# STACK DATA STRUCTURE FOR TEXT EDITOR

**A PROJECT REPORT**

***Submitted by***

**D.SADWIKA REDDY [RA2211003011072]**
**K.BHARATH ROYAL [RA2211003011082]**
**T.SANSKAR [RA2211003011130]**
**MD.SHAIK TABISH [RA2211003011087]**

***for the course 21CSC201J Data Structures and Algorithms***

***Under the Guidance of***

**Dr. Indhumathi Raman**

**Professor, Department of Computing Technologies**

***In partial satisfaction of the requirements for the degree of***

**BACHELOR OF TECHNOLOGY**
**in**
**COMPUTER SCIENCE ENGINEERING**

**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603203**

**NOVEMBER 2023**

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

## BONAFIDE CERTIFICATE

Certified that the 21CSC201J Data Structures and Algorithms course project report titled "STACK DATA STRUCTURE FOR TEXT EDITOR" is the bonafide work done by ***D.SADWIKA REDDY [RA2211003011072], K.BHARATH ROYAL[RA2211003011082], T.SANSKAR[RA221100301130], MD.SHAIK TABISH [RA2211003011087]*** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

**Dr. Indhumathi Raman**,
Professor, Faculty In-Charge,
Department of Computing Technologies,
SRMIST.

**Dr. M. Pushpalatha,**
Professor and Head,
Department of Computing Technologies,
SRMIST.

## PROBLEM DEFINITION

The problem at hand involves the development of a fundamental text editor using the C programming language. This text editor is intended to offer users a range of capabilities, including the ability to input and modify text, perform undo and redo actions, save their ongoing work to a file, and load text from an external file source. The primary aim of this project is to provide a simplified yet fully functional tool for text editing, shedding light on the effective use of stack data structures to enable the critical undo and redo operations.

In essence, this project aims to create a basic but practical text editor in C, equipping users with essential functionalities for working with text. It offers a platform where users can input and manipulate text, track changes through the undo and redo mechanisms, preserve their text in files, and recover text from previously saved files. The core focus is on showcasing how stack data structures are harnessed to facilitate the vital undo and redo operations, allowing users to manage their editing history efficiently.

## PROBLEM EXPLANATION

**Structures:** The editor uses structures, specifically TextBuffer and TextEditor, to manage text and undo/redo functionality.

**Undo/Redo Stacks:** Undo and redo operations are implemented using arrays of TextBuffer with a limit of 50 operations each.

**Core Functionalities:** The editor provides key functionalities such as makeChange, undo, redo, saveToFile, and loadFromFile.

**MakeChange:** This function saves the current text to the undo stack, updates the TextBuffer with the new text, and resets the redo stack.

