MODULE 4) OOPS Concept

1. WAP to print “Hello World” using C++

Code:

```cpp
#include <iostream>
// Including iostream for i/o operations

using namespace std;
// using std namespace

int main() {

    cout << "Hello world!";
    // Console out hello world

    return 0;
}
```

Output:

```
/tmp/4009fepU2N.o
Hello world!
```

2. What is OOP? List OOP concepts

Ans:
OOP is known as object oriented programming, this type of programming relies more on objects and classes.

OOP Concepts:

Class:

- A blueprint or template for creating objects.

Object:

- Represents a real-world entity and encapsulates data and behavior.

Encapsulation:

- Access to the internal details of the object is controlled by access modifiers.

Inheritance:

- Promotes code reuse and establishes a hierarchy of classes.

Polymorphism:

- The ability of objects of different classes to be treated as objects of a common base class.

3. What is the difference between OOP and POP?

Ans:

Object-Oriented Programming (OOP) organizes code around objects, emphasizing encapsulation, inheritance, and polymorphism for efficient and modular design. Procedural Programming (POP) centers on procedures or functions manipulating data sequentially, with less emphasis on object-oriented concepts, making it suitable for simpler, linear tasks. OOP fosters code reuse and abstraction, while POP focuses on straightforward procedural execution.

4. WAP to create simple calculator using class

Code:

Welcome | hello_world.cpp 2 | calc.cpp 2

calc.cpp > main()

```cpp
#include <iostream>

class SimpleCalculator {
public:
    SimpleCalculator() {
        result = 0; // Initiate the resultant variable
    }

    void add(int num) {
        result += num; // Add num to the resultant
    }

    void subtract(int num) {
        result -= num; // Substract num from the resultant
    }

    void multiply(int num) {
        result *= num; // Multiply the resultant by num
    }

    void divide(int num) {
        if (num != 0) {
            result /= num; // Devide the resultant by num if num is not zero
        } else {
            std::cout << "Cannot devide by zero!" << std::endl; // Print an error message if division by zero is attempted
        }
    }

    int getResult() {
        return result; // Retun the final result
    }

private:
    int result; // Variable to store the result
};

int main() {
    SimpleCalculator calculator; // Create an instance of the SimpleCalculator class

    // Perform some calculations
    calculator.add(5);
    calculator.subtract(2);
    calculator.multiply(3);
    calculator.divide(0);

    // Display the final result
    std::cout << "Final Result: " << calculator.getResult() << std::endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Cannot devide by zero!
Final Result: 9
```

5. Define a class to represent a bank account. Include the following members::

Code:

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

class BankAccount {
private:
    string depositorName;
    long accountNumber;
    string accountType;
    double balance;

public:
    // Method to initialize the account
    void initializeAccount(const string& name, long accNumber, const string& type, double initialBalance) {
        depositorName = name;
        accountNumber = accNumber;
        accountType = type;
        balance = initialBalance;
    }

    // Method to deposit an amount
    void deposit(double amount) {
        balance += amount;
        cout << "Amount " << amount << " deposited successfully." << endl;
    }

    // Method to withdraw an amount after checking balance
    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            cout << "Amount " << amount << " withdrawn successfully." << endl;
        } else {
            cout << "Insufficient balance. Withdrawal failed." << endl;
        }
    }

    // Method to display name and balance
    void display() {
        cout << "Depositor Name: " << depositorName << endl;
        cout << "Account Number: " << accountNumber << endl;
        cout << "Account Type: " << accountType << endl;
        cout << "Balance: " << balance << endl;
    }
};

int main() {
    BankAccount myAccount;

    // Initializing account details
    myAccount.initializeAccount("John Doe", 123456789, "Savings", 1000.0);

    // Displaying initial details
    cout << "Initial Account Details:" << endl;
    myAccount.display();

    // Depositing and withdrawing amounts
    myAccount.deposit(500.0);
    myAccount.withdraw(200.0);

    // Displaying updated details
    cout << "\nUpdated Account Details:" << endl;
    myAccount.display();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Initial Account Details:
Depositor Name: John Doe
Account Number: 123456789
Account Type: Savings
Balance: 1000
Amount 500 deposited successfully.
Amount 200 withdrawn successfully.

Updated Account Details:
Depositor Name: John Doe
Account Number: 123456789
Account Type: Savings
Balance: 1300
```

6. Write a C++ program to implement a class called Circle that has private member variables for radius. Include member functions to calculate the circle's area and circumference.

Code:

```cpp
#include <iostream>

class Circle {
private:
    double radius;

public:
    // Method to set the radius
    void setRadius(double r) {
        radius = r;
    }

    // Method to calculate and return the area of the circle
    double calculateArea() {
        return 3.14159 * radius * radius;
    }

    // Method to calculate and return the circumference of the circle
    double calculateCircumference() {
        return 2 * 3.14159 * radius;
    }
};

int main() {
    Circle myCircle;

    // Set the radius
    myCircle.setRadius(5.0);

    // Calculate and display the area
    std::cout << "Area of the circle: " << myCircle.calculateArea() << std::endl;

    // Calculate and display the circumference
    std::cout << "Circumference of the circle: " << myCircle.calculateCircumference() << std::endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Area of the circle: 78.5397
Circumference of the circle: 31.4159
```

7. Write a C++ program to create a class called Rectangle that has private member variables for length and width. Implement member functions to calculate the rectangle's area and perimeter.

Code:

rect_area_perim.cpp > Rectangle

```cpp
class Rectangle {
private:
    double length;
    double width;

public:
    // Method to set the length
    void setLength(double l) {
        length = l;
    }

