Key Concepts

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.

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.

```cpp
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

An **object** is a basic unit of Object-Oriented Programming and represents real-life entities. An object is an instance of a class.
Example:

```cpp
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:

```cpp
#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;
}
```

Private Data and Encapsulation

Data members are often declared as `private`, which means they can only be accessed and modified through `public` methods (getters and setters). This concept is called **encapsulation**, which helps protect the internal state of objects.

Lab Questions

1. [35 marks] **Q1: Custom String Class Extension**

   Custom String Class

```cpp
class string {
public:
    string(const char* c);
    string();
    size_t size();
    void print();
private:
    size_t m_len{0};
    char* m_arr{nullptr};
};
```

   Usage Example:

```cpp
string s;
string str1 = "Hello";
str1.print();
std::cout << str1.size();
```

   Extend the given 'string' class with the following methods:

   -
     ```cpp
     bool isPalindrome();
     ```
     Description: Checks if the string reads the same backward and forward.

   -
     ```cpp
     char at(size_t index);
     ```
     Description: Returns the character at the given index.

   -
     ```cpp
     bool contains(char c);
     ```
     Description: Checks if the character exists in the string.

   -
     ```cpp
     int find(char c);
     ```
     Description: Finds the first occurrence of a character in the string.

   -
     ```cpp
     void append(const string& other);
     ```
     Description: Appends the contents of another string object to the current string.

Marks Distribution:

Correct implementation of `isPalindrome()`	5
Correct implementation of `at(size_t index)` including bounds checking	5
Correct implementation of `contains(char c)`	5
Correct implementation of `find(char c)`	5
Correct implementation of `append(const string& other)`	6
Proper handling of edge cases (empty string, single character, invalid index)	3
Clean logic and correct return types	3
Code compilation without errors	3
Total	**35**

2. [35 marks] **Student Profile Management System**

   Develop a `Student` class to manage student data with encapsulation and validation.

```
class Student {
private:
    std::string erpId;
    std::string name;
    double cgpa;
    std::string phone;
public:
    Student(std::string name, double cgpa, std::string phone);
    std::string getErpId();
    std::string getName();
    double getCgpa();
    std::string getPhone();
    void setName(std::string name);
    void setCgpa(double cgpa);
    void setPhone(std::string phone);
    void print();
};
```

Instructions:

- Implement the `Student` class as per the schema above. Ensure that `erpId` is auto-generated.
- `erpId` must not be modifiable after object creation.
- Implement getters for all attributes.
- Implement setters for `name`, `cgpa`, and `phone` with the following validation rules:
  - `cgpa` must be between 0.0 and 4.0.
  - `phone` must contain exactly 11 numeric digits.
- Implement the `print()` method to display student details in a formatted output.
- In `main()`:
  - Attempt invalid updates for `cgpa` and `phone` and show that the program rejects them.
  - Update valid values and display the final states using `print()`.

Marks Distribution:

All attributes declared `private`	3
Correct constructor implementation	4
ERP ID auto-generation	4
ERP ID immutability (no setter / cannot be changed)	4
Getters for all attributes	3
`setName()` implemented correctly	1
`setCgpa()` implemented	2
`setPhone()` implemented	2
Validation logic placed only in setters	2
CGPA range check (0.0 – 4.0)	3
Phone number exactly 11 digits numeric	2
Invalid CGPA update tested and rejected	2
Invalid phone update tested and rejected	2
Final valid data displayed using `print()`	1
Total	**35**

3. [30 marks] **Date Class**

   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`). All declared as `private`.

   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 returns the date as a string with the day, month, 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;
}
```

Marks Distribution:

Correct data members (day, month, year as private int)	4
Constructor with month validation (1–12 range)	3
Access specifiers used properly	3
`setDay(int)` implemented correctly	3
`setMonth(int)` with validation (1–12)	4
`setYear(int)` implemented correctly	2
`getDay()`, `getMonth()`, `getYear()` getters	3
`formatDate()` returns correct string format (day/month/year)	3
Date object created and `formatDate()` tested	2
Setter methods tested (day, month, year)	2
Invalid month handling tested (month > 12)	1
Total	**30**