"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

Lab Report-05

Submitted By:

Name: Khandoker Sefayet Alam
Roll:2003121
Department: Computer Science & Engineering
Section-C
Session:2020-21
Course code: CSE 1204

Submitted to:

SUHRID SHAKHAR GHOSH
Assistant Professor
Department of Computer Science & Engineering, RUET

[MODULE-03]

Task-01: You have the create an inheritance among `Father`-->Son -->GrandSon class. The `father` class has the following data members

```
class Father{
private:
int money;
protected:
int gold;
public:
int land;
};
```

Now write the Son and GrandSon classes with `private/protected/public` access modifier
and do the following:
i) Try to access `money`, `gold` and `land` from Son class
ii) Try to access `money`, `gold` and `land` from GrandSon class
iii) Find the values of money, gold and land when different access modifer is used in the
following table

Class		In Son Class			In GrandSon Class		
Son	Grandson	money	gold	land	money	gold	land
Public	public	?	?	?	?	?	?
protected	public	?	?	?	?	?	?
private	public	?	?	?	?	?	?
Public	protected	?	?	?	?	?	?
protected	protected	?	?	?	?	?	?
private	protected	?	?	?	?	?	?
Public	private	?	?	?	?	?	?
protected	private	?	?	?	?	?	?
private	private	?	?	?	?	?	?

Solution:

Code:

```
#include<iostream>

using namespace std;

class Father{
    private:
        int money;
    protected:
        int gold;
    public:
        int land;
    Father(){
    money=500;
    gold=1000;
    land=100;
    }
    void setgold(int x){
    gold=x;
    }
     void setmoney(int x){
    money=x;
    }
```

```
     void setland(int x){
    land=x;
    }

    //getters
    void getmoney(){
    cout<<"money= "<<money<<endl;
    }
    void getgold(){
    cout<<"gold= "<<gold<<endl;
    }
    void getland(){
   cout<<"land= "<<land<<endl;
    }

};
class Son:public Father{
public:
    Son(){
    //cout<<"money= "<<money<<endl;
    cout<<"gold= "<<gold<<endl;
    cout<<"land= "<<land<<endl;
    }

};
```

```
class grandson:private Son{
public:
    grandson(){
//    cout<<"money= "<<money<<endl;
    cout<<"gold= "<<gold<<endl;
   cout<<"land= "<<land<<endl;
    }
};

int main(){
    Father obj;
    obj.setgold(500);
    obj.setland(1000);
    obj.setmoney(10000);

    Son obj2;

    cout<<endl;
    grandson obj3;
}
```

Class		In Son Class			In GrandSon Class		
Son	**Grandson**	**money**	**gold**	**land**	**money**	**gold**	**land**
Public	**public**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
protected	**public**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
private	**public**	**NO**	**YES**	**YES**	**NO**	**NO**	**NO**
Public	**protected**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
protected	**protected**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
private	**protected**	**NO**	**YES**	**YES**	**NO**	**NO**	**NO**
Public	**private**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
protected	**private**	**NO**	**YES**	**YES**	**NO**	**YES**	**YES**
private	**private**	**NO**	**YES**	**YES**	**NO**	**NO**	**NO**

<u>TASK:</u>

Topic 2 [**Types of Inheritance**]: Learn and Test different types of inheritance in C++. In each inheritance draw the class diagram with class chain and try to access the data members of bases classes from child classes.

Solution:

i) **Single inheritance**

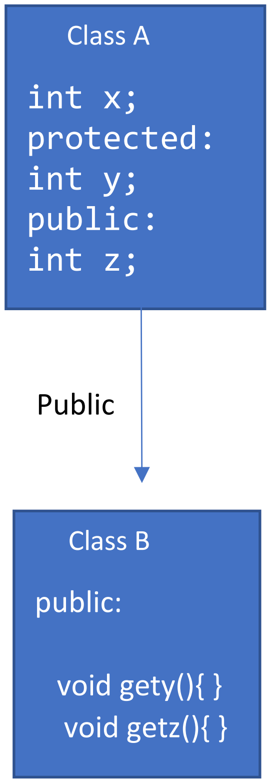
```
class A{
private:
int x=100;
protected:
int y=16;
public:
int z=4;
}
```

```
class B:public A{
public:
      /*void getx()
   /{
     cout<<"x= "<<x<<endl;
}*/
//x is private so cant  call it
   void gety(){
    cout<<"y= "<<y<<endl;
    }
    void getz(){
    cout<<"z= "<<z<<endl;
    }
```

```
int main(){
    B b;
    //b.getx();
    b.gety();
    b.getz();
    return 0;
}
```

