**Project Proposal: Notebook Application in C++**

**Group Member Names:**

**1)**Muhammad Shareq Usman

**2)**Muhammad Ghazali Khan

**3)**Sohaib Bin Tausif

**1. Problem Description and Project Aim**

In today’s busy world, people often need to organize their notes effectively, whether for work, studies, or personal activities. Traditional notebooks lack flexible organization, urgency tracking, and convenient navigation.

The aim of this project is to build a Notebook Application in C++ that allows users to:

* Add, edit, and delete notes.
* Categorize notes (e.g., work, play, personal).
* Mark notes with urgency levels and favourite bookmarks.
* Navigate notes efficiently.
* Undo/redo operations to recover accidental deletions.

This project combines multiple data structures into one practical application, showing how theoretical concepts can be applied to solve real-world problems.

**2. Key Data Structures**

The project will integrate the following **core data structures**:

1. Dynamic Array (Array List)
   * To maintain a master list of all notes.
   * Allows fast access and searching.
2. Singly Linked List
   * Used for managing notes in specific categories.
   * Provides efficient insertion and deletion for category-based grouping.
3. Doubly Linked List
   * Used for navigating through notes in both forward and backward directions, like flipping pages in a notebook.
4. Stack
   * Implements an undo/redo mechanism.
   * Stores recently deleted or edited notes to restore them if needed.
5. Pointers
   * Fundamental for dynamic memory management, linking nodes, and efficient data traversal in linked lists and stacks.

**3. Main Algorithms**

The project will rely on several key algorithms:

* Insertion Algorithm:  
  Add a new note to the array list, category list, and doubly linked list.
* Deletion Algorithm:  
  Remove a note from all structures, pushing the old note onto the stack for undo.
* Update Algorithm:  
  Modify note content; previous version is stored on the stack.
* Undo/Redo Algorithm (stack-based):
  + Undo: Pop the last action from the stack and restore it.
  + Redo: Push undone actions back into the main structures.
* Search Algorithm:  
  Search notes by ID, category, urgency, or favorite flag. Linear search will be used in arrays and linked lists.
* Traversal Algorithm:  
  Traverse linked lists to display notes by category or urgency.

**4. Data Flow (Input, Processing, Output)**

* Input:
  + User enters note details (title, content, category, urgency, favorite).
  + Commands for editing, deleting, or undoing actions.
* Processing:
  + Data is stored in an **array list** for quick access.
  + Notes are organized into **singly linked lists** per category.
  + A **doubly linked list** allows navigation between notes.
  + A **stack** tracks modifications for undo/redo.
* Output (Display):
  + Notes displayed by category or urgency.
  + Navigation through previous/next notes using doubly linked list.
  + Urgency/favorite markers shown alongside notes.
  + User notified of successful undo/redo actions.

**5. Integration of Key Concepts**

This project will integrate the following **key concepts learned in class**:

* Pointers: To dynamically allocate memory and connect nodes in linked lists and stacks.
* Linked Lists: To manage categories of notes and enable efficient insertions/deletions.
* Array Lists: To store all notes with direct indexing and searching.
* Doubly Linked Lists: For bidirectional navigation of notes.
* Stacks: To manage undo/redo functionality using LIFO operations.
* Algorithms: To implement searching, insertion, deletion, updating, and undo/redo efficiently.

By combining these structures and algorithms, the project demonstrates a practical application of data structures to solve real-world note management problems.