OOPS ASSIGNMENT – 2

16. _Write a simple class that represents a class of geometrical points each of which has three coordinates. The class should have appropriate constructor(s). Also add a member function distance() that calculates Euclidian distance between two points. Now create two points, find the distance between them and print it.

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

class Point {
private:
   double x, y, z;

   public:
   Point(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

   double distance(const Point& other) const {
      return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2) + pow(z - other.z, 2));
   }
};

int main() {
   Point p1(1.0, 2.0, 3.0);
   Point p2(4.0, 5.0, 6.0);

    cout << "Distance between p1 and p2: " << p1.distance(p2) <<  endl;

   return 0;
}
```

17. Write a class for the geometrical shape rectangle. Write suitable constructors and member functions. Add a member function area() that calculates the area of a rectangle. Create 4 rectangles and print their respective area.

```
#include <iostream>
using namespace std;
class Rectangle {
private:
   double length, width;

public:
   Rectangle(double l = 1.0, double w = 1.0) : length(l), width(w) {}
```

```
    double area() const {
        return length * width;
    }
};

int main() {
    Rectangle r1(2.0, 3.0);
    Rectangle r2(4.0, 5.0);
    Rectangle r3(6.0, 7.0);
    Rectangle r4(8.0, 9.0);

     cout << "Area of r1: " << r1.area() <<  endl;
     cout << "Area of r2: " << r2.area() <<  endl;
     cout << "Area of r3: " << r3.area() <<  endl;
     cout << "Area of r4: " << r4.area() <<  endl;

    return 0;
}
```

18. Write a class that represents a class of wireless device. A device has a location (point object may be used), a fixed unique id, and a fixed circular transmission range. Write suitable constructors and member functions for this class. Instantiates 10 such devices. Choose location (coordinates) and transmission range of the devices randomly. Now, for each of these devices, find the neighbor devices (i.e. devices that belong to the transmission range). Suppose, all of these devices have moved to a new location (randomly chosen). Find out the new set of neighbors for each of these devices.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <cstdlib>
#include <ctime>

using namespace std;
class Point {
public:
    double x, y, z;

    Point(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    double distance(const Point& other) const {
        return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2) + pow(z - other.z, 2));
    }
};
```

```
class WirelessDevice {
private:
    static int idCounter;
    int id;
    Point location;
    double range;

public:
    WirelessDevice(double range = 10.0) : id(++idCounter), range(range) {
        location = Point(rand() % 100, rand() % 100, rand() % 100);
    }

    const Point& getLocation() const {
        return location;
    }

    double getRange() const {
        return range;
    }

    void move() {
        location = Point(rand() % 100, rand() % 100, rand() % 100);
    }

    bool isNeighbor(const WirelessDevice& other) const {
        return location.distance(other.getLocation()) <= range;
    }

    int getId() const {
        return id;
    }
};

int WirelessDevice::idCounter = 0;

int main() {
    srand(static_cast<unsigned int>(time(0)));
     vector<WirelessDevice> devices(10);

    for (int i = 0; i < devices.size(); ++i) {
         cout << "Device " << devices[i].getId() << " neighbors: ";
        for (int j = 0; j < devices.size(); ++j) {
            if (i != j && devices[i].isNeighbor(devices[j])) {
                cout << devices[j].getId() << " ";
            }
            }
            cout <<  endl;
        }

         cout << "\nAfter moving devices to new locations:\n";

        for (auto& device : devices) {
            device.move();
        }

        for (int i = 0; i < devices.size(); ++i) {
             cout << "Device " << devices[i].getId() << " neighbors: ";
            for (int j = 0; j < devices.size(); ++j) {
                if (i != j && devices[i].isNeighbor(devices[j])) {
                    cout << devices[j].getId() << " ";
                }
            }
            cout <<  endl;
        }

        return 0;
    }
```

19. Write a class Vector for one dimensional array. Write suitable constructor/copy constructor. Also add member functions for perform basic operations (such as addition, subtraction, equality, less, greater etc.). Create vectors and check if those operations are working correctly.

```
    #include <iostream>
    #include <vector>
    #include <algorithm>

    using namespace std;

    class Vector {
    private:
         vector<int> data;

    public:
        Vector() = default;

        Vector(const  vector<int>& vec) : data(vec) {}

        Vector operator+(const Vector& other) const {
             vector<int> result(data.size());
        transform(data.begin(), data.end(), other.data.begin(), result.begin(),  plus<int>());
       return Vector(result);
    }

    Vector operator-(const Vector& other) const {
        vector<int> result(data.size());
        transform(data.begin(), data.end(), other.data.begin(), result.begin(),  minus<int>());
       return Vector(result);
    }

    bool operator==(const Vector& other) const {
       return data == other.data;
    }

    bool operator<(const Vector& other) const {
       return data < other.data;
    }

    bool operator>(const Vector& other) const {
       return data > other.data;
    }

    void disp() const {
       for (const auto& element : data) {
           cout << element << " ";
       }
        cout <<  endl;
    }
};

int main() {
    Vector v1({1, 2, 3});
    Vector v2({4, 5, 6});

    Vector v3 = v1 + v2;
    Vector v4 = v2 - v1;
     cout << "v1: ";
    v1.disp();

     cout << "v2: ";
    v2.disp();

     cout << "v3 (v1 + v2): ";
    v3.disp();

     cout << "v4 (v2 - v1): ";
    v4.disp();

     cout << "v1 == v2: " << (v1 == v2) <<  endl;
     cout << "v1 < v2: " << (v1 < v2) <<  endl;
     cout << "v1 > v2: " << (v1 > v2) <<  endl;

    return 0;
}
```

