Lesson 9 : Inheritance

Objectives

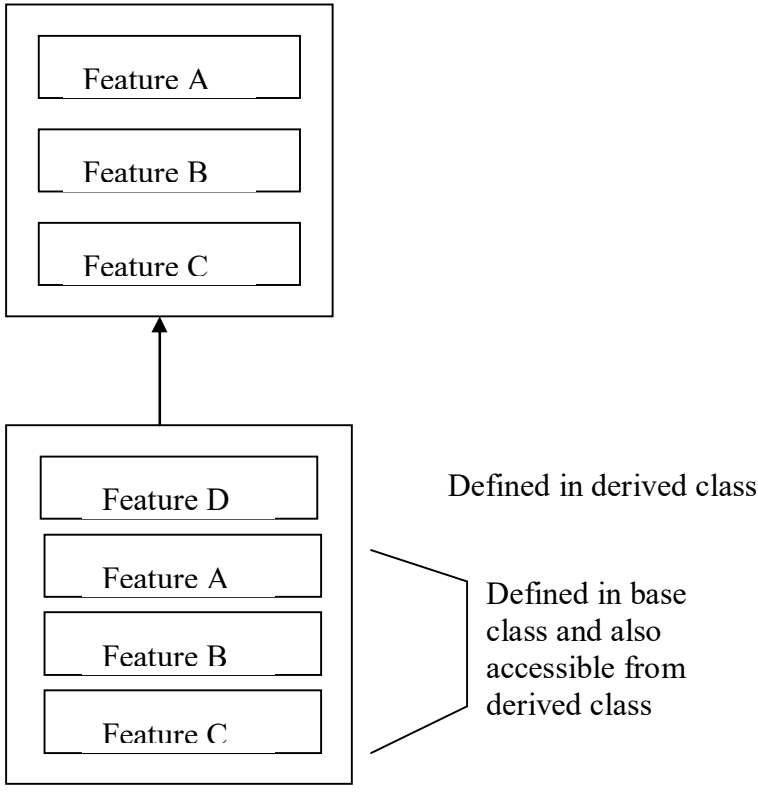
After completing this lesson you will understand about

- The concept of inheritance, and how it supports the concept of reusability.
- The derivation of a new class and the different visibility modes under which the new class is derived.
- The different types of inheritance.
- The execution of constructors in the derived class.
- The abstract class.

Structure Of The Lesson

9.1 Introduction To Inheritance

Inheritance is an important feature of object oriented programming, it is the process of extending a class to define new class. It allows new classes to be built from older and less specialized classes, instead of rewriting from the scratch. It allows the reuse of the pre-tested code of a class without changing the class. In this process the extended class is called base class and the newly defined class is called derived class. The base class is sometimes called a super class and correspondingly derived class is called sub class.



The derived class inherits all the capabilities of the base class and can add refinements and extensions of its own. There is no specific limit on the number of classes that may be derived from one another, which forms a class hierarchy.

C++ supports the access specifiers private, public and protected. As far as access limit is concerned, the private and protected members can be accessed only within the class . Public members are always accessible directly by all users of the class.

```
Base class name
{
private:
:       // visible to member functions within its class but not in
derived class
:
protected
:       //visible to member functions within its class and derived
class
:
public
:       // visible to member functions within its class,
:       //derived class and through object
};
```

9.2 Derived Class Declaration

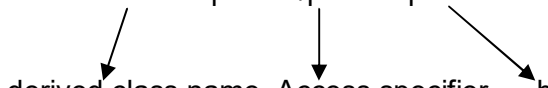
The derived class extends its features by inheriting the properties of another class, called base class and adding features of its own. The declaration of derived class specifies its relationship with the base class in addition to its own features.

The syntax for declaring a derived class is:

```
class derivedclass : [visibility mode Or Access specifier] baseclass
{
//Derived class members  (member functions and variables)
};
```

In the above syntax **class** is a keyword, **derived class** is name of the new class. "**:**" is used as a separator between the derived class name and the access speciifier. The visibility mode tells the way in which the base class is inherited (private, protected or public). The visibility mode is optional and the default mode is private. "baseclass" is the name of the class from which the properties are being inherited.

e.g.: class student : private/public person



derived class name Access specifier base class

{

…… };

Visibility Modes: There are three types of visibility modes. They are:

i) Private
ii) Protected
iii) Public

These are used for specifying the way in which the properties of the base class are inherited.

