Data Structure Lab Assignment (CS 2172)
Assignment 4: Queue

Time: 1 week

1. Implementation of an Integer Queue

In this assignment you are required to implement a queue where integer data can be en-queued and de-queued. You should define (**typedef**) an appropriate type called ***queue*** such that multiple variables of type ***queue*** can be defined.

Your implementation should support the following functions (interface) for the queue.

1. ***queue createIntegerQueue(int queueSize)*** - This allocates space for the queue to hold maximum "***queueSize***" number of integers and initializes that space. Its return type is "***queue***". The function returns ***NULL*** if creation of the queue fails.
2. ***int enqueueInteger(queue q, int d)*** - It en-queues the data ***d*** in the queue ***q***. It returns 1 if the operation is successful. If the operation fails (say, when queue ***q*** is full and ***d*** cannot be en-queued), the function returns 0.
3. ***int dequeueInteger(queue q, int *dp)*** - It de-queues from the queue ***q*** and stores the dequeued element at address ***dp***. It returns 1 if the operation is successful. If the operation fails (say, when queue s is empty and ***dequeueInteger()*** is attempted), the function returns 0.
4. ***int freeIntegerQueue(queue q)*** - It frees the space allocated for queue ***q***. It returns 1 if the operation is successful. If the operation fails (say, ***q*** does not refer to a valid queue), the function returns 0.
5. ***int isIntegerQueueFull(queue q)*** - It returns 1 if the queue associated with ***q*** is full. The function returns 0 otherwise. If ***q*** does not refer to a valid queue then too the function returns 1.
6. ***int isIntegerQueueEmpty(queue s)*** - It returns 1 if the queue associated with ***q*** is empty. The function returns 0 otherwise. If ***q*** does not refer to a valid queue then too the function returns 1.

Write a suitable main() function to demonstrate that your functions are working as desired.

Note: You should implement queue functionality using the circular queue concept. For better understanding, initially avoid having a ***count*** variable (which signifies the number of elements in the queue). Then finally, introduce the ***count*** variable to make it simple.

2. Using above Queue implementation, simulate the following

Assume there is one queue created as ***myQueue*** of size ***N***. Also, consider that a series of **positive** integers are read from the user and queued into ***myQueue***. The process of populating into the queue continues till the **queue overflows**. Now perform the following steps of operation on each element of the queue.

- ***Dequeue*** an element from ***myQueue***. Let's call that element as ***qElement***.
- If the value of the ***qElement*** is positive, then
  - process the element as follows
    - ***qElement = qElement – rValue***; where ***rValue*** is a randomly generated positive integer between 1 – 9
    - ***Enqueue*** back the updated ***qElement*** into ***myQueue.***
- Else
  - Ignore ***qElement***

This process continues till the time **queue underflows,** such that no element present in the queue.

3. Optional Assignment: Once problems 1 and 2 are completed, then attempt problem.

Implement a Queue using two Stacks: This assignment aims to implement a queue (Q) using two stacks (S1 and S2). Employ two different approaches for this and try to understand why one idea is better than the other. The pseudocode for the approaches is as follows:

Approach 1	Approach 2

`enqueue(Q, a) {`
`  push(S1, a);`
`}`
 	
`dequeue(Q){`
`  if (S1 is empty)`
`  return(error);`
 	
`  while(S1 is not empty){`
`  push(S2, pop(S1));`
`  }`
`  r <- S2.pop();`
`  while(S2 is not empty){`
`  push(S1, pop(S2));`
`  }`
`  return(r);`
`}`
	`enqueue(Q, a) {`
	`  push(S1, a);`
	`}`
	
	`dequeue(Q){`
	`  if (S1 and S2 are empty)`
	`  return(error);`
	
	`  if (S2 is empty){`
	`  while(S1 is not empty) {`
	`  push(S2, pop(S1));`
	`  }`
	`  }`
	`  return(pop(S2));`
	`}`

- ✓ Argue for yourself that the above two correctly implements a queue using two stacks.
- ✓ Try to figure out why the second implementation is much better than the first one.