CS3201: OBJECT ORIENTED PROGRAMMING LABORATORY

Lab Assessment

Topic: Function overloading, operator overloading, constructors in inheritance, friend function

Date: April 09, 2025

Questions(3 out of 4) 3 x 10 = 30

1. Implement a Password Management System that supports multiple users, each having their own login credentials and a separate master password. During registration, a user should provide their username, email, login password, and a master password. The system must store credentials (such as site name and corresponding password) for each user in a separate file named after the username (e.g., john.txt, alice.txt). The credentials must be saved in plain text format using the structure site_name password (e.g., gmail abc123). Once a user logs in with their username, email, and login password, they should be able to add new credentials, update existing credentials, or view their stored credentials. However, viewing credentials is only allowed after verifying the user's master password, which should be stored privately in memory and not in any file. Use file handling for reading and writing credentials, and implement a UserAccount class that manages the user's information. Also, implement a friend function viewUserData() that can access private members of the UserAccount class to display stored credentials, but only after successful master password verification. In addition, overload the == operator in the UserAccount class to compare two users based on their username and email.
2. Implement a String Processing Utility that performs two core operations — split and join — using only manual character array manipulation, without relying on any built-in string functions or STL containers. Create a class StringProcessor with two static functions. The split function should take a null-terminated input string (sentence) and a delimiter character, and manually extract all tokens into a 2D character array. The join function should take a single null-terminated sentence, where words are separated by spaces, and a custom delimiter string (e.g., "->" or " | "). It must convert the space-separated words into a new single string where words are joined using the given delimiter, without adding a trailing delimiter at the end. Demonstrate both functions with test cases like 'splitting "apple,banana,,grape," using , as a delimiter' and 'joining "one two three" using "->" as the joiner'.

3. Implement a C++ program that simulates a basic HTTP request logger. Begin by defining a base class named HTTPRequest that stores information about the HTTP method (such as GET or POST) and the request path (such as /home, /login). Then, create two derived classes: GETRequest and POSTRequest, both of which inherit from HTTPRequest. Use constructors in the derived classes to initialize the HTTP method and path values appropriately using base class constructors. In the POSTRequest class, additionally include a request body (e.g., username=admin&password=1234), which should also be initialized via the constructor. Within the HTTPRequest class, implement a member function log() that displays the request method and path. Overload this function with another version that accepts a string parameter to additionally display a response status code like "200 OK" or "404 Not Found". In the case of POSTRequest, the log() function should also print the body along with the method and path. You must also define a friend function named printMethod() that takes a HTTPRequest object and displays only the HTTP method used.
4. Implement a C++ program that simulates an advanced media file player supporting both audio and video files. The base class Media should store the media name and duration in seconds. It should also keep track of how many times the file has been played using a playCount variable. Derive two classes from Media: Audio and Video. Each derived class should include additional metadata. For Audio, store the artist name; for Video, store the resolution (e.g., "1080p"). The base class should have a virtual play() method that is overloaded in the derived classes to support two versions: one without parameters (default play), and one that accepts a startTime and endTime to simulate partial playback. Overload the == operator in the Media class to compare whether two media files are the same (i.e., same name and duration). Also implement a friend function called displayMediaInfo() that prints complete media details, including play count. Implement a play history tracker — an array of pointers storing the last 5 media objects played, which is updated each time any media is played.