



DSA Project Report : Using a Stack

Submitted by:

P Sai Leela (24PG00080)

Jathin D (24PG00073)

Manoj S M (24PG00018)

Sumanth (24PG00071)

Under the Guidance Of

Prof. Usha Subramanian

Course: MCA & MSC Data science

1st semester 2024 – 2025

School of Engineering

Table of contents

S.NO	CONTENTS	PAGE NO
1.	Background	3
2.	Problem statement	4
3.	System design	5
4.	Implementation	6 - 7
5.	Operation explanation	8 - 10
6.	Conclusion	11
7.	Annexure	12 - 13
8.	References	14

1. Background

A document management system allows users to modify text while ensuring they can revert or reapply changes as needed. Two essential operations in such systems are undo and redo, which enable users to remove the last entered character or restore the most recently deleted character, respectively. These operations help maintain accuracy, prevent accidental loss of data, and provide flexibility in editing.

Stacks, a Last-In, First-Out (LIFO) data structure, are well-suited for handling undo and redo operations. When a user types a character, it is pushed onto a stack, allowing the system to track the sequence of changes. Undo operations remove the last character and store it in a separate stack, enabling redo operations to retrieve and reapply it when needed. This structured approach ensures efficient and reliable text modification, mimicking real-world text editors.

These operations can occur in any sequence, ensuring that the document maintains its correct state at all times. The goal of this project is to implement these functionalities using a stack-based approach, a data structure well-suited for handling last-in, first-out (LIFO) operations.

By implementing undo and redo functionalities, this project simulates real-world text editing behavior, helping users efficiently manage changes to a document while reinforcing fundamental data structure concepts.

2. Problem Statement

The goal of this project is to implement a simple text editing system that supports undo and redo functionalities using a stack-based approach. The system should allow users to type characters, undo recent changes, and redo previously undone modifications. The implementation should maintain the correct sequence of text changes and ensure that multiple undo and redo operations can be performed efficiently.

Typing Operation : reading a line from the document.

Undo Operation : Removes the last entered character and stores it separately in a stack.

Redo Operation : Restores the most recently undone character if an undo operation was previously performed.

This system will use stack:

Undo - removes last character from the line and stores the character into the stack by using push operation

Redo - adds last character to the line and removes the character from the stack by using pop operation

The system should display the text after each operation and keep track of the number of undo and redo actions performed. Since stacks follow the Last-In, First-Out (LIFO) principle, this approach ensures that operations are executed efficiently.

3. System Design

The system consists of three main components:

1. Input Handling: read the line from the text file then , undo the last entered character, or redo a previously undone character.
2. Data Structure Management: The system utilizes stack with push and pop operation.
3. Output Display: The system displays the text after each operation to reflect the current state of the document.

Workflow: First we will read the file which contains actions to be performed. By the reference of the actions we will perform the type(reading a line from the file),undo and redo operations on the text.

4. Implementation

Data Structure Selection: Stacks are implemented using a linked list to allow dynamic memory allocation.

We use a stack data structure with push and pop operations which are undo and redo.

Implementation

1. Typing (type_)

Reading a line from the txt file.

2. Undo (undo method)

Removes the last character from the line.

Pushes it onto the Stack .

3. Redo (redo method)

Pops the last character from the Stack.

Restores the character to the line.

Complexity Analysis

Time Complexity:

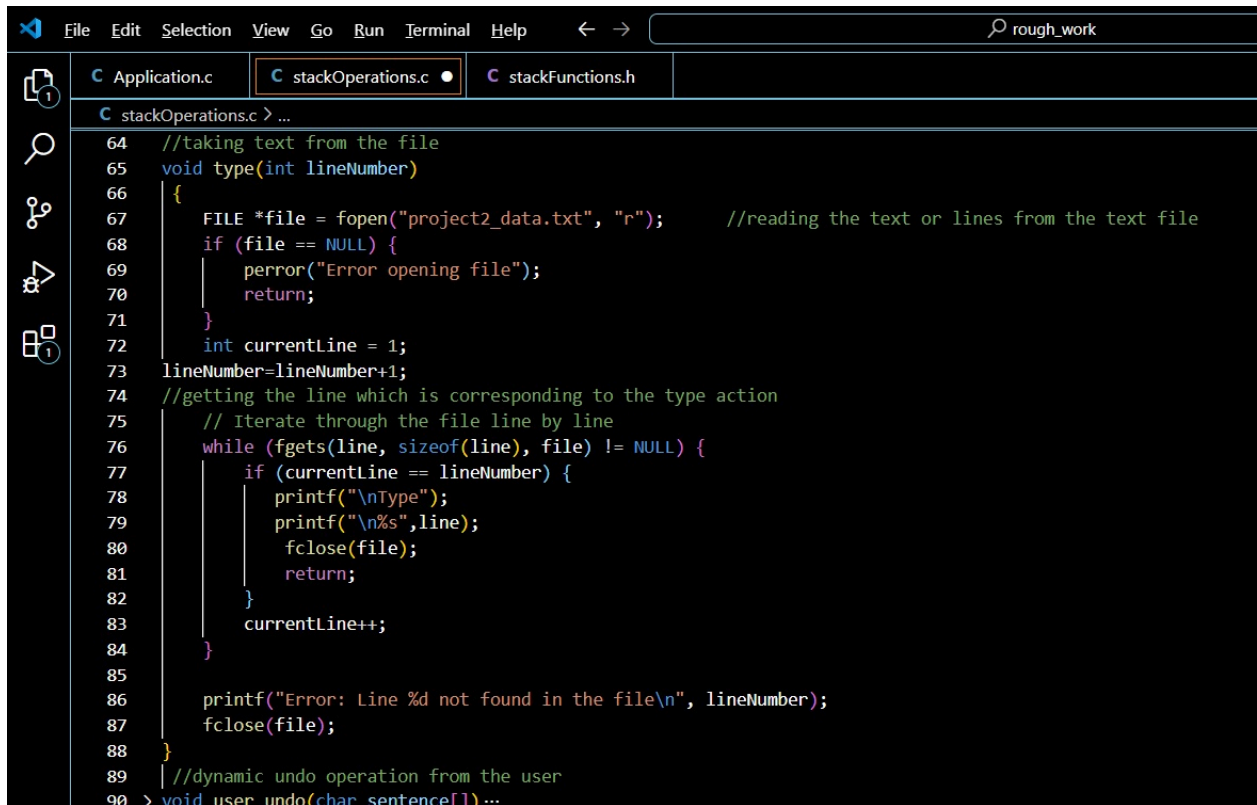
`push()` → $O(1)$

`pop()` → $O(1)$

Space Complexity: $O(N)$, where N is the number of characters stored.

- **`push()` → $O(1)$:** The push operation adds an element to the top of the stack. This happens in constant time because no matter how many elements are in the stack, the operation simply places the new element on top.
- **`pop()` → $O(1)$:** The pop operation removes the top element of the stack. This is also done in constant time, as it doesn't depend on the number of elements; it's just removing the element from the top.
- **Space Complexity: $O(N)$:** The space complexity refers to the amount of memory used to store the stack. If you're storing N elements, the space required is proportional to N .

