OOP Lab

Name: **Soham Das**

Section: **A1**

Roll No: **002311001004**

Assignment - 2

IT-UG2

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 <math.h>
using namespace std;

class Point3D
{
private:
    double x;
    double y;
    double z;

public:
    Point3D();
    Point3D(double x, double y, double z);
    double distance(Point3D other);
};

Point3D::Point3D()
{
    this->x = 0.0;
    this->y = 0.0;
    this->z = 0.0;
}

Point3D::Point3D(double x, double y, double z)
{
    this->x = x;
    this->y = y;
    this->z = z;
}

double Point3D::distance(Point3D other)
{
    return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2) + pow(z - other.z,
2));
}

int main()
```

```
{
    Point3D p1(1, 2, 3);
    Point3D p2(3, 6, 9);

    cout << "Distance = " << p1.distance(p2) << "\n";

    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;
    double width;

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

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

int main()
{
    Rectangle rect1(4, 5);
    Rectangle rect2(3, 6);
    Rectangle rect3(2, 8);
```

```
    Rectangle rect4(7, 2);

    cout << "Area of Rectangle 1: " << rect1.area() << endl;
    cout << "Area of Rectangle 2: " << rect2.area() << endl;
    cout << "Area of Rectangle 3: " << rect3.area() << endl;
    cout << "Area of Rectangle 4: " << rect4.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;

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

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

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

public:
    WirelessDevice(int id, const Point &loc, double range)
        : id(id), location(loc), transmissionRange(range) {}

    int getId() const
    {
        return id;
    }

    Point getLocation() const
    {
        return location;
    }

    double getTransmissionRange() const
    {
        return transmissionRange;
    }

    void setLocation(const Point &loc)
    {
        location = loc;
    }

    vector<int> findNeighbors(const vector<WirelessDevice> &devices) const
    {
        vector<int> neighbors;
        for (const auto &device : devices)
        {
            if (device.getId() != id &&
                location.distanceTo(device.getLocation()) <= transmissionRange)
            {
                neighbors.push_back(device.getId());
            }
        }
        return neighbors;
```

```
    }
};

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

    const int numDevices = 10;
    vector<WirelessDevice> devices;

    // Instantiate 10 devices with random locations and transmission ranges
    for (int i = 0; i < numDevices; ++i)
    {
        double x = rand() % 100;         // Random x-coordinate between 0 and
99
        double y = rand() % 100;         // Random y-coordinate between 0 and
99
        double range = 10 + rand() % 20; // Random range between 10 and 29
        devices.emplace_back(i, Point(x, y), range);
    }

    // Find neighbors for each device in the initial location
    cout << "Initial neighbors:\n";
    for (const auto &device : devices)
    {
        vector<int> neighbors = device.findNeighbors(devices);
        cout << "Device " << device.getId() << " has neighbors: ";
        for (int neighborId : neighbors)
        {
            cout << neighborId << " ";
        }
        cout << endl;
    }

    // Move devices to new random locations
    for (auto &device : devices)
    {
        double newX = rand() % 100; // Random x-coordinate between 0 and 99
        double newY = rand() % 100; // Random y-coordinate between 0 and 99
        device.setLocation(Point(newX, newY));
    }

    // Find neighbors for each device in the new location
```

```
    cout << "\nNew neighbors after moving:\n";
    for (const auto &device : devices)
    {
        vector<int> neighbors = device.findNeighbors(devices);
        cout << "Device " << device.getId() << " has neighbors: ";
        for (int neighborId : neighbors)
        {
            cout << neighborId << " ";
        }
        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>
using namespace std;

class Vector
{
private:
    vector<int> data;

public:
    Vector(int size, int val = 0) : data(size, val) {}

    Vector(const Vector &other) : data(other.data) {}

    Vector operator+(const Vector &other) const
    {
        Vector result(data.size());
        for (size_t i = 0; i < data.size(); i++)
            result.data[i] = data[i] + other.data[i];
        return result;
    }
```

```
Vector operator-(const Vector &other) const
{
    Vector result(data.size());
    for (size_t i = 0; i < data.size(); i++)
        result.data[i] = data[i] - other.data[i];
    return result;
}

