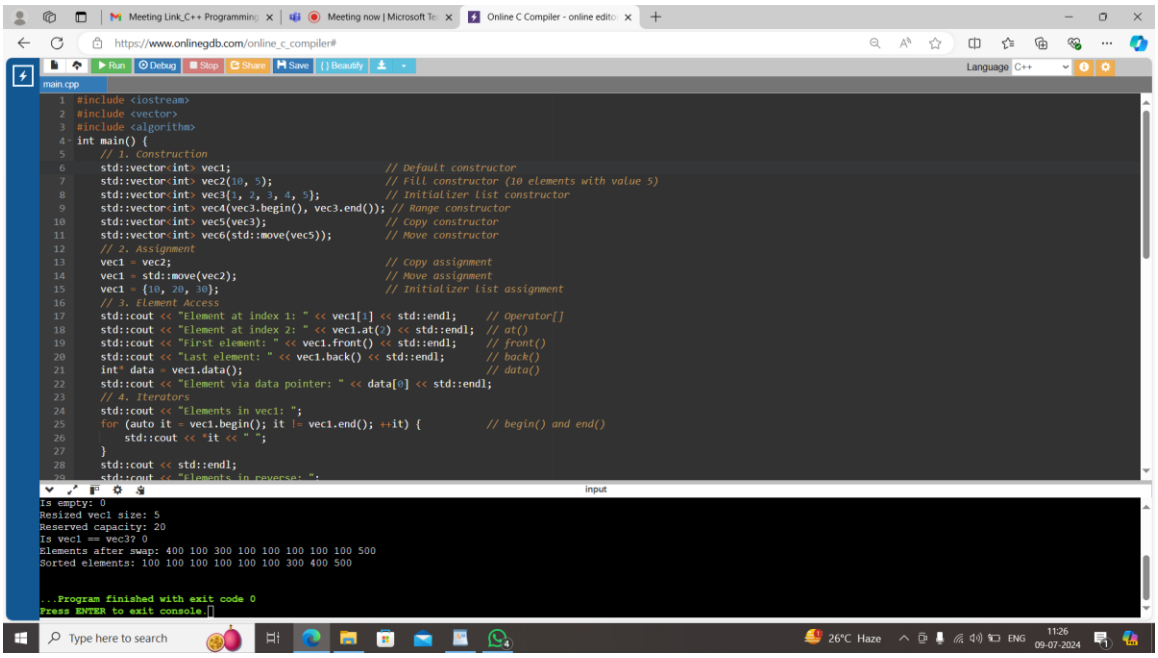
VECTOR :-

```
#include <iostream>

#include <vector>

#include <algorithm>

int main() {

    // 1. Construction

    std::vector<int> vec1;                                  // Default constructor

    std::vector<int> vec2(10, 5);                           // Fill constructor (10 elements with value 5)

    std::vector<int> vec3{1, 2, 3, 4, 5};                   // Initializer list constructor

    std::vector<int> vec4(vec3.begin(), vec3.end()); // Range constructor

    std::vector<int> vec5(vec3);                            // Copy constructor

    std::vector<int> vec6(std::move(vec5));                 // Move constructor

    // 2. Assignment

    vec1 = vec2;                                            // Copy assignment

    vec1 = std::move(vec2);                                 // Move assignment

    vec1 = {10, 20, 30};                                    // Initializer list assignment

    // 3. Element Access

    std::cout << "Element at index 1: " << vec1[1] << std::endl;        // Operator[]

    std::cout << "Element at index 2: " << vec1.at(2) << std::endl;    // at()

    std::cout << "First element: " << vec1.front() << std::endl;        // front()

    std::cout << "Last element: " << vec1.back() << std::endl;          // back()

    int* data = vec1.data();                                            // data()

    std::cout << "Element via data pointer: " << data[0] << std::endl;

    // 4. Iterators

    std::cout << "Elements in vec1: ";
```

```
for (auto it = vec1.begin(); it != vec1.end(); ++it) {                    // begin() and end()
        std::cout << *it << " ";
}
std::cout << std::endl;
std::cout << "Elements in reverse: ";
for (auto it = vec1.rbegin(); it != vec1.rend(); ++it) {                  // rbegin() and rend()
        std::cout << *it << " ";
}
std::cout << std::endl;
// 5. Capacity
std::cout << "Size: " << vec1.size() << std::endl;                        // size()
std::cout << "Capacity: " << vec1.capacity() << std::endl;               // capacity()
std::cout << "Is empty: " << vec1.empty() << std::endl;                   // empty()
vec1.resize(5);                                                           // resize()
std::cout << "Resized vec1 size: " << vec1.size() << std::endl;
vec1.reserve(20);                                                         // reserve()
std::cout << "Reserved capacity: " << vec1.capacity() << std::endl;
// 6. Modifiers
vec1.assign(7, 100);                                                      // assign()
vec1.push_back(200);                                                      // push_back()
vec1.pop_back();                                                          // pop_back()
vec1.insert(vec1.begin() + 1, 300);                                       // insert()
vec1.erase(vec1.begin() + 2);                                             // erase()
vec1.emplace(vec1.begin(), 400);                                          // emplace()
vec1.emplace_back(500);                                                   // emplace_back()
```

```
    vec1.swap(vec3);                                                    // swap()
    vec1.clear();                                                       // clear()
    // 7. Non-member Functions
    std::cout << "Is vec1 == vec3? " << (vec1 == vec3) << std::endl; // operator==
    std::swap(vec1, vec3);                                              // swap()
    std::cout << "Elements after swap: ";
    for (const auto& elem : vec1) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;
    // 8. Algorithms
    std::sort(vec1.begin(), vec1.end());                                // sort()
    std::cout << "Sorted elements: ";
    for (const auto& elem : vec1) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

OUTPUT :-



Design and implement a C++ program that utilizes vectors to efficiently store and manage student exam data. The program should allow for:

Adding new students with their names, IDs, and scores.

Finding a student by name or ID.

Calculating and displaying the average score for a specific student or for the entire class.

(Optional) Modifying existing student data (e.g., adding a new score).

