Software Engineering Assignment

Module : 4.1 (C++ Basic)

❖ WAP to print “Hello World” using C++

```
#include<iostream>
using namespace std;
main()
{
        cout<<"Hello World..";
}
```

❖ What is OOP? List OOP concepts

OOPs, Object Oriented Programming is a programming pattern where the programs are structured around objects rather than functions and logic. It makes the data partitioned into two memory areas, i.e., data and functions, and helps make the code flexible and modular.

Object-oriented programming mainly focuses on objects that are required to be manipulated. In OOPs, it can represent data as objects that have attributes and functions.

Classes:

Class can be defined as a blueprint of the object. It is basically a collection of objects which act as building blocks.

A class contains data members (variables) and member functions. These member functions are used to manipulate the data members inside the class.

Objects:

An Object can be defined as an entity that has a state and behaviour, or in other words, anything that exists physically in the world is called an object.

It can represent a dog, a person, a table, etc. An object means a combination of data and programs, which further represent an entity.

Inheritance:

Inheritance is the process in which two classes have an is-a relationship among each other and objects of one class acquire properties and features of the other class.

The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class.

Encapsulation:

The wrapping up of data and functions together in a single unit is known as encapsulation. It can be achieved by making the data members' scope private and the member function's scope public to access these data members.

Abstraction:

Abstraction helps in the data-hiding process. It helps in displaying the essential features without showing the details or the functionality to the user.

Polymorphism:

Polymorphism means many forms. It is the ability to take more than one form. It is a feature that provides a function or an operator with more than one definition. It can be implemented using function overloading, operator overload, function overriding, and virtual functions.

❖ What is the difference between OOP and POP?

OOP	POP
• OOP (object-oriented programming languages)	• POP (procedural programming language)
• It mainly focuses on creating objects to represent real-world entities.	• It mainly focuses on creating a series of procedures or functions to perform tasks
• It includes the concept of encapsulation, inheritance, and polymorphism	• It does not include the concept of encapsulation, inheritance, and polymorphism
• It supports code reusability modularity and flexibility.	• It Emphasizes simplicity, clarity, and speed

• It is used for larger, more complex projects with a lot of data to manage	• It is used for smaller, simpler projects with fewer data management requirements
• Java, Python, C++, etc.	• C, Pascal, and Fortran

Module : 4.1 (C++ Basic)

❖ WAP to create simple calculator using class

```
#include<iostream>
#include<math.h>
using namespace std;
class calculator
{
        float a,b;
        public:
                void getval()
                {
                        cout<<"Enter First Number : ";
                        cin>>a;
                        cout<<"Enter Second Number : ";
                        cin>>b;
                }
                float add()
                {
                        return a+b;
                }
                float sub()
                {
                        return a-b;
                }
                float mul()
                {
                        return a*b;
                }
```

```
        float div()
        {
            return a/b;
        }
};

int main()
{
    int n;
    calculator c;
    cout<<"Enter 1 to Addition 2 Numbers" <<endl<<
    "Enter 2 to Subtract 2 Numbers" <<endl<<
    "Enter 3 to Multiply 2 Numbers" <<endl<<
    "Enter 4 to Divide 2 Numbers" <<endl<<
    "Enter 0 To Exit"<<endl;
  do
  {
    cout<<"Enter your choice : ";
    cin>>n;


    switch(n)
    {
        case 1:
            c.getval();
            cout<<"result = "<<c.add()<<endl;
            break;
        case 2:
            c.getval();
            cout<<"result = "<<c.sub()<<endl;
        break;
    case 3:
        c.getval();
        cout<<"result = "<<c.mul()<<endl;
        break;
    case 4:
        c.getval();
        cout<<"result = "<<c.div()<<endl;
        break;
        }
    }while(n<=4 && n>=1);

    return 0;
}
```

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

➢ 1. Data Member:

- -Name of the depositor
- -Account Number
- -Type of Account
- -Balance amount in the account

