Exercise -1(Classes Objects)

Creating objects of DISTANCE class

Aim: Create a Distance class with, feet and inches as data members , member function to input distance , member function to output distance, member function to add two distance objects.

Program: To write a main function to create objectsof DISTANCE class. Input two distances and output the sum

```
#include<iostream>
using namespace std;
class dist
{
	public:
	int feet,inch,x,y,z;
	void input()
	{
		cout<<"enter feet and inches:"<<"\n";
		cin>>feet>>inch;
	}
	void show()
	{
		cout<<"The distance is ";
		cout<<feet<<" feet "<<inch<<" inch\n";
	}
	void sum(dist x,dist y)
	{
		feet=x.feet+y.feet;
		inch=x.inch+y.inch;
		if(inch>=12)
		{
			feet=feet+1;
			inch=inch-12;
		}
	}
};
int main()
{
	dist x,y,z;
	x.input();
	y.input();
	z.sum(x,y);
	z.show();
}
```

Input and Output:

```
enter feet and inches:
3 8
enter feet and inches:
4 9
The distance is 8 feet 5 inch
```

Use of Constructors and Destructors

Aim: To write a C++ Program to illustrate the use of Constructors and Destructors (use the above program.)

Program:

```
#include  <iostream>
using namespace std;
class Distance
{
        private:
        int feet;
        int inches;
        public:
        Distance() {}
        Distance(int f, int i)
        {
                feet = f;
                inches = i;
        }
        void get_distance()
        {
                cout<<"Distance is feet= "<<feet<<", inches= "<<inches<<endl;
        }
        void add(Distance &d1, Distance &d2)
        {
                feet = d1.feet + d2.feet;
                inches = d1.inches + d2.inches;
                feet = feet + (inches / 12);
                inches = inches % 12;
        }
        ~Distance()
        {
                cout<<"Distance object destroyed"<<endl;
        }
      };
       int main()
       {
        int f1, in1, f2, in2;
        cout<<"Enter feet: ";cin>>f1;
        cout<<"Enter inches: ";
        cin>>in1;
        cout<<"Enter feet: ";
        cin>>f2;
        cout<<"Enter inches: ";
        cin>>in2;
        Distance d1(f1, in1);
        Distance d2(f2, in2);
        Distance d3;
        d3.add(d1, d2);
        d3.get_distance();
        return 0;
}
```

Input and Output:

Enter feet: 3
Enter inches: 8
Enter feet: 4
Enter inches: 9
Distance is feet= 8, inches= 5
Distance object destroyed
Distance object destroyed
Distance object destroyed

Function Overloading

Aim: To write a program for illustrating function overloading in adding the distance between objects (use the above problem)

Program:

```
#include  <iostream>
using namespace std;
class Distance
{
        private:
        int feet;
        int inches;
        public:
        void set_distance()
        {
                cout<<"Enter feet: ";
                cin>>feet;
                cout<<"Enter inches: ";
                cin>>inches;
        }
        void get_distance()
        {
                cout<<"Distance is feet= "<<feet<<", inches= "<<inches<<endl;
        }
        void add(Distance d1, Distance d2)
        {
                feet = d1.feet + d2.feet;
                inches = d1.inches + d2.inches;
                feet = feet + (inches / 12);
                inches = inches % 12;
        }
        void add(Distance *d1, Distance *d2)
        {
                feet = d1->feet + d2->feet;
                inches = d1->inches + d2->inches;
                feet = feet + (inches / 12);
                inches = inches % 12;
        }
};

int main()
```

```
{
        Distance d1, d2, d3;
        d1.set_distance();
        d2.set_distance();
        d3.add(d1, d2);
        d3.get_distance();
        d3.add(&d1, &d2);
        d3.get_distance();
        return 0;
}
```

Input and Output:

```
Enter feet: 3
Enter inches: 4
Enter feet: 4
Enter inches: 9
Distance is feet= 8, inches= 1
Distance is feet= 8, inches= 1
```

Exercise -2(Access)

2.Implementing Access Specifiers public, private, protected

Aim: To write a program for illustrating Access Specifiers public, private, protected

