Final Project

In this semester we learned an array of things. No pun intended. Starting the semester, we got introduced to C ++ programming and went over some basics of how we would use it over the next 8 weeks. We then learned about arrays/vectors, followed by lists and searches, stacks/queues/sorting, hash tables, and finally Trees.

Let’s start with the basics where we learned operations like cout which is used to print text. In this section we also learned about endl which is used to end one line and start text on another. cout << "Menu:" << endl cout << " 1. Enter Bid" << endl; cout << " 2. Display Bid" << endl; cout << " 9. Exit" << endl; cout << "Enter choice: "; cin >> choice; which was used in our Lab 1-3.cpp file to display a menu for the user to use as input. Following cout & endl we were shown struct declares a new type which can be used to define a variable with sub items.

In the second week I learned about Arrays/Vectors concept. An array which is a special variable which has one name stores a list of data items. Other languages use similar constructs but rather calling them arrays, they call them vectors. In our week 2 Lab, I used vectors to structure the hold of data as follows.

Bid bid;

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

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

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

In this same week 2 lab I used looping to make sure the right vector would be returned.

for (unsigned int i = 0; i < theBids.size(); ++i) {displayBid(theBids[1]); } cout << endl; break; }

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

Continuing in week 3 with list and searches our lab put what we learned to the test with being able to search a data structure. It included using operations like find, sort, insert, and delete. Being able to search a list of data structure will come in handy when trying to find the right stored information. It is also used in practices like updating files by adding to or removing existing data. One of the pro’s is using singly linked list is that they usually process and find your data quickly. A con is that it could be complex to code into your program or understanding the logic so that it returns the correct data. LinkedList::LinkedList() {head = tail = nullptr;}

In week 4 we used stacks, queues, and sorting. In our lab we were asked to sort our list by highest bid and alphabetically. Coding these into our program data structure contains specific rules. Using a que is more intuitive of the two. It works like you’re waiting in line to buy somethings. The first element is checked, then the second, and so on. A Stack however allows you to layer your work and search from top to bottom. In our lab, I used the operation of push to move a bid to the end. bids.push\_back(bid); We also used the Quicksort algorithm to organize our bids at selected midpoints. quickSort(bids, mid +1, end);

In week 5 we learned how to use hash table and chaining. A hash table is a data structure which stores unordered items by mapping each item into a location vector/array. Hash Tables contain keys which are stored in a map index. They also consist of elements. Functions used in Hash Tables consists of modulo operator %, chaining, and linear probing. In our week 5 lab we used hash table and a combination of operations learned throughout the weeks. A con to using hash tables is that they don’t accept a “null” input. However, hash tables are usually fast and efficient way for looking up stored data. They provide the best data structure for storing large amounts of data.

struct Node {

Bid bid;

unsigned key;

Node\* next;

// Constructor default

Node() {

key = UINT\_MAX;

next = nullptr;

}

// Takes a bid

Node(Bid aBid): Node() {

bid = aBid;

}

// A bid and a Key

Node(Bid aBid, unsigned aKey) : Node(aBid) {

key = aKey;

}

};

vector<Node> nodes;

unsigned tableSize = DEFAULT\_SIZE;

Lastly, we learned about TREES and how they can make your algorithm more efficient for sorting data orders. TREES consists of Leaf, Internal node, Parent, and Roots. Theses are usually found in a Binary Tree. Some pros to using a Binary Tree Search is that they are designed to speed up the search. We can also do range queries. In our lab we used a Node root to add nodes to the order. Node\* root; void addNode(Node\* node, Bid bid); .

if (root == nullptr){

root = new Node(bid);

} else {

this->addNode(root, bid);