Assignment On

‘ADVANCED DATA STRUCTURES AND ALGORITHMS’

(Assignment-3)

Submitted by: Suganth C J
Roll number:2503B09901
(MCA).

Submitted to: Dr. Rahul Mishra.

Ouestion: Create an Undo-Redo System Using Two Stacks. Implement Push, Pop, and Display Operations.

Code:

```
import sys
# Set recursion limit higher for safety, although not strictly needed for this stack implementation
sys.setrecursionlimit(2000)

class UndoRedoSystem:
    """
    Implements an Undo-Redo system using two stacks:
    1. undo_stack: Holds the current sequence of actions (history).
    2. redo_stack: Holds actions that have been undone.
    """
    def __init__(self):
        # The main stack representing the current state/history
        self.undo_stack = []
        # The stack holding undone actions for potential re-application
        self.redo_stack = []

    def push(self, action: str):
```

```
    """
    Performs a new action. Pushes the action onto the undo stack
    and clears the redo stack (as new actions invalidate future redo history).
    """
    self.undo_stack.append(action)
    # Clearing the redo stack is crucial! Any new action breaks the redo sequence.
    self.redo_stack.clear()
    self.display_state(f"Action '{action}' performed.")

def undo(self):
    """
    Undoes the last action. Moves the top item from the undo stack
    to the redo stack.
    """
    if not self.undo_stack:
        self.display_state("Cannot undo. History is empty.")
        return

    # Move the last action to the redo stack
    last_action = self.undo_stack.pop()
    self.redo_stack.append(last_action)
    self.display_state(f"Action '{last_action}' undone.")

def redo(self):
    """
    Redoes the last undone action. Moves the top item from the redo stack
    back to the undo stack.
    """
    if not self.redo_stack:
        self.display_state("Cannot redo. No actions have been undone.")
        return
```

```
        # Move the undone action back to the undo stack
        undone_action = self.redo_stack.pop()
        self.undo_stack.append(undone_action)
        self.display_state(f"Action '{undone_action}' redone.")

    def display_state(self, operation: str):
        """
        Prints a simplified view of the current state.
        """
        print(f"\n--- {operation} ---")
        print(f"Undo History: {self.undo_stack}")
        print(f"Redo Buffer:  {self.redo_stack}")
        print("-" * 30)

# --- Example Usage (Simplified) ---

if __name__ == "__main__":
    system = UndoRedoSystem()
    print("Starting Simplified Undo/Redo Demo")

    # 1. Perform initial actions (Push)
    system.push("A")
    system.push("B")
    system.push("C")

    # 2. Undo actions
    system.undo()

    # 3. Redo an action
    system.redo()
```

Output:

```
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 3> & C:/Users/sugan/AppData/Local/Programs/Python/Python3
14/python.exe "c:/Users/sugan/OneDrive/Desktop/MCA/ADSA/Assignment 3/Task_1.py"
Starting Simplified Undo/Redo Demo

--- Action 'A' performed. ---
Undo History: ['A']
Redo Buffer:  []
------------------------------

--- Action 'B' performed. ---
Undo History: ['A', 'B']
Redo Buffer:  []
------------------------------

--- Action 'C' performed. ---
Undo History: ['A', 'B', 'C']
Redo Buffer:  []
------------------------------

--- Action 'C' undone. ---
Undo History: ['A', 'B']
Redo Buffer:  ['C']
------------------------------

--- Action 'C' redone. ---
Undo History: ['A', 'B', 'C']
Redo Buffer:  []
------------------------------
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 3>
```