National University of Computer and Emerging Sciences, Peshawar

Data Structures — Fall 2025

Assignment 01

Instructions

- Read the problem statement carefully and implement only what is asked.
- Plagiarism will not be tolerated. Any evidence of copied work, including minor instances, will result in a grade of zero.
- Late submissions will not be accepted under any circumstances.
- Submission: Keep your file name as `24p-xxxx.cpp` and submit it to GCR. Do not submit binaries or zip files.

Question 1: Linked List Text Editor in C++

:

FAST (**Foundation of Advancement of Science and Technology**) is an institution that continuously strives to push the boundaries of computing and innovation. Recently, the university's software division has been receiving numerous complaints from students, teachers, and researchers about inefficiencies in existing text editors.

Conventional editors rely on arrays or string buffers, which work fine for small documents. However, when working with theses, research reports, books, or programming code containing thousands or millions of characters, these editors become slow. Inserting in the middle of a paragraph, deleting a character, or moving the cursor often requires shifting large blocks of memory, wasting both time and resources.

To overcome this, the research department has proposed building a new text editing engine based entirely on **linked lists**. Each character will be stored as a node, enabling faster editing operations in large documents.

1. Data Representation (Implement in C++)

- Each **character** must be stored in a doubly linked list node.
- Each **line** must be represented as a doubly linked list of characters.
- The whole **document** must be stored as a doubly linked list of lines.
- The **cursor** is represented as a pointer to the current character node.

2. Cursor Movement Functions (Implement in C++)

- `moveCursorLeft()` – Move the cursor one character left.
- `moveCursorRight()` – Move the cursor one character right.
- `moveCursorUp()` – Move the cursor to the previous line (same character index if possible).
- `moveCursorDown()` – Move the cursor to the next line (same character index if possible).

3. Editing Functions (Implement in C++)

- `insertChar(char c)` – Insert a character at the cursor position.
- `backspace()` – Delete the character to the left of the cursor.
- `deleteChar()` – Delete the character to the right of the cursor.

4. Advanced Feature (Implement in C++)

- `searchChar(char c)` – Traverse the entire document and return the total number of occurrences of the given character.

What is the Cursor?

In this linked list text editor, the cursor is represented by a vertical bar `|` and marks the position where the next character will be inserted or deleted. The cursor is not a separate character but simply a pointer to the position in the linked list.

Constraints

- You are not allowed to use arrays, strings, vectors, or any pre-built containers for storage or editing.
- All functionality must be implemented using custom linked list structures only.

Question 2: Aais' Linked List Blockchain Adventure in C++

Aais, a young and curious programmer, discovered a mysterious digital world known as the **Blockchain of Tasks**. In this world, every task, treasure, or piece of information is stored in a magical block. Each block is chained to the previous one by an invisible lock called a Key. If anyone tries to change a block without following the rules, the chain breaks, causing chaos!

Each block in Aais' blockchain has the following attributes:

- **BlockID** – A unique number identifying the block.
- **Data** – A string representing a task, treasure, or message. This must be included in the block structure.
- **WorkDone** – A number representing the effort needed to complete the task.
- **PrevKey** – The Key of the previous block in the chain.
- **CurrKey** – The Key of the current block, calculated from BlockID, WorkDone, Data, and PrevKey.
- **Next1, Next2** – Pointers to the next block(s), allowing forks.

Important Clarifications (Implement in C++)

- The **Data** field stores the content of the block and does not change during tampering.
- **Tampering** simulates corruption by modifying the **CurrKey** of a block manually. Do not change **Data**.
- After tampering, your program must detect broken blocks (where **PrevKey** of a child block does not match the **CurrKey** of the parent).
- **Repairing** recalculates **CurrKey** for the tampered block and all subsequent blocks using the formula:

  $$\textbf{CurrKey}=(\textbf{BlockID} \times \textbf{37} + \textbf{WorkDone} \times \textbf{19} + \textbf{sum of ASCII values of Data} + \textbf{PrevKey})\%\textbf{100000}$$

Functions to Implement (C++ Program)

- `createGenesisBlock()` – Initialize the first block with `PrevKey = 0` and `BlockID = 1`.
- `addBlock(string data, int work)` – Add a new block at the end of the current main chain.
- `forkChain(int blockID, string data1, string data2, int work1, int work2)` – From a specified block, create two new blocks, forming a fork.
- `tamperBlock(int blockID, int newKey)` – Modify the **CurrKey** of a block to simulate tampering. Do not change **Data**.

- `repairChain(int blockID)` – Recalculate **CurrKey** for the tampered block and all subsequent blocks.
- `validateChain()` – Traverse all blocks and report any mismatches between **PrevKey** and **CurrKey**.
- `consensusRule()` – Decide the correct chain: pick the longest valid chain, or if two chains have the same length, pick the one with the highest total `WorkDone`.
- `displayAllChains()` – Print all chains from Genesis to each end block, showing `BlockID`, `Data`, and `CurrKey`.

Calculations for the Winning Chain (Implement in C++)

- **Length** – Number of blocks in the chain.
- **Total WorkDone** – Sum of `WorkDone` for all blocks in the chain.
- **Integrity Report** – List any tampered or corrupted blocks.

Rules and Constraints (C++ Program)

- Use only **linked lists**; no arrays, vectors, or pre-built containers.
- Each block may have up to two next pointers to handle forks.
- All functions – creation, traversal, tampering, repair, validation, consensus, and printing – must be implemented manually.
- Your solution must handle forks, tampering, and repairs efficiently, even for large chains.

Made by humans with love.

Crying is allowed, but AI is not!

Happy coding!