Vincent Crabtree

Programming Assignment 1

Knight’s Tour

Report

Table of Contents

Overview

The main purpose of this project was to create a program that generates a Knight's Tour on a standard eight by eight size chessboard. This means that given a starting position on the board, the program will find a path for a Knight piece that covers every spot on the board and only lands on each spot once. The first part of the program is a data entry mode, during which the user inputs starting positions on the chessboard one by one until they are satisfied, at which point they can quit. After this mode, the user is able to add new starting points, remove a starting point they entered previously or edit a starting point. At this point, a list of entries is provided to the user. In my program, this is achieved through use of a singly-linked list, where a new node is added to the list every time the user input a starting point. This means that there are always the necessary number of nodes in the list, and never more or less.

After the user is satisfied with the list of starting points, they can quit the data entry mode, at which point the Knight's Tour algorithm is invoked to find a path on the board for the first starting point. The algorithm is coded such that for the first 32 moves of the tour, the knight moves to the available spot with the least amount of exits to other available spots. An available spot is defined as a spot the knight has not landed on previously in the current tour. This method of choosing the knight's moves is called Warnsdorff's rule, and it works so well that the knight never hits a dead end on the board before completing its tour.

After the first 32 moves have been made, the algorithm utilizes a brute force depth-first search. This means that the knight will choose the first available move every turn, and continue until it either hits a dead end or finishes the tour successfully. The latter almost never happens before the former in this case, so a stack is used to store every move the knight makes in case a back track becomes necessary. Every turn, the current state of the board is pushed onto the stack

before moving on to the next turn. If the knight hits a dead end before the tour is complete, the last move it made is marked down so as not to repeat it, and the previous board state is popped off the stack, effectively backtracking the knight one move. Then, the knight tries the first available move if one exists, ignoring moves that lead to a dead end in the past. If there is no available move after popping the stack and ruling moves out, the board state is simply popped again and the knight takes another step back.

The method used for the final half of the tour simply rules out invalid paths until a successful path is found. Once a path that covers the entire board is eventually found, it is displayed to the user, along with a reminder of which starting point the tour corresponds tour. The display of the tour is simply an eight by eight board, with every spot filled by a number 0-63. That corresponds to the turn that the knight landed there. This way, the user can follow along with the moves of the knight to ensure that a proper tour was found. Once satisfied, the user can press enter to generate a tour for the next starting point in the list, until there are none left, and the program will then terminate.

Implementation

Note – Throughout the program, a file stream is opened and all of the outputs and inputs are written to a transcript file so the outcomes of the program can be viewed after termination.

Main

The program starts by telling the user to input starting positions, and shows the correct format. A while loop is used to take in the input, or tell the user if they input using an incorrect format. It uses conditional logic to decide how to proceed each iteration. If the input is "Q", the loop breaks and the program continues. If the input is a properly formatted starting position, then an insertion function is called that adds a node to the linked list that contains the coordinates of the position and the number that signifies which entry this node is (1 for first entry, 2 for second). If the entry is not "Q" or a properly formatted starting position with x and y coordinates in the range 1-8, then an error message is displayed and the loop continues without adding a node to the list.

The next part of the program is another while loop, this time acting as a menu that allows the user to add a node, remove a node or edit a node of the linked list that contains the starting positions. Conditional statements are used to see if they entered any of the above choices, or "Q" for quit. As before, if the input was not correct, then an error message is displayed and the loop continues. Depending on which option was chosen, a function is called from the functions.cpp file. These functions are "addnode", "removenode" and "editnode". Once the user inputs "Q", the loop breaks and the program continues on.

Next is the third and final while loop in the main function. This loop is used to iterate through the starting positions and perform the knight's tour for each one. It uses an index pointer to move through the list, breaking once the next node is NULL. Each iteration, the loop calls the knight's tour function by passing the current starting position, which displays the finished

product to the user. It also displays the starting point and number that corresponds to this board. Then, the user can press enter to continue the loop. After this loop breaks, the program frees memory that was allocated in the main and then terminates.

Knight's Tour (findpath.cpp)

The "findpath" function is the main function that contains the knight's tour algorithm, and is the function that is called in the third while loop of the main. It is a void functions, because it handles every aspect of the knight's tour within itself. Two main classes, a "board" class and "position" class are used in this function. The board class stores an array that acts as the chessboard, an x coordinate, y coordinate and also keeps track of how many moves have been tried from the current board, and which move was made before pushing this board to the stack. The board class also keeps track of which moves lead to dead ends from the position on the board, so the algorithm won't keep trying the same move. The position class is mainly used to keep track of the current board position separate from the board class, and it just holds coordinates in x and y.

