Inheritance:

==>Inheritance is one of the most important features of Object-Oriented Programming.

==>The capability of a class to derive properties and characteristics from another class is called Inheritance.

==>Inheritance is a feature or a process in which, new classes are created from the existing classes.

The new class created is called "derived class" or "child class" and the existing class is known as the "base class” or “parent class”.

The derived class now is said to be inherited from the base class.

==>When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class,

without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class.

The derived class is the specialized class for the base class.

==>It not only helps to reuse the old code but adds in extendings the functionality. i.e Reusability + Extensibility

Reusability:

Once a class is written and tested,it can be further used for creating new classes.These derived classes not only inherit the features of their base class,

but also have their own individualist features.

it means that if the derived class wants to use its base class properties,it can do so because those properties are also available to the deried class by the virtue of inheritance.

Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

.

Extensibility:

It is the mechanism of being able to derive classes from existing classes that provides extensibility of adding and removing classes in a hierarchy

as and when required.Any changes to data or functionalities contained within a base class are immediately inherited by all derived classes.

===================================================================================================================================================================================

==>each child has "is a " relationship with its parent.

==>Sub Class: The class that inherits properties from another class is called Subclass or Derived Class.

==>Super Class: The class whose properties are inherited by a subclass is called Base Class or Superclass.

Syntax:

class <derived\_class\_name> : < access-specifier used in modes of inheritance > <base\_class\_name>

{

//body

}

access-specifier — either of private, public or protected. If neither is specified, PRIVATE is taken as default

There are three Access specifiers in C++. These are:

Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members.

Note: If we do not specify any access modifiers for the members inside the class, then by default the access modifier for the members will be Private.

public – members are accessible from outside the class, and members can be accessed from anywhere.

The data members and member functions declared as public can be accessed by other classes and functions too.

The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

// C++ program to demonstrate public

// access modifier

#include<iostream>

using namespace std;

// class definition

class Circle

{

public:

double radius;

double compute\_area()

{

return 3.14\*radius\*radius;

}

};

// main function

int main()

{

Circle obj;

// accessing public data member outside class

obj.radius = 5.5;

cout << "Radius is: " << obj.radius << "\n";

cout << "Area is: " << obj.compute\_area();

return 0;

}

Output:

Radius is: 5.5

Area is: 94.985

In the above program, the data member radius is declared as public so it could be accessed outside the class and thus was allowed access from inside main().

======================================================================================

===============================================================================================

private – members cannot be accessed (or viewed) from outside the class, i.e members are private to that class only.

The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any

object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of the class.

// C++ program to demonstrate private access modifier

#include<iostream>

using namespace std;

class Circle

{

// private data member

private:

double radius;

// public member function

public:

double compute\_area()

{ // member function can access private

// data member radius

return 3.14\*radius\*radius;

}

};

// main function

int main()

{

// creating object of the class

Circle obj;

// trying to access private data member

// directly outside the class

obj.radius = 1.5;//error

cout << "Area is:" << obj.compute\_area();

return 0;

}

The output of the above program is a compile time error because we are not allowed to access the private data members of a class directly from outside the class.

Yet an access to obj.radius is attempted, but radius being a private data member, we obtained the above compilation error.

However, we can access the private data members of a class indirectly using the public member functions of the class.

=========================================================================================

protected – members cannot be accessed from outside the class, but, they can be accessed in inherited classes or derived classes.

The protected access modifier is similar to the private access modifier in the sense that it can’t be accessed outside of its class unless with the help of a friend class.

The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

// C++ program to demonstrate protected access modifier

#include <iostream>

using namespace std;

class Parent

{

protected:

int id\_protected;

};

class Child : public Parent

{

public:

void setId(int id)

{

// Child class is able to access the inherited protected data members of base class

id\_protected = id;

}

void displayId()

{

cout << "id\_protected is: " << id\_protected << endl;

}

};

int main() {

Child obj1;

obj1.id\_protected;//check in lab

// member function of the derived class can access the protected data members of the base class

obj1.setId(81);

obj1.displayId();

}

Output:

id\_protected is: 81

=============================================================================================

Note:

When derived class obj is created memory is allocated for base class datamembers as well as for derived class data members.

