**Project Title: Digital Forensics File Reconstruction System**

Course Name: Data Structures and Algorithms

Course Code: CS-221

Group Details:

|  |  |  |
| --- | --- | --- |
| SR. | Name | Registration No. |
| 1 | Irab Zara | 2024244 |
| 2 | Mirza Khizar Baig | 2024294 |

Due Date: Sunday, 19th October 2024

Instructor Name: Dr. Zubair Ahmad

### 

### **1.0 Introduction and Objectives**

The initial implementation phase is now complete, and this report documents the work done, focusing on the core concepts for the Digital Forensic File Reconstruction System. As defined in the project proposal, the primary goal of this phase was to implement the structures needed to manage active file metadata and track deletion history.

We successfully met this deliverable's objective by implementing:

* A **Singly Linked List** to collect and manage the initial metadata of active files.
* A **Stack** to apply the **Last-In, First-Out** recovery mechanism for logically deleted files.

These implementations demonstrate the application of core DSA concepts learned in class, and establish the foundation required for the next development phase.

### **2.0 Data Structure Proofs and Complexity Implementation**

The core requirement of the project is to ensure that all frequently executed operations - such as file collection and recovery tracking, operate with the best possible efficiency. The chosen implementations for this deliverable directly leverage **Constant time** **complexity (O(1)),** where appropriate.

#### **2.1 Active Files Metadata**

While in the later full implementation we will implement a **high-performance tree structure** for this purpose, we have temporarily used a Singly Linked List built from a node structure to manage the ‘active’ list.

* **Functionality:** This structure serves as the initial **Collection Layer**, built by traversing the designated Temp directory and using the **append** operation to collect file metadata. This is for efficient, sequential collection before the data is organized for searching.

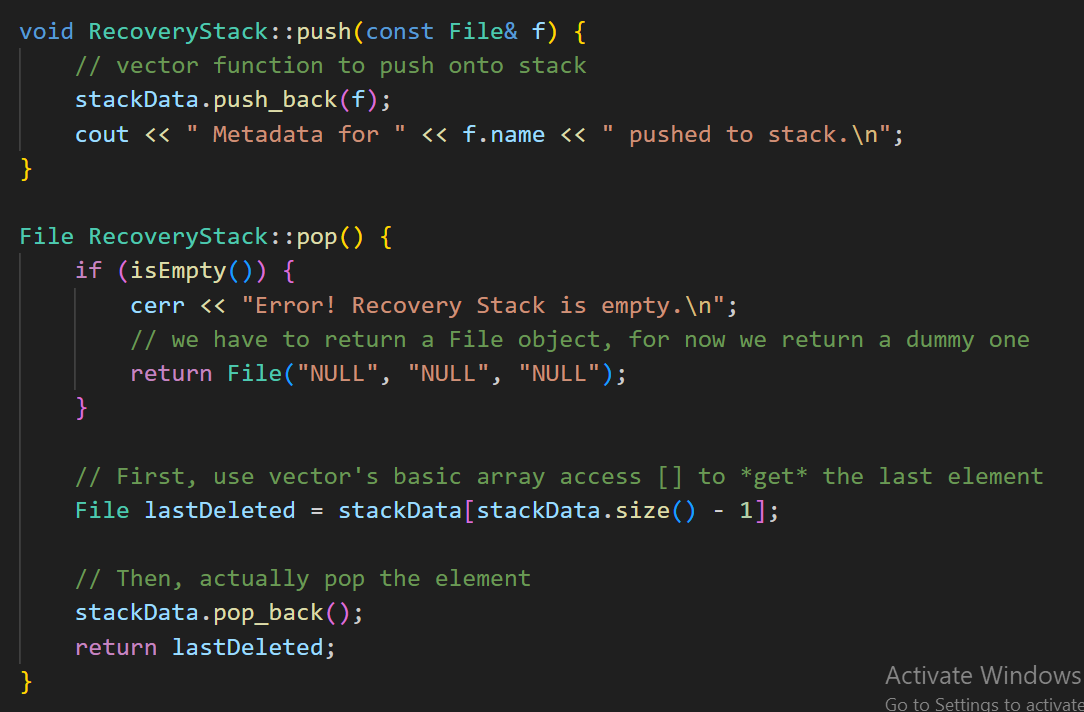
#### **2.2 Recovery History Tracking**

This is implemented inside the RecoveryStack class to apply the **LIFO concept** for file recovery, as is necessary for an "Undo" feature.

* **Implementation:** The Stack utilizes the C++ **vector** container internally. This is highly efficient because the Stack operations – **push** (simulating a logical delete) and **pop** (simulating recovery) are mapped directly to the vector's push\_back() and pop\_back() methods, ensuring **O(1) time complexity**.

**2.3 Code Proof**

#### The following code snippet from DataStructures.cpp confirms the efficient implementation of the recovery mechanism, demonstrating that the Stack operations are achieved by utilizing the vector's constant-time back-access methods.



**Figure 1:** Implementation of the Recovery Stack’s core methods, push and pop.

### **3.0 Future Development & Conclusion**

We have implemented the foundational **Linked List** and **Stack** structures to establish basic functionality. While the current Singly Linked List allows for efficient file collection, it cannot handle the fast searching required by our developing system.

To achieve prime performance, we will replace the current Linked List with a Binary Search Tree (BST) or an AVL Tree in Deliverable 3. This migration shifts the active file table's search performance from **O(n)** (linear time) to the more efficient **O(logn)** (logarithmic time) complexity.

**Future Steps:**

The next steps are as follows:

* **Finalize** the Object-Oriented Programming design through the FileManager class.
* **Implement** the user interface to accept commands.
* **Establish** the framework for the tree integration in Deliverable 3.