Assignment 2

Name: Tanish Majumdar Roll No.: 002311001077

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 Euclidean distance between two points. Now create two points, find the distance between them, and print it.

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

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

    void calculateDistance(GeometricalPoint q)
    {
        double distance = sqrt(pow(x - q.x, 2) + pow(y - q.y, 2) + pow(z -
q.z, 2));
        cout << "Distance between the points is: " << distance << endl;
    }
};

int main()
{
    GeometricalPoint p(1, 2, 3);
    GeometricalPoint q(4, 5, 6);
    p.calculateDistance(q);
    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.

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

class Rectangle
{
public:
    int length, breadth;
    Rectangle(int length, int breadth)
    {
        this->length = length;
        this->breadth = breadth;
    }

    void calculateArea()
    {
        cout << "Area of the rectangle is: " << length * breadth << endl;
    }
};

int main()
{
    Rectangle r1(5, 10);
    Rectangle r2(10, 20);
    Rectangle r3(20, 30);
    Rectangle r4(30, 40);
    r1.calculateArea();
    r2.calculateArea();
    r3.calculateArea();
    r4.calculateArea();
    return 0;
}
```

18. Write a class that represents a class of `wireless devices`. 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. Instantiate 10 such devices. Choose location (coordinates) and

transmission range of the devices randomly. Now, for each of these devices, find the neighboring 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 <bits/stdc++.h>
using namespace std;

class Point
{
public:
    int x, y;

    Point() : x(0), y(0) {}

    Point(int x, int y)
    {
        this->x = x;
        this->y = y;
    }

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

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

public:
    WirelessDevice(int x, int y, double transmissionRange)
    {
        id = rand() % 1000;
        location = Point(x, y);
        this->transmissionRange = transmissionRange;
    }
    int getId() const
    {
        return id;
    }

    Point getLocation() const
    {
        return location;
    }

    void setLocation(int x, int y)
    {
        location = Point(x, y);
    }

    bool isInRange(const WirelessDevice &other) const
    {
        return location.distanceTo(other.getLocation()) <=
transmissionRange;
    }
};

void findNeighbors(const vector<WirelessDevice> &devices)
{
    for (const auto &device : devices)
    {
        cout << "Device " << device.getId() << " neighbors: ";
        for (const auto &other : devices)
        {
            if (device.getId() != other.getId() &&
device.isInRange(other))
            {
                cout << other.getId() << " ";
            }
        }
        cout << endl;
    }
}

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

    srand(time(0));

    for (int i = 0; i < 10; ++i)
    {
        int x = rand() % 100;
        int y = rand() % 100;
        double range = (rand() % 50) + 10;
        devices.push_back(WirelessDevice(x, y, range));
    }

    cout << "Initial neighbors:" << endl;
    findNeighbors(devices);

    for (auto &device : devices)
    {
        int newX = rand() % 100;
        int newY = rand() % 100;
        device.setLocation(newX, newY);
    }

    cout << "New neighbors after moving:" << endl;
    findNeighbors(devices);

    return 0;
}
```

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

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

class Vector
{
public:
    int *arr;
    int size;

    Vector(int size)
    {
        this->size = size;
        arr = new int[size];
    }

    Vector(const Vector &v) // Copy constructor
    {
        size = v.size;
        arr = new int[size];
        for (int i = 0; i < size; i++)
        {
```

```cpp
            arr[i] = v.arr[i];
        }
    }

    void add(Vector &v)
    {
        for (int i = 0; i < size; i++)
        {
            arr[i] += v.arr[i];
        }
    }

    void subtract(Vector &v)
    {
        for (int i = 0; i < size; i++)
        {
            arr[i] -= v.arr[i];
        }
    }

    bool isEqual(Vector &v)
    {
        for (int i = 0; i < size; i++)
        {
            if (arr[i] != v.arr[i])
                return false;
        }
        return true;
    }

    bool lessThan(Vector &v)
    {
        for (int i = 0; i < size; i++)
        {
            if (arr[i] >= v.arr[i])
                return false;
        }
        return true;
    }

