Aaron Patrick Monte
k20059926
Week 4 - Worksheet

```
Algorithm ParenMatchIndex(X, n)
Input: An array X of n tokens, each of which is
either a grouping symbol or other symbols

Output: The psoitions of pairs of matched parentheses

Let S be an empty stack;
Let a be an empty list;
for i=0 to n-1 do
        if X[i] is an opening grouping symbol then
                S.push(X[i]);
                pairPosition1 = i;
        else if X[i] is a closing group symbol and matches
        with S.pop() then
                pairPosition2 = i;
                pair <- [pairPosition1, pairPosition2];
                a.insert(pair);
        return pair
```

Exercise 1

How would you modify the Parentheses Matching Algorithm if you wanted as output the pairs of positions of matched parentheses.

For example, for the input

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
(	[	(	a	+	b	)	*	c	+	d	*	e	]	/	{	(	f	+	g	)	-	h	}	)

the output should be:

(2,6) (1,13) (16,20) (15,23) (0,24)

Parentheses Matching Algorithm

X:	(	[	a	+	3	]	*	....

Algorithm ParenMatch(X,n):
Input: An array X of n tokens, each of which is either a grouping symbol or other symbols (for example: variables, arithmetic operators, numbers)
Output: **true** if and only if all the grouping symbols in X match
Let S be an empty stack
for $i = 0$ **to** n-1 **do**
  if $X[i]$ is an opening grouping symbol **then**
    S.push($X[i]$)
  else if $X[i]$ is a closing grouping symbol **then**
    if S.isEmpty() **then**
      return false { nothing to match with }
    if S.pop() does not match the type of $X[i]$ **then**
      return false { wrong type of closing symbol }
if S.isEmpty() **then**
  return true { every symbol matched }
else return false { some symbols were never matched }

match $X[i]$ with symbol pop'ed from the stack

 Stacks and Queues 15

Even if r is kept empty, the queue will be be considered full regardless, and you won't be able to enqueue any more values.

An example solution may be to treat r as an individual index, where Q[r] is assignable, but that would defeat the purpose of having an abstract data type implemenattion of a queue. Furthermore, in a circular queue context, you must always check if r is empty in all your operations, otherwise it may return null.

Exercise 2

In the array-based implementation of Queue, the array location ***r*** is kept empty. Consequently, when the queue is considered full, there is still one empty location in the array.

Could we put another element in that final empty location, and declare that the queue is full only if the size of the queue is equal to the size (length) of the array? What would be a problem with this approach and how that problem could be fixed?

Exercise 3

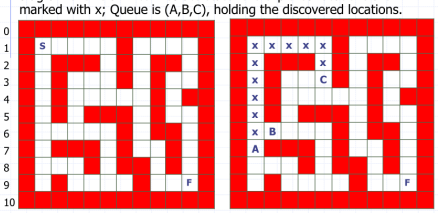
A. An example of a maze is shown on the next slide. Consider the following algorithm for exploration of such a maze to find out if the Finish location is reachable from the Start location. We can go from the current location only to a neighbouring location (sharing one side).