Private: If the access specifier "private", is used to inherit the properties of base class, then

i) The private data of the base class cannot be inherited but can be accessed through the inherited member.

ii) The protected data of the base class is inherited as the private data. It cannot be used in the main function. But, it can be accessed using the base class or the derived class member function.

iii) The public data of the base class is inherited as private data in the derived class. They are inaccessible to the objects of the derived class. It can be accessed by the member functions of the derived class.

e.g:

```
class base
{
private:
int x;
readx();
protected:
int y;
ready();
public:
int z;
readz();
}

class der: private base
{
private:
```

```
        int w;
public:
        void read();
void display();
};
```

In the above example, variable x cannot be inherited, but both y and z are inherited as private variables in the derived class. They can be accessed through the functions read() and display(). They cannot be used directly by the main().

Protected: When the access specifier “protected” is used to inherit base class properties,

- The private data of the base class cannot be inherited but can be accessed through the inherited members.
- Protected member in the base class are inherited as protected data in the derived class.
- The public data in the base class is inherited as protected data of the derived class.

```
class base
{
private:
int x;
readx();
protected:
int y;
ready();
public:
int z;
readz();
}
class der: protected base
{
private:
        int w;
public:
        void read();
        void display();
};
```

In the above example, ‘x’ cannot be inherited. y and z are inherited as protected members and thus can be used in read() and display() they can be further inherited but they can not be accessed from the ,main

Public: If the access specifier "public" is used to inherit the properties of base class, then

- The private data of the base class cannot be inherited as member of derived class but can be accessed through the inherited member functions.
- The protected member of the base class is the member of the derived class.
- The public member of the base class is the public member in the derived class.

```
class base
{
private:
int x;
readx();
protected:
int y;
ready();
public:
int z;
readz();
}
class der: public base
{
private:
        int w;
public:
        void read();
        void display();
};
```

In the above example 'x' cannot be inherited into the derived class, 'y' is inherited as a protected member of the derived class and 'z' can be accessed from the main itself.

Visibility Of Inherited Members:

Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

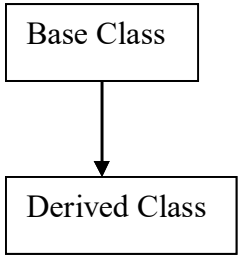
9.3 Types Of Inheritance

There are five types of inheritance

a. Single inheritance
b. Multilevel inheritance
c. Hierarchical inheritance
d. Multiple inheritance
e. Hybrid inheritance

9.3.1 Single Inheritance

Derivation of a class from only one base class is called single inheritance.



Programs To Illustrate Simple Inheritance

```
class A
{
      protected:
            int a_data;
      public:
            A ( ) {a_data =0 ;}
            A (int X)       {a_data =X ;}
            void showA ( )
            {
                  cout<< "\n\t a_data;   }
};

class B: public A
{
     private:
          int    b_data;
     public:
          B ( ) {b_data = 0 ;}
```

```
        B (int X) {b_data =X ;}
        void print _total ( )
        {
            cout<< "\n\t total ="
                << a_data +b_data;
        }
        B (int X, int Y)
        {
           a_data = X;
           b_data = Y;
        }
      };
void main ( )
{
        B    bobj (10, 20);
        bobj. print_total ( );
}
```

Program to display salaried employees data using single inheritance.

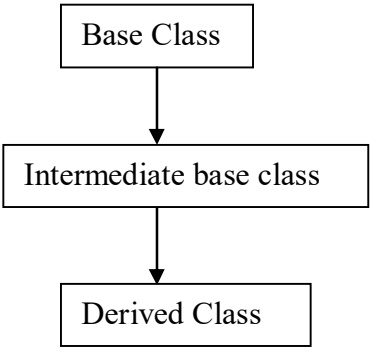
```
# include < iostream. h>
# include < string. h>
  class employee
   {
      public:
       employee (char * name, char * id, char * addr);
       void print ( );
   protected:
       char name [25], id [20], addr [100];
        double net_Sal;
   };
 class salaried_employee: public employee
 {
        public:
            salaried_employee (char * name, char *id,
char * addr, double sal);
            void show ( );
            void give rise ( );
        private:
            double basic_pay;
 };
employee:: employee (char * na, char * id, char * addr)
{
        strcpy (name, na);
```

```
            strcpy (id, idy);
            strcpy (addr, adr);
             net_sal =0;
}
void employee: : print ( )
{
        cout<< " \n\t Name:" <<name
              << "\t    ID:" <<id;
        cout<< "\n\t Address:"  << addr;
}
salaried_employee:: salaried_employee (char *na, char
*idy, char * addr, double sal): employee (na, idy, addr)
{
         basic_pay =sal;
}
          void salaried employee ::give_raise ( )
{
        basic_pay  += 100;
}
void salaried employee : : show ( )
{
        print ( );
        cout<< "\n\t  Salary:" << basic_pay;
}
void main ( )
{
        salaried Employee    e1 ( "Ravi", "ID 1", " unknown",
5000);
        e1. show ( );
        e1.  give_raise ( );
        e1. show ( );
}
```

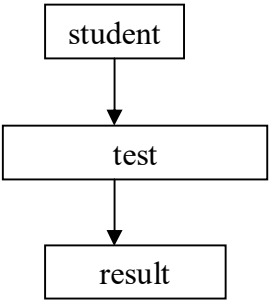
9.3.2 Multilevel Inheritance

When a class is derived from another derived class, such type of inheritance is known as Multilevel Inheritance.

