1) Demonstrate the usage of Constructor and Destructor. Define a class data with data member acct_no, balance containing constructor data to initiallize data member and a member function display for output.

```
#include<iostream>
using namespace std;
class data
{
	int acct_no;
	float balace;
	public:
		data(int a, float b)
		{
			acct_no=a;
			balace=b;
		}
		void display()
		{
			cout<<"Account Number="<<acct_no<<endl;
			cout<<"Balance="<<balace;
		}
};
int main()
{
	data d(10,100);
	d.display();
	return 0;
}
```

OUTPUT:

2) Program to demonstrate usage of a constructor and destructor function. Declare a class with public data member update information about active objects i.e.

i) no of objects created

ii) no of object destroyed

```
#include<iostream>
using namespace std;
class alpha
{
	public:
		static int count;
		alpha()
		{
			count++;
			cout<<endl<<" No. of object created=>"<<count;
		}
		~alpha()
		{
			cout<<endl<<"No. of Object Distroyed=>"<<count;
			count--;
		}

};
int alpha ::count;
int main()
{
	cout<<endl<<"Enter Main"<<endl;
	alpha A1,A2,A3,A4;

{
	cout<<endl<<"Enter Block 1";
	alpha A5;
}
{
	cout<<endl<<"Enter Block 2"<<endl;
	alpha A6;
}
cout<<endl<<"Re-Enter Main"<<endl;
return 0;
}
```

OTUPUT:

3) Program to accept the distance between city 1st & 2nd, city 2nd & 3rd. Calculate the distance between city 1st & 3rd. Define a class road with private data member km,m,d1,d2,d3 containing a member function getdata to accept values of d1,d2 and calculate for calculating distance

```
#include<iostream>
using namespace std;
class Road
{
	int km,m;
	public:
		void getdata()
		{
			cout<<"Enter kilometer => ";
			cin>>km;
			cout<<"Enter meter => ";
			cin>>m;
		}
		void calculate(Road d1,Road d2)
		{
			Road d3;
			d3.km=d1.km+d2.km;
			d3.m=d1.m+d2.m;
			if(d3.m>1000)
			{
				//1km=1000m
				d3.km=d3.km+1;
				d3.m=d3.m-1000;
			}
			cout<<d3.km<<"."<<d3.m<<"km";
		}
};
int main()
{
	Road d1,d2,d3;
	cout<<"Enter the Distance Between 1st and 2nd City:\n";
	d1.getdata();
	cout<<"\n\nEnter the Distance Between 2nd and 3rd City:\n";
	d2.getdata();
	cout<<"\n\nDistance between 1st and 3rd City => ";
	d3.calculate(d1,d2);
	return 0;
}
```

OUTPUT:

4) Demonstrate the use of operator overloading (string manipulation: + for concatenation and relational operators for alphabetical comparison

```
#include<iostream>
#include<string>
using namespace std;
class AddString
{
	public:
		char s1[25], s2[25];
		AddString(char str1[],char str2[])
		{
			strcpy(this->s1, str1);
			strcpy(this->s2, str2);
		}
		void operator+()
		{
			cout<<"\n Concatenation: "<<strcat(s1,s2);
		}
};
int main()
{
	char str1[] = "Object Oriented ";
	char str2[] = "Programming Language";
	AddString a1(str1,str2);
	+a1;
	return 0;
}
```

OUTPUT:

5) In a bank N depositor deposit the amount, write a program to find total amount deposited in the bank. Declare a class deposit with private data member Rupee and Paisa containing member function getdata, putdata

i) Use array of objects

ii) Use Operator '+' overloading

```
using namespace std;
class Deposit
{
	int rupee,paisa;
	public:
		Deposit()
		{
			rupee=0;
			paisa=0;
		}
		void getdata()
		{
			cout<<"\nEnter Rupee => ";
			cin>>rupee;
			cout<<"Enter Paisa => ";
			cin>>paisa;
		}
		void putdata()
		{
			cout<<rupee<<"."<<paisa;
		}
		Deposit operator+(Deposit d)
		{
			Deposit t;
			t.rupee=rupee+d.rupee;
			t.paisa=paisa+d.paisa;
			if(t.paisa>100)
			{
				t.rupee=t.rupee+1;
				t.paisa=t.paisa-100;
			}
			return t;
		}
};
int main()
{
	int i,n;
	Deposit oD[10],total;
	cout<<"Enter the Total Number of depositor N=>";
	cin>>n;
	for(i=0;i<n;i++)
	{
```

```
                cout<<"\n\nEnter the Deposited amount for "<<i+1<<" Depositor:";
                oD[i].getdata();
                cout<<"Balance of "<<i+1<<" Depositer =>";
                oD[i].putdata();
        }
        for(i=0;i<n;i++)
        {
                total=total+oD[i];
        }
        cout<<"\n\nTotal Amount Deposited in Bank is =>";
        total.putdata();
        return 0;
}
```