```
#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <numeric>

class Student {                                  // Student class definition

public:

    Student(const std::string& name, int id, const std::vector<int>& scores)

        : name(name), id(id), scores(scores) {}

    void addScore(int score) {
```

```
        scores.push_back(score);
    }
    double calculateAverage() const {
        if (scores.empty()) {
            return 0.0;
        }
        int sum = std::accumulate(scores.begin(), scores.end(), 0);
        return static_cast<double>(sum) / scores.size();
    }
    const std::string& getName() const { return name; }
    int getId() const { return id; }
    const std::vector<int>& getScores() const { return scores; }
    void printStudent() const {
        std::cout << "Name: " << name << ", ID: " << id << ", Scores: ";
        for (int score : scores) {
            std::cout << score << " ";
        }
        std::cout << std::endl;
    }
private:
    std::string name;
    int id;
    std::vector<int> scores;
};
Student* findStudentByName(std::vector<Student>& students, const std::string& name) {        //
Function to find a student by name
    auto it = std::find_if(students.begin(), students.end(), [&name](const Student& student) {
        return student.getName() == name;
```

```
    });
    return (it != students.end()) ? &(*it) : nullptr;
}
Student* findStudentById(std::vector<Student>& students, int id) {                    //
Function to find a student by ID
    auto it = std::find_if(students.begin(), students.end(), [id](const Student& student) {
        return student.getId() == id;
    });
    return (it != students.end()) ? &(*it) : nullptr;
}
double calculateClassAverage(const std::vector<Student>& students) {
// Function to calculate the class average score
    if (students.empty()) {
        return 0.0;
    }
    int totalSum = 0;
    int totalCount = 0;
    for (const Student& student : students) {
        totalSum += std::accumulate(student.getScores().begin(), student.getScores().end(), 0);
        totalCount += student.getScores().size();
    }
    return static_cast<double>(totalSum) / totalCount;
}
int main() {
    std::vector<Student> students;
    students.emplace_back("Arjun", 1, std::vector<int>{85, 90, 78});                // Adding
new students
    students.emplace_back("Bobby", 2, std::vector<int>{92, 88, 79});
    students.emplace_back("Charlie", 3, std::vector<int>{76, 85, 80});
```

```
    std::string searchName = "Arjun";                                    // Finding a
student by name

    Student* studentByName = findStudentByName(students, searchName);

    if (studentByName) {

        std::cout << "Found student by name '" << searchName << "':" << std::endl;

        studentByName->printStudent();

    } else {

        std::cout << "Student with name '" << searchName << "' not found." << std::endl;

    }

    // Finding a student by ID

    int searchId = 2;

    Student* studentById = findStudentById(students, searchId);

    if (studentById) {

        std::cout << "Found student by ID '" << searchId << "':" << std::endl;

        studentById->printStudent();

    } else {

        std::cout << "Student with ID '" << searchId << "' not found." << std::endl;

    }


    // Calculating and displaying the average score for a specific student

    if (studentByName) {

        std::cout << "Average score for " << studentByName->getName() << ": " <<
studentByName->calculateAverage() << std::endl;

    }

    // Calculating and displaying the class average score

    std::cout << "Class average score: " << calculateClassAverage(students) << std::endl;
```

```
    // Modifying existing student data (adding a new score)
    if (studentByName) {
        studentByName->addScore(95);
        std::cout << "Updated scores for " << studentByName->getName() << ":" << std::endl;
        studentByName->printStudent();
    }
    return 0;
}
```

OUTPUT :-