# CS256

## Chapter 17: Stacks and Queues

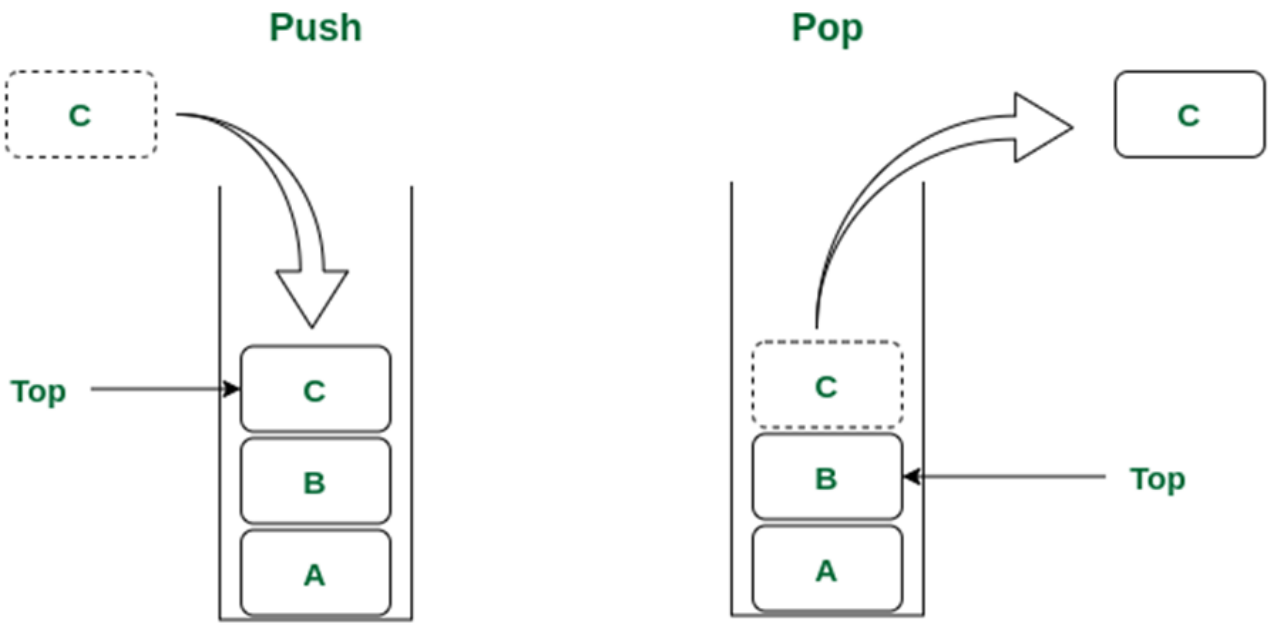
Hanoi University of Science and Technology

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Eighth Edition, Pearson, 2015
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

# Introduction to Stack

- Stack: a LIFO (last in, first out) data structure
- Implementation:
  - static: fixed size, implemented as array
  - dynamic: variable size, implemented as linked list




**Stack Data Structure**

# Stack Operations and Functions

- Operations:
  - `push`: add a value onto the top of the stack
  - `pop`: remove a value from the top of the stack
- Functions:
  - `isFull`: true if the stack is currently full, i.e., has no more space to hold additional elements
  - `isEmpty`: true if the stack currently contains no elements