    bool greaterThan(Vector &v)
    {
        for (int i = 0; i < size; i++)
        {
            if (arr[i] <= v.arr[i])
                return false;
        }
        return true;
    }
    void display()
    {
        for (int i = 0; i < size; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    ~Vector()
    {
        delete[] arr;
    }
};

int main()
{
    Vector v1(3);
    Vector v2(3);
    for (int i = 0; i < 3; i++)
    {
        v1.arr[i] = i;
        v2.arr[i] = i + 1;
    }
    v1.add(v2);
    v1.display();
    cout << (v1.isEqual(v2) ? "Vectors are equal" : "Vectors are not
equal") << endl;

    return 0;
}
```

20. Write a class `IntArray` for a 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 as adding two arrays, reversing an array, sorting an array, etc. Create an `IntArray` object with 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 objects.

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

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

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

    IntArray(const IntArray &a) // Copy constructor
    {
        size = a.size;
        arr = new int[size];
        for (int i = 0; i < size; i++)
        {
            arr[i] = a.arr[i];
        }
    }

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

    void add(IntArray &a)
    {
        for (int i = 0; i < size; i++)
        {
            arr[i] += a.arr[i];
        }
    }

    void reverse()
    {
        for (int i = 0; i < size / 2; i++)
        {
            int temp = arr[i];
            arr[i] = arr[size - i - 1];
            arr[size - i - 1] = temp;
        }
    }

    void sort()
    {
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    ~IntArray()
    {
        delete[] arr;
    }
};

int main()
{
    IntArray a(3);
    a.arr[0] = 1;
    a.arr[1] = 2;
    a.arr[2] = 3;
    cout << "Original Array: ";
    a.display();

    IntArray b = a; // Copying array
    cout << "Copied Array: ";
    b.display();

    b.reverse();
    cout << "Reversed Copied Array: ";
    b.display();

    cout << "Original Array after reversal of copied array: ";
    a.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 <bits/stdc++.h>
using namespace std;

#define INTEREST_RATE 0.05

class SavingsAccount
{
private:
    static int idCounter;
    int id;
    double balance;
    string name;

public:
    SavingsAccount(string name, double initialBalance)
    {
        id = ++idCounter;
        this->name = name;
        if (initialBalance < 1000)
        {
            cout << "Minimum balance should be 1000. Setting balance to
1000.\n";
            balance = 1000;
        }
        else
        {
            balance = initialBalance;
        }
    }
    void deposit(double amount)
    {
        balance += amount;
        cout << "Amount deposited successfully\n";
    }

    void withdraw(double amount)
    {
        if (balance - amount < 1000)
        {
            cout << "Insufficient balance\n";
            return;
        }
        balance -= amount;
        cout << "Amount withdrawn successfully\n";
    }

    void applyInterest()
    {
        balance += balance * INTEREST_RATE;
    }

    void printAccount() const
    {
        cout << "Account ID: " << id << ", Name: " << name << ", Balance: 
" << balance << endl;
    }

    double getBalance() { return balance; }
    string getName() { return name; }
};

int SavingsAccount::idCounter = 0;

int main()
{
    int n;
    cout << "Enter number of accounts: ";
    cin >> n;

    vector<SavingsAccount> accounts(n);

    srand(time(0));
    double totalInterest = 0;

    for (int i = 0; i < n; ++i)
    {
        double balance = (rand() % (100000 - 1000 + 1)) + 1000;
        string name = "Account " + to_string(i + 1);

        accounts[i] = SavingsAccount(name, balance);
        accounts[i].applyInterest();

        totalInterest += balance * INTEREST_RATE;
        accounts[i].printAccount();
        cout << "Interest to " << accounts[i].getName() << " is: " <<
accounts[i].getBalance() * INTEREST_RATE << endl;
    }

    cout << "Total interest: " << fixed << setprecision(2) <<
totalInterest << endl;

    return 0;
}
```

23. Write the definition for a class called `Complex` that has private floating point data members for storing real and imaginary parts. The class has the following public member functions:

`setReal() and setImg()` to set the real and imaginary part respectively. `getReal() and getImg()` to get the real and imaginary part respectively. `disp()` to display complex number object. `sum()` to sum two complex numbers & return a complex number. Write main function to create three complex number objects. Set the value in two objects and call `sum()` to calculate sum and assign it in third object. Display all complex numbers.

