Queues

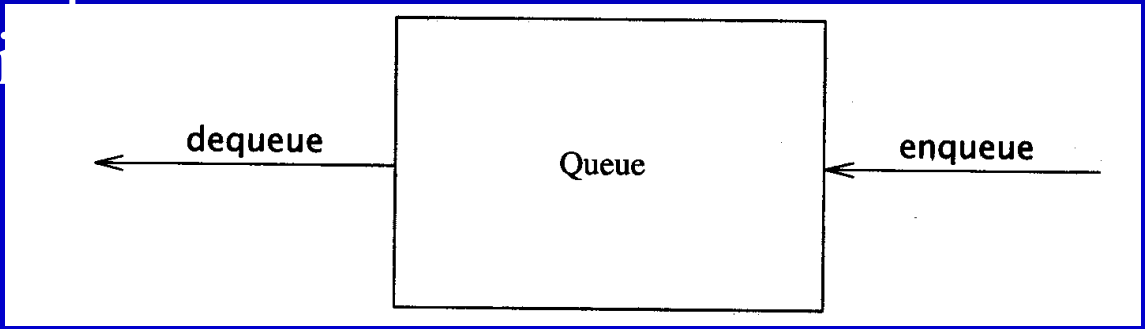
Data Structures

Queue

- Like a stack, a *queue* is also a list. However, with a queue, insertion is done at one end, while deletion is performed at the other end.
- Accessing the elements of queues follows a First In, First Out (FIFO) order.
  - Like customers standing in a check-out line in a store, the first customer in is the first customer served.

The Queue

- Another form of restricted list
  - Insertion is done at one end, whereas deletion is performed at the other end
- Basic operations:
  - enqueue: insert an element at the rear of the list
  - dequeue: delete the element at the front of the li



Implementation of Queue

- Just as stacks can be implemented as arrays or linked lists, so with queues.
- Dynamic queues have the same advantages over static queues as dynamic stacks have over static stacks

Empty or Full?

- Empty queue
  - back = front - 1
- Full queue?
  - the same!
  - Reason: n values to represent n+1 states
- Solutions
  - Use a boolean variable to say explicitly whether the queue is empty or not
  - Make the array of size n+1 and only allow n elements to be stored
  - Use a counter of the number of elements in the queue

`Queue` Class

- Attributes of `Queue`
  - `front/rear`: front/rear index
  - `counter`: number of elements in the queue
  - `maxSize`: capacity of the queue
  - `values`: point to an array which stores elements of the queue
- Operations of `Queue`
  - `IsEmpty`: return true if queue is empty, return false otherwise
  - `IsFull`: return true if queue is full, return false otherwise
  - `Enqueue`: add an element to the rear of queue
  - `Dequeue`: delete the element at the front of queue
  - `DisplayQueue`: print all the data

Create Queue

- `Queue(int size = 10)`
  - Allocate a queue array of `size`. By default, `size = 10`.
  - `front` is set to `0`, pointing to the first element of the array
  - `rear` is set to `-1`. The queue is empty initially.

```
Queue::Queue(int size /* = 10 */) {
      values              =     new double[size];
      maxSize             =     size;
      front               =     0;
      rear                =     -1;
      counter             =     0;
}
```

Queue overflow

- The condition resulting from trying to add an element onto a full queue.

```
if(!q.IsFull())
  q.Enqueue(item);
```

Queue underflow

- The condition resulting from trying to remove an element from an empty queue.

```
if(!q.IsEmpty())
  q.Dequeue(item);
```