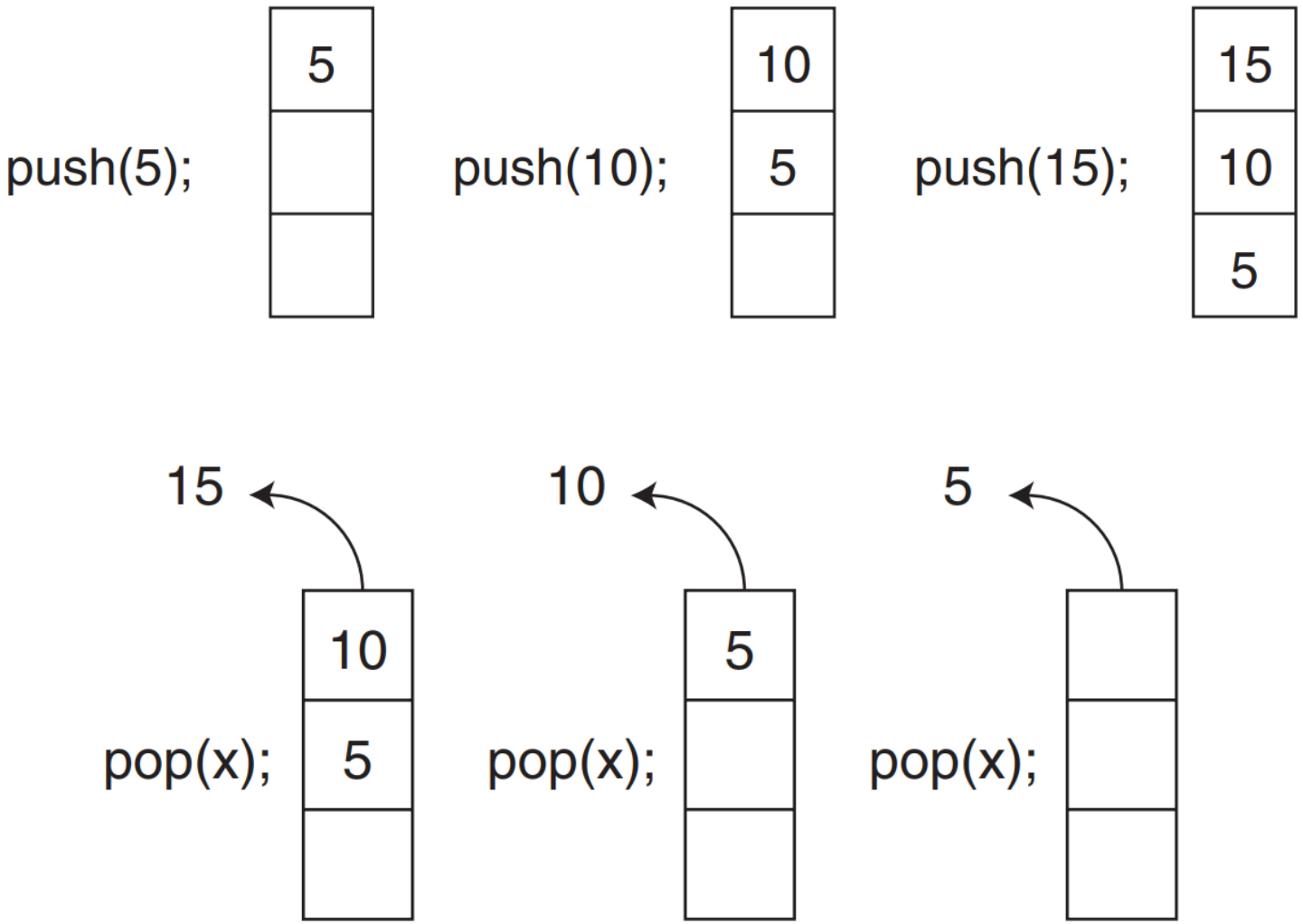
# Stack Operations and Functions

```
#ifndef INTSTACK_H
#define INTSTACK_H
class IntStack {
private:
    int *stackArray;  // Pointer to the stack array
    int stackSize;    // The stack size
    int top;          // Indicates the top of the stack

public:
    IntStack(int); // Constructor
    IntStack(const IntStack &);  // Copy constructor
    ~IntStack(); // Destructor

    // Stack operations
    void push(int);
    void pop(int &);
    bool isFull() const;
    bool isEmpty() const;
};
#endif
```

# Stack Operations and Functions



# Implementing Other Stack Operations

```
// Specification file for the MathStack class
#ifndef MATHSTACK_H
#define MATHSTACK_H
#include "IntStack.h"

class MathStack : public IntStack
{
public:
    // Constructor
    MathStack(int s) : IntStack(s) {}

    // MathStack operations
    void add();
    void sub();
};
#endif
```