Program:

```
#include  <iostream>
using namespace std;
class A
{
        protected:
        int x;
        public:
        A(int p)
        {
                x = p;
        }
};
class B : public A
{
        private:
        int y;
        public:
        B(int p, int q) : A(p)
        {
                y = q;
        }
        void show()
        {
                cout<<"x = "<<x<<endl;
                cout<<"y = "<<y<<endl;
        }
};
int main()
{
        B obj(10, 20);
        //Since show is public in class B, it is accessible in main function
        obj.show(); //x is protected in A so it is accessible in B's show function
        //y is not accessible in main as it is private to class B
        //cout<<obj.y
        return 0;
}
```

Output:

```
x = 10
y = 20
```

Implementing Friend Function

Aim: To write a program implementing Friend Function

Program:

```
#include<iostream>
  using namespace std;
  class A
{
  privat: int x;
  public: A(int p)
          {
                  x = p;
          }
          friend void display(A &);
      };
  void display(A &obj)
  {
          cout<<"x = "<<obj.x;
  }
  int main()
  {
A obj(10);
display(obj);
return 0;
  }
```

Output:

x = 10

Illustrating this pointer

Aim: To write a program to illustrate this pointer

Program:

```
#include<iostream>
using namespace std;
class A
{
private:
  int x;
  int y;
 public:A(int x, int y)
          {
          this->x=x;
          this->y=y;
          }
          void display()
          {
                  cout<<"x="<<x<<endl;
                  cout<<"y = "<<y<<endl;
          }
          A& clone()
```

```
        {
                return *this;
        }
  };
  int main()
   {
        A obj1(10, 20);
        obj1.display();
        A obj2 = obj1.clone();
        obj2.display();
        return 0;
}
```

Output:

```
x = 10
y = 20
x = 10
y = 20
```

Illustrating pointer to a class

Aim: To write a Program to illustrate pointer to a class

Program:

```
#include<iostream>
Using  namespace std;
classA
        {
        private:
        int x;
        int y;
         public: A(int x, int y)

  this->x = x;
  this->y = y;
        }
        void display()
        {
                cout<<"x = "<<x<<endl;cout<<"y = "<<y<<endl;
        }
    };
int main()
{
        A *ptr = new A(10, 30);
         //Here ptr is pointer to class Aptr->display();
        return 0;
      }
```

Output:

```
x = 10
y = 30
```

Exercise-3(OperatorOverloading)

Overloading Unary, and Binary Operators

Aim: To write a program to Overload Unary, and Binary Operators as Member Function, and Non MemberFunction

1. Unary operator as member function

Program:

```
#include <iostream>
using namespace std;
class Number
   {
        private:
             int  x;
          public: Number(int p)
         {
           x = p;
           }
        void operator -()
        {
                x = -x;
        }
        void display()
        {
                cout<<"x = "<<x;
        }
   };
int  main()
       {
          Number n(10);
          -n;
          n.display();
          return 0;
      }
```

Output:

x = -10

2. Binary operator as non member function

Program:

```
#include<iostream>
using  namespace  std;
class Complex
    {
        private:
        float  real;
        float imag;
        public: Complex(){}
        Complex(float r, float i)
        {
                real = r;
                imag = i;
        }
```

```
        void display()
        {
                cout<<real<<"+i"<<imag;
        }
        friend Complex operator +(Complex &, Complex &);
 };
Complex operator +(Complex &c1, Complex &c2)
   {
     Complex temp;
     temp.real = c1.real + c2.real;
     temp.imag = c1.imag +
     c2.imag;
     return temp;
    }
 int main()
  {
     Complex c1(3, 4);
     Complex c2(4, 6);
     Complex c3 = c1+c2;
     c3.display();
     return 0;
 }
```

Output:

7+i10

2.Overloading assignment = Operator

Aim: To write a c ++ program to implement the overloading assignment = operator

Program:

```
#include <iostream>
using namespace std;
class Number
{
private:
        int x;
   public:Number(int p)
        {
                x = p;
        }
        Number operator =(Number &n)
        {
                return Number(n.x);
        }
        void display()
        {
                cout<<"x = "<<x;
        }
};
int main()
{
        Number  n1(10);
```

```
        Number  n2 =n1;
        n2.display();
        return 0;
}
```

Output:

x = 10

EXERCISE – 4(Inheritance)

A) Write a program to illustrate single inheritance.
B) Write a program to illustrate hierarchical inheritance.
C) Write a program to illustrate multiple inheritance.
D) Write a program to illustrate multilevel inheritance.

