**Inheritance & Its Different Types with Examples in C++ | C++ Tutorials for Beginners #36**

In this tutorial, we will discuss inheritance in C++

**Inheritance in C++ an Overview**

* 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

**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.

**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.

**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.

**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.

**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.

**Inheritance Syntax & Visibility Mode in C++ | C++ Tutorials for Beginners #37**

In this tutorial, we will discuss inheritance syntax and visibility mode in C++

**Inheritance Syntax and Visibility mode in C++**

Inheritance is a process of inheriting attributes of the base class by a derived class. The syntax of the derived class is shown below.

// Derived Class syntax

class {{derived-class-name}} : {{visibility-mode}} {{base-class-name}}

{

class members/methods/etc...

}

Copy

**Code Snippet 1: Derived Class syntax**

As shown in a code snippet 1,

* After writing the class keyword we have to write the derived class name and then put a “:” sign.
* After “:” sign we have to write the visibility mode and then write the base class name.

Note:

* Default visibility mode is private
* Public Visibility Mode: Public members of the base class becomes Public members of the derived class
* Private Visibility Mode: Public members of the base class become private members of the derived class
* Private members are never inherited

An example program is shown below to demonstrate the concept of inheritance.

#include <iostream>

using namespace std;

// Base Class

class Employee

{

public:

int id;

float salary;

Employee(int inpId)

{

id = inpId;

salary = 34.0;

}

Employee() {}

};

// Creating a Programmer class derived from Employee Base class

class Programmer : public Employee

{

public:

int languageCode;

Programmer(int inpId)

{

id = inpId;

languageCode = 9;

}

void getData(){

cout<<id<<endl;

}

};

Copy

**Code Snippet 2: Inheritance Example Program**

As shown in Code snippet 2,

* 1st we created an “employee” class which consists of public data member’s integer “id” and float “salary”.
* 2nd the “employee” class consists of a parameterized constructor that takes an integer “inpid” parameter and assigns its value to the data member “id”. The value of variable “salary” is set to “34”.
* 3rd the “employee” class also consists of default constructor.
* 4th we created a “programmer” class that is inheriting “employee” class. The main thing to note here is that the “visibility-mode” is “public”.
* 5ththe “programmer” class consists of public data member’s integer “languageCode”.
* 6th the “programmer” class consists of a parameterized constructor that takes an integer “inpid” parameter and assigns its value to the data member “id”. The value of variable “languageCode” is set to “9”.
* 7th “programmer” class consists of a function “getData” which will print the value of the variable “id”.

The main program is shown in code snippet 3.

int main()

{

Employee harry(1), rohan(2);

cout << harry.salary << endl;

cout << rohan.salary << endl;

Programmer skillF(10);

cout << skillF.languageCode<<endl;

cout << skillF.id<<endl;

skillF.getData();

return 0;

}

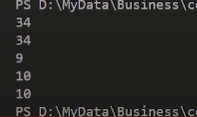
Copy

**Code Snippet 3: Main Program**

As shown in a code snippet 3,

* 1st objects “harry” and “rohan” is created of the “employee” data type. Object “harry” is passed with the value “1” and the object “rohan” is passed with the value “2”.
* 2nd the “salary” of both objects “rohan” and “harry” are printed.
* 3rd object “skillF” is created of the “programmer” data type. Object “skillF” is passed with the value “10”.
* 4th the “languageCode” and “id” of both object “skillF” is printed.
* 5th the function “getData” is called by the “skillF” object. This will print the “id”.

The output for the following program is shown in figure 1.



**Figure 1:**Program Output

**Single Inheritance Deep Dive: Examples + Code | C++ Tutorials for Beginners #38**

In this tutorial, we will discuss single inheritance in C++

**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.

An example program to demonstrate the concept of single inheritance in C++ is shown below.

class Base

{

int data1; // private by default and is not inheritable

public:

int data2;

void setData();

int getData1();

int getData2();

};

void Base ::setData(void)

{

data1 = 10;

data2 = 20;

}

int Base::getData1()

{

return data1;

}

int Base::getData2()

{

return data2;

}

Copy

**Code Snippet 1: Base Class**

As shown in a code snippet 1,

* 1st we created a “base” class which consists of private data member’s integer “data1” and public data member integer “data2”.
* 2nd the “base” class consists of three member functions “setData”, “getData1”, and “getData2”.
* 3rd the function “setData” will assign the values “10” and “20” to the data members “data1” and “data2”.
* 4th the function “getData1” will return the value of the data member “data1”.
* 5th the function “getData2” will return the value of the data member “data2”.

The derived class will inherit the base class which is shown below.

class Derived : public Base

{ // Class is being derived publically

int data3;

public:

void process();

void display();

};

void Derived ::process()

{

data3 = data2 \* getData1();

}

void Derived ::display()

{

cout << "Value of data 1 is " << getData1() << endl;

cout << "Value of data 2 is " << data2 << endl;

cout << "Value of data 3 is " << data3 << endl;

}

Copy

**Code Snippet 2: Derived Class**

As shown in Code snippet 2,

* 1st we created a “derived” class which is inheriting the base class publically. The “derived” class consists of private data member’s integer “data3”.
* 2nd the “derived” class consists of two public member functions “process” and “display”.
* 3rd the function “process” will multiply the values “data2” and “data1”; and store the values in the variable “data3”.
* 4th the function “display” will print the values of the data member “data1”, “data2”, and “data3”.

The main program is shown in code snippet 3.

int main()

{

Derived der;

der.setData();

der.process();

der.display();

return 0;

}

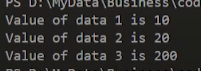
Copy

**Code Snippet 3: Main Program**

As shown in a code snippet 3,

* 1st object “der” is created of the “derived” data type.
* 2nd the function “setData” is called by the object “der”. This function will set the values of the data members “data1” and “data2”
* 3rd the function “process” is called by the object “der”. This function will multiply the values “data2” and “data1”; and store their value in the variable “data3”.
* 4th the function “display” is called by the object “der”. This function will print the values of the data member “data1”, “data2”, and “data3”.

The output for the following program is shown in figure 1.



**Figure 1:**Program Output

**Protected Access Modifier in C++ | C++ Tutorials for Beginners #39**

In this tutorial, we will discuss protected access modifiers in C++

**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.

An example program to demonstrate the concept of protected access modifiers is shown below.

#include<iostream>

using namespace std;

class Base{

protected:

int a;

private:

int b;

};

class Derived: protected Base{

};

int main(){

Base b;

Derived d;

// cout<<d.a; // Will not work since a is protected in both base as well as derived class

return 0;

}

Copy

**Code Snippet 1: Protected Access Modifier Example Program**

As shown in a code snippet 1,

* 1st we created a “Base” class which consists of protected data member integer “a” and private data member integer “b”.
* 2nd we created a “Derived” class which is inheriting the “Base” class in protected mode.
* 3rd the object “b” of the data type “Base” is created.
* 4th the object “d” of the data type “Derived” is created.
* 5th if we try to print the value of the data member “a” by using the object “d”; the program will throw an error because the data member “a” is protected and the derived class is inherited in the protected mode. So the data member “a” can only be accessed in the “derived” but not outside the class.

# Multilevel Inheritance Deep Dive with Code Example in C++ | C++ Tutorials for Beginners #40

In this tutorial, we will discuss multilevel inheritance in C++

**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.

An example program is shown below to demonstrate the concept of multilevel inheritance in C++.

#include <iostream>

using namespace std;

class Student

{

protected:

int roll\_number;

public:

void set\_roll\_number(int);

void get\_roll\_number(void);

};

void Student ::set\_roll\_number(int r)

{

roll\_number = r;

}

void Student ::get\_roll\_number()

{

cout << "The roll number is " << roll\_number << endl;

}

Copy

**Code Snippet 1: Student Class**

As shown in a code snippet 1,

* 1st we created a “student” class which consists of protected data member integer “roll\_number”.
* 2nd the “student” class consists of a public function “set\_roll\_number” and “get\_roll\_number”
* 3rd the function “set\_roll\_number” will set the value of the data member “roll\_number”.
* 4th the function “get\_roll\_number” will print the value of the data member “roll\_number”.

The code for the “exam” class is shown below which is inheriting the “student” class

class Exam : public Student

{

protected:

float maths;

float physics;

public:

void set\_marks(float, float);

void get\_marks(void);

};

void Exam ::set\_marks(float m1, float m2)

{

maths = m1;

physics = m2;

}

void Exam ::get\_marks()

{

cout << "The marks obtained in maths are: " << maths << endl;

cout << "The marks obtained in physics are: " << physics << endl;

}

Copy

**Code Snippet 2: Exam Class**

As shown in Code snippet 2,

* 1st we created an “exam” class that is inheriting “student” class in public mode.
* 2nd the “exam” class consists of protected data members float “math” and float “physics”.
* 3rd the “exam” class consists of public member functions “set\_marks” and “get\_marks”.
* 4th the function “set\_marks” will set the value of the data members “math” and “physics”.
* 5th the function “get\_marks” will print the value of the data members “math” and “physics”.

