CSE 1320 - Intermediate Programming

Stacks and Queues

Alex Dillhoff

University of Texas at Arlington

Stacks

A **stack** is an abstract data type that can be implemented with both arrays and linked lists.

It is restricted in how the data is accessed.

Stack Operations

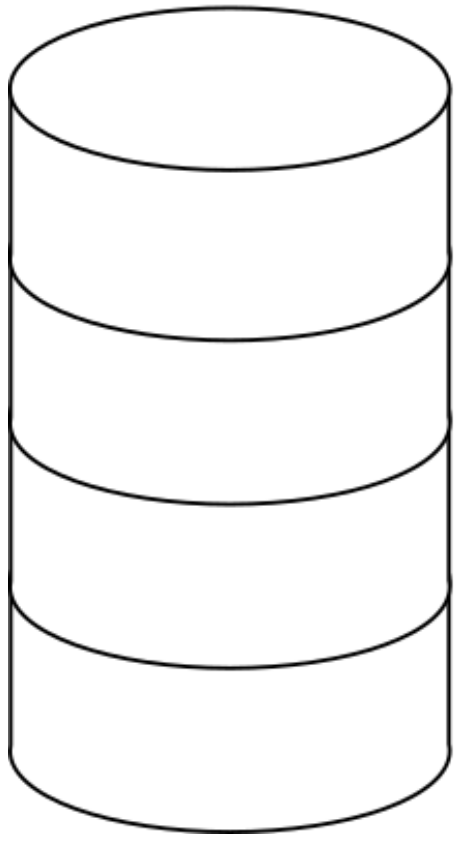

Figure: A general representation of a stack.

Stack Operations

Stacks are manipulated using two core functions: **push** and **pop**.

Adding an item is achieved by **pushing** an element onto the stack.

An item is removed by **popping** it off the stack.

Stack Operations

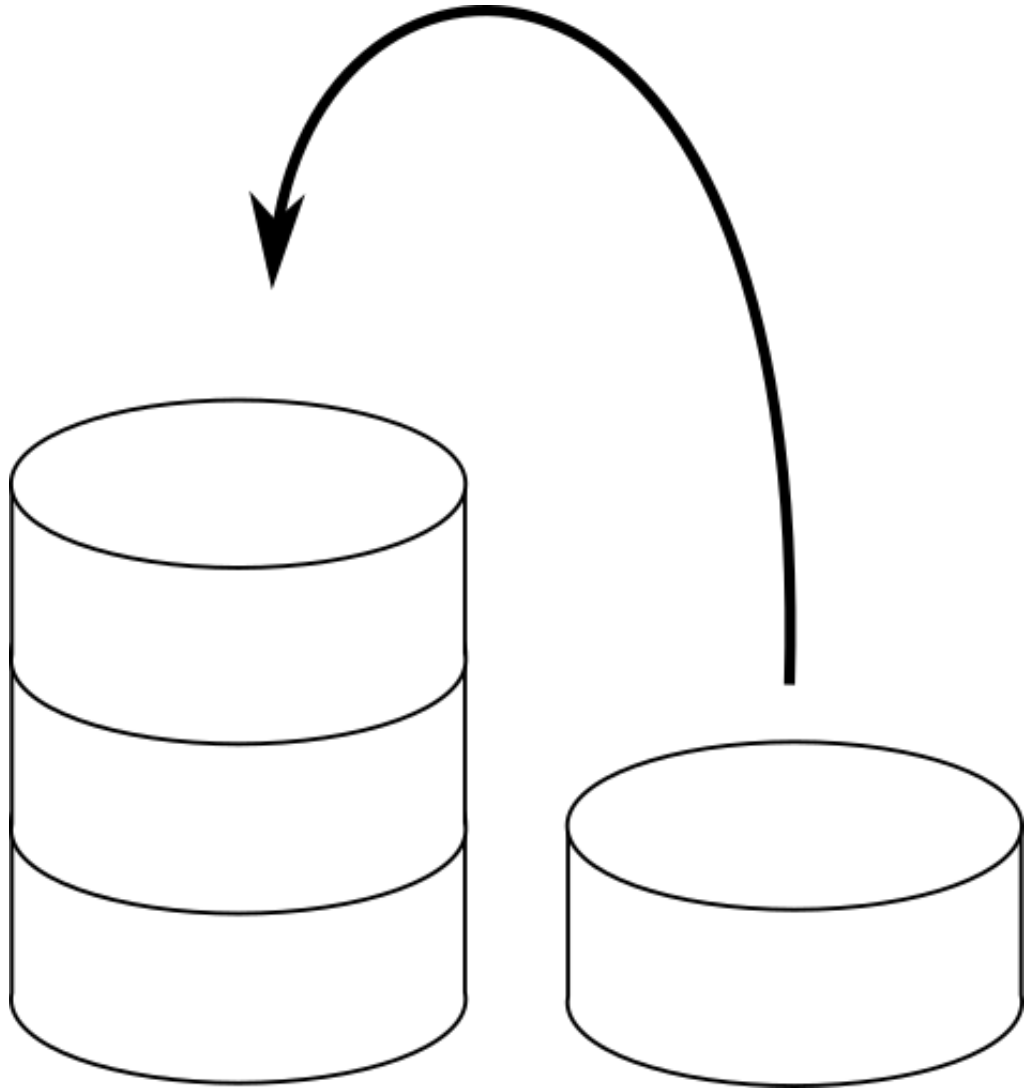

Figure: Pushing a new item onto the stack.

Stack Operations

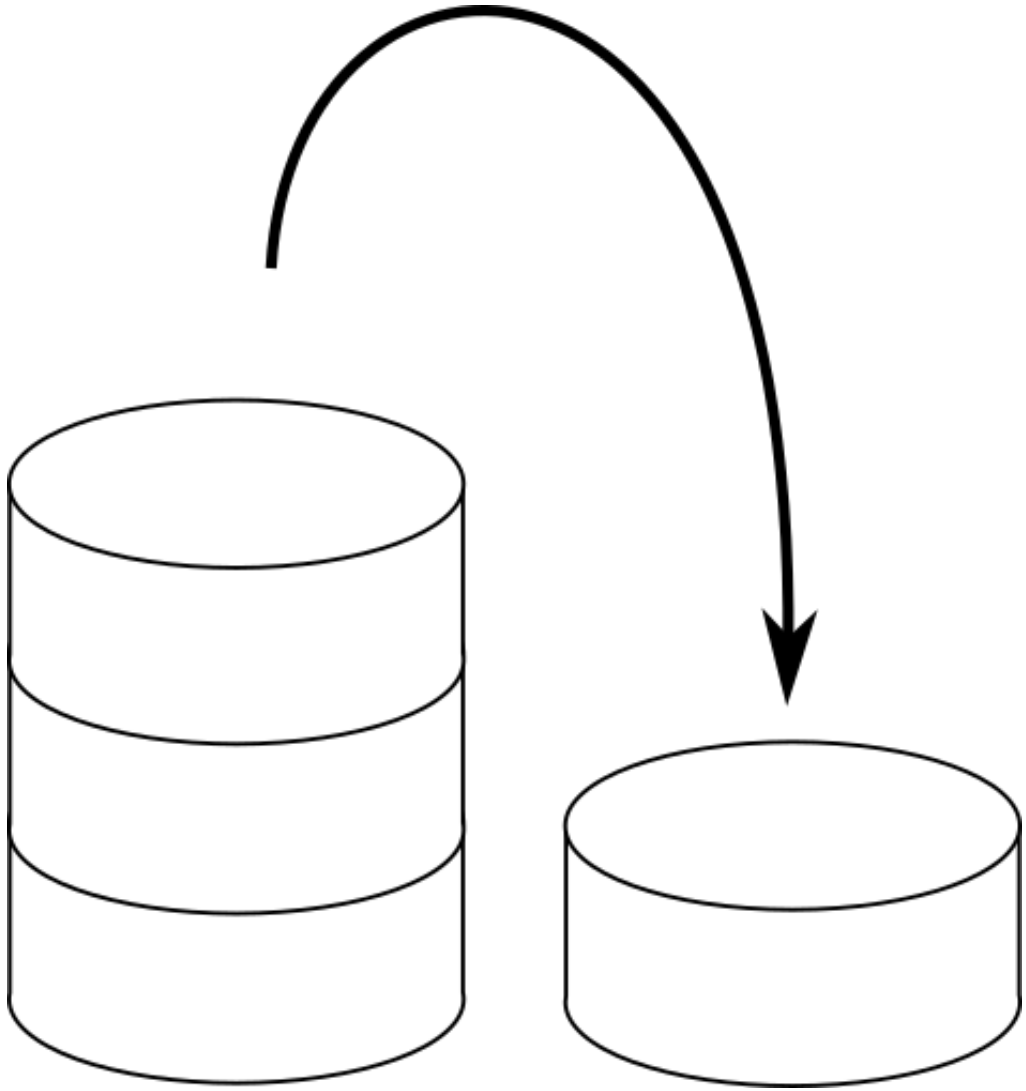

Figure: Popping an existing item from the stack.

LIFO and limitations

This restriction of pushing in popping is referred to as Last In First Out (LIFO).

This limitation implies that we can only ever access the last element pushed onto the stack.

Static Array Implementation

A stack can be implemented in both a static or dynamic array.

For a static array, the size limits the number of items that can be pushed onto the stack.

An additional integer is also necessary to indicate the position of the last element.

Static Array Implementation

Given an array A and an integer to the last element i.

To **push** an item onto the stack, the item is simply inserted at the index referenced by i. The index is increased by 1.

To **pop** an item, the index is simply decreased by 1. It does not matter that the value still remains since the index determines which item to access.

Static Array Implementation

Example: Stack with a static array

Dynamic Array Implementation

Implementing a stack using dynamic arrays requires more implementation for memory allocation.

Dynamic Array Implementation

Given an array A and an integer to the last element i.

To **push** an item onto the stack, we first allocate memory for the new item before inserting.

When **popping** an item, the array is reallocated to a smaller size.

Dynamic Array Implementation

Example: Stack with a dynamic array

Linked List Implementation

Stacks can also be implemented using a linked list.

It is most efficient when keeping a reference to the last element.

Linked List Implementation

Given a linked list L.

To **push** an item onto the stack, memory is allocated for a new node and added to the end of the list.

To **pop** an item off the stack, the last element is removed and the previous link in the list is set as the tail.

Linked List Implementation

Example: Stack with a linked list

Queues

Queues share some similarities to stacks in that only a single element can be accessed at a time.

They can also be implemented using arrays and linked lists.

Queues

A **queue** is defined as operating using a First In First Out (FIFO) paradigm.

That is, new items are inserted at the back of the queue and items can only be retrieved from the front of the queue.

Enqueue and Dequeue

These two operations, **enqueue** and **dequeue**, add and remove elements, respectively.

Enqueue

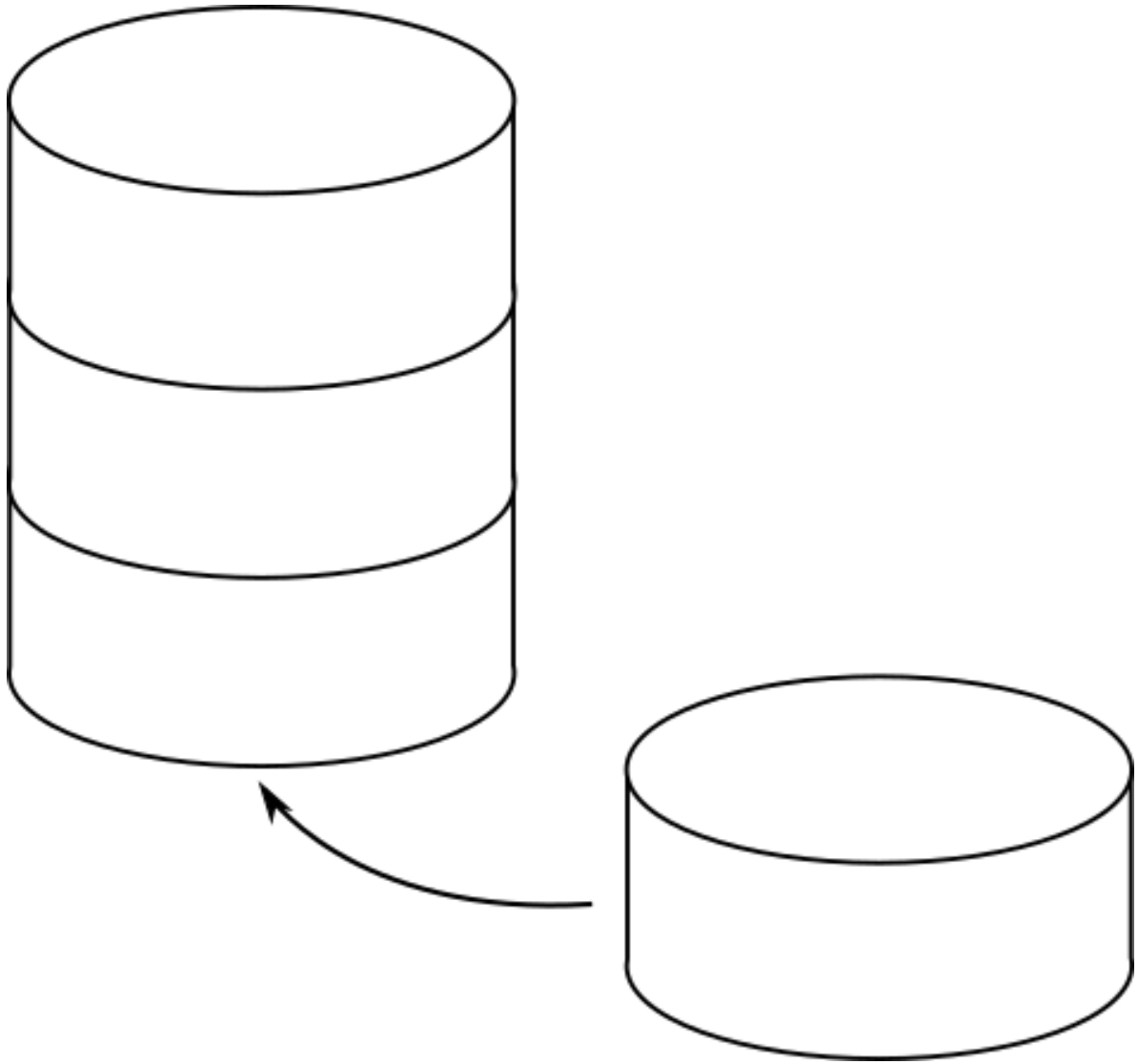

Figure: Placing a new item into the queue.

