Lab – 9

Week of Mar. 18, 2018

1 Instructions

1. Use the OCaml top-level to develop and debug your code.
2. You may assume that all inputs are valid unless otherwise stated in the problem.
3. In questions that require string outputs, be careful not to include leading or trailing whitespace.
4. You may submit and evaluate your code a *maximum* of 15 times without penalty. Subsequently, you will lose 2 marks per additional evaluation. Therefore, please ensure that you have thoroughly debugged your code before submitting.

The following submission file is *required*:

1. `langston.ml`

2 On your own

Learn how to parse *command line arguments.* For reference, you can use this short tutorial: `https://ocaml.org/learn/tutorials/command-line_arguments.html`

For this assignment, we suggest that you use the array data structure, but there is no restriction on what kind of data structure you use.

3 Langston's ants

Langston's ants essentially operates on a 2-dimensional grid of size K. Each cell in this grid can either be **black** or **white**. In the beginning, we place an "ant" on a cell, and give one of the 4 possible directions (left, right, up, down) to it. This ant moves according to the following two simple rules:

1. At a white cell, ant makes a 90 right-turn, flips the color of the cell to black, and moves forward by one cell.
2. At a black cell, ant makes a 90 left-turn, flips the color of the cell to white, and moves forward by one cell.

If the ant makes a move that forces it to "fall off" the grid, we simply make it "wrap around" to come back on the opposing side of the grid. For example, if the ant is on the cell $(2, 2)$ of a 3×3 grid, and it has to move "right", it appears back on the $(2, 0)$ cell. (Note that row and column number are 0-indexed)

Write a program that takes the following parameters as its command-line arguments in the specified order (not accept them as inputs in your program):

1. **rounds**: an integer specifying the number of steps that your ant must make
2. **infile**: an input file from which you read the initial configuration of the grid
3. **outfile**: an output file into which you will write the final configuration of the grid

Based on these parameters, your program must simulate the system, a $K \times K$ grid on which the ant is placed initially at position (i, j) with the specified direction. After making the specified number of steps, the program must output the status of the grid by printing the square grid on to the output file – use O for a white cell and X for a black cell and in the last line print one of {A, >, <, V} for the ant (corresponding to orientations of "UP", "RIGHT", "LEFT", "DOWN" respectively along with the cell co-ordinates (Please refer to the sample input/output below for the exact format). After printing this to the file, the program terminates.

3.1 How your program will be compiled and run

Your program will be run from the *command line* after compiling it into an executable. That is, your program will be compiled using the following command: `$ ocamlc -o langston langston.ml`. The executable `langston` is then run using (for example) the command: `$ ./langston 5 input.txt out.txt`. Note that, your program should consist of a *main* function that will read the contents of the `inFile` and then perform the operations as specified.

3.2 Format of the input and output files

The input file will consist of the grid, one row per line and the location + orientation of ant in the last line. The letter 'O' is used to denote white and the letter 'X' for black. There are no spaces between the letters. You may assume that the grid is a valid grid (that is, it is a k x k grid). Last line describes the position of ant (i, j) followed by a space and then the orientation of the ant which will be denoted by one of: 'A' for ant facing UP, 'V' for ant facing down, '>' for ant facing right, or '<' for ant facing left. The output file will follow exactly the same format as above.
Below provided sample input/output is for a 3X3 grid and rounds = 3 and (2,0) means bottom left (same notation as for accessing matrix elements)

Input File

```
XOX
OXO
XOO
(2,0) >
```

Output File

```
XOX
XOO
OOO
(0,1) A
```