What Are OOPS Concepts In C++?

OOPs, Object Oriented Programming C++ is an approach or 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.

There are some basic concepts that act as the building blocks of OOPs.

Classes & Objects

Abstraction

Encapsulation

Inheritance

Polymorphism

So now, you will understand each of these concepts in detail.

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. It avoids unnecessary information or irrelevant details and shows only that specific part that the user wants to see. The benefits of Abstraction includes:

1. Simplified Interface: Abstraction allows hiding complex implementation details and exposing a simplified interface to the user. This promotes ease of use and reduces complexity, as users can interact with high-level abstractions without needing to understand the underlying complexities.
2. Code Maintenance: By focusing on essential features and hiding unnecessary implementation details, abstraction improves code maintainability. Changes made to the underlying implementation are isolated, reducing the impact on other parts of the codebase.
3. Flexibility: Abstraction enables the creation of generalized interfaces, which can be implemented differently for various use cases. This provides flexibility and allows developers to adapt and extend the functionality of the code without affecting the code using abstraction

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. Encapsulation makes the data non-accessible to the outside world. The benfits of encapsulation include:

1. Data Protection: Encapsulation hides the internal implementation details of an object, protecting the data from unauthorized access or modification. It provides data integrity and maintains the consistency of the object's state.
2. Code Organization: Encapsulation helps organize code by grouping related data and functions together within a class. This improves code readability and maintainability by providing a clear structure

and reducing code clutter.
3. Code Reusability: Encapsulation facilitates code reusability by encapsulating functionality within objects. Objects can be easily reused in different parts of the code or in different projects, promoting efficient development and reducing redundant code.

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. For example, Class Vehicle is the parent class, and Class Bus, Car, and Bike are child classes. The benefits of Inheritance include:
1. Code Reusability: Inheritance allows classes to inherit properties and behavior from a base class, promoting code reuse. Derived classes can extend and specialize the functionality of the base class, eliminating the need to rewrite common code.
2. Polymorphism: Inheritance enables polymorphism, where objects of derived classes can be treated as objects of their base class. This allows for more flexible and dynamic programming, as different derived classes can be used interchangeably through a common interface.
3. Hierarchical Organization: Inheritance establishes a hierarchical relationship among classes, reflecting real-world or conceptual hierarchies. This improves code organization, understandability, and maintainability by representing relationships and dependencies in a structured manner.

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. The benfits include:
1. Code Flexibility: Polymorphism allows different objects to be treated uniformly through a common interface. This promotes code flexibility, as functions or algorithms can work with objects of different types without needing to know their specific implementations.
2. Code Extensibility: Polymorphism enables the addition of new derived classes that adhere to the same interface without modifying existing code. This extensibility simplifies code maintenance and allows for easy integration of new features.
3. Code Readability: Polymorphism enhances code readability by expressing intent and behavior through the use of abstract interfaces. It allows for more concise and expressive code, as the focus is on what needs to be done rather than how it is done.