Data Structures
Lists: Circular Queue and Linked List

Won Kim

Spring 2022

Circular Queue

Circular Queue (Ring Buffer)

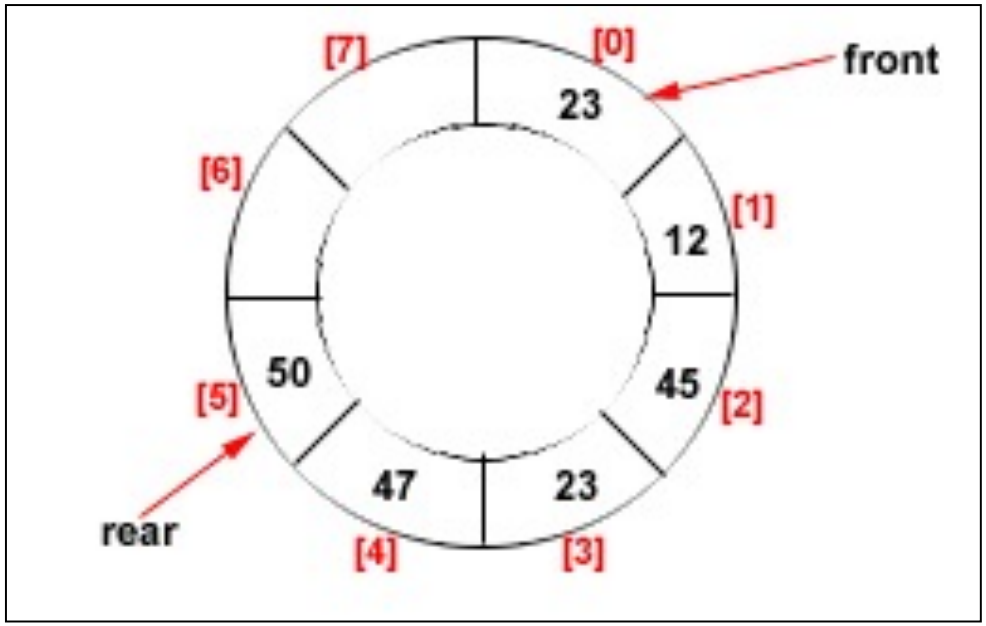
- The queue forms a circle (ring).
- The queue may be implemented using a linked list or an array.
- If an array is used, once inserts reach the end of the array, new inserts will be made to the start of the array.

Circular Queue: Empty and Full State

- Empty state
  - front = rear
  - (may be anywhere on the array)
- Full state
  - front = rear
- Solutions
  - use all N elements of the N-element array
    - queue overwrite (i.e. force an insert)
    - queue overflow (i.e. reject insert)
  - use only N-1 elements of the N-element array

Circular Queue: Full or Overflow

- If the Queue becomes full, the Front of the Queue = the Rear, if and only if the Front has moved forward.
- Otherwise it will be "Queue overflow".



Inserting into a Circular Queue

Initially rear=0 and front=0 .

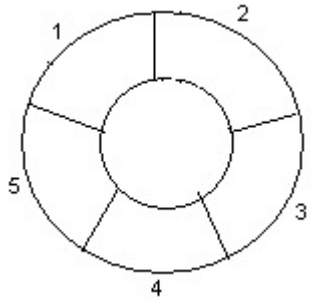
- 1. If front=0 and rear=0, set front=1 and go to step 4.
- 2. If front=1 and rear=n or front=rear+1, "circular queue overflow"(finish).
- 3. If rear=n, set rear=1 and go to step 5.
- 4. Set rear=rear+1
- 5. Store in rear of array.
- 6. Finish

Deletion from a Circular Queue

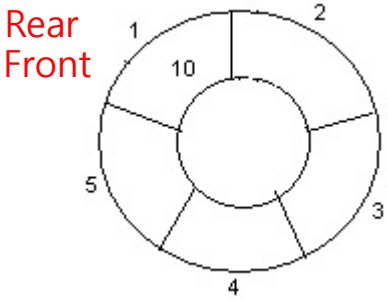
- 1. If front=0,“circular queue underflow” and finish.
- 2. Delete the front item
- 3. If front=n, set front=1 and finish.
- 4. If front=rear, set front=0 and rear=0, and finish.
- 5. Set front=front+1
- 6. Finish

Example

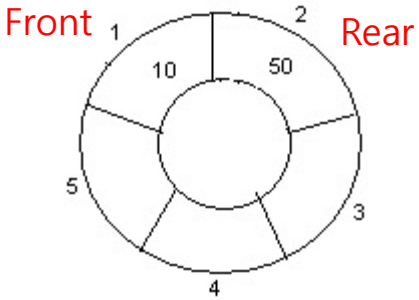
1. Initially, Rear = 0, Front = 0.



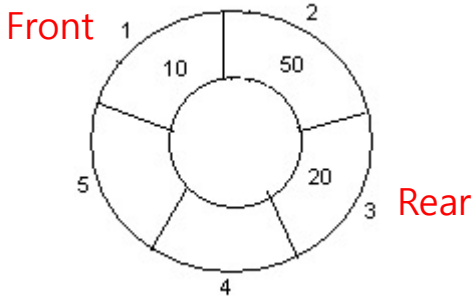
2. Insert 10, Rear = 1, Front = 1.



