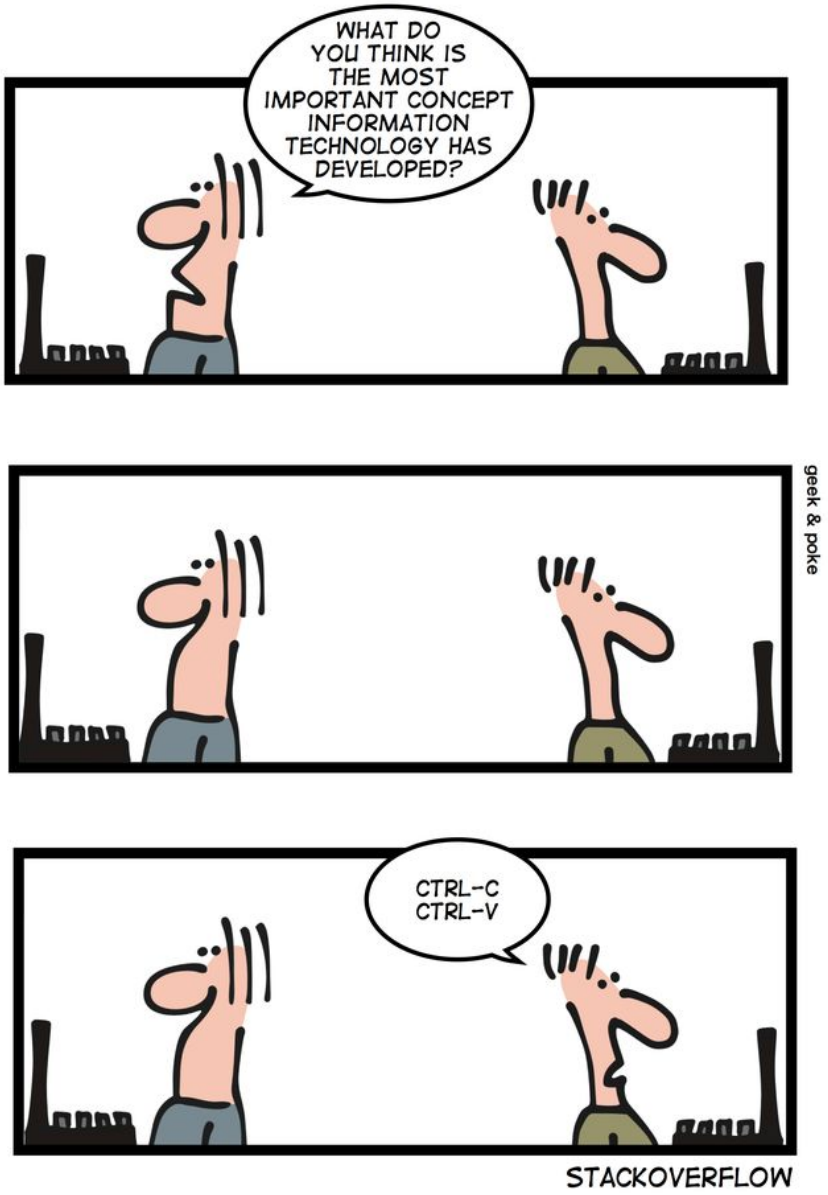
CSCI 104

Rafael Ferreira da Silva

rafsilva@isi.edu

Slides adapted from: Mark Redekopp and David Kempe



http://geek-and-poke.com/geekandpoke/2013/1/22/stackoverflow.html

Specialized Lists

STACKS AND QUEUES

Stacks & Queues

- Lists are good for storing generic sequences of items, but they can be specialized to form other useful structures
- What if we had a List, but we restricted how insertion and removal were done?
  - **Stack** – Only ever insert/remove from one end of the list
  - **Queue** – Only ever insert at one end and remove from the other

First-In, First-Out (FIFOs)

