CPSC 331 (Spring 2023)
Assignment 2
Stacks, Queues, and Search Strategies
(8% of final mark)
Due: Monday April 22 @ 11:59PM

Lead TA: The lead TA for this assignment is Joweria Ekram (joweria.ekram1@ucalgary.ca)

Objectives

1. Build and utilize stack and queue ADTs in developing a search mechanism.
2. Understand and use array-based and linked implementations.

Problem Description

In this assignment, you will write an application that searches a maze to help a trapped mouse find the hidden cheese. The maze can be represented using 4 symbols as follows:

- 1 represents a wall
- 0 represents space
- **m** represents the mouse's initial position
- **c** represents the cheese position

The following is an example input maze:

111111111111
101000001001
1**c**0000100001
101111111111
100111100001
110111111001
110000000**m**11
111110111101
111111111111

Your application must search the maze, starting at the **m** position, and stops when it reaches the **c** position or if **c** is not reachable from **m**. Not all mazes have solutions. So, the application must also stop if the maze has no solution.

For simplicity, you may assume the following:

- the maze is surrounded by 1's
- the cheese (**c** position) can be found anywhere in the maze, except at the border (surrounding 1's)

Create your own larger mazes to test your code. Your mazes must have a size of at least 50x50 characters, but they should not exceed a size that cannot fit on one screen without breaking lines.

Programming

Write a Java application that inputs the maze from a file (do **not** hardcode the maze in your program), and searches for the **c** position, starting from the **m** position.

Your application must print the updated maze, every time a new position is tried, until a solution is found, or the application discovers that a solution is impossible (**c** is not reachable from **m**).

The general structure of your search algorithm is as follows, where structure can be a *stack* or a *queue*.

```
s ← starting cell (m's position)
structure.add(s) // push() or enqueue()
more-to-search ← true
while (structure is not empty and more-to-search) do {
        j ← structure.delete() // pop() or dequeue()
        Mark j as visited
        trailQueue.enqueue(j)
        if (j is the cheese (c) position) then
                more-to-search ← false
        else
                For (every non-wall cell k that is a neighbour of j) do
                        If (k is not marked visited) then
                                structure.add(k) // push() or enqueue()
  }
```

A neighbour cell of cell *k* is one that is reachable from *k* by one hop (up, down, left or right.)

Once the search is completed (successful or otherwise), trailQueue contains the trail of steps. Use trailQueue to show the steps of the search for the user, printing the maze after each step. Use the characters <, >, ^, and v to indicate the indication of the mouse movement in the maze. Store these steps in an output file.

Elements in this structure are of type Cell. Define an appropriate class Cell to represent cell objects. At a minimum, you need the row index, column index, cell type (space, wall, mouse, or cheese), and whether the cell is visited or not.

Part I – Depth-First Search
For this part, your structure is a **stack**. You must write your own linked implementation of a stack. You are required to use double-linking. Use exception handling (i.e., do **not** use the "by contract" implementation). Note that the delete() method in the above pseudo-code refers to a stack pop() and add() refers to push().

Formulate a loop invariant for the while loop.

Part II – Breadth-First Search
For this part, your structure is a **queue**. You must write your own circular queue implementation, using double-linking. Also use exception handling. Here the delete() method refers to dequeue() and add() refers to enqueue().

Formulate a loop invariant for the while loop.

Deliverables
Submit to the appropriate dropbox:

- Java source code
- A report (maximum: one typed page) that includes a discussion answering the following questions:
    - What is the central difference between depth- and breadth-first searches as you have noticed from your output?
    - Under what circumstances would one be better than the other?
    - Compare the complexity of both search strategies.
- Loop invariants for each part in a maximum of one typed page

Submission: Submit the deliverables to the appropriate dropbox. The TA may provide extra submission requirements or instructions.

Collaboration: You are advised to work on this assignment in pairs. Groups of more than two members are not allowed. You may change your partner from the previous assignment. Any discussion with your colleagues outside your team regarding the assignment must be kept at a very high level.

Late Submission Policy: Late submissions will be penalized as follows:
-12.5% for each late day or portion of a day.

Academic Misconduct: Any similarities between submissions will be further investigated for academic misconduct. Your final submission must be your own team's original work. While you are encouraged to discuss the assignment with colleagues outside your team, this must be limited to conceptual and design decisions. Code sharing by any means with colleagues that are not in your team is prohibited. This includes and is not limited to looking at someone else's paper or screen. Any re-used code of excess of 5 lines must be cited and have its source acknowledged. Failure to credit the source will also result in a misconduct investigation.

D2L Marks: Any marks posted on D2L are tentative and are subject to change (UP or DOWN).

Marking Rubric (85 points total)

Code	**Total 60 points**
Code compiles	5
Code runs	5
Stack implementation	10
Queue implementation	10
Code produces correct search trails for both versions (trailQueue and output)	20
The maze has a size at least 50x50 and is read from a file	10
An appropriate object Cell is used	10
Code is modular	5
Code contains enough documentation	5
Report	**Total 15 Points**
Discussed the difference between both strategies	5
Discussed the complexity	5
A discussion of which strategy to use and when	5
Loop Invariants	**Total 10 points**
While loop (stack)	5
While loop (queue)	5

Submissions that do not compile will receive no points. Code that compiles but does not run will not receive more than 5/100.