OOPS LAB
ASSIGNMENT – 4

NAME : Swarnendu Banerjee
ROLL : 002311001016
DEPARTMENT : IT – A1

Q36. Write a class “Point” which stores coordinates in (x, y) form. Define necessary constructor, destructor and other reader/writer functions. Now overload ‘-’ operator to calculate the distance between two points.

Code:

```
#include<bits/stdc++.h>
using namespace std;


class point{
int x,y;
public:
point(int x=0,int y=0){
 this->x=x;
 this->y=y;
 }

double  operator-(point&ob){
point t;
t.x=x-ob.x;
t.y=y-ob.y;
double dist=sqrt(t.x*t.x+t.y*t.y);
return dist;}
void disp(){
cout<<"x= "<<x;
cout<<"y= "<<y<<endl;}
};

int main(){
int x1,y1,x2,y2;
cout<<"enter the points :";
cin>>x1>>y1;
cout<<"enter the 2nd points :";
cin>>x2>>y2;
point obj1(x1,y1),obj2(x2,y2);
cout<<"distance between those points :"<<obj1-obj2<<endl;
return 0;}
```

Q37. Design a class Complex that includes all the necessary functions and operators like =, +, -, *, /.

Code:

```
    #include<bits/stdc++.h>

    using namespace std;

    class Complex {

        private:

            double real;

            double imag;


        public:

            Complex(double r = 0.0, double i = 0.0) : real(r),
    imag(i){}

            Complex& operator=(const Complex& other) {

                if (this != &other) {

                    real = other.real;

                    imag = other.imag;

                }
                return *this;
            }

            Complex operator+(const Complex& other) const {
                return Complex(real + other.real, imag +
    other.imag);
            }

            Complex operator-(const Complex& other) const {
                return Complex(real - other.real, imag -
    other.imag);
            }

            Complex operator*(const Complex& other) const {
                return Complex(real * other.real - imag *
    other.imag,
                        real * other.imag + imag *
    other.real);
            }

            Complex operator/(const Complex& other) const {
                double denominator = other.real * other.real +
    other.imag * other.imag;
```

```
            return Complex((real * other.real + imag * other.imag) / denominator,
                    (imag * other.real - real * other.imag) / denominator);
        }
        friend ostream& operator<<(ostream& os, const Complex& c) {
            os << c.real;
            if (c.imag >= 0) {
                os << " + " << c.imag << "i";
            } else {
                os << " - " << -c.imag << "i";
            }
            return os;
        }
};
int main() {
    double r1,i1,r2,i2;
    cout<<"enter the real and img part of the first no. :";
    cin>>r1>>i1;
    cout<<"enter the real and img part of the first no. :";
    cin>>r2>>i2;

    Complex c1(r1, i1);
```

```
        Complex c2(r2, i2);


        cout << "c1: " << c1 << endl;

        cout << "c2: " << c2 << endl;

        cout << "c1 + c2: " << (c1 + c2) << endl;

        cout << "c1 - c2: " << (c1 - c2) << endl;

        cout << "c1 * c2: " << (c1 * c2) << endl;

        cout << "c1 / c2: " << (c1 / c2) << endl;


        return 0;

    }
```

Q38. Implement a class “Quadratic” that represents second-degree polynomial i.e. polynomial of type ax2+bx+c. The class will require three data members corresponding to a, b and c. Implement the following:

a. A constructor (including a default constructor which create a null polynomial)

b. Overload the addition operator to add two polynomials of degree 2.

c. Overload << and >> operators to print and read polynomials.

d. A function to compute the value of polynomial for a given x.

e. A function to compute roots of the equation ax2+bx+c=0. Remember, root may be a complex

number. You may implement “Complex” class to represent root of the quadratic equation.

Code:

```
#include <bits/stdc++.h>
using namespace std;

class ComplexRoot {
   int a, b, c;
public:
   ComplexRoot(int a = 0, int b = 0, int c = 0) : a(a), b(b), c(c)
{}

   void root() {
      double real = -b / (2.0 * a);
      double img = sqrt(abs(b * b - 4 * a * c)) / (2.0 * a);
      cout << "Complex roots are: " << real << " + i" << img <<
" and " << real << " - i" << img << endl;
   }
};

class Quadratic {
   int a, b, c;
public:
   Quadratic(int a = 0, int b = 0, int c = 0) : a(a), b(b), c(c) {}

   Quadratic operator+(const Quadratic &ob) {
        return Quadratic(a + ob.a, b + ob.b, c + ob.c);
    }

    double valueAt(double x) {
        return a * x * x + b * x + c;
    }

    void root() {
        if ((b * b - 4 * a * c) < 0) {
            ComplexRoot(a, b, c).root();
        } else {
            double real = (-b + sqrt(b * b - 4 * a * c)) / (2.0 * a);
            double real2 = (-b - sqrt(b * b - 4 * a * c)) / (2.0 * a);
            cout << "Real roots are: " << real << " and " << real2 <<
endl;
        }
    }

    friend istream &operator>>(istream &is, Quadratic &ob) {
        cout << "Enter the coefficients of the quadratic a, b, c: ";
        is >> ob.a >> ob.b >> ob.c;
        return is;
    }
    friend ostream &operator<<(ostream &os, const Quadratic
&ob) {
        os << ob.a << "x^2 + " << ob.b << "x + " << ob.c;
        return os;
    }
};

int main() {
    Quadratic q1, q2, q3;
    cin >> q1 >> q2;
    cout << q1 << endl << q2 << endl;

    q3 = q1 + q2;
    cout << "Sum of quadratics: " << q3 << endl;

    cout << "Roots of q3: ";
    q3.root();

    return 0;
}
```

Q39. A program is given as follows:

```
class INT {
int i; public
:
```

```
INT(int a):i(a){}
~INT() {}
};
 int main()
{
int x = 3;
INT y = x;
 y++ =++y;
 x = y;
return 0;
}
```

Write extra functions/operators required in the INT class to make main program work. Provide suitable implementation for the added functions/operators.

