

## 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: ";
```



```

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 :-

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 int main() {
5     // 1. Construction
6     std::vector<int> vec1; // Default constructor
7     std::vector<int> vec2(10, 5); // Fill constructor (10 elements with value 5)
8     std::vector<int> vec3{1, 2, 3, 4, 5}; // Initializer list constructor
9     std::vector<int> vec4(vec3.begin(), vec3.end()); // Range constructor
10    std::vector<int> vec5(vec3); // Copy constructor
11    std::vector<int> vec6(std::move(vec5)); // Move constructor
12
13    // 2. Assignment
14    vec1 = vec2; // Copy assignment
15    vec1 = {10, 20, 30}; // Initializer list assignment
16
17    // 3. Element Access
18    std::cout << "Element at index 1: " << vec1[1] << std::endl; // Operator[]
19    std::cout << "Element at index 2: " << vec1.at(2) << std::endl; // at()
20    std::cout << "First element: " << vec1.front() << std::endl; // front()
21    std::cout << "Last element: " << vec1.back() << std::endl; // back()
22    int* data = vec1.data(); // data()
23    std::cout << "Element via data pointer: " << data[0] << std::endl;
24
25    // 4. Iterators
26    std::cout << "Elements in vec1: ";
27    for (auto it = vec1.begin(); it != vec1.end(); ++it) { // begin() and end()
28        std::cout << *it << " ";
29    }
30    std::cout << std::endl;
31    std::cout << "Elements in reverse: ";
32    for (auto it = vec1.rbegin(); it != vec1.rend(); ++it) {
33        std::cout << *it << " ";
34    }
35    std::cout << std::endl;
36
37    // Program finished with exit code 0
38    Press ENTER to exit console.

```

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

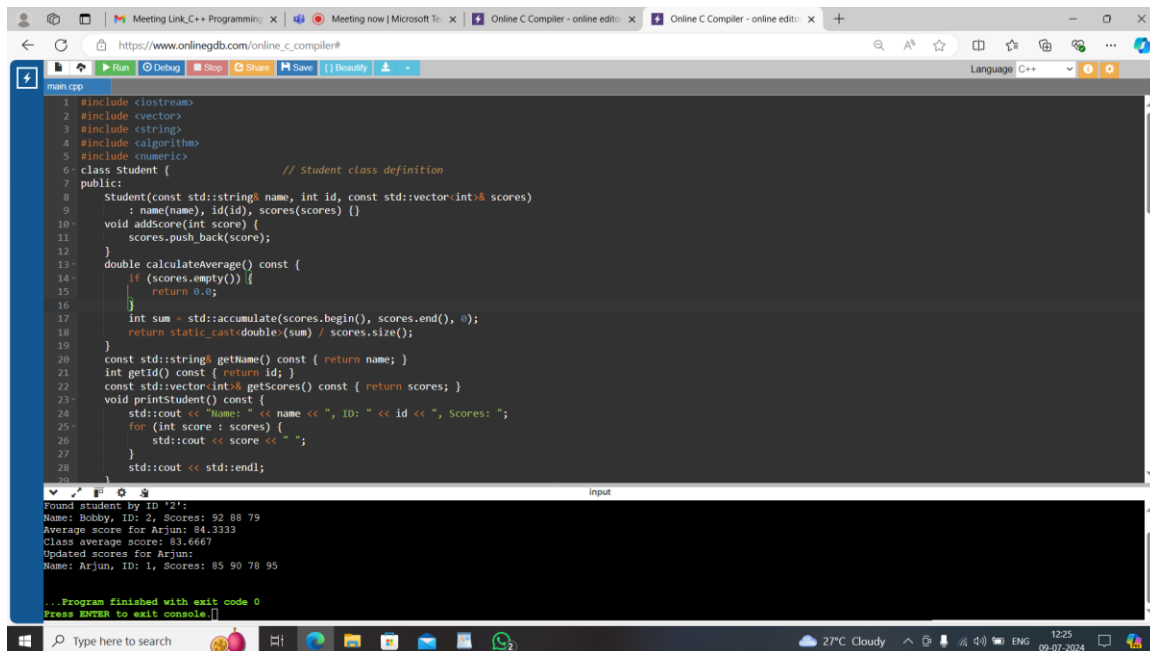
std::cout << "Class average score: " << calculateClassAverage(students) << std::endl;

```

```
// 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 :-



The screenshot shows a web browser window with an online C++ compiler. The code is in a file named `main.cpp`. It includes headers for `<iostream>`, `<vector>`, `<string>`, `<algorithm>`, and `<numeric>`. A `Student` class is defined with methods for adding a score, calculating the average, getting the name, ID, scores, and printing the student details. The `main` function finds a student by ID (2), updates their score to 95, and prints the updated scores. The output in the console shows the student's details before and after the update.

```
1 #include <iostream>  
2 #include <vector>  
3 #include <string>  
4 #include <algorithm>  
5 #include <numeric>  
6 class Student { // Student class definition  
7 public:  
8     Student(const std::string& name, int id, const std::vector<int>& scores)  
9         : name(name), id(id), scores(scores) {}  
10    void addScore(int score) {  
11        scores.push_back(score);  
12    }  
13    double calculateAverage() const {  
14        if (scores.empty()) {  
15            return 0.0;  
16        }  
17        int sum = std::accumulate(scores.begin(), scores.end(), 0);  
18        return static_cast<double>(sum) / scores.size();  
19    }  
20    const std::string& getName() const { return name; }  
21    int getId() const { return id; }  
22    const std::vector<int>& getScores() const { return scores; }  
23    void printStudent() const {  
24        std::cout << "Name: " << name << ", ID: " << id << ", Scores: ";  
25        for (int score : scores) {  
26            std::cout << score << " ";  
27        }  
28        std::cout << std::endl;  
29    }  
30 }  
  
int main() {  
    Student s1("Bobby", 1, {92, 88, 79});  
    Student s2("Arjun", 2, {84, 33, 33});  
    Student s3("Class", 3, {83, 66, 67});  
    Student* studentByName = nullptr;  
    for (Student& s : {s1, s2, s3}) {  
        if (s.getId() == 2) {  
            studentByName = &s;  
        }  
    }  
    if (studentByName) {  
        studentByName->addScore(95);  
        std::cout << "Updated scores for " << studentByName->getName() << ":" << std::endl;  
        studentByName->printStudent();  
    }  
    return 0;  
}
```

Found student by ID '2':  
Name: Bobby, ID: 2, Scores: 92 88 79  
Average score for Arjun: 84.3333  
Class average score: 83.6667  
Updated scores for Arjun:  
Name: Arjun, ID: 1, Scores: 85 90 78 95  
...Program finished with exit code 0  
Press ENTER to exit console.