A derived class with multilevel inheritance is as follows:



Program to display the result of the student using multilevel inheritance




```
#include<iostream.h>
class student
{
protected:
int rno;
public:
void getnum(int x)
{rno = x;}
void putnumb()
{ cout<<"Roll number:"<<rno<<endl;
}
};
class test: public student
{
protected:
float m1,m2;
public:
```

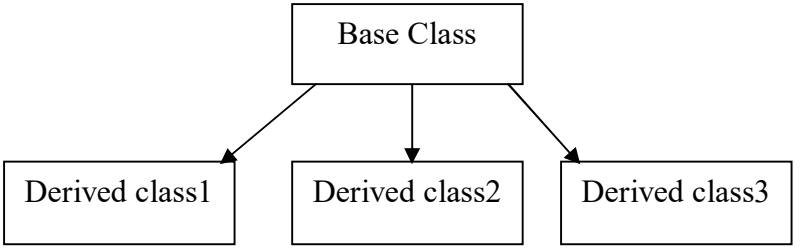
```
void getmarks(float x,float y)
{
m1 = x;
m2 = y;
}
void putmarks()
{
cout<<"Marks in sub1="<<m1<<endl;
cout<<"Marks in sub2="<<m2<<endl;
}
};

class result: public test
{
float total;
public:
void display()
{
total = m1 + m2;
putnumb();
putmarks();
cout<<"Total ="<<total<<endl;
}
};
void main()
{
result st1;
cout<<"STUDENT INFORMATION"<<endl<<endl;
st1.getnum(111);
st1.getmarks(78.0,89.5);
st1.display();
}
```

output:

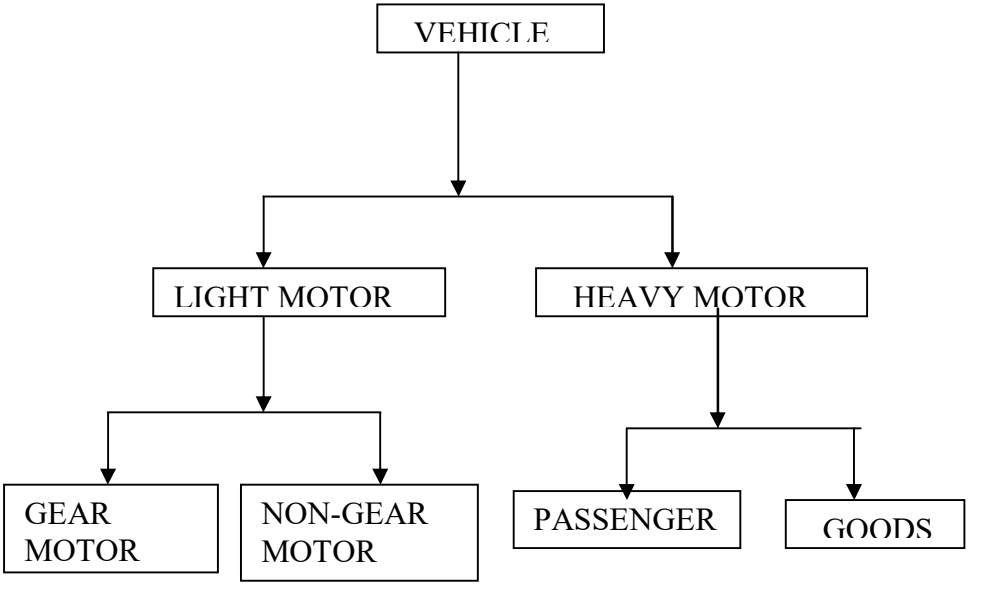
```
STUDENT INFORMATION
Roll number:111
Marks in sub1=78
Marks in sub2=89.5
Total =167.5
```

9.3.3 Hierarchical Inheritance

Derivation of several classes from a single base class i.e., the traits of one class may be inherited by more than one class, is called hierarchical inheritance.



