/\*

//============================================================================

// Name : VectorSorting.cpp

// Author : Your name

// Version : 1.0

// Copyright : Copyright © 2017 SNHU COCE

// Description : Vector Sorting Algorithms

//============================================================================

#include <algorithm>

#include <iostream>

#include <time.h>

#include "CSVparser.hpp"

using namespace std;

//============================================================================

// Global definitions visible to all methods and classes

//============================================================================

// forward declarations

double strToDouble(string str, char ch);

// define a structure to hold bid information

struct Bid {

string bidId; // unique identifier

string title;

string fund;

double amount;

Bid() {

amount = 0.0;

}

};

//============================================================================

// Static methods used for testing

//============================================================================

/\*

\* Display the bid information to the console (std::out)

\*

\* @param bid struct containing the bid info

\*

void displayBid(Bid bid) {

cout << bid.bidId << ": " << bid.title << " | " << bid.amount << " | "

<< bid.fund << endl;

return;

}

/\*

\* Prompt user for bid information using console (std::in)

\*

\* @return Bid struct containing the bid info

\*

Bid getBid() {

Bid bid;

cout << "Enter Id: ";

cin.ignore();

getline(cin, bid.bidId);

cout << "Enter title: ";

getline(cin, bid.title);

cout << "Enter fund: ";

cin >> bid.fund;

cout << "Enter amount: ";

cin.ignore();

string strAmount;

getline(cin, strAmount);

bid.amount = strToDouble(strAmount, '$');

return bid;

}

/\*

\* Load a CSV file containing bids into a container

\*

\* @param csvPath the path to the CSV file to load

\* @return a container holding all the bids read

\*

vector<Bid> loadBids(string csvPath) {

cout << "Loading CSV file " << csvPath << endl;

// Define a vector data structure to hold a collection of bids.

vector<Bid> bids;

// initialize the CSV Parser using the given path

csv::Parser file = csv::Parser(csvPath);

try {

// loop to read rows of a CSV file

for (unsigned int i = 0; i < file.rowCount(); i++) {

// Create a data structure and add to the collection of bids

Bid bid;

bid.bidId = file[i][1];

bid.title = file[i][0];

bid.fund = file[i][8];

bid.amount = strToDouble(file[i][4], '$');

//cout << "Item: " << bid.title << ", Fund: " << bid.fund << ", Amount: " << bid.amount << endl;

// push this bid to the end

bids.push\_back(bid);

}

} catch (csv::Error &e) {

std::cerr << e.what() << std::endl;

}

return bids;

}

// FIXME (2a): Implement the quick sort logic over bid.title

/\*

\* Partition the vector of bids into two parts, low and high

\*

\* @param bids Address of the vector<Bid> instance to be partitioned

\* @param begin Beginning index to partition

\* @param end Ending index to partition

\*

int partition(vector<Bid>& bids, int begin, int end) {

//pick middle element as pivot

int pivot = begin + (end - begin) /2;

//boolean variable to stop loop

bool done = false;

//bids temp;

while (!done){

//increment begin while less than pivot

while(bids.at(begin).title.compare(bids.at(pivot).title) < 0){

begin++;

}

//decrement end while more than pivot

while(bids.at(pivot).title.compare(bids.at(end).title) < 0){

end--;

}

if(begin >= end){

done = true;

} else {

//swap values, coded as in text and as in video

swap(bids.at(begin), bids.at(end));

//temp = bids.at(begin);// stores first unsorted value in temp

//bids.at(begin) = bids.at(end);// assigns minimum value to first unsorted value

//bids.at(end) = temp;// assigns value in temp to former location of minimum bid

begin++;

end--;

}

}

return end;

}

/\*

\* Perform a quick sort on bid title

\* Average performance: O(n log(n))

\* Worst case performance O(n^2))

\*

\* @param bids address of the vector<Bid> instance to be sorted

\* @param begin the beginning index to sort on

\* @param end the ending index to sort on

\*

void quickSort(vector<Bid>& bids, int begin, int end) {

int mid = 0;

