Knight’s Tour Report

Andrew Chittick

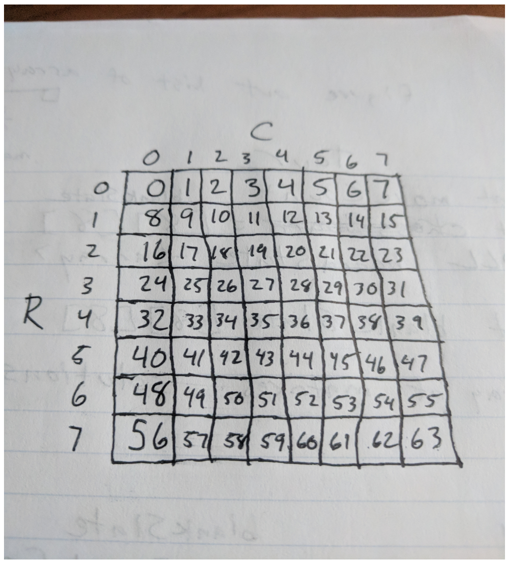
2/14/18

The knight’s tour problem involves a knight’s movement on a chess board. To solve the problem one must take a knight from any given position on a chess board to every square on that board without visiting the same square twice. The chess board is 8 square rows and 8 square columns. A knight can move forward/back two squares and over to either side one square. A knight can also move two squares to either side and one square forward or back. Their are heuristics for solving this problem developed by J.C. Warnsdoff called the Warnsdoff rule. Warnsdoff rule states that to solve the knights tour problem one should choose the square with the fewest exits from the beginning to solve this problem successfully, often on the first attempt.

My knight’s tour program solves the knight’s tour problem for as many starting positions as the user chooses. The user is able to add, edit, and delete as many positions as they want. After the user has inputed all the positions they desire they may select the begin knight’s tour option from the main menu. Once the begin knight’s tour option is selected the program iterates through the user’s list of starting positions, and giving the solution to each position as it goes through the list. The starting position and output is displayed on the screen and written to an output file named knightsTourOutput.dat. After the list has been fully iterated through the program ends. I wrote this in c++. My program has a main, user class, and tour class. There is a separate header file for user and tour. There is a makefile for compiling.

Makefile is used to make compiling the program easier. Make run will run the program, and make clean will remove the .o files after you are completed. Main has dependencies of user and tour. User has no dependencies. Tour has no dependencies. The line CXXFLAGS = -std=c++11 tells the system to compile with c++11, enabling c++11 features. The c++11 feature used in this program was forward_list which is a singly linked list in the c++11 std library. This feature is used in main for storing the users initial starting positions. The Tesla server at IUPUI does not have c++11 enabled by default making this line in the makefile necessary.

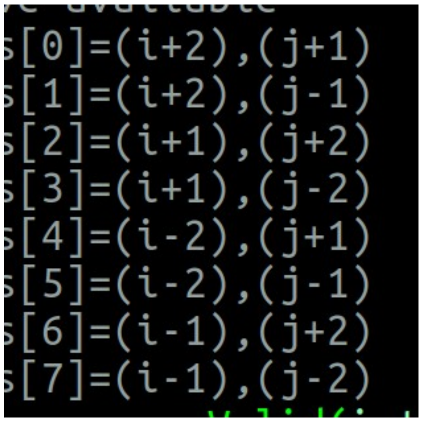
The user class contains both the main menu and the introduction to the program to the user. The main job of User Class is to communicate with the user, and get/verify user input. User::welcome displays the introduction to the program and shows them a sample chess board with the starting positions listed 0-63. User::menu displays the main menu to the user. The main menu gives the options of beginning the tour with the chosen positions, add another starting position, modify a starting position, delete a starting position, and exit. User::setPosition asks the user for a starting position, and gets the users input as a string. It then converts that string to an integer. If the input was not numerical it will convert it to a zero. It then checks if the input is less than 64, and greater than -1. If the input is less than 64 and greater than -1 it is a valid starting position and saved to the variable position. If not the default value of zero is assigned to the variable position. User::getPosition simply returns the position saved in setPositon. User::setPosChoice asks the user to choose a position (used for delete and modify). It displays a range of the choices available and asks for their choice. Input is taken as a string and then converted to an integer. If the choice does not lie within the range "Invalid choice, try again." is displayed and setPosChoice runs again. SetPosChoice will run until a choice is made within the range of choices. User::getPosChoice simply returns the choice made in setPosChoice.