Here is an example for hierarchical inheritance:




```
# include<iostream.h>
# define MAX_LEN  25
class vehicle
{
        protected:
```

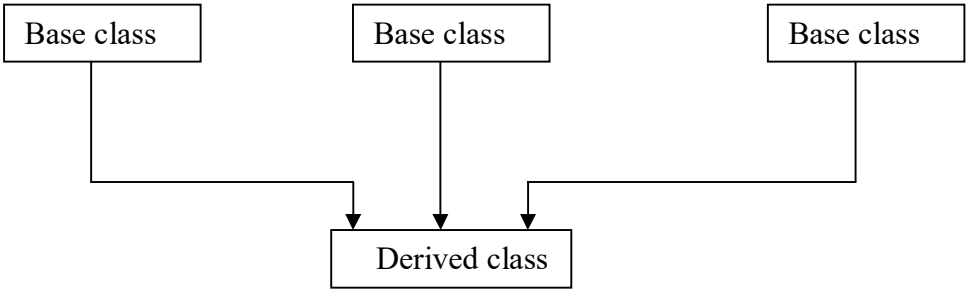
```
                    char  name[MAX_LEN];
                     unsigned wheelscount;
          public:
               void getdata( )
                {
                      cout<< " \n\t Name?";
                      cin>>name;
                      cout<< "\t Wheels?";
                      cin >> wheelscount;
}
void displaydata ( )
{
                    cout<< " \n\t Name:"
                          << name
                          << " \t Wheels :"
                           << wheelscount;
 }
};
class lightmotor : public vehicle
          {
             protected:
                int speedlimit;
             public:
                void getdata( )
                {
                          vehicle :: getdata();
                           cout<< "\n\t speed?";
                            cin>>speedlimit ;
                 }
                void displaydata()
                 {
                            vehicle :: displaydata( );
                             cout << "\n\t speed limit
:"<<speedlimit;
                  }
          };
class heavymotor : public vehicle
{
          protected:
                  int loadcapacity;
                   char permit[MAX_LEN];
           public:
                   void getdata ( )
                   {
                           vehicle :: getdata( );
```

```
                cout<< " \t load?";
                cin >>loadcapacity;
                cout<< " \t Permit?";
                cin >>permit;
            }
            void displaydata( )
            {
                vehicle :: displaydata( );
                cout << "\n\t Load :"
                    << loadcapacity
                    << " \n\t Permit:"
                    << permit;
            }
};
class gearmotor: public lightmotor
{
    protected:
        int gearcount;
    public:
        void getdata( )
        {
            lightmotor :: getdata( );
            cout<< " \t No of gears ?";
            cin>> gearcount;
        }
        void displaydata( )
        {
            lightmotor :: displaydata( );
            cout << " \t gear :" << gearcount;
        }
};
class nongearmotor : public lightmotor
{ };
class passenger : public heavymotor
{
    protected:
        int sitting, standing;
    public :
        void getdata( )
        {
            heavymotor ::getdata( );
            cout<< "\n\t Seats?";
            cin >> sitting;
            cout << "\t  standing  ?";
            cin >> standing;
```

```
                }
                void displaydata( )
                {
                        heavymotor :: displaydata( );
                        cout<< "\n\t seats :" << sitting;
                        cout<< " \t standing :" << standing;
                 }
};
class goods :public heavymotor
{ };
void main ( )
{
     gearmotor  gm;
     nongearmotor  ngm;
     passenger p;
     goods g;
     cout<< "\t Input for Gearmotor :";
     gm.getdata( );
          cout<<"\n Gearmotar\n ";


       Seats?45
        standing  ?20

 Passenger carrier

       Name:bus       Wheels :6
       Load :600
       Permit:ap
       seats :45      standing :20
       input for goods carier:
       Name?lorry
       Wheels?6
       load?2000
       Permit?ap

 Goods Carrier

       Name:lorry     Wheels :6
       Load :2000
       Permit:ap
```

9.3.4 Multiple Inheritance

Derivation of a class from two or more base classes, this type of inheritance is called multiple inheritance. Multiple inheritance is shown in the following diagram.



