Polymorphism:

1. Employee:

```
using namespace std;
#include<iostream>
#include<string.h>
struct Employee                         //Base class
{
        int emp_id;
        char name[20];
        double salary;
        Employee()
        {
                cout<<"\n\nEmp default constructor called\n";
                this->emp_id=0;
                strcpy(this->name,"not_given");
                this->salary=0;
        }
        Employee(int i,const char* n,double s)
        {
                cout<<"\n\nEmp parameterised called\n";
                this->emp_id=i;
                strcpy(this->name,n);
                this->salary=s;
        }
        void setId(int i)  //setters(mutators)
```

```
{
        this->emp_id=i;
}
void setName(const char* n)     //setters(mutators)
{
        strcpy(this->name,n);
}
void setSalary(double s) //setters(mutators)
{
        this->salary=s;
}
int getId()         //getters(accessors)
{
        return this->emp_id;
}
char* getName()         //getters(accessors)
{
        return this->name;
}
double getSalary()      //getters(accessors)
{
        return this->salary;
}
virtual void display()
{
```

```
		cout<<"\nemployees detail: \nid: "<<this->emp_id<<"\tname: "<<this->name<<"\tsalary: "<<this->salary<<"\n";
	}
	virtual double calSal()
	{
		return salary;
	}
};
struct SalesMan:public Employee		//step 1
{
	int target;
	double intensive;
	SalesMan():Employee()  //step 2(a)- default constructor of Sales manager and call given to default constructor of Employee
	{
		cout<<"\n\nSM default constructor called\n";
		this->target=0;
		this->intensive=0;
	}
	SalesMan(int i,const char* n,double s,int t,int in):Employee(i,n,s)		//step 2(b) parameterised constructor of Sales manager and call given to parameterised constructor of Employee
	{
		cout<<"\n\nSM parameterised constructor called\n";
		this->target=t;
		this->intensive=in;
	}
	void setTarget(int t)		//extra setters(mutator) required for Sales manager
```

```
	{
		this->target=t;
	}
	void setIntense(double in)			//extra setters(mutator) required for Sales manager
	{
		this->intensive=in;
	}
	int getTarget()		//extra getters(accessor) required for sales manager
	{
		return this->target;
	}
	double getIntense()			//extra getters(accessor) required for sales manager
	{
		return this->intensive;
	}
	void display()
	{
		Employee::display();	//step 3- call given to display function of Employee class
		cout<<"\ntarget: "<<this->target<<"\tintensive: "<<this->intensive;
	}
	double calSal()		//different implementation
	{
		return salary+intensive;
	}
};
```

```
struct Admin:public Employee    //step 1
{
        double allowance;
        Admin():Employee()      //step 2(a)-default constructor of Admin and call given to default constructor of Employee
        {
                cout<<"\n\nAdmin default constructor called\n";
                this->allowance=0;
        }
        Admin(int i,const char* n,double s,double a):Employee(i,n,s)      //step 2(b)-parameterised constructor of Admin and call given to parameterised constructor of Employee
        {
                cout<<"\n\nAdmin parameterised constructor called\n";
                this->allowance=a;
        }
        void setAllow(double a)             //extra setters(mutator) required for Admin
        {
                this->allowance=a;
        }
        double getAllow()         //extra getters(accessor) required for Admin
        {
                return this->allowance;
        }
        void display()    //step 3
        {
                Employee::display();      //call given to display function of Employee class
```

```
		cout<<"\tallowance: "<<this->allowance<<"\n";
	}
	double calSal()		//different implementation