		C							
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
	1	8	9	10	11	12	13	14	15
	2	16	17	18	19	20	21	22	23
	3	24	25	26	27	28	29	30	31
R	4	32	33	34	35	36	37	38	39
	5	40	41	42	43	44	45	46	47
	6	48	49	50	51	52	53	54	55
	7	56	57	58	59	60	61	62	63

The main sends to both the user class and the tour class. It also contains userList, a forward_list of user positions. A forward_list is a singly linked list in the c++11 std library. Main begins by introducing the user to the game through the user class. It then gets the first user entered position through the user class. It then saves that position to the front of the singly linked list, userList. Next main opens the file "knightsTourOutput.dat" which is the output file that the solutions will eventually be saved to. If the file is not opened "Unable to open file" is displayed to the user. Main then enters a while loop. Within the while loop the user is shown their starting positions to this point and the main menu. The program remains in the while loop until the user selects begin knight's tour or exit from the main menu. Inside the while loop is a switch with all of the main menu choices as the cases. If the user selects add a starting position (choice 2) it sends to user to get a new positions. The valid position is returned and saved to userList. If the user chooses to modify a starting position (choice 3) the user is shown the list of starting positions (userList). It then asks them for a position to modify and sends to class user to get/verify the users input. It then asks what they want to change it to, and sends to user to get/verify a new position. With the verified user position to modify and new position, it checks if it is the front element. If it is the front element it just deletes the front element and adds the new position to the front of the list, effectively modifying the front element. If it is not the front element it iterates the userList to the node before the one to be modified. It then adds the new position and deletes the position after it (the old position). This is the same as modifying a position. Delete starting position works the same way as modify starting position without adding in a new (modified) position. If case 5, exit, is selected the while loop is exited and the program ends. If case 1, begin tour, is selected it first checks if the output file was able to open. If it was able to open it iterates through the userList. As it iterates through the list each starting position in the userList is displayed to the user and output file. It then sends the position to tour.cpp to get the solution. It then writes the solution to the file. Then it repeats until all solutions in userList are displayed and written to the file. Then the while loop is exited and the program is over. If the file was not able to open case 1 does the same thing as if it were able to open minus writing to the output file. The function startingPositions takes the userList and iterator as parameters. Its job is to simply display a numbered list of user entered positions to the user.

The tour class is what runs the actual knights tour for any given, valid starting position (0-63). It takes a single starting position as a parameter. Tour contains a struct called boardState. BoardState is the state of the board. It consists of an array board[8][8] which is the representation of the chess board. It is initially set to all 1's. It contains moveNumber which is the number of the move about to be made (0-63). MovesAvailable is an integer of the amount of valid moves available from the current position (0-7). ValidMoves[8] is an array of ones and zeros, for each associated position, it contains a 1 if that move is valid and a 0 if it is either already been used or not a valid position on the chess board. MovesMade[8] is also an array with the same associated positions as validMoves[8]. It contains a one in the position if that move has been made and a zero if not. Ccolumn and cRow contains the column and row that the knight is currently in.



```
s[0]=(i+2),(j+1)
s[1]=(i+2),(j-1)
s[2]=(i+1),(j+2)
s[3]=(i+1),(j-2)
s[4]=(i-2),(j+1)
s[5]=(i-2),(j-1)
s[6]=(i-1),(j+2)
s[7]=(i-1),(j-2)
```

TakeTour takes position as a parameter and controls the flow of the tour process. It starts by taking the position and turning it into a row integer and column integer. It then marks the board with moveNumber (0) in the appropriate row and column to simulate the users starting position. It then stores the updated boardState to a doubly linked list knightTour. It then enters a while loop until the moveNumber reaches 64 (the tour has been competed). Inside the while loop it sends to setValidMoves which sets the validMoves array and sets movesAvailable. It then sends to nextMove to get an index for the next move to be made. If the index returned is 187, there are no moves available and it backtracks. Backtracking consists of saving the current movesMade array to a pastMovesMade array. It then deletes the last element of the list knightTour and updates the board state to the one previously stored, this is like moving the board state back in time one move while storing the move that was made last time. If there are more moves available than moves already made (moves that haven't been tried yet are still available) it sends it to altMove to get an alternative move. If not it keeps backtracking. AltMove gives you an index of a move to be made, and this is the end of the if statement for no moves available. If a valid move was given initially movesMade array is cleared. Now either way we have a valid nextMoveIndex, which will tell us the next move to be made. It then makes that move and updates boardState accordingly. Once the amount of movesMade reaches 64 a solution has been reached and it exits the while loop. After it exits the while loop it displays the solution to the console and stores the solution to an array solution[]. TakeTour returns this solution and this starting position has been solved.

SetValidMoves sends to moveValid to get all the valid moves from the position. It then checks each move that is valid by sending it to moveValid again to see the total number of available moves from each valid moves position. It also sets movesAvailable which gives an integer of the moves available from each valid position for use in using Wardsdoff's rule.

SetRowColumn takes the index (0-7) and makes that move. For the index it receives it adds/subtracts the appropriate amount from the row and column. This simulates making a move.

MoveValid tests each available move to see if it is on the board and not be made. It returns an array of moves available.

nextMove uses movesAvailable to choose the appropriate next move. If there are no moves available it will return 187 signaling that backtracking is needed. If there is only one move available it sets the index to that move. If there have been less than 32 moves made it uses Warnsdoff's rule to set the index to the move with the least amount of exits. If past the 32 move it simply sets the index to the highest index with a valid move available.