To begin, necessary classes are instantiated, a move count is set to 0, and the board elements are all set to negative one. The instance of the position class is called current position, and it is set to be the starting position that was passed to the function. Next is the main while loop, and each iteration acts as a turn for the knight. At the beginning, a new board instance is created, and it is filled with data from the previous boards to be on track. A dead end flag is reset as well, and then the program enters one of two main if statements. These statements divide the tour into halves, with the first utilizing Warnsdorff's rule, and the second using a brute force approach. The first half starts with a for loop that checks every move the knight could make, and

either rules it as invalid or records the number of exits that move has. This information is stored in an array called “exitcount”, with an element that corresponds to each move.

If a move is considered to be valid, the dead end flag is set to 1, and if it remains 0 after the moves are checked, then the program goes into the backtrack section. In this section, the current board is deleted, and the previous is popped from the stack and set as the new current board. Also, the move that was made is set as a bad move in the board’s bad move array, to avoid a scenario where the program tries the same move endlessly. Finally, the move count is decremented, and the current position is updated. If the spot is not a dead end, then the “findlowest” function is called to determine which move has the least amount of exits, and the result is stored in a variable “z”. After this, the board elements are set for the position, and it is pushed to the stack. Then, the function “updatecurrentpos” is called by passing z, and it sets the current position to the new position according to the chosen move stored in z. The move count then increments and the next iteration begins.

If the tour is past the first half, the algorithm remains largely the same. The main difference is that instead of counting the number of exits for a possible move, the first for loop simply checks if a move is valid and records the result in an array called “islegal”. Then, if the position is not a dead end, the first legal move available is chosen as the next move, thus eliminating all efficiency that came with Warnsdorff’s rule.

When the move count reaches 63, a “displayboard” function is called to output the finished tour to the user, then all allocated memory is freed and the function returns. At this point the main program loops through the tours as described in the previous section, and the bulk of the code has been covered.

Function Descriptions (in order of appearance in functions.cpp)

- Void addpos() – This function adds a new node to the linked list of starting positions, and fills in the elements using the data passed to the function.
- Void displaylist() – Outputs the full list of starting positions to the user.
- Void addnode() – Adds a new node to the list similar to addpos(), but with some added output and input handling
- Startpos searchbynum() – Finds the node of the list that corresponds to the number passed to the function
- Void removenode() – Removes the node (and frees its memory) in the list of starting position that corresponds to a number entered by the user. Contains input and output as well.
- Void editnode() – Allows the user to choose a node in the list and edit the data it contains.
- Void freelist() – Deallocates all of the memory previously allocated to the linked list of starting positions passed to the function.
- Int howmanyexits() – Determines the number of viable exits a spot with coordinates passed the function has on a board passed to the function. Then, the number of exits is returned.
- Int findlowest() – Returns the number of the element of a passed array that has the lowest value stored within.
- Void updatecurrentpos() – Given an object of position class and a number that denotes which move is to be made, this function adjusts the coordinates of the position class according to the move made.

- Void displayboard() – Given an object of board class, this function displays the elements of the board array to the user in a neatly formatted manner.

Example Transcript

Please enter a list of initial positions in the format 'X,Y' one at a time.

Enter 'Q' when you are finished:

1,5

7,4

3,3

9,5

Invalid input, please try again:

huiwerf iuw

Invalid input, please try again:

0

Invalid input, please try again:

Q

1: 1,5

2: 7,4

3: 3,3

Would you like to make changes to the list?

Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.

If finished, enter 'Q' to continue.

1

Please enter the starting position as before:

4,1

1: 1,5

2: 7,4

3: 3,3

4: 4,1

Would you like to make changes to the list?

Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.

If finished, enter 'Q' to continue.

2

1: 1,5

2: 7,4

3: 3,3

4: 4,1

Please enter the number of the entry you wish to remove:

3

1: 1,5

2: 7,4

3: 4,1

Would you like to make changes to the list?

Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.

If finished, enter 'Q' to continue.

6

1: 1,5

2: 7,4

3: 4,1

Would you like to make changes to the list?

Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.

If finished, enter 'Q' to continue.

5yh

1: 1,5

2: 7,4

3: 4,1

Would you like to make changes to the list?

Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.

If finished, enter 'Q' to continue.

3

1: 1,5

2: 7,4

3: 4,1

Please enter the number of the entry you wish to edit:

1

Please enter the new starting position in the same format as before:

1,6

1: 1,6

2: 7,4

3: 4,1

Would you like to make changes to the list?
Enter '1' to add an entry, '2' to remove one, or '3' to edit an entry.
If finished, enter 'Q' to continue.

