Data Structures and Algorithms

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 4

Stack

Array Based

Course Roadmap

Part 1: Linear Data Structures

Lecture 1: Complexity Analysis and Recursion

Lecture 2: Arrays

Lecture 3: Linked List

Lecture 4: Stack

Lecture 5: Queue

Lecture 6: Deque

Lecture 7: STL in C++ (Linear Data Structures)

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to Stack

2- Insertion Operation

3- Deletion Operation

4- Top Operation

5- Time Complexity & Space Complexity

Let's
STARTUP

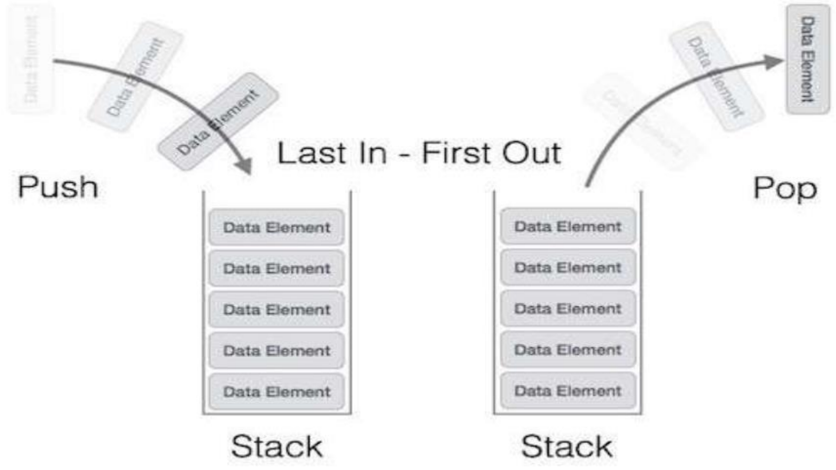
Lecture Agenda

Introduction to Stack

- A stack is an abstract data type that serves as a collection of elements, with two principal operations:

- push, which adds an element to the collection.
- pop, which removes the most recently added element that was not yet removed.

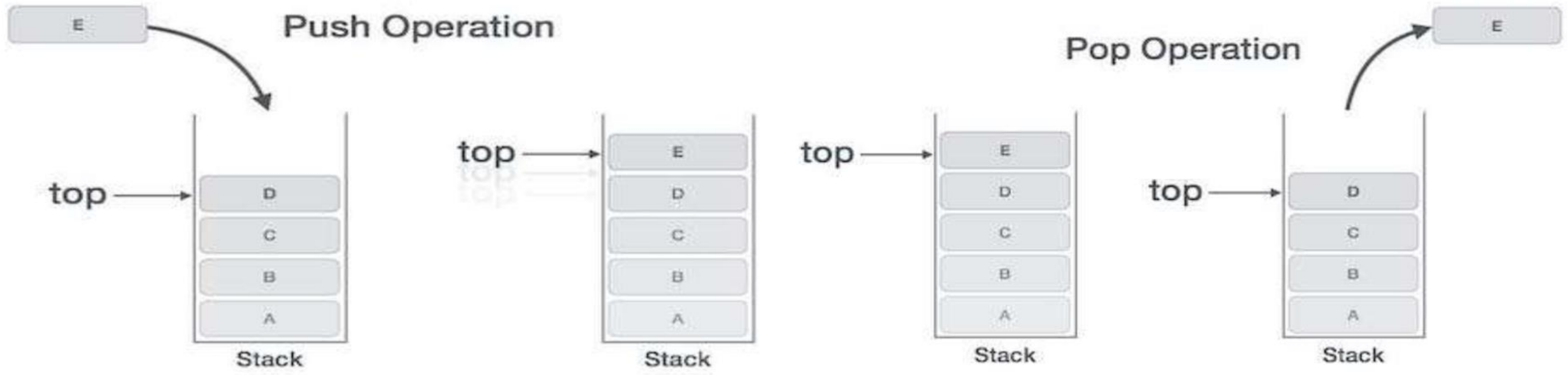
- The order in which elements come off a stack gives rise to its alternative name, LIFO (last in, first out). Additionally, a peek operation may give access to the top without modifying the stack. The name "stack" for this type of structure comes from the analogy to a set of physical items stacked on top of each other. This structure makes it easy to take an item off the top of the stack, while getting to an item deeper in the stack may require taking off multiple other items first.



