Object Oriented Programming Using Cpp

Topic:

Limitations Of Multipath Inheritance

Limitations of multipath Inheritance

The process of multipath inheritance suffers from some disadvantages as mentioned below:

1. As we can see that class E has been derived from class B, C and D which are derived from class A. Due to this class E will get multiple copies (3 copies in our example) of common base class A members i.e. same set of members are appearing for multiple times. It is known as **data redundancy or data duplicacy**.

2. Now, if we try to access the members of class A, the system would produce error message **"Member is ambiguous".** This is because class A members can be accessed by multiple paths available i.e. using E ----> B ----> A

using E ----> C ----> A

using E ----> D ----> A

This is known as ambiguity in multipath inheritance. This is explained in following program.

Program 51: Write a program in C++ to illustrate the concept of multipath inheritance.

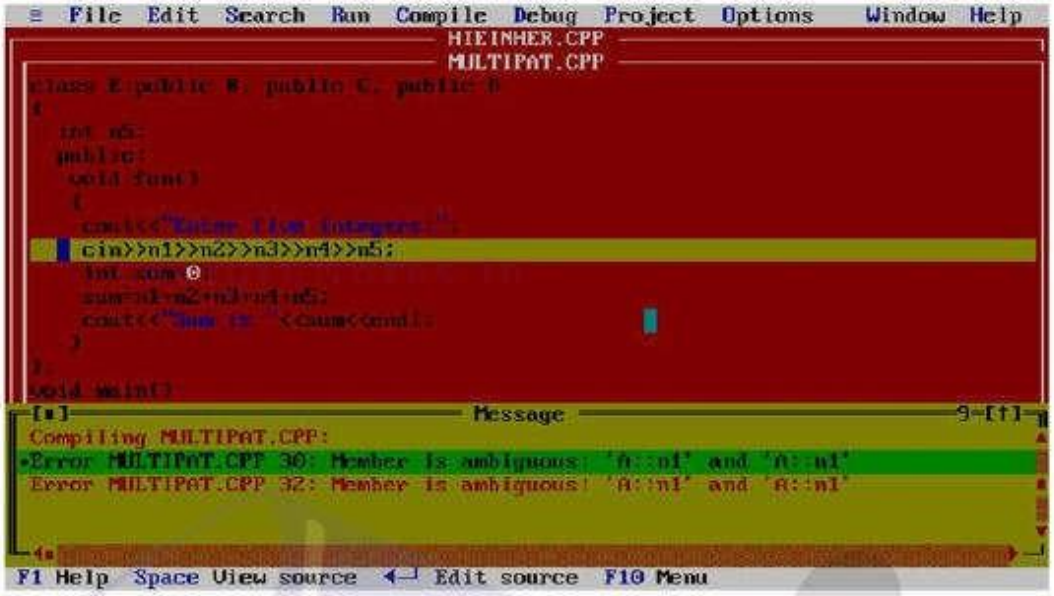
Code:

```
class A
{
    protected:
        int n1;
};
class B : public A
{
    protected:
        int n2;
};
class C : public A
{
    protected:
```

```
        int     n3;
};

class  D : public     A
{
    protected:
        int     n4;
};
class  E : public     B, public      C, public      D
{
        int     n5;
    public:
        void    fun()
        {
            cout<<"Enter five integers";
            cin>>n1>>n2>>n3>>n4>>n5;
            int         sum=0;
            sum=n1+n2+n3+n4+n5;
            cout<<"Sum is "<<sum<<endl;
        }
};
void    main()
{
    E       obj;
    obj.fun();
}
```

O/P:



The above snapshot shows the ambiguity occurred during execution of the program.

Introduction to virtual base class

The disadvantages of multipath inheritance can be avoided using a new concept known as virtual base class. The keyword "virtual" is added with the common base class at all the locations where this class is used for deriving other classes. When system sees the virtual base class, the compiler takes necessary precaution to avoid duplication of members. Hence, using virtual base class system receives only one copy of common base class members and discards the other copies. Here, the only important thing to note is that, only common base class can be made virtual.