Q

42 1 32 17 46 3 34 19

31 16 41 2 33 18 47 4

```
 0  43  38  59  40  45  20  35

15  30  53  44  37  58   5  48

54  11  60  39  52  63  36  21

29  14  55  62  57  24  49   6

10  61  12  27   8  51  22  25

13  28   9  56  23  26   7  50
```

1: Starting Position: 1,6

Press enter to continue...

```
34  31   8  29  38  15  10  13

 7  28  33  36   9  12  39  16

32  35  30  57  48  37  14  11

27   6  55  46  63  58  17  40
```

54 45 62 49 56 47 0 21

5 26 53 44 59 20 41 18

52 61 24 3 50 43 22 1

25 4 51 60 23 2 19 42

2: Starting Position: 7,4

Press enter to continue...

4 35 22 41 6 37 24 43

21 48 5 36 23 42 7 38

34 3 58 47 40 51 44 25

57 20 49 52 59 46 39 8

2 33 56 63 50 53 26 45

19 30 17 60 55 62 9 12

16 1 32 29 14 11 54 27

31 18 15   0 61 28 13 10

3: Starting Position: 4,1

Press enter to continue...

Source Code

```
//main.cpp
#include <iostream>
#include <stdio.h>
#include <string>
#include <fstream>
#include "startpos.h"
#include "functions.h"
#include "stack.h"

int main()
{
	//Declarations
	std::string inpt;
	startpos *ptr;
	startpos *head = new startpos;	//Starting node for the linked list
	stack *mystack = new stack;	//Creates a stack instance
	std::ofstream tscript;


	//Set up for later logic
	head->setnextpos(NULL);

	//This just erases the old data on the file if it already exists
	tscript.open("transcript.txt", std::ofstream::trunc);
	tscript.close();

	//Opens up output file
	tscript.open("transcript.txt", std::ofstream::app);

	//Resets the screen to look neater
	std::cout << "\033[H\033[2J";

	//Starting prompt
	std::cout << "Please enter a list of initial positions in the format 'X,Y'";
	tscript << "Please enter a list of initial positions in the format 'X,Y'";
	//For transcript
	std::cout << " one at a time.\nEnter 'Q' when you are finished:\n\n";
	tscript << " one at a time.\nEnter 'Q' when you are finished:\n\n";
	//For transcript


	//Loop for taking input
	while(1)
	{
		//Takes input
		std::getline(std::cin, inpt);
		tscript << inpt + "\n";

		//Finished with entry
		if(inpt == "Q")
		{
			break;
		}

		//Incorrect input handling
```

```
		if(inpt.length()<3 || (inpt[1] != ',') || (inpt[0]>'8'||inpt[0]<'1') ||
((inpt[2]>'8'||inpt[2]<'1')) || inpt.length()>3)
		{
			std::cout << "\nInvalid input, please try again:\n\n";
			tscript << "\nInvalid input, please try again:\n\n";
			continue;
		}

		//Calls add funtion (-48 on inputs to adjust ascii value)
		addpos(head,inpt[0]-48,inpt[2]-48);

	}

	std::cout << "\n\n";
	tscript << "\n\n";

	//Loop for allowing the user to change the list
	while(1)
	{
		//Resets the screen to look neater
		std::cout << "\033[H\033[2J";
		tscript.close();	//Closes this stream so the function can open its own
without issues
		displaylist(head);
		tscript.open("transcript.txt", std::ofstream::app);	//Reopens when done,
		//This method avoids passing the stream as an argument if that's even
possible

		//Display instructions for editing the list
		std::cout << "\n\nWould you like to make changes to the list?\n";
		std::cout << "Enter '1' to add an entry, '2' to remove one, or '3' to edit
an entry.\n";
		std::cout << "If finished, enter 'Q' to continue.\n\n";
		tscript << "\n\nWould you like to make changes to the list?\n";
		tscript << "Enter '1' to add an entry, '2' to remove one, or '3' to edit an
entry.\n";
		tscript << "If finished, enter 'Q' to continue.\n\n";

		//Takes input
		std::getline(std::cin, inpt);
		tscript << inpt + "\n";

		if(inpt == "1")
		{
			tscript.close();
			addnode(head);
			tscript.open("transcript.txt", std::ofstream::app);
		}

		else if(inpt == "2")
		{
			tscript.close();
			removenode(head);
			tscript.open("transcript.txt", std::ofstream::app);
		}

		else if(inpt == "3")
		{
```

```
                        tscript.close();
                        editnode(head);
                        tscript.open("transcript.txt", std::ofstream::app);
                }

                if(inpt == "Q")
                {
                        break;
                }

        }


        //Sets the index ptr to the head of the list
        ptr = head;

        //Loop for dipslaying the final boards
        while(1)
        {
                //Resets the screen to look neater
                std::cout << "\n\n\n";
                std::cout << "\033[H\033[2J";
                tscript << "\n\n\n";

                //Breaks once finished
                if(ptr->getnextpos() == NULL)
                        break;

                //Points index to next starting position and runs the Knight's Tour
algorithm
