

# Undo–Redo Mechanism using Stack in C

**Name:** Akanksha Vinayak Chavan

**College:** SPPU,pune

**Department:** Mathematics

## **Objective:**

To implement Undo–Redo mechanism using two stacks in C language demonstrating data structure operations such as push and pop.

## **Theory / Explanation:**

The Undo–Redo mechanism is commonly used in applications like text editors or drawing tools to reverse or reapply user actions. This can be implemented efficiently using two stacks: an Undo stack and a Redo stack. - The Undo stack stores actions performed by the user. - When the user performs an Undo, the top element of the Undo stack is popped and pushed into the Redo stack. When the user performs a Redo, the top element of the Redo stack is popped and pushed back into the Undo stack. This stack-based mechanism helps maintain the correct order of user actions and supports multiple undo/redo operations.

## **Algorithm:**

Step 1: Initialize two stacks - Undo and Redo as empty.

Step 2: When a new action is performed:

- Push the action onto Undo stack.

- Clear the Redo stack

- . Step 3: When Undo is performed:

- Pop the top element from Undo stack.

- Push it onto Redo stack.

Step 4: When Redo is performed:

- Pop the top element from Redo stack.

- Push it onto Undo stack.

Step 5: Repeat as per user's choice.

## **Code:-**

```
#include <stdio.h>
#include <string.h>
#define MAX 10
char undoStack[MAX][50];
char redoStack[MAX][50];
int topUndo = -1;
int topRedo = -1;
void performAction(char action[]);
void undo();
void redo();
void displayStacks();
int main() {
    int choice;
    char action[50];
    while (1) {
        printf("\n--- Undo-Redo Menu ---\n");
        printf("1. Perform Action\n2. Undo\n3. Redo\n4. Display Stacks\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();
        switch (choice) {
            case 1:
                printf("Enter action: ");
                fgets(action, sizeof(action), stdin);
                action[strcspn(action, "\n")]=0;
                performAction(action);
```

```
break;

case 2: undo(); break;

case 3: redo(); break;

case 4: displayStacks(); break;

case 5: return 0;

default: printf("Invalid choice!\n");

}

}

}

void performAction(char action[]) {

if (topUndo < MAX - 1) {

strcpy(undoStack[++topUndo], action);

topRedo = -1;

printf("Action performed: %s\n", action);

} else printf("Undo stack full!\n");

}

void undo() {

if (topUndo >= 0) {

strcpy(redoStack[++topRedo], undoStack[topUndo]);

printf("Undo performed: %s\n", undoStack[topUndo--]);

} else printf("Nothing to Undo!\n");

}

void redo() {

if (topRedo >= 0) {

strcpy(undoStack[++topUndo], redoStack[topRedo]);

printf("Redo performed: %s\n", redoStack[topRedo--]);

} else printf("Nothing to Redo!\n");

}
```

```
void displayStacks() {  
    int i;  
  
    printf("\nUndo Stack:\n");  
  
    for (i = topUndo; i >= 0; i--) printf("%s\n", undoStack[i]);  
  
    if (topUndo == -1) printf("(Empty)\n");  
  
    printf("\nRedo Stack:\n");  
  
    for (i = topRedo; i >= 0; i--) printf("%s\n", redoStack[i]);  
  
    if (topRedo == -1) printf("(Empty)\n");  
}
```

## Sample Output (Terminal View):

--- Undo-Redo Menu ---

1. Perform Action
2. Undo
3. Redo
4. Display Stacks
5. Exit

Enter your choice: 1

Enter action: Type A

Action performed: Type A

Enter your choice: 1

Enter action: Type B

Action performed: Type B

Enter your choice: 2

Undo performed: Type B

Enter your choice: 3

Redo performed: Type B

Enter your choice: 4

Undo Stack:

Type B

Type A

Redo Stack:

(Empty)

## **Conclusion:**

The Undo–Redo mechanism was successfully implemented using two stacks in C. This demonstrates how stack operations (push and pop) can be effectively used to manage user actions in real-world applications