CSCI 104L Lecture 8: Stacks and Queues

class QueueADT {
    public:
        void enqueue(const T& data);
        const T& peekfront() const; // look at the oldest element
        void dequeue(); // remove the oldest element
};

Notice the following:

• Enqueue cannot change the input parameter.
• Whomever called peekfront cannot change the return parameter they received.
• Peekfront cannot change any data members in the Queue (this cannot change).
• Passing a parameter by const reference allows you to avoid copying the input parameter, while promising the user you won’t change their data.

Question 1. What data structure should you use to implement a queue? How would you implement it?

Question 2. Is there anyway to do this efficiently with an array?

class StackADT {
    void push(const T& data);
    const T& top() const; // look at the newest element
    void pop(); // remove the newest element
};

Question 3. What data structure should you use to implement a stack? How would you implement it?

We can define a stack recursively; a stack is either:

1. The empty stack, or
2. S.push(data), where S is a stack, and data is a data item.

The following “stack axioms” describe stack behavior:

1. For all stacks s, s.push(data).top() = data
2. For all stacks s, s.push(data).pop() = s

Question 4. Use the above 4 points to determine the result of the following operations:

• s.push(5).push(4).pop().top()
• s.push(5).top().pop()
An algorithm you would normally solve recursively can instead be solved with a stack.

**Question 5.** Which strings are properly parenthesized?

1. ([ab])
2. ab{c}
3. ([ab{c}]de())

A queue can also be defined recursively; one is either:

1. The empty queue, or
2. Q.enqueue(data), where Q is a queue, and data is a data item.

A **Deque** allows you to add and remove elements from both ends.

**Question 6.** How should a deque be implemented?

**Question 7.** Why stacks and queues, when Deques are more powerful?

**Question 8.** Why Deques, when a vector is more powerful?