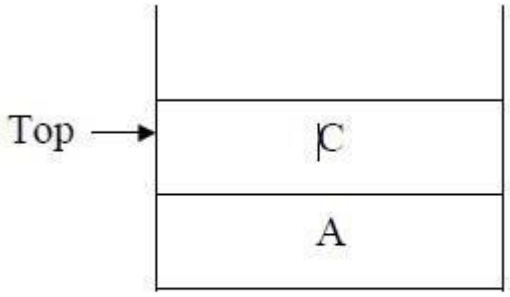
2.9 THE STACK ADT

Stack Model :

- A stack is a linear data structure which follows Last In First Out (LIFO) principle.
- LIFO principle means the **last inserted element is the first one to be removed**.
- Insertion and deletion occur at only one end called the Top.



Stack Model

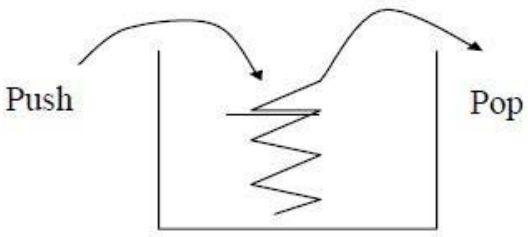
Examples: Pile of coins., a Stack of books, Laundry pile, **Undo operation** in applications.

OPERATIONS ON STACK

The fundamental operations performed on a stack are

1. Push

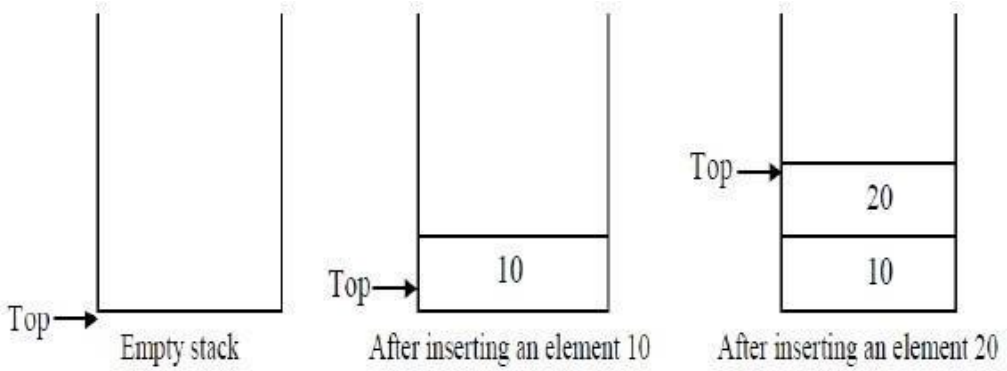
2. Pop



Operations on stack

1. PUSH :

- The process of **inserting a new element to the top of the stack.**
    - **Before insertion**, it is necessary to **check whether the stack is full**.
    - If the stack is full, **insertion cannot be performed**.
    - Attempting to insert an element when the stack is full is called **Stack Overflow**.
- For every push operation the **Top is incremented by 1**.

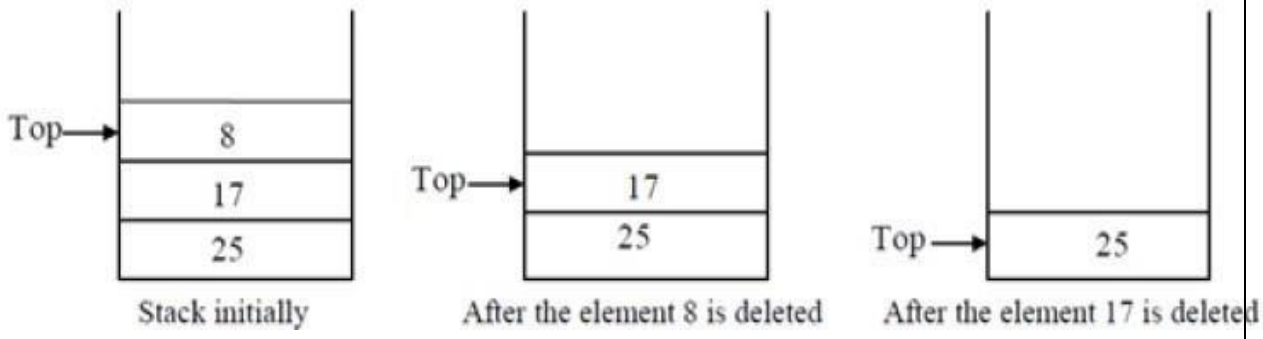


Note: Stack overflow is the condition to be checked before insertion an element in to the stack.

2. POP :

- The process of deleting an element from the top of stack is called pop operation.
    - **Before deletion**, it is necessary to **check whether the stack is empty**.
    - If the stack is empty, **deletion cannot be performed**.
    - Attempting to remove an element from an empty stack is called **Stack Underflow**.

- After every pop operation the Top pointer is decremented by 1.



Note: **Stack underflow is the condition to be checked** before removing an element from the stack**.**

IMPLEMENTATION OF STACK ADT

Stack can be implemented using

1. Array 2. Linked List.

1. Array Implementation

- In this type, the stack is implemented using a **fixed-size array**.
- Elements are stored sequentially in the array.
- The **top** keeps track of the last inserted element.

Steps Involved in Stack Operations

1. Initialize the Stack

- Set **Top = -1** to indicate that the stack is empty.

2. Push (Insertion) Operation

- **Step 1:** Check if the stack is full.
- **Step 2:** If not full, increment **Top** by 1.

- **Step 3:** Insert the new element at position stack[Top].
- **Step 4:** If the stack is full, display **Stack Overflow**.

3. Pop (Deletion) Operation

- **Step 1:** Check if the stack is empty.
- **Step 2:** If not empty, return the value at stack[Top].
- **Step 3:** Decrement **Top** by 1.
- **Step 4:** If the stack is empty, display **Stack Underflow**.

4. Peek / Top Element (Optional)

- Check if the stack is empty.
- If not empty, return the element at stack[Top] without removing it.

5. isEmpty() and isFull() Checks

- **isEmpty:** Returns true if Top = -1.
- **isFull:** Returns true if Top = MAX-1 (for array-based stack).

C++ implementation of each stack function

Check for Stack Overflow (isFull())

```
int isFull()
   {
      return top == MAX - 1;
   }
```

Check for Stack Underflow (isEmpty())

```
int isEmpty()
   {
      return top == -1;
   }
```

Push Operation (Insertion)

```
void push(int element)
{
    if(!isFull())
    {
        top++;
        stack[top] = element;
        cout << "Inserted " << element << " into stack\n";
    }
    else
    {
        cout << "Stack is full. Cannot insert.\n";
    }
}
```

Pop Operation (Deletion):

```
void pop()
{
    if(!isEmpty())
    {
        cout << "Deleted " << stack[top] << " from stack\n";
        top--;
    }
    else
    {
        cout << "Stack is empty. Cannot delete.\n";
    }
}
```

Display Stack Elements:

```
void display()
```

```
{
    if(isEmpty())
    {
        cout << "Stack is empty.\n";
    }
    else
    {
        cout << "Stack elements are:\n";
        for(int i = top; i >= 0; i--)
            cout << stack[i] << "\n";
    }
}
```

Advantages of Stack

- Easy to implement
- Efficient insertion and deletion
- Requires less memory overhead

Disadvantages:

- Size is fixed.
- Limited access to elements
- if the stack becomes full, **stack overflow** occurs.

Applications of Stack

- Function calls and recursion
- Expression evaluation
- Undo and Redo operations
- Reversing a string
- Checking balanced parentheses