (struct and .h connections)

// SFML Graphics (Sprites, Textures)

// usinging the std:: namespace

// using the sf:: namespace

/\*Function for chess piece allocation\*/

// Function will take pointers to x and y coordinates and a pointer to specifc coordinate (eg d4)

// If the coordiante is a certain string (eg: d4)

// x coordinate is set to a certain value

// y coordinate is set to a certain value

// both of these values are returned back to the main

// this is done by utlizing pointer reference and dereference (\* and &)

// Process done for A columm coordinates

// Process done for B columm coordinates

// Process done for C columm coordinates

// Process done for D columm coordinates

// Process done for E columm coordinates

// Process done for F columm coordinates

// Process done for G columm coordinates

// Process done for H columm coordinates

/\*Function for chess rules\*/

// This function will determine if a chess move is legal or illigal

// Function will take a pointer to a specific chess piece, a decided decision, the old piece poisiton and the new piece position)

// Creating integers to store int value of old and new positions of the chess piece

// an integer for old x position

// an integer for old y position

// an integer for new x position

// an integer for new y position

// Creating integers to hold the value of the difference between the old and new positions

// x coordinate difference

// y coordinate difference

// If value of distance is held as a negative value, it is changed to postive (easier evaluation)

// changing x coordinate

// changing y coordinate

// For a given chess piece, the distance is evaluated and determined if x and y movements are legal or illigal

// If the movement or difference is determined legal, the value of the decided decision is changed to 1

// If the movement or difference is determined illigal, the value of the decided decision is changed to 0

// These values are sent back to the main function where they will be used for decision making

// Process done for rooks

// Process done for knight

// Process done for bishop

// Process done for queen

// Process done for king

// Process done for pawn

/\*Min Max Algoritham for chess ai\*/

// This algoritham will help the Ai decide wheather it will attempt to capture or play defensivly

// Will an input that will determine to attack or to defend

// Will take a array of integers that identify opposing chess pieces that can be attacked

// Will take a array of integers that identify friendly chess pieces that can be attacked

// Pieces will be identifed as follows:

// pawn = 1

// knight = 2

// bishop = 3

// rook = 4

// queen = 5

// king = 6

/\*The numbers accoicated with each piece depict its value in this chess game. The Ai will aim to capture

higher value pieces, and avoid the capture of its own pieces (with higher value pieces getting a higher

priority).\*/

// Min/Max algorithm will determine the risk accociated with attacking

// Will find the highest value piece the Ai can attack

// Will find the lowest value piece that the Ai can use for attacking

// Will find the values of pieces it could possibly lose as a result (high to low order)

// Will minus the loss from the gain (What the AI can capture vs what the Ai can lose)

// If negative the Ai will not attack (decision sent back to main, using variable)

// If positive the Ai wil attack (decision sent back to main, using variable)

// Min/max algoritham will also return piece decided to used for defending/attacking

/\*A search algoritham\*/

// This algoritahm will help the Ai choose the shortest path to the piece it needs to attack or the piece it needs to defend

// Will take attacking/defending piece and the piece to be captured/defended

// Will return instructions as to what location the attacking/defending piece is to be moved to

// Decided location will be found using both piece allocation fucntion and the rules function

/\*Main function\*/

// Creating variables to hold image textures:

// Texture for game screen

// Texture for chess pieces

// Importing images to load onto texture:

// Connecting game screen texture to image

// Connecting chess piece textures to their specifc image

// Creating image sprites to connect to textures

// Game screen sprite

// Creating a vector of sprites, and connecting each one to a chess piece

// This makes the access of chess pieces quite easy, by storing it into a array

// Creating a string array to hold the coordinates of a chess board

// Since there are 64 coordinates on a chessboard, it would be troublesome to add each and every coordinate to the array

// That is why I wil create two string arrays with one to hold the alphabetical side and one to hold the numerical side

// A forloop and a nested for loop will be used to iterate through both arrays simultaneously

// Each matched coordinate will be inserted into chessboard array that will hold the general chessboard coordinates

// Placing pieces on the chess board

// Creating a tracking variable to access a certain chess piece

// For loop to interate chessboard array

// Creating a x and y coordinate and a location string

// Storing specifc coordinate in location string

// sending x/y coordinate and location string to allocation function

// function will determine what coordinate to use to output specifc piece

// the decided coordinate will be returned using the x and y coordinate created in this for loop

// Placing chess pieces in specifc locations:

// using track variable to access specifc piece, the chess piece is placed in a location on the board

// this will be done using .setPosition() from sfml

// chess pieces are also scaled to specifc sizes, using sfml's .setScale()

// This process is done for white pieces (in its own for loop)

// This process is done for black pieces (in its own for loop)

// Declarations for game logic:

// Creating a boolean to determine if a piece is to be moved

// Creating two values that will determine the past and present location of a chess piece

// Creating a int variable that will track specifc white and black pieces

// Outputting actual game window

// While loop that will hold general gui events while the window is running (Game logic)

// This is where all the ingame logic will be held

// Creating a variable for game event instance

// while loop that will hold game input (specifc game events)

// Closing the instance of the gui if the window is closed

// A mouse listener that will determine when a piece is grabbed

// the "past location" variable will be set to current location of the piece's location

// boolean for chess piece movement will be activated

// tracking variable will be set to grabbed piece

// A mouse listener that will determine when a piece is dropped

// boolean for chess piece movement will be deactivated

// the "new location" variable will be set to this new location of this piece's location

// Deciding if the move is legal or illigal

// Creating an integer to store the decision to be made

// sending specifc piece, decision, the old position and the new position to the chess rules function

// if the returned value is 1, the move is allowed and the opposing piece is captured

// if the captured piece is a king, the game will end

// if the returned value is 0, the move is not allowed and the chess piece is reversed to orginal location

// A boolean function that will move chess piece if it is true

// Ai player movement:

// Creating a variable to hold the decions of attacking or defending

// Creating a array of potential player pieces to capture

// Creating a araray of potential Ai pieces that are at risk of capture

// Going to Min/Max algoritahm for ai decision

// Sending decison variable

// Sending array of opposing chess pieces that can be attacked

// Sending array of Ai chess pieces that can be captured

// If decision is made to attack

// Sending specifc attacking piece and piece to be attacked to \*a search algoritham

// If deiciosn to defend

// Sending specfic defending piece and piece to be defended to \*a search algoritham

// Moving

// chosen defending/attacking piece is moved to its desired location

// If the ai chess piece has collided with a player chess piece it will be deemed captured

// if the captured piece is a king, the game will end

// Clearing gui screen

// Drawing game image

// Using for loop, drawing every chess piece to the board

// Updating/Refreshing the screen

// Returning zero to end the program