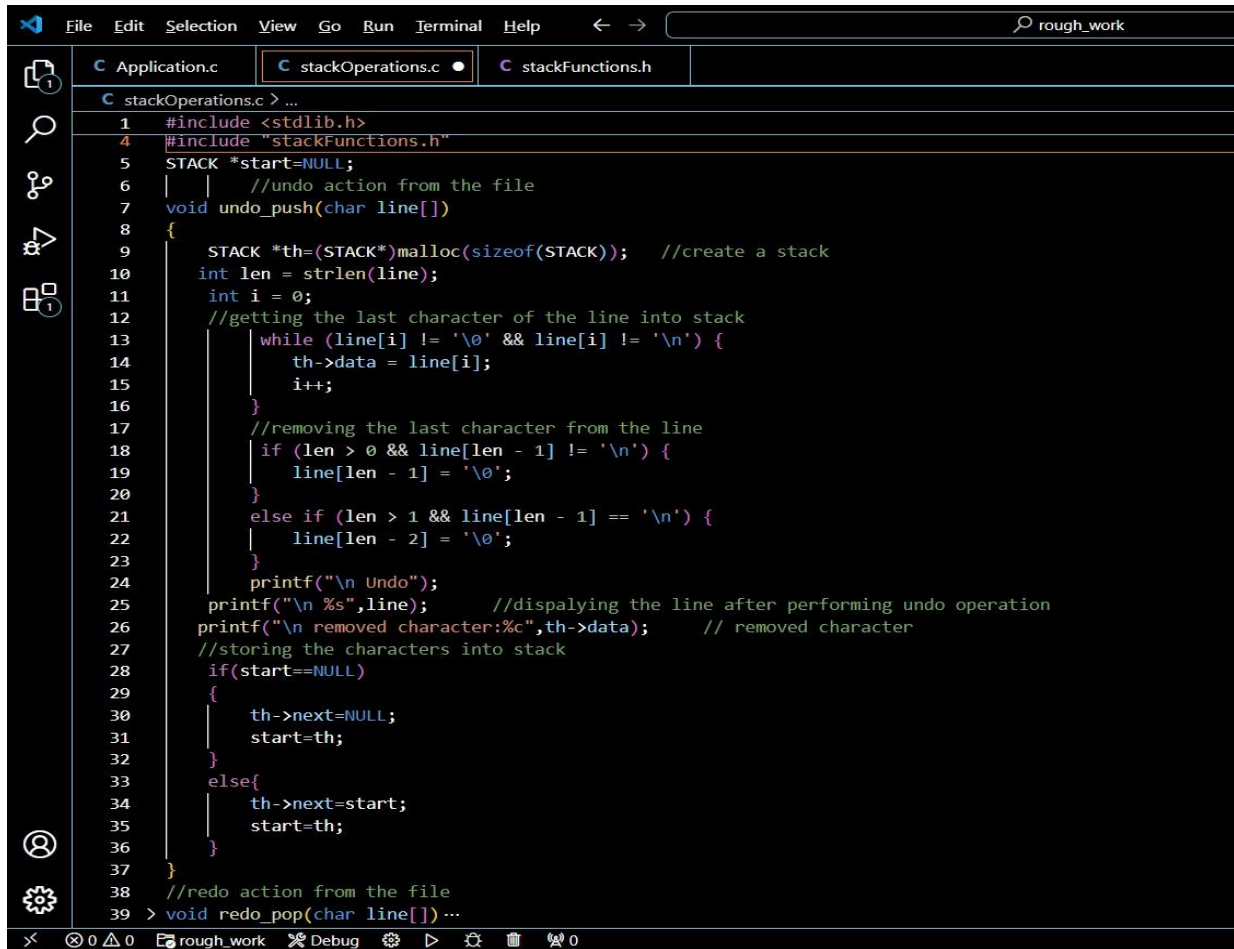
5. Operation explanation



```
64 //taking text from the file
65 void type(int lineNumber)
66 {
67     FILE *file = fopen("project2_data.txt", "r");    //reading the text or lines from the text file
68     if (file == NULL) {
69         perror("Error opening file");
70         return;
71     }
72     int currentLine = 1;
73     lineNumber=lineNumber+1;
74     //getting the line which is corresponding to the type action
75     // Iterate through the file line by line
76     while (fgets(line, sizeof(line), file) != NULL) {
77         if (currentLine == lineNumber) {
78             printf("\nType");
79             printf("\n%s",line);
80             fclose(file);
81             return;
82         }
83         currentLine++;
84     }
85     printf("Error: Line %d not found in the file\n", lineNumber);
86     fclose(file);
87 }
88 //dynamic undo operation from the user
89 void user undo(char sentence[])...
```

