the Social Media Class Hierarchy:

1. Access Control and Getters:

Create the User class with private members for username and profile picture (string).

Implement public member functions for the constructor and getters (accessor methods) for username and profile picture.

2. Post Class and Display:

Create the derived class Post inheriting from User.

Add private members for post content (string) and timestamp (date/time format of your choice).

Implement a public member function getPostInfo that returns a formatted string containing username, profile picture, post content, and timestamp.

3. Basic Interaction Function:

Define a friend function basicInteract that takes two User objects (or derived class objects) as arguments.

Inside the function, simply print a generic message like "User1 interacts with User2."

4. Overloaded Interact Functions:

Create overloaded versions of the interact function:

likePost(User& user, Post& post): This function should print a message indicating the user liked the post.

followUser(User& follower, User& followed): This function should print a message indicating the user started following another user.

5. Refactoring with Encapsulation:

Revisit the class design. Can you modify the code to reduce reliance on friend functions?

Consider adding public member functions or accessor methods within the User class to provide controlled access to relevant data instead of exposing everything through friend functions.

Bonus Challenge:

Implement a way to store and manage friend connections within the class hierarchy. You could explore a separate Friendship class or a boolean flag within User to track friend status. Modify the interact functions to incorporate this information and display more relevant messages based on the relationship between users.

#include <iostream>

#include <string>

#include <ctime>

using namespace std;

class User {

private:

string username;

string profilePicture;

protected:

// Protected accessor methods for derived classes

string getUsername() { return username; }

string getProfilePicture() { return profilePicture; }

public:

// Constructor

User(string username, string profilePicture) : username(username), profilePicture(profilePicture) {}

// Public accessor methods

string getPublicUsername() { return username; }

string getPublicProfilePicture() { return profilePicture; }

};

class Post : public User {

private:

string postContent;

time\_t timestamp;

public:

// Constructor

Post(string username, string profilePicture, string postContent) : User(username, profilePicture), postContent(postContent) {

timestamp = time(0); // Set timestamp to current time

}

// Get post information

string getPostInfo() {

return "Username: " + getUsername() + ", Profile Picture: " + getProfilePicture() + ", Post Content: " + postContent + ", Timestamp: " + ctime(&timestamp);

}

};

// Basic interaction function (refactored to use public accessor methods)

void basicInteract(User& user1, User& user2) {

cout << "User " << user1.getPublicUsername() << " interacts with User " << user2.getPublicUsername() << "." << endl;

}

// Overloaded interact functions

void likePost(User& user, Post& post) {

cout << "User " << user.getPublicUsername() << " likes Post by " << post.getPublicUsername() << "." << endl;

}

void followUser(User& follower, User& followed) {

cout << "User " << follower.getPublicUsername() << " starts following User " << followed.getPublicUsername() << "." << endl;

}

// Bonus Challenge: Implementing friend connections

class Friendship {

private:

User\* user1;

User\* user2;

public:

Friendship(User& user1, User& user2) : user1(&user1), user2(&user2) {}

void interact() {

if (areFriends()) {

cout << "Friends " << user1->getPublicUsername() << " and " << user2->getPublicUsername() << " interact." << endl;

} else {

cout << "User " << user1->getPublicUsername() << " interacts with User " << user2->getPublicUsername() << "." << endl;

}

}

bool areFriends() {

// Simple implementation: assume friendship is mutual

return true; // Replace with actual friendship logic

}

};

int main() {

User user1("Alice", "alice.jpg");

User user2("Bob", "bob.jpg");

Post post1("Alice", "alice.jpg", "Hello, world!");

basicInteract(user1, user2);

likePost(user2, post1);

followUser(user2, user1);

Friendship friendship(user1, user2);

friendship.interact();

return 0;

}

Output:

