# CS 300 Hash Tables Reflection and Pseudocode

Robert Umland

Robert.Umland@SNHU.EDU

Southern New Hampshire University

**Code Reflection**

This week’s project was to take a .csv file of eBid auction sales and hold all items in the program in a hash table. To check that the table was built correctly, we can display all bids, then search and delete a specific entry from the table. The biggest challenge this week was implementing the hash table. I had never coded a hash table before, so I had to familiarize myself with the process. Then, I made sure to go back and write small tests to ensure everything loaded, the destructor worked properly and the search and remove methods were correct. I think the process of how to implement each new element is becoming easier because we are using the same starting format and adding the new knowledge to a program we are familiar with already.

**Pseudocode**

const unsigned int DEFAULT\_SIZE = 179;

// forward declarations

double strToDouble(string str, char ch);

// Define a structure to hold bid information

structure Bid

string bidId

string title

string fund

double amount

constructor Bid()

amount = 0.0

class HashTable

private:

// Define structures to hold bids

struct Node

Bid bid

unsigned int key

Node \*next

// default constructor

Node()

key = UINT\_MAX

next = nullptr

// initialize with a bid

Node(Bid aBid) : Node()

bid = aBid

// initialize with a bid and a key

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

key = aKey

vector<Node> nodes

unsigned int tableSize = DEFAULT\_SIZE

unsigned int hash(int key)

public:

HashTable()

HashTable(unsigned int size)

virtual ~HashTable()

void Insert(Bid bid)

void PrintAll()

void Remove(string bidId)

Bid Search(string bidId)

size\_t Size()

//Default constructor

HashTable::HashTable()

nodes.resize(tableSize)

// Constructor for specifying size of the table

HashTable::HashTable(unsigned int size)

// Set tableSize to size

this->tableSize = size

// Resize nodes

nodes.resize(tableSize)

//Destructor

HashTable::~HashTable()

nodes.clear()

//Calculate the hash value of a given key.

unsigned int HashTable::hash(int key)

// Calculate a hash value

return key % tableSize

//Insert a bid

void HashTable::Insert(Bid bid)

unsigned int key = hash(stoi(bid.bidId))

if (nodes.at(key).key == UINT\_MAX)

nodes.at(key) = Node(bid, key)

else

Node\* currNode = &nodes.at(key)

// Traverse the linked list to find the end of the chain

while (currNode->next != nullptr)

currNode = currNode->next

currNode->next = new Node(bid, key)

//Print all bids

void HashTable::PrintAll()

for (auto node : nodes)

if (node.key != UINT\_MAX)

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

Node\* nextNode = node.next

while (nextNode != nullptr)

cout << nextNode->bid.bidId << ": " << nextNode->bid.title << " | " << nextNode->bid.amount << " | " << nextNode->bid.fund << endl

nextNode = nextNode->next

//Remove a bid

void HashTable::Remove(string bidId)

unsigned int key = hash(stoi(bidId))

Node\* current = &nodes.at(key)

Node\* previous = nullptr

// Loop through nodes looking for the match

while (current != nullptr && current->key != UINT\_MAX && current->bid.bidId != bidId)

previous = current

current = current->next

// If node found, remove it

if (current != nullptr && current->key != UINT\_MAX && current->bid.bidId == bidId)

if (previous == nullptr)

if (current->next == nullptr)

current->key = UINT\_MAX

current->bid = Bid()

current->next = nullptr

else

Node\* temp = current->next

current->key = temp->key

current->bid = temp->bid

current->next = temp->next

delete temp

else

previous->next = current->next

delete current

//Search for the specified bidId

Bid HashTable::Search(string bidId) {

Bid bid;

// Create the key for the given bid

unsigned int key = hash(stoi(bidId))

Node\* current = &nodes.at(key)

// Loop through nodes looking for the match

while (current != nullptr && current->key != UINT\_MAX)

if (current->bid.bidId == bidId)

return current->bid

current = current->next

return bid

// Static methods used for testing

//Display the bid information to the console (std::out)

void displayBid(Bid bid)

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

return

//Load a CSV file containing bids into a container

void loadBids(string csvPath, HashTable\* hashTable)

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

// Initialize the CSV Parser using the given path

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

// Read and display header row - optional

vector<string> header = file.getHeader()

for (auto const& c : header)

cout << c << " | "

cout << "" << endl

try

// Loop to read rows of a CSV file

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

// Create a data structure and add to the collection of bids

Bid bid

bid.bidId = file[i][1]

bid.title = file[i][0]

bid.fund = file[i][8]

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

// Insert bid into hash table

hashTable->Insert(bid)

catch (csv::Error &e)

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

//Simple C function to convert a string to a double

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

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

// Process command line arguments

string csvPath, bidKey

switch (argc)

case 2:

csvPath = argv[1]

bidKey = "98223"

break

case 3:

csvPath = argv[1]

bidKey = argv[2]

break

default

csvPath = "CS 300 eBid\_Monthly\_Sales.csv"

bidKey = "98223"

// Define a timer variable

clock\_t ticks

// Define a hash table to hold all the bids

HashTable\* bidTable

Bid bid

bidTable = new HashTable()

int choice = 0

while (choice != 9)

cout << "Menu:" << endl

cout << " 1. Load Bids" << endl

cout << " 2. Display All Bids" << endl

cout << " 3. Find Bid" << endl

cout << " 4. Remove Bid" << endl

cout << " 9. Exit" << endl

cout << "Enter choice: "

cin >> choice

switch (choice)

case 1:

// Initialize a timer variable before loading bids

ticks = clock()

// Load the bids

loadBids(csvPath, bidTable)

// Calculate elapsed time and display result

ticks = clock() - ticks

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

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

break

case 2:

bidTable->PrintAll()

break

case 3:

ticks = clock()

bid = bidTable->Search(bidKey)

ticks = 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 4:

bidTable->Remove(bidKey)

break

cout << "Good bye." << endl

return 0