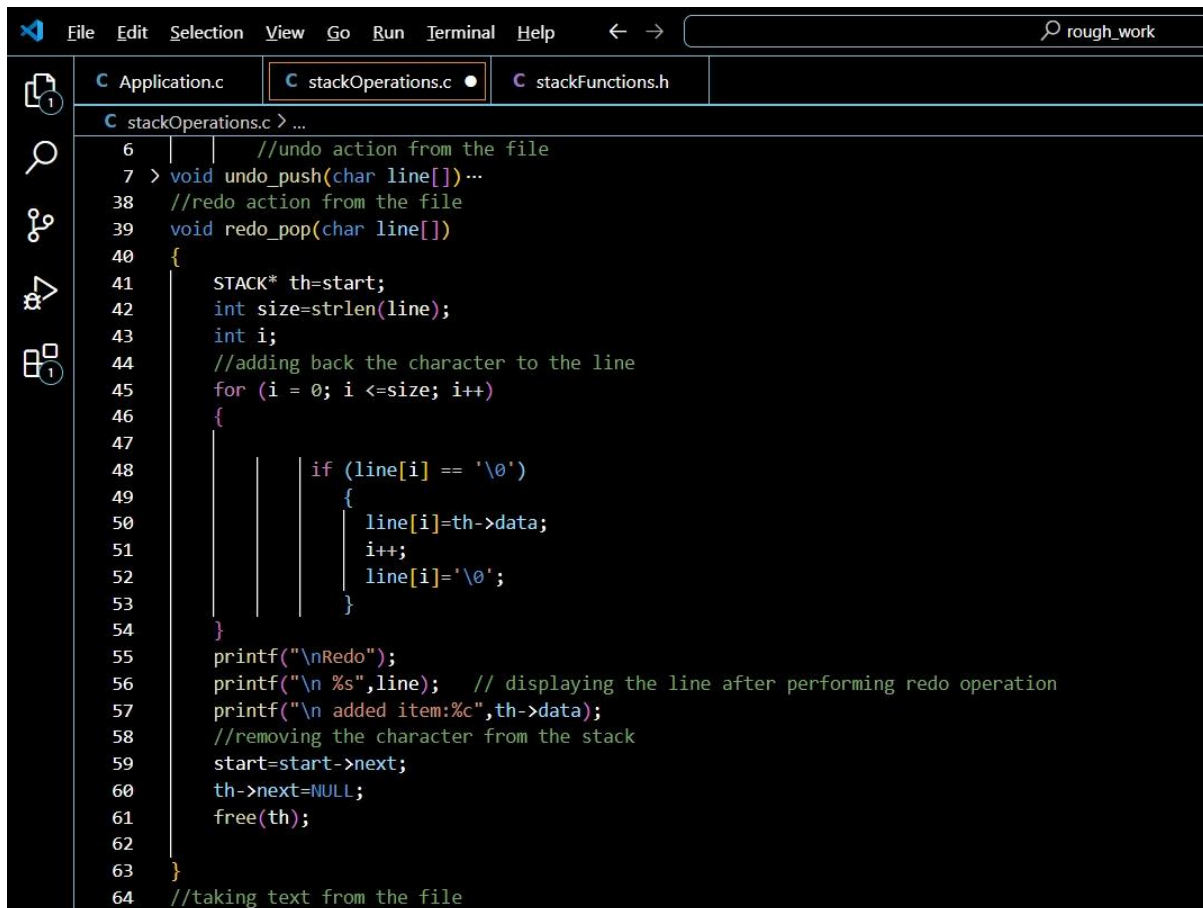
Type operation : reading the text file and performing the operations according to the actions file.

Reads the corresponding line to the type action.



```
1 #include <stdlib.h>
2 #include "stackFunctions.h"
3
4 STACK *start=NULL;
5 //undo action from the file
6 void undo_push(char line[])
7 {
8     STACK *th=(STACK*)malloc(sizeof(STACK)); //create a stack
9     int len = strlen(line);
10    int i = 0;
11    //getting the last character of the line into stack
12    while (line[i] != '\0' && line[i] != '\n') {
13        th->data = line[i];
14        i++;
15    }
16    //removing the last character from the line
17    if (len > 0 && line[len - 1] != '\n') {
18        line[len - 1] = '\0';
19    }
20    else if (len > 1 && line[len - 1] == '\n') {
21        line[len - 2] = '\0';
22    }
23    printf("\n Undo");
24    printf("\n %s",line); //displying the line after performing undo operation
25    printf("\n removed character:%c",th->data); // removed character
26    //storing the characters into stack
27    if(start==NULL)
28    {
29        th->next=NULL;
30        start=th;
31    }
32    else{
33        th->next=start;
34        start=th;
35    }
36 }
37
38 //redo action from the file
39 > void redo_pop(char line[])...
```

Undo operation : here we are creating a node of a stack ,then we traverse to the last character of the line and stores that character into the stack then removes it from the line.



```
File Edit Selection View Go Run Terminal Help ← → rough_work
C Application.c C stackOperations.c C stackFunctions.h
C stackOperations.c > ...
6 //undo action from the file
7 > void undo_push(char line[])...
38 //redo action from the file
39 void redo_pop(char line[])
40 {
41     STACK* th=start;
42     int size=strlen(line);
43     int i;
44     //adding back the character to the line
45     for (i = 0; i <=size; i++)
46     {
47         if (line[i] == '\0')
48         {
49             line[i]=th->data;
50             i++;
51             line[i]='\0';
52         }
53     }
54     printf("\nRedo");
55     printf("\n %s",line); // displaying the line after performing redo operation
56     printf("\n added item:%c",th->data);
57     //removing the character from the stack
58     start=start->next;
59     th->next=NULL;
60     free(th);
61 }
62 //taking text from the file
```

Redo operation : first we traverse into last character of the line ,then we pop out the top character from the stack and add it back to the end of the line.