Example: Let a class 'C' be derived from two base classes 'A' and 'B' then class C is defined with the following syntax

```
class c : public A, public B
{
  _ _ _ _
  _ _ _ _
  Body of class c
  _ _ _ _ _
  _ _ _ _ _ _
};
```

A program to demonstrate multiple inheritance

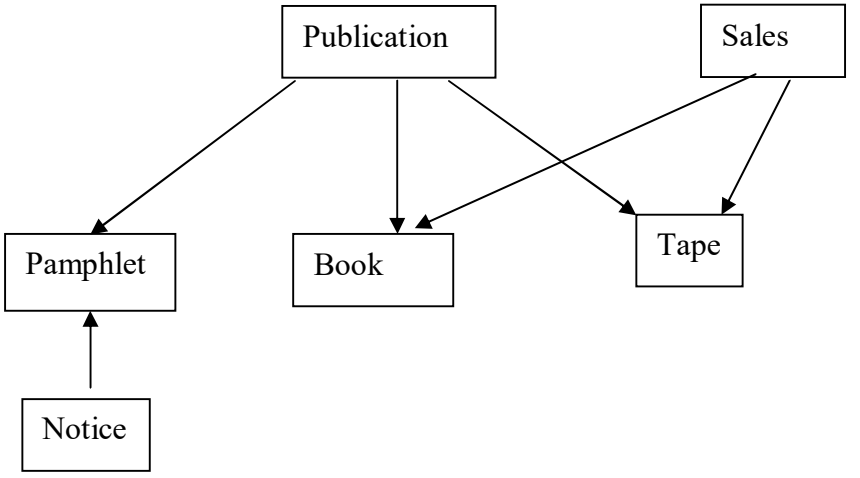
```
#include<iostream.h>
class A
{
      protected:
            int  a;
       public:
             A( ) { a = 0;}
             A( int d) {a= d;}
};
```

```
class B
{
        protected :
            int b;
         public:
            B ( ) {b=0;}
            B( int x) {b = x;}
};
class C: public A,public B
{
       private:
              int  c;
       public:
              C( ) {c = 0;}
              C(int i, int j, int k): A(i), B(j)
               {
                    c =k;
               }
               void display( )
               {
                        cout<< "\n\t sum =" <<(a+b+c);
                }
};
void main ( )
{
            C    cobj (10,20,30);
                 cobj.display ( );
}
```

output:

```
sum = 60
```

Program related to products company modeling using multiple inheritance




```
#include<iostream.h>
class publication
{
        protected :
             char title [50];
             float price;
           public:
              void getdata( )
              {
                    cout<< "\n\t Title ?";
                                  cin>> title;
                     cout<< " \t price ?";
                     cin >> price;
              }
              void putdata( )
              {
                   cout<< " \n\t Title:"   << title;
                   cout<< " \t price:"  <<price;
              }
};
class sales
{
          protected :
              float publishsales[3];
```

```
        public:
            void getdata( )
            {
                int i;
                for (i=0; i<3; i++)
                {
                    cout<< "Sales of the month" <<(i+1)
                        << " :";
                    cin>> publishsales[i];

                }
                                                    }
             void putdata( )
             {
                  int i; float tot = 0;
                  for(i =0; i<3; i++)
                  {
                       cout<< " \n sales of the month"<<(i+1)
                            << ":"<< publishsales[i];
                        tot  += publishsales[i];
                   }
                   cout<< " \n\t Total sales :"<<tot;
             }
};
class  book : public publication, public sales
{
      private:
            int   pages;
       public:
            void getdata( )
            {
                 publication :: getdata( );
                  sales :: getdata( );
                 cout<< "\n\t #of pages ?";
                  cin >> pages;
             }
             void putdata( )
             {
                publication :: putdata( );
                 sales ::putdata( );
                 cout<< "\n\t No of pages :"<<pages;
              }
};
class tape: public publication, public sales
{
```

```
        private:
              float playtime;
        public:
            void getdata( )
             {
                 publication ::getdata( );
                   sales ::getdata( );
                   cout<< " \t play time ?";
                    cin >> playtime;
              }
               void putdata( )
              {
                       publication :: putdata( );
                       sales :: putdata( );
                        cout<< " \n\t play time:"
                              << playtime;
              }
};
class pamphlet : public publication
{ };
 class notice : public pamphlet
{
        private:
             char  whom[20];
        public :
              void getdata( )
              {
                   pamphlet ::getdata( );
                   cout<< " \t To whom ?";
                    cin>> whom ;
              }
              void putdata( )
              {
                        pamphlet ::putdata();
                   cout<< " \t To whom :?"<< whom ;
                              }
 };
void main()
          {
                   book      book1;
                   tape       tape1;
                   pamphlet    p1;

                   cout<< "\n\t Enter Book data :";
                   book1.getdata( );
```

```
    cout<< " \n\t Enter Tape Data :";
    tape1.getdata( );
          cout<< "\n\t Enter notice data :";
    notice1.getdata( );
    cout<< "\n\t \t OUTPUT \n";
    cout<< "\n Book:" ; book1.putdata( );
    cout<< "\n Tape :"; tape1.putdata( );
    cout << "\n Notice:";notice1.putdata( );
      } //end   of main ( )
```

