Logging in From a Linux System or Localhost
4 days 42 min ago
Auto response
4 days 6 hours ago
Re: final issue.
4 days 22 hours ago
Re: Rocky, thank you for this
4 days 23 hours ago

Learning C/C++ Step-By-Step - Page 14

Submitted by ganesh35 (Contact Author) (Forums) on Wed, 2009-01

14. Step-by-Step C/C++ --- C++ Programming - Inheritanc

Inheritance

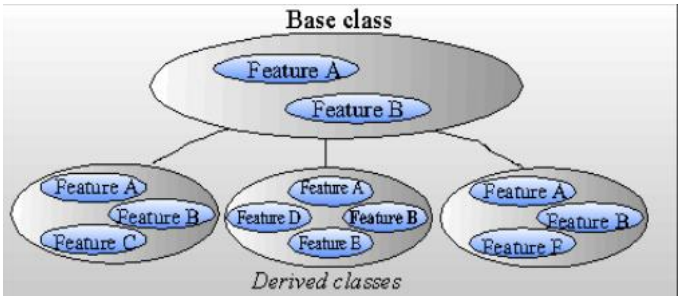
Introduction

Inheritance is the most powerful feature of Object Oriented programming. Inheritance is the process of creating new classes, called derived classes from existing or bases classes. The derived class inherits all the capabilities of the base class but can add embellishments and refinements of its own.
A class, called the derived class, can inherit the features of another class, called the base class.

To inherit the qualities of base class to derived class is known as inheritance.
Its noun is heritage. We know in our daily lives, we use the concept of classes being derived into subclasses. For E.g. Vehicle is class it's again divided into Cycles, Bikes, Autos, trucks, busses and so on.

Here Vehicle is known as Base class and the derived items are known as derived classes or subclasses.

Generally every base class has a list of qualities and features. The main theme in this inheritance is to share all the common characteristics of base class to derived classes.



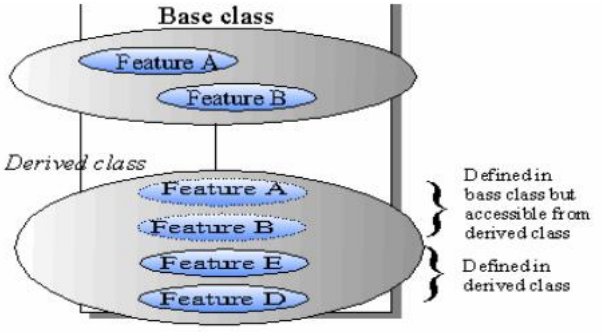
Inheritance has an important feature to allow reusability. One result of reusability is the ease of distributing class libraries. A programmer can use a class created another person or company, and, without modifying it, derive other classes from it that are suited to particular situations.

Derived class and Base class

A class, called the derived class, can inherit the features of another class, called the base class.
The derived class can add other features of its own, so it becomes a specialized version of the base class. Inheritance provides a powerful way to extend the capabilities of existing classes, and to design programs using hierarchical relationships.

Accessibility of base class members from derived classes and from objects of derived classes is an important issue. Objects of derived classes can access data or functions in the base class that are prefaced by the keyword protected from derived classes but not. Classes may be publicly privately derived from base classes. Objects of a publicly derived class can access public members of the base class, while objects of a privately derived class cannot.

Diagram shows how Derived class inherits.



A class can be derived from more than one base class. This is called multiple inheritances. A class can also be contained within another class.

Specifying the Derived Class

Class declaration is so easy using the keyword **class** as well as the derived class declaration is also easy but the class must be ends with its base class id and access specifier.
Syntax to declare a derived class:

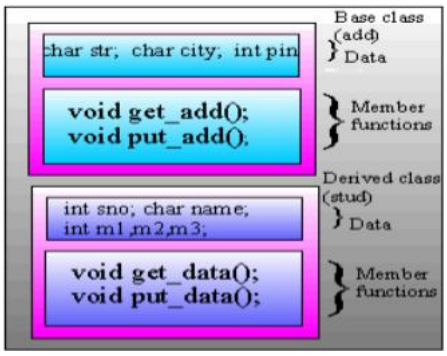
Class <*Class_name*> : <*Access_Specifier*> <*Base_Class_Name*> ....

