// 2D-vector.cpp

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<vector<int>> grid(2, vector<int>(3)); // 2 rows, 3 columns each

grid[1][2] = 99;

for (const auto& row : grid) {

for (int val : row) {

cout << val << " ";

}

cout << "\n";

}

for (size\_t i = 0; i < grid.size(); ++i) {

for (size\_t j = 0; j < grid.at(i).size(); ++j) {

cout << grid.at(i).at(j) << " ";

// cout << grid[i][j] << " "; // this works too, just no bounds check

}

cout << "\n";

}

return 0;

}

//2D-multi-threading.cpp

#include <iostream>

#include <vector>

#include <thread>

using namespace std;

// Thread function: add one row from A and B into C

void addRow(const vector<vector<int>>& A,

const vector<vector<int>>& B,

vector<vector<int>>& C,

int row) {

for (size\_t col = 0; col < A[0].size(); ++col) {

C[row][col] = A[row][col] + B[row][col];

}

}

//main.c

int main() {

const int rows = 3, cols = 4;

// Initialize matrices A and B with example values

vector<vector<int>> A(rows, vector<int>(cols, 1)); // all elements = 1

vector<vector<int>> B(rows, vector<int>(cols, 2)); // all elements = 2

vector<vector<int>> C(rows, vector<int>(cols)); // result matrix

// Create a thread for each row

vector<thread> threads;

for (int i = 0; i < rows; ++i) {

threads.emplace\_back(addRow, cref(A), cref(B), ref(C), i);

}

// Join all threads

for (auto& t : threads) {

t.join();

}

cout << "Result matrix C = A + B:\n";

for (const auto& row : C) {

for (int val : row) {

cout << val << " ";

}

cout << "\n";

}

return 0;

}

// thread5\_1.cpp

#include <iostream>

#include <vector>

#include <thread>

#include <string>

#include <sstream> // <<== include this

void task(int num) {

std::ostringstream oss;

oss << "thread id: " << std::this\_thread::get\_id()

<< " | task number: " << num << "\n";

std::string str = oss.str();

std::cout << str;

}

int main() {

std::vector<std::thread> threads;

for (int i = 0; i < 5; ++i) {

threads.emplace\_back(task, i); // constructs thread in-place

}

for (auto& t : threads) {

t.join();

}

return 0;

}