The code for the “result” class is shown below which is inheriting the “exam” class

class Result : public Exam

{

float percentage;

public:

void display\_results()

{

get\_roll\_number();

get\_marks();

cout << "Your result is " << (maths + physics) / 2 << "%" << endl;

}

};

Copy

**Code Snippet 3: Result Class**

As shown in a code snippet 3,

* 1st we created a “Result” class which is inheriting the “Exam” class in public mode.
* 2nd the “Result” class consists of private data member’s float “percentage”.
* 3rd the “exam” class consists of the public member function “display\_results”.
* 4th the function “display\_results” will call the “get\_roll\_number” and “get\_marks” functions, and add the values of “math” and “physics” variables then divide that value with “2” to get a percentage and prints it.

It can be clearly seen that the class “Exam” is inheriting class “student” and class “Results” is inheriting class “Exam”; which is an example of multilevel inheritance. The code main program is shown below.

int main()

{

Result harry;

harry.set\_roll\_number(420);

harry.set\_marks(94.0, 90.0);

harry.display\_results();

return 0;

}

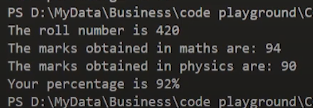
Copy

**Code Snippet 4: Main Program**

As shown in Code snippet 4,

* 1st object “harry” is created of the “Result” data type.
* 2nd the function “set\_roll\_number” is called by the object “harry” and the value “420” is passed.
* 3rd the function “set\_marks” is called by the object “harry” and the values “94.0” and “90.0” are passed.
* 4th the function “display\_results” is called by the object “harry”.

The output for the following program is shown in figure 1.



**Figure 1:**Program Output

**Multiple Inheritance Deep Dive with Code Example in C++ | C++ Tutorials for Beginners #41**

In this tutorial, we will discuss multiple inheritances in C++

**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. The syntax of inheriting multiple inheritances is shown below.

// class DerivedC: visibility-mode base1, visibility-mode base2

// {

// Class body of class "DerivedC"

// };

Copy

**Code Snippet 1: Multiple inheritances syntax**

As shown in a code snippet 1,

* After writing the class keyword we have to write the derived class name and then put a “:” sign.
* After “:” sign we have to write the visibility mode and then write the base class name and again we have to write the visibility mode and write another base class name.

An example program is shown below to demonstrate the concept of multiple inheritances in C++.

class Base1{

protected:

int base1int;

public:

void set\_base1int(int a)

{

base1int = a;

}

};

class Base2{

protected:

int base2int;

public:

void set\_base2int(int a)

{

base2int = a;

}

};

class Base3{

protected:

int base3int;

public:

void set\_base3int(int a)

{

base3int = a;

}

};

Copy

**Code Snippet 2: Base Classes**

As shown in Code snippet 2,

* 1st we created a “Base1” class which consists of protected data member integer “base1int”.
* 2nd the “Base1” class consists of a public function “set\_base1int”. This function will set the value of the data member “base1int”.
* 3rd we created a “Base2” class which consists of protected data member integer “base2int”.
* 4th the “Base2” class consists of a public function “set\_base2int”. This function will set the value of the data member “base2int”.
* 5th we created a “Base3” class which consists of protected data member integer “base3int”.
* 2nd the “Base3” class consists of a public function “set\_base3int”. This function will set the value of the data member “base3int”.

The code for the “Derived” class is shown below. “Derived” class will inherit all the base classes.

class Derived : public Base1, public Base2, public Base3

{

public:

void show(){

cout << "The value of Base1 is " << base1int<<endl;

cout << "The value of Base2 is " << base2int<<endl;

cout << "The value of Base3 is " << base3int<<endl;

cout << "The sum of these values is " << base1int + base2int + base3int << endl;

}

};

Copy

**Code Snippet 3: Derived Class**

As shown in a code snippet 3,

* 1st we created a “Derived” class which is inheriting “Base1”, “Base2”, and “Base3” classes in public mode.
* 2nd the “Derived” class consists of the public member function “show”.
* 4th the function “show” will first print the values of “base1int”, “base2int”, and “base3int” individually and then print the sum of all three values.

It can be clearly seen that the class “Derived” is inheriting class “Base1”, “Base2”, and “Base3”. This is an example of multiple inheritances. The code main program is shown below.

int main()

{

Derived harry;

harry.set\_base1int(25);

harry.set\_base2int(5);

harry.set\_base3int(15);

harry.show();

return 0;

}

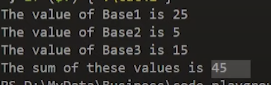
Copy

**Code Snippet 4: Main Program**

As shown in Code snippet 4,

* 1st object “harry” is created of the “Derived” data type.
* 2nd the function “set\_base1int” is called by the object “harry” and the value “25” is passed.
* 3rd the function “set\_base2int” is called by the object “harry” and the value “5” is passed.
* 4th the function “set\_base3int” is called by the object “harry” and the value “15” is passed.
* 4th the function “show” is called by the object “harry”.