AltMove takes past[] as a parameter. It compares the past[] moves made array with moves available array and returns an index of a move that has not been made yet.

Source Code

```
#Andrew Chittick

#makefile compiles knight


#enable c++11

CXXFLAGS = -std=c++11


#make a knightsTour

knightsTour: main.o user.o tour.o

	g++ main.o user.o tour.o -o knightsTour


#make a main

main: main.cpp user.h tour.h

	g++ -c main.cpp


#make a user

user: user.cpp user.h

	g++ -c user.cpp


#make a tour

tour: tour.cpp tour.h

	g++ -c tour.cpp


#make clean

clean:
	rm -f *.o

	rm knightsTour


#make run

run: knightsTour

	./knightsTour


#end makefile


//main.cpp

//2/10/18

//Andrew Chittick

//knights tour program

//main along with user

//takes a list of starting positions from the user

//allows user to add/delete/modify this list

//sends to tour which completes the knights tour for each

//starting position

//a knights tour is the knight traveling to every

//space on a chess board without visiting the

//same space twice



#include <iostream>

#include <forward_list>
```

```
#include <string>

#include <cstdlib>

#include <fstream>


#include "user.h"

#include "tour.h"


using namespace std;


int main(){//start main


        //initialize function(s)

        int startingPositions(forward_list<int> userList, forward_list<int>::iterator userListIT);


        //initialize variables

        bool keepGoing = true;

        int count;

        string choiceString;

        int choice;

        int modPos;


        //initialize a singly linked list to store

        //users starting position inputs

        forward_list <int> userList;

        forward_list <int>::iterator userListIT;
```

```
userListIT = userList.begin();


//create an instance of User(xput(input/output))
//Class User contains user interaction methods
User xput;


//start
xput.welcome();//show welcome menu


xput.setPosition();//get the users initial starting position
//store that verified position in the singly linked list(userList)
userList.push_front(xput.getPosition());


//open a file for output
fstream myFile;
myFile.open("knightsTourOutput.dat", ios::out);
bool fileOpen;
if (myFile.is_open()){
	fileOpen = true;
}
else{
	cout << ("Unable to open file.") << endl;
	fileOpen = false;
}
```

```
	while (keepGoing == true){//take users input until they select begin tour or exit

		//display the users starting positions(numbered), returns count
		count = startingPositions(userList, userListIT);
		xput.menu();//show main menu
		getline(cin, choiceString);//get choice
		choice = atoi(choiceString.c_str());//convert it to an int

		switch(choice){//menu options

			case 1://Begin Tour
				if (fileOpen == true){//file opened
					//iterate thru userList
					cout << endl << ("Solutions...") << endl;
					myFile << ("Knights Tour Solutions") << endl << endl;
					for (userListIT=userList.begin(); userListIT!= userList.end();
userListIT++){
						cout << ("Starting Position: ") << *userListIT << endl <<
endl;
						myFile << ("Starting Position: ") << *userListIT << endl
<< endl;
						Tour knightTour;
						int* solution = knightTour.takeTour(*userListIT);//solve

						//write solution to file
```

```
                for (int i=0; i<64; i++){
                    myFile.width(4);
                    myFile << solution[i];
                    if ((i+1)%8==0){
                        myFile << endl;
                    }
                }
                myFile << endl << endl;
                delete solution;
            }
        }
        else{//file unable to open
            cout << ("No output file.") << endl;
            //iterate thru userList
            cout << endl << ("Solutions...") << endl;
            for (userListIT=userList.begin(); userListIT!= userList.end();
userListIT++){
                cout << ("Starting Position: ") << *userListIT << endl <<
endl;
                Tour knightTour;
                knightTour.takeTour(*userListIT);//solve
            }
        }
        keepGoing = false;
        break;
```

```
                        case 2://Add starting position
                             xput.setPosition();//set users new position
                             userList.push_front(xput.getPosition());//store it in the list
                             break;
                        case 3://Modify starting position
                             count = startingPositions(userList, userListIT);//display positions
                             cout << ("Select position to modify") << endl;
                             xput.setPosChoice(count);//set position to modify
                             modPos = xput.getPosChoice();//get position
                             xput.setPosition();//modify to...

                             if (modPos == 1){//front element
                                  userList.pop_front();//delete it
                                  userList.push_front(xput.getPosition());//add new front element
                             }
                             else{
                                  userListIT = userList.begin();//reset iterator
                                  for (int i=0; i<(modPos-2); i++){//move iterator before node to be
modified(and adjust for 0)
                                       userListIT++;
                                  }
                                  //add new position
                                  userListIT = userList.insert_after (userListIT,
xput.getPosition());//add it
                                  userListIT = userList.erase_after(userListIT);//delete old position
```

```
                    }
                    cout << ("modified") << endl;
                    break;
                case 4://Delete starting position
                    count = startingPositions(userList, userListIT);//display positions
                    cout << ("Select position to delete") << endl;
                    xput.setPosChoice(count);//set position to delete
                    modPos = xput.getPosChoice();//get position

                    if (modPos == 1){//front element
                        userList.pop_front();//delete it
                    }
                    else{