QUEUE ADT

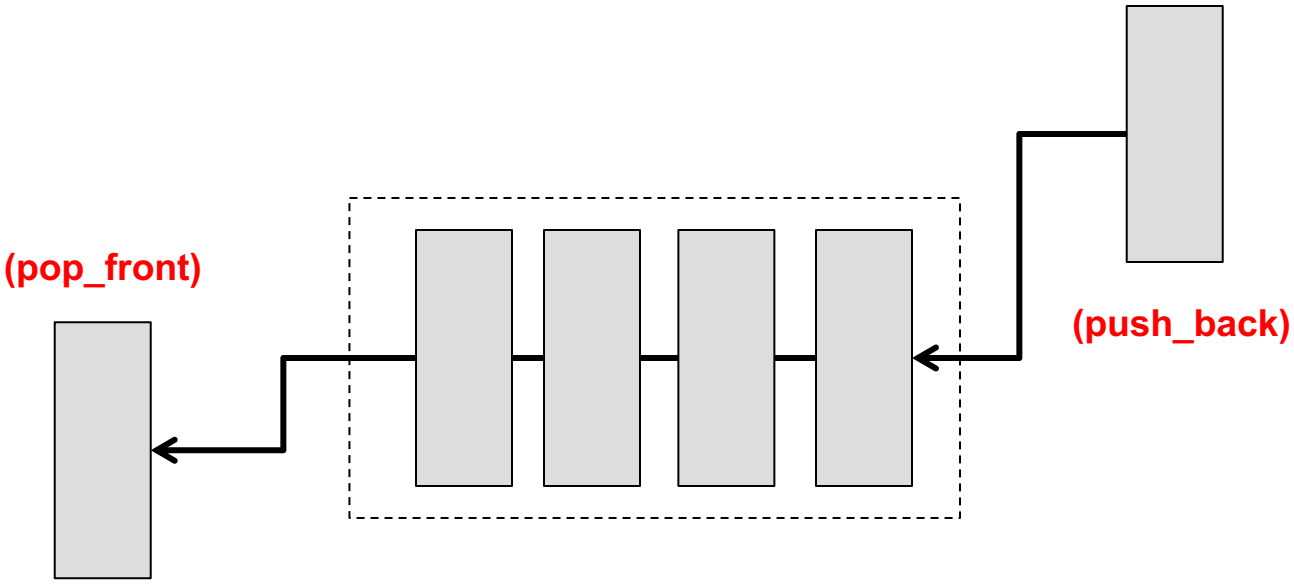
Queue ADT

- Queue – A list of items where insertion only occurs at the back of the list and removal only occurs at the front of the list
  - Like waiting in line for a cashier at a store
- Queues are FIFO (First In, First Out)
  - Items at the back of the queue are the newest
  - Items at the front of the queue are the oldest
  - Elements are processed in the order they arrive

A Queue Visual

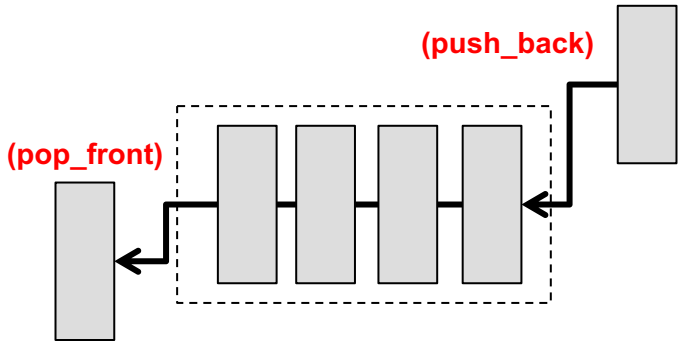
Items leave from the front
(pop_front)

Items enter at the back
(push_back)



Queue Operations

- What member functions does a Queue have?
  - push_back(item) – Add an item to the back of the Queue
  - pop_front() - Remove the front item from the Queue
  - front() - Get a reference to the front item of the Queue (don't remove it though!)
  - size() - Number of items in the Queue
  - empty() - Check if the Queue is empty



A Queue Class

- A sample class interface for a Queue
- Queue Error Conditions
  - **Queue Underflow** – The name for the condition where you call pop on an empty Queue
  - **Queue Overflow** – The name for the condition where you call push on a full Queue (a Queue that can't grow any more)
    - This is only possible for Queues that are backed by a bounded list

```
#ifndef QUEUEINT_H
#define QUEUEINT_H

class QueueInt  {
 public:
  QueueInt();
  ~QueueInt();
  int size() const;
  void push_back(const int& value); //enqueue
  void pop_front();  // dequeue
  int const & front() const;
  bool empty() const;
 private:
  // ???
};
#endif
```

Other Queue Details

- How should you implement a Queue?
  - Back it with an ArrayList
  - Back it with a linked list
  - Back it with a vector
  - Inherit from a linked list
  - Which is best?