The output for the following program is shown in figure 1.



**Figure 1:** Program Output

# Exercise on C++ Inheritance | C++ Tutorials for Beginners #42

#### Exercise on C++ Inheritance

As we have discussed a lot about inheritance and its types; we have also seen its working with example programs.  In this tutorial, I will give an exercise on C++ inheritance to be solved.

#### Questions

 You have to create 2 classes:

1. SimpleCalculator - Takes input of 2 numbers using a utility function and performs +, -, \*, / and displays the results using another function.
2. ScientificCalculator - Takes input of 2 numbers using a utility function and performs any four scientific operation of your chioice and displays the results using another function.
3. Create another class HybridCalculator and inherit it using these 2 classes

Also, answer the questions given below.

* What type of Inheritance are you using?
* Which mode of Inheritance are you using?
* Create an object of HybridCalculator and display results of simple and scientific calculator.
* How is code reusability implemented?

### Code file tut42.cpp as described in the video

#include<iostream>

using namespace std;

/\*

Create 2 classes:

1. SimpleCalculator - Takes input of 2 numbers using a utility function and performs +, -, \*, / and displays the results using another function.

2. ScientificCalculator - Takes input of 2 numbers using a utility function and performs any four scientific operations of your choice and displays the results using another function.

Create another class HybridCalculator and inherit it using these 2 classes:

Q1. What type of Inheritance are you using?

Q2. Which mode of Inheritance are you using?

Q3. Create an object of HybridCalculator and display results of the simple and scientific calculator.

Q4. How is code reusability implemented?

\*/

int main(){

**Ambiguity Resolution in Inheritance in C++ | C++ Tutorials for Beginners #43**

In this tutorial, we will discuss ambiguity resolution in inheritance in C++

**Ambiguity Resolution in Inheritance**

Ambiguity in inheritance can be defined as when one class is derived for two or more base classes then there are chances that the base classes have functions with the same name. So it will confuse derived class to choose from similar name functions. To solve this ambiguity scope resolution operator is used “::”. An example program is shown below to demonstrate the concept of ambiguity resolution in inheritance.

class Base1{

public:

void greet(){

cout<<"How are you?"<<endl;

}

};

class Base2{

public:

void greet()

{

cout << "Kaise ho?" << endl;

}

};

class Derived : public Base1, public Base2{

int a;

public:

void greet(){

Base2 :: greet();

}

};

Copy

**Code Snippet 1:** Ambiguity Resolution in Inheritance Example Program 1

As shown in a code snippet 1,

1. We have created a “Base1” class which consists of public member function “greet”. The function “greet” will print “how are you?”
2. We have created a “Base2” class which consists of public member function “greet”. The function “greet” will print “kaise ho?”
3. We have created a “Derived” class which is inheriting “Base1” and “Base2” classes. The “Derived” class consists of public member function “greet”. The function “greet” will run the “greet” function of the “Base2” class because we have used a scope resolution operator to let the compiler know which function should it run otherwise it will cause ambiguity.

The code of the main function is shown below

int main(){

// Ambibuity 1

Base1 base1obj;

Base2 base2obj;

base1obj.greet();

base2obj.greet();

Derived d;

d.greet();

return 0;

}

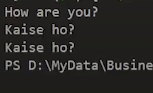
Copy

**Code Snippet 2:** Main program 1

As shown in code snippet 2,

1. Object “base1obj” is created of the “Base1” data type.
2. Object “base3obj” is created of the “Base2” data type.
3. The function “greet” is called by the object “base1obj”.
4. The function “greet” is called by the object “base2obj”.
5. Object “d” is created of the “Derived” data type.
6. The function “greet” is called by the object “d”.

The main thing to note here is that when the function “greet” is called by the object “d” it will run the “greet” function of the “Base2” class because we had specified it using scope resolution operator “::” to get rid ambiguity. The output for the following program is shown in figure 1.



**Figure 1: Output**

Another example of ambiguity resolution in inheritance is shown below.

class B{

public:

void say(){

cout<<"Hello world"<<endl;

}

};

class D: public B{

int a;

// D's new say() method will override base class's say() method

public:

void say()

{

cout << "Hello my beautiful people" << endl;

}

};

Copy

**Code Snippet 3:**  Ambiguity Resolution in Inheritance Example Program 2

As shown in a code snippet 3,

1. We have created a “B” class which consists of public member function “say”. The function “say” will print “hello world”
2. We have created a “D” class that is inheriting the “B” class. The “D” class consists of the public member function “say”. The function “say” will print “Hello my beautiful people”

The main thing to note here is that both “B” and “D” classes have the same function “say”, So if the class “D” will call the function “say” it will override the base class “say” method because compiler by default run the method which is already written in its own body. But if the function “say” was not present in the class “D” then the compiler will run the method of the class “B”.

