**Experiment No 3**

**Title:**

Real-Time Undo/Redo System Using Stack

**0bjective:**

To implement a real-time text editor simulation that supports **Undo** and **Redo** functionalities using **Stack** data structure.

**Problem Statement**

Develop a program that simulates a basic text editing application using **two stacks** to manage user actions. The application must support:

* Making changes to a document.
* Undoing the most recent change.
* Redoing a previously undone change.
* Displaying the current state of the document.

###  ****Theory****

A **stack** is a linear data structure that follows the **LIFO (Last In, First Out)** principle. It is ideal for implementing **Undo/Redo** functionalities:

* When a user makes a change, the previous state is pushed to the **Undo Stack**.
* When the user performs **Undo**, the current state is pushed to the **Redo Stack**, and the previous state is popped from the Undo Stack.
* When the user performs **Redo**, the last undone state is restored from the Redo Stack and pushed back onto the Undo Stack.

Using two stacks:

* undo\_stack: stores document states before each change.
* redo\_stack: stores states that were undone and can be redone.

###  ****Algorithm****

#### ****Initialization:****

* undo\_stack ← empty
* redo\_stack ← empty
* document ← ""

#### ****Make a Change (new\_text):****

1. Push current document to undo\_stack
2. Clear redo\_stack
3. Set document ← new\_text

#### ****Undo():****

1. If undo\_stack is not empty:
	* Push current document to redo\_stack
	* Pop top from undo\_stack and assign to document
2. Else: Print "Nothing to undo"

#### ****Redo():****

1. If redo\_stack is not empty:
	* Push current document to undo\_stack
	* Pop top from redo\_stack and assign to document
2. Else: Print "Nothing to redo"

#### ****Display():****

* Print the current value of document

### ****Test Cases****

| **Operation** | **Input Text** | **Output / Document State** |
| --- | --- | --- |
| Make Change | "Hello" | Hello |
| Make Change | "Hello World" | Hello World |
| Undo |  | Hello |
| Redo |  | Hello World |
| Undo |  | Hello |
| Make Change | "New Text" | New Text |
| Redo |  | Nothing to redo (Redo stack cleared) |

###  ****Conclusion****

This experiment successfully demonstrates the use of **stack** data structures to implement **Undo** and **Redo** operations. It simulates how modern text editors allow users to revert and reapply changes. The use of two stacks ensures an efficient and logical approach to managing editing history.