Introduction to Stack

Inserting and deleting elements:

- Stacks have restrictions on the insertion and deletion of elements. Elements can be inserted or deleted only from one end of the stack i.e. from the . The element at the top is called the element. The operations of inserting and deleting elements are called push() and pop() respectively.
- When the top element of a stack is deleted, if the stack remains non-empty, then the element just below the previous top element becomes the new top element of the stack. For example, in the stack of trays, if you take the tray on the top and do not replace it, then the second tray automatically becomes the top element (tray) of that stack.



Introduction to Stack

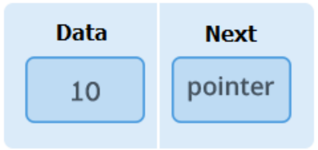
- Stacks are dynamic data structures that follow the Last In First Out (LIFO) principle. The last item to be inserted into a stack is the first one to be deleted from it. For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.
- Considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the top of the stack. This data structure makes it possible to implement a stack as a singly linked list and a pointer to the top element. A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack.
- Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and de-allocates memory space.

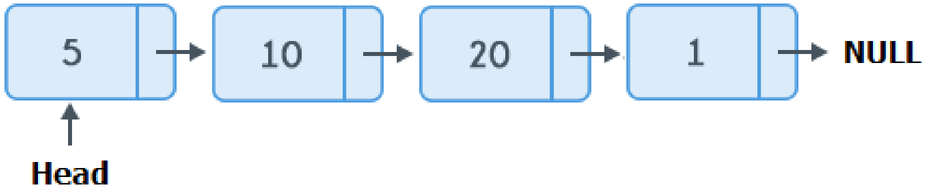
Introduction to Stack

- **Following are the basic operations supported by a stack.**
- Push: which adds an element to the collection.
- Pop: which removes the most recently added element that was not yet removed.
- Top: which gets the most recently added element.

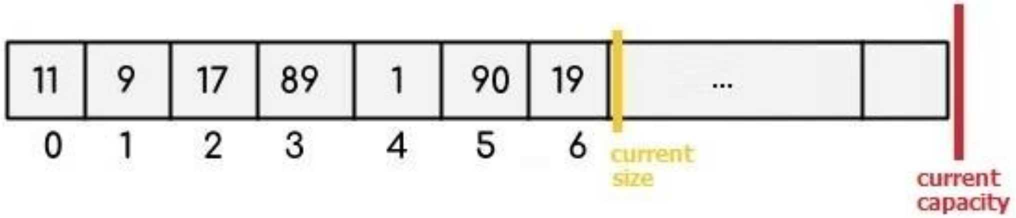
- **Stack Types**

1. Stack (Linked List Based)





2. Stack (Array Based)



Reserve Method - Stack (Array Based)

```
// Initialize a stack with dynamic length
int n;
int capacity;
int* arr;

// This function updates the capacity of the stack
void reserve(int new_capacity) {
    // Initialize a new stack with the new capacity
    int* temp = new int[new_capacity];
    // copy the elements in the current stack to the new stack
    for (int i = 0; i < n; i++)
        temp[i] = arr[i];
    // delete the old stack
    delete[] arr;
    // set the temp stack with new capacity to be the stack
    arr = temp;
    // set the current capacity of the stack to be the new capacity
    capacity = new_capacity;
}
```

Stack-Array-Based.cpp

Lecture Agenda

Insertion Operation - Stack (Array Based)

- Insert Operation is to insert a data element into a stack.