The code of the main function is shown below,

int main(){

// Ambibuity 2

B b;

b.say();

D d;

d.say();

return 0;

}

Copy

**Code Snippet 4:** Main Program 2

As shown in code snippet 4,

1. Object “b” is created of the “B” data type.
2. The function “say” is called by the object “b”.
3. Object “d” is created of the “D” data type.
4. The function “say” is called by the object “d”.

The output for the following program is shown in figure 2.

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-43/Output_2.webp

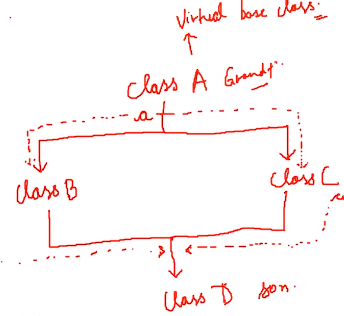
**Figure 2: Output**

**Virtual Base Class in C++ | C++ Tutorials for Beginners #44**

In this tutorial, we will discuss virtual base class in C++

**Virtual Base Class in C++**

The virtual base class is a concept used in multiple inheritances to prevent ambiguity between multiple instances. For example: suppose we created a class “A” and two classes “B” and “C”, are being derived from class “A”. But once we create a class “D” which is being derived from class “B” and “C” as shown in figure 1.



**Figure 1:** Virtual Base Class Example Diagram

As shown in figure 1,

1. Class “A” is a parent class of two classes “B” and “C”
2. And both “B” and “C” classes are the parent of class “D”

The main thing to note here is that the data members and member functions of class “A” will be inherited twice in class “D” because class “B” and “C” are the parent classes of class “D” and they both are being derived from class “A”.

So when the class “D” will try to access the data member or member function of class “A” it will cause ambiguity for the compiler and the compiler will throw an error. To solve this ambiguity we will make class “A” as a virtual base class. To make a virtual base class “virtual” keyword is used.

When one class is made virtual then only one copy of its data member and member function is passed to the classes inheriting it. So in our example when we will make class “A” a virtual class then only one copy of the data member and member function will be passed to the classes “B” and “C” which will be shared between all classes. This will help to solve the ambiguity.

The syntax of the virtual base class is shown in the code snippet below,

#include <iostream>

using namespace std;

class A {

public:

void say()

{

cout << "Hello world"<<endl;

}

};

class B : public virtual A {

};

class C : public virtual A {

};

class D : public B, public C {

};

Copy

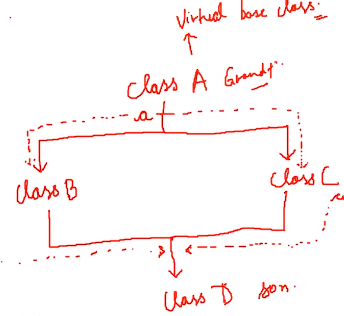
**Code Snippet 1:** Virtual Base Class Syntax Example Code

**Code Example Demonstrating Virtual Base Class in C++ | C++ Tutorials for Beginners #45**

In this tutorial, we will discuss demonstrating of virtual base class in C++

**Virtual Base Class in C++**

The virtual base class is a concept used in multiple inheritances to prevent ambiguity between multiple instances. For example: suppose we created a class “Student” and two classes “Test” and “Sports”, are being derived from class “Student”. But once we create a class “Result” which is being derived from class “Test” and “Sports” as shown in figure 1.



**Figure 1:** Virtual Base Class Example Diagram

As shown in figure 1,

1. Class “Student” is a parent class of two classes “Test” and “Sports”
2. And both “Test” and “Sports” classes are the parent of class “Result”

The main thing to note here is that the data members and member functions of class “Student” will be inherited twice in class “Result” because class “Test” and “Sports” are the parent classes of class “Result” and they both are being derived from class “Student”.

So when the class “Result” will try to access the data member or member function of class “Student” it will cause ambiguity for the compiler and the compiler will throw an error. To solve this ambiguity we will make class “Student” as a virtual base class. To make a virtual base class “virtual” keyword is used.

When one class is made virtual then only one copy of its data member and member function is passed to the classes inheriting it. So in our example when we will make class “Student” a virtual class then only one copy of data member and member function will be passed to the classes “Test” and “Sports” which will be shared between all classes. This will help to solve the ambiguity.