                else
                {
                        ptr = ptr->getnextpos();

                        //Calls Tour algorithm
                        tscript.close();
                        findpath(ptr, mystack);
                        tscript.open("transcript.txt", std::ofstream::app);

                        //Display for each board
                        std::cout << "\n\n" << ptr->getnum() << ":  " << "Starting Position:
" << ptr->getxcoord() << "," << ptr->getycoord();
                        tscript << "\n\n" << ptr->getnum() << ":  " << "Starting Position:
" << ptr->getxcoord() << "," << ptr->getycoord();
                        std::cout << "\n\nPress enter to continue...";
                        tscript << "\n\nPress enter to continue...";
                        std::getline(std::cin, inpt);
                }
        }

        tscript.close();     //Closes file stream
        freelist(head);             //Deallocates list memory
        delete mystack;

        return 0;
}
```

```
//board.h
#include "position.h"

#ifndef board_H
#define board_H

class board
{
	private:
		int curxcoord;			//Stores knight's current x coord
		int curycoord;			//Stores knight's current y coord
		int num;			//Stores number of paths tried for this position
so far
		int movemade;

	public:
		board();
		~board();
		int boarda[8][8];	//Array that stores the board state

		int badmove[8];			//Used to mark off moves that have proven to
lead to dead ends

		//Getters
		int getcurxcoord();
		int getcurycoord();
		int getnum();
		int getmovemade();

		//Setters
		void setcurxcoord(int input);
		void setcurycoord(int input);
		void setnum(int input);
		void setmovemade(int input);


};

#endif


//findpath.cpp
#include <iostream>
#include <string>
#include <stdlib.h>
#include <fstream>
#include "functions.h"
#include "position.h"
#include "stack.h"
#include "board.h"



//Main function for finding a path for the Knight that covers the whole board
void findpath(startpos *start, stack *mystack)
{
	position *currentpos, *temppos;	//Will be used to point to current position
object
```

```
	int move=0, count = 0, deadendcheck = 0;
	int exitcount[8], islegal[8];                                          //Used
to store number of exits each possible move has (
	board *currentboard, *prevb;
	int x, y, z,  flag=0, backflag = 0;
	int movesx[8] = { -2, -1, 1, 2, 2, 1, -1, -2 }; //Holds x values for all possible
moves
	int movesy[8] = { 1, 2, 2, 1, -1, -2, -2, -1 }; //Same but y values

	temppos = new position;
	currentboard = new board;
	prevb = new board;


	//Sets up the board array to begin with since it can't be initialized in a class
apparently
	for(x=0;x<8;x++)
	{
		for(y=0;y<8;y++)
		{
			currentboard->boarda[y][x] = -1;
		}
	}

	currentpos = new position;                                    //Creates an
instance for the start position
	currentpos->setxcoord(start->getxcoord()-1);    //Adjusts for array starting at 0
	currentpos->setycoord(7-(start->getycoord()-1));//Same


	//Loop for determining next spot until all are covered
	while(1)
	{
		//Skips first time and checks to make sure a new board is needed (not a
backtrack)
		if((flag != 0) && (backflag == 0))
		{
			currentboard = new board;
			*currentboard =*prevb;

			//Resets currentboard's bad moves
			for(x=0;x<8;x++)
			{
				currentboard->badmove[x] = 0;
			}
		}

		flag = 1;
		deadendcheck = 0; //resets

		//Update board
		currentboard->boarda[currentpos->getycoord()][currentpos->getxcoord()] =
count;

		//This will be for the first half that uses Warnsdoff rule
		if(count < 32)
		{
			//Loop to check for moves viability and exit count
```

```
                    for(x=0;x<8;x++)
                    {
                           //Sets up temporary object to test moves on without affecting
current position
                           *temppos = *currentpos;
                           temppos->setxcoord(temppos->getxcoord() + movesx[x]);
                           temppos->setycoord(temppos->getycoord() + movesy[x]);

                           //Checks to make sure the move is valid
                           if( (temppos->getxcoord() >= 0 && temppos->getxcoord() <= 7)
&& (temppos->getycoord() >= 0 && temppos->getycoord() <= 7)
                                  && (currentboard->boarda[temppos->getycoord()][temppos-
>getxcoord()] == -1) && (currentboard->badmove[x] != 1))
                           {
                                  deadendcheck = 1; //Sets to show not dead end
                                  exitcount[x] = howmanyexits(temppos-