    // Method to set the width
    void setWidth(double w) {
        width = w;
    }

    // Method to calculate and return the area of the rectangle
    double calculateArea() {
        return length * width;
    }

    // Method to calculate and return the perimeter of the rectangle
    double calculatePerimeter() {
        return 2 * (length + width);
    }
};

int main() {
    Rectangle myRectangle;

    // Set the length and width
    myRectangle.setLength(5.0);
    myRectangle.setWidth(3.0);

    // Calculate and display the area
    cout << "Area of the rectangle: " << myRectangle.calculateArea() << endl;

    // Calculate and display the perimeter
    cout << "Perimeter of the rectangle: " << myRectangle.calculatePerimeter() << endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Area of the rectangle: 15
Perimeter of the rectangle: 16
```

8. Write a C++ program to create a class called Person that has private member variables for name, age and country. Implement member functions to set and get the values of these variables.

Code:

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

class Person {
private:
    string name;
    int age;
    string country;

public:
    // Method to set the name
    void setName(const string& n) {
        name = n;
    }

    // Method to set the age
    void setAge(int a) {
        age = a;
    }

    // Method to set the country
    void setCountry(const string& c) {
        country = c;
    }

    // Method to get the name
    string getName() const {
        return name;
    }

    // Method to get the age
    int getAge() const {
        return age;
    }

    // Method to get the country
    string getCountry() const {
        return country;
    }
};

int main() {
    Person person;

    // Set the values
    person.setName("John Doe");
    person.setAge(25);
    person.setCountry("USA");

    // Display the values
    cout << "Name: " << person.getName() << endl;
    cout << "Age: " << person.getAge() << endl;
    cout << "Country: " << person.getCountry() << endl;

    return 0;
}
```

Output:

```
/tmp/4009fepU2N.o
Name: John Doe
Age: 25
Country: USA
```

9. Write a programto find the multiplication values and the cubic values using inline function

Code:

```cpp
#include <iostream>
using namespace std;
// Inline function to calculate multiplication
inline int multiply(int a, int b) {
    return a * b;
}

// Inline function to calculate cubic value
inline int cubic(int x) {
    return x * x * x;
}

int main() {
    // Input values
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    // Calculate and display multiplication
    cout << "Multiplication of " << num1 << " and " << num2 << ": " << multiply(num1, num2) << endl;

    // Input value for cubic calculation
    int value;

    cout << "Enter a number for cubic calculation: ";
    cin >> value;

    // Calculate and display cubic value
    cout << "Cubic value of " << value << ": " << cubic(value) << endl;

    return 0;
}
```

Output:

```
Enter two numbers: 10 20
Multiplication of 10 and 20: 200
Enter a number for cubic calculation: 20 230
Cubic value of 20: 8000
```

10. Write a program of Addition, Subtraction, Division, Multiplication using constructor

Code:

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

class Calculator {
private:
    double operand1;
    double operand2;

public:
    // Constructor to initialize operands
    Calculator(double op1, double op2) : operand1(op1), operand2(op2) {}

    // Method to perform addition
    double add() {
        return operand1 + operand2;
    }

    // Method to perform subtraction
    double subtract() {
        return operand1 - operand2;
    }

    // Method to perform multiplication
    double multiply() {
        return operand1 * operand2;
    }

    // Method to perform division
    double divide() {
        if (operand2 != 0) {
            return operand1 / operand2;
        } else {
            cerr << "Error: Division by zero!" << endl;
            return 0.0;
        }
    }
};

int main() {
    // Input operands
    double num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    // Create a Calculator object with the input operands
    Calculator myCalculator(num1, num2);

    // Perform operations and display results
    cout << "Addition: " << myCalculator.add() << endl;
    cout << "Subtraction: " << myCalculator.subtract() << endl;
    cout << "Multiplication: " << myCalculator.multiply() << endl;
    cout << "Division: " << myCalculator.divide() << endl;

    return 0;
}
```

Output:

```
Enter two numbers:10 20
Addition: 20
Subtraction: -20
Multiplication: 0
Division: 0
```

11. Write a C++ program to create a class called Car that has private member variables for company, model, and year. Implement member functions to get and set these variables.

Code:

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

class Car {
private:
    string company;
    string model;
    int year;

public:
    // Method to set the company
    void setCompany(const string& c) {
        company = c;
    }

    // Method to set the model
    void setModel(const string& m) {
        model = m;
    }

    // Method to set the year
    void setYear(int y) {
        year = y;
    }

    // Method to get the company
    string getCompany() const {
        return company;
    }

    // Method to get the model
    string getModel() const {
        return model;
    }

    // Method to get the year
    int getYear() const {
        return year;
    }
};

int main() {
    // Create a Car object
    Car myCar;

    // Set the car details
    myCar.setCompany("Toyota");
    myCar.setModel("Camry");
    myCar.setYear(2022);

    // Display the car details
    cout << "Car Details:" << endl;
    cout << "Company: " << myCar.getCompany() << endl;
    cout << "Model: " << myCar.getModel() << endl;
    cout << "Year: " << myCar.getYear() << endl;

    return 0;
}
```

Output:

```
/tmp/4009fepU2N.o
Car Details:
Company: Toyota
Model: Camry
Year: 2022
```

12. Write a C++ program to implement a class called Bank Account that has private member variables for account number and balance. Include member functions to deposit and withdraw money from the account

