DAY – 7 Assignments & LAB Tasks

Assignment- 1

Objective - 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 access or 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>
using namespace std;
/* #include <ctime>
#include <vector>
#include <algorithm>

//1 User Class with Access Control and Getters

class User {
private:
    std::string username;
    std::string profilePicture;
    std::vector<User*> friends;
```

```
public:
    User(const std::string& name, const std::string& picture) : username(name), profilePicture(picture) {}

    std::string getUsername() const {
        return username;
    }

    std::string getProfilePicture() const {
        return profilePicture;
    }
     void addFriend(User& user) {
        if (std::find(friends.begin(), friends.end(), &user) == friends.end()) {
            friends.push_back(&user);
            user.friends.push_back(this);  // Add reciprocal friendship
        }
    }

    bool isFriend(const User& user) const {
        return std::find(friends.begin(), friends.end(), &user) != friends.end();
    }

    void interactWith(User& user) const {
        if (isFriend(user)) {
            std::cout << getUsername() << " interacts with friend " << user.getUsername() << std::endl;
        } else {
            std::cout << getUsername() << " interacts with " << user.getUsername() << std::endl;
        }
    }

    // Other methods related to User can be added here
};

void likePost(User& user, Post& post) {
    std::cout << user.getUsername() << " liked the post: \n" << post.getPostInfo() << std::endl;
}

void followUser(User& follower, User& followed) {
    follower.addFriend(followed);
    std::cout << follower.getUsername() << " started following " << followed.getUsername() << std::endl;
}

    // Other methods related to User can be added here
};

//2 Post Class and Display
class Post : public User {
private:
    std::string content;
    std::time_t timestamp;

public:
    Post(const std::string& name, const std::string& picture, const std::string& postContent, std::time_t
time)
        : User(name, picture), content(postContent), timestamp(time) {}


    std::string getPostInfo() const {
        char buffer[80];
        struct tm* timeinfo = localtime(&timestamp);
        strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", timeinfo);


        return "User: " + getUsername() + "\nProfile Picture: " + getProfilePicture() + "\nContent: " +
content + "\nTimestamp: "
        + std::string(buffer);


    }
};
//3 Basic Interaction Function
void basicInteract(const User& user1, const User& user2) {
    std::cout << user1.getUsername() << " interacts with " << user2.getUsername() << std::endl;
}
//4 Overloaded Interact Functions
void likePost(User& user, Post& post) {
    std::cout << user.getUsername() << " liked the post: \n" << post.getPostInfo() << std::endl;
}


void followUser(User& follower, User& followed) {
    std::cout << follower.getUsername() << " started following " << followed.getUsername() << std::endl;
}
```

```
int main() {
    User user1("Alice", "alice_pic.png");
    User user2("Bob", "bob_pic.png");
    std::time_t now = std::time(0);

    Post post1("Alice", "alice_pic.png", "Hello, world!", now);

    basicInteract(user1, user2);

    likePost(user1, post1);
    followUser(user1, user2);

    user1.interactWith(user2);

    return 0;
}
```

2. Code for Static Variable & Member and Normal Variable & Member declaration

```
class MyClass {
    private:
    static int counter;  //static member variable 'counter' to keep track of object instances
    int count;
    public:
    MyClass(){
        count++;
        counter++;  // Increment counter for each object creation
    }
    static int getCounter(){  // Static method used to access and return the counter
        return counter;
    }
   void display() {
        cout << "This is a example of a Normal member function." <<endl;
    }
    int getCount(){
        return count;
    }
};

// Initilize static variable outside the class
int MyClass:: counter = 0 ;  // Set Initial values = '0'

int main(){
    MyClass obj1;
    MyClass obj2;
    MyClass obj3;
    cout<<"Number of Objects created -> "<<MyClass::getCounter()<<endl;
    cout<<"Object1 count method -> "<<obj1.getCount()<<endl;
    cout<<"Object2 count method -> "<<obj2.getCount()<<endl;
    cout<<"Object3 count method -> "<<obj3.getCount()<<endl;
```