	Push_back	**Pop_front**	**Front()**
ArrayList	O(1)	O(n)	O(1)
LinkedList (Singly-linked w/ tail ptr)	O(1)	O(1)	O(1)
LinkedList (Singly-linked w/o tail ptr)	O(n)	O(1)	O(1)

Queue Applications

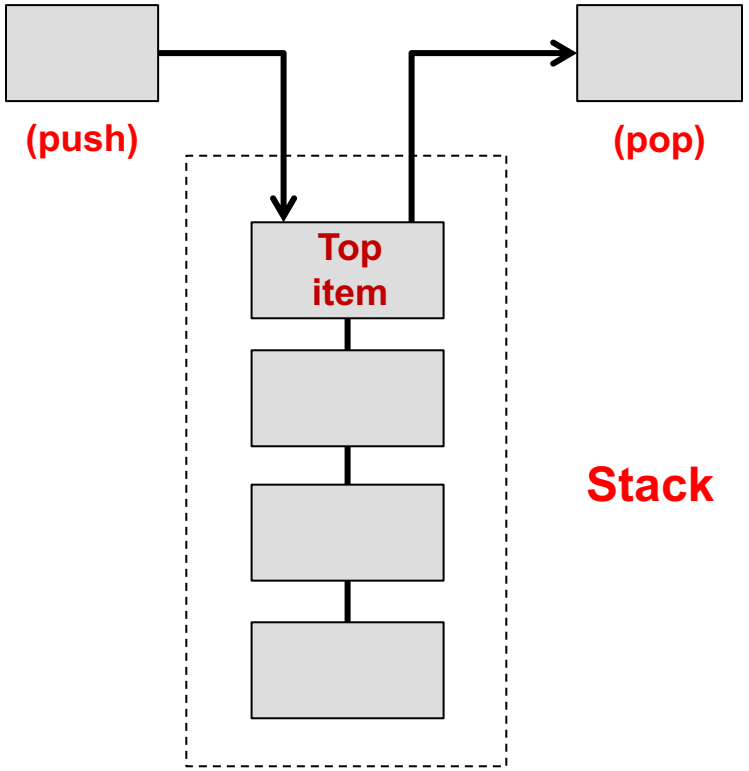
- Print Jobs
  - Click “Print” on the computer is much faster than actually printing (build a backlog)
  - Each job is processed in the order it's received (FIFO)
  - Why would you want a print queue rather than a print stack
- Seating customers at a restaurant
- Anything that involves "waiting in line"
- Helpful to decouple producers and consumers

Last-In, First-Out (LIFOs)

STACK ADT

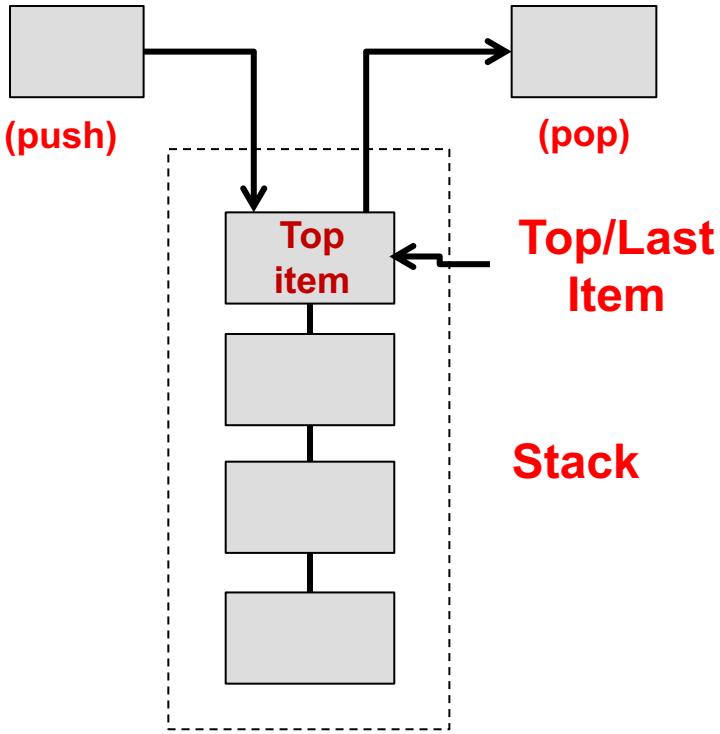
Stack ADT

- Stack: A list of items where insertion and removal only occurs at one end of the list
- Examples:
  - A stack of boxes where you have to move the top one to get to ones farther down
  - A spring-loaded plate dispenser at a buffet
  - A PEZ dispenser
  - Your e-mail inbox
- Stacks are LIFO
  - Newest item at top
  - Oldest item at bottom



Stack Operations

- What member functions does a Stack have?
  - push(item) – Add an item to the top of the Stack
  - pop() - Remove the top item from the Stack
  - top() - Get a reference to the top item on the Stack (don't remove it though!)
  - size() - Get the number of items in the Stack
- What member data does a Stack have?
  - A list of items
  - Top/Last Item Pointer/Index



A Stack Class

- A sample class interface for a Stack
- How should you implement a Stack?
  - Back it with an array
  - Back it with a vector
  - Back it with a linked list
  - Inherit from linked list
  - Which is best?
- Stack Error Conditions
  - Stack Underflow – The name for the condition where you call pop on an empty Stack
  - Stack Overflow – The name for the condition where you call push on a full Stack (a stack that can't grow any more)

```
#ifndef STACKINT_H
#define STACKINT_H

class StackInt  {
 public:
  StackInt();
  ~StackInt();
  int size() const;
  bool empty() const;
  void push(const int& value);
  void pop();
  int const & top() const;
};
#endif
```

Array Based Stack

- A sample class interface for a Stack
- If using an array list, which end should you use as the "top"?
  - Front or back?
- If using a linked list, which end should you use?
  - If you just use a head pointer only?
  - If you have a head and tail pointer?

```
#ifndef STACKINT_H
#define STACKINT_H

class StackInt  {
 public:
  StackInt();
  ~StackInt();
  int size() const;
  bool empty() const;
  void push(const int& value);
  void pop();
  int const& top() const;
private:
  AListInt mylist_;
  // or LListInt mylist_;
};
#endif
```

Stack Examples

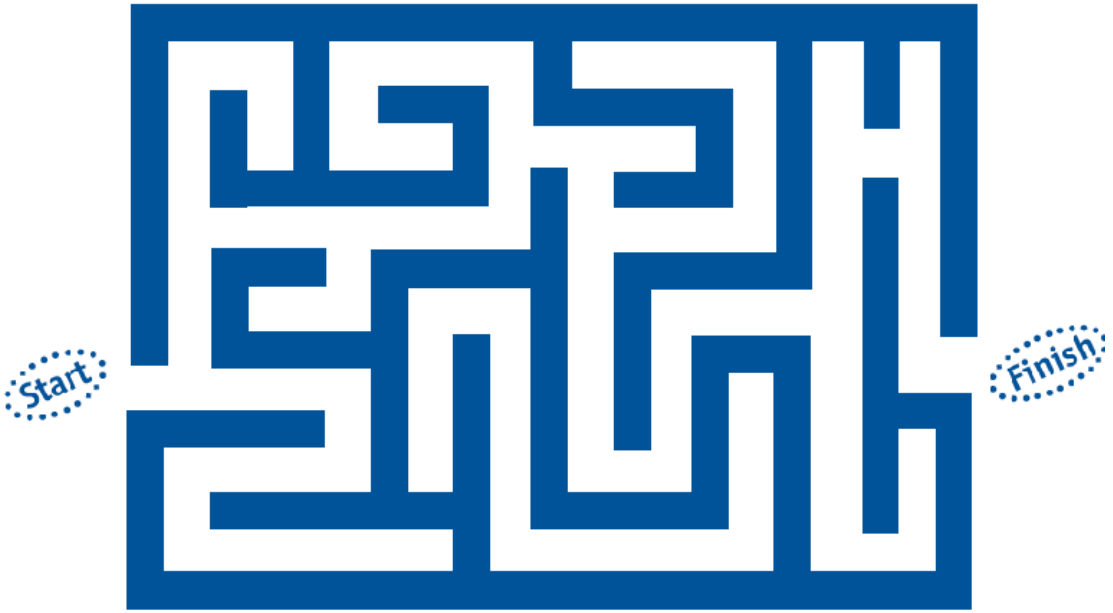
- Reverse a string

```
#include <iostream>
#include <string>
#include "stack.h"
using namespace std;
int main()
{
  StackChar s;

  string word;
  cout << "Enter a word: ";
  getline(cin,word);

  for(int i=0; i < word.size(); i++)
    s.push(word.at(i));

  while(!s.empty()){
    cout << s.top();
    s.pop();
  }
}
```