Code:

```
bank_depo.cpp > ...
#include <iostream>
using namespace std;

class BankAccount {
private:
    int accountNumber;
    double balance;

public:
    // Constructor to initialize account details
    BankAccount(int accNumber, double initialBalance) : accountNumber(accNumber), balance(initial

    // Method to deposit money into the account
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            cout << "Deposit successful. New balance: " << balance << endl;
        } else {
            cerr << "Error: Invalid deposit amount." << endl;
        }
    }

    // Method to withdraw money from the account
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            cout << "Withdrawal successful. New balance: " << balance << endl;
        } else {
            cerr << "Error: Invalid withdrawal amount or insufficient funds." << endl;
        }
    }

    // Method to get the current balance
    double getBalance() const {
        return balance;
    }
};

int main() {
    // Create a BankAccount object with an initial balance of $1000
    BankAccount myAccount(123456, 1000.0);

    // Deposit and withdraw money
    myAccount.deposit(500.0);
    myAccount.withdraw(200.0);
    myAccount.withdraw(1500.0);  // Invalid withdrawal due to insufficient funds

    // Display the final balance
    cout << "Final Balance: $" << myAccount.getBalance() << endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
ERROR!
Deposit successful. New balance: 1500
Withdrawal successful. New balance: 1300
Error: Invalid withdrawal amount or insufficient funds.
Final Balance: $1300
```

13. Write a C++ program to create a class called Triangle that has private member variables for the lengths of its three sides. Implement member functions to determine if the triangle is equilateral, isosceles, or scalene.

Code:

traingle.cpp > Triangle

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

class Triangle {
private:
    double side1;
    double side2;
    double side3;

public:
    // Constructor to initialize the sides of the triangle
    Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s

    // Method to determine if the triangle is equilateral
    bool isEquilateral() const {
        return side1 == side2 && side2 == side3;
    }

    // Method to determine if the triangle is isosceles
    bool isIsosceles() const {
        return side1 == side2 || side1 == side3 || side2 == side3;
    }

    // Method to determine if the triangle is scalene
    bool isScalene() const {
        return side1 != side2 && side2 != side3 && side1 != side3;
    }
};

int main() {
    // Create a Triangle object with sides 3, 4, and 5
    Triangle myTriangle(3.0, 4.0, 5.0);

    // Check the type of the triangle
    if (myTriangle.isEquilateral()) {
        cout << "The triangle is equilateral." << endl;
    } else if (myTriangle.isIsosceles()) {
        cout << "The triangle is isosceles." << endl;
    } else if (myTriangle.isScalene()) {
        cout << "The triangle is scalene." << endl;
    }

    return 0;
}
```

Output:

```
/tmp/4009fepU2N.o
The triangle is scalene.
```

14. Write a C++ program to implement a class called Employee that has private member variables for name, employee ID, and salary. Include member functions to calculate and set salary based on employee performance. Using of constructor

Code:

```cpp
using namespace std;

class Employee {
private:
    string name;
    int employeeID;
    double salary;

public:
    // Constructor to initialize employee details
    Employee(const string& empName, int empID, double initialSalary) : name(empName), employeeID(empID), salary(initialSalary) {}

    // Method to set the salary based on performance
    void setSalaryBasedOnPerformance(double performanceFactor) {
        // Assume performanceFactor is a percentage (e.g., 10.0 for a 10% increase)
        if (performanceFactor > 0) {
            salary += (salary * performanceFactor) / 100.0;
            cout << "Salary updated based on performance. New salary: $" << salary << endl;
        } else {
            cerr << "Error: Invalid performance factor." << endl;
        }
    }

    // Method to get the employee name
    string getName() const {
        return name;
    }

    // Method to get the employee ID
    int getEmployeeID() const {
        return employeeID;
    }

    // Method to get the employee salary
    double getSalary() const {
        return salary;
    }
};

int main() {
    // Create an Employee object with initial details
    Employee emp("John Doe", 123456, 50000.0);

    // Display initial employee details
    cout << "Employee Details:" << endl;
    cout << "Name: " << emp.getName() << endl;
    cout << "Employee ID: " << emp.getEmployeeID() << endl;
    cout << "Salary: $" << emp.getSalary() << endl;

    // Update salary based on performance (10% increase)
    emp.setSalaryBasedOnPerformance(10.0);

    // Display updated employee details
    cout << "\nUpdated Employee Details:" << endl;
    cout << "Name: " << emp.getName() << endl;
    cout << "Employee ID: " << emp.getEmployeeID() << endl;
    cout << "Updated Salary: $" << emp.getSalary() << endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Employee Details:
Name: John Doe
Employee ID: 123456
Salary: $50000
Salary updated based on performance. New salary: $55000

Updated Employee Details:
Name: John Doe
Employee ID: 123456
Updated Salary: $55000
```

15. Write a C++ program to implement a class called Date that has private member variables for day, month, and year. Include member functions to set and get these variables, as well as to validate if the date is valid

Code:

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

class Date {
private:
    int day;
    int month;
    int year;

public:
    // Constructor to initialize date details
    Date(int d, int m, int y) : day(d), month(m), year(y) {}

    // Method to set the date
    void setDate(int d, int m, int y) {
        day = d;
        month = m;
        year = y;
    }

    // Method to get the day
    int getDay() const {
        return day;
    }

    // Method to get the month
    int getMonth() const {
        return month;
    }

    // Method to get the year
    int getYear() const {
        return year;
    }

    // Method to validate if the date is valid
    bool isValidDate() const {
        // Simple validation for demonstration purposes
        return (day >= 1 && day <= 31) && (month >= 1 && month <= 12) && (year >= 1900);
    }
};

int main() {
    // Create a Date object with initial details
    Date myDate(15, 3, 2022);

    // Display initial date details
    std::cout << "Date Details:" << std::endl;
    std::cout << "Day: " << myDate.getDay() << std::endl;
    std::cout << "Month: " << myDate.getMonth() << std::endl;
    std::cout << "Year: " << myDate.getYear() << std::endl;

    // Validate if the date is valid
    if (myDate.isValidDate()) {
        std::cout << "The date is valid." << std::endl;
    } else {
        std::cerr << "Error: The date is not valid." << std::endl;
    }

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Date Details:
Day: 15
Month: 3
Year: 2022
The date is valid.
```

16. Write a C++ program to implement a class called Student that has private member variables for name, class, roll number, and marks. Include member functions to calculate the grade based on the marks and display the student's information. Accept address from each student implement using of aggregation

Code:

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

class Address {
public:
    string street;
    string city;
    string state;
    int zipCode;

    // Constructor for Address
    Address(const string& s, const string& c, const string& st, int zip)
        : street(s), city(c), state(st), zipCode(zip) {}
};

class Student {
private:
    string name;
    string className;
    int rollNumber;
    double marks;
    Address address;  // Aggregation: Student has an Address

public:
    // Constructor for Student
    Student(const string& n, const string& cls, int roll, double m, const Address& addr)
        : name(n), className(cls), rollNumber(roll), marks(m), address(addr) {}

    // Method to calculate grade based on marks
    char calculateGrade() const {
        if (marks >= 90.0) {
            return 'A';
        } else if (marks >= 80.0) {
            return 'B';
        } else if (marks >= 70.0) {
            return 'C';
        } else if (marks >= 60.0) {
            return 'D';
        } else {
            return 'F';
        }
    }

    // Method to display student information
    void displayStudentInfo() const {
        cout << "Student Information:" << endl;
        cout << "Name: " << name << endl;
        cout << "Class: " << className << endl;
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Marks: " << marks << endl;
        cout << "Grade: " << calculateGrade() << endl;
        cout << "Address: " << address.street << ", " << address.city << ", " << address.state << " " << address.zipCode << endl;
    }
};

int main() {
    // Create an Address for each student
    Address address1("123 Main St", "Cityville", "Stateville", 12345);
    Address address2("456 Oak St", "Townville", "Stateville", 67890);

    // Create two Student objects with the respective addresses
    Student student1("John Doe", "10th", 101, 92.5, address1);
    Student student2("Jane Smith", "12th", 202, 78.0, address2);

    // Display student information
    student1.displayStudentInfo();
    cout << endl;
    student2.displayStudentInfo();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Student Information:
Name: John Doe
Class: 10th
Roll Number: 101
Marks: 92.5
Grade: A
Address: 123 Main St, Cityville, Stateville 12345

Student Information:
Name: Jane Smith
Class: 12th
Roll Number: 202
Marks: 78
Grade: C
Address: 456 Oak St, Townsville, Stateville 67890
```

17. Assume a class cricketer is declared. Declare a derived class batsman from cricketer. Data member of batsman. Total runs, Average runs and best performance. Member functions input data, calculate average runs, Display data. (Single Inheritance)

Code:

```
cricketer.cpp / ...
#include <iostream>
#include <string>

class Cricketer {
protected:
    std::string name;
    int age;

public:
    // Method to input data for Cricketer
    void inputCricketerData() {
        std::cout << "Enter Cricketer's name: ";
        std::cin >> name;
        std::cout << "Enter Cricketer's age: ";
        std::cin >> age;
    }

    // Method to display Cricketer data
    void displayCricketerData() const {
        std::cout << "Cricketer Information:" << std::endl;
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

class Batsman : public Cricketer {
private:
    int totalRuns;
    double averageRuns;
    int bestPerformance;

public:
    // Method to input data for Batsman
    void inputBatsmanData() {
        inputCricketerData();  // Input basic cricketer data using the base class method
        std::cout << "Enter total runs: ";
        std::cin >> totalRuns;
        std::cout << "Enter best performance: ";
        std::cin >> bestPerformance;

        // Calculate average runs
        if (totalRuns > 0) {
            averageRuns = static_cast<double>(totalRuns) / 5.0; // Assuming 5 matches for simplicity
        } else {
            averageRuns = 0.0;
        }
    }

    // Method to display Batsman data
    void displayBatsmanData() const {
        displayCricketerData(); // Display basic cricketer data using the base class method
        std::cout << "Total Runs: " << totalRuns << std::endl;
        std::cout << "Average Runs: " << averageRuns << std::endl;
        std::cout << "Best Performance: " << bestPerformance << std::endl;
    }
};

int main() {
    // Create a Batsman object
    Batsman batsman;

    // Input and display Batsman data
    batsman.inputBatsmanData();
    std::cout << "\nBatsman Information:" << std::endl;
    batsman.displayBatsmanData();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Enter Cricketer's name: Sachin
Enter Cricketer's age: 51
Enter total runs: 12891
Enter best performance: 189
Batsman Information:
Cricketer Information:
Name: Sachin
Age: 51
Total Runs: 12891
Average Runs: 2578.2
Best Performance: 189
```

18. Write a C++ Program to find Area of Rectangle using inheritance

Code:

```cpp
#include <iostream>

using namespace std;

class Shape {
protected:
    double width;
    double height;

public:
    // Constructor for Shape
    Shape(double w, double h) : width(w), height(h) {}

