Week 3 – Friday

COMP 2100

Last time

- What did we talk about last time?
- List implementation with a dynamic array
- Running time of array-backed list
- Brief introduction to stacks

Questions?

Assignment 2

Project 1

Bitmap Manipulator

Stacks

Stack

- A stack is a simple (but useful) ADT that has three basic operations:
  - **Push** Put an item on the top of the stack
  - **Pop** Remove an item from the top of the stack
  - **Top** Return the item currently on the top of the stack (sometimes called **peek**)

Keeping track of things

- When are stacks used?
  - Implicitly, in recursion (or in any function calls)
  - Explicitly, when turning recursive solutions into iterative solutions
  - When parsing programming languages
  - When converting infix to postfix

Implementations

Array implementations

- Advantages:
  - Pop is $\Theta(1)$
  - Top is $\Theta(1)$
- Disadvantages
  - Push is $\Theta(n)$ in the very worst case, but not in the amortized case

Array implementation

```
public class ArrayStack<E> {
  private E[] data;
  private int size;

  public ArrayStack() {}
  public void push(E value) {}
  public E pop() {}
  public E peek() {} // Instead of top
  public int size() {}
}
```

Array Constructor

Array Push

Array Pop

Array Peek

Array Size

Queues

Queue

- A **queue** is a simple data structure that has three basic operations (very similar to a stack)
  - **Enqueue** Put an item at the back of the queue
  - **Dequeue** Remove an item from the front of the queue
  - **Front** Return the item at the front of the queue
- A queue is considered FIFO (First In First Out) or LILO (Last In Last Out)

Application of queues

- Queues are useful whenever you want to keep track of the order of arrival
  - A line in a fast food restaurant
  - A job in a printer queue
  - A buffer for managing data

Circular array implementation

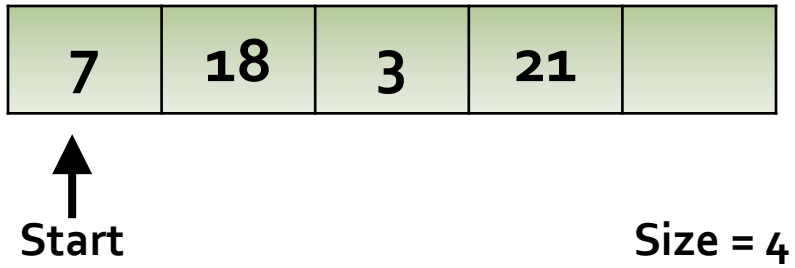
- A queue is a little bit harder to implement than a stack with an array
- The trouble is that you're enqueuing and dequeuing from different ends
- Removing something from the front seems to imply that you'll need to shift over all the contents of the array
- Enter the circular array!

Circular array

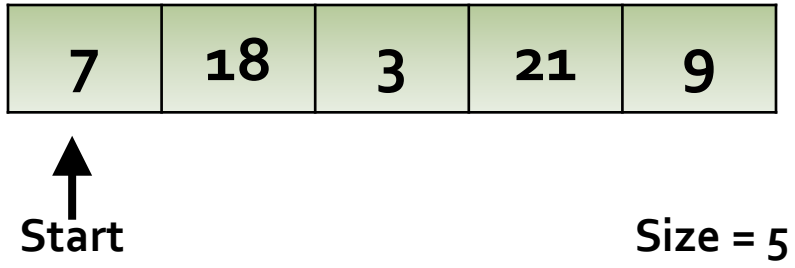
- A **circular array** is just a regular array
- However, we keep a **start** index as well as a **size** that lets us start the array at an arbitrary point
- Then, the contents of the array can go past the end of the array and wrap around
- The modulus operator (%) is a great way to implement the wrap around