**Undo:** The undo function reverts the text using the undo stack and updates the redo stack.

**Redo:** The redo function restores undone text using the redo stack and updates the undo stack.

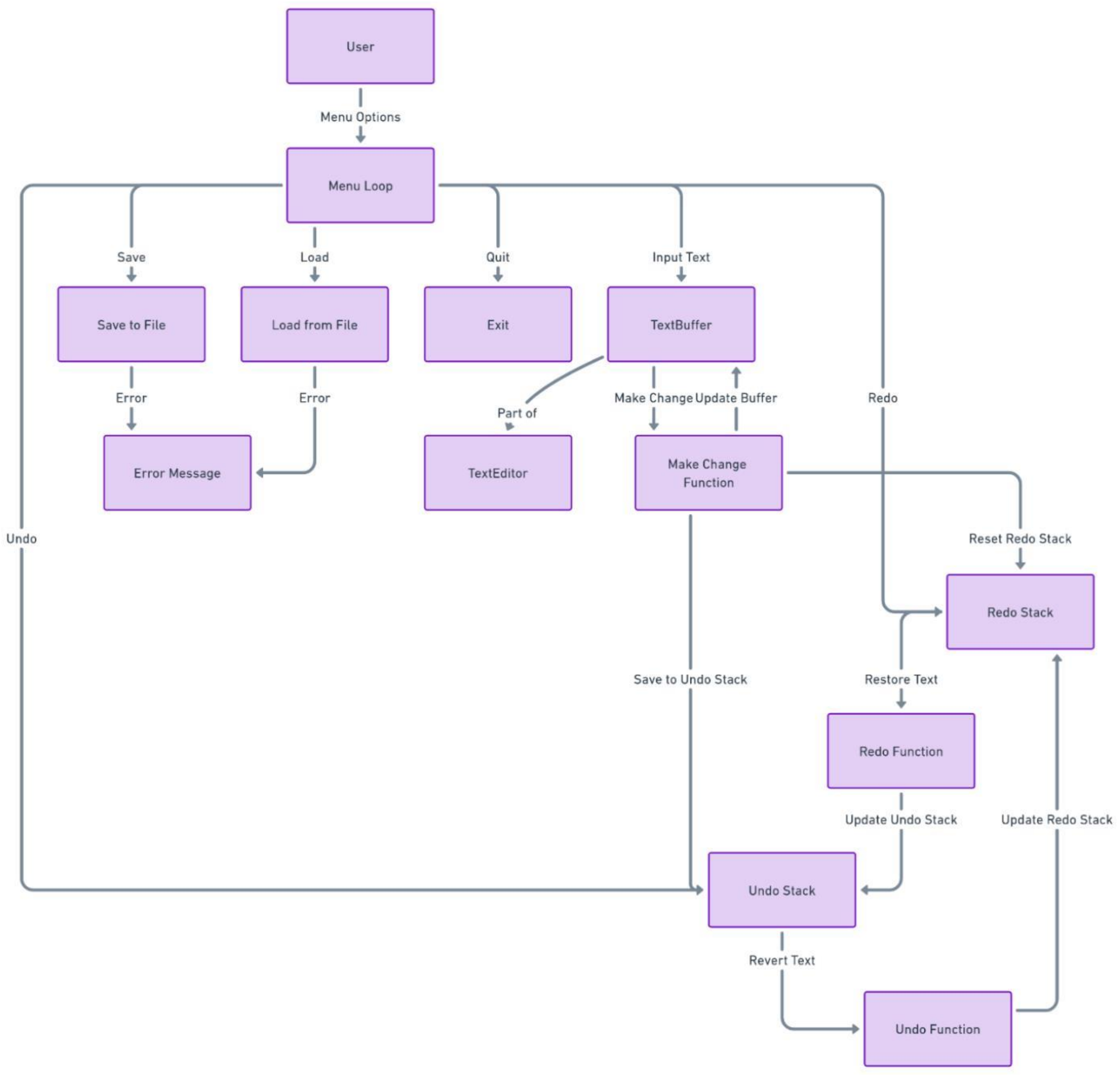
**File Operations:** Users can save text to files and load text from files using saveToFile and loadFromFile functions.

**User Interface:** The editor features a menu loop that allows users to interact with the text, perform undo/redo actions, save/load files, or exit the application.

**Error Handling:** The editor displays error messages for issues related to file operations and oversized files to provide feedback to the user.

**Quitting:** Users can exit the editor by selecting the "Quit" option in the menu loop when they

## DIAGRAM:



**DATA STRUCTURE:**

A stack is a fundamental data structure in computer science and programming that follows the Last-In, First-Out (LIFO) principle. It is used to store and manage a collection of elements, where the most recently added item is the first to be removed. Think of it like a stack of books, where you can only add or remove books from the top of the stack.

Key operations associated with a stack data structure are:

1. **Push:** This operation is used to add an element to the top of the stack.

2. **Pop:** This operation removes the element from the top of the stack.

3. **Peek (or Top):** This operation allows you to look at the element on the top of the stack without removing it.

**Applications of Stack Explained**

1. CD/DVD stand.
2. Stack of books in a book shop.
3. Call center systems.
4. Undo and Redo mechanism in text editors.
5. The history of a web browser is stored in the form of a stack.
6. Call logs, E-mails, and Google photos in any gallery are also stored in form of a stack.
7. YouTube downloads and Notifications are also shown in LIFO format(the latest appears first )