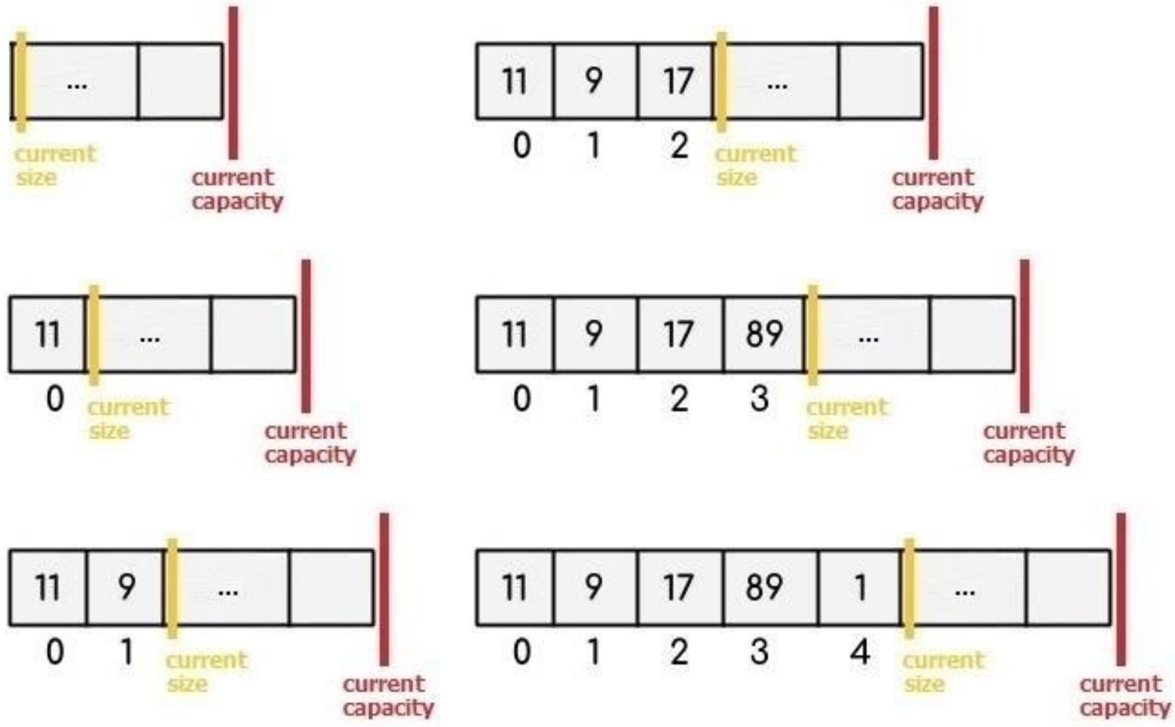
- ***Insertion Algorithm***:

If the array is full increase the capacity

$\mathbf{1.}\ \boldsymbol{arr[n] = data}$

$\mathbf{2.}\ \boldsymbol{n = n + 1}$

- Insert 11, 9, 17, 89, 1



Insertion Operation - Stack (Array Based)

```
// This function inserts an element at the begin of the stack
void push(int new_data) {
    // check if we need to update the capacity of the stack
    if (n == capacity)
        reserve(2 * capacity + 1);
    // insert the new element
    arr[n] = new_data;
    // update the size of the stack
    n = n + 1;
}
```

Stack-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Stack
- ✓ Section 2: Insertion Operation
- Section 3: Deletion Operation
- Section 4: Top Operation
- Section 5: Time Complexity & Space Complexity

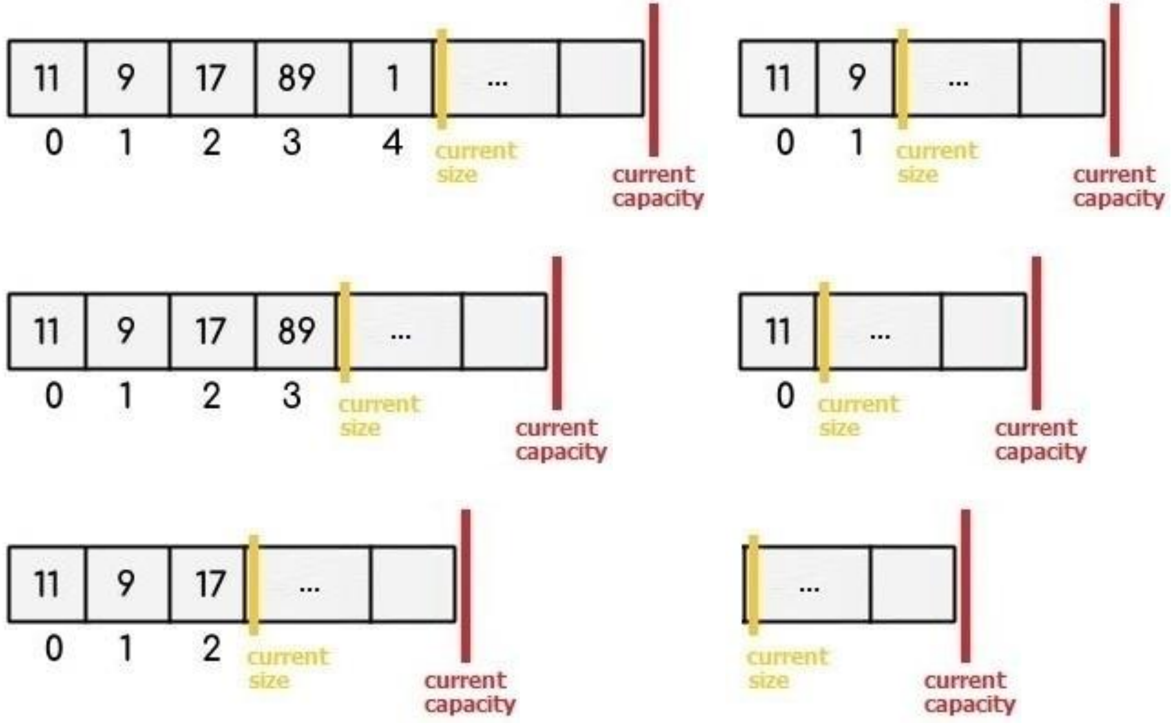
Deletion Operation - Stack (Array Based)

