Femto Text Editor – Final Project Report

1. Team Members

1. Hrujul Mendhe (B24CM1077)
2. Priyam Patel (B24CS1058)
3. Arin Nain (B24EE1096)

2. Abstract

This work presents *Femto Text Editor*, a lightweight console-based text processing tool implemented in C++. The system provides essential editing functions including line insertion, deletion, and modification, along with undo support and inter-file copy–paste operations. Additional capabilities include file statistics, similarity measurement, and efficient word search with rolling-hash-based find and replace. The editor supports word deletion, recent-file tracking, autosave on abnormal termination, and compression for large text files. All features are implemented using standard data structures and C++ libraries.

3. Project Objectives

1. Provide editing capabilities including line insertion, modification, and deletion.
2. Support undo operations and a cross-file copy–paste buffer.
3. Generate file statistics such as word and character counts.
4. Implement a file similarity checker for comparative analysis.
5. Enable search, find, and rolling-hash-based replace operations.

6. Support compression techniques for handling large text files.
7. Maintain a history of recently accessed files for the current session.
8. Ensure data preservation through autosave on unexpected program exit.

4. Code Structure

The system follows a modular directory structure, with include/ containing header files and src/ containing source files. The major modules are:

autosave: uses signal handling to detect abnormal termination and write a backup file
clipboard: provides copy and paste operations across files.
compression: performs Huffman-based compression and decompression for large text files.
file_ops: manages file reading, writing, and line-level editing using a linked-list buffer.
find_and_replace: applies rolling-hash techniques for word search and replace operations.
main_menu: coordinates user interaction and dispatches tasks to appropriate modules.
search: implements word search, frequent-word detection, and related utilities.
recent_files: maintains a list of the five most recently accessed files for the current session.
similarity: computes word-based similarity between two files.
stats: reports character and word statistics.
undo: supports reversing recent actions using an undo stack.

5. Data Structures Used

- Linked list: used to store the text buffer line by line, making insertion, editing, and deletion operations efficient.
- Stack: used to support the undo feature by following the LIFO principle to reverse recent actions.
- Huffman tree: used for file compression by constructing a binary tree based on character frequencies.
- Hash map: used for the file similarity checker and for counting word occurrences.
- Rolling hash (Rabin–Karp): used for efficient word search and find-and-replace operations.
- Standard C++ libraries: used for file handling, text processing, and binary encoding while avoiding non-standard headers.
- Deque: used to maintain recent files history during the current session.
- Huffman coding algorithm: used to generate optimal prefix-free binary codes for compression.

6. Team Contributions

- Hrujul: implemented file reading/opening, line edit/insert/delete, undo mechanism, and Huffman-based file compression.
- Priyam: implemented word, line, and character statistics; rolling-hash-based replace; file similarity using hash maps; recent-files history; and multi-line deletion.
- Arin: implemented copy–paste buffer operations, case-insensitive word search, most-frequent-word detection, and autosave functionality.