```
  obj1.display();

  return 0;

}
```

3. Assignment-2

// Objective : Distance Converter:

Create a class named DistanceConverter. Include the following static

methods:

convertMilesToKm(double miles): Converts miles to kilometers

(1 mile = 1.60934 kilometers).convertKmToMiles(double kilometers):

Converts kilometers to miles. In your main function, prompt the user for a distance and a unit (miles or kilometers). Use the appropriate static method from the DistanceConverter class to perform the conversion and display the result to the user.

Math Utility Class:

Design a class named MathUtil. Include static methods for basic

mathematical operations:

add(int a, int b): Adds two integers.

subtract(int a, int b): Subtracts two integers.

multiply(int a, int b): Multiplies two integers.

divide(int a, int b) (optional): Divides two integers with error handling for division by zero. In your main function, prompt theuser for two numbers and an operation (+, -, *, or /). Use the corresponding static method from the MathUtil class to perform the calculation and display the result.

Simple Currency Converter:

Create a class named CurrencyConverter. Define a static variable

named exchangeRate (e.g., USD to EUR exchange rate). Implement

static methods: convertToEur(double amount): Converts an amount from the base currency (USD) to EUR based on the exchange rate. convertFromEur(double amount): Converts an amount from EUR to the base currency (USD). In your main function, prompt the user for an amount and a conversion direction (USD to EUR or EUR to USD).Use the appropriate static method from the CurrencyConverter class to perform the conversion and display the result.

1. Code for Distance Convertor

```
class DistanceConverter {
public:
    static double convertMilesToKm(double miles) {  // 1 mile = 1.609 k.m
        return miles * 1.609;
    }

    static double convertKmToMiles(double kilometers) {
        return kilometers / 1.609;
    }
};

int main() {
    // Distance Conversion
    cout << "Distance Conversion:" << endl;
    double distance;
    string unit;
    cout << "Enter distance: ";
    cin >> distance;
    cout << "Enter unit (miles/km): ";
    cin >> unit;
    if (unit == "miles") {
        cout << distance << " miles is " << DistanceConverter::convertMilesToKm(distance) << " kilometers." << endl;
    } else if (unit == "km") {
        cout << distance << " kilometers is " << DistanceConverter::convertKmToMiles(distance) << " miles." << endl;
    } else {
        cout << "Invalid unit!" << endl;
    }
}
```

2. Code for Arthimetic Calculator

```
class MathUtil {
public:
    static int add(int a, int b) {
        return a + b;
    }

    static int subtract(int a, int b) {
        return a - b;
    }

    static int multiply(int a, int b) {
        return a * b;
    }
};


int main(){
    cout << "\nMath Operations:" << endl;
    int a, b;
    char operation;
    cout << "Enter the first number: ";
    cin >> a;
    cout << "Enter the second number: ";
    cin >> b;
    cout << "Enter the operation to perform (+, -, *) ? :";
    cin >> operation;

        if (operation == '+') {
            cout << "Result: " << MathUtil::add(a, b) << endl;
        } else if (operation == '-') {
            cout << "Result: " << MathUtil::subtract(a, b) << endl;
        } else if (operation == '*') {
            cout << "Result: " << MathUtil::multiply(a, b) << endl;
       }
          else {
            cout << "Invalid operation asked!" << endl;
        }
 }
```

3. Code for Currency Convertor

```
class CurrencyConverter {
public:
    static double exchangeRate;

    static double convertToEur(double amount) {

        return amount * exchangeRate;
    }

    static double convertFromEur(double amount) {
        return amount / exchangeRate;
    }
};

// double CurrencyConverter::exchangeRate = 0.85; // Example exchange rate from USD to EUR
int main() {
    // Currency Conversion
    cout << "\nCurrency Conversion:" <<endl;
    double amount;
    string option;
    cout << "Enter amount: ";
    cin >> amount;
    cout << "Enter conversion direction (USD to EUR (Type-1)/EUR to USD(Type-2)): ";
    cin >> option;
    if (option == "1") {
        cout << amount << " USD is " << CurrencyConverter::convertToEur(amount) << " EUR." << endl;
    } else if (option == "2") {
        cout << amount << " EUR is " << CurrencyConverter::convertFromEur(amount) << " USD." <<
endl;
    } else {
        cout << "Invalid conversion direction!" << endl;
    }
    return 0;
}
```

4. Code for Use of Function Templates

Showcasing the flexibility and reusability of template functions in C++.

