Assignment # 3

Stacks and Queues

Submission Dead Line: 17/10/2022

Problem 1: Rat in a Maze

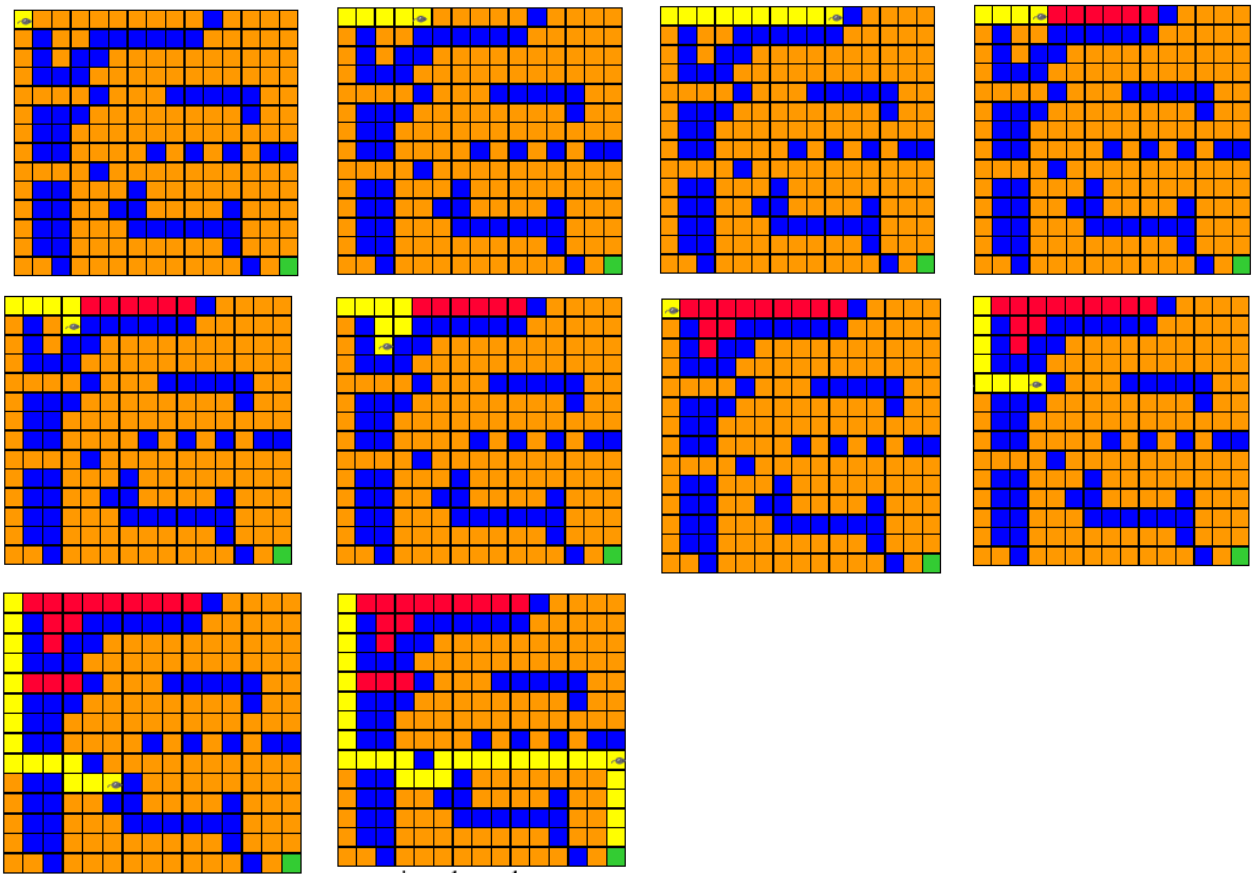
Write a C++ program that solves the maze problem. It will use a **2-D** dynamic array grid (as a maze). The dimensions of array will be taken as input from the user. Following figures shows layout of maze. You will implement the graphical user interface with highlighted blocked cells and also the final path found at end.

Here are some guide lines for the solution.

a. First step for the rat is to enter in maze (you can ask the user to provide source and destination cells).
b. Orange and green squares in figure are squares in which the rat can move.
c. Blue squares are hurdles and rat cannot move there. You can use any marking colors or values (0, 1, 2) for your game.
d. Move order is just in four directions: Right, Down, Left, Up
e. At the end the path from destination till entry cell should be marked and printed in 2-D maze matrix.
f. Placement of Hurdles should be random it should change with each instance of new game.

1. Stack Based Solution: Backtracking

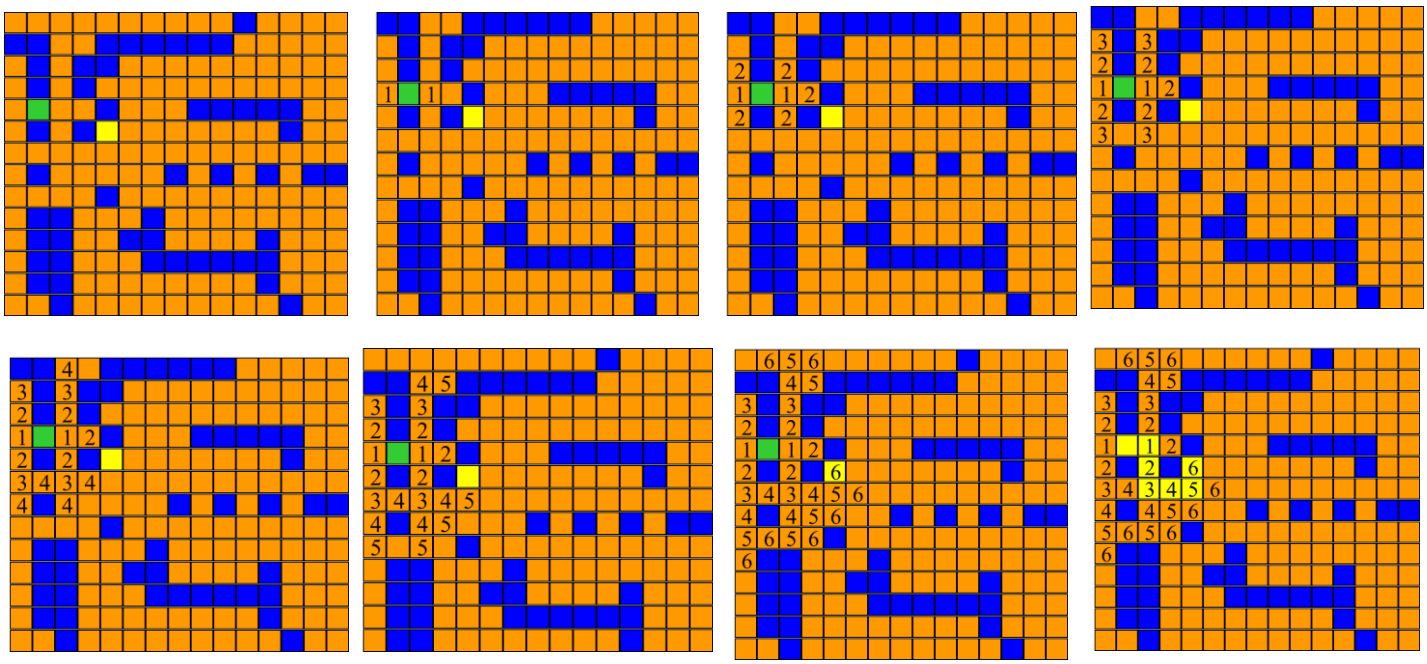
Path from maze entry to any current position of rat must operates as a stack. Block visited squares to avoid revisit them by any marker value or color (as shown red in figure).



2. Queue Based Solution: Lee's Algorithm

In this solution a queue of reachable squares is used. The queue has initially position of source cell empty, and the source square/cell has a distance value of 0.

Dequeue the cell position and examine **unreached**, **unblocked** squares adjacent to this cell. Mark them with the distance (this is 1 more than the distance value of the dequeued cell). Add position of these marked cells in the queue. Then a cell is removed from the queue and made the new examine cell. This process is repeated until the end pin is reached or the queue becomes empty. If destination cell is reached trace back the path till source by visiting labeled cells with value -1 of current cell.



3. Compare the Solutions:

Find out which solution Stack or Queue provides the shortest path.
Which solution is efficient in terms of Time and Space Complexity?

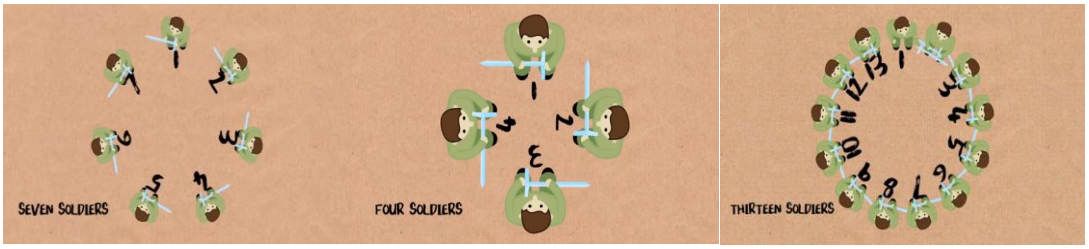
Problem 2: Josephus Problem

History: This problem is named after Flavius Josephus, a Jewish historian living in the 1st century. According to Josephus' account of the siege of Yodfat, he and his 40 soldiers were trapped in a cave by Roman soldiers. They chose suicide over capture, and settled on a serial method of committing suicide by drawing lots. Josephus states that by luck or possibly by the hand of God, he and another man remained until the end and surrendered to the Romans rather than killing themselves. This is the story given in Book 3, Chapter 8, part 7 of Josephus' The Jewish War (writing of himself in the third person)

In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game. https://en.wikipedia.org/wiki/Josephus_problem

1. People (any number N >1) are standing in a circle waiting to be executed.

2. Counting begins at a specified point (S selected randomly) in the circle and proceeds around the circle in a specified direction. You can assume that is clock wise.
3. After a specified number of people are skipped say (k-1), the next (kth) person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people (k-1), until only one person remains, and is freed.

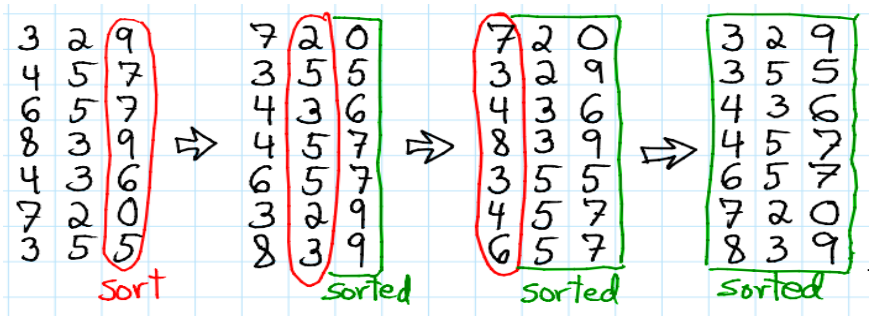


Write a C++ program that solves the Josephus problem by using a Queue data structure. Your program should take N and K input from user. Design a graphical user interface that will update with every move as shown in figure above.

Problem 3: (Radix Sort)

Radix sort is a sorting algorithm, which sorts the keys based on the values of digits in keys. It takes a queue containing **n** keys to be sorted, where each key consists of **k** number of digits, and there could be **m** possible values for each digit **0** through **m-1**. Radix sort uses an array consisting of (**m**) queues for sorting of these keys.

For example, if each key contains **k** = 3 digits and each individual digit has **m**=10 possible values 0 to 9, then it will use an array consisting of 10 queues 0-9 in sorting process of considering all digits one by one as follows.



Following is the pseudo code of this algorithm.

```
For each of the k digits in a key:
      While the queue q is not empty:
            1. Dequeue a key from q.
            2. Isolate the kth digit from key starting from the right; call
               it d.
            3. Enqueue the key in the dth queue in array of queues.
      For each of the 0 to m-1 queues in array:
            While all m queues are not empty
            Dequeue a key from queue in order starting from (0th queue) and
            enqueue it in original queue q.
```

You have to implement a two Radix sorts which can sort **integer** and **string** keys with any value of **n** and **k**.