➢ 2. Member Functions

- -To assign values
- -To deposited an amount
- -To withdraw an amount after checking balance
- -To display name and balance

```
#include<iostream>
using namespace std;
class detail
{
        public:
        int accNum, bal, amt;
        string name, accTyp;

        void assignValue()
        {
                cout<<"Enter Account Holder Name : ";
                cin>>name;
                cout<<"Enter Account Number : ";
                cin>>accNum;
                cout<<"Enter Account Type : ";
                cin>>accTyp;
                cout<<"Enter Account Balance : ";
                cin>>bal;
        }
        void depositAmnt()
        {
                cout<<"Account holder name : "<<name<<endl;
                cout<<"Enter Deposit Amount : ";
                cin>>amt;
                bal+=amt;
                cout<<"Amount "<< amt <<" Deposited Successfully";
        }
```

```
	void withdrawAmnt()
	{
		cout<<"Your Account Balance : "<<bal<<endl;
		cout<<"Enter withdraw Amount : ";
		cin>>amt;
		bal-=amt;
		cout<<"Amount "<< amt <<" Withdraw Successfully";
	}
	void dispData()
	{
		cout<<"Account holder name : "<<name<<endl;
		cout<<"Account Balance : "<<bal<<endl;
	}
};
int main()
{
	int n;
	detail det;
	cout<<"------Wellcome------"<<endl;
	det.assignValue();
	do
	{
		cout<<endl<<endl;
		cout<<"1. Deposit Amount"<<endl;
		cout<<"2. Withdraw Amount"<<endl;
		cout<<"3. check Balance"<<endl;
		cout<<"Enter any other key to exit"<<endl<<endl;

		cout<<"Enter your Choice key : ";
		cin>>n;
		switch(n)
		{
		case 1:
			det.depositAmnt();
			break;
		case 2:
			det.withdrawAmnt();
			break;
		case 3:
			det.dispData();
			break;
		}
	}while(n>=1 && n<=3);

	return 0;
}
```

❖ Write a program to find the multiplication values and the cubic values using inline function

```
#include<iostream>
using namespace std;
inline cube(int a)
{
        int result;
        result = a*a*a;
        return result;
}
main()
{
        int a;
        cout<<"enter a number A :";
        cin>>a;
        cout<<"Cube of A is :"<<cube(a);
}
```

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

```
#include<iostream>
using namespace std;
class procces
{
        public:
                procces(int a, int b, int c)
                {
                        cout<<a<<"+"<<b<<"="<<a+b<<endl;
                }
                procces(int a, int b)
                {
                        cout<<a<<"-"<<b<<"="<<a-b<<endl;
                        cout<<a<<"*"<<b<<"="<<a*b;
                }
};
main()
{
        procces pa=procces(20,37,25);
        procces ps=procces(37,18);
}
```

❖ **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)**

```
#include <iostream>
#include <string>
class Cricketer {
protected:
    std::string name;
    int age;

public:
    void inputCricketerData() {
        std::cout << "Enter Cricketer's Name: ";
        std::cin >> name;
        std::cout << "Enter Cricketer's Age: ";
        std::cin >> age;
    }
};

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

public:
    void inputBatsmanData() {
        inputCricketerData(); // Inherit the input function from the base class

        std::cout << "Enter Total Runs Scored: ";
        std::cin >> totalRuns;
        std::cout << "Enter Average Runs Scored: ";
        std::cin >> averageRuns;
        std::cout << "Enter Best Performance (in a single match): ";
        std::cin >> bestPerformance;
    }

    void calculateAverageRuns() {
        // Calculation of average runs can be implemented here
        // For example: averageRuns = totalRuns / numberOfMatches;
```

```
    }

    void displayData() {
        std::cout << "\nBatsman Information" << std::endl;
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << " years" << std::endl;
        std::cout << "Total Runs Scored: " << totalRuns << std::endl;
        std::cout << "Average Runs Scored: " << averageRuns << std::endl;
        std::cout << "Best Performance (in a single match): " << bestPerformance <<
std::endl;
    }
};

int main() {
    Batsman batsman;
    batsman.inputBatsmanData();
    batsman.calculateAverageRuns();
    batsman.displayData();

    return 0;
}
```

