**Sabastian Fasano**

**January 25, 2025**

**CS-300**

**Module 3 Code Reflection**

The code is broken down into the following functions/structs/classes:

* **Class LinkedList**
* **Definition of LinkedList objects, which contain:**
* **struct Node:**
  + Holds a Bid struct.
  + Includes a pointer to the next node.
* **Methods:**
  + Append, Prepend, PrintList, Remove, Search, Size.
* **Private members:**
  + head, tail, size.
* **LinkedList()**
  + Default constructor that initializes head and tail to NULL.
* **LinkedList:Append(Bid)**
  + Accepts a Bid as a parameter.
  + Adds the passed Bid as a new node to the end of the list.
  + Updates the head and tail pointers to reflect the changes as needed.
* **LinkedList:Prepend(Bid)**
  + Accepts a Bid as a parameter.
  + Adds the passed Bid as a new node to the beginning of the list.
  + Updates the head and tail pointers as necessary to maintain the list structure.
* **LinkedList:PrintList()**
  + Loops through the LinkedList starting at the head.
  + Outputs four values from the Bid struct to the console.
* **LinkedList:Remove(String)**
  + Starts at the head and searches for the node with the matching string.
  + Removes the node containing the specified string and frees its memory.
  + Updates the head or tail pointers as necessary if the removed node is the head or tail.
* **LinkedList:Search(String)**
  + Starts at the head and searches for the node with the matching string.
  + Returns the node containing the specified string.
* **LinkedList:Size()**
  + A getter method that returns the value of the private size member.
* **strToDouble**
  + Converts CSV file data into a usable numerical format.
* **Bid**
  + A struct containing the data for each bid.
  + Used with a vector that will be sorted.
* **displayBid**
  + Outputs the values contained in the vector to the console.
* **getBid**
  + Allows the user to append or prepend a bid to the list.
* **loadBids**
  + Reads data from a CSV file.
  + Accepts the CSV file path from arguments or uses a default path.
  + Populates the Bid structure with data and adds each Bid to the unsorted LinkedList.
* **main**
  + The main function drives the application.
  + Provides a menu allowing the user to:
    - Enter a bid, load data, view data, delete a node, or exit the application.
    - Reports the execution time of each algorithm using the time.h library.

**Code Observations**

The code was straightforward to work with since the parser was already provided. However, challenges arose when deleting nodes, especially the tail node. These issues were resolved by using multiple pointer access calls (e.g., head->next->next) to navigate the list efficiently.

**Pseudocode:**

* **Main Function()**
  + Read command-line arguments.
    - Store the argument as a CSV file path.
    - Use a default path if no argument is provided.
  + Loop until the user's choice is '9':
    - Display the menu and get user input.
    - Validate the input, throwing an error if it’s not between 1-4 or 9.
    - Process the choice:
      * **Choice 1:**
        + Call getBid() to store a user-entered bid.
        + Append or prepend the bid.
        + Call displayBid().
      * **Choice 2:**
        + Start the clock and record the time.
        + Call loadBids() to load data into the LinkedList.
        + Display the number of records in the CSV file.
        + Stop the clock and display the elapsed time.
      * **Choice 3:**
        + Call PrintList() to display the bids.
      * **Choice 4:**
        + Start the clock and record the time.
        + Call Search() to find a bid by key.
        + Stop the clock and display the elapsed time.
      * **Choice 5:**
        + Call Remove() to delete a bid by key.
      * **Choice 9:**
        + Exit the application.
    - Display “Goodbye!”
* **LinkedList::Append(Bid)**
  + Create a new node.
  + If head is NULL:
    - Set head and tail to the new node.
  + Otherwise:
    - Update the tail pointer to the new node.
  + Increment size by one.
* **LinkedList::Prepend(Bid)**
  + Create a new node for the bid.
  + If head is not NULL:
    - Set the new node as the head.
    - Increment size by one.
  + Otherwise:
    - Call Append().
* **LinkedList::PrintList()**
  + Create a new node pointer.
  + Loop through the list starting at head.
    - Output bidId, title, amount, and fund to the console.
* **LinkedList::Search(String)**
  + Create a new node pointer (cursor) and set it to head.
  + Loop until cursor is NULL:
    - If the node contains a bidId matching the string, return it.
    - Move to the next node.
* **LinkedList::Remove(String)**
  + Create a cursor node pointer and set it to head.
  + Create a tempNode pointer.
  + Loop through the list:
    - If the head matches the string:
      * Update the head pointer.
      * Delete the cursor memory.
      * Decrement size.
    - If the tail matches the string:
      * Update the tail pointer.
      * Set the previous node’s next pointer to NULL.
      * Delete the memory of the old tail.
      * Decrement size.
    - If a node in between matches the string:
      * Update pointers to skip the node.
      * Delete the node’s memory.
      * Decrement size.