Code:

```
#include <iostream>


using namespace std;


class INT {
    int i;


public:
    INT(int a) : i(a) {}
    INT(const INT& other) : i(other.i) {}

    INT& operator=(const INT& other) {
       i = other.i;
       return *this;
    }

    INT operator++() {
       ++i;
       return *this;
    }

    INT operator++(int) {
       INT temp = *this;
       ++i;
       return temp;
    }

    operator int() const {
       return i;
    }

    friend ostream& operator<<(ostream& os, const INT& obj) {
       os << obj.i;
```

```
        return os;
    }
};

int main() {
    int x = 3;
    INT y = x;
    y++ = ++y;
    x = y;
    cout << "x: " << x << ", y: " << y << endl;
    return 0;}
```

Q40. Design and implement class(es) to support the following main program.

```
int main() {
 IntArray i(10);
 for(int k = 0; k < 10;k++)
i[k] = k;
cout << i;
return 0;
}
```

<u>Code:</u>

```
#include <iostream>

using namespace std;
```

```
class IntArray {
private:
    int* arr;
    int size;

public:
    IntArray(int s) : size(s) {
        arr = new int[size];
    }

    ~IntArray() {
        delete[] arr;
    }

    int& operator[](int index) {
        return arr[index];
    }

    friend ostream& operator<<(ostream& os, const IntArray& intArray) {
        for (int k = 0; k < intArray.size; k++) {
            os << intArray.arr[k] << " ";
        }
        return os;
    }
};

int main() {
    IntArray i(10);
    for (int k = 0; k < 10; k++)
        i[k] = k;
    cout << i;
    return 0;
}
```

Q41. You are given a main program:

```
int main() {
Integer a = 4, b = a, c; c
= a+b++;
int i = a; cout << a
<< b << c; return 0;


}
```

Design and implement class(es) to support the main program.

Code:

```
#include <iostream>
using namespace std;
class Integer {
```

```
public:
    int value;
    Integer(int val=0) : value(val) {}
    Integer(const Integer& other) : value(other.value) {}
    Integer& operator=(const Integer& other) {
        if (this != &other) {
            value = other.value;
        }
        return *this;
    }
    Integer operator+(const Integer& other) {
        return Integer(this->value + other.value);
    }
    Integer operator++(int) {
        Integer temp = *this;
        value++;
        return temp;
    }
    friend ostream& operator<<( ostream& os, const Integer& obj) {
        os << obj.value;
        return os;
    }
};
int main() {
    Integer a = 4, b = a, c;
    c = a + b++;
    int i = a.value;
    cout << a << b << c;
    return 0;
}
```

Q42. Design and implement class(es) to support the following code segment.

Code:

```
Table t(4, 5), t1(4, 5);
cin >> t; t[0][0] = 5;
int x = t[2][3]; t1 = t;
cout << t << "\n" << t1;
```

```
#include <iostream>
#include <vector>
using namespace std;

class Table {
private:
    int rows;
    int cols;
    vector<vector<int> > data;
```

```
public:
    Table(int r, int c) : rows(r), cols(c), data(r, vector<int>(c, 0)) {}

    friend istream& operator>>(istream& is, Table& table) {
        for (int i = 0; i < table.rows; ++i)
            for (int j = 0; j < table.cols; ++j)
                is >> table.data[i][j];
        return is;
    }

    friend ostream& operator<<(ostream& os, const Table& table) {
        for (int i = 0; i < table.rows; ++i) {
            for (int j = 0; j < table.cols; ++j) {
                os << table.data[i][j] << " ";
            }
            os << "\n";
        }
        return os;
    }

    vector<int>& operator[](int index) {
        return data[index];
    }

    Table& operator=(const Table& other) {
        if (this != &other) {
            rows = other.rows;
            cols = other.cols;
            data = other.data;
        }
        return *this;
    }
};

int main() {
    Table t(4, 5), t1(4, 5);
    cin >> t;
    t[0][0] = 5;
    int x = t[2][3];
    t1 = t;
    cout << t << "\n" << t1;
    return 0;
}
```

Q43. Design and implement class(es) to support the following code segment.

```
        Index in(4), out(10);

        int x = in; int y = in

        + out; in = 2;

        Integer i; i

        = in;
```

Code:

```
#include <iostream>
using namespace std;

class Index {
private:
int value;
public:
Index(int v = 0) : value(v) {}
operator int() const {
return value;
}
Index operator+(const Index& other) const {
return Index(this->value + other.value);
}
Index& operator=(int v) {
value = v;
return *this;
}
Index& operator=(const Index& other) {
value = other.value;
return *this;
}
};
```

```
class Integer {
private:
int value;
public:
Integer() : value(0) {}
Integer& operator=(const Index& index) {
value = static_cast<int>(index);
return *this;
}
operator int() const {
return value;
}
};

int main() {
Index in(4), out(10);
int x = in;
int y = in + out;
in = 2;
Integer i;
i = in;

cout << "x: " << x << "\n";
cout << "y: " << y << "\n";
cout << "in: " << static_cast<int>(in) << "\n";
cout << "i: " << static_cast<int>(i) << "\n";

return 0;
}
```