Abstract Classes And Interface

Abstract Classes in C++

- An abstract class in C++ is a class that contains at least one **pure virtual function**.
- A pure virtual function is a function declared in a base class that has no definition within that class and is marked by assigning `0` to it.
  Example: virtual void walk() = 0;
- Abstract classes cannot be instantiated directly; they are meant to be inherited by other classes, which must provide implementations for the pure virtual functions.

Key Points:

- **Pure Virtual Function**: A function declared by assigning `0` to it.
- **Cannot Instantiate**: You cannot create an object of an abstract class.
- **Inheritance**: Classes that inherit from an abstract class must implement the pure virtual functions, or they will also be considered abstract.

Interface

- In C++, there is no direct concept of "interfaces" as in languages like Java.
- However, you can achieve the same behavior by using abstract classes that contain only pure virtual functions and no data members or implemented functions. Such a class effectively serves as an interface.

Key Points:

- **No Data Members**: An interface should not contain any data members.
- **Only Pure Virtual Functions**: All functions in an interface are pure virtual.
- **Multiple Inheritance**: C++ allows a class to inherit from multiple interfaces (abstract classes).

All About Static

- In C++, the `static` keyword is a versatile feature that can be used with variables, functions, and members of a class. It changes the behavior of these elements in terms of their lifetime, scope, and linkage.

1. Static Variables

a. Static Local Variables

- A **static local variable** inside a function retains its value between function calls. Unlike regular local variables, which are created and destroyed each time a function is called, static local variables are initialized only once, and their value persists across multiple function calls.

b. Static Global Variables

- A **static global variable** (or static variable at the file scope) limits the variable's scope to the file in which it is declared. This is different from a regular global variable, which is accessible across multiple files.

  // file1.cpp static int counter = 0; // Static global variable, not accessible outside this file

- **File Scope**: The static global variable is only visible within the file it is declared in.
- **Linkage**: It has internal linkage, meaning it is not accessible from other translation units (i.e., other `.cpp` files).

2. Static Member Variables

- A **static member variable** of a class is shared among all objects of that class. It does not belong to any specific object but rather belongs to the class itself.
- **Shared Across Objects**: The static member variable `count` is shared by all instances of `MyClass`.
- **Class-level Access**: It can be accessed without creating an instance of the class using `ClassName::staticMember`.
- Syntax:

```
class MyClass
{
  public:
        static int count; // Static member variable
        MyClass() {
                count++;
        }
        static void displayCount() {
                cout << "Count: " << count << endl;
        }
};
```

3. Static Member Functions

- A **static member function** belongs to the class rather than any particular object. It can only access static member variables and other static member functions.
- **No Object Required**: Static member functions can be called without creating an instance of the class.
- **Access**: They can only access static data members or other static member functions of the class.
- Syntax:

```
class MyClass
{
  public:
        static int count; // Static member variable
        static void incrementCount() {
                // Static member function
                count++;
        }
};
```

***Static Functions (File Scope)**: Restrict their visibility to the file in which they are declared.*