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

bool operator<(const Vector &other) const
{
    for (size_t i = 0; i < data.size(); i++)
    {
        if (data[i] < other.data[i])
            return true;
        if (data[i] > other.data[i])
            return false;
    }
    return false;
}

bool operator>(const Vector &other) const
{
    for (size_t i = 0; i < data.size(); i++)
    {
        if (data[i] > other.data[i])
            return true;
        if (data[i] < other.data[i])
            return false;
    }
    return false;
}

void print() const
{
    for (int val : data)
        cout << val << " ";
    cout << endl;
```

```
    }
};

int main()
{
    Vector v1(3, 1), v2(3, 2);

    Vector v3 = v1 + v2;
    Vector v4 = v2 - v1;

    cout << "v1 + v2: ";
    v3.print();
    cout << "v2 - v1: ";
    v4.print();

    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>

using namespace std;

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

public:
    // Constructor
```

```
    IntArray(int size) : size(size)
    {
        data = new int[size];
    }

    // Constructor with initializer list
    IntArray(initializer_list<int> list) : size(list.size())
    {
        data = new int[size];
        int index = 0;
        for (int value : list)
        {
            data[index++] = value;
        }
    }

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

    // Destructor
    ~IntArray()
    {
        delete[] data;
    }

    // Get size of the array
    int getSize() const
    {
        return size;
    }

    // Overload the [] operator for array access
    int &operator[](int index)
    {
        if (index < 0 || index >= size)
        {
```

```
            throw out_of_range("Index out of range");
        }
        return data[index];
    }

    const int &operator[](int index) const
    {
        if (index < 0 || index >= size)
        {
            throw out_of_range("Index out of range");
        }
        return data[index];
    }