>getxcoord(),temppos->getycoord(),currentboard->boarda);
                           }
                           else
                           {
                                  exitcount[x] = 10; //Number 10 will signify that this
move is invalid
                           }
                    }

                    //If the current position is a deadend
                    if(deadendcheck == 0)
                    {
                           //Deletes currentboard and pops previous in order to backtrack
                           delete currentboard;
                           currentboard = mystack->pop();
                           currentpos->setxcoord(currentboard->getcurxcoord());
                           currentpos->setycoord(currentboard->getcurycoord());
                           currentboard->badmove[currentboard->getmovemade()] = 1;//Sets
this move in the array so it is not repeated if this board is popped
                           backflag = 1; //sets to show that this was a backtrack, so no
new board is needed at top of the loop
                           count--;
                           continue;

                    }
                    //If not
                    else
                    {
                           z = findlowest(exitcount);

                           //Sets the position on the board and preps for pushing to
stack
                           currentboard->setcurxcoord(currentpos->getxcoord());
                           currentboard->setcurycoord(currentpos->getycoord());
                           currentboard->setmovemade(z);
                           *prevb = *currentboard;

                           mystack->push(currentboard);

                           //Sets up new position
                           updatecurrentpos(currentpos, z);
                           backflag = 0;
```

```
                        count++;

                }

        }



    //*************************************************************************

    //*************************************************************************

    //*************************************************************************
        //This section is for the second half where Warnsdoff rule is no longer
used
        else if(count < 63)
        {
                //Loop to check for moves viability and exit count
                for(x=0;x<8;x++)
                {
                        //Sets up temporary object to test moves on without affecting
current position
                        *temppos = *currentpos;
                        temppos->setxcoord(temppos->getxcoord() + movesx[x]);
                        temppos->setycoord(temppos->getycoord() + movesy[x]);

                        //Checks to make sure the move is valid
                        if( (temppos->getxcoord() >= 0 && temppos->getxcoord() <= 7)
&& (temppos->getycoord() >= 0 && temppos->getycoord() <= 7)
                                && (currentboard->boarda[temppos->getycoord()][temppos-
>getxcoord()] == -1) && (currentboard->badmove[x] != 1))
                        {
                                deadendcheck = 1; //Sets to show not dead end
                                islegal[x] = 1; //Now used to
                        }
                        else
                        {
                                islegal[x] = 0; //Number 0 will signify that this move
is invalid
                        }
                }

                //If the current position is a deadend
                if(deadendcheck == 0)
                {
                        //Deletes currentboard and pops previous in order to backtrack
                        delete currentboard;
                        currentboard = mystack->pop();
                        currentpos->setxcoord(currentboard->getcurxcoord());
                        currentpos->setycoord(currentboard->getcurycoord());
                        currentboard->badmove[currentboard->getmovemade()] = 1;//Sets
this move in the array so it is not repeated if this board is popped
                        backflag = 1; //sets to show that this was a backtrack, so no
new board is needed at top of the loop
                        count--;
                        continue;
```

```
                        }
                        //If not
                        else
                        {
                                //Simply chooses the first viable move to try next
                                for(x=7;x>=0;x--)
                                {
