# Assignment 1: Text Editor

## Introduction

In the world of software development, passionate debates often emerge over the choice of text editors, believe it or not. Whether it's the minimalist appeal of Vim and Emacs or the modern, feature-rich environments of Visual Studio Code and Sublime Text, each has its own set of staunch supporters. Even though many text editors look different and provide unique features, at their core, they all share fundamental functionalities that are pivotal for text manipulation; namely, the abilities to edit text, and to perform undo and redo operations.

## Objective

In this assignment, you will be implementing a simplified text editor in C. This text editor will focus on essential functionalities such as inserting and deleting text, as well as undoing recent changes. This will be your first exercise in which you have to critically analyse the problem at hand, and decide which data structure(s) suit the problem description best.

## Assignment Details

As mentioned before, your task is to figure out which approach is the best to solve this exercise, and implement said approach. You do this by making use of one of the provided data structure implementations. Please download `texteditor.zip` from Themis to get started with the assignment. You should only modify the files `TextEditor.c` and `TextEditor.h`, and replace the comments with the corresponding code.

We have provided you with a `Makefile` file that you can use to compile, run, and test your code with. For more information on how to use `Makefiles`, please read through the documentation.

### Core Functionalities

Your text editor must support the following operations:

1. **Insertion**: Insert a single lowercase letter at a specified position in the text.
2. **Deletion**: Delete the letter at a specified position.
3. **Undo**: Revert the last insertion or deletion operation.
4. **Output**: After processing a series of operations, output the final text state or `EMPTY` if the text is empty.

## Input and Output Format

**Input**: Operations are provided line by line with the following notations:

- Insertions: `i` followed by the position and the character.
- Deletions: `d` followed by the position.
- Undo operations: `u`.
- The sequence ends with `q`.

**Output**: Display the final state of the text on a single line. Print `EMPTY` if the text is empty. Make sure you end the output with a newline character `\n`.

## Example

**Input**:

```
i 0 h
i 1 e
i 2 l
i 3 l
u
i 3 o
d 1
q
```

**Output**:

```
hlo
```

### Walk-through

The example operates on an initially empty text and processes the specified operations. If we look at each operation, and what the new value of the text is after performing that operation, we get the following table:

| Operation | Text |
|---|---|
| `i 0 h` | `h` |
| `i 1 e` | `he` |
| `i 2 l` | `hel` |
| `i 3 l` | `hell` |
| `u` | `hel` |
| `i 3 o` | `helo` |
| `d 1` | `hlo` |
| `q` | `hlo` |

At the end of these operations, the text editor contains the word `hlo`, which is what should be printed to `stdout`.

## Assumptions

- The input text consists solely of lowercase letters.
- Insertions and deletions are provided with valid indices. There will be no negative indices or indices beyond the current text length.
- If an undo operation is called when there are no previous operations, the operation should be ignored.

## Submission details

You can submit your program on Themis, as you are used to by now. Note that you should **not** upload the following files to Themis, as they are already being included in your submission (and hence shouldn't have been modified by you): `EditOperation.h`, `LibList.c`, `LibList.h`, `LibQueue.c`, `LibQueue.h`, `LibStack.c`, and `LibStack.h`. These files are automatically inserted for you in your submission. Each test case has a hint in case you get stuck. Note, by the way, that we will run Valgrind on each test case, so make sure that you handle memory management properly!

# Bonus Challenge: Implementing Redo Functionality (+1 Bonus Point)

Elevate your text editor by adding redo functionality, enabling users to reapply changes they've undone. This addition not only enhances usability but also tests your ability to manage application state efficiently. There is a separate submission section on Themis for the bonus assignment. The principle is the same as before, but now you have one more command `r` to take into account.

## Example

**Input**:

```
i 0 h
i 1 e
i 2 l
i 3 l
u
u
r
i 3 o
d 1
q
```

**Output**:

```
hlo
```

Note that you should only be able to redo an operation that you have previously undone. If there are no previous undo actions to redo, calling the `r` command should simply be ignored.