OUTPUT:

6) Declare class event and accept time of event and second event and find the difference between 1st and 2nd event. Containing public member function getdata and display with private data member hour, minute, second and total

i) Use Operator '-' overloading

```
#include<iostream>
using namespace std;
class Event
{
	int hours,minutes,seconds,total;
	public:
		Event()
		{
			hours=minutes=seconds=total=0;
		}
		void getdata()
		{
			cout<<"\nEnter Hours:";
			cin>>hours;
			cout<<"Enter Minutes:";
			cin>>minutes;
			cout<<"Enter Seconds:";
			cin>>seconds;
		}
		void putdata()
		{
			cout<<hours<<":"<<minutes<<":"<<seconds;
		}
		Event operator-(Event e2)
		{

			Event diff;
			diff.hours=e2.hours-hours;
			diff.minutes=e2.minutes-minutes;
			diff.seconds=e2.seconds-seconds;
			if(diff.minutes<0)
			{
				diff.hours=diff.hours-1;
				diff.minutes=diff.minutes+60;
			}
			if(diff.seconds<0)
			{
				diff.minutes=diff.minutes-1;
				diff.seconds=diff.seconds+60;
			}
			return diff;
		}
};
```

```
int main()
{


	Event oE1;
	cout<<"\nEnter the time [24 Hour Format] ]of 1st Event:";
	oE1.getdata();
	cout<<"1st Event Time => ";
	oE1.putdata();

	Event oE2;
	cout<<"\n\nEnter the time [24 Hour Format] of 2nd Event:";
	oE2.getdata();
	cout<<"2nd Event Time => ";
	oE2.putdata();

	Event diff;
	diff=oE1-oE2;
	cout<<"\n\nDifference Between Two Event => ";
	diff.putdata();

	return 0;
}
```

OUTPUT:

7) Program to demonstrate Single Inheritance. Declare a class B and derive publically class D from B.

i) The class B contains private data member a, public data member b with member function get_ab, get_a, show_a.

ii) The derived class D contains data member c with member function mul and display.

```
#include<iostream>
using namespace std;
class B
{
		int a;
	public:
		int b;
		void get_ab()
		{
			cout<<"\nEnter the value of A and B => ";
			cin>>a>>b;
		}
		int get_a()
		{
			return a;
		}
		void show_a()
		{
			cout<<"\nA => "<<a;
		}
};
class D:public B
{
	int c;
	public:
		void mul()
		{
			c=b*get_a();
		}
		void display()
		{
			cout<<"\n\nA => "<<get_a();
			cout<<"\nB => "<<b;
			cout<<"\nC => "<<c;
		}
};
int main()
{
	D d;
	d.get_ab();
	d.mul();
```

```
        d.show_a();
        d.display();

        d.b=20;
        d.mul();
        d.display();
        return 0;
}
```

OUTPUT:

8) Program to demonstrate Multiple Inheritances. Declare class M and N and derive publically class P from M and N.

i) Declare a class M with protected data member m and public member function get_m.

ii) Declare a class N with protected data member n containing member function get_n.

iii) Declare class P containing member function display.

```
#include<iostream>
using namespace std;
class M
{
	protected:
		int m;
	public:
		void get_m(int x)
		{
			m=x;
		}
};
class N
{
	protected:
		int n;
	public:
		void get_n(int y)
		{
			n=y;
		}
};
class P:public M,public N
{
	public:
		void display()
		{
			cout<<"\nm= "<<m;
			cout<<"\nn= "<<n;
			cout<<"\nm*n= "<<m*n;
		}
};
int main()
{
	P oP;
	oP.get_m(10);
	oP.get_n(20);
	oP.display();
	return 0;
}
```

OUTPUT:

9) Program to demonstrate Multilevel Inheritance. Declare a class student and derivepublically a class test and derive publically class result from class test.

i) The class student contains protected data member roll_number with publicmember functions get_number and put_number.

ii) The class test containing protected data member sub1, sub2 with public memberfunction get_marks and put_marks.

