📚 Pre-Class Reading Notes: Understanding Stacks

1. Welcome to Stacks! 📖

Hey there! 👋 In this lesson, we're diving into **Stacks**, a fundamental **linear data structure** that follows the **Last-In-First-Out (LIFO)** principle. Imagine a **stack of plates**—you can only take the top one first! 🍽️

By the end of this reading, you'll understand:
✔ What stacks are and how they work
✔ Key stack operations (`push`, `pop`, `top`)
✔ How to implement stacks using **arrays** and **linked lists**
✔ Applications of stacks in real-world scenarios
✔ Expression notations (Infix, Prefix, Postfix)

Let's get started! 🚀

2. What is a Stack? 📦

A **stack** is a type of **linear data structure** where:

- **Elements are inserted and removed only from one end** (called the **top**).
- It follows the **LIFO (Last-In-First-Out) principle**.

🔹 Real-Life Examples of Stacks:

- **A pile of books** 📚 → You remove the top one first.
- **Browser history** ↩️ → The most recent page you visited is popped first when you hit "Back".
- **Undo/Redo in text editors** 🔄 → Recent changes are stored in a stack!

3. Stack Abstract Data Type (ADT) & Operations ⚙️

A stack supports the following key operations:

Operation	Description
`push(x)`	Adds element `x` to the **top** of the stack.

Operation	Description
`pop()`	Removes and returns the **top** element.
`SIZE()`	Returns the **number of elements** in the stack.
`STACK-EMPTY()`	Checks if the stack is **empty**.
`TOP-ELEMENT()`	Returns the **top** element without removing it.

🔍 Working Example: `push(x)`

1️⃣ Increment the `top` variable.
2️⃣ Insert element at `top` position.
3️⃣ If the stack is **full**, an **OVERFLOW** error occurs.

🔍 Working Example: `pop()`

1️⃣ Remove element from `top` position.
2️⃣ Decrement the `top` variable.
3️⃣ If the stack is **empty**, an **UNDERFLOW** error occurs.

---

4. Implementing a Stack 🛠️

🔹 Using Arrays 📌

- Stores stack elements in a fixed-size array.
- **Operations (** `push` **,** `pop` **) take O(1) time.**
- `top` keeps track of the **last inserted element**.

Example: `PUSH(S, x)` **(Using Arrays)**

```
PUSH(S, x) {
  if (top == MAX-SIZE)
    error "OVERFLOW";
  else {
    top = top + 1;
    S[top] = x;
  }
}
```

Example: `POP()` (Using Arrays)

```
POP(S) {
  if (top == 0)
    error "UNDERFLOW";
  else {
    top = top - 1;
    return S[top + 1];
  }
}
```

🔹 Using Linked Lists 🔗

- **Doesn't require contiguous memory.**
- `push` and `pop` operations are performed at the **beginning** of the list (O(1) time complexity).

---

5. Applications of Stacks 🚀

Stacks are everywhere! Here are some **key applications**:
✔️ **Function Calls & Recursion** – Each function call is pushed onto a stack.
✔️ **Undo/Redo in text editors** – Your actions are stored in a stack.
✔️ **Balanced Parentheses Checker** – Used in programming languages.
✔️ **Expression Evaluation & Conversion** – Used in **mathematical expressions**.
✔️ **String Reversal** – Useful in reversing text.

---

6. Understanding Polish Notation (Infix, Prefix, Postfix) ➕➖

Stacks play a crucial role in **expression conversion and evaluation**.

📝 Expression Notations

Notation	Format	Example
Infix	`<Operand> <Operator> <Operand>`	`A + B`
Prefix (Polish Notation)	`<Operator> <Operand> <Operand>`	`+ A B`
Postfix (Reverse Polish Notation)	`<Operand> <Operand> <Operator>`	`A B +`

🔹 Converting Infix to Postfix

1️⃣ **Scan the expression left to right.**
2️⃣ **Push** operators onto a stack while maintaining precedence.
3️⃣ **Pop and append** operators when necessary.

Example: Infix ➡️ Postfix

```
(A + B) * C
```

✅ **Postfix:** `A B + C *`

---

7. Common Mistakes & How to Avoid Them ❌ ✅

🚨 **Forgetting LIFO Rule** → Always remove the **last inserted** element first!
🚨 **Not handling Stack Overflow/Underflow** → Always check before `push` or `pop`.
🚨 **Confusing Infix, Prefix, and Postfix Notations** → Use step-by-step conversion!

---

8. Conclusion & Next Steps 🎯

Great job! 🎉 You've now learned:
✅ What **Stacks** are and how they work.
✅ How to implement Stacks using **Arrays** and **Linked Lists**.
✅ **Expression Notations** and their conversion.
✅ Real-world **applications** of Stacks.

Next, we'll **dive deeper into recursion** and how stacks help execute **function calls efficiently**! Get ready for some **hands-on coding!** 💻 🔥

Happy Learning! 🚀