	{
		return salary+allowance;
	}
};
struct HrManager:public Employee	//step 1
{
	double commission;
	HrManager():Employee()		//step 2(a)-default constructor of HR manager and call given to default constructor of Employee
	{
		cout<<"\n\nHR default constructor called\n";
		this->commission=0;
	}
	HrManager(int i,const char* n,double s,double c):Employee(i,n,s)		//step 2(b)-parameterised constructor of HR manager and call given to parameterised constructor of Employee
	{
		cout<<"\n\nHR parameterised constructor called\n";
		this->commission=c;
	}
	void setComm(double c)			//extra setters(mutator) required for HR manager
	{
		this->commission=c;
	}
```

```
        double getComm()        //extra getters(accessor) required for HR manager
        {
                return this->commission;
        }
        void display()
        {
                Employee::display();      //step 3- call given to display function of Employee class
                cout<<"\tCommission: "<<this->commission<<"\n";
        }
        double calSal()             //different implementation
        {
                return salary+commission;
        }
};
int main()
{
        int id,target;
        char name[20];
        double salary,intensive,allowance,commission;
        Employee *e;
        Employee e1;
        Employee e2(42,"pragati",60000);
        e=&e2;
        e->display();
        SalesMan m1;
```

```
cout<<"enter sale managers id:\n";
cin>>id;
cout<<"\nenter the name of sales manager:\n";
cin>>name;
cout<<"\nenter salary of sales manager:\n";
cin>>salary;
cout<<"\nenter target of sales manager:\n";
cin>>target;
cout<<"\nenter intensive for target completion:\n";
cin>>intensive;
m1.setId(id);
m1.setName(name);
m1.setSalary(salary);
m1.setTarget(target);
m1.setIntense(intensive);
e=&m1;
e->display();
cout<<"\nCalculated salary is: "<<e->calSal();
SalesMan m2(22,"pragati",50000,45,4500);
e=&m2;
e->display();
cout<<"\nCalculated salary is: "<<e->calSal();
Admin a1;
cout<<"enter admin id:\n";
cin>>id;
```

```
cout<<"\nenter name of the admin:\n";
cin>>name;
cout<<"\nenter salary of admin:\n";
cin>>salary;
cout<<"\nallowance for admin:\n";
cin>>allowance;
a1.setId(id);
a1.setName(name);
a1.setSalary(salary);
a1.setAllow(allowance);
e=&a1;
e->display();
cout<<"\nCalculated salary is: "<<e->calSal();
Admin a2(101,"pragati",50000,4500);
e=&a2;
e->display();
cout<<"\nCalculated salary is: "<<e->calSal();
HrManager h1;
cout<<"\nenter hr managers id:\n";
cin>>id;
cout<<"\nenter name of hr manager:\n";
cin>>name;
cout<<"\nenter salary of hr manager:\n";
cin>>salary;
cout<<"\nenter commission for hr manager:\n";
```

```
        cin>>commission;
        h1.setId(id);
        h1.setName(name);
        h1.setSalary(salary);
        h1.setComm(commission);
        e=&h1;
        e->display();
        cout<<"\nCalculated salary is: "<<e->calSal();
        HrManager h2(202,"pragati",50000,5000);
        e=&h2;
        e->display();
        cout<<"\nCalculated salary is: "<<e->calSal();
        return 0;
}
```

2. Shape:

```
using namespace std;
#include<iostream>
struct Shape      //Base class
{
        double area;
        Shape()             //Default constructor
        {
                cout<<"\nShape default constructor called\n";
                this->area=0;
```

```
        }
        Shape(double a)                 //Parameterised constructor
        {
                cout<<"\nShape parameterised constructor called\n";
                this->area=a;
        }
        void setArea(double a)  //seeter(mutator)
        {
                this->area=a;
        }
        double getArea()        //getter(accessor)
        {
                return this->area;
        }
        virtual void drawn()
