Queues

- A **Queue** is a sequential organization of items in which first element entered is first removed. They are often referred to as FIFO, which stands for "first in first out."
- **Examples:** standing in a line, printer queue.
- The basic operations are:
  - *insert(*x*)* places x at the beginning of the queue.
  - *remove()* returns and deletes the item at the end of the queue.

Example

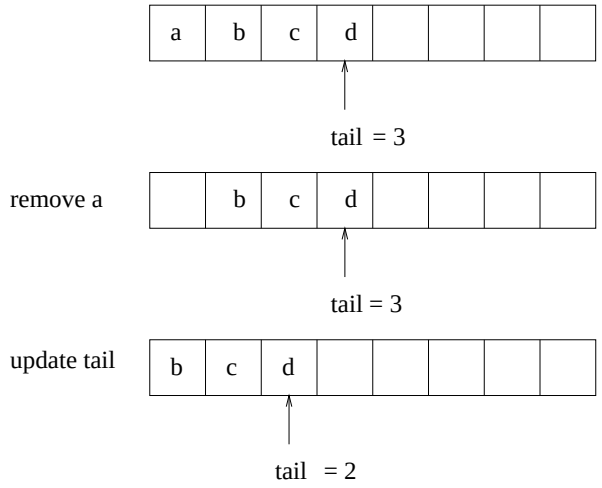
Operation	Queue	Return
CreateQueue	()	
insert(7)	(7)	
insert(8)	(7,8)	
insert(5)	(7,8,5)	
remove()	(8,5)	7
remove()	(5)	8

Queue Applications

- Operating systems:
  - Queue of jobs or processes ready to run (waiting for CPU):
  - Queues of processes waiting for I/O.
  - Files sent to printer
- Simulation of real-world queuing systems:
  - Customers in a grocery store, bank, etc.
  - Orders in a factory
  - Hospital emergency room or doctor's office
  - Telephone calls for airline reservations, customer orders, information, etc.
- Problem applications:
  - Topological ordering: given a sequence of $events$, and pairs (a, b) indicating that event a should occur prior to b, provide a schedule.

Queues: Naive Implementation

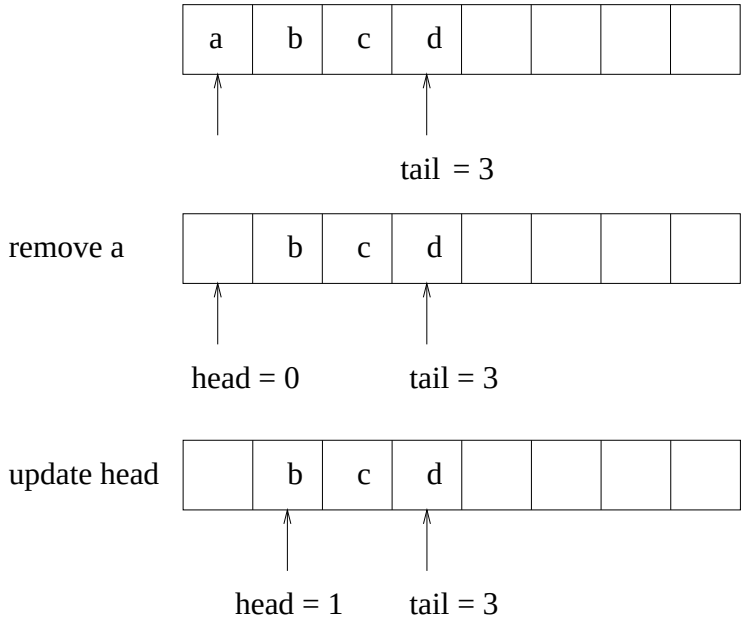
- Using array:
  - Store items in an array. The head it the first element, and the tail is indicated by a variable *tail*.
  - *insert*(x) is easy: increment $tail$, and insert element.
  - *remove*() is inefficient: all elements have to be shifted. Thus, *remove* is $O(n)$.



- How can we improve this?

Queues: A Better Implementation

- Keep track of both the *head* and the *tail*.
- To remove, increment $front$.



- There is still a problem. What is it, and how can we fix it?

Queues: Circular Array Implementation

- Previous implementation is $O(1)$ per operation, which is great.
- However, after n inserts (where n is the size of the array), the array is full even if the queue is logically nearly empty.
- **Solution:** Use wraparound to reuse the cells at the start of the array. To $increment$, add one, but if that goes past end, reset to zero.

- How do you detect a *full* or *empty* queue?
- We will give a simplified implementation for the queue data structure. A better implementation would detect an empty (full) queue before performing a dequeue (enqueue) operation.

Queue C++ Declaration

- Here is a C++ declaration of an integer queue using a “circular” array:

```
class Queue {
private:
  int *queue;
  int cap,head,tail;
public:
Queue(int s=100) {
      cap= s;
      queue = new int[cap];
      head = 0; tail = 0; }
~Queue() { delete queue; }
void Enqueue(int v) {
        queue[tail] = v;
        tail = (tail+1) % cap;
     }
int Dequeue() {
       int t = queue[head];
       head = (head + 1) % cap;
       return t;
    }
int Empty() {return (head == tail ); }
int Full() {return (head==(tail+1)%cap); }
};
```

Queue JAVA Declaration

```
public class Queue {
   private int[] queue;
   private int cap,head,tail;
   public Queue(int s) {
      cap=s;
      queue = new int[cap];
      head = 0;
      tail = 0; i
    }
    public void Enqueue(int v) {
       queue[tail] = v;
       tail = (tail+1) % cap;
    }
    public int Dequeue() {
       int t = queue[head];
       head = (head + 1) % cap;
       return t;
    }
    public int Empty() {
       return (head == tail ); }
    public int Full() {
       return (head == ((tail+1)%cap)); }
}
```

Example of Queue Using Circular Array

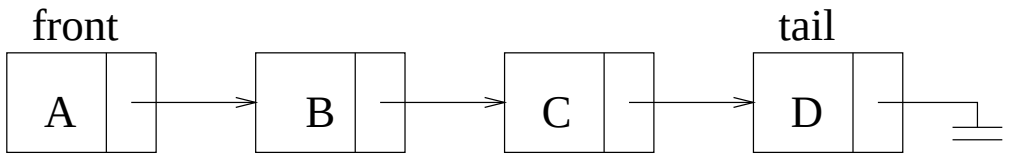
- Here we use an array $E[0..3]$ to store the elements and the variables h (head) and t (tail) to keep track of the beginning and end of the queue. The value *R* is the return value of the operation.

Operation	h	t	E0	E1	E2	E3	R
create	0	0	?	?	?	?	
insert(55)	0	1	55	?	?	?	
insert(-7)	0	2	55	-7	?	?	
insert(16)	0	3	55	-7	16	?	
remove()	1	3	55	-7	16	?	55
insert(-8)	1	0	55	-7	16	-8	
remove()	2	0	55	-7	16	-8	-7
remove()	3	0	55	-7	16	-8	16
insert(11)	3	1	11	-7	16	-8	

- Note that some of the values remain physically in the array, but are logically no longer in the queue.

Queues: Linked List Implementation

- We can maintain $front$ and $tail$ pointers.
- $remove$: advance $front$.
- $insert$: add to end of list and adjust $tail$.



- The details are not very hard to work out, so will not be presented.