DSALG – Data Structures and Algorithms

Worksheet 1: Stack, Queue

Introduction

This worksheet is intended to get you started using different data structures for your programs. You will begin with the ADTs of Stacks and Queues containing the function definitions from the lecture/booklet. Create the Stack and Queue first, and experiment by some exercises.

The answer to each exercise is not unique. Feel free to add your own solutions to the exercises. Please do have an evaluation of the BigO (time complexity and space complexity) of your solution.

Hint: Don't get stuck in any particular exercise for too long. Please set a maximum time spent on each exercise. You are encouraged to attempt most exercises and leave the tricky ones for a discussion with me during the practical sessions or the drop-in sessions. A short discussion will help sort the question and clear up your concerns quickly.

Programming exercises

1. Implement a Queue using an Array/List:

   Recall that a Queue ADT must support the following methods:

● Enqueue	Add an element to the tail (rear) of the queue
● Dequeue	Remove element from the head (front) of the queue
● Peek	Examine element at the head of the queue
● Length	Give the size (number of elements) of the queue

2. Implement a Stack using an Array/List:

   Recall that a Stack ADT must support the following methods:

● Push	Add an element to the top of the stack
● Pop	Remove an element from the top of the stack
● Peek	Examine element at the top of the stack
● Length	Give the size (number of elements) of the stack

3. (Advanced) Implement a Stack using the Queue that you defined in Ex 1.

   Hint: Pop in a stack deletes the last inserted element. The access to the last inserted element in a queue (the tail) requires removing (dequeueing) all the preceding elements. How about the dequeued elements? Extra storage might be needed.

4. (Advanced) Implement a Queue using the Stack that you defined in Ex 2.

   Hint: Dequeue in a queue deletes the first inserted element. The access to the first inserted element in a stack requires removing (popping) all the succeeding elements. How about the popped elements? Extra storage might be needed.

5. Design an algorithm to reverse an input Queue using Stacks and Queues.

   (For example, Queue “12345” -> Queue “54321”)

6. (Advanced) Implement a Circular Queue using an Array/List:

   Recall that a Circular Queue has its last position (tail) connected back to the first position (head) to make a circle:

● isEmpty	How do you determine if a circular queue is empty?
● isFull	How do you determine if a circular queue is full?

7. Design an algorithm to remove all the adjacent duplicate letters from an input string continuously until there are no adjacent duplicate letters in the output string using a Stack or a Queue. Can you implement using the other ADT instead?

   (For example, “abbccd” -> “ad”, “dsallasg” -> “dg”, “abccbadd” -> “”)

8. The brackets in a string are valid when open brackets are closed by the same type of brackets in the correct order. Design an algorithm to determine if the brackets in an input string are valid using a Stack or a Queue. Can you implement this using the other ADT instead?

   (For example, “(abc)” -> True, “[ds](alg)” -> True, [a{b]c} -> False)