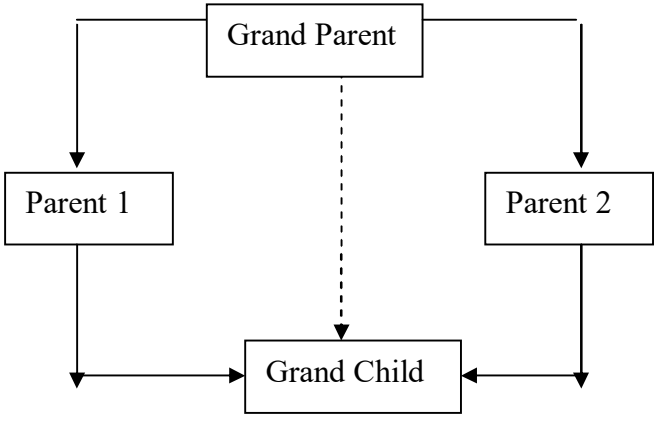
output:

```
          Enter Book data :
          Title ?c++
          price ?300
    Sales of the month1 :45
    Sales of the month2 :56
    Sales of the month3 :67


          #of pages ?450

          Enter Tape Data :
          Title ?Bhajans
          price ?50
    Sales of the month1 :60
    Sales of the month2 :70
    Sales of the month3 :80
          play time ?60

          Enter notice data :
          Title ?Games
          price ?10
          To whom ?students
          play time ?60


     OUTPUT

    Book:
          Title:c++       price:300
    sales of the month1:45
    sales of the month2:56
    sales of the month3:67
          Total sales :168
          No of pages :450
```

Tape :
Title:Bhajans price:50
sales of the month1:60
sales of the month2:70
sales of the month3:80
Total sales :210
play time:60
Notice: Title:Games price:10 To whom :?students

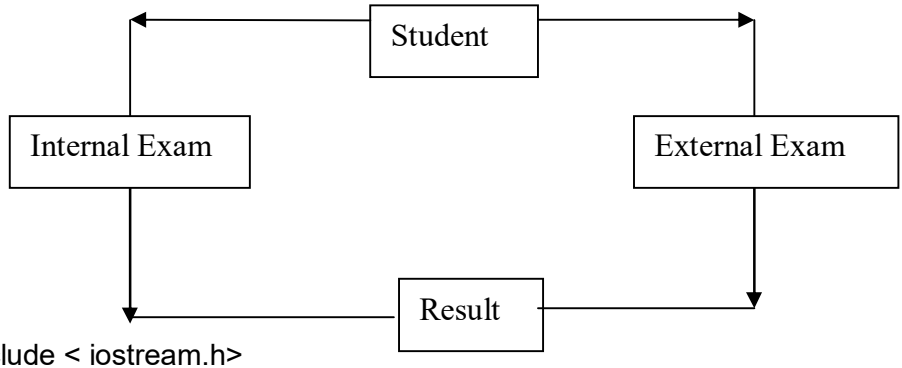
9.3.5 Multipath Inheritance

The form of inheritance, deriving a new class by multiple inheritance of base classes, which are derived earlier from the same base class, is known as multipath inheritance. This type of inheritance involves multiple forms of inheritance namely multilevel, multiple and hierarchical inheritance. The child class is derived from the same base class parent1 and parent2(multiple inheritance), which themselves have a common base class grandparent(hierarchical inheritance). The child inherits the properties of grandparent via two separate paths. The classes, parent1 and parent2 are referred as direct base classes, whereas grandparent is referred to as indirect base class. The inheritance diagram for multi-path inheritance is as shown below



Multipath inheritance may cause duplicate items to be derived into the child class, twice via parent1 and parent2, of the grandparent class. C++ avoids this ambiguity by introducing virtual base classes. This is done by making the common base class as a virtual base class, while declaring the direct or intermediate classes (parent1 & parent2).

