OOPS ASSIGNMENT – 4

36. 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.

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

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

    ~Point()
    {
        cout<<"Point destroyed"<<endl;
    }


    void setX(double x)
    {     this->x=x;
    }

    void setY(double y)
    {     this->y=y;
    }


    double operator-(Point &ob)
    {
        double dx = x - ob.x;
        double dy = y - ob.y;

        return sqrt(dx*dx + dy*dy);
    }
};

int main() {
 Point p1(3.0, 4.0);
 Point p2(0.0, 0.0);

cout << "Distance: " << (p1 - p2) << endl;

 return 0;
```

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

```
#include <iostream>
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+( Complex& ob)  {
      return Complex(real + ob.real, imag + ob.imag);
   }

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

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

   Complex operator/( Complex& ob)  {
      double denominator = ob.real * ob.real + ob.imag * ob.imag;
      return Complex(
         (real * ob.real + imag * ob.imag) / denominator,
         (imag * ob.real - real * ob.imag) / denominator
      );
   }
```

```
    void show()  {
        cout << real;
        if (imag >= 0)
            cout << " + " << imag << "i" << endl;
        else
            cout << " - " << -imag << "i" << endl;
    }
};

int main() {
    double r1, i1, r2, i2;

    cout << "Enter real and imaginary part of first complex number: ";
    cin >> r1 >> i1;

    cout << "Enter real and imaginary part of second complex number: ";
    cin >> r2 >> i2;

    Complex a(r1, i1);
    Complex b(r2, i2);

    Complex sum = a + b;
    Complex difference = a - b;
    Complex product = a * b;
    Complex quotient = a / b;

    cout << "a: "; a.show();
    cout << "b: "; b.show();
    cout << "a + b: "; sum.show();
    cout << "a - b: "; difference.show();
    cout << "a * b: "; product.show();
    cout << "a / b: "; quotient.show();

    return 0;
}
```

38. 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.

```
#include <iostream>
#include <cmath>

using namespace std;

class Complex
{
private:
    double real, imag;

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

    friend ostream &operator<<(ostream &os, Complex &c)
    {
        if (c.imag >= 0)
            os << c.real << " + " << c.imag << "i";
        else
            os << c.real << " - " << -c.imag << "i";
        return os;
    }
};

class Quadratic
{
private:
    double a, b, c;

public:
    Quadratic(){
        a=0;
        b=0;
        c=0;
    }   Quadratic(double a, double b, double c){
        this->a=a;
        this->b=b;
        this->c=c;
    }
    Quadratic operator+(Quadratic &q)
    {
        return Quadratic(a + q.a, b + q.b, c + q.c);
    }

    friend ostream &operator<<(ostream &os, Quadratic &q)
    {
        os << q.a << "x^2 + " << q.b << "x + " << q.c;
        return os;
    }

    friend istream &operator>>(istream &is, Quadratic &q)
    {
        is >> q.a >> q.b >> q.c;
        return is;
    }

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

    void computeRoots()
    {
        double discriminant = b * b - 4 * a * c;
        if (discriminant > 0)
        {
                double root1 = (-b + sqrt(discriminant)) / (2 * a);
                double root2 = (-b - sqrt(discriminant)) / (2 * a);
                cout << "Real roots: " << root1 << " and " << root2 << endl;
        }
        else if (discriminant == 0)
        {
                double root = -b / (2 * a);
                cout << "One real root: " << root << endl;
        }
        else
        {
                Complex root1((-b) / (2 * a), sqrt(-discriminant) / (2 * a));
                 Complex root2((-b) / (2 * a), -sqrt(-discriminant) / (2 * a));
                 cout << "Complex roots: " << root1 << " and " << root2 << endl;
        }
    }
};

int main()
```

```
{
   Quadratic q1(1, -3, 2);
   Quadratic q2(1, 2, 1);
   Quadratic sum = q1 + q2;

   cout << "q1: " << q1 << endl;
   cout << "q2: " << q2 << endl;
   cout << "Sum: " << sum << endl;

   cout << "q1 evaluated at x=1: " << q1.evaluate(1) << endl;

   cout << "Roots of q1: ";
   q1.computeRoots();

   return 0;
}
```

39. 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.

```
#include <iostream>

using namespace std;

class INT {
private:
   int i;

public:
   INT(int a) : i(a) {}
```

```
        ~INT() {}

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

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

        INT& operator=( INT& other) {
           if (this != &other) {
              this->i = other.i;
           }
           return *this;
        }

        operator int()  {
           return i;
        }
     };


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

40. 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;
}
```

```
#include <iostream>

using namespace std;

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

public:
    IntArray(int size)
    {
        this->size = size;
        arr = new int[size];
    }

    ~IntArray()
    {
        delete[] arr;
    }

    int &operator[](int index)
    {
        if (index >= 0 && index < size)
        {
            return arr[index];
        }
        else
        {
            cout << "Index out of bounds";

        }
    }

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

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

    cout << i;
    return 0;
}
```

41. 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;

}
```

```
#include <iostream>
using namespace std;
class Integer
{
private:
   int val;

public:
   Integer(int val = 0)
   {
      this->val = val;
   }

   Integer(const Integer &ob1)
   {
      this->val = ob1.val;
   }

   Integer operator=(const Integer &ob1)
   {
      if (this != &ob1)
      {
         val = ob1.val;
      }
      return *this;
   }

   Integer operator++(int)
   {
      Integer temp = *this;
      val++;
      return temp;
   }

   Integer operator+(const Integer &ob1) const
   {
   {
      return Integer(val + ob1.val);
   }

   operator int()
   {
      return val;
   }

   friend ostream &operator<<(ostream &os, const Integer &ob2)
   {
      os << ob2.val;
      return os;
   }
};

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

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

```
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>
using namespace std;
```

```
class Table
{
    int rows, cols;
    int *data;

public:
    Table(int r, int c)
    {
        rows = r;
        cols = c;
        data = new int[r * c]();
    }

    ~Table()
    {
        delete[] data;
    }

    int *operator[](int i)
    {
        return data + i * cols;
    }

    Table &operator=(Table &ob)
    {
        if (this != &ob)
        {
            delete[] data;
            rows = ob.rows;
            cols = ob.cols;
            data = new int[rows * cols];
            for (int i = 0; i < rows * cols; ++i)
                data[i] = ob.data[i];
        }
        return *this;
    }

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

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

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;
}
```

43. 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;
```

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

class Index {
    int value;

public:
    Index(int v = 0) {
        value = v;
    }

    operator int() {
        return value;
    }

    Index& operator=(int v) {
        value = v;
        return *this;
    }
};

class Integer {
    int value;

    public:
    Integer(int v = 0) {
        value = v;
    }

    Integer& operator=(Index idx) {
        value = static_cast<int>(idx);
        return *this;
    }

    operator int() {
        return value;
    }
};
int main() {
    Index in(4), out(10);
    int x = in;
    int y = in + out;
    in = 2;
    Integer i;
    i = in;
}
```