Lab 1: OOP Basics with C++ (Classes)

Objective:

Introduce students to Object-Oriented Programming (OOP) fundamentals in C++, focusing on class creation, object manipulation, and the interaction between objects of different classes.

IDE: Visual Studio 2022

Introduction to OOP in C++:

This lab session explores the basics of OOP, a programming paradigm that uses objects and classes for more efficient and organized code. By the end of this session, students will understand how to create classes, instantiate objects, and enable interactions between different objects, embodying real-world relationships in software design.

Classes vs. Structures:

- **Access Control**: Classes have private access by default, enhancing data encapsulation. Structs are public by default, suited for simpler data structures.
- **Inheritance**: Both classes and structs can inherit from others, but classes are primarily used in OOP for their encapsulation benefits.
- **Syntax and Usage**: Though the syntax is similar, classes are used for complex data structures requiring methods, whereas structs are typically used for simpler data structures.
- **Pros and Cons**:
    - **Classes**:
        - **Pros**: Offer better encapsulation and data protection; suitable for complex systems.
        - **Cons**: Can introduce complexity due to enforced access controls.
    - **Structs**:
        - **Pros**: Simpler to use for basic data storage without the need for encapsulation.
        - **Cons**: Limited functionality in terms of encapsulation and object-oriented features.
- **Key Difference**: The default access control and conventional usage; classes support encapsulation and are used in complex scenarios, whereas structs are preferred for simpler data structures.

Sample Code Demonstrations and Expected Outputs:

Step 1: Defining a Class and Creating Objects

- Code:

```
class Person {
public:
   string name;
   int age;

   void displayDetails() {
      cout << "Name: " << name << ", Age: " << age << endl;
   }
};

int main() {
   Person person1;
   person1.name = "Alice";
   person1.age = 30;
   person1.displayDetails();
   return 0;
}
```

- Expected Output:

```
Name: Alice, Age: 30
```

Step 2: Using Constructors and Destructors

- Code:

```
class Person {
public:
   Person(string n, int a) : name(n), age(a) {
      cout << "Person object created" << endl;
   }

   ~Person() {
      cout << "Person object destroyed" << endl;
   }

   void displayDetails() {
      cout << "Name: " << name << ", Age: " << age << endl;
   }

private:
   string name;
   int age;
};
```

```
int main() {
    Person person1("Alice", 30);
    person1.displayDetails();
    return 0;
}
```

- Expected Output upon Creation and Deletion:

```
Person object created
Name: Alice, Age: 30
Person object destroyed
```

Step 3: Interaction Between Different Class Objects

- Code:

```
class Dog {
public:
    string name;
    void bark() {
        cout << name << " says Woof!" << endl;
    }
};

class Person {
public:
    string name;
    Dog pet;

    void introducePet() {
        cout << "My dog's name is " << pet.name << endl;
        pet.bark();
    }
};

int main() {
    Person person1;
    person1.name = "Alice";
    person1.pet.name = "Buddy";
    person1.introducePet();
    return 0;
}
```

- Expected Output:

```
My dog's name is Buddy
Buddy says Woof!
```

In-Lab Tasks:

Task - 1. Expand the `Dog` Class:

- Add breed and age attributes. Implement a method to display the dog's details. Expected output should include the dog's name, breed, and age.

Task - 2. Create a `House` Class:

- Implement functionality to add a `Person` to the house and display all persons' details. Expected output should list all persons in the house by name and age.
-

Task - 3. Interaction Between `Person`, `Dog`, and `House`:

- Model a scenario where a person owns a dog and lives in a house. Expected output should detail the person, their dog, and their house in a structured format.

Extra Tasks for Home:

Task - 4. Enhance the `Person` Class:

- Implement a method to change the owner's dog, demonstrating dynamic relationships.
- Expected output should show the person introducing a new pet.

Task - 5. Implement a `Neighborhood` Class:

- Create functionality to add houses and display information about each house and its inhabitants.
- Expected output should list all houses and their residents.

Task - 6. Partially Implemented `Car` Class:

- Students are given a partially implemented **Vehicle** class and asked to complete the class's functionality, focusing on encapsulation and utilizing methods effectively.
- Complete the implementation of a method that checks if a car needs servicing based on its mileage.

- Code:

```
#include <iostream>
using namespace std;

class Vehicle {
private:
   string make;
   string model;
   int year;

public:
   Vehicle(string mk, string mdl, int yr) : make(mk), model(mdl), year(yr) {}
```

```
    void setMake(string mk) {
        make = mk;
    }

    // TASK: Implement setModel method
    void setModel(string mdl) {
        // Your implementation here
    }

    // TASK: Implement setYear method
    void setYear(int yr) {
        // Your implementation here
    }

    void displayVehicleInfo() {
        // TASK: Complete this method to display the vehicle's information
        cout << "Make: " << make << ", Model: " << model << ", Year: " << year
<< endl;
    }
};

int main() {
    Vehicle myVehicle("Toyota", "Corolla", 2020);
    myVehicle.displayVehicleInfo();
    // Expected to modify and display new details using setModel and setYear
    return 0;
}
```

- **Expected Output:**
    - (Assuming servicing is needed after 10,000 miles): Car needs servicing. ALON
    - Students are expected to implement the **setModel** and **setYear** methods and complete the **displayVehicleInfo** method to correctly display the vehicle's details.

Conclusion:

This lab introduces OOP basics in C++, guiding students through class and object manipulation, constructors, destructors, and the distinction between classes and structs. These foundational skills are vital for advancing in software development with C++, preparing students for more complex programming challenges. Through hands-on tasks and examples, students will gain practical experience and a deeper understanding of OOP principles.