# Implementing Other Stack Operations

```
#include "MathStack.h"

//*****************************************
// Member function add. add pops          *
// the first two values off the stack and *
// adds them. The sum is pushed onto the stack. *
//*****************************************

void MathStack::add()
{
    int num, sum;

    // Pop the first two values off the stack.
    pop(sum);
    pop(num);

    sum += num; // Add the two values, store in sum.
    push(sum); // Push sum back onto the stack.
}
```

# Implementing Other Stack Operations

```
//*****************************************
// Member function sub. sub pops the      *
// first two values off the stack. The    *
// second value is subtracted from the    *
// first value. The difference is pushed  *
// onto the stack.                        *
//*****************************************

void MathStack::sub()
{
    int num, diff;

    // Pop the first two values off the stack.
    pop(diff);
    pop(num);

    diff -= num; // Subtract num from diff.
    push(diff); // Push diff back onto the stack.
}
```

# Stack Container in C++

```
stack<int, vector<int>> iStack; // Vector stack
stack<int, list<int>> iStack;   // List stack
stack<int> iStack;              // Default - deque stack
```

**Table 18-3**

| Member Function | Examples and Description |
|---|---|
| `empty` | `if (myStack.empty())`<br>The `empty` member function returns `true` if the `stack` is empty. If the `stack` has elements, it returns `false`. |
| `pop` | `myStack.pop();`<br>The `pop` function removes the element at the top of the stack. |
| `push` | `myStack.push(x);`<br>The `push` function pushes an element with the value `x` onto the stack. |
| `size` | `cout << myStack.size() << endl;`<br>The `size` function returns the number of elements in the `list`. |
| `top` | `x = myStack.top();`<br>The `top` function returns a reference to the element at the top of the stack. |

# Stack Container in C++

```
// This program demonstrates the STL stack
// container adapter.
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int main()
{
    const int MAX = 8;   // Max value to store in the stack
    int count;           // Loop counter

    // Define an STL stack
    stack< int, vector<int> > iStack;
```

# Stack Container in C++

```
    // Push values onto the stack.
    for (count = 2; count < MAX; count += 2)
    {
        cout << "Pushing " << count << endl;
        iStack.push(count);
    }

    // Display the size of the stack.
    cout << "The size of the stack is ";
    cout << iStack.size() << endl;

    // Pop the values off the stack.
    for (count = 2; count < MAX; count += 2)
    {
        cout << "Popping " << iStack.top() << endl;
        iStack.pop();
    }
    return 0;
}
```

# Introduction to Queue

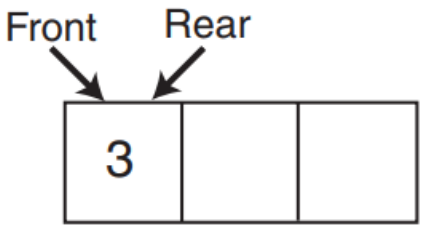
- Queue: a FIFO (first in, first out) data structure
- Implementation:
  - static: fixed size, implemented as array
  - dynamic: variable size, implemented as linked list

# Queue Locations and Operations

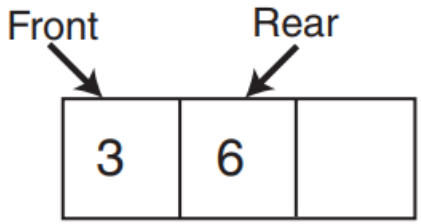
- rear: position where elements are added
- front: position from which elements are removed
- enqueue: add an element to the rear of the queue
- dequeue: remove an element from the front of a queue

# Enqueue

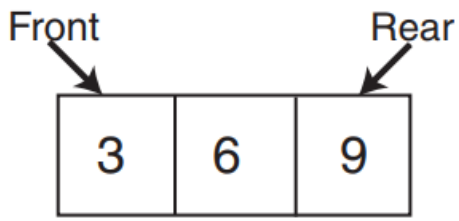
enqueue(3);




enqueue(6);