        {
                cout<<"\nArea of shape is: "<<this->area<<"\n";
        }
        virtual void calArea()
        {
                cout<<"\nArea is: "<<area<<"\n";
        }
};//Shape class ends here
struct Circle:public Shape              //derived class
{
```

```
    double radius;
    Circle()  //default constructor of circle where default constructor of shape class called implicitly
    {
        cout<<"\nCircle default constructor called\n";
        this->area=0;
    }
    Circle(double r) //parameterised constructor of circle where default constructor of shape class called implicitly
    {
        cout<<"\nCircle Parameterised constructor called\n";
        this->radius=r;
    }
    void setRadius(double r)        //extra setter(mutator) required for circle
    {
        this->radius=r;
    }
    double getRadius()              //extra getter(accessor) required for circle
    {
        return this->radius;
    }
    void drawn()
    {
        cout<<"\nRadius of circle is: "<<this->radius<<"\n";
        Shape::drawn();         //call to drawn function of shape class
    }
    void calArea()              //different implementation (function to calculate area of circle)
```

```
	{
		cout<<"\nArea of Circle is: "<<3.14*radius*radius<<"\n";
	}
};
struct Rectangle:public Shape	//derived class
{
	double length,breadth;
	Rectangle()		//default constructor of Rectangle class where default constructor of Shape class called implicitly
	{
		cout<<"\nRectangle default constructor called\n";
		this->length=0;
		this->breadth=0;
	}
	Rectangle(double l,double b)	//parameterised constructor of Rectangle class where default constructor of Shape class called implicitly
	{
		cout<<"\nRectangle parameterized constructor called\n";
		this->length=l;
		this->breadth=b;
	}
	void setLength(double l)		//extra setter required for rectangle class
	{
		this->length;
	}
	void setBreadth(double b)		//extra setter required for rectangle class
```

```
	{
		this->breadth;
	}
	double getLength()		//extra getter required for rectangle class
	{
		return this->length;
	}
	double getBreadth()		//extra getter required for rectangle class
	{
		return this->breadth;
	}
	void drawn()
	{
		cout<<"\nLength of Rectangle is: "<<this->length<<"\nLength of Rectangle is: "<<this->breadth<<"\n";
		Shape::drawn();		//call given to drawn function of Shape class
	}
	void calArea()		//different implementation (function to calculate area of rectangle)
	{
		cout<<"\nArea of Rectangle is: "<<length*breadth<<"\n";
	}
};
struct Triangle:public Shape	//derived class
{
	double height,base;
```

```
	Triangle()			//default constructor of Triangle class where default constructor of Shape class called implicitly
	{
		cout<<"\nTriangle default constructor called\n";
		this->base=0;
		this->height=0;
	}
	Triangle(double b,double h)		//parameterised constructor of Triangle class where default constructor of Shape class called implicitly
	{
		cout<<"\nTriangle parameterised constructor called\n";
		this->base=b;
		this->height=h;
	}
	void setBase(double b)		//extra setter required for Triangle class
	{
		this->base=b;
	}
	void setHeight(double h)		//extra setter required for Triangle class
	{
		this->height=h;
	}
	double getBase()		//extra getter required for Triangle class
	{
		return this->base;
	}
```

```
	double getHeight()			//extra getter required for Triangle class
	{
		return this->height;
	}
	void drawn()
	{
		cout<<"\nBase of Triangle is: "<<this->base<<"\nHeight of Triangle is: "<<this->height<<"\n";
		Shape::drawn();		//call given to drawn function of Shape class
	}
	void calArea()		//different implementation (function to calculate arean of triangle)
	{
		cout<<"\nArea of Triangle is: "<<0.5*base*height<<"\n";