// base case

if (begin >= end){

return;

}

// partition bids into low and high parts

mid = partition(bids, begin, end);

// recursively call quicksort to partition lower part

quickSort(bids, begin, mid);

// recursively call quicksort to partition higher part

quickSort(bids, mid + 1, end);

return;

}

// FIXME (1a): Implement the selection sort logic over bid.title

/\*

\* Perform a selection sort on bid title

\* Average performance: O(n^2))

\* Worst case performance O(n^2))

\*

\* @param bid address of the vector<Bid>

\* instance to be sorted

\*

void selectionSort(vector<Bid>& bids) {

// current minimum bid

unsigned int min;

//bids temp;

// outer loop controls first unsorted record

for (unsigned i = 0; i < bids.size(); i++) {

min = i;

// inner loop cycles through the vector looking for lowest value and stores it in min

for (unsigned j = i+1; j < bids.size(); j++) {

if(bids.at(j).title.compare(bids.at(min).title) < 0){

min = j;

}

}

// if min is different than the first unsorted value they are swapped

// coded following process in text and process in video

if(min != i){

swap(bids.at(i), bids.at(min));

//temp = bids.at(i);// stores first unsorted value in temp

//bids.at(i) = bids.at(min);// assigns minimum value to first unsorted value

//bids.at(min) = temp;// assigns value in temp to former location of minimum bid

}

}

}

/\*

\* Simple C function to convert a string to a double

\* after stripping out unwanted char

\*

\* credit: http://stackoverflow.com/a/24875936

\*

\* @param ch The character to strip out

\*

double strToDouble(string str, char ch) {

str.erase(remove(str.begin(), str.end(), ch), str.end());

return atof(str.c\_str());

}

/\*

\* The one and only main() method

\*

int main(int argc, char\* argv[]) {

// process command line arguments

string csvPath;

switch (argc) {

case 2:

csvPath = argv[1];

break;

default:

csvPath = "eBid\_Monthly\_Sales.csv";

}

// Define a vector to hold all the bids

vector<Bid> bids;

// Define a timer variable

clock\_t ticks;

int choice = 0;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Bids" << endl;

cout << " 2. Display All Bids" << endl;

cout << " 3. Selection Sort All Bids" << endl;

cout << " 4. Quick Sort All Bids" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

// Initialize a timer variable before loading bids

ticks = clock();

// Complete the method call to load the bids

bids = loadBids(csvPath);

cout << bids.size() << " bids read" << endl;

// Calculate elapsed time and display result

ticks = clock() - ticks; // current clock ticks minus starting clock ticks

cout << "time: " << ticks << " clock ticks" << endl;

cout << "time: " << ticks \* 1.0 / CLOCKS\_PER\_SEC << " seconds" << endl;

break;

case 2:

// Loop and display the bids read

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

displayBid(bids[i]);

}

cout << endl;

break;

// FIXME (1b): Invoke the selection sort and report timing results

case 3:

// Initialize a timer variable before sorting bids

ticks = clock();

// Complete the method call to selection sort

selectionSort(bids);

cout << bids.size() << " bids read" << endl;

// Calculate elapsed time and display result

ticks = clock() - ticks; // current clock ticks minus starting clock ticks

cout << "time: " << ticks << " clock ticks" << endl;

cout << "time: " << ticks \* 1.0 / CLOCKS\_PER\_SEC << " seconds" << endl;

break;

// FIXME (2b): Invoke the quick sort and report timing results

case 4:

// Initialize a timer variable before sorting bids

ticks = clock();

// Complete the method call to quick sort

quickSort(bids, 0, bids.size() - 1);

cout << bids.size() << " bids read" << endl;

// Calculate elapsed time and display result

ticks = clock() - ticks; // current clock ticks minus starting clock ticks

cout << "time: " << ticks << " clock ticks" << endl;

cout << "time: " << ticks \* 1.0 / CLOCKS\_PER\_SEC << " seconds" << endl;

break;

}

}

cout << "Good bye." << endl;

return 0;

}

\*/