Data Structures & Algorithms – Theory

Project Report

Name	Student ID#
Huzefa Saifuddin	22K-5125
Ruhaan Ahmed	22K-6014
Ijlal Iqbal	22K-5034

Contents

Report on the Text Editor in C++

Introduction

The code under review is a simple text editor written in C++. The text editor provides basic file operations such as creating, reading, updating, and deleting files. It also supports undo and redo operations, copy, paste, and cut operations.

Code Structure and Functionality

The code is structured around a main `menu` function that displays a menu of operations and performs the selected operation. The operations are performed by calling other functions such as `undo`, `redo`, `copyFile`, `pasteFile`, and `cutFile`.

Undo and Redo Operations

The `undo` function undoes the last operation performed on the file by popping the last operation from the `undoStack` and pushing it to the `redoStack`. The `redo` function does the opposite: it pops the last operation from the `redoStack` and pushes it to the `undoStack`. These operations are implemented using stack data structures, which are a type of abstract data type that follows a LIFO (Last In, First Out) principle.

Copy, Paste, and Cut Operations

The `copyFile`, `pasteFile`, and `cutFile` functions perform the copy, paste, and cut operations, respectively. The `copyFile` function reads the content of a file and pushes it to the `copyStack`. The `pasteFile` function pops the content from the `copyStack` and appends it to the file. The `cutFile` function removes all content from the file. These operations are implemented using file I/O operations, which allow the program to read from and write to files.

HashMap

The `unordered_map` in this code, named `wordLineMap`, is a key-value data structure that stores each word as a key and a vector of line numbers where the word is found as the value. This map is used to facilitate the search feature in the program.

The `unordered_map` works on the principle of hashing, which is a mechanism of assigning a unique code to a variable or attribute using an algorithm to enable easy. In the context of `unordered_map`, the hash function is used to compute an index into an array of buckets or slots, from which the desired value can be found.

When a word is added to the `unordered_map`, the word's hash code is computed. This hash code is used to determine the index at which the word (and its corresponding line numbers) should be stored in the map. If two words have the same hash code, a collision occurs. In such cases, `unordered_map` uses a technique called chaining to handle the collision. Each index in the array points to a linked list of entries that have the same hash code.

When a word is searched in the `unordered_map`, the hash code of the word is computed again. This hash code is used to find the index of the array where the word (and its corresponding line numbers) should be stored. If the word is found at the calculated index, the line numbers are returned. If the word is not found, a message is printed indicating that the word was not found.

This use of hashing allows the `unordered_map` to provide fast access to the line numbers of a word in the file, making the search feature efficient.

Menu Function

The `menu` function is the main function of the program. It displays the menu of operations and performs the selected operation. It uses a switch statement to select the operation based on the user's input. The `menu` function also handles the display of messages to the user, such as *"File Created Successfully"*, *"File Updated Successfully"*, *"File Emptied Successfully", "File Deleted Successfully"*, and *"File Not Found"*. These messages are displayed based on the value of the `msg` parameter passed to the `menu` function.

Code Presentation

The code is well-structured and easy to understand. It follows good programming practices such as using meaningful variable names, using comments to explain what the code does, and organizing the code into functions that each perform a specific task. The code also uses the `iostream` library for input and

output operations, the `fstream` library for file operations, and the custom `Stack.h` header file for stack operations.

Video Demo

TEXT EDITOR MENU SCREENSHOT.

You can check out the video demo for this project here.

Conclusion

The text editor in C++ provides a simple yet functional interface for basic file operations. It demonstrates the use of stack data structures for undo and redo operations, and file I/O operations for copy, paste, and cut operations. The code is well-structured and easy to understand, making it a good example of a text editor in C++.

To visit GitHub repo, click here.