CS2001 – Data Structures

Assignment 04: Console-Based Notepad with AVL Tree Dictionary

Deadline: 17th November

Objective:

Create a console-based notepad application with spell-checking and dictionary support using an AVL tree. Each **letter** in a word will be stored as an individual node in a linked list, forming words. When a space is entered, the last completed word will be spell-checked using a stack or queue.

Requirements:

1. **AVL Tree Dictionary**:
    a. Load a provided text file containing words in alphabetical order.
    b. Insert each word into an AVL tree to ensure balanced storage, preventing degeneration and enabling efficient word searches.
    c. The AVL tree should support balanced insertion and efficient search operations.
2. **Console-Based Notepad Functionality**:
    a. Implement a notepad where each **letter** typed by the user is stored as a separate node in a linked list.
    b. Use any `library for handling key presses to support functionalities like Backspace, Ctrl + L, Ctrl + S, and Esc (or any alternatives).`

Notepad Features:

c. **Text Addition**: Each letter typed by the user is added as a new node in the linked list, allowing flexible editing. As the text is updated, the screen is displayed again with the updated text. *(Hint: clear screen after every update and display updated screen)*
d. **Text Deletion**: Use the Backspace key to delete the last letter typed.
e. **Text Loading**: Use "Ctrl + L" to load text from a file into the notepad.

f. **Text Saving**: Use "Ctrl + S" to save the current text to a file named `save.txt`.
g. **Quit**: Use the "Esc" key to exit the application, releasing all memory occupied by the linked list and AVL tree.

3. **Word Tracking and Spell Checker**:
    a. As the user types, each letter is stored in the linked list and simultaneously pushed onto another data structure (stack, queue, etc), forming the most recent word.
    b. Once a space is entered, combine the characters from the stack/queue to form the full word.
    c. Check this word against the AVL tree dictionary to determine if it is correctly spelled.
    d. If the word is not found in the dictionary, display suggestions based on common spelling modifications. Modify the word according to the following techniques and search for each modified word in the dictioanry. If it is found as an existing word, display it as a modification. Display minimum one modification from each technique.

Spell-Check Modification Techniques:

e. **Letter Substitution**: go over all the characters in the misspelled word and try to replace a character with any other character. In this case, if there are k characters in a word, the number of modifications to try is 25*k. *For example, in a misspelled word 'lat', substituting 'c' instead of 'l' will produce a word 'cat', which is in the dictionary*.
f. **Letter Omission**: try to omit (in turn, one by one) a single character in the misspelled word and see if that gets you a word in the dictionary. In this case, there are k modifications to try where k is the number of characters in the word. *For example, if the misspelled word is 'catt', omitting the last character 't' will produce the word 'cat', which is in the dictionary*.
g. **Letter Insertion**: try to insert a letter in the misspelled word. If a word is k characters long, there are 26*(k+1) modifications to try, since there are 26 characters to insert and k+1 places (including in the beginning and at the end of the word) to insert a character. *For example, for 'plce', inserting 'a' after 'p' produces a correctly spelled word 'place'*.
h. **Letter Reversal**: Try swapping every 2 adjacent characters. For a word of length k, there are k - 1 pairs to swap. *For example, in 'paernt', swapping 'e' and 'r' produces a correctly spelled word 'parent'*.

4. **Data Structures**:
    a. **Linked List**: Store the text in the notepad, with each letter as a node.
    b. **Stack/Queue etc for Word Formation**: Track the letters of the most recent word in a stack or queue, allowing easy access when spell-checking is triggered.
    c. **AVL Tree**: Store dictionary words to support fast lookups and suggest corrections.
5. **Memory Management**:
    a. Ensure all dynamically allocated memory (nodes in linked list, AVL tree, stack/queue) is released on exiting the program.

Example Commands and Usage:

1. **Typing**: Each letter typed is added as a node in the linked list. A stack or queue also tracks each word in progress.
2. **Backspace**: Deletes the last letter typed.
3. **Space**: After a word is completed with a space, the word is spell-checked against the AVL dictionary.
4. **Ctrl + L**: Loads text from a file.
5. **Ctrl + S**: Saves the text to `save.txt`.
6. **Esc**: Exits the notepad and releases memory.

Submission Information:

Submit a `.zip` file named in the format

`[RollNo]_[section].zip` (e.g., `22i2036_DSC.zip`), containing all source files and documentation.

Marking Rubric:

Task		Marks
Data Structures		**60**
	Correct structure definitions for AVL tree and linked list	30
	AVL insertion with balancing	20
	Other basic functionalities of both structures	10
Notepad Functionality		**30**
	Text addition (enter letter)	10
	Text Deletion (backspace)	5
	Text loading (ctrl + L)	5
	Text Saving (ctrl + S)	5
	Quit (esc)	5
Spell Checker		**40**
	Letter Substitution	10
	Letter Omission	10
	Letter Insertion	10
	Letter Reversal	10
Total		**130**

+10 for any extra creative functionality.