Program 52: Write a program in C++ to illustrate the concept of multipath inheritance. Use virtual base class.

Code:

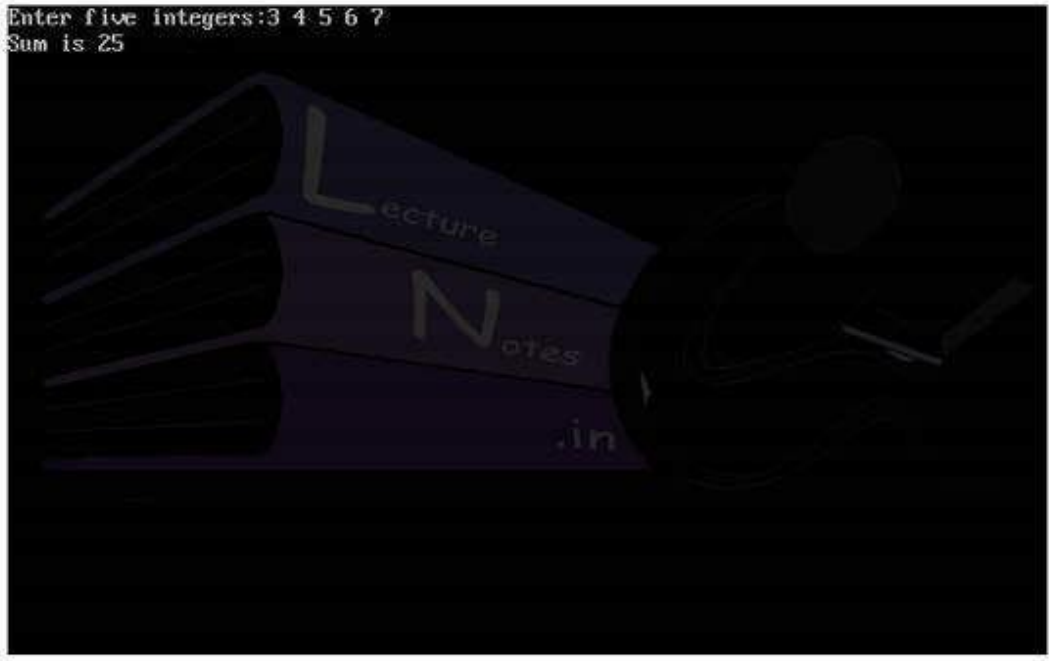
```
class  A
{
        protected:
                int     n1;
};
```

```
class  B : virtual     public    A
{
        protected:
                int     n2;
};
class  C : virtual     public    A
{
        protected:
                int     n3;
};

class  D : virtual     public    A
{
        protected:
                int     n4;
};
class  E : public      B, public      C, public      D
{
                int     n5;
        public:
                void    fun()
                {
                        cout<<"Enter five integers";
                        cin>>n1>>n2>>n3>>n4>>n5;
                        int         sum=0;
                        sum=n1+n2+n3+n4+n5;
                        cout<<"Sum is "<<sum<<endl;
```

```
        }
};
void main()
{
    E obj;
    obj.fun();
}
```

O/P:



Here, we can see that using virtual base class, the program does not show any error message and it gives the desired output.

8.4 Constructor, Destructor and Inheritance

We know that constructors are used to initialize the data members of the object whereas destructors are used to destroy the object once its requirement is complete. In case of inheritance, though constructors and destructors are not inherited, they are executed. Now, constructors can be used in following two ways:

1. Default constructor: If base class has a default constructor then it is not compulsory to write a constructor in the derived class. However, since object declaration is always done for derived class in inheritance, it is a good practice to declare a derived class constructor. For execution, derived class object is declared. First base class constructor is executed and then derived class constructor is executed. The destructor executes in reverse order of constructor execution.

Program 53: Write a program in C++ to explain the working of default constructor in inheritance.