The following example of student database uses multipath inheritance to derive the classes in the chart:




```
# include < iostream.h>
const int MAX_LEN =25;
class student
{
        protected:
              int     rollno;
              char  name [MAX_LEN];
        public:
               void read ( )
               {
                     cout<< "\n\t Roll_No ?";
                     cin >> rollno;
                     cout<< " \t name ?";
                     cin >> name;
               }
               void print ( )
               {
                      cout<< " \n\t Name:"
                           <<name << "\t Roll_No:"
                            <<rollno;
               }
};
class internalexam: public virtual student
{
             protected:
                   int marks1, marks2;
             public:
                    void readdata( )
                    {
                          cout<< "\n\t Input marks in2 subjects:";
                          cin>> marks1 >>marks2;
                     }
```

```
                    void displaydata( )
                    {
                        cout<< "\n\t Internal marks:"
                            <<marks1<< " "
                             <<marks2;
                        cout<< "\n\t\t Total:"
                             << gettotalinternal( );
                    }
                    int gettotalinternal( )
                    {
                            return (marks1+marks2);
                     }
};
class externalexam :public  virtual student
{
        protected:
            int marks1, marks2;
        public:
             void readdata( )
             {
                 cout<< "\n\t Input marks in 2 subjects:";
                 cin>> marks1 >>marks2;
             }
             void displaydata( )
             {
                 cout<< "\n\t External marks:"
                      << marks1<< " "  << marks2;
                 cout<< "\n\t\t Total:"
                       << gettotalexternal ( );
             }
 int gettotalexternal( )
  {
        return(marks1+marks2);
   }
};
class result:public internalexam, public externalexam
{
        public:
            int gettotalmarks( )
            {
                return (gettotalinternal( )+gettotalexternal( ) );
             }
};

void main ( )
```

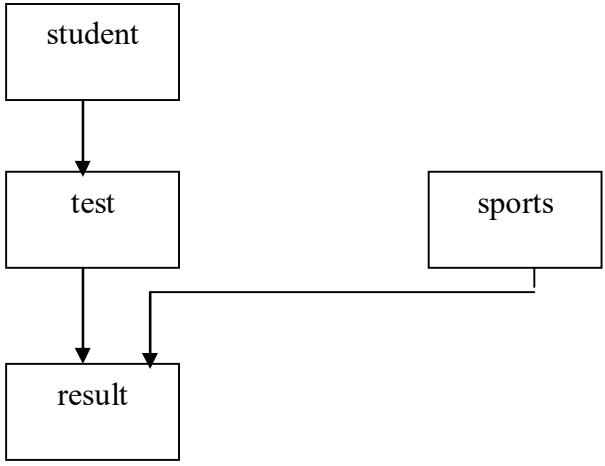
```
{
    result stud1;
    cout<< "\n\t Input student info:";
    stud1.read( );
    cout<< "\n\t Reading internal marks:";
    stud1.internalexam::readdata( );
    cout<< "\n\t Reading external marks:";
    stud1.externalexam ::readdata( );
    cout<<"\n\t\t OUTPUT:";
    stud1.print( );
    stud1.internalexam::displaydata ( );
    stud1.externalexam:: displaydata( );
    cout<<"\n\tTotal marks:"<< stud1.gettotalmarks();
}
```

output:

```
    Name:qwer       Roll_No:1
    Internal marks:23 24
          Total:47
    External marks:24 35
          Total:59
    Total marks:106
    Input student info:
    Roll_No ?1
    name ?sai

    Reading internal marks:
    Input marks in2 subjects:15  18

    Reading external marks:
    Input marks in 2 subjects:75  71

          OUTPUT:
    Name:sai       Roll_No:1
    Internal marks:15 18
          Total:33
    External marks:75 71
          Total:146
    Total marks:179
```

9.3.6 Hybrid Inheritance

Derivation of a class involving more than one form of inheritance is known as hybrid inheritance.



Program to calculate the result of the student using hybrid inheritance

```
#include<iostream.h>
class student
{
protected:
int rno;
public:
void getnum(int x)
{rno = x;}
void putnumb()
{ cout<<"Roll number:"<<rno<<endl;
}
};
class test: public student
{
protected:
float m1,m2;
public:
void getmarks(float x,float y)
{
m1 = x;
```

```
m2 = y;
}
void putmarks()
{
cout<<"Marks in sub1="<<m1<<endl;
cout<<"Marks in sub2="<<m2<<endl;
}
};
class sports
{
protected:
float score;
public:
void getscore(float s)
{score = s;}
void putscore()
{
cout<<"Sports score:"<<score<<endl;
}
};
class result: public test, public sports
{
float total;
public:
void display()
{
total = m1 + m2 + score;
putnumb();
putmarks();
putscore();
cout<<"Total ="<<total<<endl;
}
};
void main()
{
result st1;
cout<<"STUDENT INFORMATION"<<endl<<endl;
st1.getnum(111);
st1.getmarks(78.0,89.5);
st1.getscore(6.0);
st1.display();
}
```

output:

```
STUDENT INFORMATION
Roll number:111
Marks in sub1=78
Marks in sub2=89.5
Sports score:6
Total =173.5
```

9.4 Constructors In Derived Class

- As long as the base class has no constructor this derived class need not have constructor.
- If base class has the constructor with one or more arguments then it is compulsory for the derived class to have constructor and it must pass the arguments to the base class. The base class constructor is executed first then the derived.
- In multiple inheritance the base classes are constructed in the order in which they appear in the declaration of the derived class.
- In the multilevel inheritance the constructor will be executed in the order of inheritance.