An example program of the following diagram is shown in a code snippet below,

#include<iostream>

using namespace std;

class Student{

protected:

int roll\_no;

public:

void set\_number(int a){

roll\_no = a;

}

void print\_number(void){

cout<<"Your roll no is "<< roll\_no<<endl;

}

};

class Test : public Student{

protected:

float maths, physics;

public:

void set\_marks(float m1, float m2){

maths = m1;

physics = m2;

}

void print\_marks(void){

cout << "You result is here: "<<endl

<< "Maths: "<< maths<<endl

<< "Physics: "<< physics<<endl;

}

};

class Sports: public Student{

protected:

float score;

public:

void set\_score(float sc){

score = sc;

}

void print\_score(void){

cout<<"Your PT score is "<<score<<endl;

}

};

class Result : public Test, public Sports{

private:

float total;

public:

void display(void){

total = maths + physics + score;

print\_number();

print\_marks();

print\_score();

cout<< "Your total score is: "<<total<<endl;

}

};

Copy

**Code Snippet 1:**Virtual Base Class Example Program

As shown in a code snippet 1,

1. We have created a “Student” class that consists of protected data member “roll\_no” and member functions “set\_number” and “print\_number”. The function “set\_number” will assign the value to the protected data member “roll\_no” and the function “print\_number” will print the value of data member “roll\_no”.
2. We have created a “Test” class that is inheriting the virtual base class “Student”. The “Test” consists of protected data members “maths” and “physics” and member functions “set\_marks” and “print\_marks”. The function “set\_number” will assign the values to the protected data members “maths” and “physics” and the function “print\_marks” will print the value of data members “maths” and “physics”.
3. We have created a “Sports” class that is inheriting the virtual base class “Student”. The “Sports” consists of protected data member “score” and member functions “set\_score” and “print\_score”. The function “set\_score” will assign the values to the protected data members “score” and “physics” and the function “print\_score” will print the value of data members “score”.
4. We have created a “Result” class which is inheriting base classes “Test” and “Sports”. The “Result” consists of protected data member “total” and member functions “display”. The function “display” will first add the values of data members “math”, “physics”, and “score” and assign the value to the protected data members “total” and second the “display” function will call the functions “print\_number”, ”print\_marks”, and “print\_score” and also print the value of the data member “total”.

The code of the main function is shown below,

int main(){

Result harry;

harry.set\_number(4200);

harry.set\_marks(78.9, 99.5);

harry.set\_score(9);

harry.display();

return 0;

}

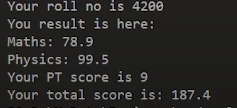
Copy

**Code Snippet 2:**Main Program

As shown in code snippet 2,

1. Object “harry” is created of the “Result” data type.
2. The function “set\_number” is called by the object “harry” and the value “4200” is passed.
3. The function “set\_marks” is called by the object “harry” and the values “48.9” and “99.5” are passed.
4. The function “set\_score” is called by the object “harry” and the value “9” is passed.
5. The function “display” is called by the object “harry”.

The main thing to note here is that there will be no ambiguity because we have made the “Student” class as a virtual base class but if we remove the “virtual” keyword then the compare will throw an error. The output of the following program is shown below.



**Figure 2:** Program Output

< < Previous

**Constructors in Derived Class in C++ | C++ Tutorials for Beginners #46**

In this tutorial, we will discuss constructors in derived class in C++

**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)

Copy

**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

**Solution to Exercise on Cpp Inheritance | C++ Tutorials for Beginners #47**

A solution to Exercise on Inheritance in C++

As I have given you an exercise on inheritance to solve in the previous tutorial. In this tutorial, we will see the solution to that exercise. So the question was to make three classes “SimpleCalculator”, “ScientificCalculator” and “HybridCalculator”.

* In “SimpleCalculator” class you have to take input of 2 numbers and perform function (+, -, \*, /)
* In “ScientificCalculator” class you have to take input of 2 numbers and perform any 4 scientific operations
* You have to inherit both “SimpleCalculator” and “ScientificCalculator” classes with the “HybridCalculator” class. You have to make an object of the “HybridCalculator” class and display the results of “SimpleCalculator” and “ScientificCalculator” classes.

The solution to the above Question is shown below,

class SimpleCalculator {

int a, b;

public:

void getDataSimple()

{

cout<<"Enter the value of a"<<endl;

cin>>a;

cout<<"Enter the value of b"<<endl;

cin>>b;

}

void performOperationsSimple(){

cout<<"The value of a + b is: "<<a + b<<endl;

cout<<"The value of a - b is: "<<a - b<<endl;

cout<<"The value of a \* b is: "<<a \* b<<endl;

cout<<"The value of a / b is: "<<a / b<<endl;

}

};

Copy

**Code snippet 1:** Simple Calculator Class

As shown in a code snippet 1,

1. We created a class “SimpleCalculator” which contains two private data members “a” and “b”
2. The class “SimpleCalculator” contains two member functions “getDataSimple” and “performOperationsSimple”
3. The function “getDataSimple” will take 2 numbers as input
4. The function “performOperationsSimple” will perform the operations “+, -, \*, /”

class ScientificCalculator{

int a, b;

public:

void getDataScientific()

{

cout << "Enter the value of a" << endl;

cin >> a;

cout << "Enter the value of b" << endl;

cin >> b;

}

void performOperationsScientific()

{

cout << "The value of cos(a) is: " << cos(a) << endl;

cout << "The value of sin(a) is: " << sin(a) << endl;

cout << "The value of exp(a) is: " << exp(a) << endl;

cout << "The value of tan(a) is: " << tan(a) << endl;

}

};

Copy

**Code Snippet 2:** Scientific Calculator Class

As shown in code snippet 2,

1. We created a class “ScientificCalculator” which contains two private data members “a” and “b”
2. The class “ScientificCalculator” contains two member functions “getDataScientific” and “performOperationsScientific”
3. The function “getDataScientific” will take 2 numbers as input
4. The function “performOperationsScientific” will perform the operations “cos, sin, exp, tan”

class HybridCalculator : public SimpleCalculator, public ScientificCalculator{

};

Copy

**Code Snippet 3:**Hybrid Calculator Class

As shown in code snippet 3, we created a “HybridCalculator” class which is inheriting the “SimpleCalculator” class and “ScientificCalculator” class.

int main()

{

HybridCalculator calc;

calc.getDataScientific();

calc.performOperationsScientific();

calc.getDataSimple();

calc.performOperationsSimple();

return 0;

}

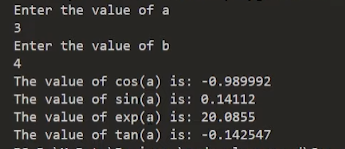
Copy

**Code Snippet 4:**Main Program

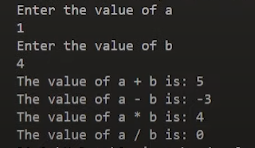
As shown in code snippet 4,

1. We created an object “calc” of the data type “hybridCalculator”
2. The function “getDataScientific” is called using the object “calc”
3. The function “performOperationsScientific” is called using the object “calc”
4. The function “getDataSimple” is called using the object “calc”
5. The function “performOperationsSimple” is called using the object “calc”

The output of the following program is shown in the figure below,



**Figure 1:**Program Output 1



**Figure 2:**Program Output 2

Q1. What type of Inheritance are you using?

Ans. Multiple inheritances

Q2. Which mode of Inheritance are you using?

Ans. public SimpleCalculator, public ScientificCalculator

**Code Example: Constructors in Derived Class in Cpp | C++ Tutorials for Beginners #48**

In this tutorial, we will discuss constructors in derived class with code example in C++

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

As we have discussed before about the constructors in derived class in a code snippet below three cases are given to clarify the execution of constructors.

/\*

Case1:

class B: public A{

// Order of execution of constructor -> first A() then B()

};

Case2:

class A: public B, public C{

// Order of execution of constructor -> B() then C() and A()

};

Case3:

class A: public B, virtual public C{

// Order of execution of constructor -> C() then B() and A()

};

\*/

Copy

**Code Snippet 1:**Constructors Execution Example Cases

As shown in Code Snippet 1,

1. In case 1, class “B” is inheriting class “A”, so the order of execution will be that first the constructor of class “A” will be executed and then the constructor of class “B” will be executed.
2. In case 2, class “A” is inheriting two classes “B” and “C”, so the order of execution will be that first constructor of class “B” will be executed and then the constructor of class “C” will be executed and at the end constructor of class “A” will be executed.
3. In case 3, class “A” is inheriting two classes “B” and virtual class “C”, so the order of execution will be that first constructor of class “C” will be executed because it is a virtual class and it is given more preference and then the constructor of class “B” will be executed and at the end constructor of class “A” will be executed.