Code:

```
class  BASE
{
        public:
                BASE()
                {
                        cout<<"Base class constructor"<<endl;
                }
                ~BASE()
                {
                        cout<<"Base class destructor"<<endl;
                }
};
class  DERIVED : public      BASE
{
        public:
                DERIVED()
                {
                        cout<<"Derived class constructor"<<endl;
                }
```

```
        ~DERIVED()
        {
                cout<<"Derived class destructor"<<endl;
        }
};
void    main()
{
        DERIVED     obj;
}
```

O/P:

Base class constructor

Derived class constructor

Derived class destructor

Base class destructor

2. Parameterized constructor: In this method, it is compulsory to define a parameterized constructor in derived class as this constructor will pass parameters to the base class parameterized constructor for execution.

Program 54: Write a program in C++ to illustrate the use of parameterized constructor in inheritance.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE
{
        protected:
                int n1,n2;
        public:
```

```
        ONE(intx,int y)
        {
                n1=x;
                n2=y;
                cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl;
        }
};
class  TWO
{
    protected:
        int n3,n4;
    public:
        TWO(intp,int q)
        {
                n3=p;
                n4=q;
                cout<<"n3="<<n3<<endl<<"n4="<<n4<<endl;
        }
};
class THREE:public ONE, public TWO
{
        int n5,n6;
    public:
        THREE(inta,intb,intc,intd,inte,int f):ONE(c,d),TWO(e,f)
        {
                n5=a;
                n6=b;
```

```
        cout<<"n5="<<n5<<endl<<"n6="<<n6<<endl;
    }
    void sum()
    {
        int sum=0;
        sum=n1+n2+n3+n4+n5+n6;
        cout<<"Sum is "<<sum<<endl;
    }
};
void main()
{
    clrscr();
    THREE obj(5,10,15,20,25,30);
    obj.sum();
    getch();
}
```

O/P:

n1=15

n2=20

n3=25

n4=30

n5=5

n6=10

Sum is 105

8.4.2 Constructor and Destructor with virtual class

When virtual base class is involved in inheritance then the order of constructor and destructor execution is as below:

- Virtual base class constructor is executed first.
- Non virtual base class constructor (as per inheritance) is executed after that.
- Derived class constructor is executed in last
- Destructor execution order is just reverse of constructor execution.

Program 55: Write a program to illustrate the execution of constructor and destructor in case of virtual base class.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE
{
        public:
                ONE()
                {
                        cout<<"Class ONE constructor"<<endl;
                }
};
class  TWO
{
        public:
                TWO()
                {
                        cout<<"Class TWO constructor"<<endl;
                }
};
```

```
class  THREE : public        ONE, virtual       public         TWO
{
        public:
                ONE()
                {
                        cout<<"Class ONE constructor"<<endl;
                }
};
```

O/P:

Class TWO constructor

Class ONE constructor

Class THREE constructor

8.5 Object Delegation

The process of declaring an object of a class as member of another class is known as object delegation. In object delegation, class which contains an object of another class as its member, is known as container class. This kind of relationship is known as containership.

Program 56: Write a program to explain the concept of object delegation.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE
{
        public
                int     a;
```

```
        ONE()
        {
            a=10;
        }
};
class TWO
{
    public:
        int     k;
        ONE  ob;
        TWO()
        {
            k=20;
        }
        void   show()
        {
            cout<<"a="<<ob.a<<"   "<<"x="<<x<<endl;
        }
};
void   main()
{
    clrscr();
    TWO  t2;
    t2.show();
}
```

O/P:

a=10 k=20

8.6 Abstract Class

A class for which object can't be created is known as abstract class. Hence, abstract class can act as base class. It just gives a structure based on which other classes are derived.

8.7 Advantages and Disadvantages of Inheritance

8.7.1 Advantages

- The most important advantage of inheritance is code reusability. In inheritance from a base class, lots of new classes can be derived which provides the reusability of code. It reduces software development effort and time.
- Using inheritance, object of a class becomes more dominant as it is able to access the members of other class also.
- A single base class can be used to derive a number of new classes.

8.7.2 Disadvantages

- Unorganized use of inheritance may cause increase in complexity of program.
- In class hierarchy, various data elements remains unused which causes memory wastage.