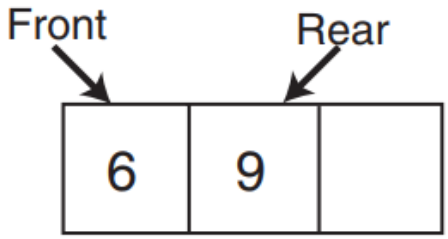



enqueue(9);

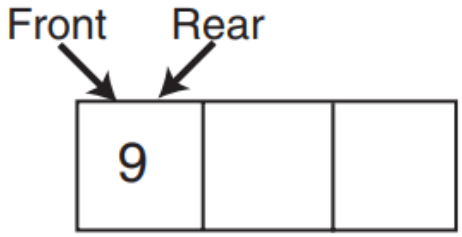


# Dequeue

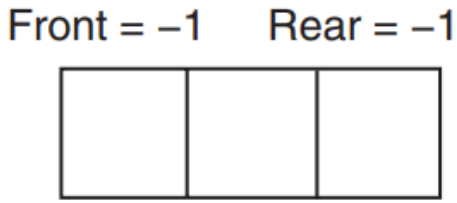
dequeue();



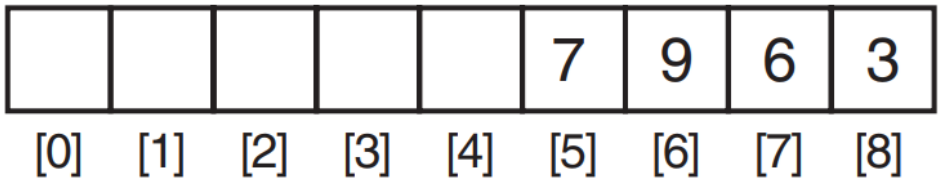

dequeue();



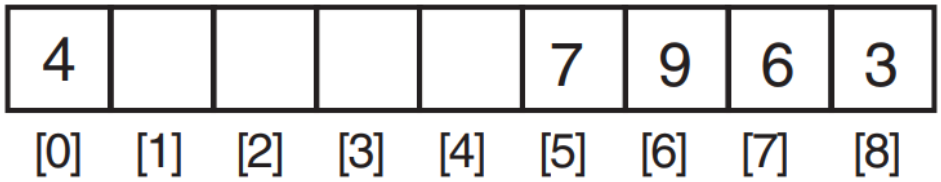

dequeue();



# Queue with Circular Array




rear = 8
front = 5




rear = 0
front = 5

# IntQueue class

```
#ifndef INTQUEUE_H
#define INTQUEUE_H

class IntQueue
{
private:
     int *queueArray;   // Points to the queue array
     int queueSize;     // The queue size
     int front;         // Subscript of the queue front
     int rear;          // Subscript of the queue rear
     int numItems;      // Number of items in the queue
public:
     // Constructor
     IntQueue(int);

     // Copy constructor
     IntQueue(const IntQueue &);
```

# IntQueue class

```
    // Destructor
    ~IntQueue();

    // Queue operations
    void enqueue(int);
    void dequeue(int &);
    bool isEmpty() const;
    bool isFull() const;
    void clear();
};
#endif
```

# deque and queue Containers

| Feature | **std::queue** (Queue) | **std::deque** (Double-ended queue) |
| --- | --- | --- |
| **Add / Remove Elements** | **push** at the back, **pop** at the front only | Both ends (**push** / **pop** at both **front** and **back**) |
| **Random Access** | Not allowed ❌ | Allowed (using **[i]**) ✅ |

# deque and queue Containers

- Given a string containing just the characters (, ), {, }, [, and ], determine if the input string is valid. Example: "{[()]}" is valid, but "{[(])}" is invalid.
- Given a positive integer, print its binary representation.
- Given a string. If two identical letters are adjacent, they cancel each other out. Return the final string after all possible cancellations. (Example: "abbaca" $\rightarrow$ cancels bb to "aaca" $\rightarrow$ cancels aa to "ca").
- Given a string of characters. Check if the string is a Palindrome (reads the same forwards and backwards). Example: "RADAR"