                        userListIT = userList.begin();//reset iterator
                        for (int i=0; i<(modPos-2); i++){//move iterator before node to be
deleted(and adjust for 0)
                            userListIT++;
                        }
                        userListIT = userList.erase_after(userListIT);//delete the position
                    }
                    cout << ("deleted") << endl;
                    break;
                case 5://exit
                    cout << ("goodbye") << endl;
                    keepGoing = false;
                    break;

                default:

                    cout << ("invalid response") << endl;

                    break;

            }//end switch

        }//end while


        myFile.close();//close file


        return(0);

}//end main


//displays the users starting position(numbered), and returns the total number of starting positions

int startingPositions(forward_list<int> userList, forward_list<int>::iterator userListIT){

        int count = 1;

        cout << ("Your chosen starting positions") << endl;

        for (userListIT = userList.begin(); userListIT != userList.end(); userListIT++){

            cout << count << ". ";

            cout << *userListIT << endl;

            count++;

        }

        count--;//adjust count

        return(count);//returns number of positions

}//end startingPositions()
```

```
//user.h

//header file for user.cpp

//Andrew Chittick

//2/10/18


#ifndef USER_H_EXISTS

#define USER_H_EXISTS


#include <iostream>

#include <cstdlib>

#include "user.h"


using namespace std;


class User{

	public:

		User();

		void welcome();

		void menu();

		int getPosition();

		void setPosition();

		int getPosChoice();

		void setPosChoice(int count);
	private:
		int position;
		string posString;
		int startPos;
		string posChoiceString;
		int posChoice;
};

#endif

//user.cpp
//Andrew Chittick
//2/10/18
//contains menus, and user interactons
//gets all user input
//including starting positions

#include "user.h"

using namespace std;

User::User(){
	User::position = (0);//set default position
}//end default constructor
```

```
void User::welcome(){//Welcomes/explains program to user and shows a chess board with positions


    cout << ("Welcome to Knight's Tour") << endl;

    cout << ("You will input starting positions for a knight on a chess board") << endl;

    cout << ("Below is a chess board representation of the starting positions") << endl;


    //print a chess board with starting position 0-63

    int number = 0;

    for (int r=0; r<8; r++){

        for (int c=0; c<8; c++){

            cout.width(4);

            cout << number;

            number++;

        }

        cout << endl;

    }

}//no longer welcome



void User::menu(){//main menu

    cout << ("Would you like to...") << endl;

    cout << ("1. Begin Knight's Tour") << endl;

    cout << ("2. Add a starting position") << endl;

    cout << ("3. Modify a starting position") << endl;
```

```
	cout << ("4. Delete a starting position") << endl;

	cout << ("5. Exit") << endl;

	cout << ("Your choice (1-5):  ");

}//end menu()



void User::setPosition(){//asks user for a starting position and verifies

	//ask for users input

	cout << ("Please choose a starting position for the knight (0-63):  ") << endl;

	getline(cin, posString);//get input

	startPos = atoi(posString.c_str());//convert input into int


	if (startPos<64){//not too big

		if (startPos > (-1)){//not too small

			position = startPos;//data verified

		}

	}

	else{

		position = 0;//default(bad input/user)

	}

}//end setPosition()


int User::getPosition(){

	return (position);

}//end getPosition()
```

```
void User::setPosChoice(int count){
        bool keepGoing = true;

        while (keepGoing == true){
                cout << ("Your choice (1-") << count << ("):  ");
                getline(cin, posChoiceString);//get input
                posChoice = atoi(posChoiceString.c_str());//convert userPosString into an int

                if (posChoice > count){
                        cout << ("Invalid choice, try again.") << endl;
                }
                else if (posChoice < 1){
                        cout << ("Invalid choice, try again.") << endl;
                }
                else{
                        posChoice = (posChoice);
                        keepGoing = false;
                }
        }//end while
}//end setCount()


int User::getPosChoice(){
        return (posChoice);
}//end getCount()


//tour.h
//header file for tour.cpp
//2/10/18
//Andrew Chittick


#ifndef TOUR_H_EXISTS
#define TOUR_H_EXISTS


#include <iostream>
#include <cstdlib>
#include <list>


#include "tour.h"


using namespace std;


class Tour{
        public:
                Tour();
                int* takeTour(int position);
                void setValidMoves();
                void setRowColumn(int index);
```

```
		int* moveValid(int tRow, int tCol);
		int nextMove();
		int altMove(int past[]);
	private:
		struct boardState{
			int board[8][8];
			int moveNumber;
			int movesAvailable;
			int validMoves[8];
			int cRow;
			int cColumn;
			int movesMade[8];
		};
		boardState current;
		int row;
		int column;
};

#endif


//tour.cpp
//2/10/18
//Andrew Chittick
//contains a doubly linked list of boardstates
//boardstate contains all the data for each
```

```
//move made on the knights tour



#include "tour.h"


using namespace std;


Tour::Tour(){
	//reset the current boardState
	for(int i=0; i<8; i++){//rows
		for(int j=0; j<8; j++){//columns
			current.board[i][j] = (-1);
		}
		current.validMoves[i]=0;
		current.movesMade[i]=0;
	}
	current.moveNumber = 0;
	current.movesAvailable = 0;
	current.cRow=0;
	current.cColumn=0;

	row=0;
	column=0;
}//end constructor
```

```
//repeats until tour is finished then displays solution
//makes each move then stores the boardState before
//making another move
//backtracks if a dead end is reached before completion
int* Tour::takeTour(int position){

        //initialize local variables
        int nextMoveIndex;
        int pastMovesMade[8];
        int sumIndexes = 0;



        //adjust/store starting position
        row = (position / 8);
