Class:- Combining data and functions into a single unit called class

Object:- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, youneed to create objects.

Encapsulation:- Combining data and functions into a single unit called class and the process is known as Encapsulation

Data hiding:- Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details

Abstraction:- Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details

- o Abstraction using classes
- o Abstraction in header files

Dynamic binding:- Dynamic binding, also known as late binding or runtime polymorphism. Dynamic binding is a mechanism in object-oriented programming where the choice of which method or function to execute is determined at runtime, based on the actual type of the object being used.

Inheritance:- Inheritance is a mechanism in OOP where a new class (derived or child class) can acquire the properties (data members) and behaviors (methods/functions) of an existing class (base or parent class).

Types of Inheritance:-

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance

POLYMORPHISM:- A greek term means ability to take more than one form. An operation may exhibite different behaviours in different instances.

DOT(.) Operator:- The data members and member functions of class can be accessed using the dot(‘.’)operator with the object. Also known as direct member access operator

scope resolution operator:- The scope resolution operator (::) in C++ is used to specify the scope or context in which a class member or global variable is defined and to access that member.

3 uses of void:

1) to specify return type

2) to indicate empty argument list and

3) to declare generic pointers.

User Defined Data Types:-

- Class
- Structure
- Enumeration

Enumeration:- An enumeration (enum) is a user-defined data type in C++ used to create a collection of named integral constants. Enumerations simplify code readability by assigning meaningful names to values, making the code more self-explanatory. Enumerations provide a way to group related constants together under a single type, enhancing code maintainability and reducing the likelihood of errors due to the use of incorrect integer values

```
enum EnumName {
   Identifier1,   // Implicitly assigned value 0
   Identifier2 = 5,
   Identifier3,
   // ...
};
```

Structure:- Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collecion of data of different data types.

Pointers:

- Pointers are variables that store memory addresses of other variables.
- They are derived from the fundamental data types and are used to work with memory locations directly.

default argument:- A default argument is a value provided in a function declaration that is automaticallyassigned by the compiler if the caller of the function doesn't provide a value for the argument with a default value

visibility modifiers:- To provide the facility of security by means of data hiding, is get done by some access specifies and these access specifies are known as visibility modifiers or access specifiers. These visibility modifiers are used to secure the data member and member function of a class

1. Public
2. Private
3. Protected

Constructor:- In C++, a constructor is a special member function of a class that is automatically called when an object of the class is created. Constructors are used to initialize the data members (attributes) of an object and set its initial state. Constructors have the same name as the class and do not have a return type, not even **void**

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types.
- They can not be inherited, though a derived class can call the base constructor
- They have default arguments
- Constructor can not be virtual
- We can not refer to their address.

Types of constructor:

- Default constructor
- Parameterized constructor
- Overloaded constructor
- Constructor with default value
- Copy constructor
- Inline constructor

Destructor:- Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed

Properties of Destructor:-

- The destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.

- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of the destructor.

Ambiguity: - In multiple inheritance the ambiguity arises when same method name is being used by two derived class and further derivation from these two base classes. To resolve this ambiguity we are using scope resolution operator(::).

virtual function:- a virtual function is used in the base class in order to ensure that the function is **overridden**. This especially applies to cases where a pointer of base class points to an object of a derived class.

A virtual function is a member function in the base class that we expect to redefine in derived classes.

Friend Function:- Friend function is a special type of function, which declares inside the class. Friend function can access the private, protected and public data of the class. If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It can not be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It can not access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

Early Binding: The binding which can be resolved at compile time by the compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time

Late binding: In the late binding or dynamic binding, the compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.

Method Overloading:- Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overriding:- Method Overriding is a **Run time polymorphism**. In method overriding, the derived class provides the specific implementation of the method that is already provided by the base class or parent class. In method overriding, the return type must be the same or co-variant (return type may vary in the same direction as the derived class).

Virtual Base class:- a virtual base class is used to prevent the "diamond problem" or "diamond inheritance" issue that can arise in multiple inheritance scenarios. The diamond problem occurs when a class inherits from two or more classes that have a common base class.

Abstract Class:- An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class. A class is made abstract if at least one pure virtual function defined

Recursion:- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH)

Data Structure:- A data structure is a particular way of organising data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks**.**

Types of DS:

1. Linear Data Structure
   - (i) Static Data Structure
     - Array
   - (ii) Dynamic Data Structure
     - Queue
     - Stack
     - Linked List
2. Non-Linear Data Structure
   - Tree
   - Graph

Primitive data structures:- The primitive data structures are nothing but the predefined data structures, which are already defined; we do not require giving a particular definition. To derive the non-primitive data structures. These data structures involve int, float, char, etc

Non-Primitive data structures:- The non-primitive data structures are nothing but the defined data structures used to create particular data structures by using the primitive data structures. It is mainly used to store the collection of elements; it may be of the same data types and may differ depending on the program's need

Applications of Data Structures:

Data structures are used in various fields such as:

- Operating system
- Graphics
- Computer Design
- Blockchain
- Genetics
- Image Processing
- Simulation

Stack: A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the **top**. A stack follows the **LIFO** (Last In First Out) principle, i.e., the element inserted at the last is the first element to come out. The insertion of an element into the stack is called push operation, and the deletion of an element from the stack is called **pop** operation. In stack, we always keep track of the last element present in the list with a pointer called **top**.

Queue:- A queue is a fundamental data structure in computer science and information technology that follows the First-In-First-Out (FIFO) principle. It's a linear data structure where elements are inserted at one end and removed from the other end.

Early Binding	Late Binding
It is a compile-time process	It is a run-time process
The method definition and method call are linked during the compile time.	The method definition and method call are linked during the run time.
Actual object is not used for binding.	Actual object is used for binding.
For example: Method overloading	For example: Method overriding
Program execution is faster	Program execution is slower

Method Overloading	Method Overriding
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading helps to increase the readability of the program.	Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.

Method Overloading	Method Overriding
In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Poor Performance due to compile time polymorphism.	It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime.
Private and final methods can be overloaded.	Private and final methods can’t be overridden.
The argument list should be different while doing method overloading.	The argument list should be the same in method overriding.