Through derived class obj, we can access base class members except private as well as derived class members.

// C++ program to demonstrate implementation of Inheritance

#include <iostream>

using namespace std;

class Parent {

public:

int id\_p;

};

class Child : public Parent {

public:

int id\_c;

void display()

{

cout<<id\_p;//allowed

}

};

int main()

{

Child obj1;

// An object of class child has all data members and member functions of class parent

obj1.id\_c = 7;//allowed

obj1.id\_p = 91;//allowed

cout << "Child id is: " << obj1.id\_c << '\n';

cout << "Parent id is: " << obj1.id\_p << '\n';

cout<<sizeof(obj1);

}

====================================================================

// C++ program to demonstrate implementation of Inheritance

#include <iostream>

using namespace std;

class Parent {

public:

int id\_p;

};

class Child : public Parent {

public:

int id\_c;

void display()

{

cout<<id\_p;//allowed

}

};

int main()

{

Child obj1;

// An object of class child has all data members and member functions of class parent

obj1.id\_c = 7;//allowed

obj1.id\_p = 91;//allowed

cout << "Child id is: " << obj1.id\_c << '\n';

cout << "Parent id is: " << obj1.id\_p << '\n';

cout<<sizeof(obj1);

}

=================================================

// C++ program to demonstrate implementation of Inheritance

#include <iostream>

using namespace std;

class Parent

{

protected:

int id\_p;

};

class Child : public Parent {

public:

int id\_c;

void display()

{

cout<<id\_p;// allowed

}

};

int main()

{

Child obj1;

// An object of class child has all data members and member functions of class parent

obj1.id\_c = 7;//allowed

//obj1.id\_p = 91;//not allowed

cout << "Child id is: " << obj1.id\_c << '\n';

// cout << "Parent id is: " << obj1.id\_p << '\n';

cout<<sizeof(obj1);//8

}

===========================================================================

Note 2:

When base class obj is created memory is allocated for base class datamembers only. Through base class obj, we can access base class members only ,we cannot access derived class members.

#include <iostream>

using namespace std;

class Parent {

public:

int id\_p;

};

class Child : public Parent {

public:

int id\_c;

};

int main()

{

Parent obj1;

// obj1.id\_c = 7;//error

obj1.id\_p = 91;

// cout << "Child id is: " << obj1.id\_c << '\n';//error

cout << "Parent id is: " << obj1.id\_p << '\n';

cout<<sizeof(obj1);

}

==============================================================================================

Modes of Inheritance: There are 3 modes of inheritance.

Modes of inheritance specifies the way in which a class is derived. It tells abouth the access rights given to the derived class to access its base class attributes and functions.

These mode are:

private(default)

public

protected

ex:

class derived\_class\_name :: visibility-mode base\_class\_name

{

// body of the derived class.

}

Where,

derived\_class\_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

Example:

1. class ABC : private XYZ //private derivation

{ }

2. class ABC : public XYZ //public derivation

{ }

3. class ABC : protected XYZ //protected derivation

{ }

4. class ABC: XYZ //private derivation by default

{ }

Public Mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

1. C++ public Inheritance

In this example, public inheritance is demonstrated. Since private and protected members will not be directly accessed from main( )

so we have had to create functions name getPVT( ) to access the private variable and getProt( ) to access the protected variable from the inherited class.