    // Method to calculate the area (to be overridden by derived classes)
    virtual double calculateArea() const {
        return 0.0; // Default implementation, to be overridden
    }
};

class Rectangle : public Shape {
public:
    // Constructor for Rectangle
    Rectangle(double w, double h) : Shape(w, h) {}

    // Override method to calculate the area for Rectangle
    double calculateArea() const override {
        return width * height;
    }
};

int main() {
    // Create a Rectangle object
    Rectangle myRectangle(5.0, 8.0);

    // Calculate and display the area of the Rectangle
    double area = myRectangle.calculateArea();
    cout << "Area of the Rectangle: " << area << endl;

    return 0;
}
```

Output:

```
Area of the Rectangle: 40
```

19. Create a class person having members name and age. Derive a class student having member percentage. Derive another class teacher having member salary. Write necessary member function to initialize, read and write data. Write also Main function (Multiple Inheritance)

Code:

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

class Person {
protected:
    string name;
    int age;

public:
    // Constructor to initialize Person data
    Person(const string& n, int a) : name(n), age(a) {}

    // Function to read Person data
    void readPersonData() {
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter age: ";
        cin >> age;
    }

    // Function to display Person data
    void displayPersonData() const {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

class Student : public Person {
protected:
    double percentage;

public:
    // Constructor to initialize Student data
    Student(const string& n, int a, double p) : Person(n, a), percentage(p) {}

    // Function to read Student data
    void readStudentData() {
        readPersonData(); // Call base class function to read common data
        cout << "Enter percentage: ";
        cin >> percentage;
    }

    // Function to display Student data
    void displayStudentData() const {
        displayPersonData(); // Call base class function to display common data
        cout << "Percentage: " << percentage << "%" << endl;
    }
};

class Teacher : public Person {
protected:
    double salary;

public:
    // Constructor to initialize Teacher data
    Teacher(const string& n, int a, double s) : Person(n, a), salary(s) {}

    // Function to read Teacher data
    void readTeacherData() {
        readPersonData(); // Call base class function to read common data
        cout << "Enter salary: ";
        cin >> salary;
    }

    // Function to display Teacher data
    void displayTeacherData() const {
        displayPersonData(); // Call base class function to display common data
        cout << "Salary: $" << salary << endl;
    }
};

int main() {
    // Create a Student object
    Student myStudent("John Doe", 20, 85.5);

    // Display Student data
    cout << "Student Information:" << endl;
    myStudent.displayStudentData();
    cout << endl;

    // Create a Teacher object
    Teacher myTeacher("Jane Smith", 30, 50000.0);

    // Display Teacher data
    cout << "Teacher Information:" << endl;
    myTeacher.displayTeacherData();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Student Information:
Name: John Doe
Age: 20
Percentage: 85.5%

Teacher Information:
Name: Jane Smith
Age: 30
Salary: $50000
```

20. Write a C++ Program display Student Mark sheet using Multiple inheritance

Code:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Base class for student information
class StudentInfo {
protected:
    string name;
    int rollNumber;

public:
    // Constructor to initialize student information
    StudentInfo(const string& n, int roll) : name(n), rollNumber(roll) {}

    // Function to read student information
    void readStudentInfo() {
        cout << "Enter student name: ";
        cin >> name;
        cout << "Enter roll number: ";
        cin >> rollNumber;
    }

    // Function to display student information
    void displayStudentInfo() const {
        cout << "Student Name: " << name << endl;
        cout << "Roll Number: " << rollNumber << endl;
    }
};

// Derived class for exam details
class Exam : public StudentInfo {
protected:
    int marks;

public:
    // Constructor to initialize exam details
    Exam(const string& n, int roll, int m) : StudentInfo(n, roll), marks(m) {}

    // Function to read exam details
    void readExamDetails() {
        readStudentInfo(); // Call base class function to read student information
        cout << "Enter marks obtained: ";
        cin >> marks;
    }

    // Function to display exam details
    void displayExamDetails() const {
        displayStudentInfo(); // Call base class function to display student information
        cout << "Marks Obtained: " << marks << endl;
    }
};

// Derived class for result calculations
class Result : public Exam {
private:
    int total;
    double percentage;

public:
    // Constructor to initialize result details
    Result(const string& n, int roll, int m) : Exam(n, roll, m) {
        calculateResult();
    }

    // Function to calculate result details
    void calculateResult() {
        total = marks; // For simplicity, total is considered the same as marks
        percentage = static_cast<double>(marks) / 100.0 * 100.0; // Assuming total marks are 100 for simplicity
    }

    // Function to display result details
    void displayResult() const {
        displayExamDetails(); // Call base class function to display exam details
        cout << "Total Marks: " << total << endl;
        cout << "Percentage: " << percentage << "%" << endl;
    }
};

int main() {
    // Create a Result object
    Result studentResult("John Doe", 101, 75);

