**Inheritance in C++**

* Reusability is a very important feature of OOPs
* In C++ we can reuse a class and add additional features to it
* Reusing classes saves time and money
* Reusing already tested and debugged classes will save a lot of effort of developing and debugging the same thing again

**What is Inheritance in C++?**

* The concept of reusability in C++ is supported using inheritance
* We can reuse the properties of an existing class by inheriting it
* The existing class is called a base class
* The new class which is inherited from the base class is called a derived class
* Reusing classes saves time and money
* There are different types of inheritance in C++

**Forms of Inheritance in C++:**

* Single Inheritance
* Multiple Inheritance
* Hierarchical Inheritance
* Multilevel Inheritance
* Hybrid Inheritance

**1. Single Inheritance in C++:**

Single inheritance is a type of inheritance in which a derived class is inherited with only one base class. For example, we have two classes “employee” and “programmer”. If the “programmer” class is inherited from the “employee” class which means that the “programmer” class can now implement the functionalities of the “employee” class.

**2. Multiple Inheritances in C++:**

 Multiple inheritances are a type of inheritance in which one derived class is inherited with more than one base class. For example, we have three classes “employee”, “assistant” and “programmer”. If the “programmer” class is inherited from the “employee” and “assistant” class which means that the “programmer” class can now implement the functionalities of the “employee” and “assistant” class.

**3. Hierarchical Inheritance:**

A hierarchical inheritance is a type of inheritance in which several derived classes are inherited from a single base class. For example, we have three classes “employee”, “manager” and “programmer”. If the “programmer” and “manager” classes are inherited from the “employee” class which means that the “programmer” and “manager” class can now implement the functionalities of the “employee” class.

**4. Multilevel Inheritance in C++**

Multilevel inheritance is a type of inheritance in which one derived class is inherited from another derived class. For example, we have three classes “animal”, “mammal” and “cow”. If the “mammal” class is inherited from the “animal” class and “cow” class is inherited from “mammal” which means that the “mammal” class can now implement the functionalities of “animal” and “cow” class can now implement the functionalities of “mammal” class.

**5. Hybrid Inheritance in C++**

Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. In hybrid inheritance, a class is derived from two classes as in multiple inheritances. However, one of the parent classes is not a base class. For example, we have four classes “animal”, “mammal”, “bird”, and “bat”. If “mammal”  and “bird” classes are inherited from the “animal” class and “bat” class is inherited from “mammal” and “bird” classes which means that “mammal” and “bird” classes can now implement the functionalities of “animal” class and “bat” class can now implement the functionalities of “mammal” and “bird” classes

**Protected Access Modifiers in C++**

Protected access modifiers are similar to the private access modifiers but protected access modifiers can be accessed in the derived class whereas private access modifiers cannot be accessed in the derived class. A table is shown below which shows the behavior of access modifiers when they are derived “public”, “private”, and “protected”.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Public Derivation** | **Private Derivation** | **Protected Derivation** |
| **Private members** | Not Inherited | Not Inherited | Not Inherited |
| **Protected members** | Protected | Private | Protected |
| **Public members** | Public | Private | Protected |

As shown in the table,

1. If the class is inherited in public mode then its private members cannot be inherited in child class.
2. If the class is inherited in public mode then its protected members are protected and can be accessed in child class.
3. If the class is inherited in public mode then its public members are public and can be accessed inside child class and outside the class.
4. If the class is inherited in private mode then its private members cannot be inherited in child class.
5. If the class is inherited in private mode then its protected members are private and cannot be accessed in child class.
6. If the class is inherited in private mode then its public members are private and cannot be accessed in child class.
7. If the class is inherited in protected mode then its private members cannot be inherited in child class.
8. If the class is inherited in protected mode then its protected members are protected and can be accessed in child class.
9. If the class is inherited in protected mode then its public members are protected and can be accessed in child class.

**Constructors in Derived Class in C++:**

* We can use constructors in derived classes in C++
* If the base class constructor does not have any arguments, there is no need for any constructor in the derived class
* But if there are one or more arguments in the base class constructor, derived class need to pass argument to the base class constructor
* If both base and derived classes have constructors, base class constructor is executed first.

**Constructors in Multiple Inheritances:**

* In multiple inheritances, base classes are constructed in the order in which they appear in the class deceleration. For example if there are three classes “A”, “B”, and “C”, and the class “C” is inheriting classes “A” and “B”. If the class “A” is written before class “B” then the constructor of class “A” will be executed first. But if the class “B” is written before class “A” then the constructor of class “B” will be executed first.
* In multilevel inheritance, the constructors are executed in the order of inheritance. For example if there are three classes “A”, “B”, and “C”, and the class “B” is inheriting classes “A” and the class “C” is inheriting classes “B”. Then the constructor will run according to the order of inheritance such as the constructor of class “A” will be called first then the constructor of class “B” will be called and at the end constructor of class “C” will be called.

**Special Syntax:**

* C++ supports a special syntax for passing arguments to multiple base classes
* The constructor of the derived class receives all the arguments at once and then will pass the call to the respective base classes
* The body is called after the constructors is finished executing

**Syntax Example:**

Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)

{

….

} Base 1-Constructor (arg1,arg2)

**Special Case of Virtual Base Class:**

* The constructors for virtual base classes are invoked before a non-virtual base class
* If there are multiple virtual base classes, they are invoked in the order declared
* Any non-virtual base class are then constructed before the derived class constructor is executed

**Polymorphism in C++**

“Poly” means several and “morphism” means form. So we can say that polymorphism is something that has several forms or we can say it as one name and multiple forms. There are two types of polymorphism:

* Compile-time polymorphism
* Run time polymorphism

**1.Compile Time Polymorphism:**

In compile-time polymorphism, it is already known which function will run. Compile-time polymorphism is also called **early binding**, which means that you are already bound to the function call and you know that this function is going to run. There are two types of compile-time polymorphism:

1. **Function Overloading:**

This is a feature that lets us create more than one function and the functions have the same names but their parameters need to be different. If function overloading is done in the program and function calls are made the compiler already knows that which functions to execute.

1. **Operator Overloading:**

This is a feature that lets us define operators working for some specific tasks. For example, we can overload the operator “+” and define its functionality to add two strings. Operator loading is also an example of compile-time polymorphism because the compiler already knows at the compile time which operator has to perform the task.

**2. Run Time Polymorphism:**

In the run-time polymorphism, the compiler doesn’t know already what will happen at run time. Run time polymorphism is also called **late binding.** The run time polymorphism is considered slow because function calls are decided at run time. Run time polymorphism can be achieved from the virtual function.

1. **Virtual Function:**

A function that is in the parent class but redefined in the child class is called a virtual function. “virtual” keyword is used to declare a virtual function.