Lab Exam(Group B) **May9th, 2024**

Task1: Bank Management System

Objectives:

Develop a simplified banking system using templates to manage different types of accounts with varying interest calculation methods. Additionally, ensure memory safety and efficient memory management by using smart pointers. Utilize a stack data structure to track transaction history for each account.

Requirements: Define a templated class named Account with the following properties and methods:

Properties:

accountNumber (string): Unique identifier for the account.

balance (double): Current balance in the account.

transactionHistory (std::stack<Transaction>) : A stack to store transaction history, where Transaction is a struct containing details such as transaction type (deposit, withdrawal), amount, and timestamp.

Methods:

void deposit(double amount): Adds the specified amount to the account balance and pushes the transaction details onto the transaction history stack.

void withdraw(double amount): Deducts the specified amount from the account balance and pushes the transaction details onto the transaction history stack.

Use smart pointers (std::unique_ptr or std::shared_ptr) to manage memory for objects of the Account class.

Implement a function named **displayAccountDetails** that takes a vector of smart pointers to Account objects as a parameter and prints the details (account number, balance, and transaction history) of each account.

Create a main function to demonstrate the functionality of your banking system. In main, create instances of different types of accounts (using the Account template) and store them in a vector of smart pointers to Account. Then, perform various operations such as deposits, and withdrawals, on these accounts. Finally, call the **displayAccountDetails** function to display the details of each account.

Task 2(A): Template Container Class

Define a template class Container for storing different types of values such as int, double, float, etc. The Container class should consist of the following member variables:

T* values: A pointer to a set of values it contains. This member points to the dynamically allocated array (allocated in the constructor).

capacity: An integer that shows the total capacity of the container.

counter: An integer counter variable that increments upon every insertion operation and decrements upon removal of any element, representing the total number of elements currently present in the container.

The Container should consist of the following functions:

Constructor with capacity as a parameter.

isFull(): Returns true if the counter reaches capacity, otherwise false.

insert(): Receives a value of some type (int/double/float) and inserts it into the container if it's not already full.

search(): Returns true if the container contains the value passed as a parameter.

remove(): Removes a value, if it exists, from the container and shifts all subsequent values one index back.

print(): Prints all values contained in the container.

Now create a test class to create three instances of Container for types: int, float, double, and call all functions separately for each instance. Ensure your output is properly formatted.

Task2 (B)

Extend the functionality of Question #02(A) by creating a template function named testContainer before the main function. This function will accept a reference to a Container<T> object and several values as parameters: valueToBeInserted, increment, valueToSearch, and valueToRemove.

The purpose of this function is to call each function of the Container class in sequence:

It inserts the valueToBeInserted into the container initially and then keeps inserting values until it reaches its capacity. Each new insertion will be incremented by increment.

It prints the existing values in the container.

It searches for valueToSearch in the container.

It removes the valueToRemove from the container if it exists.

Finally, it prints the existing values in the container after removal.

In the main function, create a new instance of Container for some type and call the testContainer template function by passing all required arguments. The template function will handle the rest of the operations.

Note: Ensure your code adheres to proper coding standards, including appropriate error handling and formatting of the output.

Task3: **Student Records Processing(use ranges)**

In this lab task, you'll work with a collection of student records, each containing information such as

name, ID, and GPA.

Utilize functions such as find_if, accumulate, sort, and count from the <algorithm> library to

implement the following functionalities:

Sort: Create a functor to sort the student records based on two criteria:

- Primary: GPA (descending order)
- Secondary: Student name (alphabetical order)

Create range view using composite function with help of |:

Count: After sorting, determine the number of students with names longer than 8 characters using a custom filter function.

Transform: Create a new list containing the student IDs transformed to uppercase using a lambda function.