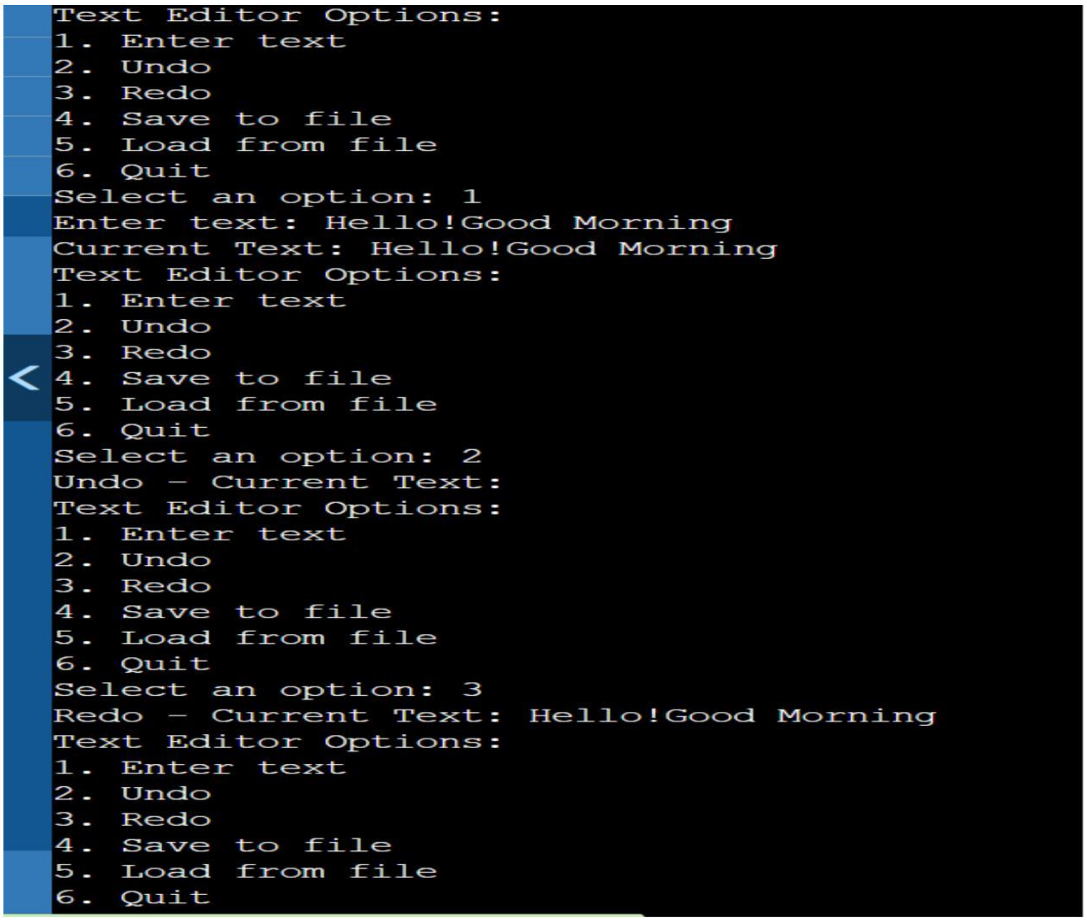
**CODE:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_TEXT_SIZE 10000

#define STACK_SIZE 50


typedef struct {

    char *text;  // Use dynamic memory allocation

} TextBuffer;


typedef struct {

    TextBuffer buffer;

    TextBuffer undo_stack[STACK_SIZE];

    TextBuffer redo_stack[STACK_SIZE];

    int undo_top;

    int redo_top;