	}
};
int main()
{
	double radius,length,breadth,base,height;
	Shape s1;
	Shape s2(36);
	Shape *s;
	Circle c1;
	cout<<"\nEnter radius of circle: ";
	cin>>radius;
	c1.setRadius(radius);
	s=&c1;
```

```
s->drawn();
s->calArea();
Circle c2(5);
s=&c2;
s->drawn();
s->calArea();
Triangle t1;
cout<<"\nEnter base and height of triangle: ";
cin>>base>>height;
t1.setBase(base);
t1.setHeight(height);
s=&t1;
s->drawn();
s->calArea();
Triangle t2(3,5);
s->drawn();
s->calArea();
Rectangle r1;
cout<<"\nEnter length and breadth of rectangle: ";
cin>>length>>breadth;
r1.setLength(length);
r1.setBreadth(breadth);
s=&r1;
s->drawn();
s->calArea();
```

```
	Rectangle r2(5,3);
	s=&r2;
	s->drawn();
	s->calArea();
	return 0;
}
```

3. Bicycle:

```
using namespace std;
#include<iostream>
#include<string.h>
struct Cycle
{
	int no_of_wheel,no_of_bottle_holder;
	char cname[20],brake_type[20];
	Cycle()
	{
		cout<<"\nCycle default constructor called\n";
		strcpy(this->cname,"not given");
		this->no_of_wheel=0;
		this->no_of_bottle_holder=0;
		strcpy(this->brake_type,"not given");
	}
	Cycle(char* cn,int w,int bh,char* b)
	{
```

```
        cout<<"\nCycle parameterised constructor called\n";
        strcpy(this->cname,cn);
        this->no_of_wheel=w;
        this->no_of_bottle_holder=bh;
        strcpy(this->brake_type,b);
}
void setCompany(char* cn)
{
        strcpy(this->cname,cn);
}
void setWheels(int w)
{
        this->no_of_wheel=w;
}
void setBottleHolder(int bh)
{
        this->no_of_bottle_holder=bh;
}
void setBrake(char* bt)
{
        strcpy(this->brake_type,bt);
}
char* getCompany()
{
        return this->cname;
```

```
	}
	int getWheel()
	{
		return this->no_of_wheel;
	}
	int getBottleHolder()
	{
		return this->no_of_bottle_holder;
	}
	char* getBrake()
	{
		return this->brake_type;
	}
	virtual void display()
	{
		cout<<"\nCycles details:\nCompany name: "<<this->cname<<"\nNo of wheels: "<<this->no_of_wheel<<"\nNo of bottle holder: "<<this->no_of_bottle_holder<<"\nBrake type: "<<this->brake_type<<"\n";
	}
	virtual void toStart()
	{
		cout<<"\nCycle starts\n";
	}
};
struct GearCycle:public Cycle
{
```

```
int no_of_gears;
GearCycle():Cycle()
{
	cout<<"\nGear cycle default constructor called\n";
	this->no_of_gears=0;
}
GearCycle(char* cn,int w,int bh,char* bt,int g):Cycle(cn, w, bh, bt)
{
	cout<<"\nGear cycle parameterised constructor called\n";
	this->no_of_gears=g;
}
void setGear(int g)
{
	this->no_of_gears=g;
}
int getGear()
{
	return this->no_of_gears;
}
void display()
{
	Cycle::display();
	cout<<"No of gears: "<<this->no_of_gears<<"\n";
}
void toStart()
```

```
        {
                cout<<"\nChange gear to neutral and start pandaling\n";
        }
};
struct ElectricCycle:public Cycle
{
        double battery_power;
        ElectricCycle():Cycle()
        {
                cout<<"\nElectric cycle default constructor called\n";
                this->battery_power=0;
        }
        ElectricCycle(char* cn,int w,int bh,char* bt,double b):Cycle(cn, w, bh, bt)
        {
                cout<<"\nElectric cycle parameterised constructor called\n";
                this->battery_power=b;
        }
        void setBattery(double b)
        {
                this->battery_power=b;
        }
        double getBattery()
        {
                return this->battery_power;
        }
```

```
        void display()
        {
                Cycle::display();
                cout<<"Battery power: "<<this->battery_power<<"\n";
        }
        void toStart()
        {
                cout<<"\nUsing start button starts\n";
        }
};
int main()
{
        Cycle *c;
        //GearCycle g1;
        //c=&g1;
        //c->display();
        //c->toStart();