To demonstrate the concept of constructors in derived classes an example program is shown below.

class Base1{

int data1;

public:

Base1(int i){

data1 = i;

cout<<"Base1 class constructor called"<<endl;

}

void printDataBase1(void){

cout<<"The value of data1 is "<<data1<<endl;

}

};

class Base2{

int data2;

public:

Base2(int i){

data2 = i;

cout << "Base2 class constructor called" << endl;

}

void printDataBase2(void){

cout << "The value of data2 is " << data2 << endl;

}

};

class Derived: public Base2, public Base1{

int derived1, derived2;

public:

Derived(int a, int b, int c, int d) : Base2(b), Base1(a)

{

derived1 = c;

derived2 = d;

cout<< "Derived class constructor called"<<endl;

}

void printDataDerived(void)

{

cout << "The value of derived1 is " << derived1 << endl;

cout << "The value of derived2 is " << derived2 << endl;

}

};

Copy

**Code Snippet 2:** Constructors in Derived Class Example Program

As shown in code snippet 2,

1. We have created a “Base1” class which consists of private data member “data1” and parameterized constructor which takes only one argument and set the value of data member “data1”. The “Base1” class also contains the member function “printDataBase1” which will print the value of data member “data1”.
2. We have created a “Base2” class which consists of private data member “data2” and parameterized constructor which takes only one argument and set the value of data member “data2”. The “Base2” class also contains the member function “printDataBase2” which will print the value of data member “data2”.
3. We have created a “Derived” class that is inheriting base classes “Base1” and “Base2”. The “Derived” class consists of private data members “derived1” and “derived2”. The “Derived” class contains parameterized constructor which calls the “Base1” and “Base2” class constructors to pass the values, it also assigns the values to the data members “derived1” and “derived2”. The “Derived” class also contains member functions “printDataDerived”. The function “printDataDerived” will print the values of the data member “derived1” and “derived2”.

The main thing to note here is that the constructors will be executed in the order in which the classes are being inherited. As in the example program above the “Base2” class is being inherited first and then “Base1” class is being inherited, so the constructor of “Base2” class will be executed first. The main program of the following example code is shown below.

int main(){

Derived harry(1, 2, 3, 4);

harry.printDataBase1();

harry.printDataBase2();

harry.printDataDerived();

return 0;

}

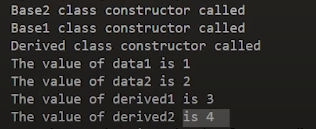
Copy

**Code Snippet 3:** Main Program

As shown in code snippet 3,

1. Object “harry” is created of the “Derived” data type and the values (1, 2, 3, 4) are passed.
2. The function “printDataBase1” is called by the object “harry”.
3. The function “printDataBase2” is called by the object “harry”.
4. The function “printDataDerived” is called by the object “harry”.

The output of the following program is shown below,



**Figure 1:** Program Output

**Initialization list in Constructors in Cpp | C++ Tutorials for Beginners #49**

In this tutorial, we will discuss the Initialization list in Constructors in C++

**Initialization list in Constructors in C++**

The initialization list in constructors is another concept of initializing the data members of the class. The syntax of the initialization list in constructors is shown below.

/\*

Syntax for initialization list in constructor:

constructor (argument-list) : initilization-section

{

assignment + other code;

}

Copy

**Code Snippet 1:** Initialization list in Constructors Syntax

As shown in a code snippet 1,

1. A constructor is written first and then the initializations section is written
2. In the initialization section, the data members are initialized

To demonstrate the concept of Initialization list in Constructors an example program is shown below,

class Test

{

int a;

int b;

public:

Test(int i, int j) : a(i), b(j)

{

cout << "Constructor executed"<<endl;

cout << "Value of a is "<<a<<endl;

cout << "Value of b is "<<b<<endl;

}

};

int main()

{

Test t(4, 6);

return 0;

}

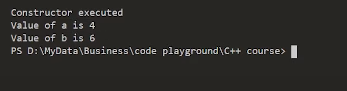
Copy

**Code Snippet 2:** Initialization list in Constructors Example Program 1

As shown in code snippet 2,

1. We have created a “test” class that consists of private data member “a” and “b” and parameterized constructor which takes two arguments and sets the value of data member “a” and “b” by using the initialization list. The constructor will also print the value of data member “a” and “b”.
2. In the main program object “t” is created of the “test” data type and the values (4, 6) are passed.

The output of the following program is shown below,



**Figure 1:** Program Output

**Main Points**

The main thing to note here is that if we use the code shown below to initialize data members the compiler will throw an error because the data member “a” is being initialized first and the “b” is being initialized second so we have to assign the value to “a” data member first.

Test(int i, int j) : b(j), a(i+b)

Copy

**Code Snippet 3:**Initialization list in Constructors Example 1

But if we use the code shown below to initialize data members the compiler will not throw an error because the data member “a” is being initialized first and we are assigning the value to the data member “a” first.

Test(int i, int j) : a(i), b(a + j)

Copy

**Code Snippet 4:** Initialization list in Constructors Example 2