```
//PROGRAM TO DEMONSTRATE THE CONCEPT OF POINTER TO DERIVED CLASS OBJECT
#include<conio.h>
#include<iostream.h>



class base
{
    public:
        void display()
        {
            cout<<"base class display called\n";
            }
            };
            class derv1:public base
            {
                public:
                    void display()
                    {
                        cout<<"derv1 display's called \n";
                        }
                        };
                        class derv2:public base
                        {
                            public:
                                void display()
```

```
{
cout<<"derv2 display's called \n";
}
};
int main()
{
   base *ptr; //pointer to base class
//   base b;
  // derv1 *p;
   derv1 d1; //object of first derieved class
  // derv2 d2; //object of second derieved class
   ptr=&d1; //address of d1 to base pointer
   ptr->display();
   //ptr=&d2;
   //ptr->display();
  // p=&b;
  // p->display();
   getch();
   return 0;
   }
```

//PROGRAM TO DEMONSTRATE THE CONCEPT OF VIRTUAL FUNCTION

```
#include<conio.h>
#include<iostream.h>
class base
{
```

```
public:
    virtual void display()=0;
    {
        cout<<"base class display called\n";
        }
        };
        class derv1:public base
        {
            public:
                void display()
                {
                    cout<<"derv1 display's called \n";
                    }
                    };
                    class derv2:public base
                    {
                        public:
                            void display()
                            {
                            cout<<"derv2 display's called \n";
                            }
                            };
                            int main()
                            {
                               base *ptr; //pointer to base class
```

```
                derv1 d1; //object of first derieved class
                derv2 d2; //object of second derieved class
                ptr=&d1; //address of d1 to base pointer
                ptr->display();
                ptr=&d2;
                ptr->display();
                getch();
                return 0;
                }
        //PROGRAM TO CALCULATE THE AREA OF DIFFERENT SHAPES
#include<conio.h>
#include<iostream.h>
class shape
{
    protected:
        double a,b;
        public:
            void read()
            {

               cin>>a>>b;
               }
             virtual void cal_area()=0; //virtual function
           //    {
            //          cout<<"virtual function providing single interface";
```

```
//          }
            };
            class rectangle : public shape
            {
                public:
                    void cal_area()
                    {
                        double area=a*b;
                        cout<<"area of rectangle ="<<area;
                        }
                        };
                        class triangle :public shape
                        {
                            public:
                                void cal_area()
                                {
                                    double area=(a*b)/2;
                                    cout<<"area of triangle = "<<area;
                                    }
                                    };
                                    int main()
                                    {
                                       shape *ptr; //pointer to shape class
                                       shape s;
                                       rectangle r1;
```

```
                                        cout<<"enter the length and breadth of rectangle :";
                                        r1.read();
                                        ptr=&r1;//assigning address of r1 to ptr
                                        ptr->cal_area(); //invoke cal_area of rectangle
                                        triangle t1;
                                        cout<<"\nenter the base and perpendicular of triangle :";
                                        t1.read();
                                        ptr=&t1;
                                        ptr->cal_area();
                                        getch();
                                        return 0;
                                        }
                        //PROGRAM TO CALCULATE THE AREA OF DIFFERENT SHAPES USING PURE
VIRTUAL FUNCTION
#include<conio.h>
#include<iostream.h>
class shape //abstract class
{
    protected:
            double a,b;
            public:
                void read()
                {

                   cin>>a>>b;
                   }
```

```
  virtual void cal_area()=0 ;// pure virtual function
//  cout<<"hello";
        };
        class rectangle : public shape
        {
             public:
                  void cal_area()
                  {
                      double area=a*b;
                      cout<<"area of rectangle ="<<area;
                      }
                      };
                      class triangle :public shape
                      {
                           public:
                                void cal_area()
                                {
                                    double area=(a*b)/2;
                                    cout<<"area of triangle = "<<area;
                                    }
                                    };
                                    int main()
                                    {
                                      //  shape s1;
                                        shape *ptr[2]; //pointer to shape class
```

```
                                        rectangle r1;
                                        cout<<"enter the length and breadth of rectangle :";
                                        r1.read();
                                        //ptr=&r1;//assigning address of r1 to ptr
                                        //ptr->cal_area(); //invoke cal_area of rectangle
                                        triangle t1;
                                        cout<<"\nenter the base and perpendicular of triangle :";
                                        t1.read();
                                        ptr[0]=&r1; //assigning address of r1 to ptr[0]
                                        ptr[1]=&t1; //assigning address of t1 to ptr[1]
                                        for(int i=0;i<2;i++)
                                        ptr[i]->cal_area();
                                        getch();
                                        return 0;
                                        }
                                //PROGRAM TO DEMONSTRTE THE NEED OF VIRTUAL DESTRUCTOR
#include<conio.h>
#include<iostream.h>
class base
{
    public:
        base()
        {
            cout<<"\nbase constructor";
            }
```

```
~base()
{
    cout<<"\nbase destructor ";
    }
    };
    class derive:public base
    {
        int *a;
        public:
            derive()
            {
                cout<<"\nderive constructor allocating 10 bytes of memory";
              //    a=new int [5];
                }
                ~derive()
                {
                    cout<<"derive destructor frees 10 bytes of memory\n";
                  //  delete[]a;
                    }
                    };
                    int main()
                    {
                       base*ptr;
                       ptr=new derive;//create dynamic derived objecct
                       delete ptr;//destroying dynamic derived object
```

```
                                        getch();
                                         return 0;
                                         }
//PROGRAM TO DEMONSTRTE THE NEED OF VIRTUAL DESTRUCTOR
#include<conio.h>
#include<iostream.h>
class base
{
    public:
        base()
        {
            cout<<"\nbase constructor";
            }
           virtual ~base()
            {
                cout<<"\nbase destructor ";
                }
                };
                class derive:public base
                {
                    int *a;
                    public:
                        derive()
                        {
                            cout<<"\nderive constructor allocating 10 bytes of memory";
```

```
a=new int [5];
}
~derive()
{
        cout<<"\nderive destructor frees 10 bytes of memory\n";
        delete[]a;
        }
        };
        int main()
        {
           base*ptr;
           ptr=new derive;//create dynamic derived objecct
           delete ptr;//destroying dynamic derived object
          getch();
           return 0;
           }
```