        GearCycle g2("keysto",2,1,"disc_brake",14);
        c=&g2;
        c->display();
        c->toStart();
        ElectricCycle e1;
        c=&e1;
        c->display();
        c->toStart();
```

```
	ElectricCycle e2("keysto",2,2,"drum_brake",5000);
	c=&e2;
	c->display();
	c->toStart();
	return 0;
}
```

4. Payment method:

```
using namespace std;
#include<iostream>
#include<string.h>
struct PaymentMethod
{
	char sender[20],receiver[20];
	double amount;
	PaymentMethod()
	{
		cout<<"\nPayment default construct called\n";
		strcpy(this->sender,"not_given");
		strcpy(this->receiver,"not_given");
		this->amount=0;
	}
	PaymentMethod(char* s,char* r,double a)
	{
		cout<<"\nPayment parameterised construct called\n";
```

```
        strcpy(this->sender,s);
        strcpy(this->receiver,r);
        this->amount=a;
}
void setSender(char* s)
{
        strcpy(this->sender,s);
}
void setReceiver(char* r)
{
        strcpy(this->receiver,r);
}
void setAmount(double a)
{
        this->amount=a;
}
char* getSender()
{
        return this->sender;
}
char* getReceiver()
{
        return this->receiver;
}
double getAmount()
```

```
	{
		return this->amount;
	}
	virtual void display()
	{
		cout<<"\nPayment Details:\nSender name: "<<this->sender<<"\t Receiver name: "<<this->receiver<<"\t Amount: "<<this->amount<<"\n";
	}
	virtual void payment()
	{
		cout<<"\nPayment can done by cash\n";
	}
};
struct DebitCard:public PaymentMethod
{
	int accNo,cvv;
	DebitCard():PaymentMethod()
	{
		cout<<"\nDebit card default constructor called\n";
		this->accNo=0;
		this->amount=0;
	}
	DebitCard(char* s,char* r,double a,int acc,int cvv):PaymentMethod(s,r,a)
	{
		cout<<"\nDebit card parameterised constructor called\n";
		this->accNo=acc;
```

```
        this->cvv=cvv;
}
void setAccNo(int acc)
{
        this->accNo=acc;
}
void setCvv(int cvv)
{
        this->cvv=cvv;
}
int getAccNo()
{
        return this->accNo;
}
int getCvv()
{
        return this->cvv;
}
void display()
{
        PaymentMethod::display();
        cout<<"\nAccount no: "<<this->accNo<<"\t cvv: "<<this->cvv<<"\n";
}
void payment()
{
```

```
                cout<<"\nPayment done by swiping card using swipe machine\n";
        }
};
struct UPI:public PaymentMethod
{
        int upiId;
        UPI():PaymentMethod()
        {
                cout<<"\nUpi default constructor called\n";
                this->upiId=0;
        }
        UPI(char* s,char* r,double a,int ui):PaymentMethod(s,r,a)
        {
                cout<<"\nUpi parameterised constructor called\n";
                this->upiId=ui;
        }
        void setUpiId(int ui)
        {
                this->upiId=ui;
        }
        int getUpiId()
        {
                return this->upiId;
        }
        void display()
```

```
	{
		PaymentMethod::display();
		cout<<"\nUPI id: "<<this->upiId<<"\n";
	}
	void payment()
	{
		cout<<"\nPayment done by scanning QR code using mobile\n";
	}
};
int main()
{
	char sender[20],receiver[20];
	int accNo,cvv,upiId;
	double amount;
	PaymentMethod *p;
	DebitCard d1;
	cout<<"\nEnter sender name: ";
	cin>>sender;
	cout<<"\nEnter receiver name: ";
	cin>>receiver;
	cout<<"\nEnter amount to pay: ";
	cin>>amount;
	cout<<"\nEnter account no.: ";
	cin>>accNo;
	cout<<"\nEnter cvv: ";
```

```
cin>>cvv;
d1.setSender(sender);
d1.setReceiver(receiver);
d1.setAmount(amount);
d1.setAccNo(accNo);
d1.setCvv(cvv);
p=&d1;
p->display();
p->payment();
DebitCard d2("Prakruti","Pragati",50000,2330409,234);
p=&d2;
p->display();
p->payment();
UPI u1;
cout<<"\nEnter sender name: ";
cin>>sender;
cout<<"\nEnter receiver name: ";
cin>>receiver;
cout<<"\nEnter amount to pay: ";
cin>>amount;
cout<<"\nEnter upi id: ";
cin>>upiId;
u1.setSender(sender);