E) Write a program to illustrate hybrid inheritance.
F) Write a program to show Virtual Base Class.

i) AIM: Write a program to illustrate single inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class vehicle
{
   public:
      vehicle()
      {
         cout<<"\nThis is vehicle class.";
      }
};
class car:public vehicle
{
   public:
      car()
      {
         cout<<"\nThis is a car.";
      }
};
int main()
{
   car c;
}
```

OUTPUT:

```
This is vehicle class.
This is a car.
```

ii) AIM: Write a program to illustrate hierarchical inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class vehicle
{
   public:
      vehicle()
      {
         cout<<"\nVehicles are:";
      }
};
class bike:public vehicle
{
   public:
   bike()
   {
      cout<<"\nBike is a 2 wheeler vehicle.";
  }
};
class car:public vehicle
{
   public:
      car()
      {
         cout<<"\nCar is a 4 wheeler vehicle.";
      }
};
int main()
{
```

```
    bike b;
    car c;
}
```

OUTPUT:

```
Vehicles are:
Bike is a 2 wheeler vehicle.
Vehicles are:
Car is a 4 wheeler vehicle.
```

iii) AIM: Write a program to illustrate multiple inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class vehicle
{
    public:
        vehicle()
        {
            cout<<"\nThis is vehicle class.";
        }
};
class fourwheeler
{
    public:
        fourwheeler()
        {
            cout<<"\nThis is a 4 wheeler vehicle.";
        }
};
class car:public vehicle,fourwheeler
{
    public:
        car()
        {
            cout<<"\nThis is a car.";
        }
};
int main()
{
    car c;
}
```

OUTPUT:

```
This is vehicle class.
This is a 4 wheeler vehicle.
This is a car.
```

iv) AIM: Write a program to illustrate multilevel inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class vehicle
{
    public:
        vehicle()
        {
            cout<<"\nThis is vehicle class.";
        }
};
class fourwheeler:public vehicle
{
    public:
```

```
    fourwheeler()
    {
       cout<<"\nThis is a 4 wheeler vehicle.";
    }
};
class car:public fourwheeler
{
  public:
    car()
    {
       cout<<"\nThis is a car.";
    }
};
int main()
{
  car c;
}
```

OUTPUT:

```
This is vehicle class.
This is a 4 wheeler vehicle.
This is a car.
```

v) AIM: Write a program to illustrate hybrid inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class vehicle
{
  public:
    vehicle()
    {
       cout<<"\nThis is vehicle class.";
    }
};
class farevehicle
{
  public:
    farevehicle()
    {
       cout<<"\nFare vehicle.";
    }
};
class car:public vehicle
{
};
class bus:public vehicle,farevehicle
{
};
int main()
{
  car c;
  bus b;
}
```

OUTPUT:

```
This is vehicle class.
This is vehicle class.
Fare vehicle.
```

vi) AIM: Write a program to show Virtual BaseClass.

SOURCE CODE:

```
#include<iostream>
```

```
using namespace std;
class A
{
   protected:
      int a;
};
class B:virtual public A
{
   protected:
      int b;
};
class C:virtual public A
{
   protected:
      int c;
};
class D:public B,C
{
   protected:
      int d;
   public:
      void getdata()
      {
         cout<<"Enter a,b,c,d values:"<<endl;
         cin>>a>>b>>c>>d;
      }
      void putdata()
      {
         cout<<"Values are:"<<endl;
         cout<<"a="<<a<<"\nb="<<b<<"\nc="<<c<<"\nd="<<d;
      }
};
int main()
{
   D d;
   d.getdata();
   d.putdata();
}
```

OUTPUT:

```
Enter a,b,c,d values:
2
6
4
3
Values are:
a=2
b=6
c=4
d=3
```

2) Write a Program in C++ to illustrate the order of execution of constructors and destructors in inheritance.

AIM: Write a Program in C++ to illustrate the order of execution of constructors and destructors in inheritance.

SOURCE CODE:

```
#include<iostream>
using namespace std;
class BaseClass
{
   public:
```

```
    BaseClass()
    {
       cout<<"Base Class constructor called."<<endl;
    }
    ~BaseClass()
    {
       cout<<"Base class destructor called."<<endl;
    }
};
class DerivedClass:public BaseClass
{
   public:
    DerivedClass()
    {
       cout<<"Derived Class constructor called."<<endl;
    }
    ~DerivedClass()
    {
       cout<<"Derived class destructor called."<<endl;
    }
};
int main()
{
   DerivedClass obj;
}
```

OUTPUT:

Base Class constructor called.
Derived Class constructor called.
Derived class destructor called.
Base class destructor called.

EXERCISE -5(TEMPLATES, EXCEPTION HANDLING)

a) Illustrating template class

Aim: To write a C++ Program to illustrate template class

Program:

```
#include <iostream>
using namespace std;
template<class T>
```

```
class Swapper
{
        private:
         T x;
        T y;
        public:
        Swapper(T x, T y)
        {
                this->x = x;
                this->y = y;
        }
        void swap()
        {
                T temp = x;
                x = y;
                y = temp;
        }
        void display()
        {
                cout<<"After swap x = "<<x<<", y = "<<y<<endl;
        }
};
int main()
       {
        Swapper<int> s1(2, 4);
        s1.swap();
        s1.display(); Swapper<double>
        s2(4.2, 6.9);s2.swap();
        s2.display();
        return 0;
        }
```

Output:

After swap x = 4, y = 2 After
swap x = 6.9, y = 4.2

b) Member Function Templates

Aim: To write a Program to illustrate member function templates

Program:

```
#include<iostream>
using namespace std;
// template function
template <class T> T
Large(T n1, T n2)
{
        return (n1 > n2) ? n1 : n2;
}
int main()
{
         int i1, i2; float
        f1, f2;char c1,
        c2;
        cout << "Enter two integers:\n";cin >> i1 >>
        i2;
        cout << Large(i1, i2) <<" is larger." << endl;
```

```
        cout << "\nEnter two floating-point numbers:\n";cin >> f1 >>
        f2;
        cout << Large(f1, f2) <<" is larger." << endl;

        cout << "\nEnter two characters:\n";cin >> c1 >>
        c2;
        cout << Large(c1, c2) << " has larger ASCII value.";

        return 0;
}
```

Output:

```
Enter two integers:5
10
10 is larger.

Enter two floating-point numbers:12.4
10.2
12.4 is larger.

Enter two characters:z
Z
z has larger ASCII value
```

c)Exception Handling Divide by Zero

Aim: To write a Program for Exception Handling Divide by zero

Program:

```
#include <iostream>
#include <stdexcept>
using namespace std;
//handling divide by zero
float Division(float num, float den){
  if (den == 0) {
    throw runtime_error("Math error: Attempted to divide by Zero\n");
  }
  return (num / den);
}
int main(){
  float numerator, denominator, result;
  numerator = 12.5;
  denominator = 0;
  try {
    result = Division(numerator, denominator);
    cout << "The quotient is " << result << endl;
  }
  catch (runtime_error& e) {
```

```
    cout << "Exception occurred" << endl << e.what();
  }
}
```

Output:

Exception occurred

Math error: Attempted to divide by Zero

d)Rethrow an Exception

Aim: To write a Program to rethrow an Exception

Program:

```
#include <iostream>
using namespace std;
 int main()
{
   try {
      try {
         throw 20;
      }
      catch (int n) {
         cout << "Handle Partially ";
         throw; // Re-throwing an exception
      }
   }
   catch (int n) {
      cout << "Handle remaining ";
   }
   return 0;
}
```

OUTPUT:

Handle Partially Handle remaining

EXERCISE-6

1.Write C++ program illustratin user defined string processing functions using pointers.

a) String length:

```
int main() {
char str[20], *pt;
int i = 0;
cout &lt;&lt; &quot;Enter Any string [below 20 chars] : &quot;;
cin&gt;&gt;str;
// Assign to Pointer Variable
pt = str;
while (*pt != &#39;\0&#39;) {
i++;
pt++;
}
cout &lt;&lt; &quot;\nLength of String : &quot; &lt;&lt; i;
return 0;
}
```

Output:

```
Enter Any string [below 20 chars] : cpplab
Length of String : 6
```

b) String copy

```
int main()
{
string s1, s2;
cout &lt;&lt; &quot;Enter string s1: &quot;;
getline (cin, s1);
s2 = s1;
cout &lt;&lt; &quot;s1 = &quot;&lt;&lt; s1 &lt;&lt; endl;
cout &lt;&lt; &quot;s2 = &quot;&lt;&lt; s2;

return 0;
}
```

Output:

```
Enter string s1: wise
S1=wise
S2=wise
```

c) String concatenation

```
#include &lt;iostream&gt;
using namespace std;

int main()
{
string s1, s2, result;

cout &lt;&lt; &quot;Enter string s1: &quot;;
getline (cin, s1);

cout &lt;&lt; &quot;Enter string s2: &quot;;
getline (cin, s2);

result = s1 + s2;
```

```
cout &lt;&lt; &quot;Resultant String = &quot;&lt;&lt; result;

return 0;
}
```

Output:

```
Enter string s1: wise
Enter string s2: college
Resultant String =wisecollege
```

2. a) Write C++ program illustrating Virtual classes.

```
#include &lt;iostream&gt;
using namespace std;
class A {
public:
int a;
A(){
a = 10;
}
};
class B : public virtual A {
};
class C : public virtual A {
};
class D : public B, public C {
};
int main(){
//creating class D object
D object;
cout &lt;&lt; &quot;a = &quot; &lt;&lt; object.a &lt;&lt; endl;
return 0;
}
```

Output:

```
a = 10
```

b) Write C++ program illustrating Virtual functions

```
#include &lt;iostream&gt;
using namespace std;

class Base{
public:

virtual void Output(){
cout &lt;&lt; &quot;Output Base class&quot; &lt;&lt; endl;
}
void Display(){
cout &lt;&lt; &quot;Display Base class&quot; &lt;&lt; endl;
}
};
class Derived : public Base{
public:
void Output(){
cout &lt;&lt; &quot;Output Derived class&quot; &lt;&lt; endl;
}
void Display()
{
cout &lt;&lt; &quot;Display Derived class&quot; &lt;&lt; endl;
```

```
}
};

int main(){
Base* bpointr;
Derived dpointr;
bpointr = &amp;dpointr;
// virtual function binding
bpointr-&gt;Output();
// Non-virtual function binding
bpointr-&gt;Display();
}
```

Output:

Output Derived class
Display Base class

3. Write C++ program that implement bubble sort, to sort a given list of integers in ascending order.

```
#include&lt;iostream&gt;
using namespace std;
int main()
{
int n, i, arr[50], j, temp;
cout&lt;&lt;&quot;Enter the Size (max. 50): &quot;;
cin&gt;&gt;n;
cout&lt;&lt;&quot;Enter &quot;&lt;&lt;n&lt;&lt;&quot; Numbers: &quot;;
for(i=0; i&lt;n; i++)
cin&gt;&gt;arr[i];
cout&lt;&lt;&quot;\nSorting the Array using Bubble Sort Technique..\n&quot;;
for(i=0; i&lt;(n-1); i++)
{
for(j=0; j&lt;(n-i-1); j++)
{
if(arr[j]&gt;arr[j+1])
{
temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
}    }   }
cout&lt;&lt;&quot;          \nArray Sorted Successfully!\n&quot;;
cout&lt;&lt;&quot;          \nThe New Array is: \n&quot;;
for(i=0; i&lt;n; i++)
cout&lt;&lt;arr[i]&lt;&lt;&quot; &quot;;
cout&lt;&lt;endl;
return 0;
}
```

Output:

Enter the Size (max. 50): 5
Enter 5 Numbers: 7 8 9 5 2
Sorting the Array using Bubble Sort Technique..

Array Sorted Successfully!

The New Array is:
2 5 7 8 9