20. Write a class IntArray for one dimensional integer array. Implement the necessary constructor, copy constructor, and destructor (if necessary) in this class. Implement other member functions to perform operations, such adding two arrays, reversing an array, sorting an array etc. Create an IntArray object having elements 1, 2 and 3 in it. Print its elements. Now, create another IntArray object which is an exact copy of the previous object. Print its elements. Now, reverse the elements of the last object. Finally print elements of both the objects.

```
#include <iostream>
#include <algorithm>

using namespace std;

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

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

    IntArray(const IntArray& other) : size(other.size) {
        arr = new int[size];
        copy(other.arr, other.arr + size, arr);
    }

    ~IntArray() {
        delete[] arr;
    }

    IntArray& operator=(const IntArray& other) {
        if (this == &other) return *this; // Handle self-assignment

        delete[] arr;
        size = other.size;
        arr = new int[size];
        copy(other.arr, other.arr + size, arr);
        return *this;
    }

    void setElements(const int* elements) {
        copy(elements, elements + size, arr);
    }

    IntArray operator+(const IntArray& other) const {
        if (size != other.size) {
            throw invalid_argument("Arrays must be of the same size");
        }
        IntArray result(size);
        for (int i = 0; i < size; ++i) {
            result.arr[i] = arr[i] + other.arr[i];
        }
        return result;
    }

    void reverseArray() {
        reverse(arr, arr + size);
    }

    void sortArray() {
        sort(arr, arr + size);
    }

    void display() const {
        for (int i = 0; i < size; ++i) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    int elements1[] = {1, 2, 3};
    IntArray arr1(3);
    arr1.setElements(elements1);

    cout << "arr1: ";
    arr1.display();

    IntArray arr2 = arr1;
    cout << "arr2 (copy of arr1): ";
    arr2.display();

    arr2.reverseArray();
    cout << "arr2 after reversing: ";
    arr2.display();

    cout << "arr1 (unchanged): ";
    arr1.display();

    return 0;
}
```

21. Create a simple class SavingsAccount for savings account used in banks as follows: Each SavingsAccount object should have three data members to store the account holder's name, unique account number and balance of the account. Assume account numbers are integers and generated sequentially. Note that once an account number is allocated to an account, it does not change during the entire operational period of the account. The bank also specifies a rate of interest for all savings accounts created. Write relevant methods (such as withdraw, deposit etc.) in the class. The bank restricts that each account must have a minimum balance of Rs. 1000. Now create 100 SavingsAccount objects specifying balance at random ranging from Rs. 1,000 to 1,00,000. Now, calculate the interest for one year to be paid to each account and deposit the interest to the corresponding balance. Also find out total amount of interest to be paid to all accounts in one year.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <numeric>

using namespace std;
class SavingsAccount {
private:
    static int accountCounter;
     string name;
    int accountNumber;
    double balance;
    static const double interestRate;

public:
    SavingsAccount( string name, double balance) : name(name),
accountNumber(++accountCounter), balance(balance) {}
```

```
    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (balance - amount >= 1000) {
            balance -= amount;
        } else {
             cout << "Cannot withdraw. Minimum balance required is 1000." <<  endl;
        }
    }

    void addInterest() {
        double interest = balance * interestRate;
        deposit(interest);
    }

    double getBalance() const {
        return balance;
    }

    static double totalInterestPaid(const  vector<SavingsAccount>& accounts) {
        return  accumulate(accounts.begin(), accounts.end(), 0.0, [](double sum, const
SavingsAccount& acc) {
            return sum + acc.balance * interestRate;
        });
    }
};

int SavingsAccount::accountCounter = 0;
const double SavingsAccount::interestRate = 0.04;

int main() {
    srand(static_cast<unsigned int>(time(0)));

     vector<SavingsAccount> accounts;
    for (int i = 0; i < 100; ++i) {
        accounts.push_back(SavingsAccount("Account" +  to_string(i+1), 1000 + rand() %
100000));
    }

    for (auto& account : accounts) {
        account.addInterest();
    }

    double totalInterest = SavingsAccount::totalInterestPaid(accounts);
```

```
     cout << "Total interest paid to all accounts: " << totalInterest <<  endl;

    return 0;
}
```

22. Write some programs to understand the notion of constant member functions, mutable data members etc.

```
#include <iostream>

using namespace std;

class Demo {
private:
    int a;
    mutable int b;

public:
    Demo(int x, int y) : a(x), b(y) {}

    int getA() const {
        return a;
    }

    int getB() const {
        return b;
    }

    void modifyB(int newValue) const {
        b = newValue;
    }
};

int main() {
    Demo obj(10, 20);

     cout << "a: " << obj.getA() <<  endl;
     cout << "b: " << obj.getB() <<  endl;

    obj.modifyB(30);
     cout << "b after modification: " << obj.getB() <<  endl;

    return 0;
}
```