	};	

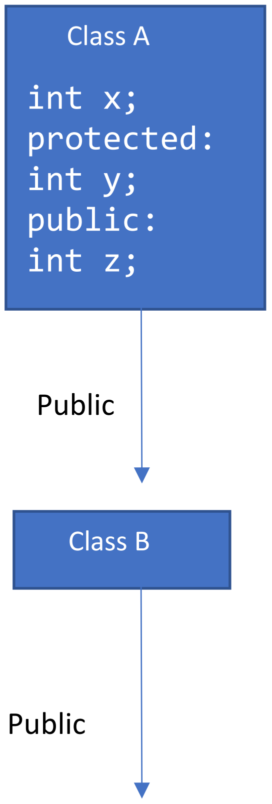
Class chain:

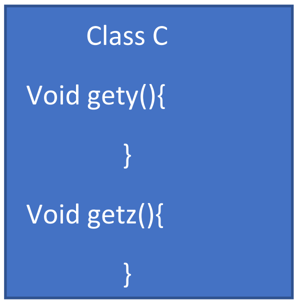

Class A
int x;
protected:
int y;
public:
int z;
Public
Class B
public:
void gety(){ }
void getz(){ }

(ii)Multi-level inheritance:

class A{
private:
int x;
protected:
int y;
public:
int z;
}
	class B:public A{
	};
		class C:public B{
		public:
		/*void getx()
		/{
		cout<<"x= "<<x<<endl;
		}*/
		void gety(){
		cout<<"y= "<<y<<endl;
		}
		void getz(){
		cout<<"z= "<<z<<endl;
		}
		};
			int main(){
			C c;
			c.gety();
			c.getz();
			return 0;
			}

Class chain:



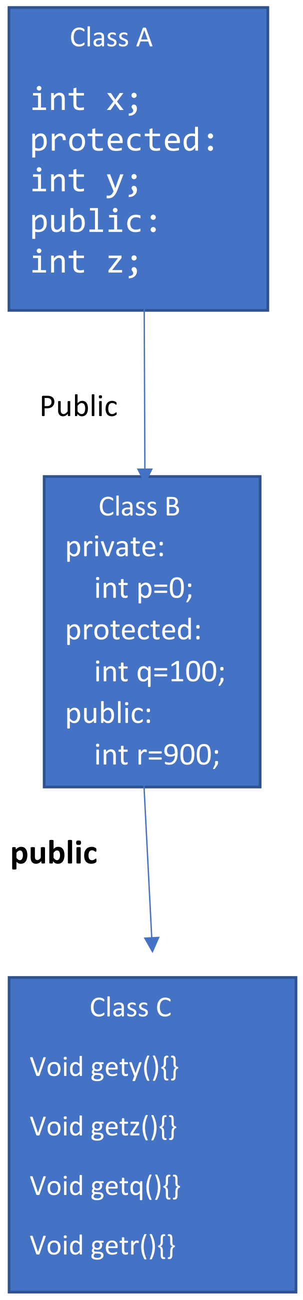


iii) Multiple inheritance

class A
{
private:
    int x=0;
protected:
    int y=100;
public:
    int z=5;
};
	class B
	{
	private:
	   int p=0;
	protected:
	   int q=100;
	public:
	   int r=900;
	};
		class C:public A,public B
		{
		//write public method
		//to access
		//x,y,z,p,q & r
		public:
		   /*void getx()
		   /{
		     cout<<"x= "<<x<<endl;
		}*/
		     void gety(){
			int main()
			{
			   C c;
			//call
			//methods of
			//class C
			   c.getq();
			   c.getr();
			   c.gety();
			   c.getz();
			   return 0;
			}

		`cout<<"y= "<<y<<endl; } void getz(){ cout<<"z= "<<z<<endl; } // void getp(){ //cout<<"p= "<<p<<endl; //} void getq(){ cout<<"q= "<<q<<endl; } void getr(){ cout<<"r= "<<r<<endl; } };`	

Class chain:


Class A
int x;
protected:
int y;
public:
int z;
Public
Class B
private:
int p=0;
protected:
int q=100;
public:
int r=900;
public
Class C
Void gety(){}
Void getz(){}
Void getq(){}
Void getr(){}

iv) Heirarchical inheritance

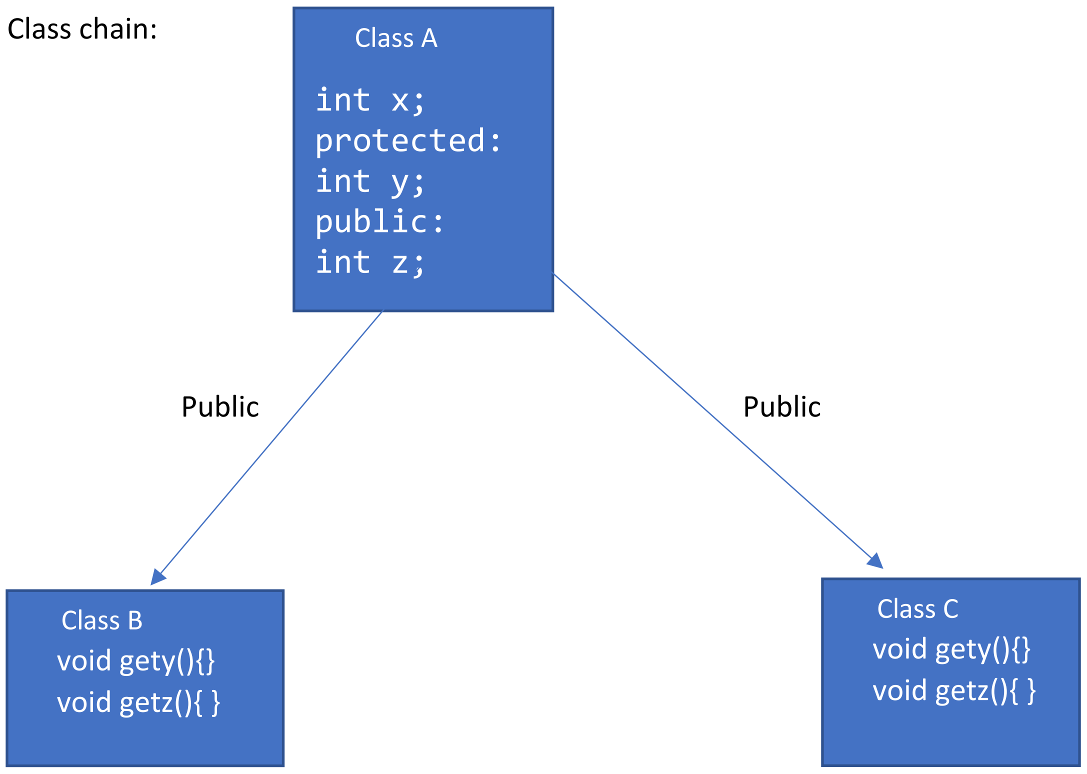
```
class A
{
private:
    int x=5;
protected:
    int y=10;
public:
    int z=15;
};
```

```
class B:public A
{
//write public method to
access x,y & z
public:
      /*void getx()
   /{
     cout<<"x=
"<<x<<endl;
}*/
   //x is private
   void gety(){
   cout<<"y= "<<y<<endl;
   }
   void getz(){
   cout<<"z= "<<z<<endl;
   }

};
```

```
class C:public A{
//write method public
to access x,y & z
  public:
   /*void getx()
   /{
     cout<<"x=
"<<x<<endl;
}*/
   //x is private

   void gety(){
   cout<<"y=
"<<y<<endl;
   }
   void getz(){
   cout<<"z=
"<<z<<endl;
   }


};
```

```
int main()
{
   B b;
   C c;
//call
//methods of
//class B & C
   b.gety();
   b.getz();

   cout<<endl;
   c.gety();
   c.getz();
   return 0;
}
```