        column = (position % 8);

        //initialize a doubly linked list (knightTour) of boardState(s)
        list <boardState> knightTour;
        knightTour.push_back(current);

        //set starting position
        current.board[row][column]=current.moveNumber;
        current.moveNumber++;
        current.cRow=row;
        current.cColumn=column;

        setValidMoves();

        //save initial position to knightTour list of boardState(s)

        knightTour.push_back(current);

        //start the tour

        //while move< 64(tour finished)

        while(current.moveNumber<64){


                setValidMoves();//see valid moves

                //get next move

                nextMoveIndex = nextMove();


                if (nextMoveIndex == 187){//fail

                        //backtrack until more paths available

                        do{

                                //initialize pastMovesMade and sumIndexes

                                //gives an array of previously tried moves

                                //and the sum of those moves

                                sumIndexes = 0;

                                for (int i=0; i<8; i++){

                                        pastMovesMade[i]=current.movesMade[i];

                                        sumIndexes = sumIndexes + pastMovesMade[i];

                                }

                                //delete last boardstate from list

                                knightTour.pop_back();
```

```
				//roll back one element
				current = knightTour.back();
				row = current.cRow;
				column = current.cColumn;
			}while(current.movesAvailable <= sumIndexes);

			//update current.movesMade
			for (int i=0; i<8; i++){
				current.movesMade[i]=pastMovesMade[i];
			}
			//make an alternative move
			nextMoveIndex = altMove(pastMovesMade);
		}

		else{//nextMoveIndex valid
			//reset it
			for (int i=0; i<8; i++){
				current.movesMade[i]=0;
			}
		}
		//next move index valid
		//count the move being made
		current.movesMade[nextMoveIndex]=1;
		setRowColumn(nextMoveIndex);
		//set new position
```

```
        current.cRow=row;

        current.cColumn=column;

        //make a move

        current.board[row][column]=current.moveNumber;

        //count it

        current.moveNumber++;

        //update valid moves

        setValidMoves();

        //save it

        knightTour.push_back(current);

}//end while (tour over)


//display/save solution

int counter = 0;

int *solution = new int[64];

current = knightTour.back();

for (int i=0; i<8; i++){

        for (int j=0; j<8; j++){

                cout.width(4);

                cout << current.board[i][j];

                solution[counter] = current.board[i][j];

                counter++;

        }

        cout << endl;

}
```

```
        cout << endl << endl;

        return(solution);

}//end takeTour




void Tour::setValidMoves(){

        //see what moves are valid from the current position

        //saves to current.validMoves array

        //sends to moveValid to add potential moves from that

        //position to current.validMoves array


        int* tempValidMoves = moveValid(row, column);


        //reset current.movesAvailable

        current.movesAvailable=0;

        //set validMoves and movesAvailable

        for (int i=0; i<8; i++){

                current.validMoves[i] = tempValidMoves[i];

                current.movesAvailable=current.movesAvailable+tempValidMoves[i];

        }


        //add the amount of moves available from each

        //valid move to the corresponding position of

        //current.validMoves array
```

```
//finished product:current.validMoves[i]=0-not a valid move
//                      current.validMoves[i]=1-valid move/dead end
//                      current.validMoves[i]=2-valid move/1 move from this position
//                      current.validMoves[i]=8-valid move/all moves but previous valid
//i=0:(i+2),(j+1); 1:(i+2),(j-1)
//  2:(i+1),(j+2); 3:(i+1),(j-2)
//  4:(i-2),(j+1); 5:(i-2),(j-1)
//  6:(i-1),(j+2); 7:(i-1),(j-2)
if (current.validMoves[0] == (1)){//move is valid
        //check valid moves from this position
        int* tempValidMoves = moveValid((row+2),(column+1));
        //add sum of tempValidMoves to validMoves[]
        for (int i=0; i<8; i++){
                current.validMoves[0]=current.validMoves[0]+tempValidMoves[i];
        }
}
if (current.validMoves[1] == (1)){//move is valid
        //check valid moves from this position
        int* tempValidMoves = moveValid((row+2),(column-1));
        //add sum of tempValidMoves to validMoves[]
        for (int i=0; i<8; i++){
                current.validMoves[1]=current.validMoves[1]+tempValidMoves[i];
        }
}
if (current.validMoves[2] == (1)){//move is valid
```

```
        //check valid moves from this position
        int* tempValidMoves = moveValid((row+1),(column+2));
        //add sum of tempValidMoves to validMoves[]
        for (int i=0; i<8; i++){
                current.validMoves[2]=current.validMoves[2]+tempValidMoves[i];
        }
}
if (current.validMoves[3] == (1)){//move is valid
        //check valid moves from this position
        int* tempValidMoves = moveValid((row+1),(column-2));
        //add sum of tempValidMoves to validMoves[]
        for (int i=0; i<8; i++){
                current.validMoves[3]=current.validMoves[3]+tempValidMoves[i];
        }
}
if (current.validMoves[4] == (1)){//move is valid
        //check valid moves from this position
        int* tempValidMoves = moveValid((row-2),(column+1));
        //add sum of tempValidMoves to validMoves[]
        for (int i=0; i<8; i++){
                current.validMoves[4]=current.validMoves[4]+tempValidMoves[i];
        }
}
if (current.validMoves[5] == (1)){//move is valid
        //check valid moves from this position
```

```
                int* tempValidMoves = moveValid((row-2),(column-1));
                //add sum of tempValidMoves to validMoves[]
                for (int i=0; i<8; i++){
                        current.validMoves[5]=current.validMoves[5]+tempValidMoves[i];
                }
        }
        if (current.validMoves[6] == (1)){//move is valid