**Sabastian Fasnao**

**CS-300**

**January 25, 2025**

**Module 3 Milestone: Pseudocode**

**Load** text parsing libraries and headers

**Define** a struct to hold course data

**struct Course {}**

*courseID*

*courseName*

*preCount*

*preList*

Course() (constructor) {courseID = courseName = ””; preCount = 0; preList = “”}

Main()

* Create a new list named courseList of type Course.
* Prompt the user to input the path to a CSV file.
  + If no path is provided, use the default file location.
* Call txtParser() and pass the CSV file path.
* Call validateList() and pass courseList.
* Prompt the user to input a search value and store it in userSearch.
* Call printCourse() and pass userSearch.

End

txtParser(String)

* Create a local list named tempList.
* Open the file at the specified path using parsing libraries.
* Iterate through the file row by row until the end of the file (EOF).
  + If both the first and second strings are present:
    - Assign the first string to courseID in the struct.
    - Assign the second string to courseName in the struct.
    - Loop through the remaining columns until no more prerequisites are found:
      * Increment a variable preCount for each prerequisite encountered.
      * Append each prerequisite to a local string preNames.
    - Assign preCount to the struct's preCount.
    - Assign preNames to the struct's preList.
* Return tempList.

End

searchList(String)

* Create a temporary variable tempCourse of type Course.
* Iterate through the list of courses:
  + If the String matches courseID, set tempCourse to the current course.
* Return tempCourse.

End

printCourse(String)

* Create a temporary variable tempCourse of type Course.
* Set tempCourse to the result of searchList(String).
* Output courseID to the console.
* Output courseName to the console.
* Loop from 0 to preCount:
  + For each course in preList, call printCourse() with the current prerequisite.

End

validateList()

* Create a temporary variable tempCourse of type Course.
* Create a variable valid and set it to True.
* Iterate through each course:
  + If valid is False, exit the loop.
  + Loop through prerequisites from 0 to preCount:
    - Set tempCourse to the result of searchList(preList token).
    - If the courseID of tempCourse is empty, set valid to False.
* Return valid.