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

#define db double

class Complex
{
private:
    db real;
    db imaginary;

public:
    Complex(db real, db imaginary)
    {
        this->real = real;
        this->imaginary = imaginary;
    }

    void setReal(db real)
    {
        this->real = real;
    }
    db getReal()
    {
        return real;
    }
    void setImaginary(db imaginary)
    {
        this->imaginary = imaginary;
    }
    db getImaginary()
    {
        return imaginary;
    }
    void display()
    {
        cout << real << "+" << imaginary << "i" << endl;
    }
    Complex add(Complex &c)
    {
        Complex temp(real + c.real, imaginary + c.imaginary);
        return temp;
    }
};

int main()
{
    Complex c1(3, 4), c2(5, 6);
    Complex c3 = c1.add(c2);
    c3.display();
    return 0;
}
```

24. Complete the class with all function definitions for a `stack`

```
class Stack {
int *buffer, top;
public :
Stack(int); //create a stack with specified size
void push(int); //push the specified item
int pop(); //return the top element
```

```
    void disp(); //displays elements in the stack in top to bottom order

};
```

Now, create a stack with size 10, push 2, 3, 4 and 5 in that order and finally pop one element. Display elements present in the stack.

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

class Stack
{
    int *buffer;
    int top;

public:
    Stack(int size)
    {
        buffer = new int[size];
        top = -1;
    }

    void push(int element)
    {
        top++;
        buffer[top] = element;
    }

    int pop()
    {
        int x = buffer[top];
        top--;
        return x;
    }

    void display()
    {
        for (int i = 0; i <= top; i++)
        {
            cout << buffer[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Stack s(5);
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);
    s.display();
    s.pop();
    s.display();
    return 0;
}
```

25. Complete the class with all function definitions for a `circular queue`

```
class Queue {
int *data;
int front, rear;
public :
Queue(int ); //create queue with specified size
void add(int);//add specified element to the queue
int remove();//delete element from the queue

void disp(); //displays all elements in the queue(front to rear order)

};
```

Now, create a queue with size 10 add 2, 3, 4 and 5 in that order and finally delete two elements. Display elements present in the stack.

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

class Queue
{
    int *data;
    int front, rear, size;

public:
    Queue(int size)
    {
        data = new int[size];
        front = rear = -1;
        this->size = size;
    }
    void add(int element)
    {
        if (rear == size - 1)
        {
            cout << "Queue is full" << endl;
            return;
        }
        rear++;
        data[rear] = element;
    }
    int remove()
    {
        if (front == rear)
        {
            cout << "Queue is empty" << endl;
            return -1;
        }
        front++;
        return data[front];
    }
    void display()
    {
        for (int i = front + 1; i <= rear; i++)
        {
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Queue q(5);
    q.add(1);
    q.add(2);
    q.add(3);
    q.add(4);
    q.add(5);
    q.display();
    q.remove();
    q.display();
    return 0;
}
```

26. Write a class for your `Grade card`. The grade card is given to each student of a department per semester.

The grade card typically contains the name of the department, name of the student, roll number,semester, a list of subjects with their marks and a calculated CGPA. Create 60 such grade cards in a 3rd semester with relevant data and find the name and roll number of student having highest CGPA.

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

class GradeCard
{
public:
    int roll;
    string name;
    string dept;
    int semester;
    vector<int> marks;
    float cgpa;

    GradeCard(int roll, string name, string dept, int semester,
vector<int> marks, float cgpa)
    {
        this->roll = roll;
        this->name = name;
        this->dept = dept;
        this->semester = semester;
        this->marks = marks;
        this->cgpa = cgpa;
    }

    void display()
    {
        cout << "Roll: " << roll << endl;
        cout << "Name: " << name << endl;
        cout << "Department: " << dept << endl;
        cout << "Semester: " << semester << endl;
        cout << "Marks: ";
        for (int i = 0; i < marks.size(); i++)
        {
            cout << marks[i] << " ";
        }
        cout << endl;
        cout << "CGPA: " << cgpa << endl;
    }
};
```

```
int main()
{
    srand(time(0));
    vector<GradeCard> gradeCards;
    for (int i = 0; i < 60; i++)
    {
        vector<int> marks;
        for (int j = 0; j < 5; j++)
        {
            marks.push_back(rand() % 100);
        }
        float cgpa;
        int sum = 0;
        for (int i = 0; i < 5; i++)
        {
            sum += marks[i] / 10;
        }
        cgpa = sum / 5.0;
        gradeCards.push_back(GradeCard(i + 1, "Student " + to_string(i +
1), "IT", 3, marks, cgpa));
    }
    GradeCard max = gradeCards[0];
    for (int i = 1; i < gradeCards.size(); i++)
    {
        if (gradeCards[i].cgpa > max.cgpa)
        {
            max = gradeCards[i];
        }
    }
    max.display();
}
```