                //check valid moves from this position
                int* tempValidMoves = moveValid((row-1),(column+2));
                //add sum of tempValidMoves to validMoves[]
                for (int i=0; i<8; i++){
                        current.validMoves[6]=current.validMoves[6]+tempValidMoves[i];
                }
        }
        if (current.validMoves[7] == (1)){//move is valid
                //check valid moves from this position
                int* tempValidMoves = moveValid((row-1),(column-2));
                //add sum of tempValidMoves to validMoves[]
                for (int i=0; i<8; i++){
                        current.validMoves[7]=current.validMoves[7]+tempValidMoves[i];
                }
        }//current.validMoves[] all set
}//end setValidMoves()
```

```
void Tour::setRowColumn(int index){
        //set new row and column using given index
        switch(index){
                case 0://(i+2),(j+1)
                        Tour::row=Tour::row+2;
                        Tour::column=Tour::column+1;
                        break;
                case 1://(i+2),(j-1)
                        Tour::row=Tour::row+2;
                        Tour::column=Tour::column-1;
                        break;
                case 2://(i+1),(j+2)
                        Tour::row=Tour::row+1;
                        Tour::column=Tour::column+2;
                        break;
                case 3://(i+1),(j-2)
                        Tour::row=Tour::row+1;
                        Tour::column=Tour::column-2;
                        break;
                case 4://(i-2),(j+1)
                        Tour::row=Tour::row-2;
                        Tour::column=Tour::column+1;
                        break;
                case 5://(i-2),(j-1)
                        Tour::row=Tour::row-2;
```

```
                    Tour::column=Tour::column-1;

                    break;

                case 6://(i-1),(j+2)

                    Tour::row=Tour::row-1;

                    Tour::column=Tour::column+2;

                    break;

                case 7://(i-1),(j-2)

                    Tour::row=Tour::row-1;

                    Tour::column=Tour::column-2;

                    break;

                default://problems

                    cout << ("BIG TROUBLE") << endl;

                    break;

        }//end switch

}//end setRowColumn()




//Tour::makeMove()

//returns an array(moves) of 8 positions

//0=move not available

//1=move available

//moves[0]=(i+2),(j+1)

//moves[1]=(i+2),(j-1)

//moves[2]=(i+1),(j+2)

//moves[3]=(i+1),(j-2)
```

```
//moves[4]=(i-2),(j+1)

//moves[5]=(i-2),(j-1)

//moves[6]=(i-1),(j+2)

//moves[7]=(i-1),(j-2)

int* Tour::moveValid(int tRow, int tCol){

	//initialize an array of moves available

	int* moves = new int[8];//making moves

	for (int i=0; i<8; i++){

		moves[i]=0;

	}


	if (tRow<6){//row+2 valid

		if (tCol<7){//column+1 valid

			if (current.board[tRow+2][tCol+1] == (-1)){

				moves[0]=1;//(i+2),(j+1)

			}

		}

		if (tCol>0){//column-1 valid

			if (current.board[tRow+2][tCol-1] == (-1)){

				moves[1]=1;//(i+2),(j-1)

			}

		}

	}

	if (tRow<7){//row+1 valid

		if (tCol<6){//column+2 valid
```

```
                if (current.board[tRow+1][tCol+2] == (-1)){
                        moves[2]=1;//(i+1),(j+2)
                }
        }
        if (tCol>1){//column-2 valid
                if (current.board[tRow+1][tCol-2] == (-1)){
                        moves[3]=1;//(i+1),(j-2)
                }
        }
}
if (tRow>1){//row-2 valid
        if (tCol<7){//column+1 valid
                if (current.board[tRow-2][tCol+1] == (-1)){
                        moves[4]=1;//(i-2),(j+1)
                }
        }
        if (tCol>0){//column-1 valid
                if (current.board[tRow-2][tCol-1] == (-1)){
                        moves[5]=1;//(i-2),(j-1)
                }
        }
}
if (tRow>0){//row-1 valid
        if (tCol<6){//column+2 valid
                if (current.board[tRow-1][tCol+2] == (-1)){
                        moves[6]=1;//(i-1),(j+2)
                    }
                }
                if (tCol>1){//column-2 valid
                    if (current.board[tRow-1][tCol-2] == (-1)){
                        moves[7]=1;//(i-1),(j-2)
                    }
                }
        }
        return (moves);
}//end moveValid()


int Tour::nextMove(){//returns index of next move to be made
        //initialize local variables
        int nextMoveIndex;
        int lowMoves=9;

        if (current.movesAvailable == 0){//no moves available
                //backtrack
                nextMoveIndex=187;
        }
        else if (current.movesAvailable == 1){//only 1 possible move
                for (int i=0; i<8; i++){//find available move
                        if (current.validMoves[i]>0){
```

```
                        nextMoveIndex=i;//make move
                }
        }
}
else{//multiple moves available
        if (current.moveNumber<32){//w/warnsdoff
                //compare validMoves available
        //save fewest exits available to lowMoves
                for (int i=0; i<8; i++){
                        if (current.validMoves[i]>0){//move valid
                                if (current.validMoves[i]<lowMoves){//fewer exits
                                        lowMoves=current.validMoves[i];
                                        nextMoveIndex=i;
                                }
                        }
                }
        }
        else{//without warnsdoff
                for (int i=0; i<8; i++){
                        if (current.validMoves[i]>0){//move valid
                                nextMoveIndex=i;
                        }//highest valid index saved
                }
        }
}
```

```
	return(nextMoveIndex);
}//end nextMove()



//makes a move opposite of normal(nextMove)
int Tour::altMove(int past[]){//returns index of next alternative move to be made
	//initialize local variable
	int nextMoveIndex = 987;

	for (int i=0; i<8; i++){
		if (past[i]<1){//move hasnt been made yet
			if (current.validMoves[i]>0){//move valid
				nextMoveIndex = i;
			}
		}
	}
	return(nextMoveIndex);
}//end altMove()
```