    // Display student's mark sheet
    cout << "Student's Mark Sheet:" << endl;
    studentResult.displayResult();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Student's Mark Sheet:
Student Name: John Doe
Roll Number: 101
Marks Obtained: 75
Total Marks: 75
Percentage: 75%
```

21. Assume that the test results of a batch of students are stored in three different classes. Class Students are storing the roll number. Class Test stores the marks obtained in two subjects and class result contains the total marks obtained in the test. The class result can inherit the details of the marks obtained in the test and roll number of students. (Multilevel Inheritance)

Code:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Base class for student information
class Student {
protected:
    int rollNumber;

public:
    // Constructor to initialize student information
    Student(int roll) : rollNumber(roll) {}

    // Function to read student information
    void readStudentInfo() {
        cout << "Enter student roll number: ";
        cin >> rollNumber;
    }

    // Function to display student information
    void displayStudentInfo() const {
        cout << "Student Roll Number: " << rollNumber << endl;
    }
};

// Derived class for test details
class Test : public Student {
protected:
    int marksSubject1;
    int marksSubject2;

public:
    // Constructor to initialize test details
    Test(int roll, int marks1, int marks2) : Student(roll), marksSubject1(marks1), marksSubject2(marks2) {}

    // Function to read test details
    void readTestDetails() {
        readStudentInfo(); // call base class function to read student information
        cout << "Enter marks obtained in Subject 1: ";
        cin >> marksSubject1;
        cout << "Enter marks obtained in Subject 2: ";
        cin >> marksSubject2;
    }

    // Function to display test details
    void displayTestDetails() const {
        displayStudentInfo(); // call base class function to display student information
        cout << "Marks Obtained in Subject 1: " << marksSubject1 << endl;
        cout << "Marks Obtained in Subject 2: " << marksSubject2 << endl;
    }
};

// Derived class for result calculations
class Result : public Test {
private:
    int totalMarks;

public:
    // Constructor to initialize result details
    Result(int roll, int marks1, int marks2) : Test(roll, marks1, marks2) {
        calculateTotalMarks();
    }

    // Function to calculate total marks
    void calculateTotalMarks() {
        totalMarks = marksSubject1 + marksSubject2;
    }

    // Function to display result details
    void displayResult() const {
        displayTestDetails(); // call base class function to display test details
        cout << "Total Marks Obtained: " << totalMarks << endl;
    }
};

int main() {
    // Create a Result object
    Result studentResult(101, 75, 85);

    // Display student's result
    cout << "Student's Result:" << endl;
    studentResult.displayResult();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Student's Result:
Student Roll Number: 101
Marks Obtained in Subject 1: 75
Marks Obtained in Subject 2: 85
Total Marks Obtained: 160
```

22. Write a C++ Program to show access to Private Public and Protected using Inheritance

Code:

```cpp
#include <iostream>

using namespace std;

// Base class
class Base {
private:
    int privateMemberBase;

protected:
    int protectedMemberBase;

public:
    int publicMemberBase;

    // Constructor to initialize members
    Base(int privateVal, int protectedVal, int publicVal)
        : privateMemberBase(privateVal), protectedMemberBase(protectedVal), publicMemberBase(publicVal) {}

    // Function to display values
    void displayBase() {
        cout << "Private Member (Base): Not Accessible" << endl;
        cout << "Protected Member (Base): " << protectedMemberBase << endl;
        cout << "Public Member (Base): " << publicMemberBase << endl;
    }
};

// Derived class
class Derived : public Base {
public:
    // Constructor to initialize members
    Derived(int privateVal, int protectedVal, int publicVal)
        : Base(privateVal, protectedVal, publicVal) {}

    // Function to display values
    void displayDerived() {
        // Private member of the base class is not accessible in the derived class
        // cout << "Private Member (Base) in Derived: " << privateMemberBase << endl;

        // Protected and public members are accessible
        cout << "Protected Member (Base) in Derived: " << protectedMemberBase << endl;
        cout << "Public Member (Base) in Derived: " << publicMemberBase << endl;
    }
};

int main() {
    // Create an object of the derived class
    Derived derivedObject(1, 2, 3);

    // Access and display members from the derived class
    cout << "Accessing Members in Derived Class:" << endl;
    derivedObject.displayDerived();

    // Access and display members from the base class through the derived class
    cout << "\nAccessing Members in Base Class through Derived Class:" << endl;
    derivedObject.displayBase();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Accessing Members in Derived Class:
Protected Member (Base) in Derived: 2
Public Member (Base) in Derived: 3

Accessing Members in Base Class through Derived Class:
Private Member (Base): Not Accessible
Protected Member (Base): 2
Public Member (Base): 3
```

23. Write a C++ Program to illustrates the use of Constructors in multilevel inheritance

Code:

```cpp
#include <iostream>

using namespace std;

// Base class
class Base {
protected:
    int baseValue;

public:
    // Constructor to initialize baseValue
    Base(int value) : baseValue(value) {
        cout << "Base Constructor called with value: " << baseValue << endl;
    }
};

// Derived class 1
class Derived1 : public Base {
protected:
    int derived1Value;

public:
    // Constructor to initialize derived1Value
    Derived1(int value, int valueDerived1) : Base(value), derived1Value(valueDerived1) {
        cout << "Derived1 Constructor called with value: " << derived1Value << endl;
    }
};

// Derived class 2
class Derived2 : public Derived1 {
private:
    int derived2Value;

public:
    // Constructor to initialize derived2Value
    Derived2(int value, int valueDerived1, int valueDerived2) : Derived1(value, valueDerived1), derived2Value(valueDerived2) {
        cout << "Derived2 Constructor called with value: " << derived2Value << endl;
    }

    // Function to display values
    void displayValues() {
        cout << "Base Value: " << baseValue << endl;
        cout << "Derived1 Value: " << derived1Value << endl;
        cout << "Derived2 Value: " << derived2Value << endl;
    }
};

int main() {
    // Create an object of Derived2
    Derived2 derived2Object(1, 2, 3);

    // Display values using a member function
    cout << "\nDisplaying Values using a Member Function:" << endl;
    derived2Object.displayValues();

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Base Constructor called with value: 1
Derived1 Constructor called with value: 2
Derived2 Constructor called with value: 3

Displaying Values using a Member Function:
Base Value: 1
Derived1 Value: 2
Derived2 Value: 3
```