27. Create a class for `Book`. A book has unique isbn (string), title, a list of authors, and a price. Write relevant functions.

Now write a class BookStore which has a list of books. There may be multiple copies of a book in the book store. Write relevant member functions. Write a function `books()` that returns list of unique isbn numbers of the books, a function `noOfCopies()` that returns the number of copies available for a given isbn number and a function `totalPrice()` that returns the total price of all the books. Create a book store having a number of books (multiple

copies). Now, for each book, print number of copies of that book along with its title.

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

class Book
{
public:
    string title;
    string isbn;
    int price;
    vector<string> authors;

    Book(string title, string isbn, int price, vector<string> authors)
    {
        this->title = title;
        this->isbn = isbn;
        this->price = price;
        this->authors = authors;
    }
};

class BookStore
{
public:
    vector<Book> books;
    vector<int> copies;

    BookStore(vector<Book> books, vector<int> copies)
    {
        this->books = books;
        this->copies = copies;
    }
};

set<string> getISBN(BookStore store)
{
    set<string> isbns;
    for (int i = 0; i < store.books.size(); i++)
    {
        isbns.insert(store.books[i].isbn);
    }
    return isbns;
}
```

```cpp
int noOfCopies(BookStore store, string isbn)
{
    for (int i = 0; i < store.books.size(); i++)
    {
        if (store.books[i].isbn == isbn)
        {
            return store.copies[i];
        }
    }
    return 0;
}

int totalCost(BookStore store)
{
    int total = 0;
    for (int i = 0; i < store.books.size(); i++)
    {
        total += store.books[i].price * store.copies[i];
    }
    return total;
}

int main()
{
    vector<string> authors1 = {"Author1", "Author2"};
    vector<string> authors2 = {"Author3", "Author4"};
    vector<string> authors3 = {"Author5", "Author6"};
    vector<string> authors4 = {"Author7", "Author8"};
    vector<string> authors5 = {"Author9", "Author10"};

    Book book1("Book1", "ISBN1", 100, authors1);
    Book book2("Book2", "ISBN2", 200, authors2);
    Book book3("Book3", "ISBN3", 300, authors3);
    Book book4("Book4", "ISBN4", 400, authors4);
    Book book5("Book5", "ISBN5", 500, authors5);

    vector<Book> books = {book1, book2, book3, book4, book5};
    vector<int> copies = {1, 2, 3, 4, 5};

    BookStore store(books, copies);

    set<string> isbns = getISBN(store);
    cout << "ISBNs: ";
    for (auto isbn : isbns)
    {
        cout << isbn << " ";
    }
    cout << endl;

    cout << "No of copies of ISBN2: " << noOfCopies(store, "ISBN2") <<
endl;

    cout << "Total cost: " << totalCost(store) << endl;

    return 0;
}
```