CS110: Computer Programming Lab
Department of CSE
IIT, Guwahati

Module 04 Stage 01 Exercise 07

Exercise

Some of supportive initial code (without comments) is shown in the appendix to this exercise. You may use it if you wish; the program performs more functions than the specifications of this assessment exercise require.

This exercise is about a recursive program and is called Knight Tour. A chessboard is a matrix (two dimensional array) of 8 rows and 8 columns. There is a piece called Knight (Ghora in Hindi). The piece can move from its position on the board to one of the 8 other positions (if they are on the board) in a single step. These positions are reached by moving two positions along one axis and one positions along the other. Thus, from square at `(x, y)`, a knight can be any of these positions in the next step: `(x+2, y+1), (x+2, y-1), (x+1, y-2), (x-1, y-2), (x-2, y-1), (x-2, y+1), (x-1, y+2), (x+1, y+2).`

A knight tour is a sequence of 64 moves (steps) that takes the knight to each square on the board. No square is repeated in the tour.

The program segment in the appendix is part of the program I wrote. It has two recursive functions. Function `tour()` takes the arguments specifying the present position of the knight and the number of steps it used to reach this position. This is a recursive function that will be called recursively with the arguments/parameters updated for the next position of the knight. If the tour cannot be completed than the choice of the next position (step) was wrong and we need to attempt constructing the tour path through a different next step.

As the knight tour progresses, I have recorded in a struct three pieces of information: x and y coordinates of the previous square on the board and the step number. You may decide to simplify and use only the step number in the square to mark the fact that the knight has visited the square.

A solution is copied here for you to note; there may be other solutions too!

```
Found

[8, 8,  1]  [0, 2, 60]  [0, 1, 39]  [4, 2, 34]  [6, 1, 31]  [7, 1, 18]  [7, 2,  9]  [5, 1, 64]
[2, 2, 38]  [3, 0, 35]  [4, 0, 32]  [1, 0, 61]  [6, 0, 10]  [4, 3, 63]  [7, 3, 30]  [6, 3, 17]
[1, 4, 59]  [0, 0,  2]  [0, 3, 37]  [2, 0, 40]  [2, 1, 33]  [3, 3, 28]  [5, 0, 19]  [6, 4,  8]
[1, 1, 36]  [0, 5, 49]  [4, 4, 42]  [4, 5, 27]  [3, 1, 62]  [4, 1, 11]  [7, 5, 16]  [5, 2, 29]
[2, 3, 43]  [0, 6, 58]  [1, 2,  3]  [1, 3, 50]  [3, 2, 41]  [4, 6, 24]  [7, 6,  7]  [6, 2, 20]
[1, 7, 48]  [3, 4, 51]  [3, 7, 46]  [4, 7, 55]  [6, 6, 26]  [7, 4, 21]  [5, 3, 12]  [5, 6, 15]
[2, 7, 57]  [0, 4, 44]  [0, 7, 53]  [2, 4,  4]  [6, 7, 23]  [7, 7, 14]  [5, 4, 25]  [5, 7,  6]
[1, 5, 52]  [2, 5, 47]  [3, 5, 56]  [1, 6, 45]  [2, 6, 54]  [3, 6,  5]  [5, 5, 22]  [6, 5, 13]
```

As you would see in the solution, the step 1 is at (0, 0) and the last square for the tour is (7, 0).

The tricky function to code is `int getNext(struct path b[8][8], int cnt, int x, int y, struct path *nxt);`

Array `b` is the board, `(x, y)` is the current position of the knight and `struct *nxt` will receive the next position of the knight selected by the call to the function. Parameter `cnt` and return value of the function are related to the possible alternate positions for the next position (step) of the knight. Recall that a knight can have up to 8 different next locations. Parameter `cnt` tells the variant that has already been generated. The new position that will be inserted in `struct *nxt` will be a position after this variant. The identifying number for the new variant that is generated is returned as the return value for the function call. It must be remembered and passed as parameter `cnt` in the next call. Thus when function `getNext()` is called with `cnt` value 5, then it will return the 6th alternate as the possible next location for the knight tour. However, if that position was already visited, it can be 7th or 8th alternative too. Indeed, if no more feasible position is possible for the next step, it will return 0.

Next position of a knight can be infeasible because of any of these reasons:

1. The position is outside the board. Or,
2. It is an already visited position.

As you note the problems in this drill are challenging. (This is how the lab modules are organized. Only the best students should reach these levels and be rewarded.) Use `assert()` declarations liberally to catch the inadvertent errors and mistakes that you may make while writing your code.

My function `tour()` is about 40 lines of C code including many `assert()` declarations. Function `getNext()` is a big case statement of some 60 lines with `asset()` declarations. I wrote it as a recursive program; however, you may write it as a non-recursive code.

Appendix

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

struct path
{
    int r;
    int c;
    int when;
};

void printBoard(struct path b[8][8])
```

```
{
    int x, y;
    for (x= 0; x<8; x++)
    {
        printf("\n");
        for (y= 0; y<8; y++)
            printf("[%d, %d, %2d]  ", b[x][y].c,
                     b[x][y].r, b[x][y].when);
    }
    printf("\n");
}

int getNext(struct path b[8][8], int cnt,
            int x, int y, struct path *nxt);

int tour(struct path b[8][8], int x, int y, int step);


int main()
{
    struct path b[8][8];
    int x, y;

    for (x= 0; x<8; x++)
        for (y= 0; y<8; y++)
        {
            b[x][y].r = 9;
            b[x][y].c = 9;
            b[x][y].when = -1;
        }

    b[0][0].r = 8;
    b[0][0].c = 8;
    b[0][0].when = 1;

    if (tour(b, 0, 0, 1))
    {
        printf ("Found\n");
        printBoard(b);
    }
    else printf("NO TOUR\n");
}
```

Error Reporting and Suggestions for Improvements

My sincere apologies if the document has errors or mistakes. Please report errors in this document to vmm@iitg.ernet.in. Also, I welcome suggestions and advice to improve the quality of the document for the students of CS110.