ECE 2574: Data Structures and Algorithms - Queue ADT

C. L. Wyatt

Today we will look at the Queue ADT:

- Single-ended Queue ADT
- Double-ended Queue (deque) ADT
- Array implementations
- Linked-list implementations
- Examples

The Queue ADT is a list in which the first item inserted is the first item retrieved.

Unlike the stack, a queue is a "fair" system.
Example: the first person in line is the first person to be served.
Queues have a front and a rear (also called the back).

Single-ended Queue ADT

A queue is a list of items with one end denoted the front, the other the back.

```
+isEmpty(): boolean
+enqueue(newEntry: ItemType): boolean
+dequeue(): boolean
+peekFront(): ItemType
```

Note the similarity to the Stack ADT.

Warmup

After the following operations, what are the contents of the queue ?

```
1. queue q;
2. q.enqueue(41);
3. q.enqueue(12);
4. q.enqueue(8);
5. q.dequeue();
6. q.enqueue(36);
7. q.dequeue();
8. q.dequeue();
```

Interface for Queue

See `abstract_queue.h`.

Array Implementation of Single-ended Queue

This is a straight-forward reuse of ArrayList using composition or private inheritance.

- pick one position as the front, the other as the back (e.g. front $= 0$, back $=$ getLength())
- enqueue just calls insert at back position
- dequeue just calls remove at the front position
- peekFront just calls getEntry at the front position

What is the complexity of these operations?

Linked Implementation of Single-ended Queue

This is also a straight-forward reuse of LinkedList using composition or private inheritance.

- pick one position as the front, the other as the back (e.g. front $= 0$, back $=$ getLength())
- enqueue just calls insert at back position
- dequeue just calls remove at the front position
- peekFront just calls getEntry at the front position

What is the complexity of these operations?

Since the operations are the same for ArrayList and LinkedList implementations of Queue, we can make it generic with respect to the List implementation.

See “queue.h”

Double-ended Queue (deque) ADT

A Queue in which you can enqueue or dequeue at either end is called a double-ended queue or *deque* (pronounced "deck").

```
+isEmpty(): boolean
+enqueue_front(newEntry: ItemType): boolean
+dequeue_front(): boolean
+peekFront(): ItemType
+enqueue_back(newEntry: ItemType): boolean
+dequeue_back): boolean
+peekBack(): ItemType
```

This gives a combination of a stack and a queue.

Interface for Deque

See `abstract_deque.h`.
An adaptor using AbstractList is similar to the regular queue.

- pick one position as the front, the other as the back (e.g. front = 0, back = getLength())
- enqueue_front just calls insert at front position
- enqueue_back just calls insert at back position
- dequeue_front just calls remove at the front position
- dequeue_back just calls remove at the back position-1
- peekFront just calls getEntry at the front position
- peekback just calls getEntry at the back position

Array Implementation of fixed-size deque (ring buffer)

This is a faster way to implement a deque (or a queue) of **fixed size**.
Sometimes called a *ring buffer*.
See `ring_buffer.h\.txx`.

Implementation of deque using a combination of linked-list and arrays

This is the most efficient way to implement a deque that can grow arbitrarily on modern systems.
See `std::deque`.
We will look at it in detail Friday.

Applications of Queues

Just a few examples:

- Message Queues between processes and threads
- Breadth-first search on graphs
- Printer Jobs (or any buffer)
- Embedded systems: interrupt handler enques, main thread deques.

Next Actions and Reminders

- Read CH pp. 379-388
- Warmup due before noon on Wednesday.
- Program 3 due Tommorrow by 11:59 pm.