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

```
#include <iostream>
#include <string>
using namespace std;
class Person {
public:
    string name;
    int age;

    void readPersonData() {
        cout << "Enter name: ";
        cin >> name;
```

```
        cout << "Enter age: ";
        cin >> age;
    }
    void displayPersonData() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << " years" << endl;
    }
};

// Derived class Student
class Student : public Person {
public:
    double percentage;

    Student(const std::string& _name, int _age, double _percentage) : Person(_name,
_age), percentage(_percentage) {}

    void readStudentData() {
        readPersonData();
        std::cout << "Enter percentage: ";
        std::cin >> percentage;
    }

    void displayStudentData() {
        displayPersonData();
        cout << "Percentage: " << percentage << "%" << endl;
    }
};

// Derived class Teacher
class Teacher : public Person {
public:
    double salary;

    Teacher(const std::string& _name, int _age, double _salary) : Person(_name, _age),
salary(_salary) {}

    void readTeacherData() {
        readPersonData();
        std::cout << "Enter salary: $";
        std::cin >> salary;
    }

    void displayTeacherData() {
        displayPersonData();
        std::cout << "Salary: $" << salary << std::endl;
```

```
    }
};

int main() {
    Student student("Alice", 20, 85.5);
    Teacher teacher("Mr. Smith", 35, 50000.0);

    std::cout << "Student Data:" << std::endl;
    student.displayStudentData();

    std::cout << "\nTeacher Data:" << std::endl;
    teacher.displayTeacherData();

    return 0;
}
```

- **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)**

```
#include <iostream>
using namespace std;
class Students                  // Class to store roll number
{
        protected:
        int rollNumber;

        public:
        Students(int roll) : rollNumber(roll) {}

    void displayRollNumber() {
        cout << "Roll Number: " << rollNumber << endl;
    }
};
```

```
class Test : public Students          // Store marks obtained in two subjects
{
        protected:
        int subject1Marks;
        int subject2Marks;

        public:
        Test(int roll, int s1, int s2) : Students(roll),
                subject1Marks(s1), subject2Marks(s2) {}

    void displayMarks() {
        cout << "Subject 1 Marks: " << subject1Marks << endl;
        cout << "Subject 2 Marks: " << subject2Marks << endl;
    }
};
class Result : public Test    // Store obtained marks in the test and calculate the total
{
        public:
        Result(int roll, int s1, int s2) : Test(roll, s1, s2) {}

    int calculateTotalMarks() {
        return subject1Marks + subject2Marks;
    }

    void displayTotalMarks() {
        cout << "Total Marks Obtained: " << calculateTotalMarks() << endl;
    }
};

int main() {
    // Create an instance of the Result class
    Result studentResult(101, 85, 90);

    // Display roll number, marks, and total marks
    studentResult.displayRollNumber();
    studentResult.displayMarks();
    studentResult.displayTotalMarks();

    return 0;
}
```

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

```
#include<iostream>
using namespace std;
class Calculation
{
	public:
		int add(int a, int b) {
			return a + b;
		}
		double add(double a, double b) {
			return a + b;
		}
		int sub(int a, int b) {
			return a - b;
		}
		double sub(double a, double b) {
			return a - b;
		}
		int mutiply(int a, int b) {
			return a * b;
		}
		double mutiply(double a, double b) {
			return a * b;
		}
		int divide(int a, int b) {
			return a/b;
		}
		double divide(double a, double b) {
			return a/b;
		}
};
main() {
	Calculation calculator;
	calculator.add(24,24);
	cout << "result :" <<calculator.add(15.56,38.63);
}
```

❖ Write a Program of Two 1D Matrix Addition using Operator Overloading

```
#include <iostream>
using namspace std;
class Matrix
{
        private:
        int size;
        int* data;

        public:
        Matrix(int n) : size(n) {
        data = new int[size];
    }

    Matrix operator+(const Matrix& other) const {
        if (size != other.size) {
                        throw std::invalid_argument("Matrix sizes are not compatible for
addition.");
        }

    Matrix result(size);
        for (int i = 0; i < size; i++) {
            result.data[i] = data[i] + other.data[i];
        }
        return result;
    }

    void display() const {
        for (int i = 0; i < size; i++) {
            std::cout << data[i] << " ";
        }
        std::cout << std::endl;
    }
};

int main()
{
    Matrix matrix1(5);
    Matrix matrix2(5);

    for (int i = 0; i < 5; i++) {
        matrix1.data[i] = i + 1;
        matrix2.data[i] = i * 2;
```

```
    }

    Matrix result = matrix1 + matrix2;

    cout << "Matrix 1: ";
    matrix1.display();
    cout << "Matrix 2: ";
    matrix2.display();
    cout << "Result: ";
    result.display();

    return 0;
}
```

❖ Write a program to concatenate the two strings using Operator Overloading

```
#include <iostream>
#include <string>
class ConcatenatedString
{
        private:
        std::string str;

        public:
        ConcatenatedString(const std::string& s) : str(s) {}

            ConcatenatedString operator+(const ConcatenatedString& other) const {
            return ConcatenatedString(str + other.str);
            }

    // Function to get the concatenated string
        std::string getString() const {
        return str;
            }
};

int main()
{
    ConcatenatedString str1("Hello, ");
    ConcatenatedString str2("world!");
```

```
    // Concatenate the strings using operator overloading
    ConcatenatedString result = str1 + str2;

    // Display the concatenated string
    std::cout << "Concatenated String: " << result.getString() << std::endl;

    return 0;
}
```

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

- ➢ **Rectangle: Area * breadth**
- ➢ **Triangle: ½ *Area* breadth**
- ➢ **Circle: Pi * Area *Area**

```
#include <iostream>
#include <cmath>
using namespace std;
const double PI = 3.14159265359;

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

// Function to calculate the area of a triangle
double calculateArea(double base, double height, char shape) {
    if (shape == 'T' || shape == 't') {
        return 0.5 * base * height;
    } else {
        cout << "Invalid shape code for triangle." << endl;
        return 0.0;
    }
}

// Function to calculate the area of a circle
double calculateArea(double radius, char shape) {
    if (shape == 'C' || shape == 'c') {
        return PI * radius * radius;
```

```
    } else {
        cout << "Invalid shape code for circle." << endl;
        return 0.0;
    }
}

int main() {
    char shape;
    cout << "Enter the shape code (R for rectangle, T for triangle, C for circle): ";
    cin >> shape;

    double area = 0.0;

    if (shape == 'R' || shape == 'r') {
        double length, breadth;
        cout << "Enter length and breadth of rectangle: ";
        cin >> length >> breadth;
        area = calculateArea(length, breadth);
    } else if (shape == 'T' || shape == 't') {
        double base, height;
        cout << "Enter base and height of triangle: ";
        cin >> base >> height;
        area = calculateArea(base, height, 'T');
    } else if (shape == 'C' || shape == 'c') {
        double radius;
        cout << "Enter the radius of circle: ";
        cin >> radius;
        area = calculateArea(radius, 'C');
    } else {
        cout << "Invalid shape code." << endl;
        return 1;
    }

    cout << "Area of the shape: " << area << endl;

    return 0;
}
```

❖ Write a program to swap the two numbers using friend function without using third variable

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

```
class A
{
        private: // private data member
                int x, y;

        public:
                friend void fun(); // Friend function
};
void fun() {
  A obj;
  obj.x = 10;
  obj.y = 20;
  obj.x = obj.x + obj.y ;
  obj.y = obj.x - obj.y ;
  obj.x = obj.x - obj.y ;
  cout << "X :" << obj.x << endl;
  cout << "Y :" << obj.y << endl;
}
int main()
{
        cout << "Hello W      orld!\n";
        fun();
}
```

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

```
#include <iostream>
using namespace std;
class MaxFinder
{
        private:
        int num1;
        int num2;

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

        // Friend Function to access private
    friend int findMax(MaxFinder mf);
};
```

```
int findMax(MaxFinder mf) {
    return (mf.num1 > mf.num2) ? mf.num1 : mf.num2;   // using teranary operator
}

int main()
{
    int num1, num2;
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;

    MaxFinder mf(num1, num2);

    int max = findMax(mf);

    cout << "The maximum number is: " << max << endl;

    return 0;
}
```

Module : 4.2 (C,C++ Templates)

❖ Write a program of to swap the two values using templates

```
#include <iostream>
using namespace std;
template <class T> void swapping(T &a, T &b)
{
        T temp = a;
        a = b;
        b = temp;
}
int main()
{
        char a = 'A', b = 'B';
        int x = 20, y = 30;

        cout << "Before Swapping "<< "A :" << a << " B :" << b << endl;
        swapping(a, b);
        cout << "After Swapping "<< "A :" << a << " B :" << b << endl;

        cout << "Before Swapping "<< "X :" << x << " Y :" << y << endl;
        swapping(x, y);
        cout << "After Swapping "<< "X :" << x << " Y :" << y << endl;
}
```

❖ Write a program to sort the array using templates.

```
#include <iostream>
using namespace std;
template <class T> void bubbleSort(T a[], int n)
{
   for (int i = 0; i < n - 1; i++)
       for (int j = n - 1; i < j; j--)
           if (a[j] < a[j - 1])
               swap(a[j], a[j - 1]);
}

int main()
{
```

```
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);

    cout << " Sorted array : ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " "<< endl;

    return 0;
}
```