- Delete Operation removes the most recently added element that was not yet removed.
- $Deletion\ Algorithm$:

$\mathbf{1.}\ \boldsymbol{n = n - 1}$

If the array is half full decrease the capacity

- Delete 1, 89, 17, 9, 11



Deletion Operation - Stack (Array Based)

```
// This function deletes the first element in the stack
void pop() {
    // check if the stack is empty
    if (n == 0)
        return;
    // update the size of the stack
    n = n - 1;
    // check if we need to update the capacity of the stack
    if (n < capacity / 2)
        reserve(capacity / 2);
}
```

Stack-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Stack
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- Section 4: Top Operation
- Section 5: Time Complexity & Space Complexity

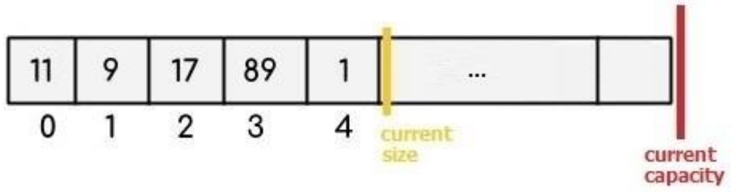
Top Operation - Stack (Array Based)

- Top Operation gets the most recently added element.

- $Top\ Algorithm$:

```
1. $return\ arr[n-1]$
```

- Top is 1



Top Operation - Stack (Array Based)

```
// This function returns the value of the first element in the stack
int top() {
    // check if the stack is empty
    // to return the biggest integer value as an invalid value
    if (n == 0)
        return INT_MAX;
    // otherwise return the real value
    else
        return arr[n-1];
}
```

Stack-Array-Based.cpp

Functionality Testing - Stack (Array Based)

- Initialize a global array

```
#include <bits/stdc++.h>
using namespace std;

// Initialize a stack with dynamic length
int n;
int capacity;
int* arr;
```

- In the Main function:

```
cout << "Stack top: " << top() << '\n';
```

- Expected Output:

```
Stack top: 2147483647
```

Stack-Array-Based.cpp

Functionality Testing - Stack (Array Based)

- In the Main function:

```
push(10);
cout << "Stack top: " << top() << '\n';
push(20);
cout << "Stack top: " << top() << '\n';
push(30);
cout << "Stack top: " << top() << '\n';
push(40);
cout << "Stack top: " << top() << '\n';
push(50);
cout << "Stack top: " << top() << '\n';
```

- Expected Output:

```
Stack top: 10
Stack top: 20
Stack top: 30
Stack top: 40
Stack top: 50
```

Stack-Array-Based.cpp

Functionality Testing - Stack (Array Based)

- In the Main function:

```cpp
while (n > 0) {
    cout << "Stack top: " << top() << '\n';
    pop();
    cout << "Stack top has been deleted\n";
}
cout << "Stack is empty now\n";
```