Execution Of Base Class Constructor :

Method of Inheritance	Order of execution

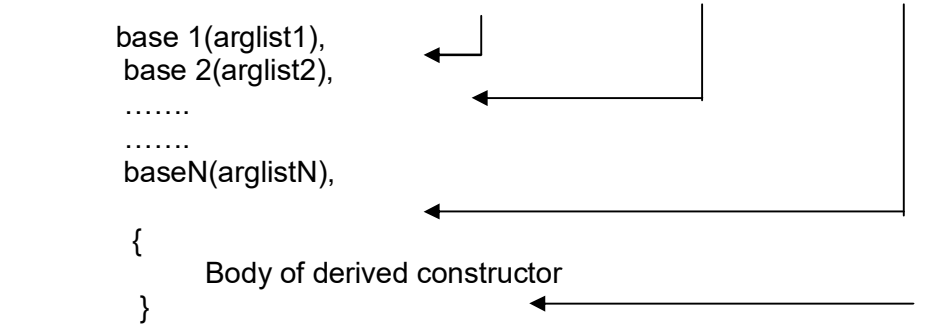
class B: public A
{
 	
};
	A( ): base constructor
	B( ): derived constructor

class A: public B , public C
{
};
	B( ) :base(first)
	C( ) :base(second)
	A( ) :derived

class A: public B, public virtual C
{
 	
};
	C( ) :virtual base
	B( ) :ordinary base
	A( ) :derived

The General Form Of Defining A Derived Constructor Is:

Derived-constructor (Arglist 1, Arglilst 2, …. ArglistN , Arglist (D)



A sample program using derived class constructors

```
#include<iostream.h>
class alpha
{
        int x;
         public :
            alpha ( int i )
             {
               x=i;
               cout<<"Alpha is initialized\n ";
             }
            void show_x( )
             {
                cout<<"x="<<x<<endl;
              }
};
class beta
{
       float y;
        public :
           beta ( float j)
                              {
                                        y = j;
                                        cout<<"Beta initialized\n";
                              }
```

```
                void show_y(void)
                        {
                                cout<<"y="<< y<<"\n";


                        }
};
class gamma:public beta,public alpha
{
        int m,n;
        public:
                gamma(int a, float b,int c, int d): alpha(a),beta(b)
        {
                m = c;
                n = d;
                cout<<"Gamma initialized\n";
        }
                void show_mn(void)
        {
                cout<<"m= "<<m<<"\nn="<<n<<"\n";
        }
};
void main()
{
        gamma g(5,10.75,20,30);
        cout<<"\n";
        g.show_x();
        g.show_y();
        g.show_mn();
}
```

output:

```
        Beta initialized
        Alpha is initialized
        Gamma initialized

        x=5
        y=10.75
        m= 20
         n=30
```

9.5 Abstract Class

An abstract class is a class which is not used to create objects. It acts as only a base class for the other derived class. A class that contains at least one pure function is known as abstract class.

9.6 Summary

- We have covered the concept of inheritance. The derivation of a new class from the base class.
- The different visibility modes and the visibility of the base class inheritance in different modes are studied.
- The different types of inheritance are covered in detail.
- The usage of constructors in the derived class is also covered.

9.7 Technical Terms

Abstract class: A class that serves as only base class from which classes are derived. No object of abstract base class are created.

Base class: A class from which other classes are derived.

Derived class: A class that inherits some or all of its functions from another class, called the base class.

Intermediate class: A class that lies on an inheritance path connecting a base class and a derived class.

Protected member: A protected member is same as a private member except that protected members of the base class can be inherited.

Reusability: A feature of OOP where it allows the reuse of existing class without redifinition.

Virtual base class: A base class that can be qualified as virtual in the inheritance definition. For a virtual base class, only one copy of its members can be inherited regardless of the number of inheritance paths between the base class and the derived class.

Visibility : The ability of one object to be server to others.

9.8 Model Questions

1. What is inheritance? Explain with example.
2. What is the difference between base and derived classes?
3. What are the different forms of inheritance supported by C++? Explain with examples.
4. Explain the importance of Inheritance.

9.9 References

Object-oriented programming with C++,
by E. Bala Guruswamy.

Problem solving with C++
by Walter Savitch

Mastering C++
by K.R.Venugopal, RajkumarBuyya, T.RaviShankar

AUTHOR:

M. NIRUPAMA BHAT, MCA., M.Phil.,
Lecturer,
Dept. Of Computer Science,
JKC College,
GUNTUR.