Presentation

Team #8:
Project Title: Customers of Broadcom Tech Company Database

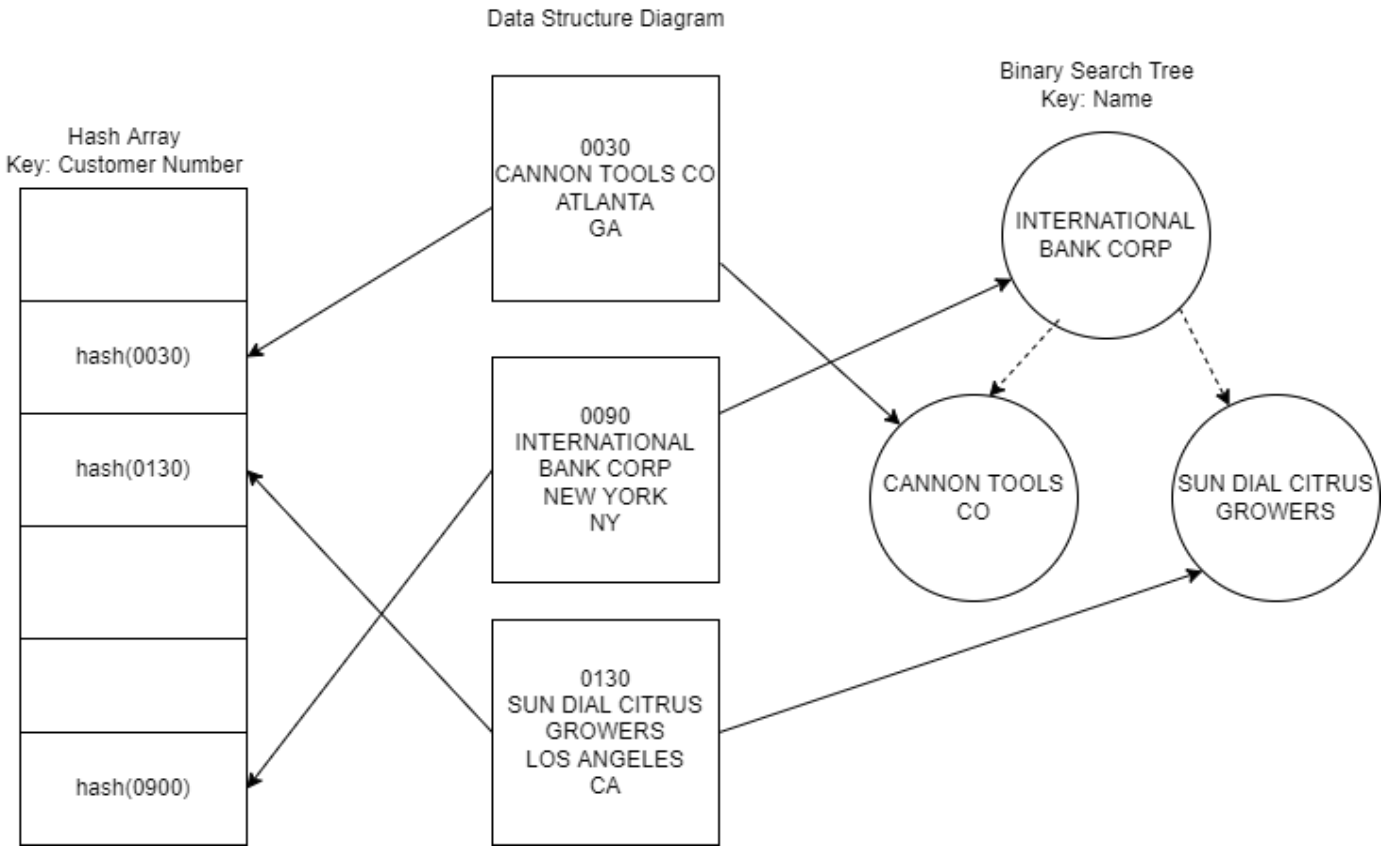
Unit 1 Team Coordination: Kenneth Ao
Unit 2 BST Algorithms: Joel Morancy
Unit 3 Hash List Algorithms: Masato Ishizuka
Unit 4 Screen Output: Pegah Zargarian