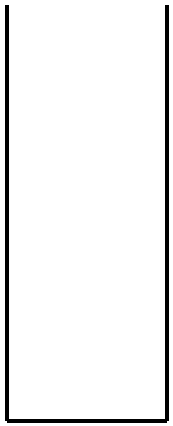
Type in: "hello"
Output: "olleh"

Another Stack Example

- Depth First Search (See Graph Traversals later in this semester)
- Use a stack whenever you encounter a decision, just pick and push decision onto stack. If you hit a dead end pop off last decision (retrace steps) and keep trying, etc.
  - Strait or Left
    - Choose straight…dead end
    - Pop straight and make next choice…left
      - Straight or Right…etc.



http://www.pbs.org/wgbh/nova/einstein/images/lrk-maze.gif

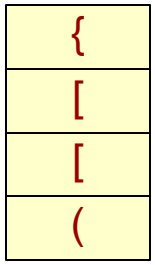


Stack Usage Example

- Check whether an expression is properly parenthesized with '(', '[', '{', '}', ']', ')'
  - Correct: (7 * [8 + [9/{5-2}]])
  - Incorrect: (7*8
  - Incorrect: (7*8]
- Note: The last parentheses started should be the first one completed
- Approach
  - Scan character by character of the expression string
  - Each time you hit an open-paren: '(', '[', '{' **push** it on the stack
  - When you encounter a ')', ']', '}' the **top** character on the stack should be the matching opening paren type, otherwise ERROR!

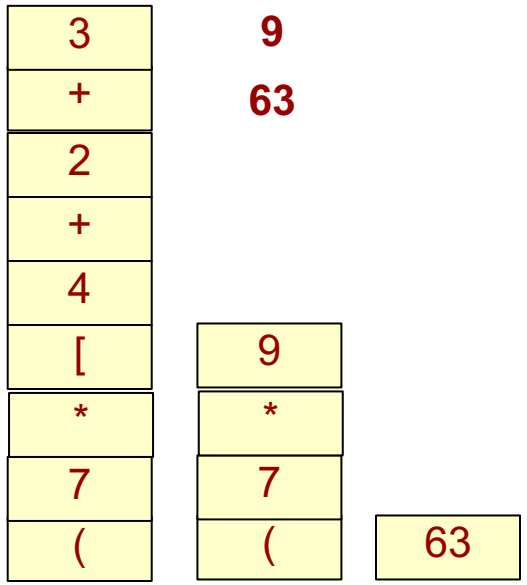
(7 * [8 + [9/{5-2}]])

([[{ }]])



(7 * [4 + 2 + 3])