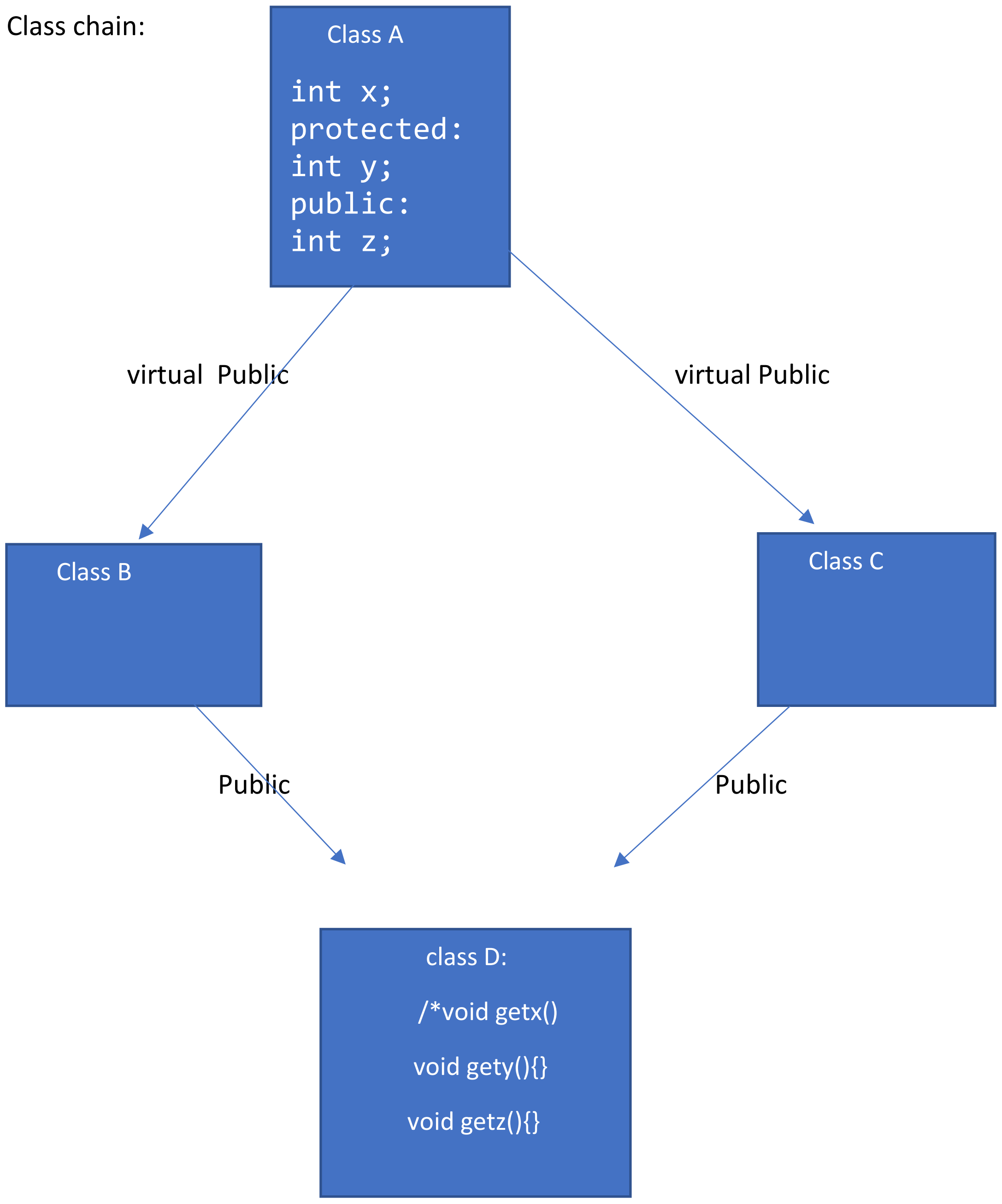
Class chain:


Class A
int x;
protected:
int y;
public:
int z;
Public
Public
Class B
void gety(){}
void getz(){ }
Class C
void gety(){}
void getz(){ }

v) Hybrid (Diamond) inheritance [virtual class]

class A	class B:virtual	class C:virtual	class D:public	
{	public A	public A	B,public C{	
private:	{	{	//write public	int main()
int x=0;	};	};	method to	{
protected:			access x,y & z	D d;
int y=1;			public:	//call
public:			/*void getx()	//methods of
int z=2;			/{	//class D
};			cout<<"x=	d.gety();
			"<<x<<endl;	d.getz();
			}*/	return 0;
			void gety()	}
			{	
			cout<<"y=	
			"<<y<<endl; }	
			void getz()	
			{	
			cout<<"z=	
			"<<z<<endl;	
			}	
			};	

Class chain:


Class A
int x;
protected:
int y;
public:
int z;
virtual Public
virtual Public
Class B
Class C
Public
Public
class D:
/*void getx()
void gety(){}
void getz(){}

Topic 3 [Constructor & Destructor in inheritance]: Write the constructors & destructors for different types of inheritance are given as follows. Also follow and write the sequence of their execution.

