# Stack

A **stack** is a linear data structure that follows the **Last In, First Out (LIFO)** or **First In, Last Out (FILO)** principle.

Elements are inserted and removed from only one end, known as the **top**.

---

## Basic Operations

1. **Push**:
    - The process of inserting an element into the stack.
    - Time Complexity: ( O(1) ).
2. **Pop**:
    - The process of removing the top element from the stack.
    - Time Complexity: ( O(1) ).
3. **Peek (or Top)**:
    - Returns the element at the top of the stack without removing it.
    - Time Complexity: ( O(1) ).
4. **isEmpty**:
    - Checks whether the stack is empty.
    - Returns `true` if the stack is empty, otherwise `false`.
5. **isFull**:
    - Checks whether the stack is full (relevant for fixed-size stacks).
    - Returns `true` if the stack is full, otherwise `false`.

---

## Error Conditions

1. **Stack Underflow**:
    - Occurs when trying to **pop** an element from an empty stack.
    - Example: Calling `pop()` on a stack with no elements.
2. **Stack Overflow**:
    - Occurs when trying to **push** an element into a full stack (in fixed-size implementations).
    - Example: Calling `push()` on a stack that has reached its maximum capacity.

---

## Pseudocode

### Stack Operations

```
// Push operation
public void push(int data){
    if(isFull())
        throw new StackOverFlowException("Stack Overflow");
```

```
    arr[++top]=data;

}

// Pop operation
public void pop(){
    if(isEmpty())
        throw new StackUnderFlowException("Stack UnderFlow");
    arr[top--]=0;
}

// Peek operation
public int peek(){
    if(isEmpty())
        throw new StackUnderFlowException("Stack UnderFlow");
    return arr[top];
}

// isEmpty operation
public boolean isEmpty(){
    return top==-1;
}

// isFull operation (for fixed-size stacks)
public boolean isEmpty(){
    return top==-1;
}
```

---

## Applications of Stack

1. **Function Calls**:
    - Stacks manage function calls and returns in programming languages.
2. **Expression Evaluation**:
    - Used in parsing and evaluating arithmetic expressions (e.g., infix to postfix conversion).
3. **Undo/Redo Operations**:
    - Stacks track changes in applications like text editors.
4. **Backtracking Algorithms**:
    - Examples include maze-solving and depth-first search (DFS).
5. **Syntax Checking**:
    - Compilers use stacks to check balanced parentheses or tags (e.g., HTML).

---

### Summary

- **LIFO Principle**: Last element added is the first one removed.
- **Key Operations**: Push, Pop, Peek, isEmpty, isFull.

- **Errors**: Underflow (empty stack) and Overflow (full stack).
- **Applications**: Function calls, expression evaluation, undo/redo, backtracking, and syntax checking.