                                        if(islegal[x] == 1)
                                        {
                                                z = x;
                                                break;
                                        }
                                }

                                //Sets the position on the board and preps for pushing to
stack
                                currentboard->setcurxcoord(currentpos->getxcoord());
                                currentboard->setcurycoord(currentpos->getycoord());
                                currentboard->setmovemade(z);
                                *prevb = *currentboard;

                                mystack->push(currentboard);

                                //Sets up new position
                                updatecurrentpos(currentpos, z);
                                backflag = 0;
                                count++;

                        }
                }

                //Breaks the loop once finished
                else
                        break;
        }


        displayboard(currentboard);
        mystack->freestack();
        delete prevb;
        delete currentboard;
        delete currentpos;
        delete temppos;
}



//board.cpp
#include "board.h"

board::board()
{
        int boarda[8][8] = {{-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
```

```
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1, },
                                        {-1, -1, -1, -1, -1, -1, -1, -1 }};

	this->num = 0;
}

board::~board()
{
}


//Getters
int board::getcurxcoord()
{
	return this->curxcoord;
}

int board::getcurycoord()
{
	return this->curycoord;
}

int board::getnum()
{
	return this->num;
}

int board::getmovemade()
{
	return this->movemade;
}


//Setters
void board::setcurxcoord(int input)
{
	this->curxcoord = input;
}

void board::setcurycoord(int input)
{
	this->curycoord = input;
}

void board::setnum(int input)
{
	this->num = input;
}

void board::setmovemade(int input)
{
	this->movemade = input;
}
```

```
//functions.h
#ifndef functions_H
#define funtions_H

#include "startpos.h"
#include "stack.h"

//Function definitions
void *addpos(startpos *head, int x, int y);
void displaylist(startpos *head);
void addnode(startpos *head);
void removenode(startpos *head);
void editnode(startpos *head);
startpos *searchbynum(startpos *head, double num);
void freelist(startpos *head);
void findpath(startpos *start, stack *mystack);
int howmanyexits(int xcoord, int ycoord, int board[8][8]);
int findlowest(int list[8]);
void updatecurrentpos(position *current, int num);
void displayboard(board *currentboard);

#endif


//functions.cpp
#include <stdio.h>
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
#include "startpos.h"
#include "position.h"
#include "board.h"

void *addpos(startpos *head, int x, int y)
{
	startpos *node;
	int count=1;

	node = head;


	//Finds the end of the list
	while(node->getnextpos() != NULL)
	{
		node = node->getnextpos();
		count++;
	}

	//Creates new node in list
	startpos *newnode = new startpos;

	//Fills new node with user input and spot on list
	newnode->setxcoord(x);
	newnode->setycoord(y);
	newnode->setnum(count);
	newnode->setnextpos(NULL);
	node->setnextpos(newnode);
```

```
	return 0;

}



void displaylist(startpos *head)
{
	startpos *node;
	std::ofstream tscript;

	tscript.open("transcript.txt", std::ofstream::app);



	if(head->getnextpos() == NULL)
	{
		std::cout << "\n\nThe list is empty!\n\n";
		tscript.close();
		return;
	}

	node = head->getnextpos();

	//Loop for displaying each node in list
	while(1)
	{
		printf( "%d:   %d,%d\n\n", node->getnum(), node->getxcoord(), node-
>getycoord());
		tscript << node->getxcoord() << ":   " << node->getxcoord() << "," <<
node->getycoord() << "\n\n";
		if(node->getnextpos() == NULL)
		{
			break;
		}

		else
		{
			node = node->getnextpos();
		}
	}

	tscript.close();
}



void addnode(startpos *head)
{
	std::string inpt;
	std::ofstream tscript;

	tscript.open("transcript.txt", std::ofstream::app);

	//Resets the screen to look neater
	std::cout << "\033[H\033[2J";
```

```
	std::cout << "Please enter the starting position as before:\n\n";
	tscript << "Please enter the starting position as before:\n\n";

	//Loop in case of incorrect entry
	while(1)
	{
		//Takes input
		std::getline(std::cin, inpt);

		tscript<< inpt + "\n\n";

		//Incorrect input handling
		if(inpt.length()<3 || (inpt[1] != ',') || (inpt[0]>'8'||inpt[0]<'1') ||
((inpt[2]>'8'||inpt[2]<'1')) || inpt.length()>3)
		{