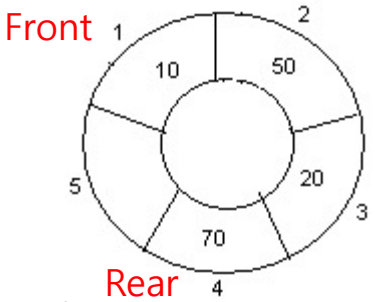
3. Insert 50, Rear = 2, Front = 1.



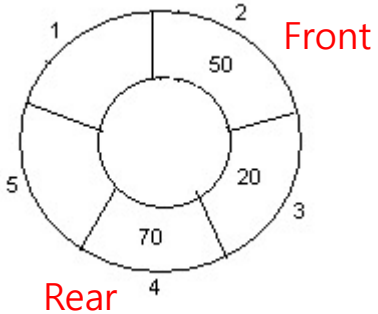
4. Insert 20, Rear = 3, Front = 1.



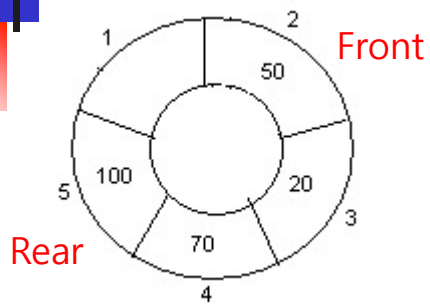
5. Insert 70, Rear = 4, Front = 1.



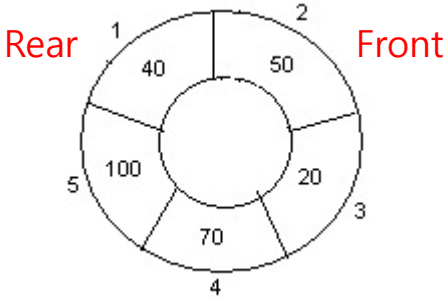
6. Delete front, Rear = 4, Front = 2.



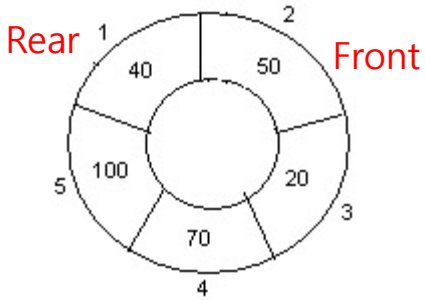
7. Insert 100, Rear = 5, Front = 2.



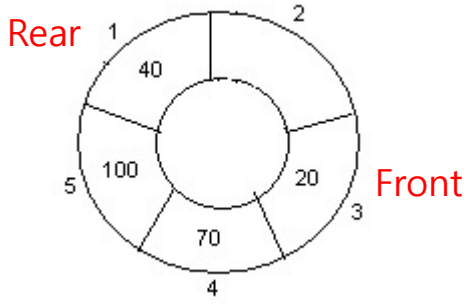
8. Insert 40, Rear = 1, Front = 2.



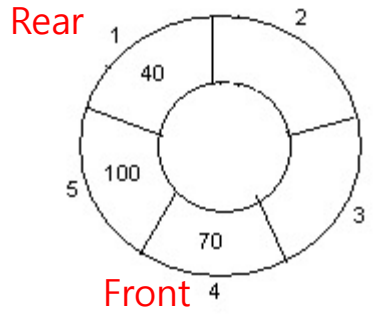
9. Insert 140, Rear = 1, Front = 2.
As Front = Rear + 1, so Queue overflow.



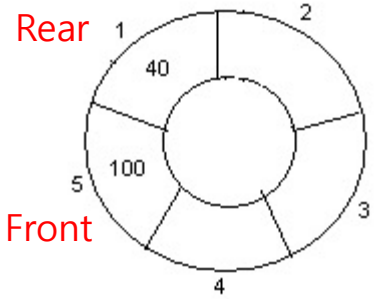
10. Delete front, Rear = 1, Front = 3.



11. Delete front, Rear = 1, Front = 4.



12. Delete front, Rear = 1, Front = 5.



Exercise

- Using buffer overwrite, do the same sequence of inserts and deletes shown in pages 8-9 (Draw the changes in the ring buffer).
- Using buffer overflow, create a ring buffer with an array of 4 elements, and do the same sequence of inserts and deletes shown in pages 8-9 (Draw the changes in the ring buffer).

Review on Linked List with struct

Programming Homework (PHW) 1

PHW 1-1 (10 points)

- Implement a ring buffer with an array of 5 elements that uses buffer overflow.
- Test the program using the sequence of inserts and deletes shown in class (pages 8-9).

PHW 1-2 (15 points)

- Implement and Test a Stack Program, Using a Singly Linked List.
  - for the same 4 functions (of Lab 1-1).
  - ** stack_full test??

PHW 1-3 (15 points)

- Implement and Test a Queue Program, Using a Singly Linked List.
  - for the same 4 functions (of Lab 1-2).
  - ** queue_full test?

PHW 1-4 (15 points)

- Implement a Virtual Integer Stack using 2 Queues.
  - Must use the Queue program (of Lab 1-2).

Solution Hints

- Create queue1 and queue2
- push: use queue1
- pop:
  - dequeue queue1, and enqueue queue2 (except the rear data)
  - dequeue queue2, and enqueue queue1

Points To Be Deducted For…

- changing the dequeue function
  - dequeue function should simply delete the front of the queue (and nothing else)
- directly copying data between the 2 queues
  - (e.g.) enqueue2[k] = enqueue1[j];
  - (should use enqueue2(dequeue1()))

End of Class