SECTION 7

PROGRAMMING PARADIGMS

Unstructured Programming

Unstructured programming is the earliest form of programming, where code is written as a continuous sequence of instructions without any modular organization. It relies heavily on GOTO statements to control program flow, making it difficult to read, debug, and maintain.

Early languages like Assembly and BASIC followed this approach, leading to spaghetti code, where logic becomes increasingly complex due to excessive jumps as the program grows.

As programming evolved, structured programming emerged, introducing loops, conditional structures, and functions to improve code organization. While unstructured programming is rarely used today, it laid the foundation for modern programming paradigms.

Structured Programming

Structured programming is a programming approach that focuses on organizing code in a clear and logical manner. By using well-defined structures such as control statements and loop statements (iterations), it makes code more readable, easier to maintain, and less prone to errors during development.

Structured programming encourages reusability through modular design by breaking programs into functions (procedures, subroutines).

Unlike unstructured programming, structured programming avoids spaghetti code. It also differs from object-oriented programming (OOP), which focuses on objects and classes.

Object Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which are instances of classes. OOP aims to model real-world entities, making code more flexible, reusable, and easier to maintain.

Key Concepts of OOP:

- **Classes and Objects:** A class is a template for creating objects. It defines properties (attributes) and behaviors (methods) that the objects created from it will have. An object is an instance of a class, representing a specific entity with its own values for the attributes.

- **Encapsulation:** Encapsulation is about bundling data (attributes) and methods (functions) that operate on the data into a single unit, the object. Encapsulation hides the internal state of the object and only exposes the necessary information through public methods (getters and setters). It helps prevent unauthorized access and modification of data.
- **Inheritance:** Inheritance allows a class (derived class) to inherit properties and behaviors from another class (base class). This promotes code reusability and establishes a relationship between different classes.
- **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method or function to behave differently based on the object calling it. For example, a method makeSound() might behave differently for a Dog (barks) and a Cat (meows), even though both are treated as Animal objects.
- **Abstraction:** Abstraction hides complex implementation details and shows only the essential features of an object. It simplifies interaction with complex systems by focusing on high-level operations.

Advantages of OOP:

- **Reusability**: OOP promotes code reuse through inheritance and modularity.
- **Maintainability**: Code is easier to maintain due to encapsulation and modular design.
- **Scalability**: OOP systems are easier to scale as new classes can be added without affecting existing code.
- **Flexibility**: Polymorphism and abstraction provide flexibility in designing and interacting with systems.

OOP is particularly useful for large, complex applications where different components need to interact with each other in an organized and scalable manner. It has become the foundation of modern software development.