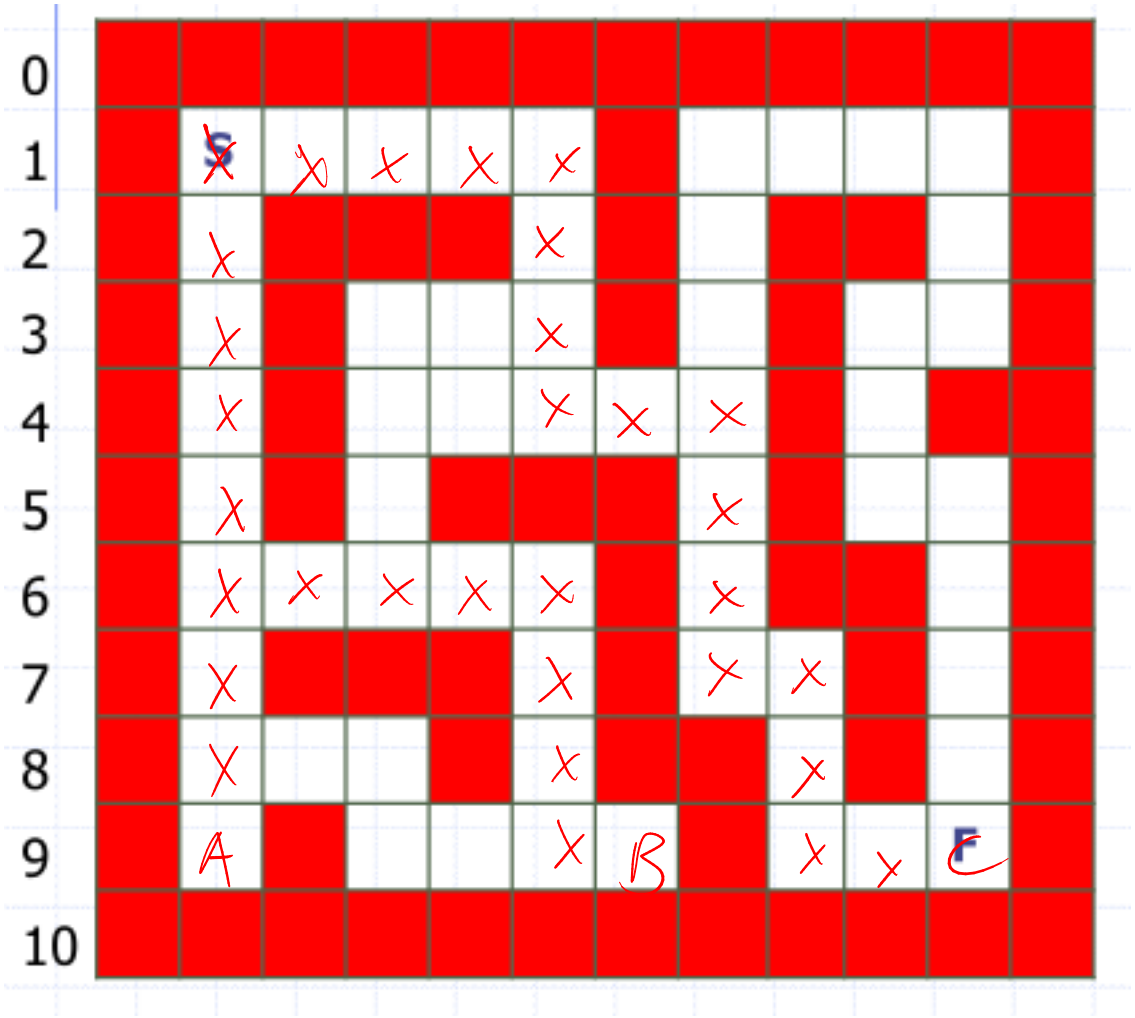
Algorithm:

- Each location is either unknown, discovered or explored.
- Initially Start is discovered and all other locations are unknown.
- All discovered locations are kept in a Queue.
- While Queue is not empty:
  - take (remove) a (discovered) location from Queue,
  - mark this location as explored,
  - mark all its unknown neighbouring locations as discovered and add them to Queue.
- Stop when Finish is discovered (Finish is reachable from Start) or when Queue becomes empty (Finish is not reachable).

Exercise 3 (cont.)

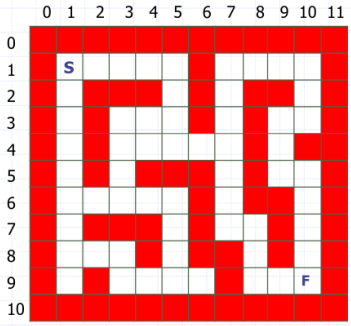
Show the status of each location (unknown, discovered, or explored) when Finish is discovered. Assume the neighbouring locations of the current location are considered in the order: S, E, N, W. The right diagram shows an intermediate state: the explored locations are marked with x; Queue is (A,B,C), holding the discovered locations.





Exercise 3 (cont.)

B. How could we extend this algorithm to output a path from the location Start to the location Finish, if Finish is reachable from Start?



Implement a stack data structure alongside the queue.

If the queue reaches a dead end, pop values until you get to a point where you can traverse the maze again.

Once you reach the final point, reverse the values currently in the stack and return that value.

Exercise 3 (cont.)

C. What would happen, if we used Stack instead of Queue in this algorithm? Trace the computation showing the status of the locations (unknown, discovered, or explored) and the contents of the Stack. Assume the neighbouring locations of the current location are considered in the order: S, E, N, W.