			std::cout << "\nInvalid input, please try again:\n\n";
			tscript << "\nInvalid input, please try again:\n\n";
			continue;
		}

		else
			break;
	}


	//Calls add funtion (-48 on inputs to adjust ascii value)
	addpos(head,inpt[0]-48,inpt[2]-48);

	tscript.close();
}



//Simply returns a pointer to the element of the list that matches the number passed if a
match exists
startpos *searchbynum(startpos *head, double num)
{
	startpos *node;

	node = head->getnextpos();

	while(1)
	{
		if(num == node->getnum())
		{
			return node;
		}

		else if(node->getnextpos() == NULL)
		{
			std::cout << "\n\nNot Found!";
			return NULL;
		}

		else
		{
			node = node->getnextpos();
		}
```

```
	}

}


void removenode(startpos *head)
{
	startpos *node1, *node2;
	std::string entry;
	double entrydoub;
	std::ofstream tscript;

	//Resets the screen to look neater
	std::cout << "\033[H\033[2J";
	displaylist(head);
	tscript.open("transcript.txt", std::ofstream::app);

	node2 = head;

	std::cout << "Please enter the number of the entry you wish to remove:\n\n";
	tscript	  << "Please enter the number of the entry you wish to remove:\n\n";

	//Loop in case of incorrect entry
	while(1)
	{
		std::getline(std::cin, entry);

		tscript << entry + "\n\n";
		try
		{
			entrydoub = std::stod(entry);
		}
		catch(std::exception &e)
		{
			std::cout << "\n\nInvalid input, please try again:\n\n";
			tscript << "\n\nInvalid input, please try again:\n\n";
			continue;
		}


		node1 = searchbynum(head, entrydoub);

		if(node1 == NULL)
		{
			std::cout << " Try Again:\n\n";
			tscript << " Try Again:\n\n";
			continue;
		}
		else
			break;
	}

	//Loop for setting up previos node pointer
	while(node2->getnextpos() != node1)
	{
		node2 = node2->getnextpos();
```

```
	}

	node2->setnextpos(node1->getnextpos());
	delete node1;

	if(node2->getnextpos() == NULL)
	{
		return;
	}

	//Fixes numbers after removed item
	while(1)
	{
		node2 = node2->getnextpos();
		node2->setnum((node2->getnum())-1);

		if(node2->getnextpos() == NULL)
			break;
	}

	tscript.close();
}



//Allows the user to change an entry in the list
void editnode(startpos *head)
{
	startpos *node;
	std::ofstream tscript;
	std::string entry;
	double entrydoub;

	//Resets the screen to look neater
	std::cout << "\033[H\033[2J";
	displaylist(head);
	tscript.open("transcript.txt", std::ofstream::app);

	std::cout << "Please enter the number of the entry you wish to edit:\n\n";
	tscript	  << "Please enter the number of the entry you wish to edit:\n\n";

	//Loop in case of incorrect input
	while(1)
	{
		std::getline(std::cin, entry);
		tscript << entry + "\n\n";

		try
		{
			entrydoub = std::stod(entry);
		}
		catch(std::exception &e)
		{
			std::cout << "\n\nInvalid input, please try again:\n\n";
			tscript << "\n\nInvalid input, please try again:\n\n";
			continue;
		}
```

```
		node = searchbynum(head, entrydoub);

		if(node == NULL)
		{
			std::cout << " Try Again:\n\n";
			tscript << " Try Again:\n\n";
			continue;
		}
		else
			break;

	}

	//Loop for second entry
	std::cout << "\n\nPlease enter the new starting position in the same format as
before:\n\n";
	tscript << "\n\nPlease enter the new starting position in the same format as
before:\n\n";
	while(1)
	{
		std::getline(std::cin, entry);
		tscript << entry + "\n\n";

		//Incorrect input handling
		if(entry.length()<3 || (entry[1] != ',') || (entry[0]>'8'||entry[0]<'1') ||
((entry[2]>'8'||entry[2]<'1')) || entry.length()>3)
		{
			std::cout << "\nInvalid input, please try again:\n\n";
			tscript << "\nInvalid input, please try again:\n\n";
			continue;
		}

		else
			break;


	}

	//Finally, changes the node according to the input
	node->setxcoord(entry[0]-48);
	node->setycoord(entry[2]-48);

	tscript.close();

}



//Deallocates the memory used by the list
void freelist(startpos *head)
{
	startpos *node, *temp;

	node = head->getnextpos();

	while(node != NULL)
	{
		temp = node;
```

```
			node = node->getnextpos();
			delete temp;
		}

		delete head;
}



//Returns the number of viable exits the passed spot on the board has for the knight
int howmanyexits(int xcoord, int ycoord, int board[8][8])
{
		int x, count = 0;
		int movesx[8] = { -2, -1, 1, 2, 2, 1, -1, -2 }; //Holds x values for all possible
moves
		int movesy[8] = { 1, 2, 2, 1, -1, -2, -2, -1 }; //Sam for y values

		for(x=0;x<8;x++)
		{
			if(board[ycoord + movesy[x]][xcoord + movesx[x]] == -1 && (ycoord +
movesy[x] <= 7)
				&& (ycoord + movesy[x] >= 0) && (xcoord + movesx[x] <= 7) && (xcoord
+ movesx[x] >= 0))
			{
				count++;
			}
		}


		return count;
}



