```
//Order of execution of constructor and destructor during multilevel(Parameterized constructor)
#include<iostream>
using namespace std;
class A
{
int x,y;
public:
A(int r,int s)
{
x=r;
y=s;
cout<<"\nCalling base class constructor:"<<x<<" "<<y;
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
int l,m;
public:
B(int p,int q,int r,int s):A(r,s)
{
l=p;
m=q;
cout<<"\nCalling derived class B constructor:"<<l<<" "<<m;
}
~B()
{
cout<<"\nCalling derived B class destructor";
}
};
class C:public B
{
int n,m;
public:
C(int u,int v,int p,int q,int r,int s):B(p,q,r,s)
{
n=u;
m=v;
cout<<"\nCalling derived class C constructor with values:"<<n<<" "<<m;
}
~C()
{
cout<<"\nCalling derived class C destructor";
}
};
int main()
```

```
{
C obj1(1,2,3,4,5,6);
return 0;
}

#include<iostream>
using namespace std;
class A
{
public:
A()
{
cout<<"\nCalling base class constructor";
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
public:
B()
{
cout<<"\nCalling derived class B constructor";
}
~B()
{
cout<<"\nCalling derived B class destructor";
}
};
class C:public B
{
public:
C()
{
cout<<"\nCalling derived class C constructor";
}
~C()
{
cout<<"\nCalling derived class C destructor";
}
};
int main()
{
C obj1;
return 0;
}
---------------------------------------
```

```
#include<iostream>
using namespace std;
class A
{
protected:
int x;
public:
A(int a)
{
x=a;
}
};
class B:public A
{
protected:
int y;
public:
B(int a,int b):A(a)
{
y=b;
}
};
class C:public B
{
public:
C(int a,int b):B(a,b)
{
//No definition
}
void reverse()
{
   int num=x+y;
   int rev=0,digit;
   cout<<"\nSum of x and y is:"<<num;
   while(num!=0)
   {
      digit=num%10;
      rev=rev*10+digit;
      num=num/10;
   }
   cout<<"\nReverse is:"<<rev;
}
};
int main()
{
C obj(120,131);
obj.reverse();
return 0;
}
```

```
#include<iostream>
using namespace std;
class M
{
	protected:
		int m;
		public:
		M(int x)
		{
		   m=x;
		   cout<<"\nIn M";
		}
};
class N
{
	protected:
		int n;
		public:
		N(int y)
		{
		   n=y;
		   cout<<"\nIn N";

		}
};
class P:public N,public M//ORDER OF INHERITANCE(Order of execution depends upon this)
{
	int l;
	public:
		P(int p,int q,int r):N(q),M(r)//Order of execution does not depend upon this sequence
		{
		l=p;
		cout<<"\nIn P";
		}
		void display()
		{
		cout<<"m="<<m<<" "<<"n="<<n<<" "<<"l="<<l;
		}
};
int main()
{
	P obj1(3,2,1);
	obj1.display();
	return 0;
}
```

Note:virtual base class will take more priority as compared to normal base class

---

```
#include<iostream>
```

```
using namespace std;
class M
{
	protected:
		int m;
		public:
		M(int x)
		{
		   m=x;
		   cout<<"\nIn M";
		}
		~M()
		{
			cout<<"\n Base class desturctor";
		}
};
class N:public M
{
	protected:
		int n;
		public:
		N(int y):M(y)
		{
		   n=y;
		   cout<<"\nIn N:"<<n;

		}
		~N()
		{
			cout<<"\n Derived class N desturctor";
		}

};
class P:public M
{
	int l;
	public:
		P(int p):M(p)
		{
		l=p;
		cout<<"\nIn P:"<<l;
		}
		~P()
		{
			cout<<"\n Derived class P desturctor";
		}
};
int main()
{
```

```
        N obj1(1);
        P obj2(2);
        return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class A
{
public:
A()
{
  cout<<"\nA";
}
~A()
{
  cout<<"\nA Destructor";
}

};
class B:public A
{
public:
B()
{
  cout<<"\nB";
}
~B()
{
  cout<<"\nB Destructor";
}
};
class C:public A
{
public:
C()
{
  cout<<"\nC";
}
~C()
{
  cout<<"\nC Destructor";
}
};
int main()
{
B obj1;
C obj2;
return 0;
```

```
}
---------------------------------------
#include<iostream>
using namespace std;
class overloading
{
public:
int area(int side)
{
	return (side*side);
}
int area(int length,int breadth)
{
	return (length*breadth);
}
float area(float radius)
{
	return (3.14*radius*radius);
}
};
int main()
{
	overloading obj1;
	int square,rectangle;
	float circle;
	square=obj1.area(5);
	cout<<"\n Area of square is:"<<square;
	rectangle=obj1.area(3,4);
	cout<<"\n Area of rectangle is:"<<rectangle;
	circle=obj1.area(3.4f);
	cout<<"\n Area of circle is:"<<circle;
	return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class binary
{
int x;
public:
binary()
{
  x=0;
}
binary(int x1)
{
x=x1;
}
binary operator*(binary obj1)
```

```
{
binary temp;
temp.x=x*obj1.x;
return temp;
}
void show_data()
{
cout<<x<<"\n";
}
};
int main()
{
binary o2(5),o3(2),o1;
o1=o2*o3;
//o1=o2.operator*(o3);
o1.show_data();
}
```