<u>Transcript</u>

```
chittick@cof-ubuntu:~/Dropbox/school/cs362/knight$ make run
g++ -std=c++11    -c -o main.o main.cpp
g++ -std=c++11    -c -o user.o user.cpp
g++ -std=c++11    -c -o tour.o tour.cpp
```

```
g++ main.o user.o tour.o -o knightsTour

./knightsTour

Welcome to Knight's Tour

You will input starting positions for a knight on a chess board

Below is a chess board representation of the starting positions

   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

Please choose a starting position for the knight (0-63):

24

Your chosen starting positions

1. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):
```

```
63

Your chosen starting positions

1. 63

2. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):

48

Your chosen starting positions

1. 48

2. 63

3. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):
```

```
9

Your chosen starting positions

1. 9

2. 48

3. 63

4. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):

1

Your chosen starting positions

1. 1

2. 9

3. 48

4. 63

5. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position
```

```
4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):

13

Your chosen starting positions

1. 13

2. 1

3. 9

4. 48

5. 63

6. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  3

Your chosen starting positions

1. 13

2. 1

3. 9

4. 48

5. 63
```

```
6. 24

Select position to modify

Your choice (1-6):  3

Please choose a starting position for the knight (0-63):

19

modified

Your chosen starting positions

1. 13

2. 1

3. 19

4. 48

5. 63

6. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):

35

Your chosen starting positions

1. 35

2. 13
```

```
3. 1

4. 19

5. 48

6. 63

7. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  4

Your chosen starting positions

1. 35

2. 13

3. 1

4. 19

5. 48

6. 63

7. 24

Select position to delete

Your choice (1-7):  5

deleted

Your chosen starting positions

1. 35
```

```
2. 13

3. 1

4. 19

5. 63

6. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position

3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  2

Please choose a starting position for the knight (0-63):

40

Your chosen starting positions

1. 40

2. 35

3. 13

4. 1

5. 19

6. 63

7. 24

Would you like to...

1. Begin Knight's Tour

2. Add a starting position
```

```
3. Modify a starting position

4. Delete a starting position

5. Exit

Your choice (1-5):  1


Solutions...

Starting Position: 40


  13  36   9  56  41  38   7  46

  10  57  12  37   8  45  42  39

  35  14  55  58  43  40  47   6

  54  11  34  51  48  59  44  63

  15  50  53  60  33  62   5  26

   0  21  18  49  52  27  32  29

  19  16  23   2  61  30  25   4

  22   1  20  17  24   3  28  31



Starting Position: 35


  37  34  13  44  41  32  11  28

  14  43  36  33  12  29  24  31

  35  38  45  42  63  40  27  10

  46  15  62  39  48  23  30  25

  59  50  47   0  61  26   9  22
  16  53  60  49  56  19   6   3

  51  58  55  18   1   4  21   8

  54  17  52  57  20   7   2   5



Starting Position: 13


  30  13  28  35  32  15  40   1

  27  36  31  14  41   0  33  16

  12  29  42  37  34  47   2  39

  43  26  59  48  45  38  17  54

  60  11  44  63  58  53  46   3

  25   8  61  52  49  20  55  18

  10  51   6  23  62  57   4  21

   7  24   9  50   5  22  19  56



Starting Position: 1


  37   0  17  22  39  24  15  12

  18  21  38  35  16  13  40  25

   1  36  19  42  23  48  11  14

  20  61  44  47  34  41  26  49

  45   2  33  60  43  50  55  10

  62   5  46  51  54  29  58  27
   3  52   7  32  59  56   9  30

   6  63   4  53   8  31  28  57
```