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

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

    // Sort the array (Bubble Sort)
    void sort()
```

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

    // Print the elements of the array
    void print() const
    {
        for (int i = 0; i < size; ++i)
        {
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    // Create an IntArray object with elements 1, 2, 3
    IntArray arr1 = {1, 2, 3};
    cout << "Elements of arr1: ";
    arr1.print();

    // Create a copy of arr1
    IntArray arr2 = arr1;
    cout << "Elements of arr2 (copy of arr1): ";
    arr2.print();

    // Reverse the elements of arr2
    arr2.reverse();
    cout << "Elements of arr2 after reversing: ";
    arr2.print();
```

```
    // Print elements of arr1 to ensure it is unchanged
    cout << "Elements of arr1 after reversing arr2: ";
    arr1.print();

    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 <string>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

class SavingsAccount
{
private:
    string accountHolderName;
    int accountNumber;
    double balance;
    static double interestRate;
    static int accountCounter;

public:
    SavingsAccount(string name, double initialBalance)
        : accountHolderName(name), accountNumber(++accountCounter)
    {
        if (initialBalance < 1000)
```

```
        {
            cerr << "Initial balance must be at least Rs. 1000. Setting balance
to Rs. 1000.\n";
            balance = 1000;
        }
        else
        {
            balance = initialBalance;
        }
    }

    void deposit(double amount)
    {
        if (amount > 0)
        {
            balance += amount;
        }
        else
        {
            cerr << "Deposit amount must be positive.\n";
        }
    }

    void withdraw(double amount)
    {
        if (amount > 0)
        {
            if (balance - amount >= 1000)
            {
                balance -= amount;
            }
            else
            {
                cerr << "Cannot withdraw. Minimum balance of Rs. 1000 must be
maintained.\n";
            }
        }
        else
        {
            cerr << "Withdrawal amount must be positive.\n";
        }
    }
```

```
    double calculateInterest() const
    {
        return balance * interestRate / 100;
    }

    void addInterest()
    {
        balance += calculateInterest();
    }

    double getBalance() const
    {
        return balance;
    }

    static double getInterestRate()
    {
        return interestRate;
    }

    static void setInterestRate(double rate)
    {
        if (rate >= 0)
        {
            interestRate = rate;
        }
        else
        {
            cerr << "Interest rate must be non-negative.\n";
        }
    }

    int getAccountNumber() const
    {
        return accountNumber;
    }
};

double SavingsAccount::interestRate = 3.5;
int SavingsAccount::accountCounter = 0;

int main()
{
```

```
    const int NUM_ACCOUNTS = 100;
    vector<SavingsAccount> accounts;
    srand(time(0));

    for (int i = 0; i < NUM_ACCOUNTS; ++i)
    {
        double initialBalance = 1000 + rand() % 99001;
        accounts.emplace_back("AccountHolder" + to_string(i + 1),
initialBalance);
    }

    double totalInterest = 0;
    for (auto &account : accounts)
    {
        double interest = account.calculateInterest();
        totalInterest += interest;
        account.addInterest();
    }

    cout << "Total interest paid to all accounts in one year: Rs. " <<
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 regularVar;
    mutable int mutableVar;

public:
    Demo(int r, int m) : regularVar(r), mutableVar(m) {}

    void setRegularVar(int r) { regularVar = r; }
```

```
    void setMutableVar(int m) const { mutableVar = m; }

    int getRegularVar() const { return regularVar; }

    int getMutableVar() const { return mutableVar; }
};

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

    cout << "Initial regularVar: " << d.getRegularVar() << endl;
    cout << "Initial mutableVar: " << d.getMutableVar() << endl;

    d.setRegularVar(15);
    cout << "Updated regularVar: " << d.getRegularVar() << endl;

    d.setMutableVar(25);
    cout << "Updated mutableVar: " << d.getMutableVar() << 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;
```

```
class Complex
{
private:
    float real;
    float img;

public:
    void setReal(float r)
    {
        real = r;
    }

    void setImg(float i)
    {
        img = i;
    }

    float getReal() const
    {
        return real;
    }

    float getImg() const
    {
        return img;
    }

    void disp() const
    {
        cout << real << " + " << img << "i" << endl;
    }

    Complex sum(const Complex &c) const
    {
        Complex result;
        result.real = real + c.real;
        result.img = img + c.img;
        return result;
    }
};

int main()
{
```

```
    Complex c1, c2, c3;

    c1.setReal(3.2);
    c1.setImg(4.5);

    c2.setReal(1.8);
    c2.setImg(2.3);

    c3 = c1.sum(c2);

    cout << "Complex number 1: ";
    c1.disp();

    cout << "Complex number 2: ";
    c2.disp();

    cout << "Sum of complex numbers: ";
    c3.disp();

    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
{
private:
    int *buffer;
    int top;
    int size;

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

    void push(int item)
    {
        if (top < size - 1)
        {
            buffer[++top] = item;
        }
        else
        {
            cout << "Stack overflow, unable to push " << item << endl;
        }
    }

    int pop()
    {
        if (top >= 0)
        {
            return buffer[top--];
        }
        else
        {
            cout << "Stack underflow, no elements to pop" << endl;
            return -1;
        }
    }

    void disp()
    {
```

```
        if (top >= 0)
        {
            cout << "Stack elements (top to bottom): ";
            for (int i = top; i >= 0; --i)
            {
                cout << buffer[i] << " ";
            }
            cout << endl;
        }
        else
        {
            cout << "Stack is empty" << endl;
        }
    }

    ~Stack()
    {
        delete[] buffer;
    }
};

int main()
{
    Stack s(10);

    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);

    s.pop();

    s.disp();

    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
{
private:
    int *data;
    int front, rear;
    int size;

public:
    Queue(int s) : size(s), front(-1), rear(-1)
    {
        data = new int[size];
    }

    void add(int item)
    {
        if ((rear + 1) % size == front)
        {
            cout << "Queue overflow, unable to add " << item << endl;
        }
        else
        {
            if (front == -1)
                front = 0;
```

```
            rear = (rear + 1) % size;
            data[rear] = item;
        }
    }

    int remove()
    {
        if (front == -1)
        {
            cout << "Queue underflow, no elements to remove" << endl;
            return -1;
        }
        else
        {
            int removedItem = data[front];
            if (front == rear)
            {
                front = -1;
                rear = -1;
            }
            else
            {
                front = (front + 1) % size;
            }
            return removedItem;
        }
    }

    void disp()
    {
        if (front == -1)
        {
            cout << "Queue is empty" << endl;
        }
        else
        {
            cout << "Queue elements (front to rear): ";
            int i = front;
            while (true)
            {
                cout << data[i] << " ";
                if (i == rear)
                    break;
```

```
                i = (i + 1) % size;
            }
            cout << endl;
        }
    }

    ~Queue()
    {
        delete[] data;
    }
};

int main()
{
    Queue q(10);

    q.add(2);
    q.add(3);
    q.add(4);
    q.add(5);

    q.remove();
    q.remove();

    q.disp();

    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 <string>
#include <cstdlib>

using namespace std;
```

```
class GradeCard
{
private:
    string department;
    string studentName;
    string rollNumber;
    int semester;
    string subjects[5];
    double marks[5];
    double CGPA;

    double calculateCGPA()
    {
        double totalMarks = 0;
        for (int i = 0; i < 5; ++i)
        {
            totalMarks += marks[i];
        }
        return totalMarks / 5;
    }

public:
    GradeCard(string dept, string name, string roll, int sem, string subs[5],
double mks[5])
        : department(dept), studentName(name), rollNumber(roll), semester(sem)
    {
        for (int i = 0; i < 5; ++i)
        {
            subjects[i] = subs[i];
            marks[i] = mks[i];
        }
        CGPA = calculateCGPA();
    }

    double getCGPA() const
    {
        return CGPA;
    }

    string getStudentName() const
    {
        return studentName;
```

```
    }

    string getRollNumber() const
    {
        return rollNumber;
    }

    void disp() const
    {
        cout << "Department: " << department << endl;
        cout << "Student Name: " << studentName << endl;
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Semester: " << semester << endl;
        cout << "Subjects and Marks: " << endl;
        for (int i = 0; i < 5; ++i)
        {
            cout << subjects[i] << ": " << marks[i] << endl;
        }
        cout << "CGPA: " << CGPA << endl;
    }
};

int main()
{
    const int numStudents = 60;
    GradeCard *gradeCards[numStudents];
    string department = "Computer Science";
    int semester = 3;
    string subjects[5] = {"Math", "Physics", "Chemistry", "Computer Science",
"Electronics"};

    for (int i = 0; i < numStudents; ++i)
    {
        string studentName = "Student" + to_string(i + 1);
        string rollNumber = "CS2023" + to_string(100 + i);
        double marks[5];
        for (int j = 0; j < 5; ++j)
        {
            marks[j] = rand() % 101;
        }
        gradeCards[i] = new GradeCard(department, studentName, rollNumber,
semester, subjects, marks);
    }
```

```
    GradeCard *highestCGPAStudent = gradeCards[0];
    for (int i = 1; i < numStudents; ++i)
    {
        if (gradeCards[i]->getCGPA() > highestCGPAStudent->getCGPA())
        {
            highestCGPAStudent = gradeCards[i];
        }
    }

    cout << "Student with the highest CGPA:\n";
    cout << "Name: " << highestCGPAStudent->getStudentName() << endl;
    cout << "Roll Number: " << highestCGPAStudent->getRollNumber() << endl;
    cout << "CGPA: " << highestCGPAStudent->getCGPA() << endl;

    for (int i = 0; i < numStudents; ++i)
    {
        delete gradeCards[i];
    }

    return 0;
}
```

