**Brandon Hobbs**

**Jan. 19, 2022**

**CS-300**

**Module 3 LinkedList**

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

* Class LinkedList
  + Class definition of the LinkedList objects that hold
    - struct Node
      * Consists of a Bid struct
      * Pointer to he next node
    - Append, prepend, PrintList, Remove, Search, Size methods
    - Private members: head, tail, size
* LinkedList()
  + Default constructor that sets the head and tail no NULL
* LinkedList::Append(Bid)
  + A Bid is passed
  + The function then adds the passed Bid as a new Node
  + The new Node is placed at the end of the list and updates the head and tail accordingly
* LinkedList::Prepend(Bid)
  + A Bid is passed
  + The function then adds the passed Bid as a new Node
  + The new Node is placed at the beginning of the list and updates the head and tail accordingly
* LinkedList::PrintList()
  + Function loops through the LinkedList starting at the head and outputs, to the console, 4 values from the Bid struct
* LinkedList::Remove(String)
  + Function starts at the head and searches for the passed String
  + Upon finding the String the Node containing it is freed from memory
  + The head and tail are updated accordingly if either is the Node containing String
* LinkedList::Search(String)
  + Function starts at the head and searches for the passed String
  + Upon finding the String the Node containing it is returned
* LinkedList::Size()
  + A get method used to access the private member, size
* strToDouble
  + Used to convert the CSV file data into useable value
* Bid
  + Struct containing the data
  + Used with the vector that will be sorted
* displayBid
  + Used to send the values contained in the vector to the console
* getBid
  + Allows a user to append or prepend a bid to the list
* loadBids
  + Function used to read in the csv data
  + Can read the csv path in from arguments or use a default path
  + Add parts of the data into the Bid structure and then adds that Bid to the unsorted LinkedList
* main
  + *main* is the primary driver for the application
  + *main* has a menu to allow a user to enter a bid, load the data, view the data, and delete a node and then exit the application
  + *main* also reports the timing each algorithm takes to perform the sort using the *time.h* library

The code was straight forward, especially since the parser was delivered to me. I faced a bit of issue when deleting nodes – especially if that node was the tail. I think I figured it out and learned that I can access linked items by using more than one pointer access call, i.e., head->next->next.

Pseudocode:

**Main** Function()

**Read** cmd arguments

**Store** argument as CSV file path

**If** no cmd arguments load default CSV file path

**Loop** while choice is not equal to ‘9’

**Output** menu

**Get** user input; Store in choice

**Validate** user input

**If** choice is not 1-4 or 9 throw an error

**If** choice equals ‘1’

**Call** getBid to store user entered Bid

**Prepend** or **Append** user Bid

**Call** displayBId()

**If** choice equals ‘2’

**Start** the clock and **store** in ticks

**Call** loadBids and store CSV data in LinkedList *bidList*

**Output** number of records in the CSV file

**Stop** the clock

**Output** the elapsed time needed to read in the CSV file

**If** choice equals ‘3’

**Call** PintList() with *bidList*

**If** choice equals ‘4’

**Start** the clock and store in ticks

**Call** Search() passing a *bidKey* to search for

**Stop** the clock

**Output** the elapsed time needed to find the *bidKey*

**If** choice equals ‘5’

**Call** Remove() passing *bidKey*

**If** Choice equals ‘9’

**Exit** the application

**Output** ‘Good bye’

**End**

**LinkedList::Append(Bid)**

**Create** a new *Node*

**If** *head* is NULL

**Set** head and tail equal to the new *Node*

**Else**

**Set** *tail* pointer equal to the new *Node*

**Set** *tail* equal to the new *Node*

**Set** size to one larger

**End**

**LinkedList::Prepend(*Bid*)**

**Create** a new *Node* for *Bid*

**If** *head’s* next pointer is not NULL

**Set** the new Node as the head

**Set** size to one larger

**Else**

**Call Append()**

**End**

**LinkedList::PrintList()**

**Create** a new Node

**Loop** through the list; starting at the head

**Output** to console: *bidId, title, amount, fund*

**End**

**LinkedList::Search(*String*)**

**Create** a new *Node* pointer called *cursor*

**Set** *cursor* to the head

**Loop** until *cursor* is NULL (end of the list)

**If** the Node at cursor contains a *bidId* equal to *String*

**Return** cursor

**Set** *cursor* equal to the next Node

**End**

**LinkedList::Remove(String)**

**Create** a new *Node* pointer called *cursor* and **Set** to *head*

**Create** a new *Node* called *tempNode*

**Loop** *cursor* from *head* node to length minus 1 nodes

**If** *head* matches *String*

**Set** *head* equal to cursor’s *next* pointer

**Delete** *cursor* memory

**Set** size to one less

**If** *tail* matches *String*

**Set** *tempNode* equal to *cursor’s* next pointer

**Set** *tail* equal to *cursor*

**Set** *cursor’s* next pointer to NULL since its now the tail

**Delete** tempNode

**Set** *size* to one less

If the cursor Node contains a bidId equal to String

**Set** *tempNode* equal to *cursor’s* next pointer

**Set** *cursor* equal to *tempNode*; now 2 copies of *Node* following *cursor*

**Delete** tempNode

**Set** *size* to one less

**Set** *cursor* equal to the *Node* pointed at in *next*

**End**