} TextEditor;
```

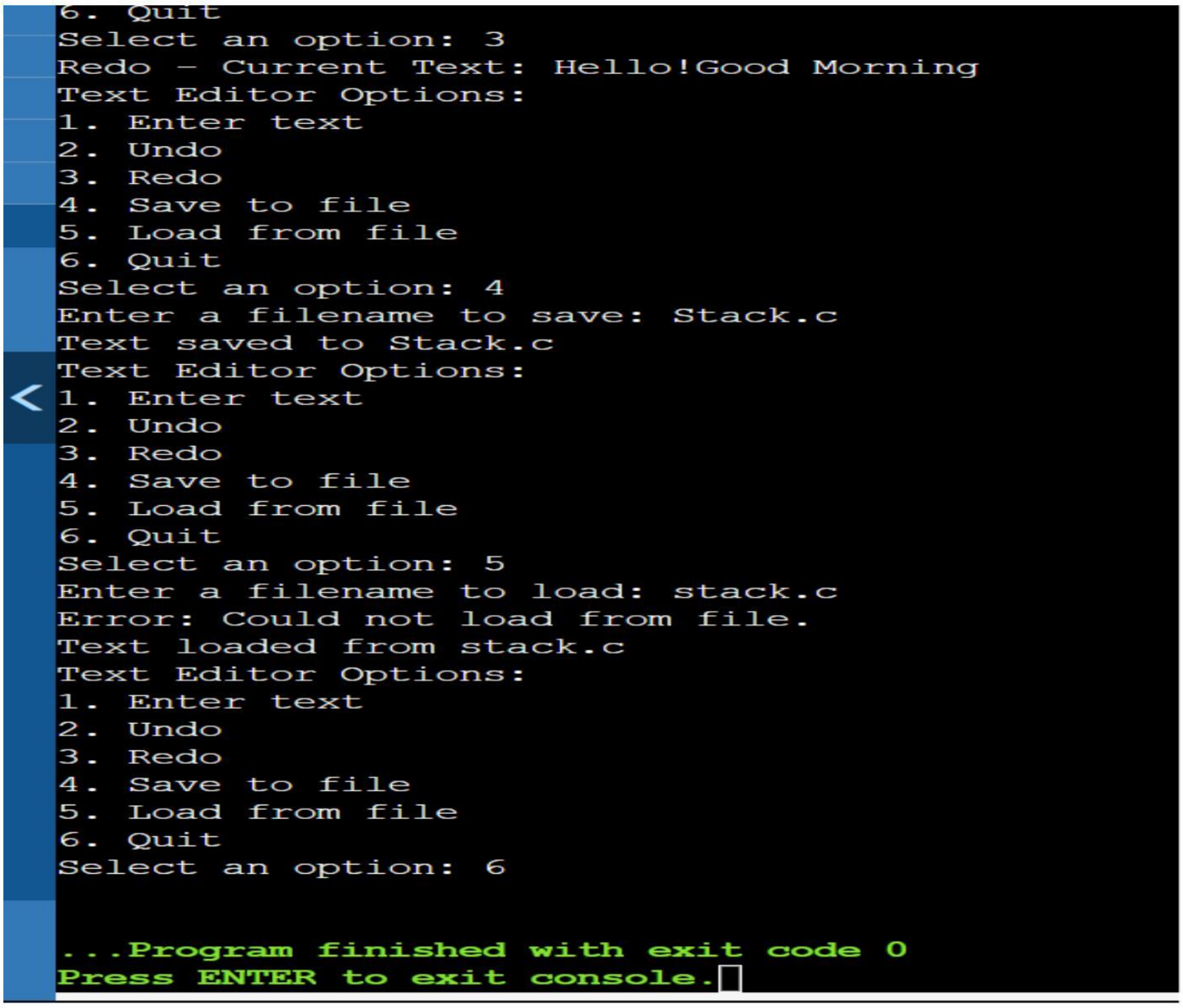
```
void initTextEditor(TextEditor *editor) {
    editor->undo_top = -1;
    editor->redo_top = -1;
    editor->buffer.text = (char *)malloc(MAX_TEXT_SIZE); // Allocate memory for the text buffer
    editor->buffer.text[0] = '\0'; // Initialize text buffer to an empty string
}

void makeChange(TextEditor *editor, const char *new_text) {
    if (editor->undo_top < STACK_SIZE - 1) {
        editor->undo_top++;
        strcpy(editor->undo_stack[editor->undo_top].text, editor->buffer.text);
        strcpy(editor->buffer.text, new_text);
        editor->redo_top = -1;
    }
}

// ... (rest of the code remains the same)
```

```
int main() {
    TextEditor editor;
    initTextEditor(&editor);
    char input[MAX_TEXT_SIZE];

    while (1) {
        // ... (rest of the code remains the same)
    }

    // Free dynamically allocated memory
    free(editor.buffer.text);
      return 0;
}
```

**OUTPUT:**



```
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 1
Enter text: Hello!Good Morning
Current Text: Hello!Good Morning
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 2
Undo - Current Text:
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 3
Redo - Current Text: Hello!Good Morning
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
```



```
6. Quit
Select an option: 3
Redo - Current Text: Hello!Good Morning
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 4
Enter a filename to save: Stack.c
Text saved to Stack.c
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 5
Enter a filename to load: stack.c
Error: Could not load from file.
Text loaded from stack.c
Text Editor Options:
1. Enter text
2. Undo
3. Redo
4. Save to file
5. Load from file
6. Quit
Select an option: 6


...Program finished with exit code 0
Press ENTER to exit console.
```

## CONCLUSION:

The code outputs a simple text editor that can be used to interact with text. Users can input text, undo and redo changes, save to and load from files, and finally quit the program. The time complexity of undo and redo operations depends on the size of the text being managed, but it is generally linear with respect to the number of characters in the text. The file I/O operations have their own time complexity, typically dependent on the file size and storage media. In this simplified implementation, the undo and redo stacks are limited to 50 operations each, which can be easily adjusted according to your requirements. The use of stacks ensures efficient undo and redo functionality with a space complexity of O(50) in this case. Overall, while this code provides a basic text editor with undo and redo functionality, real-world text editors are far more complex and optimized for performance, handling larger text files, and providing a wide range of features.