/\*

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

// Name : LinkedList.cpp

// Author : Lesley Potts-Langdon

// Version : 1.0

// Copyright : Copyright © 2017 SNHU COCE

// Description : Lab 3-3 Lists and Searching

// Date : January 28, 2018

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

#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;

}

};

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

// Linked-List class definition

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

\*

\* Define a class containing data members and methods to

\* implement a linked-list.

class LinkedList {

private:

// FIXME (1): Internal structure for list entries, housekeeping variables

struct Node{

Bid bid; //declare instance of Bid struct

Node\* next; //Node pointer to variable next

//default constructor for struct Node

Node() {

next = nullptr;//sets next pointer to null important for finding end

}

//constructor to initialize a node with a bid

Node(Bid aBid){

bid = aBid;

next = nullptr;

}

};

Node\* head;

Node\* tail;

int size = 0;

public:

LinkedList();

virtual ~LinkedList();

void Append(Bid bid);

void Prepend(Bid bid);

void PrintList();

void Remove(string bidId);

Bid Search(string bidId);

int Size();

};

\*

\* Default constructor

LinkedList::LinkedList() {

// FIXME (2): Initialize housekeeping variables

head = nullptr;

tail = nullptr;

}

\*

\* Destructor

LinkedList::~LinkedList() {

}

\*

\* Append a new bid to the end of the list

void LinkedList::Append(Bid bid) {

// FIXME (3): Implement append logic

Node\* node = new Node(bid);//new node pass bid

if (head == nullptr) {

head = node;

//tail = node;

} else {

if (tail != nullptr) {

tail->next = node;

}

}

//new node is always the tail

tail = node;

size ++;

}

\*

\* Prepend a new bid to the start of the list

void LinkedList::Prepend(Bid bid) {

// FIXME (4): Implement prepend logic

Node\* node = new Node(bid);//new node pass bid

if (head != nullptr){

node->next = head;

}

head = node;

size++;//increment size

}

\*

\* Simple output of all bids in the list

void LinkedList::PrintList() {

// FIXME (5): Implement print logic

Node\* current = head;//set current to the head node

//loop through linked list and print info from each node

while (current != nullptr){

cout << current->bid.bidId << ": " << current->bid.title << " | "

<< current->bid.amount << " | " << current->bid.fund << endl;

current = current->next;

}

}

\*

\* Remove a specified bid

\*

\* @param bidId The bid id to remove from the list

void LinkedList::Remove(string bidId) {

// FIXME (6): Implement remove logic

if (head != nullptr){

//special case if head is the node to be removed

if(head->bid.bidId.compare(bidId) == 0){//compares id values, if they match it returns 0

Node\* tempNode = head->next;//assign node after one to be removed to a temp node

delete head;

head = tempNode;//assign node following one removed to head

size--;//decrement size

}

else{

Node\* current = head;//node pointer current points to head

//loop through linked list and search for match

while (current->next != nullptr){

//compare method will return 0 if values are equal

if(current->next->bid.bidId.compare(bidId) == 0){

//points to node to be removed

Node\* tempNode = current->next;

//make current node the one after the one to be removed

current->next = tempNode->next;

//delete selected node

delete tempNode;

size--;//decrement size

return;

}

current = current->next;

}

}

}

}

\*

\* Search for the specified bidId

\*

\* @param bidId The bid id to search for

Bid LinkedList::Search(string bidId) {

// FIXME (7): Implement search logic

Node\* current = head;

//loop through linked list and search for match

while (current != nullptr){

//compare method will return 0 if values are equal

if(current->bid.bidId.compare(bidId) == 0){

return current->bid;

}

current = current->next;

}

}

\*

\* Returns the current size (number of elements) in the list

int LinkedList::Size() {

return size;

}

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

// Static methods used for testing

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

\*

\* Display the bid information

\*

\* @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

\*

\* @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 LinkedList

\*

\* @return a LinkedList containing all the bids read

void loadBids(string csvPath, LinkedList \*list) {

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

// initialize the CSV Parser

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

try {

// loop to read rows of a CSV file

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

// initialize a bid using data from current row (i)

Bid bid;

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

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

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

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

cout << bid.bidId << ": " << bid.title << " | " << bid.fund << " | " << bid.amount << endl;

// add this bid to the end

list->Append(bid);

}

} catch (csv::Error &e) {

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

}

}

\*

\* 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

\*

\* @param arg[1] path to CSV file to load from (optional)

\* @param arg[2] the bid Id to use when searching the list (optional)

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

// process command line arguments

string csvPath, bidKey;

switch (argc) {

case 2:

csvPath = argv[1];

bidKey = "98109";

break;

case 3:

csvPath = argv[1];

bidKey = argv[2];

break;

default:

csvPath = "eBid\_Monthly\_Sales\_Dec\_2016.csv";

bidKey = "98109";

}

clock\_t ticks;

LinkedList bidList;

Bid bid;

int choice = 0;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Enter a Bid" << endl;

cout << " 2. Load Bids" << endl;

cout << " 3. Display All Bids" << endl;

cout << " 4. Find Bid" << endl;

cout << " 5. Remove Bid" << endl;

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

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

bid = getBid();

bidList.Append(bid);

displayBid(bid);

break;

case 2:

ticks = clock();

loadBids(csvPath, &bidList);

cout << bidList.Size() << " bids read" << endl;

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

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

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

break;

case 3:

bidList.PrintList();

break;

case 4:

ticks = clock();

bid = bidList.Search(bidKey);

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

if (!bid.bidId.empty()) {

displayBid(bid);

} else {

cout << "Bid Id " << bidKey << " not found." << endl;

}

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

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

break;

case 5:

bidList.Remove(bidKey);

break;

}

}

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

return 0;

}

\*/