Starting Position: 19

```
 11  14  41  36   1  16  19  44

 40  37  12  15  42  45   2  17

 13  10  39   0  35  18  43  20

 38  51  34  61  46  53  56   3

  9  62  47  52  33  60  21  54

 50  29  32  63  24  55   4  57

 31   8  27  48  59   6  25  22

 28  49  30   7  26  23  58   5
```

Starting Position: 63

```
 36  39   8  41  50  53   6  47

  9  42  35  38   7  48  51  54

 34  37  40  49  52  57  46   5

 43  10  33  60  45  26  55  58

 32  61  44  27  56  59   4  19

 11  14  31  62  25  20   1  22

 30  63  16  13  28  23  18   3
```

```
  15  12  29  24  17   2  21   0
```

Starting Position: 24

```
  31  16  19   2  33  36  21   4
  18   1  32  37  20   3  34  39
  15  30  17  54  35  38   5  22
   0  47  42  45  52  55  40  63
  29  14  53  56  41  44  23   6
  48  11  46  43  26  51  62  59
  13  28   9  50  57  60   7  24
  10  49  12  27   8  25  58  61
```

Output File

Knights Tour Solutions

Starting Position: 40

```
  13  36   9  56  41  38   7  46
  10  57  12  37   8  45  42  39
  35  14  55  58  43  40  47   6
  54  11  34  51  48  59  44  63
  15  50  53  60  33  62   5  26
   0  21  18  49  52  27  32  29
```

```
 19  16  23   2  61  30  25   4
 22   1  20  17  24   3  28  31
```

Starting Position: 35

```
 37  34  13  44  41  32  11  28
 14  43  36  33  12  29  24  31
 35  38  45  42  63  40  27  10
 46  15  62  39  48  23  30  25
 59  50  47   0  61  26   9  22
 16  53  60  49  56  19   6   3
 51  58  55  18   1   4  21   8
 54  17  52  57  20   7   2   5
```

Starting Position: 13

```
 30  13  28  35  32  15  40   1
 27  36  31  14  41   0  33  16
 12  29  42  37  34  47   2  39
 43  26  59  48  45  38  17  54
 60  11  44  63  58  53  46   3
 25   8  61  52  49  20  55  18
 10  51   6  23  62  57   4  21
```

```
  7 24  9 50  5 22 19 56


Starting Position: 1


 37  0 17 22 39 24 15 12

 18 21 38 35 16 13 40 25

  1 36 19 42 23 48 11 14

 20 61 44 47 34 41 26 49

 45  2 33 60 43 50 55 10

 62  5 46 51 54 29 58 27

  3 52  7 32 59 56  9 30

  6 63  4 53  8 31 28 57


Starting Position: 19


 11 14 41 36  1 16 19 44

 40 37 12 15 42 45  2 17

 13 10 39  0 35 18 43 20

 38 51 34 61 46 53 56  3

  9 62 47 52 33 60 21 54

 50 29 32 63 24 55  4 57

 31  8 27 48 59  6 25 22

 28 49 30  7 26 23 58  5
```

```
Starting Position: 63


 36  39   8  41  50  53   6  47

  9  42  35  38   7  48  51  54

 34  37  40  49  52  57  46   5

 43  10  33  60  45  26  55  58

 32  61  44  27  56  59   4  19

 11  14  31  62  25  20   1  22

 30  63  16  13  28  23  18   3

 15  12  29  24  17   2  21   0



Starting Position: 24


 31  16  19   2  33  36  21   4

 18   1  32  37  20   3  34  39

 15  30  17  54  35  38   5  22

  0  47  42  45  52  55  40  63

 29  14  53  56  41  44  23   6

 48  11  46  43  26  51  62  59

 13  28   9  50  57  60   7  24

 10  49  12  27   8  25  58  61
```