Stacks and Queues

C.S. Marron
cmarron@umbc.edu

CMSC 341 — Data Structures

Abstract Data Types

ADT Definition

1. Abstract model for a data type
2. Defined by behavior (semantics)
3. From the *user* point-of-view

This is in contrast to *Data Structures* which are representations of data from the *implementer* point-of-view.

Sparse Vector ADT

Example

A *sparse vector* is an ADT. We have expectations of how we can interact with a sparse vector as a user, but we don't need to know how it is implemented. We saw that it could be implemented with either an array or linked list data structure.

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

1. `push(e)`: place the element e on the stack

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

1. `push(e)`: place the element e on the stack
2. `pop()`: return the element most recently placed on the stack and remove it from the stack

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

1. `push(e)`: place the element e on the stack
2. `pop()`: return the element most recently placed on the stack and remove it from the stack
3. `top()`: return the element most recently placed on the stack, but do not remove it from the stack

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

1. `push(e)`: place the element e on the stack
2. `pop()`: return the element most recently placed on the stack and remove it from the stack
3. `top()`: return the element most recently placed on the stack, but do not remove it from the stack

The Stack ADT

Definition

A Stack is a last-in, first-out (LIFO) ADT that supports the following operations:

1. `push(e)`: place the element e on the stack
2. `pop()`: return the element most recently placed on the stack and remove it from the stack
3. `top()`: return the element most recently placed on the stack, but do not remove it from the stack

Stacks may provide additional operations such as `size()` and `empty()`.

Stack Implementation

Array or List?

1. How can we implement the Stack ADT with an array?
2. How can we implement it with a linked list?
3. What is the running time of the CRUD operations?
4. Are the implementations memory-efficient?
5. Which do you prefer and why?

The Queue ADT

Definition

A Queue is a first-in, first-out (FIFO) ADT that supports the following operations:

The Queue ADT

Definition

A Queue is a first-in, first-out (FIFO) ADT that supports the following operations:

1. `enque(e)`: place the element e in the queue

The Queue ADT

Definition

A Queue is a first-in, first-out (FIFO) ADT that supports the following operations:

1. `enque(e)`: place the element e in the queue
2. `dequeue()`: return the element that has been in the queue the longest and remove it from the queue

The Queue ADT

Definition

A Queue is a first-in, first-out (FIFO) ADT that supports the following operations:

1. `enque(e)`: place the element e in the queue
2. `dequeue()`: return the element that has been in the queue the longest and remove it from the queue
3. `front()`: return the element that has been in the queue the longest, but do not remove it from the queue

The Queue ADT

Definition

A Queue is a first-in, first-out (FIFO) ADT that supports the following operations:

1. `enque(e)`: place the element e in the queue
2. `dequeue()`: return the element that has been in the queue the longest and remove it from the queue
3. `front()`: return the element that has been in the queue the longest, but do not remove it from the queue

Stacks may provide additional operations such as `size()` and `empty()`.

Queue Implementation

Array or List?

1. How can we implement the Queue ADT with an array?
2. How can we implement it with a linked list?
3. What is the running time of the CRUD operations?
4. Are the implementations memory-efficient?
5. Which do you prefer and why?

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue
4. `eraseBack()`: remove the item at the back of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue
4. `eraseBack()`: remove the item at the back of the dequeue
5. `front()`: return the element at the front of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue
4. `eraseBack()`: remove the item at the back of the dequeue
5. `front()`: return the element at the front of the dequeue
6. `back()`: return the element at the back of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue
4. `eraseBack()`: remove the item at the back of the dequeue
5. `front()`: return the element at the front of the dequeue
6. `back()`: return the element at the back of the dequeue

Variation: Double-Ended Queue ADT

Definition

A Dequeue is a combination of a queue and a stack. It supports the following operations:

1. `insertFront(e)`: place the element e at the front of the dequeue
2. `insertBack(e)`: place the element e at the back of the dequeue
3. `eraseFront()`: remove the item at the front of the dequeue
4. `eraseBack()`: remove the item at the back of the dequeue
5. `front()`: return the element at the front of the dequeue
6. `back()`: return the element at the back of the dequeue

Dequeues may also provide `size()` and `empty()`.

How to pronounce “dequeue”?

Your book says this is pronounced like “deck” to avoid confusion with the `dequeue()` operator for queues. I’ve never liked that. How about...

How to pronounce "dequeue"?

Your book says this is pronounced like "deck" to avoid confusion with the `dequeue()` operator for queues. I've never liked that. How about...

- Since it's two-ended, we could just as well call it a bi-queue and pronounce it "bike"

How to pronounce "dequeue"?

Your book says this is pronounced like "deck" to avoid confusion with the `dequeue()` operator for queues. I've never liked that. How about...

- Since it's two-ended, we could just as well call it a bi-queue and pronounce it "bike"
- Since it's a combination of a queue and a stack, how about "quest?"

I don't think either will catch on, so it's safer to stick with "deck."

Reading Assignment

Read the following sections in the textbook

1. Section 5.1: Stacks; including the applications in 5.1.6 and 5.1.7
2. Section 5.2: Queues
3. Section 5.3: Double-Ended Queues; omit Adapters (5.3.4)