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 <string>
#include <unordered_map>
#include <vector>
using namespace std;

class Book
{
private:
```

```
    string isbn;
    string title;
    vector<string> authors;
    double price;

public:
    Book(string isbn, string title, vector<string> authors, double price)
        : isbn(isbn), title(title), authors(authors), price(price) {}

    string getIsbn() const { return isbn; }
    string getTitle() const { return title; }
    double getPrice() const { return price; }
};

class BookStore
{
private:
    unordered_map<string, pair<Book, int>> books; // isbn -> (Book, number of
copies)

public:
    void addBook(const Book &book, int copies)
    {
        books[book.getIsbn()] = make_pair(book, copies);
    }

    vector<string> getBooks() const
    {
        vector<string> isbnList;
        for (const auto &entry : books)
            isbnList.push_back(entry.first);
        return isbnList;
    }

    int noOfCopies(const string &isbn) const
    {
        if (books.find(isbn) != books.end())
            return books.at(isbn).second;
        return 0;
    }

    double totalPrice() const
    {
```

```
        double total = 0;
        for (const auto &entry : books)
            total += entry.second.first.getPrice() * entry.second.second;
        return total;
    }

    void printBookCopies() const
    {
        for (const auto &entry : books)
        {
            const Book &book = entry.second.first;
            cout << "Title: " << book.getTitle() << ", Copies: " <<
entry.second.second << endl;
        }
    }
};

int main()
{
    Book b1("12345", "C++ Programming", {"Author A", "Author B"}, 500.0);
    Book b2("67890", "Data Structures", {"Author C", "Author D"}, 300.0);

    BookStore store;
    store.addBook(b1, 5);
    store.addBook(b2, 3);

    cout << "Books in the store: " << endl;
    store.printBookCopies();

    cout << "Total price of all books in store: " << store.totalPrice() <<
endl;

    string isbnToCheck = "12345";
    cout << "No. of copies of book with ISBN " << isbnToCheck << ": " <<
store.noOfCopies(isbnToCheck) << endl;

    return 0;
}
```