DAS Stack Case Study

Members:

- **Modasir Hossain Khan**
- **Bishal Shah**
- **Sherin M Joseph**
- **Punitha L Gowda**
- **Rajnish Kumar**

Understanding Stacks: Data Structure in Computer Science

A **stack** is a fundamental data structure that operates on the **Last-In, First-Out (LIFO)** principle, meaning the last element added to the stack is the first one to be removed. This concept can be visualized as a stack of plates: you add or remove plates only from the top.

Core Concepts of Stacks

1. **LIFO Principle**:

   o   Elements are processed in reverse order of their insertion.

   o   Example: The last plate added to a stack is the first to be removed.

2. **Basic Operations**:

   o   **Push**: Adds an element to the top of the stack.

   o   **Pop**: Removes the top element from the stack.

   o   **Peek (or Top)**: Lets you view the top element without removing it.

   o   **isEmpty**: Checks whether the stack is empty.

Applications of Stacks

1. **Function Call Management**:

   o   When a function is called, its details (e.g., parameters, local variables) are stored in a stack (call stack).

- o As functions return, their details are removed (popped) from the stack.

- o Essential for recursion and nested function calls.

2. **Expression Evaluation**:

   - o Used in evaluating and converting expressions (e.g., infix to postfix).

   - o Ensures operators and operands are processed in the correct order.

3. **Backtracking**:

   - o In problems like maze solving or navigating decision trees, stacks are used to keep track of previous states to backtrack when necessary.

4. **Undo/Redo Mechanisms**:

   - o In text editors or drawing applications, every action is pushed onto a stack. Undoing an action involves popping the stack, while redoing involves adding back to the stack.

5. **Browser History Management**:

   - o Browsers maintain history using two stacks:

     - ▪ One stack for the "Back" button.

     - ▪ Another stack for the "Forward" button.

6. **Syntax Parsing**:

   - o Compilers use stacks to check for balanced parentheses or ensure proper syntax in programming code.

Implementations of Stacks

Stacks can be implemented in various ways, depending on the requirements:

1. **Using Arrays**:

   - o A fixed-size array can be used to implement a stack.

   - o Simple but requires predefining the maximum stack size, leading to limited flexibility.

2. • A linked list provides dynamic resizing and avoids the limitations of a fixed-size array.
3. • Each node contains data and a reference to the next node.

Advantages and Limitations

Advantages	Limitations
Simple and efficient for LIFO operations	Fixed size in arrays (unless dynamically resized)
Useful for recursion and backtracking	Limited by memory in linked list implementation
Intuitive for handling nested structures	Inefficient for random access of elements

Summary:

Stacks are a versatile and powerful data structure widely used in programming. They serve as the backbone of many critical operations, from managing function calls to enabling user-friendly features like undo/redo. Implementations via arrays and linked lists offer different trade-offs, allowing developers to choose based on their specific requirements.