# CS ID2230 DATA STRUCTURES & APPLICATIONS

ASSIGNMENT 1

**Instructions**

- Please read the below description of the assignment carefully.
- Deadline of submission : September 3, 2023 EOD
- Grading Policy:
  - 5 marks if your program runs correctly on the inputs the TAs give
  - 3 marks if the design/logic of the program is correct. That is, it meets all design and implementation expectations but may have 1 - 2 minor problems because of which it may not run correctly
  - 2 marks if the code styling is good. One example of such standards is `https://cs.brynmawr.edu/Courses/cs206/fall2012/codingStandards.html`.
- You will have to code the algorithms in C programming language.

This assignment is courtesy `https://courses.cs.washington.edu/courses/cse373/13wi/homework/3/spec.pdf`.

## 1. Part A : Implementation of Deque

You have to write your own implementation of a deque using an array where you will use the ideas of a circular buffer that was discussed in class. This is to ensure that you utilize the array space well. The operations of adding and removing elements at both ends must run in $\mathcal{O}(1)$ average runtime. You need to implement the associated manipulator operations : $add\_first(x)$, $remove\_first()$, $add\_last(x)$ and $remove\_last()$ and also the required access and constructor functions.

## 2. Part B : 2D-Maze Solver Description

The input will be a .txt file which will give the description of a 2D maze as shown in the left part of Fig.2.1. The description captures a maze as shown in the right part of the figure. The hashes represent walls/blocks in the maze, S denotes the start position and E denotes the final/end position. You will have to use your deque implementation (Part A) to design an algorithm that searches for a path from the start position to the end position. The pseudocode is as given in Fig 2.2.

**input file maze1.txt:**

```
##########
#  S     #
# ### ## #
# #    # #
# #  # # #
# ## #####
# #       #
# # #  # #
##### ####
#   #    #
# #    #E#
##########
```

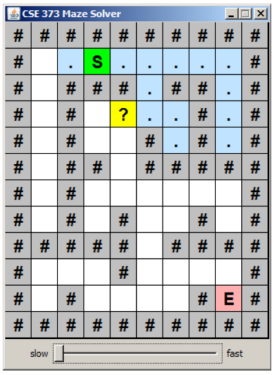


**Figure 2.1.** Maze description

```
solve(maze):
    Keep a deque of squares to visit, initially empty.
    Put the maze's start location in the deque.
    Repeat the following until the deque is empty, or until we solve the maze:
        L := Remove the first location from the deque.
        If L is the maze's end location, then we have solved the maze.
        If we have already visited L before, or L is a wall, ignore it.
        Otherwise:
            Mark L as being visited.
            Add the neighbors of L (up 1, down 1, left 1, right 1) to your deque:
                If the neighbor is closer to the end location than L,
                add it to the front of the deque.  Otherwise add it to the back.
    If you end up with an empty deque and have not found the end, there is no solution.
```

**Figure 2.2.** Pseudocode

The algorithm examines possible paths, looking for a path from the start to the end. It is more efficient if it examines paths that move closer to the end first. So it takes advantage of the nature of a deque by adding closer neighbors to the front of the deque, meaning that they will be processed sooner, and further neighbors to the end of the deque, meaning that they will be processed later. Maze locations have to represented as points on the $xy$ plane and should have a distance method you can use to see how far apart they are from each other. We can use the standard screen coordinate system – "up" means negative $y$ and "down" means positive $y$; "left" means negative $x$ and "right" means positive $x$.

The output should be like Fig 2.3. This is for the maze given above (maze1.txt). There are two more mazes (maze2.txt and maze3.txt) for which you need to able to run the program.

```
Maze file name? maze1.txt
Searching for a path ...

Here is your solved maze:
##########
# .S.....#
# ###.##.#
# # ...#.#
# # .#.#.#
# ##.#####
# # .....#
# # #..#.#
#####.####
#   #....#
# #    #E#
##########
```

**Figure 2.3.** Solution

**Extra Credit.** You can optionally turn in a text file named mymaze.txt containing your own data file representing a 2-D maze of your own creation, in the same format as the provided data files. To receive credit, you must submit a valid file (rectangular in shape, consisting only of proper characters such as # S E ' ', etc.). Your file should not be just a copy or slightly-changed copy of one of the provided input files; it should be your own work.