End

**//============================================================================**

**// Name : LinkedList.cpp**

**// Author : Sabastian Fasano; CS-300; Jan 25, 2025**

**// Version : 1.0**

**// Copyright : Copyright © 2023 SNHU COCE**

**// Description : Lab 3-2 Lists and Searching**

**//============================================================================**

**#include <algorithm>**

**#include <iostream>**

**#include <time.h>**

**#include <Windows.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:**

**//Internal structure for list entries, housekeeping variables**

**struct Node {**

**Bid bid;**

**Node \*next;**

**// default constructor**

**Node() {**

**next = nullptr;**

**}**

**// initialize 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 (1): Initialize housekeeping variables**

**// set head and tail equal to null**

**head = NULL;**

**tail = NULL;**

**}**

**/\*\***

**\* Destructor**

**\*/**

**LinkedList::~LinkedList() {**

**// start at the head**

**Node\* current = head;**

**Node\* temp;**

**// loop over each node, detach from list then delete**

**while (current != nullptr) {**

**temp = current; // hang on to current node**

**current = current->next; // make current the next node**

**delete temp; // delete the orphan node**

**}**

**}**

**/\*\***

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

**\*/**

**void LinkedList::Append(Bid bid) {**

**// FIXME (2): Implement append logic**

**//Create new node**

**//if there is nothing at the head...**

**// new node becomes the head and the tail**

**//else**

**// make current tail node point to the new node**

**// and tail becomes the new node**

**//increase size count**

**Node\* newNode = new Node(bid);**

**if (head == NULL) {**

**head = newNode;**

**tail = newNode;**

**}**

**else {**

**tail->next = newNode;**

**tail = newNode;**

**}**

**size++;**

**}**

**/\*\***

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

**\*/**

**void LinkedList::Prepend(Bid bid) {**

**// FIXME (3): Implement prepend logic**

**// Create new node**

**// if there is already something at the head...**

**// new node points to current head as its next node**

**// head now becomes the new node**

**//increase size count**

**Node\* newNode = new Node(bid);**

**if (head->next != nullptr) {**

**newNode->next = head;**

**head = newNode;**

**size++;**

**}**

**else {**

**Append(bid);**

**}**

**}**

**/\*\***

**\* Simple output of all bids in the list**

**\*/**

**void LinkedList::PrintList() {**

**// FIXME (4): Implement print logic**

**// start at the head**

**// while loop over each node looking for a match**

**//output current bidID, title, amount and fund**

**//set current equal to next**

**Node\* currentNode = head;**

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

**cout << currentNode->bid.bidId << ": " << currentNode->bid.title << " | " << currentNode->bid.amount**

**<< " | " << currentNode->bid.fund << endl;**

**currentNode = currentNode->next;**

**}**

**}**

**/\*\***

**\* Remove a specified bid**

**\***

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

**\*/**

**void LinkedList::Remove(string bidId) {**

**// FIXME (5): Implement remove logic**

**// special case if matching node is the head**

**// make head point to the next node in the list**

**//decrease size count**

**//return**

**// start at the head**

**// while loop over each node looking for a match**

**// if the next node bidID is equal to the current bidID**

**// hold onto the next node temporarily**

**// make current node point beyond the next node**

**// now free up memory held by temp**

**// decrease size count**

**//return**

**// curretn node is equal to next node**

**Node\* cursor = head;**

**Node\* tempNode;**

**for (int i = 0; i < size - 1; i++) {**

**if ((cursor->bid.bidId == bidId) && (i == 0)) {//match the head**

**head = cursor->next;**

**free(cursor);**

**size--;**

**break;**

**}**

**else if ((cursor->next->bid.bidId == bidId) && (cursor->next->next == nullptr)) {//match the tail**

**tempNode = cursor->next;**

**tail = cursor;**

**cursor->next = nullptr;**

**free(tempNode);**

**size--;**

**break;**

**}**

**else if (cursor->bid.bidId == bidId) {**

**tempNode = cursor->next;**

**cursor->bid = tempNode->bid;**

**cursor->next = tempNode->next;**

**free(tempNode);**

**size--;**

**break;**

**}**

**cursor = cursor->next;**

**}**

**}**

**/\*\***

**\* Search for the specified bidId**

**\***

**\* @param bidId The bid id to search for**

**\*/**

**Bid LinkedList::Search(string bidId) {**

**Bid bid;**

**// FIXME (6): Implement search logic**

**// special case if matching node is the head**

**// make head point to the next node in the list**

**//decrease size count**

**//return**

**// start at the head of the list**

**// keep searching until end reached with while loop (next != nullptr**

**// if the current node matches, return it**

**// else current node is equal to next node**

**Node\* cursor = head;**

**while (cursor != NULL) {**

**if (cursor->bid.bidId == bidId) {**

**return cursor->bid;**

**}**

**cursor = cursor->next;**

**}**

**return bid;**

**}**

**/\*\***

**\* 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 (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 = "U:/CS 300 Linked List Assignment Student Files/eBid\_Monthly\_Sales\_Dec\_2016.csv"; //small set**

**//csvPath = "U:/CS 300 Linked List Assignment Student Files/eBid\_Monthly\_Sales"; //large set**

**//bidKey = "97990"; //head**

**//bidKey = "98009"; //middle**

**bidKey = "98223"; //tail**

**}**

**clock\_t ticks;**

**LinkedList bidList;**

**Bid bid;**

**const int GLOBAL\_SLEEP\_VALUE = 5000;**

**int choice = 0;**

**string anyKey = " ";**

**bool goodInput;**

**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: ";**

**try {**

**cin >> choice;**

**if ((choice > 0 && choice < 6) || (choice == 9)) {// limit the user menu inputs to good values**

**goodInput = true;**

**}**

**else {//throw error for catch**

**goodInput = false;**

**throw 1;**

**}**

**switch (choice) {**

**case 1:**

**bid = getBid();**

**//bidList.Append(bid);**

**bidList.Prepend(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;**

**Sleep(GLOBAL\_SLEEP\_VALUE);**

**break;**

**case 3:**

**bidList.PrintList();**

**cout << "\n Enter \'y\' to continue..." << endl;**

**cin >> anyKey;**

**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;**

**Sleep(GLOBAL\_SLEEP\_VALUE);**

**break;**

**case 5:**

**bidList.Remove(bidKey);**

**Sleep(GLOBAL\_SLEEP\_VALUE);**

**break;**

**case 9:**

**break;**

**default:**

**throw 2;**

**}**

**}**

**catch (int err) {**

**std::cout << "\nPlease check your input." << endl;**

**Sleep(GLOBAL\_SLEEP\_VALUE);**

**}**

**//need to clear the cin operator of extra input, e.g., 9 9, or any errors generated by bad input, e.g., 'a'**

**cin.clear();**

**cin.ignore();**

**//clear the consolse to redraw a fresh menu**

**system("cls");**

**}**

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

**Sleep(GLOBAL\_SLEEP\_VALUE);**

**return 0;**

**}**