Queue ADT

- Abstract Data Type: Queue
  - First In First Out (FIFO)

Queue: FIFO

- Queues are conceptually like waiting in line.
  - First in first out behavior

Queues in Programming

- "Waiting in line" happens all the time in programs
  - Networked programs: incoming connection requests
  - Queued by OS until application can handle them
  - Things to do at certain times may be queued
    - Might be "priority queue" (later)
  - Some algorithms use queues
    - Compute "more things to do"
    - Put them in a queue
    - Take "next thing to do" from queue
    - (Will see some of these later too)
Queue ADT

- Can have a Queue of pretty much anything
  - Good use of a template!

```
template<typename T>
class Queue {
public:
    void enqueue(T item);  //put item in line
    T    dequeue();        //remove front
    T &  peek();           //examine front
    bool isEmpty() const;  //check empty
    int  size() const;     //how many?
};
```
Queue ADT

- Can have a Queue of pretty much anything
  - Good use of a template!
  
  template<typename T>
  class Queue {
    public:
      virtual void enqueue(T item) = 0;
      virtual T dequeue() = 0;
      virtual T & peek() = 0;
      virtual bool isEmpty() const = 0;
      virtual int size() const = 0;
  };

  ADT: so all methods abstract (pure virtual)

Queue Implementation

- We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

  Conceptually, think of head and tail indices as "pointing" at the queue

  Enqueue at the tail, and increase the tail index.

  Enqueue A: Put A in data[tail]. Increment tail

  We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

  Enqueue B: Put B in data[tail]. Increment tail

  We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

  Dequeue from head, and increment head index

  Dequeue. Result is data[head] (A)

  We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

  Dequeue. Result is data[head] (B)
Queue Implementation

- We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

Suppose we enqueue more things (C,D,...,J).

head = 2
tail = 0
data =

A B C D E F G H I J

• The tail has to wrap back around to 0 (increment mod the array size)

Adding two more items (K,L) results in a full queue.

head = 2
tail = 0
data =

K L C D E F G H I J

- We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

If we dequeue a bunch of items....

head = 0
tail = 2
data =

K L C D E F G H I J

• We could implement a Queue with an array
  - Particularly good if "fixed size" queue
  - Can also implement it with a Linked List (coming soon!)

Let's go back to our full queue for a second....

- What if we try to enqueue something?
  - Option 1: Its an error (fixed size queue)
    - Provide isFull() in interface, design code which uses to prevent
  - Option 2: Make the queue larger
    - Involves copying data

Growing our queue:

- Need more space (allocate it)
- Conceptually tail (place to add) moves to the start of new space
- Copy the data....

Finish this by freeing old memory, updating information

- delete[] data
- head = 0 (now conceptually in new space, at the start)
- data = newSpace
Queue Growth

- Growing the queue: $O(N)$ operation
  - We need to copy $N$ elements from the old to the new
  - Do this occasionally? Fine
  - Do it frequently? Performance will be slow
  - $N$ adds will have $O(N)$ performance
  - Can’t make worst case better, but can make average case better
    - Amortize cost of copying over more adds between copies
    - Double size of array each time it needs to grow
      - Now we know we get $N$ adds before we do $N$ work
      - $N/N = 1$, maintain $O(1)$ average time addition

- Good rule for growing array based structures in general:
  - Double the size each time you must grow
  - Amortize your copying costs

Wrap Up

- Queue ADT
  - First in First Out: FIFO
  - Useful when data needs to be processed in the order it came in
  - Example C++ template for the ADT
    - Many variants in interface, depending on exactly what you want
  - Conceptual discussion of array based implementation