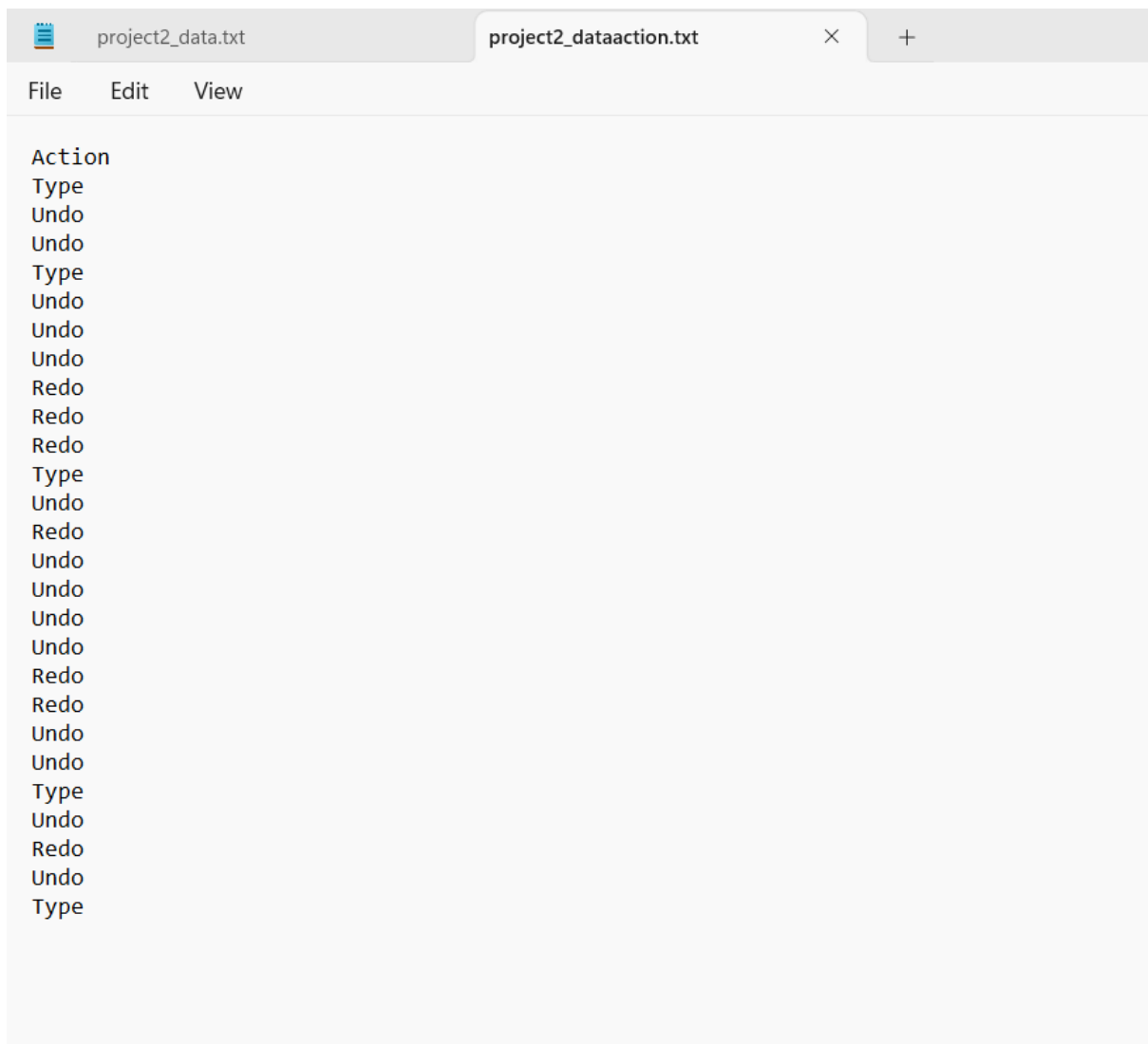
6. Conclusion

We can dynamically take text file from the user and perform undo and redo operations on it and each operation resulted in the correct state of the document as it existed before the undo and redo operations. Here undo operation deleted the last character in the line and redo operation added the last character back to the line.

This system ensures that every undo operation can be reversed by the redo operation and vice versa, allowing for a flexible and consistent document editing experience. The system should efficiently manage the states using stacks to support both operations without limits on the number of actions.

7. Annexure

Annexure 1: we are taking file which contains the actions as an input file.



Annexure 2: We are taking a text file as an input file.

[illegible]

8. References

Chat gpt :

<https://chatgpt.com/#:~:text=simple%20C%20program-,that,-reads%20a%20specific.>

We typed: (c program to read a specific line from the text file)
Pdf of the sessions which mam shared.