```
#include<iostream>
using namespace std;
class BC
{
	public:
	   void printBC()
	   {
	      cout<<"\nPrinting message in base class"<<endl;
	   }
	      void show()
	      {
	            cout<<"\nshow() of base class"<<endl;
	      }
};
class DC:public BC
{
	public:
	   void printDC()
	   {
	      cout<<"\nPrinting message in derived class"<<endl;
	   }
	      void show()
	      {
	            cout<<"\nshow() of derived class"<<endl;
	      }
};
int main()
{
	BC *bptr;
	BC base;
	bptr=&base;//Base pointer can point towards base class
	cout<<"bptr points to base objects\n";
	bptr->show();
	//derived class
	DC derived;
	bptr=&derived;//Base pointer can point towards derived class
	cout<<"bptr now points to derived objects\n";
//    bptr->printDC();//Base pointer cannot access specific members of derived directly (error)
	bptr->show(); //Base pointer can access the common members in base and derived, show() of base is
called due to early binding
   //accessing data using a pointer of type derived class DC//
   DC *dptr;    //derived type pointer
   dptr=&derived;//derived pointer can point towards its own object only
   cout<<"dptr is derived type pointer\n";
   dptr->show();//derived pointer can access its own members
   dptr->printDC();//derived pointer can access its own members
   cout<<"using ((DC*)bptr)\n";
   ((DC*)bptr)->show();//Base pointer can access members of derived through type casting
   ((DC*)bptr)->printDC();//Base pointer can access members of derived through type casting
```

```
//    dptr=&base;//Derived pointer cannot point towards base class(error)
   return 0;
   }
   ---------------------------------
   #include<iostream>
using namespace std;
class BC
{
      public:
            void show()
            {
                  cout<<"\nshow() of base class"<<endl;
            }
};
class DC:public BC
{
      public:
            void show()
            {
cout<<"\nshow() of derived class"<<endl;
            }
};
int main()
{
      BC *bptr;
      BC base;
      bptr=&base;
      cout<<"\n-----Early Binding------";
      cout<<"bptr points to base objects\n";
      bptr->show();//Base class show is called
      //derived class
      DC derived;
      bptr=&derived;
      cout<<"bptr now points to derived objects\n";
      bptr->show(); //Base class show is called
   return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class BC
{
      public:
            virtual void show()
            {
                  cout<<"\nshow() of base class"<<endl;
            }
};
class DC:public BC
```

```
{
	public:
		void show()
		{
cout<<"\nshow() of derived class"<<endl;
		}
};
int main()
{
	BC *bptr;
	BC base;
	bptr=&base;
	cout<<"\n-----Runtime polymorphism-----";
	cout<<"\nbptr points to base objects\n";
	bptr->show();//Base class show is called
	//derived class
	DC derived;
	bptr=&derived;
	cout<<"bptr now points to derived objects\n";
	bptr->show(); //Derived class show is called
   return 0;
}
---------------------------------------
//Program to show Early and Late binding

#include <iostream>
using namespace std;
class base {
public:
	virtual void print()
	{
		cout << "print base class" << endl;
	}

	void show()
	{
		cout << "show base class" << endl;
	}
};

class derived : public base {
public:
	void print()
	{
		cout << "print derived class" << endl;
	}
	void show()
	{
		cout << "show derived class" << endl;
```

```
        }
};
int main()
{
        base* bptr;
        derived d;
        bptr = &d;
        // virtual function, binded at runtime (Late binding)
        bptr->print();
// Non-virtual function, binded at compile time(Early binding)
        bptr->show();
}
---------------------------------------
#include<iostream>
using namespace std;
class sample
{
public:
virtual void example()=0;
void show()
{
cout<<"\nThis is sample abstract class";
}
};
class derived1:public sample
{
public:
void example()
{
cout<<"C++";
}
};
int main()
{
sample *ptr;
//sample obj;//Compile time error(Creating object of abstract class)
derived1 obj1;
ptr=&obj1;
ptr->example();
ptr->show();
}
-----------------------------------
#include<iostream>
using namespace std;
class shape
{
public:
virtual void area()=0;
};
```

```
class circle:public shape
{
float radius;
public:
void area()
{
float area1;
cout<<"\nEnter radius:";
cin>>radius;
area1=3.14*radius*radius;
cout<<"\nArea of circle is:"<<area1;
}
};
class rectangle:public shape
{
int length,breadth;
public:
void area()
{
int ans;
cout<<"\nEnter length and breadth:";
cin>>length>>breadth;
ans=length*breadth;
cout<<"\nArea of rectangle is:"<<ans;
}
};
int main()
{
shape *bptr;
circle obj1;
rectangle obj2;
bptr=&obj1;
bptr->area();
bptr=&obj2;
bptr->area();
return 0;
}
```