Circular array example

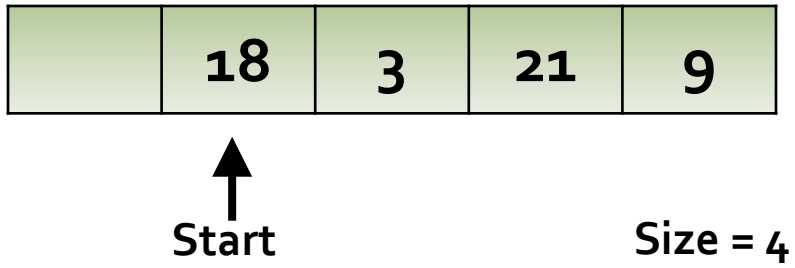
1. Starting array



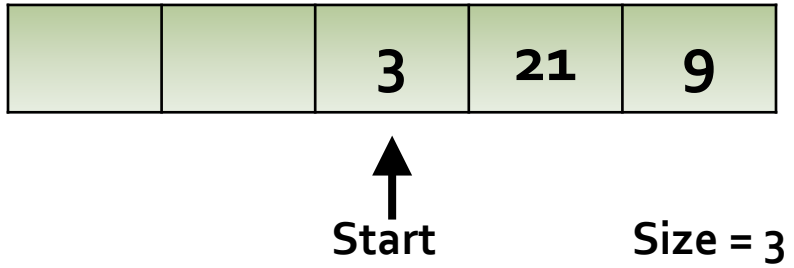
2. Enqueue 9



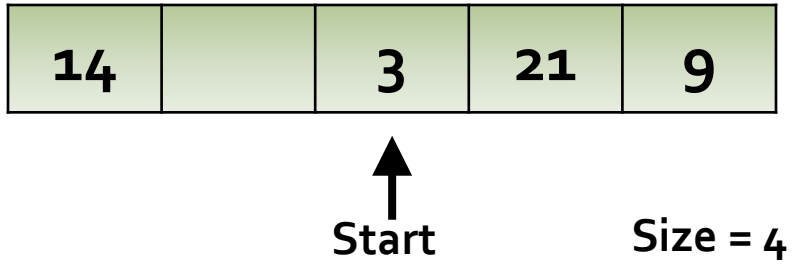
3. Dequeue



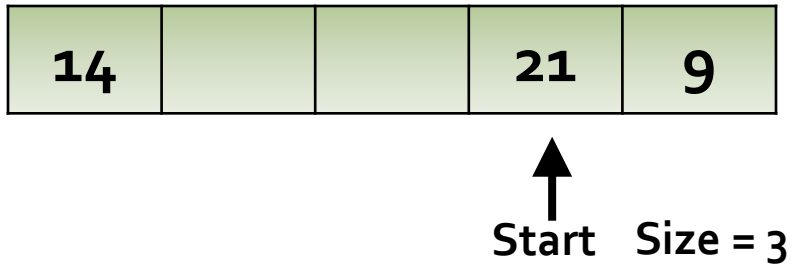
4. Dequeue



5. Enqueue 14



6. Dequeue



Circular array implementation

- Advantages:
  - Dequeue is $\Theta(1)$
  - Front is $\Theta(1)$
- Disadvantages
  - Enqueue is $\Theta(n)$ in the very worst case, but not in the amortized case

Circular array implementation

```
public class ArrayQueue {
  private E[] data = (E[]) new Object[10];
  private int start = 0;
  private int size = 0;

  public void enqueue(E value) {}
  public E dequeue() {}
  public E front() {}
   public int size() {}
}
```

Circular Array Front

Circular Array Get Size

Circular Array Enqueue

Circular Array Dequeue

Upcoming

Next time…

- Array implementation of queues
- Introduction to linked lists

Reminders

- Keep reading section 1.3
- Finish Assignment 2
  - **Due tonight by midnight!**
- Keep working on Project 1
  - **Due next Friday, September 20 by midnight**
- Due to a meeting, I will not be available for most of my office hours from 1:45-2:45 today
  - **However! Professor Stucki and Dr. Çal also have lab hours at those times and can help you**