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.

#include <iostream>

#include <string>

#include <vector>

class Task {

public:

std::string description;

int priority;

std::string due\_date;

Task(const std::string& desc, int prio, const std::string& due)

: description(desc), priority(prio), due\_date(due) {

std::cout << "Task Created: " << description << std::endl;

}

Task(const Task& other)

: description(other.description), priority(other.priority), due\_date(other.due\_date) {

std::cout << "Task Copied: " << description << std::endl;

}

};

int main() {

std::vector<Task> tasks;

Task preCreatedTask("Finish report", 1, "2024-07-10");

tasks.insert(tasks.begin(), preCreatedTask);

tasks.insert(tasks.end(), preCreatedTask);

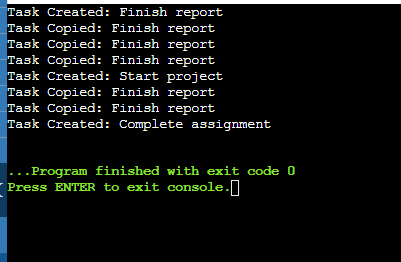
tasks.emplace(tasks.begin(), "Start project", 2, "2024-08-01");

tasks.emplace\_back("Complete assignment", 3, "2024-07-15");

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>

struct Student {

std::string name;

int id;

std::vector<int> scores;

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

};

class ExamManager {

private:

std::vector<Student> students;

public:

void addStudent(const std::string& name, int id, int score) {

Student newStudent(name, id);

newStudent.scores.push\_back(score);

students.push\_back(newStudent);

}

void findStudentByName(const std::string& name) {

for (const auto& student : students) {

if (student.name == name) {

std::cout << "Student found: " << student.name << " (ID: " << student.id << ")\n";

return;

}

}

std::cout << "Student not found.\n";

}

void findStudentByID(int id) {

for (const auto& student : students) {

if (student.id == id) {

std::cout << "Student found: " << student.name << " (ID: " << student.id << ")\n";

return;

}

}

std::cout << "Student not found.\n";

}

void calculateAverageScore(const std::string& name) {

for (const auto& student : students) {

if (student.name == name) {

int sum = 0;

for (int score : student.scores) {

sum += score;

}

double average = static\_cast<double>(sum) / student.scores.size();

std::cout << "Average score for " << student.name << ": " << average << "\n";

return;

}

}

std::cout << "Student not found.\n";

}

void calculateClassAverage() {

int totalSum = 0;

int totalScores = 0;

for (const auto& student : students) {

for (int score : student.scores) {

totalSum += score;

totalScores++;

}

}

double classAverage = static\_cast<double>(totalSum) / totalScores;

std::cout << "Class average score: " << classAverage << "\n";

}

void modifyStudentScore(const std::string& name, int newScore) {

for (auto& student : students) {

if (student.name == name) {

student.scores.push\_back(newScore);

std::cout << "Score updated for " << student.name << ".\n";

return;

}

}

std::cout << "Student not found.\n";

}

};

int main() {

ExamManager manager;

manager.addStudent("John Doe", 1, 80);

manager.addStudent("Jane Smith", 2, 90);

manager.addStudent("Bob Johnson", 3, 70);

manager.findStudentByName("Jane Smith");

manager.findStudentByID(2);

manager.calculateAverageScore("John Doe");

manager.calculateClassAverage();

manager.modifyStudentScore("Jane Smith", 95);

manager.calculateAverageScore("Jane Smith");

return 0;

}

Output:

