UCLA Samueli Computer Science

CS32: Introduction to Computer Science II

Discussion Week 4

Yichao (Joey)

April 24, 2020

Credit: Carey Nachenberg, Junheng Hao

Announcements

UCLA Samueli Computer Science

- Homework 2 is due on 11:00 PM Thursday, April 30

Outline Today

UCLA Samueli Computer Science

- Stack
- Queue
- Homework 2: Guide

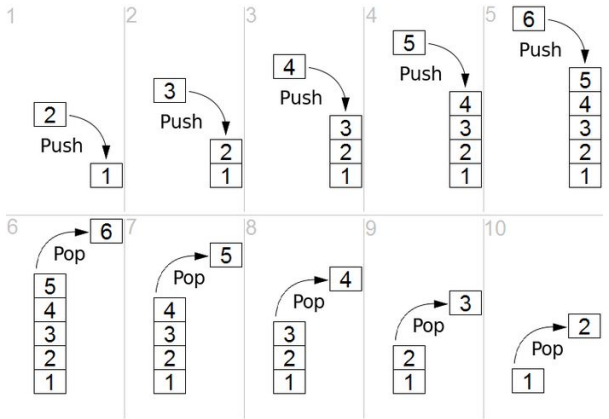
Stack: FILO

Review: Basics

UCLA Samueli Computer Science

- FILO: First In, Last Out
- A standard stack implementation
  - `push()` and `pop()`
  - Other methods: `top()`, `count()`
- Applications:
  - Stack memory: function call
  - Check expressions: matching brackets
  - Depth-first graph search
- Question: How do you implement stack with linked list?

```
class Stack
{
public:
      bool push(const ItemType& item);
      ItemType pop();
      bool empty() const;
      int count() const;
private:
      // some features
};
```



Stack

Implement stack with linked list

UCLA Samueli Computer Science

- Container: linked list
- Functions:
  - `push()`: Insert node before head.
  - `pop()`: Remove head and return the head value.
  - `top()`: Read head node.
  - `count()`: Maintain a private `int` member.

Stack

Examples

UCLA Samueli Computer Science

- Given a math expression or text sequence:
  - 6+((5+2)*3-(7+11)*5)*6 → Consider calculation of Reverse Polish Notation (**postfix notation**)
  - Latex: f^{\text{DNN}}(X,\mathbf{W}) = \sigma \left(\mathbf{W} \cdot X + \mathbf{b} \right)

$$f^{\text{DNN}}(X, \mathbf{W}) = \sigma\left(\mathbf{W} \cdot X + \mathbf{b}\right)$$

- How to check the brackets of all types are valid in the sequence? How to calculate expression in Reverse Polish Notation (RPN)?

Regular Expression: 2 + 3 * (5 - 1)

Reverse Polish Notation (RPN): [2] [3] [5] [1] [-] [*] [+]

Stack

Application: Algebraic Expressions

UCLA Samueli Computer Science

- **Infix Notation**
  - Operators are written in between their operands → X + Y
  - Ambiguous - needs extra rules built in about operator precedence and associativity and parentheses
- **Postfix Notation**
  - Operators are written after their operands → X Y +
  - Operators are evaluated left-to-right. They act on the two nearest values on the left.
- **Prefix Notation**
- **Tasks**
  - Evaluating Postfix Expressions
  - Converting Infix to Postfix Expressions

Stack

Examples

UCLA Samueli Computer Science

Algorithm

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
…..**3.1** If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
…..**3.2** Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output
8. Pop and output from the stack until it is not empty.

Stack

Application: Converting Infix to Postfix Expressions

UCLA Samueli Computer Science

Infix expression: `a - ( b + c * d ) / e`

ch	aStack (bottom to top)	postfixExp	
a		a	
–	–	a	
(	– (	a	
b	– (	ab	
+	– (+	ab	
c	– (+	abc	
*	– (+ *	abc	
d	– (+ *	abcd	
)	– (+	abcd*	Move operators from stack to **postfixExp** until "("
	– (	abcd*+	
	–	abcd*+	
/	– /	abcd*+	Copy operators from stack to **postfixExp**
e	– /	abcd*+e	
		abcd*+e/–	

Stack

Application: Evaluating Postfix Expressions

UCLA Samueli Computer Science

Postfix Expression
2 3 4 + *

Infix Expression
2 * (3 + 4)

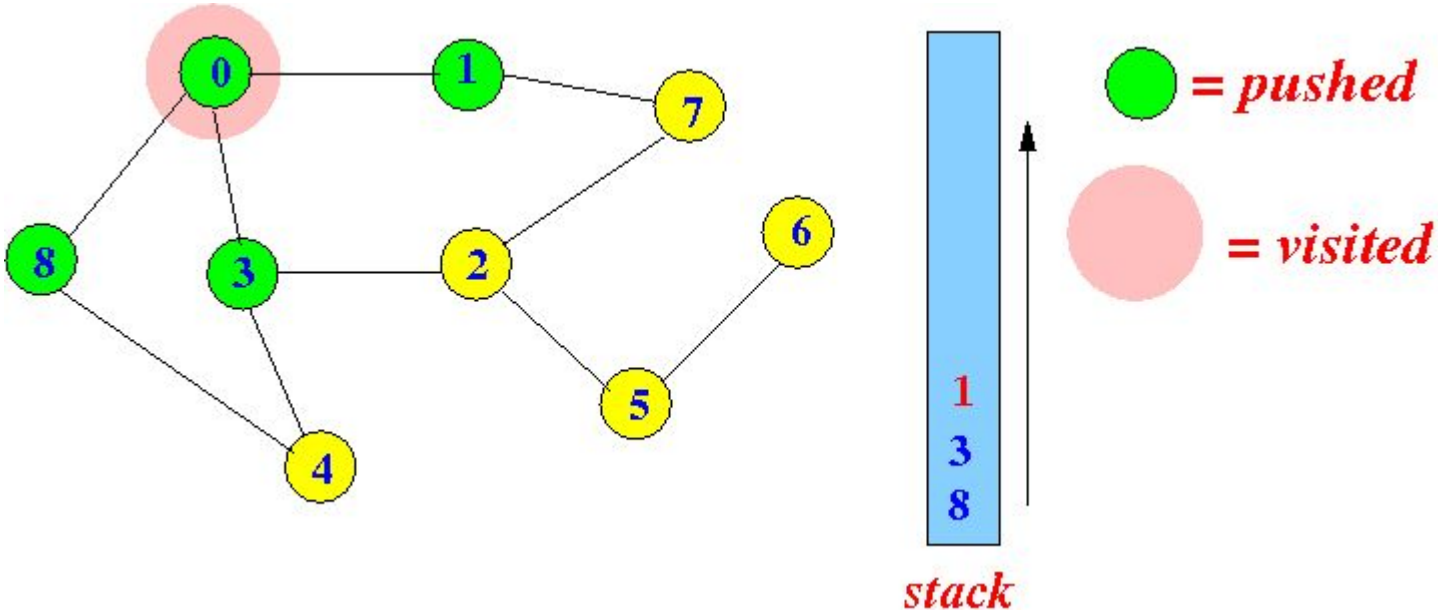
Key entered	Calculator action		Stack (bottom to top):
2	push 2		2
3	push 3		2 3
4	push 4		2 3 4
+	operand2 = peek	(4)	2 3 4
	pop		2 3
	operand1 = peek	(3)	2 3
	pop		2
	result = operand1 + operand2	(7)	
	push result		2 7
*	operand2 = peek	(7)	2 7
	pop		2
	operand1 = peek	(2)	2
	pop		
	result = operand1 * operand2	(14)	
	push result		14

Stack

Example: stack and depth-first search

UCLA Samueli Computer Science

- Depth-first Search (DFS) on graph (will be later lectures or CS180)



Stack*

Example: Use stack to implement DFS [Link]

UCLA Samueli Computer Science

```
void Graph::DFS(int s)
{
    vector<bool> visited(V, false);     // Initially mark all vertices as not visited
    stack<int> stack; // Create a stack for DFS
    stack.push(s);          // Push the current source node
    while (!stack.empty())
    {
        s = stack.top(); // Pop a vertex from stack and print it
        stack.pop();
        // Print the popped item only if it is not visited.
        if (!visited[s])  { cout << s << " "; visited[s] = true;  }
        for (auto i = adj[s].begin(); i != adj[s].end(); ++i)
            if (!visited[*i])
                stack.push(*i);
    }
}
```

Note: Get all adjacent vertices of the popped vertex s.
If the adjacent has not been visited, then push it to stack .

Stack*

Example: Remove All Adjacent Duplicates In String

UCLA Samueli Computer Science

Given a string S of lowercase letters, a duplicate removal consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on S until we no longer can.

Example 1:

Input: "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Stack*

Example: Remove All Adjacent Duplicates In String

UCLA Samueli Computer Science

```
string removeDuplicates(string a) {
	stack<char> st;
	string ans="";
	for(auto curr:a) {
		if(st.empty()) st.push(curr);
		else if(st.top() == curr) st.pop();
		else st.push(curr);
	}

	while(!st.empty()) {
		ans += st.top();
		st.pop();
	}

	reverse(ans.begin(), ans.end());

	return ans;
}
```

Stack*

Problem: Find largest rectangle all-1sub-matrix

UCLA Samueli Computer Science

Question:

Given a binary matrix, find the area of maximum size rectangle binary-sub-matrix with all 1's.

Level: (Super) Difficult

Example 1:

Input:

```
0 1 1 0
1 1 1 1
1 1 1 1
1 1 0 0
```

Output: 8

Example 2:

Input:

```
0 1 1 0
1 1 1 1
1 1 1 1
1 1 0 0
1 1 0 1
1 1 1 0
```

Output: 10

```
Step 1: Find maximum area for row[0]
Step 2:
    for each row in 1 to N - 1
        for each column in that row
            if A[row][column] == 1
              update A[row][column] with
                A[row][column] += A[row - 1][column]
    find area for that row and update maximum area so far
```

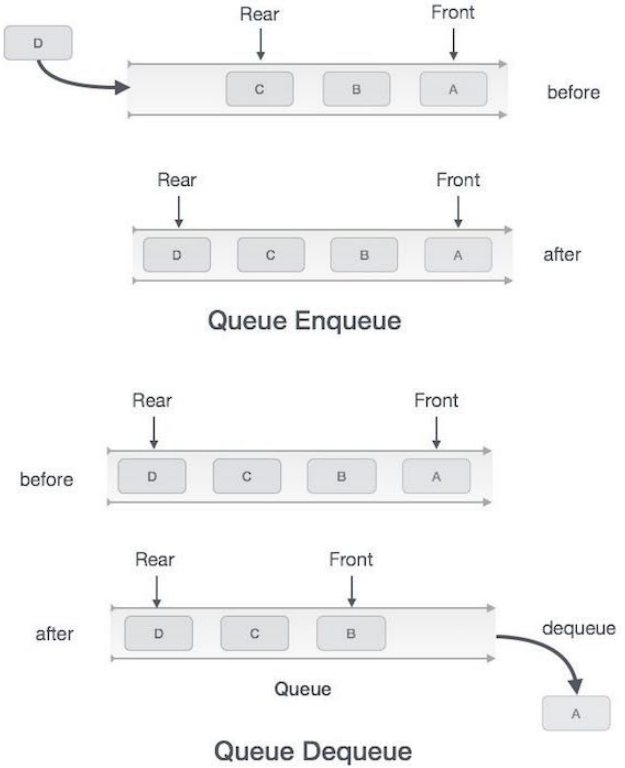
```
// Your solution here
// Think about how you can use stack to solve this problem and compare with brute-forth methods
// Tips: Think about the problem of Largest Rectangular Area in a Histogram solved by using stack
// One possible solution: [Link]
```

Queue: FIFO

Review: Basics

UCLA Samueli Computer Science

- FIFO: First In, First Out
- Basic methods:
  - `enqueue(), dequeue()`
  - `front(), back()`
  - `count()`
- Applications
  - Data streams
  - Process scheduling (DMV service request)
  - Breadth-first graph search
- How to implement queue with linked lists or dynamic arrays?



Queue

Extension: DEque (double-ended queue)

UCLA Samueli Computer Science

```
class DEque
{
     public:
          bool push_front(const ItemType& item);
          bool push_back(const ItemType& item);
          bool pop_front(const ItemType& item);
          bool pop_back(const ItemType& item);
          bool empty() const; // true if empty
          int count() const; // number of items
     private:
          int size; // Some data structure that keeps the items.
};
```

Question: How to implement `class DEque` with linked lists?

Queue

Extension: Priority queue

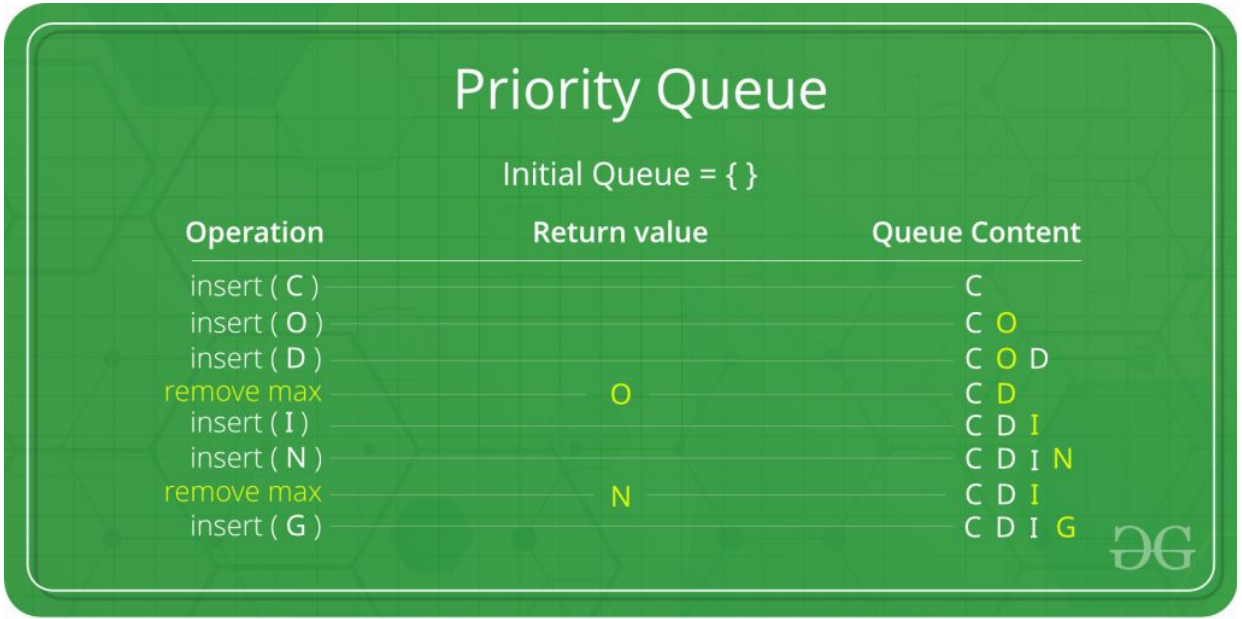
UCLA Samueli Computer Science

- Data: A finite number of objects, not necessarily distinct, having the same data type and ordered by priority
- Operations:
  - Add a new entry to the queue based on priority
  - Remove the entry with the highest priority from the queue
- We will learn priority queue (and heap) later this quarter after tree!

Queue

Extension: Priority queue

UCLA Samueli Computer Science



A typical priority queue supports following operations.

insert(item, priority): Inserts an item with given priority.

getHighestPriority(): Returns the highest priority item.

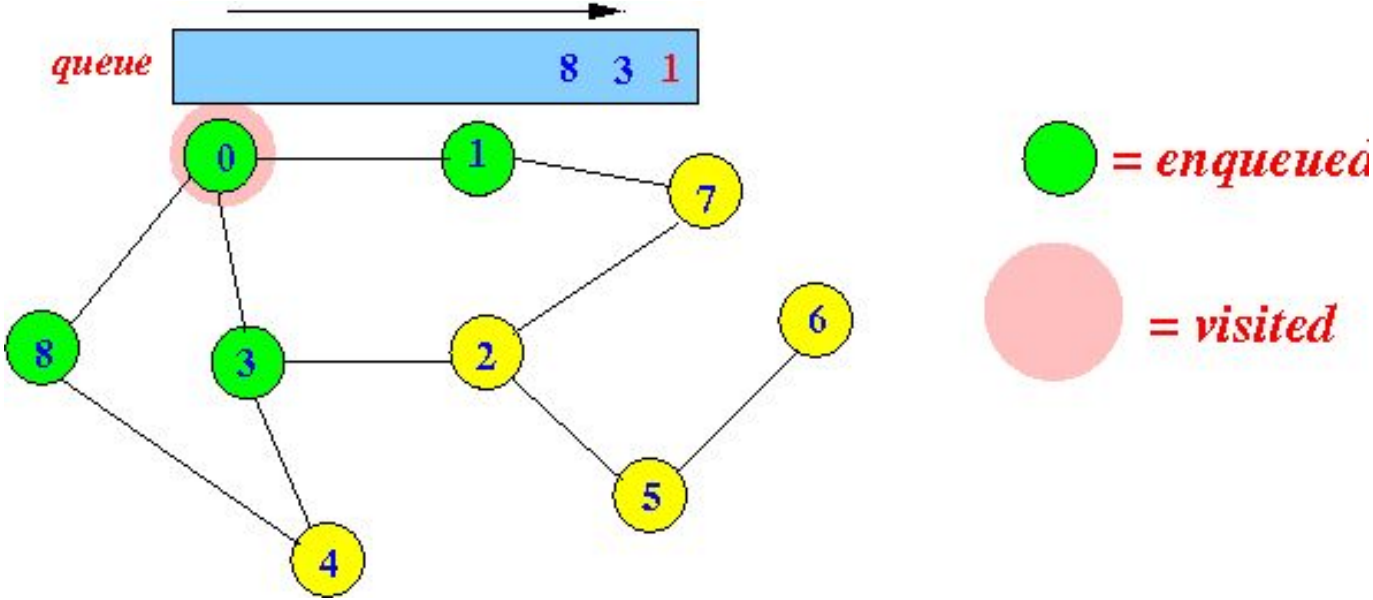
deleteHighestPriority(): Removes the highest priority item.

Queue*

Example: queue and breadth-first search

UCLA Samueli Computer Science

- Breadth-first Search (BFS) on graph (will be later lectures or CS180)



Queue*

Example: Use queue to implement BFS [Link]

UCLA Samueli Computer Science

```
void Graph::BFS(int s)
{
    bool *visited = new bool[V];  // Mark all the vertices as not visited
    for(int i = 0; i < V; i++) { visited[i] = false; }
    list<int> queue; // Create a queue for BFS
    visited[s] = true; // Mark the current node as visited and enqueue it
    queue.push_back(s);
    list<int>::iterator i;
    while(!queue.empty()) {
        s = queue.front(); // Dequeue a vertex from queue and print it
        queue.pop_front();
        for (i = adj[s].begin(); i != adj[s].end(); ++i) {
            if (!visited[*i]) {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

Note: Get all adjacent vertices of the dequeued vertex s. If a adjacent has not been visited, then mark it visited and enqueue it.

Queue*

Example: Implement Queue using Stacks

UCLA Samueli Computer Science

Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue.
pop() -- Removes the element from in front of queue.
peek() -- Get the front element.
empty() -- Return whether the queue is empty.

Example:

```
MyQueue queue = new MyQueue();

queue.push(1);
queue.push(2);
queue.peek();  // returns 1
queue.pop();   // returns 1
queue.empty(); // returns false
```

Queue*

Example: Implement Queue using Stacks

UCLA Samueli Computer Science

```
class Queue {
    stack<int> que_in, que_out;
public:
    void connect() {
        while(que_in.size()) {
            que_out.push(que_in.top());
            que_in.pop();
        }
    }

    // Push element x to the back of queue.
    void push(int x) {
        que_in.push(x);
    }

    // Removes the element from in front of queue.
    void pop(void) {
        if (que_out.empty()) {
            connect();
        }
        que_out.pop();
    }

    // Get the front element.
    int peek(void) {
        if (que_out.empty()) {
            connect();
        }
        return que_out.top();
    }

    // Return whether the queue is empty.
    bool empty(void) {
        return que_out.empty() && que_in.empty();
    }
};
```

Suggestions on Stack and Queue

UCLA Samueli Computer Science

- Drawing pictures and carefully tracing the current status through your code, updating the picture with each statement, can help you find bugs in your code.
- **Infix to postfix conversion is very important!**
- We have shown that stack and queue can be used for traversal on graphs. They can also be applied on Trees (topics later in this quarter). Understand different data structures you use will result in different traversal order.
- You can use the given **Standard Template Library (STL)** to implement stack and queue.