//Finds element with the lowest value in an array (or one of them if there are ties)
int findlowest(int list[8])
{
		int x, low;

		low = 0;

		for(x=0;x<8;x++)
		{
			if(list[x] >= 9)
			{
				continue;
			}

			if(list[x] <= list[low])
			{
				low = x;
			}
		}


		return low;
}
```

```
//Adjusts the current position based on the number given
void updatecurrentpos(position *current, int num)
{
	int movesx[8] = { -2, -1, 1, 2, 2, 1, -1, -2 }; //Holds x values for all possible
moves
	int movesy[8] = { 1, 2, 2, 1, -1, -2, -2, -1 }; //Sam for y values

	int x;

	for(x=0;x<8;x++)
	{
		if(x == num)
		{
			current->setxcoord(current->getxcoord() + movesx[x]);
			current->setycoord(current->getycoord() + movesy[x]);
			return;
		}
	}
}



//Displays the board
void displayboard(board *currentboard)
{
	int x, y, count;
	std::ofstream tscript;

	//Opens stream to transcript
	tscript.open("transcript.txt", std::ofstream::app);

	for(x=0;x<8;x++)
	{
		for(y=0;y<8;y++)
		{
			printf("%2d  ", currentboard->boarda[x][y]);
			tscript << std::setw(2) << currentboard->boarda[x][y] << "  ";
		}

		std::cout << "\n\n";
		tscript << "\n\n";
	}

	//Closes transcript stream
	tscript.close();
}



//position.h
#ifndef position_H
#define position_H

//This is just a class used to store a board location easily
class position
{
```

```
	private:
		int xcoord;
		int ycoord;
		int count; //keeps track of how many movement options have been tried in
findpath function

	public:
		position();
		~position();

		//Getters
		int getxcoord();
		int getycoord();
		int getcount();

		//Setters
		void setxcoord(int input);
		void setycoord(int input);
		void setcount(int input);

};

#endif


//position.cpp
#include "position.h"

position::position()
{
	this->count = 0;
}

position::~position()
{

}


//Getters
int position::getxcoord()
{
	return this->xcoord;
}

int position::getycoord()
{
	return this->ycoord;
}

int position::getcount()
{
	return this->count;
}


//Setters
void position::setxcoord(int input)
```

```
{
	this->xcoord = input;
}

void position::setycoord(int input)
{
	this->ycoord = input;
}

void position::setcount(int input)
{
	this->count = input;
}


//stack.h
#include "position.h"
#include "board.h"

#ifndef stack_H
#define	stack_H

class stack
{
	private:
		board *stacka[65];
		int top;

	public:
		stack();
		~stack();
		board *pop();
		void push(board *input);
		int isempty();
		void freestack();

};


//stack.cpp
#include <iostream>
#include "stack.h"
#include "board.h"


stack::stack()
{
	this->top = -1;			//Initializes top to -1 to start as empty
}

stack::~stack()
{

}

//Checks if stack is empty
int stack::isempty()
{
```

```
	if(this->top <= -1)
		return 1;
	else
		return 0;
}

//Pops the top object off of the stack if not empty and moves top to next object
board *stack::pop()
{
	int temp;


		temp = this->top;
		this->top--;
		return this->stacka[temp];

}

//Pushes a given position onto the stack if not full
void stack::push(board *input)
{
	if(top >= 65)
	{
		std::cout << "The stack is full!!";
		return;
	}

	else
	{
		this->top++;
		this->stacka[top] = input;
	}

}

void stack::freestack()
{
	int x, tc1 = 0;
	board *ptr;

	while(this->top > -1)
	{
		ptr = this->stacka[top];
		tc1++;
		delete ptr;
		top--;
	}
}


//starpos.h
#ifndef startpos_H
#define startpos_H

class startpos
{
	public:
```

```
            startpos();          //Constructor
            ~startpos();  //Destructor

            //Getters
            int getxcoord();
            int getycoord();
            int getnum();
            startpos *getnextpos();

            //Setters
            void setxcoord(int input);
            void setycoord(int input);
            void setnum(int input);
            void *setnextpos(startpos *input);


      private:

            //Attributes
            int xcoord;
            int ycoord;
            int num;
            startpos *nextpos;



};


#endif


//starpos.cpp
#include "startpos.h"

startpos::startpos()
{

}

startpos::~startpos()
{

}


//Getters
int startpos::getxcoord()
{
      return startpos::xcoord;
}

int startpos::getycoord()
{
      return startpos::ycoord;
}

int startpos::getnum()
```

```
{
	return startpos::num;
}

startpos *startpos::getnextpos()
{
	return this->nextpos;
}


//Setters
void startpos::setxcoord(int input)
{
	startpos::xcoord = input;
}

void startpos::setycoord(int input)
{
	startpos::ycoord = input;
}

void startpos::setnum(int input)
{
	startpos::num = input;
}

void *startpos::setnextpos(startpos *input)
{
	startpos::nextpos = input;
	return 0;
}
```