i) **Single inheritance**

i) Single inheritance

Code:

```
#include<iostream>


using namespace std;

class A{

private:

int ax;

public:

//write constructor to initialize ax

//Write

   A(){

   cout<<"IN CLASS A"<<endl;;

   ax=50;

   }

   A(int a){

   cout<<"IN CLASS A"<<endl;

   ax=a;

   }

   int getax(){

   return ax;
```

```
    }
      ~A(){
    cout<<"calling the destructor of A"<<endl;
    }
};
class B:public A{
private:
int bx;
public:
//write constructor to
    B(){
    cout<<"IN CLASS B"<<endl;
    bx=60;
    }
 B(int a){
    cout<<"IN CLASS B"<<endl;
    bx=a;
    }
//Write method to sum ax and bx
    void getsum(){
        cout<<"ax= "<<A::getax()<< " bx= "<<bx<<endl;
    cout<<"sum= "<<A::getax()+bx<<endl;
    }
//Write destructor
    ~B(){
    cout<<"calling the destructor of B"<<endl;
    }
```

```
};
int main(){
   B b;
//call methods of class B
   b.getsum();


return 0;
}
```

SEQUENCE AND OUTPUT:

```
IN CLASS A
IN CLASS B
ax= 50 bx= 60
sum= 110
calling the destructor of B
calling the destructor of A
```

ii) Multi-level inheritance

Code:

```
#include<iostream>


using namespace std;
class A{
private:
int ax;
public:
//write constructor to initialize ax
//Write
```

```
    A(){
    cout<<"IN CLASS A"<<endl;;
    ax=50;
    }
    A(int a){
    cout<<"IN CLASS A"<<endl;
    ax=a;
    }
    int getax(){
    return ax;
    }
    ~A(){
    cout<<"calling the destructor of A"<<endl;
    }
};
class B:public A{
private:
int bx;
public:
//write constructor
    B(){
    cout<<"IN CLASS B"<<endl;
    bx=10;
    }
 B(int a){
    cout<<"IN CLASS B"<<endl;
    bx=a;
    }

//Write destructor
```

```
    ~B(){
    cout<<"calling the destructor of B"<<endl;
        }

    int getbx(){
    return bx;
    }
};

class C:public B{
private:
int cx=9;
public:
//write constructor to initialize cx
//Write method to
public:
    C(){
    cout<<"IN CLASS C"<<endl;
    cx=100;
    }
    C(int a){
    cx=a;
    cout<<"IN CLASS C"<<endl;
    }
    //Write destructor
    void getsum(){
    cout<<"ax= "<<A::getax()<<" bx= "<<B::getbx()<<" cx= "<<cx<<endl;
    cout<<"sum= "<<cx+B::getbx()+A::getax()<<endl;
    }
    ~C(){
     cout<<"calling the destructor of C"<<endl;
    }

};
int main(){

    C c;
    c.getsum();

return 0;
}
```

SEQUENCE AND OUTPUT:

```
IN CLASS A
IN CLASS B
IN CLASS C
ax= 50 bx= 10 cx= 100
sum= 160
calling the destructor of C
calling the destructor of B
calling the destructor of A
```

ii) Multiple inheritance

Code:

```
#include<iostream>

using namespace std;
class A{
private:
```

```
int ax;

public:

//write constructor to initialize ax

//Write

   A(){

   cout<<"IN CLASS A"<<endl;;

   ax=100;

   }

   A(int a){

   cout<<"IN CLASS A"<<endl;

   ax=a;

   }

   int getax(){

   return ax;

   }

     ~A(){

   cout<<"calling the destructor of A"<<endl;

   }

};

class B{

private:

int bx;

public:

//write constructor to

   B(){

   cout<<"IN CLASS B"<<endl;

   bx=50;
```

```
    }
 B(int a){
    cout<<"IN CLASS B"<<endl;
    bx=a;
    }
     int getbx(){
    return bx;
    }

//Write destructor
    ~B(){
    cout<<"calling the destructor of B"<<endl;
    }

};

class C:public A,public B{
private:
int cx;
public:
//write constructor to
    C(){
    cout<<"IN CLASS C"<<endl;
    cx=40;
    }
 C(int a){
    cout<<"IN CLASS C"<<endl;
    cx=a;
    }

  void getsum(){
     cout<<"ax= "<<A::getax()<<" bx= "<<B::getbx()<<" cx= "<<cx<<endl;
  cout<<"sum= "<<A::getax()+B::getbx()+cx<<endl;
  }
//Write destructor
    ~C(){
    cout<<"calling the destructor of C"<<endl;
    }

};
int main(){
   C c;
   c.getsum();

return 0;
}
```

