# DAY – 12 Assignment and LABs

**// Today's Class Codes start from here....**

**// Pre - defined header files or Pre - processor directives**

**// Code - 1 Explanation of Vector Class. (STL) [Import. Concept]**

```
#include <iostream>

#include <vector>

#include <algorithm>

#include <string>


/* int main() {


    // 1. Construction

    std::vector<int> vec1;                    // Default constructor : Creates an empty vector of integers.

    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 : Creates a vector initialized with a list of values: 1, 2, 3, 4, 5.

    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

    // Copies the contents of vec2 to vec1.

    vec1 = vec2;                              // Copy assignment
```

```
vec1 = std::move(vec2);                    // Move assignment
vec1 =    {10, 20, 30};                    // Initializer list assignment


    // {vec1[0],vec1[1],vec1[2]};


// 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 i = vec1.begin(); i != vec1.end(); ++i) {           // begin() and end()
   std::cout << *i << " ";
}
std::cout << std::endl;
std::cout << "Elements in reverse: ";
for (auto i = vec1.rbegin(); i != vec1.rend(); ++i) {         // rbegin() and rend()
   std::cout << *i << " ";
}
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() the vector
std::cout << "Resized vec1 size: " << vec1.size() << std::endl;
vec1.reserve(20);                                               // reserve() the vector
std::cout << "Reserved capacity: " << vec1.capacity() << std::endl;

// 6. Modifiers
vec1.assign(7, 100);                                            // assign()
vec1.push_back(200);  // add element at back                    // push_back()
vec1.pop_back();                                                // pop_back()
vec1.insert(vec1.begin() + 1, 300);  // 2nd position insertion   // insert()
vec1.erase(vec1.begin() + 2);                                   // erase()
vec1.emplace(vec1.begin() + 1, 600);                            // 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==
//  Swap function
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;

} */
```

**// Code - 2**

**/* Assignment : Imagine you're building a program to manage a list of tasks. Each task is represented by a Task object containing details like**

**description, priority, and due date. You want to add tasks to a vector that stores these Task objects. */**

**/* Challenge:**

**You have two options for adding new tasks:**

**Pre-created Tasks: You might have a pre-defined Task object with all its details set.Creating Tasks on the Fly: You might need to create a new**

**Task object on the fly while adding it to the vector, specifying the details during insertion.**

**Understanding the Difference:**

**insert: Use this if you already have a complete Task object ready to be inserted. insert takes the existing Task object and places it at a**

**specific position in the vector. This might involve copying the object's data. emplace: Use this if you need to create a new Task object with specific**

**details while adding it to the vector. emplace calls the Task constructor directly within the vector's memory, initializing the new object with the provided values. This avoids unnecessary copying. */**

```
/* class Task {
public:
    std::string description;
    int priority;
    std::string dueDate;

    Task(const std::string& desc, int prio, const std::string& due)
        : description(desc), priority(prio), dueDate(due) {}

    void printTask() const {
        std::cout << "Description: " << description << ", Priority: " << priority  << ", Due Date: "
<< dueDate << std::endl;
    }
```

```
};
int main() {
    std::vector<Task> taskList;

    // Pre-created Task
    Task preCreatedTask("Complete assignment", 1, "1-3-2024");
    taskList.insert(taskList.begin(), preCreatedTask);

    // Create Tasks on the Fly using emplace
    taskList.emplace_back("Completing Practicals", 2, "2-3-2024");
    taskList.emplace(taskList.begin() + 1, "Solve PYQ's", 3, "4-3-2024");

    // Display all tasks
    for (const auto& task : taskList) {
        task.printTask();
    }
    return 0;
} */
```

**// Task - 2**

**// Assignment - 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 <numeric>
#include <algorithm>

class Student {
public:
    std::string name;
    int id;
    std::vector<int> scores;

    Student(const std::string& name, int id) : name(name), id(id) {}

    void addScore(int score) {
        scores.push_back(score);
    }

    double getAverageScore() const {
        if (scores.empty()) return 0.0;
        int total = std::accumulate(scores.begin(), scores.end(), 0);
        return static_cast<double>(total) / scores.size();
```

```
    }
};

class StudentManager {
private:
    std::vector<Student> students;

public:
    void addStudent(const std::string& name, int id) {
        students.emplace_back(name, id);
    }

    Student* findStudentByName(const std::string& name) {
        for (auto& student : students) {
            if (student.name == name) {
                return &student;
            }
        }
        return nullptr;
    }

    Student* findStudentById(int id) {
        for (auto& student : students) {
            if (student.id == id) {
                return &student;
```

```
        }
    }
    return nullptr;
}

double calculateClassAverage() const {
    if (students.empty()) return 0.0;
    double total = 0.0;
    int count = 0;
    for (const auto& student : students) {
        total += student.getAverageScore();
        ++count;
    }
    return total / count;
}

void displayStudentData(const Student* student) const {
    if (student) {
        std::cout << "Name: " << student->name << "\n";
        std::cout << "ID: " << student->id << "\n";
        std::cout << "Scores: ";
        for (int score : student->scores) {
            std::cout << score << " ";
        }
        std::cout << "\n";
```

```
            std::cout << "Average Score: " << student->getAverageScore() << "\n";
        } else {
            std::cout << "Student not found.\n";
        }
    }
};

int main() {
    StudentManager manager;

    // Add students
    manager.addStudent("Jimmy", 1);
    manager.addStudent("Jismon", 2);

    // Add scores
    Student* jimmy = manager.findStudentByName("Jimmy");
    if (jimmy) {
        jimmy->addScore(96);
        jimmy->addScore(90);
    }

    Student* jismon = manager.findStudentById(2);
    if (jismon) {
        jismon->addScore(88);
        jismon->addScore(86);
```

```
    }

    // Display student data
    std::cout << "Data for Jimy:\n";
    manager.displayStudentData(jimmy);

    std::cout << "\nData for Jismon:\n";
    manager.displayStudentData(jismon);

    // Calculate and display class average
    std::cout << "\nClass Average Score: " << manager.calculateClassAverage() << "\n";

    // Modify student data
    if (jismon) {
      jismon->addScore(85);
    }

    std::cout << "\nUpdated Data for Jismon:\n";
    manager.displayStudentData(jismon);

    return 0;
}
```