u1.setReceiver(receiver);
u1.setAmount(amount);
```

```
	u1.setUpiId(upiId);
	p=&u1;
	p->display();
	p->payment();
	UPI u2("Prakruti","Pragati",50000,2330904);
	p=&u2;
	p->display();
	p->payment();
	return 0;
}
```

5. Water Source:

```
using namespace std;
#include<iostream>
#include<string.h>
struct WaterSource
{
	char name[20];
	double waterSalinity;
	char waterQuality[20];
	WaterSource()
	{
		cout<<"\nWater source default constructor called\n";
		strcpy(this->name,"not given");
```

```
        this->waterSalinity=-1;
        strcpy(this->waterQuality,"not_given");
}
WaterSource(char* n,double s,char* q)
{
        cout<<"\nWater source parameterised constructor called\n";
        strcpy(this->name,n);
        this->waterSalinity=s;
        strcpy(this->waterQuality,q);
}
void setName(char* n)
{
        strcpy(this->name,n);
}
void setWaterQuality(char* q)
{
        strcpy(this->waterQuality,q);
}
void setWaterSalinity(double s)
{
        this->waterSalinity=s;
}
char* getName()
{
        return this->name;
```

```
	}
	char* getWaterQuality()
	{
		return this->waterQuality;
	}
	double getWaterSalinity()
	{
		return this->waterSalinity;
	}
	virtual void display()
	{
		cout<<"\nName: "<<this->name<<"\t  Quality: "<<this->waterQuality<<"\t  Salinity: "<<this->waterSalinity<<"\n";
	}
	virtual void giveWater()
	{
		cout<<"\nWater Source provide us stored water\n";
	}
};
struct Dam:public WaterSource
{
	int noDoors;
	double capacity;
	Dam():WaterSource()
	{
		cout<<"\nDam default constructor called\n";
```

```
        this->noDoors=0;
        this->capacity=0;
}
Dam(char* n,double s,char* q,int d,double c):WaterSource(n,s,q)
{
        cout<<"\nDam parameterised constructor called\n";
        this->noDoors=d;
        this->capacity=c;
}
void setDoors(int d)
{
        this->noDoors=d;
}
void setCapacity(double c)
{
        this->capacity=c;
}
int getDoors()
{
        return this->noDoors;
}
double getCapacity()
{
        return this->capacity;
}
```

```
	void display()
	{
		cout<<"\nWater source: Dam\nDetails:\n";
		WaterSource::display();
		cout<<"\nNo of doors: "<<this->noDoors<<"\tCapacity: "<<this->capacity<<" tmc\n";
	}
	void giveWater()
	{
		cout<<"\nBy opening doors of dam, water stored in dam is becomes available for us to use\n";
	}
};
struct River:public WaterSource
{
	double depth,width;
	char origin[20];
	River():WaterSource()
	{
		cout<<"\nRiver default constructor called\n";
		strcpy(this->origin,"not given");
		this->depth=0;
		this->width=0;
	}
	River(char* n,double s,char* q,char* sn,double d,double w):WaterSource(n,s,q)
	{
		cout<<"\nRiver parameterised constructor called\n";
```

```
        strcpy(this->origin,sn);
        this->depth=d;
        this->width=w;
}
void setOrigin(char* sn)
{
        strcpy(this->origin,sn);
}
void setDepth(double d)
{
        this->depth=d;
}
void setWidth(double w)
{
        this->width=w;
}
char* getOrigin()
{
        return this->origin;
}
double getDepth()
{
        return this->depth;
}
double getWidth()
```

```
	{
		return this->width;
	}
	void display()
	{
		cout<<"\nWater source: River\nDetails:\n";
		WaterSource::display();
		cout<<"\nOrigin of river: "<<this->origin<<"\t  Depth: "<<this->depth<<"\t Width: "<<this->width<<"\n";
	}
	void giveWater()
	{
		cout<<"\nRiver water is open to us to use\n";
	}
};
int main()
{
	WaterSource *w;
	Dam d1;
	w=&d1;
	w->display();
	w->giveWater();
	Dam d2("koyana_dam",20,"Medium",36,98.77);
	w=&d2;
	w->display();
	w->giveWater();
```

```
        River r1;
        w=&r1;
        w->display();
        w->giveWater();
        River r2("koyana",32,"good","krishna_river",130,81);
        w=&r2;
        w->display();
        w->giveWater();
        return 0;
}
```