1	2	1	2	3	1	2	3	4	1	2	3	4	5	Assignment
X		X			X				X					**Unit 1:** Team Coordination
	X			X			X				X			**Unit 2:** BST Algorithms
X			X			X				X				**Unit 3:** Hash list algorithms
	X	X						X					X	**Unit 4:** Screen Output
	X			X			X					X		**Unit 5:** File I/O
X	X	X	X	X	X	X	X	X	X	X	X	X	X	**Test Plan:** Options and data so that anyone could use it to demonstrate the project. **Presentation Outline:** Activity, duration, etc.
X	X	X	X	X	X	X	X	X	X	X	X	X	X	**Project Documentation:** (see below)

Introduction:
The program reads from a file and fills both a Hash Table and a Binary Search Tree (BST) with Customer objects, using unique Customer Numbers and Names, which can have duplicates.

Primary keys are hashed to provide an index in the Hash Table. The Customer objects are held in Nodes which are then placed into Linked Lists at the index in the hash table. Customer objects are inserted into the BST by sorting the name string.

The program then displays a list of commands and allows users to choose how they want to edit the data structure or to print the data.


Data Structure Diagram
Hash Array
Key: Customer Number
hash(0030)
hash(0130)
hash(0900)
0030
CANNON TOOLS CO
ATLANTA
GA
0090
INTERNATIONAL
BANK CORP
NEW YORK
NY
0130
SUN DIAL CITRUS
GROWERS
LOS ANGELES
CA
Binary Search Tree
Key: Name
INTERNATIONAL
BANK CORP
CANNON TOOLS
CO
SUN DIAL CITRUS
GROWERS

Structure Chart Indented List

main calls the following functions:

- readData()
- menu()
  - printInstructions()
  - add()
    - HashTable.search()
    - BinarySearchTree.insert()
    - HashTable.insert()
  - Stack.push()
  - searchPrimary()
    - Customer.setCusNum
    - HashTable.search()
  - SearchSecondManager()
    - Customer.setName()
    - BinarySearchTree.search()
  - BinarySearchTree.inOrder()
    - vDisplay()
  - writeData()
    - HashTable.populateStack()
    - writeToFile()
  - statistics()
    - HashTable.getLoadFactor()
    - HashTable.totalCollisions()
    - HashTable.longestLinkedList()
  - Stack.pop()
  - BinarySearchTree.insert()
  - HashTable.insert()
  - BinarySearchTree.printTree()
    - iDisplay()
  - printTeam()
  - HashTable.populateStack()
    - getOccupied()
    - HashNode.populateStack()
      - LinkedList.populateStack()
  - vDisplay()

A short description of each person's assignment:

Kenneth Ao:

My assignment was to implement a menu that integrated all parts of the program. I coordinated with each team member to create diagrams and outlines of how the data would be used, which helped us write code that flowed together. I also helped test the program and worked on the presentation part of the project.

Joel Morancy:

My task was to implement the binary search tree and handle the file input/output. The binary search tree takes in a compare function and uses it to organize the data objects into a tree. The file input handles creating the hash table at the appropriate size. The size is found by multiplying the number of lines by 2, and finding the nearest prime number. The nearest prime number is found by finding the remainder of current_size and every integer < current_size. If at any point an even division takes place, the size is incremented. This loops until a current_size is found that can only be divided by itself. File output empties a stack of objects and formats the data to a file.

Masato Ishizuka:
hash table description:
A hash table is a data structure that stores data in units of pairs of keys and corresponding values, and when a key is specified, the corresponding value can be obtained at high speed.

Pegah Zargarian : unit 4

In the unit 4 , there are two functions to manage primary key search and secondary search.

primary search manager is a void function,and it's parameter is Hash Table object, it would

get the customer Id which is a unique key from the user and finds if match exist by calling hash search function and print it's info.

Secondary search manager is also a void function, and it's parameter is Binary Search Tree object, it would get the customer name which is not a unique key from the user and finds if match exists by calling Binary Tree search function and print it's info. If there is more matches prints them.

Display manager is a void function that gets customer object as a parameter and prints list of data sorted by secondary key by calling Inorder function of the tree class.