SEQUENCE AND OUTPUT:

```
IN CLASS A
IN CLASS B
IN CLASS C
ax= 100 bx= 50 cx= 40
sum= 190
calling the destructor of C
calling the destructor of B
calling the destructor of A
```

iv) Heirarchical inheritance

Code:

```
#include<iostream>

using namespace std;
class A{
private:
int ax;
public:
//write constructor to initialize ax
//Write
  A(){
  cout<<"IN CLASS A"<<endl;;
  ax=50;
  }
  A(int a){
  cout<<"IN CLASS A"<<endl;
  ax=a;
  }
  int getax(){
  return ax;
  }
    ~A(){
```

```
    cout<<"calling the destructor of A"<<endl;

    }

};

class B:public A{

private:

int bx;

public:

//write constructor to

    B(){

    cout<<"IN CLASS B"<<endl;

    bx=10;

    }

 B(int a){

    cout<<"IN CLASS B"<<endl;

    bx=a;

    }

     int getbx(){

    return bx;

    }


//Write destructor

    ~B(){

    cout<<"calling the destructor of B"<<endl;

    }


};
```

```
class C:public A{
private:
int cx;
public:
//write constructor to
    C(){
    cout<<"IN CLASS C"<<endl;
    cx=30;
    }
 C(int a){
    cout<<"IN CLASS C"<<endl;
    cx=a;
    }

  void getsum(){
    B b;
    cout<<"ax= "<<A::getax()<<" bx= "<<b.getbx()<<" cx= "<<cx<<endl;
    cout<<"sum= "<<A::getax()+b.getbx()+cx<<endl;
  }
//Write destructor
    ~C(){
    cout<<"calling the destructor of C"<<endl;
    }

};
int main(){
   B b;
   cout<<"bx= "<<b.getbx()<<endl<<endl;
   C c;
    c.getsum();

return 0;
}
```

SEQUENCE AND OUTPUT:

```
IN CLASS A
IN CLASS B
bx= 10

IN CLASS A
IN CLASS C
IN CLASS A
IN CLASS B
ax= 50 bx= 10 cx= 30
sum= 90
calling the destructor of B
calling the destructor of A
calling the destructor of C
calling the destructor of A
calling the destructor of B
calling the destructor of A
```

v) Hybrid (Diamond) inheritance [virtual class]:

Code:

```
#include<iostream>


using namespace std;
class A{
private:
int ax;
public:
//write constructor to initialize ax
//Write
   A(){
   cout<<"IN CLASS A"<<endl;;
   ax=50;
   }
   A(int a){
   cout<<"IN CLASS A"<<endl;
   ax=a;
   }
   int getax(){
   return ax;
   }
     ~A(){
   cout<<"calling the destructor of A"<<endl;
   }
```

```
};
class B:virtual public A{
private:
int bx;
public:
//write constructor to
   B(){
   cout<<"IN CLASS B"<<endl;
   bx=10;
   }
 B(int a){
   cout<<"IN CLASS B"<<endl;
   bx=a;
   }
    int getbx(){
   return bx;
   }

//Write destructor
   ~B(){
   cout<<"calling the destructor of B"<<endl;
   }

};

class C:virtual public A{
private:
```

```
int cx;
public:
//write constructor to
   C(){
   cout<<"IN CLASS C"<<endl;
   cx=30;
   }
 C(int a){
   cout<<"IN CLASS C"<<endl;
   cx=a;
   }
    int getcx(){
   return cx;
   }

//Write destructor
   ~C(){
   cout<<"calling the destructor of C"<<endl;
   }


};
class D:public C,public B{
private:
   int dx;
public:
   D(){
   cout<<"IN CLASS D"<<endl;
```

```
    dx=40;
    }
 D(int a){
    cout<<"IN CLASS d"<<endl;
    dx=a;
    }

  void getsum(){
  cout<<"sum= "<<A::getax()+B::getbx()+C::getcx()+dx<<endl;
  }
//Write destructor
    ~D(){
    cout<<"calling the destructor of D"<<endl;
    }



};
int main(){

  D d;
  d.getsum();



return 0;
}
```

SEQUENCE AND OUTPUT:

```
IN CLASS A
IN CLASS C
IN CLASS B
IN CLASS D
sum= 130
calling the destructor of D
calling the destructor of B
calling the destructor of C
calling the destructor of A
```