Stacks and Queues

CS 240 Spring 2019

ADTs describe behavior

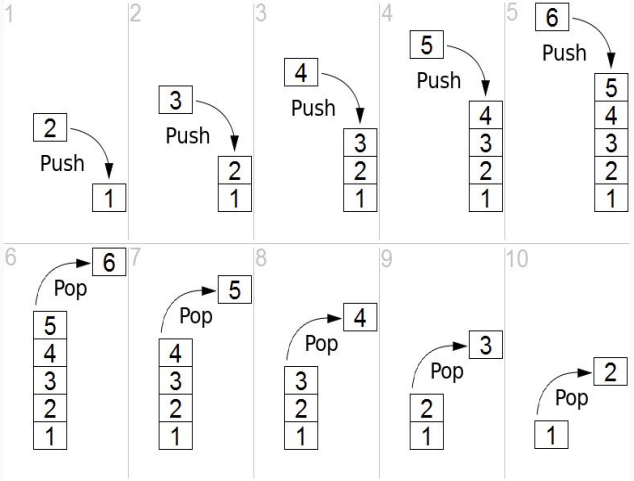
- Recall that ADTs describe a Data Structure's behavior, not its implementation
- This means that a specific ADT should have a conventional interface, but implementation can vary greatly

The Stack

- What if we need a Collection Data Structure...
- but we want to restrict how the data is accessed...
- and it must also be a Sequential data structure

LIFO

- Last in First Out
- Stacks limit the user to removing elements in reverse order of their insertion
- Stack of books in a box



Push / Pop

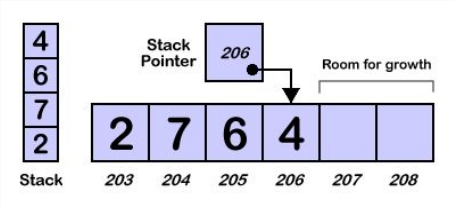
- Push
  - Inserting onto a stack is called a ***push***
- Pop
  - Removing from the stack is called ***pop***
- To access an element, you must take every element before it off the stack

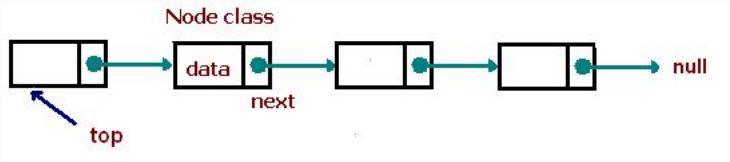
Stack ADT Interface

```
class Stack { // Stack class ADT
        void clear(); // Reinitialize the stack.
        boolean push(Object it);  // Push "it" onto the top of the stack
        Object pop();  // Remove and return the element at the top of the stack
        Object peek();  // Return a copy of the top element
};
```

Two Implementations

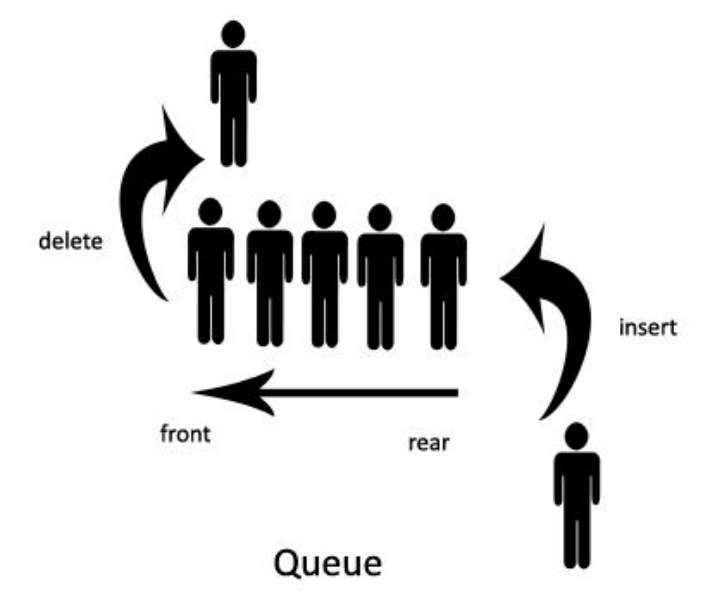
- Array Based
  - top - an index pointing to the ‘top’ of the list
  - the initialization size
  - pros
    - *easier memory management*
  - cons
    - *requires a static size*
- Linked Based
  - Use linked list as the underlying data structure
  - pros
    - *internal memory is dynamic*
  - cons
    - *more complex memory management*





Queue

- What if we need something similar to a stack but we want our access in insertion order
  - First in First Out



Dequeue / Enqueue

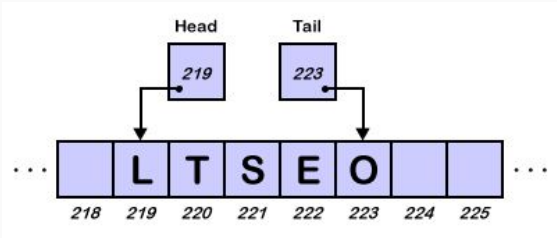
- Enqueue
  - Inserting onto a queue is called an **enqueue**
- Dequeue
  - Removing from the stack is called **dequeue**
- To access an element, you must take every element before it off

Queue Public Interface

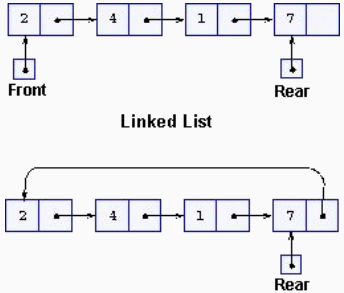
```
class Queue { // Queue class ADT
        void clear(); // Reinitialize the Queue.
        boolean enqueue(Object it);  // Push "it" into the Queue
        Object dequeue();  // Remove and return the element at the end of the Queue
        Object peek();  // Return a copy of the oldest element
}
```

Two Implementations

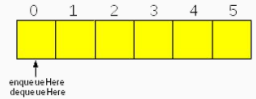
- Array Based
  - top and bottom - an index pointing to the ends of the list
  - need initial size
  - pros
    - *easier memory management*
  - cons
    - *requires a static size*



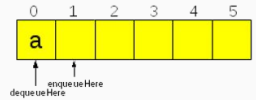
- Linked Based
  - Can use your existing linked list as the underlying data structure
  - pros
    - *internal memory is dynamic*
  - cons
    - *more complex memory management*



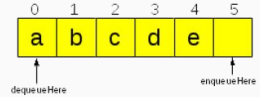
Array Based Queue Implementation



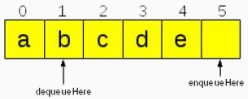
enqueue("a")



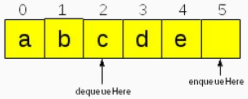
enqueue("e")



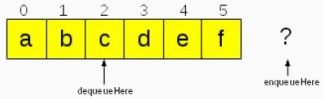
dequeue() returns "a"



dequeue() returns "b"



enqueue("f")



Classwork: Stacks and Queues