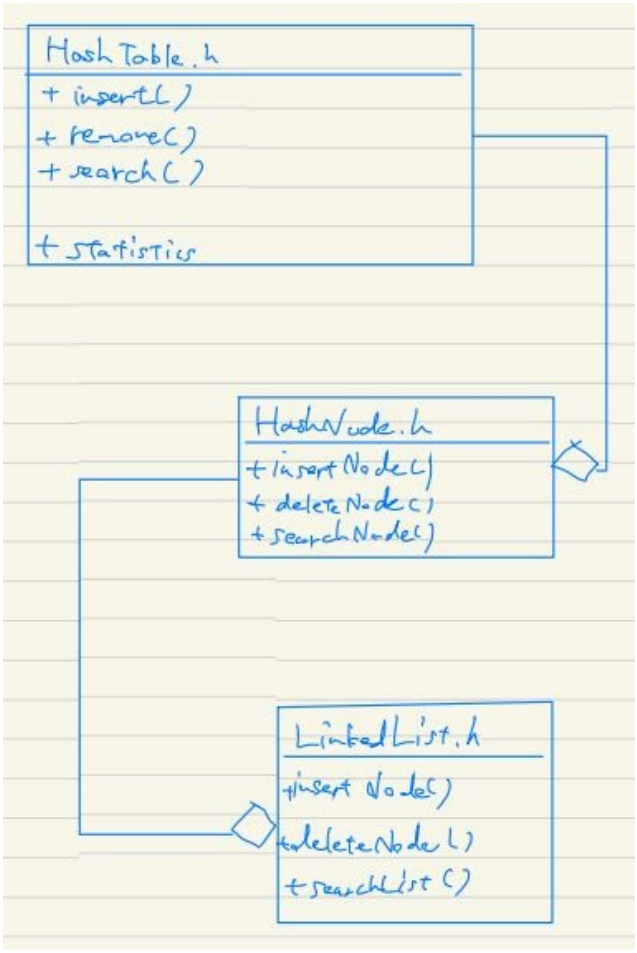
Undo delete would take place if costumer wants to undo deleted item. Generally all deleted items would be stored in a stack which is saving in reverse order. When the user selects to save in a file, stack will be empty.

Hash Function:

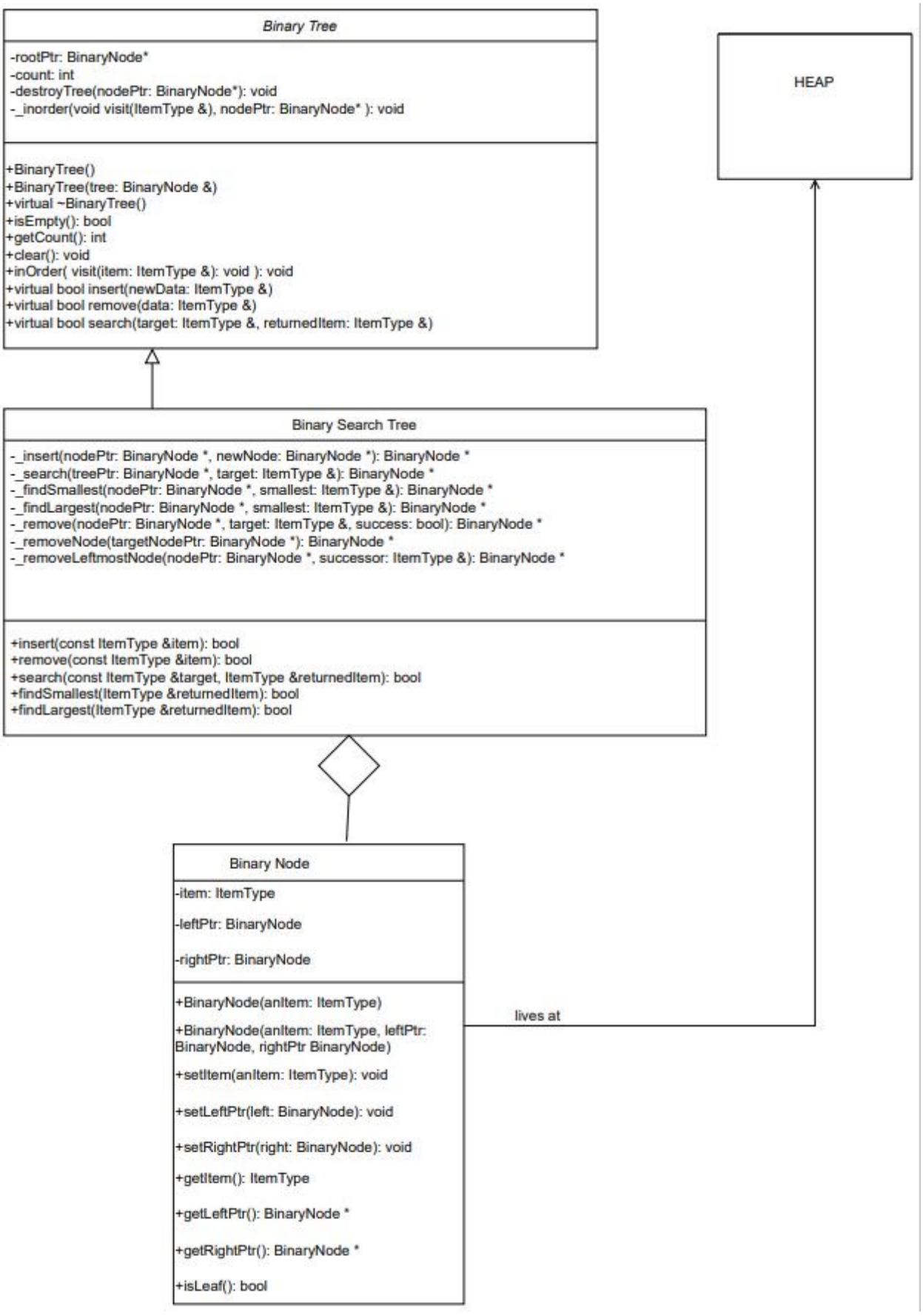
```
int key_to_index(const Customer &key, int size)
{
    string k = key.getName();
    int sum = 0;
    for (int i = 0; k[i]; i++)
        sum += k[i];
    return sum % size;
};
```

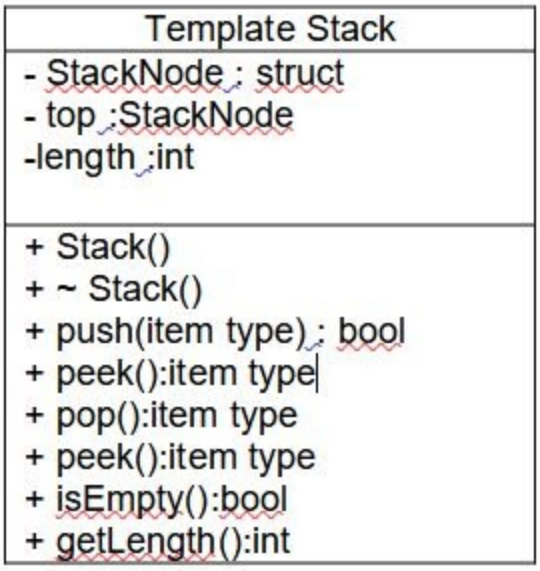
Collision Resolution Method:
Adding the collision into a linked list at the same index