For. E.g. **class** result **: public stud;**

```
/*  program to accept and display a student record  */
#include <iostream>
using namespace std;
class add
{
     private :
            char str[20];
            char city[20];
            int pin;
     public :
            void get_add()
            {
                 cout << "Enter Address  street,city,pin";
                 cin >> street >>city>>pin;
            }
            void put_data()
            {
                  cout << "Address is   "<< str
                       <<endl<<city  <<endl<<pin;
            }
};
class stud  : public add
{
      private :
             int sno;
             char name[20];
             int m1,m2,m3;
      public :
             void get_data()
             {
                   cout << "Enter Student No.  "; cin >> sno;
                   cout << "Enter Student Name  "; cin >> name;
                   cout << "Enter Student 3subjects marks  ";
                   cin >> m1 << m2 << m3;
             }
             void put_data()
             {
                   cout  << "Student number  :" << sno;
                   cout << "Student name      :" << name;
                   cout << "Student marks     :" << m1 << "  " <<m2<<"  "<<m3;
             }
};
int main()
{
```

```
        stud s;
        s.get_add();
        s.get_data();
        s.put_add();
        s.put_data();
        return 0;
}
```

Diagramed explanation for the above program



Derived Class Constructors

If a class is declared with its own constructors it is a base class of another. The derived class is also having its own constructors. If an object is declared which is the constructor will be executed? No doubt it executed the constructor of the derived class. It you still want to execute the constructor of Base class or both Derived and Base class constructors simply call the Base constructor in Derived class constructor.

```
/* Constructors in derived class */
#include <iostream>
using namespace std;
class Add
{
     protected :                                // NOTE : not private
            unsigned int a;
     public :
            Add()  {  a = 0; }                  // constructor , no args
            Add( int c ) { a = c; }             // constructor , one args
            int get_val(){     return a; }      // return A value
            Add operator ++ ()                  // increment count
            {
                 a++;                           // increment count, return
                 return Add(a);                 // an unnamed temporary object
            }                                   // initialized to this count
};

class Sub : public Add
{
      public:
           Sub() : Add() { }                     // Constructor, no args
           Sub(int c) : Add(c) { }               // Constructor, one args
           Sub operator -- ()                    // decrement value of A, return
           {                                     // an unnamed temporary object
                a--;                             //initialized to this Value
                return Sub(a);
           }
};
int main()
{
      Sub ob1;                                  // class Sub
      Sub ob2(100);
      cout << "\nOb1  =" << ob1.get_val();      // display
      cout << "\nOb2  =" << ob2.get_val();      // display
      ob1++; ob1++; ob1++;                      // increment ob1
      cout << "\nOb1  =" << ob1.get_val();      // display
      ob2--; ob2--;                             // decrement ob2
      cout << "\nOb2  =" << ob2.get_val();      // display
      Sub ob3=ob2--;                            // create ob3 from ob2
      cout << "\nOb3  =" << ob3.get_val();      // display
      return 0;
}
```

ACCESS SPECIFIERS

Access specifiers are used to control, hide, secure the both data and member functions. Access specifiers are of 3 types

1. Public Access Specifier
2. Private Access Specifier

3. Protected Access Specifier.

Public :

If a member or data is a public it can be used by any function with in class and its derived classes also.
In **C++** members of a **struct** or **union** are public by default.
Public Member of a class can be inherited to the derived class when the class is inherited publicly but not the member functions(privately).

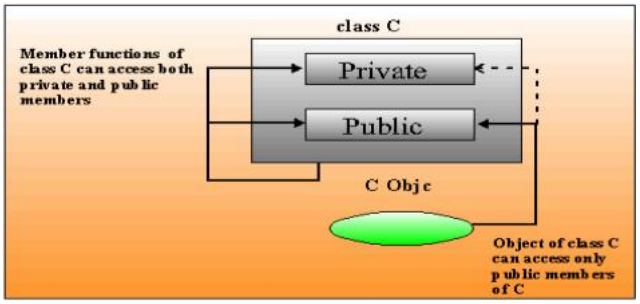
Private :

Member functions and friend of the class in which it is declared can only use it.
Members of a class are private by default.
Private member of a class doesn't be inherited to a derived class when the base class is inherited publicly or privately. It there is need we have to write member function, which are returns, those values.

Protected :

It is access as the same as for private in addition, the member can be used by member functions and friends of classes derived from the declared class but not only in Objects of the derived type.
The protected member of a class can be inherited to the next derived class only. But not to the later classes.

Access Specifiers without Inheritance



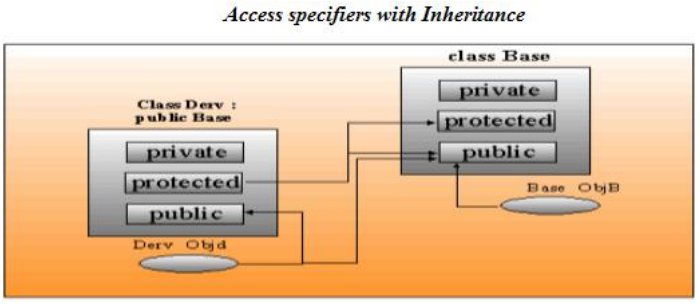
More About Protected Access Specifier

To provide the functionality without modifying the class. ***Protected*** can accessed by it self and derived class-protected members only but in objects or the subderived class or the outside class.

