/\*

\* assign3\_test.cpp

\* Test runner for assignment 3

\*/

#include <iostream>

#include <algorithm>

#include <functional>

#include <random>

#include <vector>

using namespace std;

/\* make\_random\_vector(len)

Returns a vector<int> of random values, where each entry is between 0 and

INT\_MAX. The optional second parameter lets you specify the seed to be used

for the RNG.

\*/

std::vector<int> make\_random\_vector(

std::size\_t len,

int seed = 1)

{

std::default\_random\_engine generator(seed);

std::uniform\_int\_distribution<int> distribution;

auto gen = std::bind(distribution, generator);

// Fill with random values

std::vector<int> ret(len, 0);

for(std::size\_t i = 0; i < len; ++i)

ret.at(i) = gen() % 100;

return ret;

}

/\* make\_random\_permutation(len)

Returns a vector of length len containing a random permutation of the

integers 0...len-1. This can, of course, be used to randomly permute any

vector of length len.

\*/

std::vector<unsigned> make\_random\_permutation(

std::size\_t len,

int seed = 1)

{

std::default\_random\_engine generator(seed);

std::vector<unsigned> ret(len, 0);

// Initialize vector to 0...len-1

for(std::size\_t i = 0; i < len; ++i)

ret.at(i) = i;

std::shuffle(ret.begin(), ret.end(), generator);

return ret;

}

/\* These functions must be defined in the student's code \*/

void merge(int\* input, int size, int\* output, bool output\_asc);

int\* mergesort(int\* input, int size);

void mergesort(int \*input, int size, int\* output, bool output\_asc);

// Convenience function for using mergesort on vectors

int\* mergesort(const vector<int>& data) {

return mergesort(const\_cast<int\*>(data.data()), data.size());

}

/\* is\_sorted(data, size)

Returns true if the data is sorted ascending.

\*/

bool is\_sorted(int\* data, int size) {

for(int\* p = data + 1; p < data + size; ++p) {

if(\*p < \*(p-1))

return false;

}

return true;

}

bool is\_permutation(int\* input, int size, int\* sorted) {

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

// Check if input[i] is in sorted

int elem = input[i];

bool found = false;

for(int j = 0; j < size; ++j)

if(sorted[j] == elem) {

found = true;

break;

}

if(!found)

return false;

// Check if sorted[i] is in input

elem = sorted[i];

found = false;

for(int j = 0; j < size; ++j)

if(input[j] == elem) {

found = true;

break;

}

if(!found)

return false;

}

return true;

}

/\* out << vec

Convenience overload for printing vector<int>

\*/

ostream& operator<<(ostream& out, const vector<int>& data) {

out << "{";

for(unsigned i = 0; i < data.size() - 1; ++i)

out << data[i] << ", ";

out << data.back() << "}";

return out;

}

/\* random\_growth(start,size,asc)

Generates a vector whose values start at start and either increase (if asc is

true) or decrease by random increments. The increment is in the range

0...8.

\*/

void random\_growth(int\* data, int start, int size, bool asc) {

std::default\_random\_engine generator(17);

std::uniform\_int\_distribution<int> distribution(0,9);

auto rnd = std::bind(distribution, generator);

const int step = asc ? +1 : -1;

if(size > 0) {

data[0] = start;

for(int i = 1; i < size; ++i)

data[i] = data[i-1] + step \* rnd();

}

}

/\* test\_merge()

Test the merge() function.

This basically just checks merge() to make sure that the output is merged

in the correct order. We also check things like merging small arrays

(size 0, 1, 2, and 3) since those are easy to get wrong. The system also

checks the amount of space allocated before and after this function is

called and will return false if anything has been malloc()'d.

\*/

bool test\_merge() {

cout << "Testing simple two-element merge: \n";

vector<int> v1 = { 1, 2 };

vector<int> vout = { -1, -1 };

// Merge asc.

merge(&v1[0], 2, &vout[0], true);

if(vout[0] != 1 || vout[1] != 2) {

cout << "FAILED: merge result was " << vout << ".\n";

return false;

}

// Merge desc.

merge(&v1[0], 2, &vout[0], false);

if(vout[0] != 2 || vout[1] != 1) {

cout << "FAILED: merge result was " << vout << ".\n";

return false;

}

cout << "OK\n";

// Generate asc-desc dataset for testing

cout << "Testing 20-element merge (10 and 10): ";

vector<int> data(20);

random\_growth(&data[0], 0, 10, true);

random\_growth(&data[10], 2, 10, false);

vector<int> dataout(20);

merge(&data[0], data.size(), &dataout[0], true);

if(!is\_sorted(&dataout[0], data.size())) {

cout << "FAILED: merge did not produce sorted output : "

<< dataout << endl;

return false;

}

cout << "OK\n";

cout << "Testing 21-element merge: ";

data.resize(21);

random\_growth(&data[0], 0, 10, true);

random\_growth(&data[10], 2, 11, false);

dataout.resize(21);

merge(&data[0], data.size(), &dataout[0], true);

if(!is\_sorted(&dataout[0], data.size())) {

cout << "FAILED: merge did not produce sorted output : "

<< dataout << endl;

return false;

}

cout << "OK\n";

return true;

}

/\* test\_mergesort()

Test mergesort on a variety of inputs.

\*/

bool test\_mergesort() {

cout << "Sorting empty sequence:";

vector<int> no\_data;

int\* no\_data\_sorted = mergesort(no\_data);

// No data means nothing to check!

delete[] no\_data\_sorted;

cout << "OK\n";

vector<int> sizes = {2, 3, 4, 5, 7, 8, 15, 16, 19, 20, 50, 64, 100, 128};

for(int s : sizes) {

// Test sorting a vector of random data

vector<int> data = make\_random\_vector(s);

cout << "Sorting random vector of size " << s << ":\n" << data << "\n";

int\* data\_sorted = mergesort(data);

if(!is\_sorted(data\_sorted, data.size())) {

cout << "FAILED: result is not sorted:\n";

cout << "{";

for(int\* i = data\_sorted; i != data\_sorted + data.size() - 1; ++i)

cout << \*i << ", ";

cout << data\_sorted[data.size() - 1] << "}\n";

return false;

}

else if(!is\_permutation(&data[0], data.size(), data\_sorted)) {

cout << "FAILED: result is not a permutation of the input sequence:\n";

cout << "{";

for(int\* i = data\_sorted; i != data\_sorted + data.size() - 1; ++i)

cout << \*i << ", ";

cout << data\_sorted[data.size() - 1] << "}\n";

return false;

}

else

cout << "OK\n";

}

return true;

}

int main() {

cout << "\*\*\*\* Testing mergesort \*\*\*\*\n";

if(test\_merge() &&

test\_mergesort())

cout << "\*\*\*\* All tests passed! \*\*\*\*\n";

return 0;

}