iii) The class result contains data member total and public member function display.

```
#include<iostream>
using namespace std;
class student
{
	protected:
		int roll_number;
	public:
		void get_number(int a)
		{
			roll_number=a;
		}
		void put_number()
		{
			cout<<"\nRoll Number: "<<roll_number;
		}
};
class test:public student
{
	protected:
		float sub1,sub2;
	public:
		void get_marks(float x,float y)
		{
			sub1=x;
			sub2=y;
		}
		void put_marks()
		{
			cout<<"\n Marks in Sub1 = "<<sub1;
			cout<<"\n Marks in Sub2 = "<<sub2;
		}
};
class result:public test
{
	float total;
	public:
		void display()
		{
			total=sub1+sub2;
			put_number();
```

```
                        put_marks();
                        cout<<"\n Total = "<<total;
                }
};
int main()
{
        result stud1;
        stud1.get_number(1);
        stud1.get_marks(75,59);
        stud1.display();
        return 0;

}
```

OUTPUT:

10) Program to demonstrate Hierarchical Inheritance. Declare a class Side and derivepublically class Square from base class side and also derive publically class cubefrom base class side.

i) Class Side contains protected data member L with a member function set_values.

ii) Class Square contains member function sq.

iii) Class Cube contains member function cub.

```
#include<iostream>
using namespace std;
class Side
{
	protected:
		int L;
	public:
		void set_values()
		{
			cout<<"\nEnter the number=>";
			cin>>L;
		}
};
class Square:public Side
{
	public:
		void sq()
		{
			cout<<"Square of "<<L<<" is "<<L*L;
		}
};
class Cube:public Side
{
	public:
		void cub()
		{
			cout<<"Cube of "<<L<<" is "<<L*L*L;
		}
};
int main()
{
	Square oS;
	oS.set_values();
	oS.sq();

	Cube oC;
	oC.set_values();
	oC.cub();
	return 0;
}
```

OUTPUT

11A) Program to demonstrate usage of normal virtual function

```
#include<iostream>
using namespace std;
class Base
{
	public:
		void display()
		{
			cout<<"\n Display Base";
		}
		virtual void show()
		{
			cout<<"\n Show Base";
		}
};
class Derive:public Base
{
	public:
		void display()
		{
			cout<<"\n Display Derive";
		}
		void show()
		{
			cout<<"\n Show Derive";
		}
};
int main()
{
		Base oB;
	Derive oD;
	Base *bptr;

	cout<<"\n bptr point to the Base";
	bptr=&oB;
	bptr->display();
	bptr->show();

	cout<<"\n\n bptr point to the Derive";
	bptr=&oD;
	bptr->display();
	bptr->show();
		return 0;
}
```

OUTPUT:

11B) Program to demonstrate usage pure virtual Function with abstract class.

```
#include<iostream>
using namespace std;
class Base
{
	public:
		void display()
		{
			cout<<"\n Display Base";
		}
		virtual void show()=0;
};
class Derive:public Base
{
	public:
		void display()
		{
			cout<<"\n Display Derive";
		}
		void show()
		{
			cout<<"\n Show Derive";
		}
};
int main()
{

	Derive oD;
	Base *bptr;
	cout<<"\n\n bptr point to the Derive";
	bptr=&oD;
	bptr->display();
	bptr->show();

	return 0;
}
```

OUTPUT:

12) Program to determine whether the input is +ve or –ve through exception.

```
#include<iostream>
using namespace std;
int main()
{
	int x;
	cout<<"Enter the Number=>";
	cin>>x;
	try
	{
		if(x==0)
			throw 'x';
		if (x>1)
			throw 1;
		if(x<0)
			throw -1.0;
	}
	catch(char c)
	{
		cout<<"Caught Zero Error";
	}
	catch(int i)
	{
		cout<<"Caught Positive Number";
	}
	catch(double d)
	{
		cout<<"Caught Negative Number";
	}
}
```

OUTPUT 1:

OUTPUT 2:

OUTPUT 3:

13) Program to raise exception if an attempt is made to perform divide-by-zero.

```
#include<iostream>
using namespace std;
int main()
{
	int a,b;
	cout<<"Enter the Two Number=>";
	cin>>a>>b;
	try
	{
		if(b!=0)
			cout<<a<<"/"<<b<<" = "<<a/b;
		else
			throw(b);
	}
	catch(int x)
	{
		cout<<"Exception Caught: Divide By Zero";
	}
}
```

OUTPUT 1:

OUTPUT 2: