Lecture review

Introduction to Object Oriented Programming (OOP):

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which encapsulate data and behavior into a single unit. It provides a way to model real-world problems by representing entities as objects with properties (attributes) and methods (functions). The main aim of OOP is to bind the data and the functions that operate on them so that no other part of the code can access these data except that function.

Class in C++

A class is a blueprint or template that defines the structure and behavior of objects. It specifies:

- Attributes (data members): Variables that describe the properties of the object.
- Methods (member functions): Functions that define the actions the object can perform.

For Example: Consider the Class of Cars. There may be many cars with different names and brands, but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So, here, the car is the class, and wheels, speed limits, mileage are their properties.

```
class Car {
public:
    string brand;  // Attribute: Brand of the car
    int speed;     // Attribute: Speed of the car

    void drive() {  // Method: Behavior of the car
        cout << brand << " is driving at " << speed << " km/h." << endl;
    }
};
```

Objects

It is a basic unit of Object-Oriented Programming and represents real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated, but when it is instantiated (i.e. an object is created), memory is allocated. An object has identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know the details of each other's data or code, it is sufficient to know the type of message accepted and the type of response returned by the objects.

Example:

```
    int main() {
    Car car1;  // Object created from the Car class
    car1.brand = "Toyota";
    car1.speed = 120;

    Car car2;  // Another object from the Car class
    car2.brand = "Honda";
    car2.speed = 100;
```

```
    car1.drive();  // Output: Toyota is driving at 120 km/h.
    car2.drive();  // Output: Honda is driving at 100 km/h.

    return 0;
}
```

Constructors

A constructor is a special function in a class that is automatically called when an object is created. Its purpose is to initialize the attributes of the object. It has the same name as the class and does not have a return type.

Types of Constructors:

1. Default Constructor: A constructor with no parameters.
2. Parameterized Constructor: A constructor with parameters to initialize specific values.

Example:

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

class Car {
private:
    string brand;
    int speed;

public:
    // Default Constructor
    Car() {
        brand = "Unknown";
        speed = 0;
    }

    // Parameterized Constructor
    Car(string b, int s) {
        brand = b;
        speed = s;
    }

    void display() const {
        cout << "Brand: " << brand << ", Speed: " << speed << " km/h" << endl;
    }
};

int main() {
    // Using Default Constructor
    Car car1;
    car1.display(); // Output: Brand: Unknown, Speed: 0 km/h

    // Using Parameterized Constructor
```

```
    Car car2("Toyota", 120);
    car2.display(); // Output: Brand: Toyota, Speed: 120 km/h

    return 0;
}
```

Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class in C++. This concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding.

Encapsulation can be achieved by making *instance variables* of the class **`private`** and the access to the members is then controlled by **`public`** *member functions* of the class.

For instance, the `Student` class below has three private instance variables `erpID`, `name`, and `cgpa`.

```
class Student {
private:
    int erpID;
    string name;
    float cgpa;
public:
    Student(int id, string n, float c) { // constructor
        erpID = id;
        name = n;
        cgpa = c;
    }
    void setCGPA(float c) {    // setter
        cgpa = c;
    }
    int getErpID() {           // getter
        return erpID;
    }
    string getName() {         // getter
        return name;
    }
    float getCGPA() {          // getter
        return cgpa;
    }
};
```

The public interface of the class includes a constructor that takes three parameters and initializes the instance variables `erpID`, `name`, and `cgpa`.

There is a public member functions `setCGPA()` to set the value of the instance variable `cgpa`. Such member functions are also called *setters* or *mutators*. Note that we have not provided setter functions for `erpID` and `name`. This is because we do not want to allow the user to change these values once they are initialized in the constructor.

It also has *getter* or *accessor* functions `getErpID()`, `getName()`, and `getCGPA()` to access the values of the instance variables.

Lab exercises

Exercise 1

Create a class called `Date` that includes three pieces of information as data members—a `day` (type **`int`**), a `month` (type **`int`**) and a `year` (type **`int`**).

Your class should have a constructor with three parameters that uses the parameters to initialize the three data members. For the purpose of this exercise, assume that the values provided for the year and day are correct, but ensure that the month value is in the range 1–12; if it isn't, set the month to 1.

Provide a `set` and a `get` function for each data member.

Provide a member function `formatDate` that return the date as `string` with the month, day and year separated by forward slashes (/).

Write a test program that demonstrates class `Date`'s capabilities. Alternatively, you can use the `main()` function below.