24. Write a program to Mathematic operation like Addition, Subtraction, Multiplication, Division Of two number using different parameters and Function Overloading

Code:

```cpp
#include <iostream>

using namespace std;

// Function to perform addition
int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

// Function to perform subtraction
int subtract(int a, int b) {
    return a - b;
}

double subtract(double a, double b) {
    return a - b;
}

// Function to perform multiplication
int multiply(int a, int b) {
    return a * b;
}

double multiply(double a, double b) {
    return a * b;
}

// Function to perform division
int divide(int a, int b) {
    if (b != 0) {
        return a / b;
    } else {
        cerr << "Error: Division by zero!" << endl;
        return 0;
    }
}

double divide(double a, double b) {
    if (b != 0.0) {
        return a / b;
    } else {
        cerr << "Error: Division by zero!" << endl;
        return 0.0;
    }
}

int main() {
    int intNum1, intNum2;
    double doubleNum1, doubleNum2;

    // Input for integers
    cout << "Enter two integers: ";
    cin >> intNum1 >> intNum2;

    // Input for doubles
    cout << "Enter two doubles: ";
    cin >> doubleNum1 >> doubleNum2;

    // Perform integer operations
    cout << "\nInteger Operations:" << endl;
    cout << "Addition: " << add(intNum1, intNum2) << endl;
    cout << "Subtraction: " << subtract(intNum1, intNum2) << endl;
    cout << "Multiplication: " << multiply(intNum1, intNum2) << endl;
    cout << "Division: " << divide(intNum1, intNum2) << endl;

    // Perform double operations
    cout << "\nDouble Operations:" << endl;
    cout << "Addition: " << add(doubleNum1, doubleNum2) << endl;
    cout << "Subtraction: " << subtract(doubleNum1, doubleNum2) << endl;
    cout << "Multiplication: " << multiply(doubleNum1, doubleNum2) << endl;
    cout << "Division: " << divide(doubleNum1, doubleNum2) << endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Enter two integers: 20 100
Enter two doubles: 20 20
Integer Operations:
Addition: 120
Subtraction: -80
Multiplication: 2000
Division: 0

Double Operations:
Addition: 40
Subtraction: 0
Multiplication: 400
Division: 1
```

25. Write a program to concatenate the two strings using Operator Overloading

Code:

```cpp
#include <iostream>

using namespace std;

// Class to represent a concatenated string
class ConcatenatedString {
private:
    char* concatenatedString;

public:
    // Constructor to initialize the concatenated string
    ConcatenatedString(const char* str1, const char* str2) {
        int length1 = 0;
        while (str1[length1] != '\0') {
            length1++;
        }

        int length2 = 0;
        while (str2[length2] != '\0') {
            length2++;
        }

        concatenatedString = new char[length1 + length2 + 1]; // +1 for null terminator

        // Copy the characters from the first string
        for (int i = 0; i < length1; ++i) {
            concatenatedString[i] = str1[i];
        }

        // Append the characters from the second string
        for (int i = 0; i < length2; ++i) {
            concatenatedString[length1 + i] = str2[i];
        }

        // Null-terminate the concatenated string
        concatenatedString[length1 + length2] = '\0';
    }

    // Destructor to free memory
    ~ConcatenatedString() {
        delete[] concatenatedString;
    }

    // Function to display the concatenated string
    void displayConcatenatedString() const {
        cout << "Concatenated String: " << concatenatedString << endl;
    }
};

int main() {
    // Input two strings
    char str1[50], str2[50];

    cout << "Enter the first string: ";
    cin.getline(str1, sizeof(str1));

    cout << "Enter the second string: ";
    cin.getline(str2, sizeof(str2));

    // Create a ConcatenatedString object
    ConcatenatedString concatenated(str1, str2);

    // Display the concatenated string
    concatenated.displayConcatenatedString();

    return 0;
}
```

Output:

```
Enter the first string: hello
Enter the second string: world
Concatenated String: helloworld
```

26. Write a program to calculate the area of circle, rectangle and triangle using Function Overloading

Code:

```
#include <iostream>

using namespace std;

const double PI = 3.14159265358979323846;

// Function to calculate the area of a rectangle
double calculateAreaRectangle(double length, double breadth) {
    return length * breadth;
}

// Function to calculate the area of a triangle
double calculateAreaTriangle(double base, double height) {
    return 0.5 * base * height;
}

// Function to calculate the area of a circle
double calculateAreaCircle(double radius) {
    return PI * radius * radius;
}

int main() {
    double length, breadth, base, height, radius;

    // Input for rectangle
    cout << "Enter length and breadth of the rectangle: ";
    cin >> length >> breadth;

    // Calculate and display the area of the rectangle
    cout << "Area of Rectangle: " << calculateAreaRectangle(length, breadth) << endl;

    // Input for triangle
    cout << "\nEnter base and height of the triangle: ";
    cin >> base >> height;

    // Calculate and display the area of the triangle
    cout << "Area of Triangle: " << calculateAreaTriangle(base, height) << endl;

    // Input for circle
    cout << "\nEnter radius of the circle: ";
    cin >> radius;

    // Calculate and display the area of the circle
    cout << "Area of Circle: " << calculateAreaCircle(radius) << endl;

    return 0;
}
```

Output:

```
Enter length and breadth of the rectangle: 10 20
Area of Rectangle: 200

Enter base and height of the triangle: 20 1
Area of Triangle: 10

Enter radius of the circle: 219
Area of Circle: 150674
```