- Expected Output:

```
Stack top: 50
Stack top has been deleted
-------------------------------
Stack top: 40
Stack top has been deleted
-------------------------------
Stack top: 30
Stack top has been deleted
-------------------------------
Stack top: 20
Stack top has been deleted
-------------------------------
Stack top: 10
Stack top has been deleted
-------------------------------
Stack is empty now
```

Stack-Array-Based.cpp

Functionality Testing - Stack (Array Based)

- In the Main function:

```
push(10);
cout << "Stack top: " << top() << '\n';
push(20);
cout << "Stack top: " << top() << '\n';
push(30);
cout << "Stack top: " << top() << '\n';
```

- Expected Output:

```
Stack top: 10

Stack top: 20

Stack top: 30
```

Stack-Array-Based.cpp

Functionality Testing - Stack (Array Based)

- In the Main function:

```cpp
while (n > 0) {
    cout << "Stack top: " << top() << '\n';
    pop();
    cout << "Stack top has been deleted\n";
}
cout << "Stack is empty now\n";
```

- Expected Output:

```
Stack top: 30
Stack top has been deleted
---------------------------
Stack top: 20
Stack top has been deleted
---------------------------
Stack top: 10
Stack top has been deleted
---------------------------
Stack is empty now
```

Stack-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Stack
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Top Operation
- Section 5: Time Complexity & Space Complexity

Time Complexity & Space Complexity

- Time Analysis

	Worst Case	Average Case
• Push	$\Theta(n)$	$\theta(1)$
• Pop	$\Theta(n)$	$\theta(1)$
• Top	$\Theta(1)$	$\theta(1)$

Lecture Agenda

- ✓ Section 1: Introduction to Stack
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Top Operation
- ✓ Section 5: Time Complexity & Space Complexity

Practice

Practice

1- Reverse string using stack
2- Check string is palindrome or not
3- Convert Infix Expression to Postfix Expression
4- Convert Infix Expression to Prefix Expression
5- Convert Postfix Expression to Infix Expression
6- Convert Prefix Expression to Infix Expression
7- Convert Postfix Expression to Prefix Expression
8- Convert Prefix Expression to Postfix Expression
9- Evaluation of Prefix & Infix & Postfix Expressions
10- Reverse a stack using recursion
11- Check for balanced parentheses in an expression
12- Length of the longest valid substring
13- Minimum number of bracket reversals needed to make an expression balanced
14- Next Greater Element
15- Delete middle element of a stack

Practice

16- Reverse individual words
17- Largest Rectangular Area in a Histogram
18- Find maximum depth of nested parenthesis in a string
19- Expression contains redundant bracket or not
20- Check if two expressions with brackets are same
21- Delete consecutive same words in a sequence
22- Remove brackets from an algebraic string
23- Range Queries for Longest Correct Bracket Subsequence
24- Check if stack elements are pairwise consecutive
25- Reverse a number using stack
26- Tracking current Maximum Element in a Stack
27- Decode a string recursively encoded as count followed by substring
28- Find maximum difference between nearest left and right smaller elements
29- Find if an expression has duplicate parenthesis or not
30- Find index of closing bracket for a given opening bracket in an expression

Assignment

Implement STL Stack

- Stacks are a type of container adaptor, specifically designed to operate in a (last-in first-out) context, where elements are inserted and extracted only from one end of the container.
- Stacks are implemented as container adaptors, which are classes that use an encapsulated object of a specific container class as its underlying container, providing a specific set of member functions to access its elements. Elements are pushed/popped from the "back" of the specific container, which is known as the top of the stack.
- Stacks are a type of container adaptors with (Last In First Out) type of working, where a new element is added at one end and (top) an element is removed from that end only.

More Info: cplusplus.com/reference/stack/stack/
More Info: en.cppreference.com/w/cpp/container/stack
More Info: geeksforgeeks.org/stack-in-cpp-stl/

Implement STL Stack

- Member functions:
  - (constructor) Construct stack (public member function)
  - (empty) Test whether container is empty (public member function)
  - (size) Return size (public member function)
  - (top) Access next element (public member function)
  - (push) Insert element (public member function)
  - (pop) Remove top element (public member function)
  - (swap) Swap contents (public member function)

More Info: cplusplus.com/reference/stack/stack/
More Info: en.cppreference.com/w/cpp/container/stack

DO
MORE.