Data Structures and Algorithm

CSIS217 Advanced Data Structures Fall 24-25

Project

Objective: The objective of this project is to empower you with hands-on experience implementing appropriate data structures in C++ to enhance program efficiency. The primary goal of this project is to deepen your understanding of the data structure through practical implementation.

Project Idea: Build the Lebanese Chart of Accounts

Description: You are asked to build a class **ForestTree**, described at the end of section 15.3 in the textbook: "ADT Data Structures and Problem Solving". The data structure **ForestTree** can be used to represent any tree using nodes that have only two links-that is, by a binary tree. We use one of these links to connect siblings together (that is, all the children of a given node) in the order in which they appear in the tree, from left to right. The other link in each node points to the first node in this linked list of its children.

Problem Statement: Given a set of accounts and subaccounts (check the related txt file), implement a ForestTree class that is able to represent all the account and their related subaccounts as well as their corresponding transactions. Using the provided data, create a user-friendly application that will read the data from the provided txt file and save all the given accounts into an object of type ForestTree. The program must allow the user to manipulate (add, remove) the accounts and subaccounts, and to apply debit and credit transactions to any existing account.

Key Components:

1. **Account:** Each account or subaccount has a unique number (1 to 5 digits), a unique description, a balance and a vector of **Transaction**s**.** Accounts and subaccounts are related by their most significant digit(s). For example, subaccount 11 is a child of account 1 and subaccount 111 is a child of subaccount 11.
   Functionalities for the transaction must be defined including overloaded operators << and >> and others if needed.

2. **Transaction:** each transaction (called JV: Journal Voucher) is related to a specific subaccount, a positive amount and a debit_credit character. When a transaction is added to an account, the amount of a debit transaction is to be added to the balance of the related account and the amount of a credit transaction must be subtracted from the balance of the related account/subaccount. The effect of this addition must update the amount of all higher related subaccounts up to the main account.
   Example: if a credit transaction with an amount of 50$ is added to the subaccount 1112, the value 50 must be subtracted from the balance of the subaccount 1112 as well as 111 and 11 and the main account 1.

Functionalities for the transaction must be defined including overloaded operators << and >> and others if needed.

Deliverables:

1. **Source Code:** Provide well-structured C++ code for the forest tree data structure. Your C++ code should include, at a minimum, 7 functionalities (functions) including but not limited to:
   1. Function to initialize an empty forest tree.
   2. Functions to build a chart of accounts by reading the data from a text file.
   3. Functions to add an account that doesn't exist.
   4. Functions to add and delete a transaction from the user.
   5. Functions to print, into a file that you have to generate its name: account number followed by utmost the first 10 characters of the description, a detailed report for an account that includes the subaccount(s), if any, and its(their) transactions as well as the balance(s).
   6. Function to search in the data structure for a specific account using its number.
   7. Function to print the forest tree into a file.

Evaluation Criteria: Your work will be evaluated based on the following criteria:

- **Delivery:** Complete all the requirements, Deliver on time, and in correct format. (5%)
- **Code Standards:** Emphasizes clean code, includes name(s) of author(s), date, and assignment title, excellent use of white space and empty lines, indentation, lines length, creatively organized work, use of variables (no global variables, unambiguous naming) (15%).
- **Documentation:** Clear and effective documentation (files, functions: specific purpose is noted for each function as well as input requirements, and output results, control structures and code segments including descriptions of all variables. The documentation must serve as a comprehensive guide (15%).
- **Runtime**: Execution without errors, excellent user prompts, good use of symbols, spacing in output. Thorough and organized testing has been completed and output from test cases is included (50%).
- **Efficiency:** Solution is efficient (time/memory wise), easy to understand and to maintain. (15%).

Project Group Limit: This project should be completed with a maximum group size of three students.