Example:

// C++ program to demonstrate the working of public

// inheritance

#include <iostream>

using namespace std;

class Base {

private:

int pvt = 1;

protected:

int prot = 2;

public:

int pub = 3;

int getPVT() { return pvt; }

};

class PublicDerived : public Base {

public:

// function to access protected member from Base

int getProt() { return prot; }

//public: pub=3,getpvt(); //dont run

//protected:prot;//dont run

};

int main()

{

PublicDerived object1;

cout << "Private = " << object1.getPVT() << endl;

cout << "Protected = " << object1.getProt() << endl;

cout << "Public = " << object1.pub << endl;

//stat which will generate error

//cout<<object1.prot ;//error

//cout<<object1.pvt ;//error

return 0;

}

=========================================================================================

Protected Mode: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

We know that protected members can only be accessed from the Derived class. These members cannot be directly accessed from outside the class. So we cannot use getPVT() from ProtectedDerived.This is also why we need to create getPub() function in the Derived class in order to access the pub variable.

#include <iostream>

using namespace std;

class Base {

private:

int pvt = 1;

protected:

int prot = 2;

public:

int pub = 3;

int getPVT() { return pvt; }

};

class ProtectedDerived : protected Base {

public:

int getProt() { return prot; }

int getPub() { return pub; }

//protected: prot,pub,getpvt means they cannot be called with the helpof

//derived class object

};

int main()

{

ProtectedDerived object1;

cout << "Protected = " << object1.getProt() << endl;

cout << "Public = " << object1.getPub() << endl;

// cout << "Public = " << object1.getPVT() << endl; -------->ERROR

return 0;

}

==========================================================================

Private Mode: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

Note: The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.

/ C++ program to demonstrate the working

// of private inheritance

#include <iostream>

using namespace std;

class Base {

private:

int pvt = 1;

protected:

int prot = 2;

public:

int pub = 3;

int getPVT() { return pvt; }

};

class PrivateDerived : private Base {

public:

int getProt() { return prot; }

int getPub() { return pub; }

//PRIVATE:prot,pub,getPVT();

};

int main()

{

PrivateDerived object1;

cout << "Protected = " << object1.getProt() << endl;

cout << "Public = " << object1.getPub() << endl;

//cout << "PPRIVATE = " << object1.getPVT() << endl; //ERROR

}

===================================================================================================

When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class.

Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.

ex:

#include<iostream>

using namespace std;

class A

{

int c;

public:

int a;

protected:

int b;

public:

void display()

{

cout<<"in display of A\n";

}

};

class B:private A

{

public:

int c;

public:

void show()

{

display();//allowed

cout<<a<<b;

}

};

class C:private B

{

public:

void show()

{

// display(); not allowed

//cout<< a<<b; not allowed

cout<<c;

}

};

int main()

{

B bobj;

bobj.show();

// bobj.display(); not allowed

}

In the above example, class A is privately inherited. Therefore, the display() function of class 'A' cannot be accessed by the object of class B. It can only be accessed by the member function of class B.

How to make a Private Member Inheritable

The private member is not inheritable. If we modify the visibility mode by making it public, but this takes away the advantage of data hiding.

C++ introduces a third visibility modifier, i.e., protected. The member which is declared as protected will be accessible to all the member functions within the class as well as the class immediately derived from it.

====================================================================================

When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the derived class.

#include<iostream>

using namespace std;

class A

{

int c;

public:

int a;

protected:

int b;

public:

void display()

{

cout<<"in display of A\n";

}

/\*protected:

void profun()

{

cout<<"in pro\n";

}\*/

};

class B:public A

{

public:

public:

void show()

{

display();

cout<<a<<b;

}

};

class C:public B

{

public:

void show()

{

cout<< a<<b;

}

};

int main()

{

B bobj;

bobj.show();

bobj.display();

//bobj.profun();

}

=====================================================================

When the base class is protectedly inherited by the derived class, the public members of the base class are not accessible by the objects of the derived class

(because public members of base class becomes protected in derived class and protected datamember cannot be accessed outside the class) only by the member functions of the derived class.

#include<iostream>

using namespace std;

class A

{

int c;

public:

int a;

protected:

int b;

void display()

{

cout<<"in display of A\n";

}

};

class B:protected A

{

public:

int d;

public:

void show()

{

B bobj;

bobj.display();

cout<<a<<b;

}

//PROTECTED: a,b,display()

};

int main()

{

B bobj;

bobj.show();

}

==================================================================================