```
int main() {
    Date d1(19, 1, 2024);
    cout << d1.formatDate() << endl; // should print 19/1/2024
    d1.setDay(17);
    cout << d1.formatDate() << endl; // should print 17/1/2024
    d1.setMonth(5);
    cout << d1.formatDate() << endl; // should print 17/5/2024

    Date d2(29, 13, 2024); // should set month to 1
    cout << d2.formatDate() << endl; // should print 29/1/2024

    return 0;
}
```

Exercise 2

Define the `WareHouse` class, which represents a warehouse. The class should contain two vectors as private members, with the first containing the products' codes and the second one their corresponding quantities. The class should contain the following functions:

- **`void`** `stock(`**`int`** `c , `**`int`** `q)` which adds the code `c` in the codes vector only if it is not already stored and the quantity `q` in the quantities vector. If the code is already stored, the function should add the quantity `q` to the existing quantity.
- **`void`** `order(`**`int`** `c, `**`int`** `q)` which corresponds to an order for the product with code `c`. The function should check if the code is valid and if there is quantity available and, if so, subtract `q` from it, otherwise a related message should be displayed.
- **`void`** `show(`**`int`** `c)` which checks if the code `c` is valid, and if yes it displays the available quantity for the respective product, otherwise a related message.
- The default constructor.

Write a program that tests the operation of the class. Alternatively, you can use the `main()` function below.

```
int main() {
    WareHouse w;
    w.stock(1, 10); // print "10 items with code 1 added"
    w.stock(2, 20); // print "20 items with code 2 added"
    w.stock(3, 30); // print "30 items with code 3 added"
    w.stock(1, 5);  // print "5 more items with code 1 added, total 15"
    w.order(1, 3);  // print "3 items delivered with code 1"
    w.order(2, 25); // print "the requested quantity is not available"
    w.order(2, 11); // print "11 items delivered with code 2"
    w.order(4, 5);  // print "Code 4 not found"
    w.show(1);      // print "12 items are stored with code 1"
    w.show(2);      // print "9 items are stored with code 2"
    w.show(3);      // print "30 items are stored with code 3"
    w.show(4);      // print "Code 4 not found"
    return 0;
}
```

Exercise 3

Define the `Book` class with private members the `title` of the book, its `code`, and its `price`. It should also contain the following public functions:

- **`void`** `review(`**`float`** `grd)` which grades the book with the grade `grd`. The `review()` can be called multiple times for the same book. The final score is the average of the grades.
- a constructor that accepts as parameters the data of the book and assigns them to the corresponding members of the object.

Add to the class any other function or member you think is needed.

The following test program declares an array of five objects of type `Book` and initializes it. It calls the `review()` of the objects few times. Then, the program read the code of a book from user and, if the book exists, the program display its score and price. If the book does not exist or it exists but has not been graded, the program display a related message.

```
int main() {
    Book books[5] = {
        Book("The C++ Programming Language", 1, 60),
        Book("The Mythical Man-month", 2, 40),
        Book("The Pragmatic Programmer: Your Journey to Mastery", 3, 50),
        Book("The Art of Computer Programming", 4, 50),
        Book("C++ For Dummies", 5, 30)
    };
    books[0].review(5);
    books[0].review(4);
    books[1].review(4);
    books[2].review(2);
    books[3].review(5);
    int code;
    cout << "Enter code: ";
    cin >> code;
    for (int i = 0; i < 5; i++) {
        if (books[i].getCode() == code) {
            if (books[i].getScore() == 0) {
                cout << "The book has not been graded yet" << endl;
            } else {
                cout << "The book has score " << books[i].getScore() << " and price " <<
                    books[i].getPrice() << endl;
            }
            return 0;
        }
    }
    cout << "The book does not exist" << endl;
    return 0;
}
```