CS302

Assignment 2: Knight Problem

Description

The mighty knight. The piece in chess that is the most confusing. I was always told that it moves in an 'L' shape, that helped me remember whenever I played chess as a kid. Thus, there are a maximum of 8 possible moves a knight can make from a location on the chessboard, it moves up two spots and then one to the left or up two spots and one to the right or up one spot and then to the left two spots, or up one spot and then two to the right, the last 4 are the same idea but the knight moves down one/two and then to the left/right one/two.

However, the knight does not always get used as much as it should (well I wouldn't really know I'm not a professional chess player, but I'll make this claim anyway), so we want to have a chessboard and let the knight character complete a tour around the entire board. The knight will have some starting position, and your job is to construct a tour (write the knight move counter) onto each element of the board to construct a knight tour on a given $n \times n$ board and given the knight's initial coordinates. You will need to use a recursive back tracking approach (using a heuristic approach defined later in this write up). Here is the class you will need to implement

```
class knightType
{
public:
  struct position
  {
    position(int r = 0, int c = 0, int o = 0)
    {
      row = r;
      col = c;
      onwardMoves = o;
    }

    int row;
    int col;
    int onwardMoves;
  };

  knightType(int = 8);
  bool knightTour(int, int);
  void outputTour() const;
private:
  bool knightTour(int, int, int);
  std::vector<position> getAvailableMoves(int, int); / checks out of bound & visited positions
  bool fullBoard();
  std::vector< std::vector<int> > board;
  int functionsCalled;
};
```

Each element contains/performs the following

- **std::vector< std::vector<int> > board** - chessboard that stores the tour index after each knight move
- int **functionsCalled** - counter that stores the amount of times the recursive function was called
- struct **position** - struct that stores the coordinates of a knight and the amount of onward moves from that coordinate
- knightType::knightType**(int dim)** - constructor that allocates a *dim* × *dim* vector to the board object and sets each element to 0, sets functionsCalled to 0
- **std::vector<**knightType**::position>** knightType**::getAvailableMoves(int r,** int **c)** - function that returns a list of possible moves a knight can make from coordinate [r, c] along with the onward moves that the knight can make if the new potential location is chosen, for example you would create the following

  ```
  std::vector<position> knightMoves;

  //if r equals 1 and c equals 1, a possible element in knightMoves could be

  knightMoves.push_back( position(0, 0, 0) );
  knightMoves[0].row = 3;
  knightMoves[0].col = 2;

  //You would need to set knightMoves[0].onwardMoves
  //to a value that denotes the amount of moves the
  //knight would be able to make from location
  //[3, 2]
  ```

  Then you would repeat this for every possible legal move from [r, c], and then at the end return knightMoves

- bool knightType**::fullBoard()** - returns true if the knight tour is complete (every cell in board contains an integer larger than 0), and false otherwise
- void knightType**::outputTour()** const - outputs the board and the amount of functions called, see the sample output
- bool knightType**::knightTour(int r,** int **c)** - function that makes the first initial call to the recursive function, the function body is just return **knightTour(r, c, 1)**;, this is just a wrapper function that is called in main
- bool knightType**::knightTour(int r,** int **c,** int **tourIndex)** - this recursive function simulates the knight movement, the steps are

  *increment functions call by 1

  1. Set the board[r][c] with tourIndex
  2. Check if the board is full if so, return true
  3. Get all the available moves from location [r, c]
  4. Pick the available move with the minimal onward moves
  5. Make a recursive call to this function, pass in the new r and c values using the chosen available move from the above step

  *set up base case outward moves ≤0
  ×

6. If the function returns true then simply return true, otherwise go back to step 4 but pick the next smallest onward moves from available moves

Contents of main

Main is given to you

Specifications

- Comment your code and functions
- No counter controlled loops allowed
- Do not modify the class
- Make sure your knight tour function is recursive

Sample Run

```
Enter board dimension: 8
Enter initial knight position: 4 5

Knight FTW

   A  B  C  D  E  F  G  H
A  24 33 10 5  26 31 12 3
B  9  6  25 32 11 4  27 30
C  34 23 8  49 36 29 2  13
D  7  48 35 58 1  50 37 28
E  22 59 20 47 56 63 14 41
F  19 46 57 64 51 40 55 38
G  60 21 44 17 62 53 42 15
H  45 18 61 52 43 16 39 54

Functions called: 64

Another go? y

Enter board dimension: 8
Enter initial knight position: 8 8

Knight FTW

   A  B  C  D  E  F  G  H
A  63 10 29 26 41 12 31 16
B  28 25 64 11 30 15 42 13
C  9  62 27 54 43 40 17 32
D  24 51 44 61 36 53 14 39
E  45 8  59 52 55 38 33 18
F  50 23 48 37 60 35 56 3
G  7  46 21 58 5  2  19 34
H  22 49 6  47 20 57 4  1

Functions called: 64
```

← use loop for char [A] 'A'

Don't have to use <IOmanip>

```
Another go? n
```

Submission

Submit Assignment01.cpp and any header files to the codegrade submission by the deadline

References

- Supplemental Video https://youtu.be/bCcXB0imQ-4
- Link to the top image can be found at *https : //clipartpng.com/*?2756, *knight − white − chess − piece − png − clip − art*