```
#include <stack>
#include <queue>
std::stack<type> variableName;
std::queue<type> variableName;
```

Hints for Homework 2

UCLA Samueli Computer Science

Task: Use stack or queue to implement the `pathExists()` function in the maze.

```
// Homework 2 (P1-4): One possible solution

bool pathExists(string maze[], int sr, int sc,
int er, int ec){
  // Some basic case to return false??
  // initialize stack or queue
  // Push starting coordinate (sr,sc)
  // Loop & check (when stack/queue is non-empty)
  // Return true or false
}
int main(){ … } //test your code
```

Note & Reminders:

1. Follow the pseudocode if you still have no idea. The question itself is somewhat similar to BFS and DFS.
2. How to mark the `(r,c)` as visited?
3. Using stack and queue will result **different Coord visit order** in your `hw.txt` to be submitted.
4. Reminder: **You may either write your own queue class, or use the queue type from the C++ Standard Library.**

Group Exercises: Worksheet

UCLA Samueli Computer Science

- Exercise problems from **Worksheet 4** (see “LA worksheet” tab in CS32 website). Answers will be posted after all discussions.
- Questions for today:
  - Valid Parentheses and Brackets
  - Reverse a Queue
  - Implement Stack Using Queue

Group Exercises: Worksheet #1

UCLA Samueli Computer Science

Valid Parentheses and Brackets

Given a string of '(', ')', '[', and ']', write a function to check if the input string is *valid*. Validity is determined by each '(' having a corresponding ')', and each '[' having a corresponding ']', with parentheses being properly nested and brackets being properly nested. (Hint: Use a stack)

```
bool isValid(string symbols);
```

Examples:

“[()([])[[([][])]]]” → Valid

“((([(])))“ → Invalid

“(()))” → Invalid

“()[]” → Valid

Group Exercises: Worksheet #1

UCLA Samueli Computer Science

Pseudocode:

- we maintain a stack containing all opened paren/bracket to keep track
- if we encounter an open paren/bracket we push it onto stack
- if we encounter a closed paren/brack and it matches the top of the stack: ‘)’ matches ‘(‘ and ‘]’ matches ‘[’
  - then we pop off of the stack
- else (the brackets do not match), return false
- return true at end

Group Exercises: Worksheet #2

UCLA Samueli Computer Science

Reverse a Queue

Give an algorithm for reversing a queue Q. Only following standard operations are allowed on queue:

a) Q.push(x) : Add an item x to the back of the queue.
b) Q.pop() : Remove an item from the front of the queue.
c) Q.front() : Return the item at the front of the queue
d) Q.empty() : Checks if the queue is empty or not.

You may use an additional data structure if you wish. (Hint: Use a stack)

Group Exercises: Worksheet #2

UCLA Samueli Computer Science

Key: Removing values from a stack occurs in reverse order

Example:

Push order: 1, 2, 3, 4, 5

Pop order: 5, 4, 3, 2, 1

Pseudocode:

- Push all values from the queue into another stack
- Pop values from the stack into the queue again

Group Exercises: Worksheet #3

UCLA Samueli Computer Science

Create a Stack from a Queue

Implement a Stack class using only queues as data structures. This class should implement the ***empty***, ***size***, ***top***, ***push***, and ***pop*** member functions, as specified by the standard library's implementation of stack. (The implementation will not be very efficient.)

```
bool empty() const;
size_t size() const;
int& top();
void push(const int& value);
void pop();
```

Group Exercises: Worksheet #3

UCLA Samueli Computer Science

Using the STL queue class:

empty: use empty function for queue

size: using the size function for queue

top: use the back function for queue (back of queue = top of stack)

push: use the push function for queue

pop: remove n - 1 elements from the queue onto the back when we reach the nth element, only remove this element - allows us to retain all other elements except the top of the stack, or the last element in queue