Dequeue

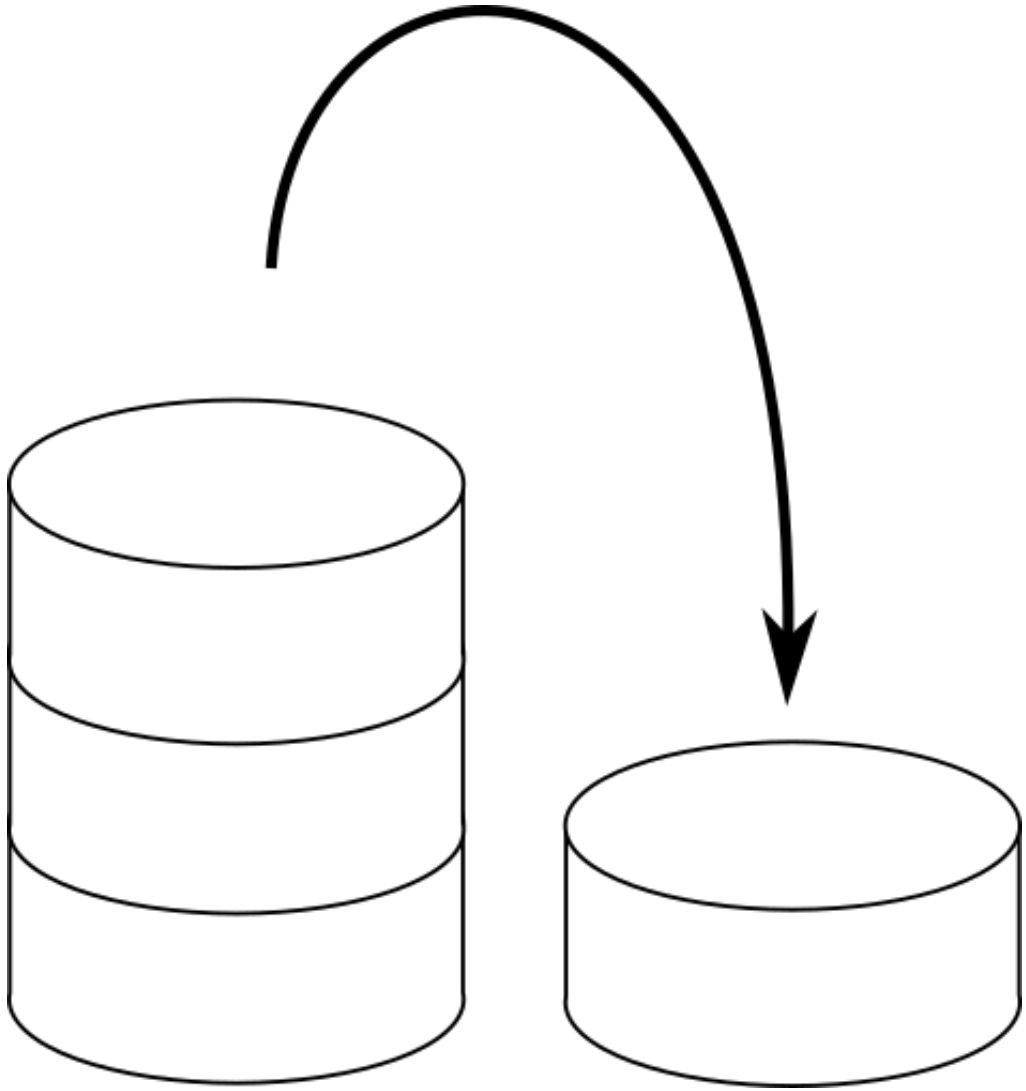

Figure: Removing an item from the queue.

Static Array Implementation

Implementing these operations using arrays requires that all items in the queue be shifted as each item is added and removed.

This is not very efficient when compared to a linked list.

Static Array Implementation

Given an array A and size index i.

To **enqueue** an item, the new element is simply added to the location referenced by i. This index is then increased to reflect the new size.

To **dequeue** an item, the element at the head of the list is first removed. All elements to the right of the first element must be shifted to the left. Finally, the index i is decreased by 1.

Static Array Implementation

Example: Queue with a static array

Dynamic Array Implementation

Besides the requirement of managing the allocated memory, there is not much difference when implementing a queue using dynamic arrays.

Dynamic Array Implementation

Given an array A and size index i.

To **enqueue** an item, space is first allocated in the array. The new element is then added to the location referenced by i. This index is then increased to reflect the new size.

To **dequeue** an item, the element at the head of the list is first removed. All elements to the right of the first element must be shifted to the left. Finally, the index i is decreased by 1.

Dynamic Array Implementation

Example: Queue with a dynamic array

Linked List Implementation

A linked list is arguable the most efficient structure when considering queues, dependent on how it is implemented.

If an external reference to the head and tail is kept, **enqueue** and **dequeue** require minimal operations.

Static Array Implementation

Given a linked list L.

To **enqueue** an item, a node is added to the tail of the list.

To **dequeue** an item, the element at the head of the list is first removed. The second item in the list is now the head of the list. Only the pointers need to be updated.

Linked List Implementation

Example: Queue with a linked list