Inheritance basics

**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 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...

}

**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;

}

};

**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;

}

**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”.

#### 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;

}

**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;

}

**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;

}

**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”.

**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;

}

**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 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;

}

**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;

}

**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;

}

};

**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;

}

**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”.