3
5
9

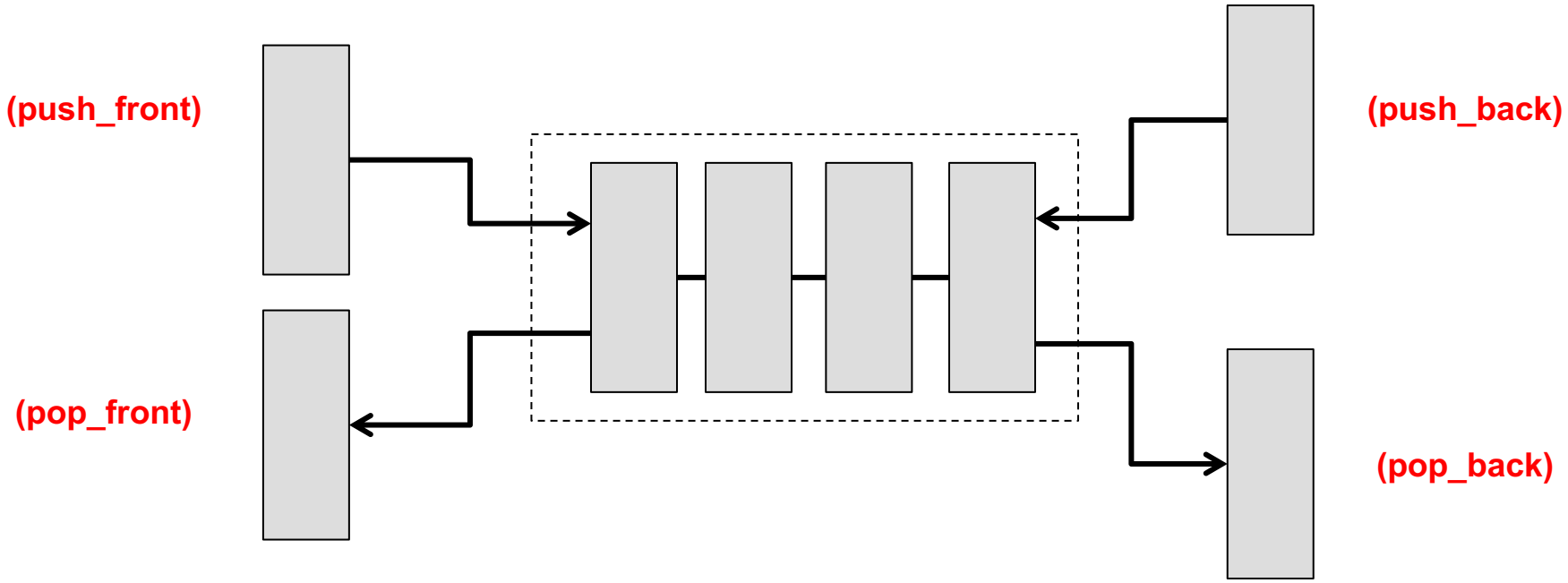


Double-ended Queues

DEQUE ADT

The Deque ADT

- **D**ouble-**e**nded **que**ues
- Equally good at pushing and popping on either end



STL Deque Class

- Similar to vector but allows for push_front() and pop_front() options
- Useful when we want to put things in one end of the list and take them out of the other

1

my_deq

0	1	2	3	4
50	51	52	53	54

2

my_deq

0	1	2	3	4
51	52	53	54	60

after 1st iteration

3

my_deq

0	1	2	3	4
60	61	62	63	64

after all iterations

4

my_deq

```
#include <iostream>
#include <deque>

using namespace std;

int main()
{
  deque<int> my_deq;
  for(int i=0; i < 5; i++){
    my_deq.push_back(i+50);
  }
  cout << "At index 2 is: " << my_deq[2] ;
  cout << endl;

  for(int i=0; i < 5; i++){
    int x = my_deq.front();
    my_deq.push_back(x+10);
    my_deq.pop_front();
  }
  while( ! my_deq.empty()){
    cout << my_deq.front() << " ";
    my_deq.pop_front();
  }
  cout << endl;

}
```

1

2

3

4

STL Vector vs. Deque

- std::vector is essentially a Dynamic Array List
  - Slow at removing and inserting at the front or middle
  - Fast at adding/remove from the back
  - Implies it could be used well as a (stack / queue)
- std::deque gives fast insertion and removal from front and back along with fast random access (i.e. at(i))
  - Almost has "look and feel" of linked list with head and tail pointers providing fast addition/removal from either end
  - Implies it could be used well as a (stack / queue)
  - Practically it is likely implemented as a circular array buffer

Circular Buffers

- Take an array but imagine it wrapping into a circle to implement a deque
- Setup a head and tail pointer
  - Head points at first occupied item, tail at first free location
  - Push_front() and pop_front() update the head pointer
  - Push_back() and pop_back() update the tail pointer
- To overcome discontinuity from index 0 to MAX-1, use modulo operation
  - Index = 7; Index++ should cause index = 0
  - index = (index + 1)%MAX
  - Index = 0; Index-- should cause index = 7
  - if(--index < 0) index = MAX-1;
- Get item at index i
  - It's relative to the head pointer
  - Return item at (head + i)%MAX