```
template <class T> T add(T &a,T &b) // a and b are refrences to T
{
    T result = a+b;
    return result;
};
int main()
{
    int i = 2;
    int j = 3;
    float m = 2.3;
    float n = 1.2;
    cout << "(Integer) Addition of i and j is : "<<add(i,j)<<endl;
    cout << " (Floating) Addition of m and n is : "<<add(m,n);
```

```
   return 0;
}
```

5. Code for Function Templates for Multiple Parameters

```
template <class X,class Y> void fun( X a,Y b)
{
   cout<<" The Value of a is: "<<a<<endl;
   cout<<" The Value of b is: "<<b<<endl;
}
 int main()
 {
    fun(15,12.3);  // Two data types one is 'int' other is 'float'.
    return 0;
 }
```

5. Code for Overloading Function Templates.

```
template <class X> void fun(X a)
{
   cout<<"The value of a is :"<<a<<endl;
}
template <class X,class Y> void fun(X b,Y c)
{
   cout<<"The value of b is :"<<b<<endl;
   cout<<" The value of c is :"<<c<<endl;
}
```

```
int main()
{
    fun(10);
    fun(30,30.5);
    return 0;
}
```

Assignment-3

Objective : 1. Design a function template named compare that takes two arguments of the same type and returns a boolean value indicatingwhether the first argument is greater than, less than, or equal to

the second argument. How would you adapt this template to work withcustom data types?

2. Implement a function template named swap that exchanges the values of two variables of the same type. Discuss the potential limitations of this approach when dealing with complex data structures.

3.Consider a scenario where you need to find the minimum value in an array. Create a function template named findMin that works with any data type for which the comparison operator (<) is defined. Explain how function templates promote code reusability in this case.

```
template <typename T>
bool compare(const T& a, const T& b) {
    if (a > b) {
        cout << "First argument is greater than the second argument." << endl;
        return true;
    } else if (a < b) {
        cout << "First argument is less than the second argument." << endl;
        return false;
    } else {
        cout << "Both arguments are equal." << endl;
        return false;
    }
};

// Example custom data type
struct CustomType {
    int value;
    bool operator>(const CustomType& other) const {
        return value > other.value;
    }
    bool operator<(const CustomType& other) const {
        return value < other.value;
    }
    bool operator==(const CustomType& other) const {
        return value == other.value;
 }
```

```
int main() {
    int a = 5, b = 3;
    compare(a, b);

    CustomType x = {10};
    CustomType y = {20};
    compare(x, y);

return 0;
}
```

2. Swap Function template

```
template <class T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    int a = 5, b = 10;
    cout << "Before swap: a = " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "After swap: a = " << a << ", b = " << b << endl;
    double x = 1.1, y = 2.2;
    cout << "Before swap: x = " << x << ", y = " << y << endl;
    swap(x, y);
    cout << "After swap: x = " << x << ", y = " << y << endl;

    return 0;
}
```

3. FindMin Function Template

```
template <class T>
T findMin(T arr[], int size) {
    T minVal = arr[0];
    for (int i = 1; i < size; ++i) {
        if (arr[i] < minVal) {
            minVal = arr[i];
        }
    }
    return minVal;
}

int main() {
    int arrInt[] = {6, 2, 4, 7, 1};
    cout << "Minimum integer value is : " << findMin(arrInt, 5) << endl;

    double arrDouble[] = {4.8, 7.2, 3.2, 1.5};
    cout << "Minimum double value is : " << findMin(arrDouble, 4) << endl;

    return 0;

}
```