27. Write a programto swap the two numbers using friend function without using third variable

Code:

```cpp
#include <iostream>

using namespace std;

class NumberSwapper {
private:
    int num1;
    int num2;

public:
    // Constructor to initialize the numbers
    NumberSwapper(int a, int b) : num1(a), num2(b) {}

    // Friend function to swap the numbers without using a third variable
    friend void swapNumbers(NumberSwapper& ns);

    // Function to display the numbers
    void displayNumbers() const {
        cout << "After swapping: num1 = " << num1 << ", num2 = " << num2 << endl;
    }
};

// Friend function definition to swap the numbers without using a third variable
void swapNumbers(NumberSwapper& ns) {
    ns.num1 = ns.num1 + ns.num2;
    ns.num2 = ns.num1 - ns.num2;
    ns.num1 = ns.num1 - ns.num2;
}

int main() {
    int a, b;

    // Input two numbers
    cout << "Enter the first number: ";
    cin >> a;

    cout << "Enter the second number: ";
    cin >> b;

    // Create a NumberSwapper object
    NumberSwapper numbers(a, b);

    // Display the numbers before swapping
    cout << "Before swapping: num1 = " << a << ", num2 = " << b << endl;

    // Swap the numbers using the friend function
    swapNumbers(numbers);

    // Display the numbers after swapping
    numbers.displayNumbers();

    return 0;
}
```

Output:

```
Enter the first number: 1
Enter the second number: 2
Before swapping: num1 = 1, num2 = 2
After swapping: num1 = 2, num2 = 1
```

28. Write a program to find the max number from given two numbers using friend function

Code:

```cpp
#include <iostream>

using namespace std;

class MaxFinder {
private:
    int num1;
    int num2;

public:
    // Constructor to initialize the numbers
    MaxFinder(int a, int b) : num1(a), num2(b) {}

    // Friend function to find the maximum number
    friend int findMax(const MaxFinder& mf);

    // Function to display the numbers
    void displayNumbers() const {
        cout << "Numbers: num1 = " << num1 << ", num2 = " << num2 << endl;
    }
};

// Friend function definition to find the maximum number
int findMax(const MaxFinder& mf) {
    return (mf.num1 > mf.num2) ? mf.num1 : mf.num2;
}

int main() {
    int a, b;

    // Input two numbers
    cout << "Enter the first number: ";
    cin >> a;

    cout << "Enter the second number: ";
    cin >> b;

    // Create a MaxFinder object
    MaxFinder numbers(a, b);

    // Display the numbers
    numbers.displayNumbers();

    // Find and display the maximum number using the friend function
    cout << "Maximum Number: " << findMax(numbers) << endl;

    return 0;
}
```

Output:

```
Enter the first number: 20
Enter the second number: 30
Numbers: num1 = 20, num2 = 30
Maximum Number: 30
```

29. Write a program of to swap the two values using template

Code:

```
#include <iostream>

using namespace std;

// Template function to swap two values
template <typename T>
void swapValues(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    // Swap integers
    int num1 = 5, num2 = 10;
    cout << "Before swapping: num1 = " << num1 << ", num2 = " << num2 << endl;
    swapValues(num1, num2);
    cout << "After swapping: num1 = " << num1 << ", num2 = " << num2 << endl;

    // Swap doubles
    double double1 = 3.14, double2 = 2.71;
    cout << "\nBefore swapping: double1 = " << double1 << ", double2 = " << double2 << endl;
    swapValues(double1, double2);
    cout << "After swapping: double1 = " << double1 << ", double2 = " << double2 << endl;

    // Swap characters
    char char1 = 'A', char2 = 'B';
    cout << "\nBefore swapping: char1 = " << char1 << ", char2 = " << char2 << endl;
    swapValues(char1, char2);
    cout << "After swapping: char1 = " << char1 << ", char2 = " << char2 << endl;

    return 0;
}
```

Output:

```
/tmp/4O09fepU2N.o
Before swapping: num1 = 5, num2 = 10
After swapping: num1 = 10, num2 = 5

Before swapping: double1 = 3.14, double2 = 2.71
After swapping: double1 = 2.71, double2 = 3.14

Before swapping: char1 = A, char2 = B
After swapping: char1 = B, char2 = A
```

30. Write a program of to sort the array using templates

Code:

```
#include <iostream>

using namespace std;

// Template function to perform bubble sort on an array
template <typename T>
void bubbleSort(T arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                T temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Template function to display elements of an array
template <typename T>
void displayArray(const T arr[], int size) {
    cout << "Sorted Array: ";
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    // Sort integers
    int intArr[] = {5, 2, 8, 1, 3};
    int intSize = sizeof(intArr) / sizeof(int);
    cout << "Original Array: ";
    for (int i = 0; i < intSize; ++i) {
        cout << intArr[i] << " ";
    }
    cout << endl;

    bubbleSort(intArr, intSize);
    displayArray(intArr, intSize);

    // Sort doubles
    double doubleArr[] = {3.14, 1.0, 2.71, 0.5, 2.0};
    int doubleSize = sizeof(doubleArr) / sizeof(double);
    cout << "\nOriginal Array: ";
    for (int i = 0; i < doubleSize; ++i) {
        cout << doubleArr[i] << " ";
    }
    cout << endl;

    bubbleSort(doubleArr, doubleSize);
    displayArray(doubleArr, doubleSize);

    return 0;
}
```

Output:

```
Original Array: 5 2 8 1 3
Sorted Array: 1 2 3 5 8

Original Array: 3.14 1 2.71 0.5 2
Sorted Array: 0.5 1 2 2.71 3.14
```