Scope of Access Specifiers

Access Specifier	Accessible from Own class	Accessible from derived class	Accessible from Objects outside class
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

Access specifiers with Inheritance



Types of Inheritance

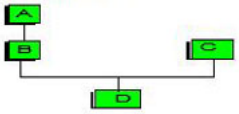
Types of Inheritance

1. **Single Inheritance :**
   If a class is derived from a base class is called single inh
   A → B

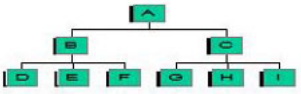
2. **Multiple Inheritance :**
   If a class is derived from more than one Base class is called Multiple Inheritance.

3. **Multilevel Inheritance :**
   If a class is derived from a derived class is called Multilevel Inheritance.

4. **Hybrid Inheritance :**
   The class with a combination of both Multilevel and Multiple Inheritances is known as Hybrid Inheritance.



5. **Hierarchy Inheritance :**
   It consists of a Base class and its multiple derived classes. The Base class has the ability to control all the derived classes.




```
/* Program to demonstrate Multiple Inheritance  */
#include <iostream>
using namespace std;
class M
{
     protected :
            int m;
     public :
            void getm()
            {
                  cout << "\nEnter M value  :";
                  cin >> m;
            }
};
class N
{
     protected :
            int n;
     public :
            void getn()
            {
                  cout << "\nEnter N value  :";
                  cin >> n;
            }
};
class P : public N, public M
{
     public :
            void disp()
            {
                  cout << "\n  M   =  " << m;
                  cout << "\n  N   =  " << n;
                  cout << "\n  M*N =  " << m*n;
            }
};
int main()
{
```

```
      P p;
      p.getm();
      p.getn();
      p.disp();
      return 0;
}
```

If a base class is publicly inherited then the public members, member function can be accessible to the member functions of the derived class and to the Objects also where as If a base class is inherited privately then the public member of base class are inherited to the member functions of the derived class only but not to the objects.

```
/*  A program to demonstrate Multilevel Inheritance  */
class student
{
          int rno;
      public:
          void getrno()
          {
               cout << "Enter  Number :";
               cin >> rno;
          }
          void showrno()
          {
               cout << "Student Number:" << rno;
          }
};
class test : public student
{
          int m1,m2;
      public :
          void getmarks()
          {
                cout << "Enter marks 1  :" ; cin >> m1;
                cout << "Enter marks 2  :" ; cin >> m2;
          }
          int  retm1()
          {
                return m1;
          }
          int retm2()
          {
                return m2;
          }
};
class result : public test
{
          int tot;
      public:
          void get()
          {
                getrno();
                getmarks();
          }
          void showresult();
          void show()
          {
                showrno();
                showresult();
          }
};
void result::showresult()
{
      int s1,s2;
      s1=retm1();
      s2=retm2();
      tot=s1+s2;
      cout << "\nMarks  " << s1 << "  "<< s2;
      cout << "\nTotal marks  " << tot;
}
int main()
{
      result a;
      a.get();
      a.show();
      return 0;
}
```

```
/*  Program to demonstrate Hybrid Inheritance  */
#include <iostream>
using namespace std;
class student
{
          int rno;
     public:
          void getrno()
          {
                cout << "Enter  Number :";
                cin >> rno;
          }
          void showrno()
          {
                cout << "\nStudent Number   :" << rno;
          }
};
class test : public student
{
     protected :
          int m1,m2;
     public :
          void getmarks()
          {
                cout << "Enter marks 1  :" ; cin >> m1;
                cout << "Enter marks 2  :" ; cin >> m2;
          }
          void showmarks()
          {
                cout << "\nMarks  of 2 subjects " << m1 << "  " << m2;
          }
};
class sports
{
     protected :
          int score;
```

```
     public :
          void getscore()
          {
               cout << "Enter   Score  :";
               cin >> score;
          }
};
class result : public test, public sports
{
     public :
          void getdata()
          {
               getrno();
               getmarks();
               getscore();
          }
          void putdata()
          {
               showrno();
               showmarks();
               cout << "\nScore  is  " << score;
               cout << "\n Total marks   " << m1+m2;
          }
};
int main()
{
      result  r;
      r.getdata();
      r.putdata();
      return 0;
}
```

Pictorial representation of the above program:

In the above figure student class inherited to result in two ways. One is via test another one is via sports then two sets of members, member functions of common base class student are inherited to the derived class result at the time of execution the system will get confuse to use what set of member functions of base class.
This can be avoided by making the common base class as virtual base class.

Eg:

```
class student { };
class test : virtual public student { };
class sports : virtual public student { };
class result : public test, sports { };
```

Ref: Object-oriented Programming in Turbo C++: Robert Lafore

previous up next
Learning C/C++ Step-By-Step - Page 13 Learning C/C++ Step-By-Step - Page 15

add comment | view as pdf | print: this | all page(s)

Related Tutorials

- Beginner's Guide To c++
- An Explanation of Pointers (C++)

Please do not use the comment function to ask for help! If you need help, please use our forum.
Comments will be published after administrator approval.