



SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai



DEPARTMENT OF INFORMATION TECHNOLOGY

19ITT101-PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT 4 – STACK AND QUEUE

TOPIC 1 – Stack ADT



INTRODUCTION

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.
- A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only.
- Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.





Stack Data Structure



- This feature makes it LIFO data structure. LIFO stands for Last-in-first-out.
- Here, the element which is placed (inserted or added) last, is accessed first.
- In stack terminology, insertion operation is Called PUSH operation and removal operation is called POP operation.





Operation of Stack

- **push()** – Pushing (storing) an element on the stack.
- **pop()** – Removing (accessing) an element from the stack.
- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.



peek()



- **peek()** – get the top data element of the stack, without removing it.

```
int peek() {  
    return stack[top];  
}
```



isfull()



➤ **isFull()** – check if stack is full.

```
bool isfull() {  
    if(top == MAXSIZE)  
        return true;  
    else  
        return false;  
}
```



isEmpty()



- **isEmpty()** – check if stack is empty.

```
bool isEmpty() {  
    if(top == -1)  
        return true;  
    else  
        return false;  
}
```



Push()



- **push()** – Pushing (storing) an element on the stack.
 - ❖ **Step 1** – Checks if the stack is full.
 - ❖ **Step 2** – If the stack is full, produces an error and exit.
 - ❖ **Step 3** – If the stack is not full, increments top to point next empty space.
 - ❖ **Step 4** – Adds data element to the stack location, where top is pointing.
 - ❖ **Step 5** – Returns success.





Push()

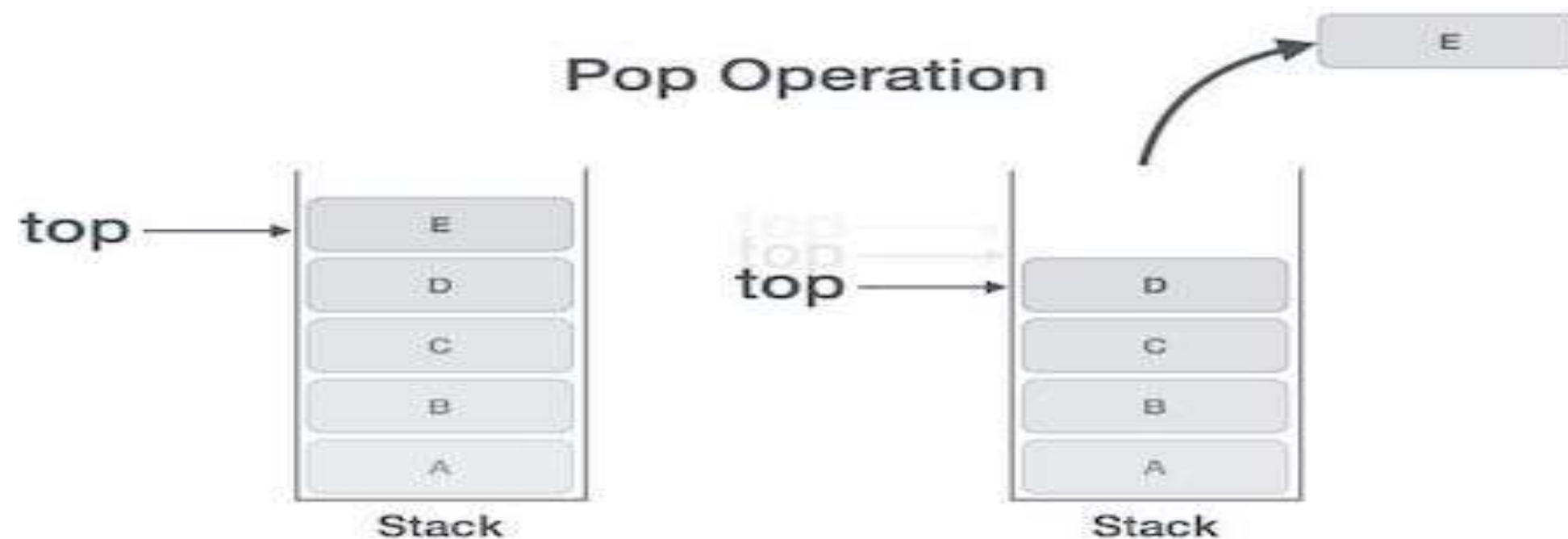


```
void push(int data) {  
    if(!isFull()) {  
        top = top + 1;  
        stack[top] = data;  
    } else {  
        printf("Could not insert data, Stack is full.\n");  
    }  
}
```



Pop()

- **pop()** – Removing (accessing) an element from the stack.
 - ❖ **Step 1** – Checks if the stack is empty.
 - ❖ **Step 2** – If the stack is empty, produces an error and exit.
 - ❖ **Step 3** – If the stack is not empty, accesses the data element at which top is pointing.
 - ❖ **Step 4** – Decreases the value of top by 1.
 - ❖ **Step 5** – Returns success.





Pop()



```
int pop(int data) {  
  
    if(!isempty()) {  
        data = stack[top];  
        top = top - 1;  
        return data;  
    } else {  
        printf("Could not retrieve data, Stack is empty.\n");  
    }  
}
```

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.htm