#include <iostream>

#include <vector>

#include <omp.h>

// Function to perform parallel reduction for finding minimum value

template<typename T>

T parallel\_min(const std::vector<T>& vec) {

T min\_val = vec[0];

#pragma omp parallel for reduction(min:min\_val)

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

if (vec[i] < min\_val) {

min\_val = vec[i];

}

}

return min\_val;

}

// Function to perform parallel reduction for finding maximum value

template<typename T>

T parallel\_max(const std::vector<T>& vec) {

T max\_val = vec[0];

#pragma omp parallel for reduction(max:max\_val)

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

if (vec[i] > max\_val) {

max\_val = vec[i];

}

}

return max\_val;

}

// Function to perform parallel reduction for finding sum

template<typename T>

T parallel\_sum(const std::vector<T>& vec) {

T sum\_val = 0;

#pragma omp parallel for reduction(+:sum\_val)

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

sum\_val += vec[i];

}

return sum\_val;

}

// Function to perform parallel reduction for finding average

template<typename T>

double parallel\_average(const std::vector<T>& vec) {

T sum\_val = parallel\_sum(vec);

return static\_cast<double>(sum\_val) / vec.size();

}

int main() {

// Example vector

std::vector<int> numbers = {5, 9, 2, 7, 4, 8, 1, 6, 3};

// Find minimum

int min\_val = parallel\_min(numbers);

std::cout << "Minimum value: " << min\_val << std::endl;

// Find maximum

int max\_val = parallel\_max(numbers);

std::cout << "Maximum value: " << max\_val << std::endl;

// Find sum

int sum\_val = parallel\_sum(numbers);

std::cout << "Sum: " << sum\_val << std::endl;

// Find average

double avg\_val = parallel\_average(numbers);

std::cout << "Average: " << avg\_val << std::endl;

return 0;

}