**Assignment C++**

***MODULE: 4.1 (C++ Basic)***

**Q1 - WAP to print “Hello World” using C++**

**Ans –**

#include<iostream>

using namespace std;

int main()

{

cout<<"\nHello World\n";

}

**Q2 - What is OOP? List OOP concepts**

**Ans –**

OOP, or Object-Oriented Programming, is a programming paradigm that organizes and structures code around the concept of "objects." Objects are instances of classes, which serve as blueprints or templates for creating objects. OOP is designed to model real-world entities and their interactions, making it a powerful and widely used programming paradigm. Here are some key OOP concepts:

1. \*\*Class:\*\* A class is a blueprint or a template for creating objects. It defines the attributes (data members) and behaviors (methods) that its objects will have.

2. \*\*Object:\*\* An object is an instance of a class. It is a concrete entity that contains data and can perform actions defined by its class.

3. \*\*Encapsulation:\*\* Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit (the class). It enforces access restrictions on the data to prevent unauthorized modification and ensures that the data's integrity is maintained.

4. \*\*Abstraction:\*\* Abstraction involves simplifying complex systems by breaking them into smaller, more manageable parts. In OOP, classes and objects provide abstraction by hiding the underlying implementation details and exposing only the necessary interfaces.

5. \*\*Inheritance:\*\* Inheritance is a mechanism that allows a class (subclass or derived class) to inherit properties and behaviors from another class (superclass or base class). It promotes code reuse and the creation of specialized classes.

6. \*\*Polymorphism:\*\* Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the use of a single interface to represent a general class of actions, making code more flexible and extensible. Polymorphism can be achieved through method overriding and interfaces/abstract classes.

7. \*\*Method:\*\* A method is a function defined within a class that can perform actions or operations on the object's data. Methods are also referred to as member functions.

8. \*\*Constructor:\*\* A constructor is a special method used for initializing objects when they are created. It typically sets initial values for the object's attributes.

9. \*\*Destructor:\*\* A destructor is a special method used for cleaning up resources when an object is no longer needed. It is less commonly used in many modern programming languages due to automatic memory management.

10. \*\*Access Control:\*\* OOP languages often provide access control mechanisms (e.g., public, private, protected) to restrict the visibility and modification of class members (attributes and methods). This helps in maintaining the encapsulation principle.

11. \*\*Composition:\*\* Composition is a design principle where complex objects can be created by combining smaller objects as parts. It allows for building more complex and flexible structures by aggregating objects.

12. \*\*Association:\*\* Association represents relationships between classes. It can be one-to-one, one-to-many, or many-to-many and can be used to model various types of connections between objects.

These OOP concepts collectively provide a structured approach to software development, making code more organized, reusable, and maintainable. Different programming languages may implement these concepts in slightly different ways, but the fundamental principles remain consistent across most OOP languages.

**Q3 - What is the difference between OOP and POP?**

**Ans –**

OOP (Object-Oriented Programming) and POP (Procedural-Oriented Programming) are two distinct programming paradigms with different approaches to organizing and structuring code. Here are the key differences between OOP and POP:

1. \*\*Paradigm Focus:\*\*

- \*\*OOP:\*\* OOP focuses on organizing code around objects, which are instances of classes. It emphasizes modeling real-world entities as objects and their interactions. OOP promotes encapsulation, abstraction, inheritance, and polymorphism.

- \*\*POP:\*\* POP organizes code around procedures or functions. It emphasizes a linear flow of control and the use of procedures to manipulate data. There is less emphasis on modeling real-world entities as objects.

2. \*\*Data and Behavior:\*\*

- \*\*OOP:\*\* In OOP, data (attributes or properties) and behavior (methods or functions) are encapsulated within objects. Objects can have their own data and methods, making it easy to model complex systems and relationships between objects.

- \*\*POP:\*\* In POP, data is often treated as global or passed explicitly to functions as parameters. Functions manipulate data, and the organization of data and functions is often less structured compared to OOP.

3. \*\*Encapsulation:\*\*

- \*\*OOP:\*\* OOP enforces strong encapsulation, which means that data is typically hidden from direct access outside the object. Access to data is controlled through methods, ensuring data integrity.

- \*\*POP:\*\* Encapsulation in POP is weaker since data is often global or accessible by any function. There is less control over who can modify data, which can lead to potential issues with data integrity.

4. \*\*Abstraction:\*\*

- \*\*OOP:\*\* OOP provides a high level of abstraction by allowing developers to model complex systems using objects and abstracting away implementation details. Classes and objects provide a natural form of abstraction.

- \*\*POP:\*\* While POP can use functions to encapsulate behavior, it may not provide the same level of abstraction as OOP. Developers need to manually manage and organize functions to achieve abstraction.

5. \*\*Inheritance and Polymorphism:\*\*

- \*\*OOP:\*\* OOP supports inheritance, allowing classes to inherit properties and behaviors from other classes. Polymorphism allows objects of different classes to be treated as instances of a common superclass.

- \*\*POP:\*\* POP does not inherently support inheritance and polymorphism as OOP does. Achieving these features may require more manual coding and may not be as natural.

6. \*\*Modularity:\*\*

- \*\*OOP:\*\* OOP promotes modularity through the use of classes and objects. Objects can be easily reused and extended, promoting code reusability.

- \*\*POP:\*\* POP also supports modularity through the use of functions, but the organization of code may be less structured, making it potentially harder to manage and reuse code.

7. \*\*Examples:\*\*

- \*\*OOP:\*\* Languages like Java, C++, Python, and C# are commonly associated with OOP.

- \*\*POP:\*\* Languages like C, Pascal, and Fortran are often used in a procedural programming style.

In summary, OOP and POP represent different approaches to software development, with OOP focusing on objects, encapsulation, and modeling real-world entities, while POP relies on procedures and a linear flow of control. The choice between these paradigms often depends on the specific requirements of a project and personal programming style preferences.