**Pseudocode**

This pseudocode was retrieved from the, “CS-260 Data Structures & Algorithms” class. Also used content and knowledge from zyBooks (Data Structures Essentials with C++):

“PSEUDOCODE: LIST AND SEARCHING

//Import classes/import statements

INCLUDE <algorithm>

INCLUDE <iostream>

INCLUDE <time.h>

//Reference the CSVParser library

INCLUDE "CSVparser.hpp"

USING namespace std;

// Include global definitions visible to all methods and classes

// Forward declarations

DECLARE double strToDouble(string str, char ch)

DEFINE structure to hold bid information

STUCT Bid

// Linked-List class definition

// Define a class containing data members and methods to implement a linked-list

CLASS LinkedList

PRIVATE

DEFINE internal structure for list entries and housekeeping variables

STRUCT node

PUBLIC

MEMBER LinkedList()

MEMBER virtual ~LinkedList()

MEMBER void Append(Bid bid)

MEMBER void Prepend(Bid bid)

MEMBER void PrintList()

MEMBER void Remove(string bidId)

MEMBER Bid Search(string bidId)

MEMBER int Size()

// Default constructor

CREATE LinkedList::LinkedList()

// Initializes housekeeping variables

INITIALIZE head = nulptr

INITIALIZE tail = nullptr

// Destructor

LinkedList::~LinkedList()

// Append a new bid to the end of the list

CREATE void LinkedList::Append(Bid bid)

// Implemening the append logic

DECLARE node\* nodeNew = new node(bid)

DECLARE tail = node

IF (head == nullptr)

head = nextNode

ELSE

IF (tail != nullptr)

tail->nextNode = nodeNew

// Prepend a new bid to the start of the list

CREATE void LinkedList::Prepend(Bid bid)

// Implementing prepend logic

DECLARE node\* nodeNew = new node(bid)

DECLARE nodeNew->nextNode = nullptr

IF (head == nullptr)

head = nextNode

tail = nodeNew

size++

ELSE

nodeNew->nextNode = head

head = nodeNew

size++

//Simple output of all bids in the list

CREATE void LinkedList::PrintList()

// Implementing print logic

DECLARE node\* entry = head;

WHILE (entry != nullptr)

PRINT << enrty->bid.bidId << " ! "

PRINT << entry->bid.title << " | "

PRINT << entry->bid.amount << " | "

PRINT << entry->bid.fund << endl

entry = entry->nextNode

// Remove a specified bid

//@param bidId The bid id to remove from the list

CREATE void LinkedList::Remove(string bidId)

// Implementing remove logic

DECLARE node\* entry = head

IF (entry == nullptr)

RETURN

ELSE IF (head->bid.bidId == bidId && head->nextNode == nullptr)

head = nullptr

tail = nullptr

DELETE entry

ELSE

WHILE (entry->bid.bidID != bidId)

prevNode = entry

entry = entry-> nextNode

prevNode->nextNode = entry->nextNode

DELETE entry

// Search for the specified bidId

// @param bidId The bid id to search for

CREATE Bid LinkedList::Search(string bidId)

// Implementing search logic

DECLARE node\* entry = head

DECLARE node\* current = new node

current->bid.bidId = ""

WHILE (entry != nullptr)

PRINT<< entry->bid.bidId << endl

IF (entry->bid.bidId == bidId)

RETURN entry->data

entry = entry->nextNode

RETURN current->data

//Returns the current size (number of elements) in the list

MEMBER int LinkedList::Size()

RETURN size

// Static methods used for testing

// Display the bid information

//@param bid struct containing the bid info

CREATE void displayBid(Bid bid)

PRINT << bid.bidId << ": " << bid.title << " | " << bid.amoun << " | " << bid.fund

<< endl

RETURN

// Prompt user for bid information

// @return Bid struct containing the bid info

CREATE Bid getBid()

INITIALIZE Bid bid;

PRINT << "Enter Id: "

GET cin.ignore()

GET getline(cin, bid.bidId)

PRINT << "Enter title: "

GET getline(cin, bid.title)

PRINT << "Enter fund: "

GET >> bid.fund

PRINT << "Enter amount: "

GET cin.ignore()

DECLARE string strAmount;

GET getline(cin, strAmount)

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

RETURN bid

// Load a CSV file containing bids into a LinkedList

// @return a LinkedList containing all the bids read

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

PRINT << "Loading CSV file " << csvPath << endl

// Initialize the CSV Parser

INITIALIZE 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)

INITIALIZE Bid bid

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

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

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

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

// 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

CREATE double strToDouble(string str, char ch)

FUNCTION 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

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

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

// Process command line arguments

USING csvPath, bidKey

CREATE switch (argc)

CASE 2

ASSIGN csvPath = argv[1

ASSIGN bidKey = "98109"

BREAK

CASE 3

ASSIGN csvPath = argv[1]

ASSIGN bidKey = argv[2]

BREAK

DEFAULT

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

ASSIGN bidKey = "98109"

INITIALIZE clock\_t ticks

INITIALIZE LinkedList bidList

INITIALIE Bid bid

DECLARE int choice = 0

WHILE (choice != 9)

PRINT << "Menu:" << endl

PRINT << " 1. Enter a Bid" << endl

PRINT << " 2. Load Bids" << endl

PRINT << " 3. Display All Bids" << endl

PRINT << " 4. Find Bid" << endl

PRINT << " 5. Remove Bid" << endl

PRINT << " 9. Exit" << endl

PRINT << "Enter choice: "

GET >> choice

SWITCH (choice)

CASE 1

INITIALIZE bid = getBid()

INITIALIZE bidList.Append(bid)

INITIALIZE displayBid(bid)

BREAK

CASE 2

INITIALIZE ticks = clock()

CREATE loadBids(csvPath, &bidList)

PRINT << bidList.Size() << " bids read" << endl

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

PRINT << "time: " << ticks << " milliseconds" << endl

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

BREAK

CASE 3

INITIALIZE bidList.PrintList()

BREAK

CASE 4

INITIALIZE ticks = clock()

INITIALIZE bid = bidList.Search(bidKey)

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

IF (!bid.bidId.empty())

PRINT displayBid(bid)

ELSE

PRINT << "Bid Id " << bidKey << " not found." << endl

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

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

BREAK

CASE 5

INITIALIZE bidList.Remove(bidKey)

BREAK

PRINT << "Good bye." << endl

RETURN”

**References**

zyBooks. (2019). Data Structures Essentials with C++. CS-260 Data Structures. Retrieved from, https://learn.zybooks.com/

SNHU. (2020). CS 499 Final Project Guidelines and Rubric. Retrieved from, https://learn.snhu.edu/content/enforced/623492-CS-499-Q1527-OL-TRAD-UG.20EW1/