1. Basic Principles of OOP

1. **Encapsulation**
    - Wrapping data and methods that operate on the data into a single unit (class).
    - Example: Protecting sensitive data with private access modifiers and accessing it via public methods.
2. **Abstraction**
    - Hiding implementation details and exposing only the necessary functionality to the user.
    - Example: A car's dashboard shows only essential controls like speed and fuel gauge, not the engine's internal workings.
3. **Inheritance**
    - Deriving new classes from existing classes to reuse code and add new features.
    - Example: A `Car` class inheriting properties from a generic `Vehicle` class.
4. **Polymorphism**
    - The ability to take many forms, allowing a single interface to represent different types.
    - Types:
        - Compile-Time Polymorphism: Function overloading, Operator overloading.
        - Run-Time Polymorphism: Function overriding using virtual functions.

---

2. Classes and Objects

1. **Class**
    - Blueprint for creating objects. Defines data members and member functions.
2. **Object**
    - Instance of a class, representing real-world entities.
3. **Access Specifiers**
    - Control access to class members:
        - **Public**: Accessible anywhere.
        - **Private**: Accessible only within the class.
        - **Protected**: Accessible within the class and its derived classes.

---

3. Constructors and Destructors

1. **Constructors**
    - Special functions called automatically when an object is created.
    - Types:
        - Default Constructor
        - Parameterized Constructor
        - Copy Constructor

2. **Destructors**
    - Special functions called automatically when an object goes out of scope to release resources.

---

4. Function Overloading and Overriding

1. **Function Overloading**
    - Defining multiple functions with the same name but different parameters.
2. **Function Overriding**
    - Redefining a base class function in the derived class with the same signature.

---

5. Operator Overloading

- Redefining the behavior of operators (+, -, *, etc.) for user-defined types.

---

6. Inheritance

1. **Types of Inheritance**
    - Single Inheritance
    - Multiple Inheritance
    - Multilevel Inheritance
    - Hierarchical Inheritance
    - Hybrid Inheritance
2. **Base and Derived Classes**
    - Base class: Parent class.
    - Derived class: Child class inheriting from the base class.
3. **Access Control in Inheritance**
    - Public, Protected, Private inheritance.

---

7. Polymorphism

1. **Compile-Time Polymorphism**
    - Achieved via function overloading and operator overloading.
2. **Run-Time Polymorphism**
    - Achieved via virtual functions and function overriding.
3. **Virtual Functions and Pure Virtual Functions**
    - Virtual Function: Enables polymorphism by overriding methods in derived classes.

- Pure Virtual Function: A virtual function declared in a base class with no implementation, forcing derived classes to implement it.

---

8. Abstraction and Interfaces

1. **Abstract Classes**
   - Classes that cannot be instantiated and may contain pure virtual functions.
2. **Interfaces**
   - A collection of pure virtual functions that provide a contract for derived classes.

---

9. Encapsulation

- Protecting data by making fields private and accessing them through getter and setter methods.

---

10. Static Members

1. **Static Variables**
   - Shared by all objects of the class.
2. **Static Functions**
   - Can be called without creating an object of the class.

---

11. Friend Functions and Friend Classes

1. **Friend Function**
   - A function that is not a member of the class but can access its private and protected members.
2. **Friend Class**
   - A class that can access private and protected members of another class.

---

12. Operator Overloading

- Defining custom behavior for operators (e.g., +, -, <<, etc.) for user-defined types.

---

13. Exception Handling

- Mechanism to handle runtime errors using `try`, `catch`, and `throw`.

---

14. File Handling

- Reading and writing data to files using classes like `ifstream` and `ofstream`.

---

15. Dynamic Memory Management

1. **New and Delete Operators**
    - Allocating (`new`) and deallocating (`delete`) memory dynamically.
2. **Smart Pointers**
    - `unique_ptr`, `shared_ptr`, and `weak_ptr` for automatic memory management.

---

16. Templates

1. **Function Templates**
    - Generic functions that work with any data type.
2. **Class Templates**
    - Generic classes that work with any data type.

---

17. Namespaces

- Grouping related functions, classes, and objects to avoid name conflicts.

---

18. Type Casting

1. **Static Cast**
2. **Dynamic Cast**
3. **Const Cast**
4. **Reinterpret Cast**

---

19. Virtual Destructor

- Ensures the proper destruction of objects in an inheritance hierarchy when deleted through a base class pointer.

---

20. Access Control and Modifiers

1. **Const Objects and Functions**
    - Objects or methods that cannot modify class data.
2. **Mutable Keyword**
    - Allows a member variable to be modified even in a `const` object.

---

21. Overloading Constructors and Assignment Operators

1. **Overloading Constructors**
    - Creating multiple constructors for different initializations.
2. **Overloading Assignment Operator (=)**
    - Custom implementation of object copying.

---

22. Virtual Table (VTable)

- Mechanism used to implement dynamic polymorphism in C++.

---

23. Multi-threading (Advanced)

- Writing concurrent code using threads for better performance.

---

24. Design Patterns (Advanced)

- Singleton, Factory, Observer, etc., to solve common design problems.