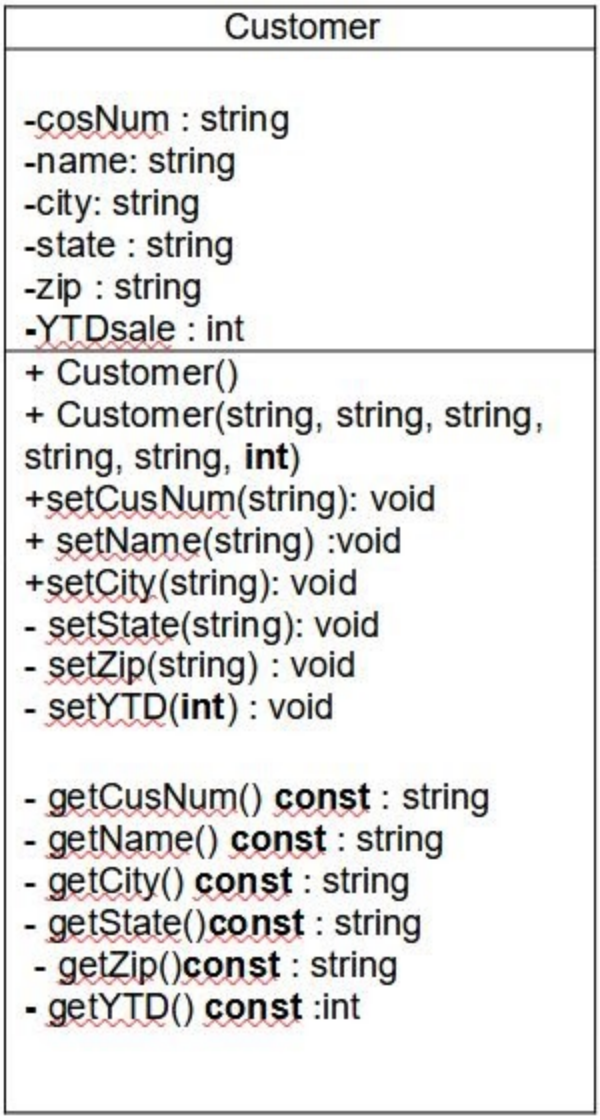
UML Diagrams:



main
+ add(HashTable<Customer>&, BinarySearchTree<Customer>&): void + printInstructions(): void +deleteData(HashTable<Customer>&, BinarySearchTree<Customer>&): Customer +printTeam(): void +searchPrimary(HashTable<Customer>&): void +SecondSearchManager(BinarySearchTree<Customer>&, void visit(Customer&), int compare(Customer, Customer)): void +vDisplay(Customer&): void +iDisplay(Customer&, int): void +statistics(HashTable<Customer>&): void +writeToFile(string filename, Stack<Customer>&): void +writeData(HashTable<Customer>&): void


Binary Tree
-rootPtr: BinaryNode*
-count: int
-destroyTree(nodePtr: BinaryNode*): void
-_inorder(void visit(ItemType &), nodePtr: BinaryNode*): void
+BinaryTree()
+BinaryTree(tree: BinaryNode &)
+virtual ~BinaryTree()
+isEmpty(): bool
+getCount(): int
+clear(): void
+inOrder(visit(item: ItemType &): void): void
+virtual bool insert(newData: ItemType &)
+virtual bool remove(data: ItemType &)
+virtual bool search(target: ItemType &, returnedItem: ItemType &)
HEAP
Binary Search Tree
-_insert(nodePtr: BinaryNode *, newNode: BinaryNode *): BinaryNode *
-_search(treePtr: BinaryNode *, target: ItemType &): BinaryNode *
-_findSmallest(nodePtr: BinaryNode *, smallest: ItemType &): BinaryNode *
-_findLargest(nodePtr: BinaryNode *, smallest: ItemType &): BinaryNode *
-_remove(nodePtr: BinaryNode *, target: ItemType &, success: bool): BinaryNode *
-_removeNode(targetNodePtr: BinaryNode *): BinaryNode *
-_removeLeftmostNode(nodePtr: BinaryNode *, successor: ItemType &): BinaryNode *
+insert(const ItemType &item): bool
+remove(const ItemType &item): bool
+search(const ItemType &target, ItemType &returnedItem): bool
+findSmallest(ItemType &returnedItem): bool
+findLargest(ItemType &returnedItem): bool
Binary Node
-item: ItemType
-leftPtr: BinaryNode
-rightPtr: BinaryNode
+BinaryNode(anItem: ItemType)
+BinaryNode(anItem: ItemType, leftPtr: BinaryNode, rightPtr BinaryNode)
+setItem(anItem: ItemType): void
+setLeftPtr(left: BinaryNode): void
+setRightPtr(right: BinaryNode): void
+getItem(): ItemType
+getLeftPtr(): BinaryNode *
+getRightPtr(): BinaryNode *
+isLeaf(): bool
lives at


Template Stack
- StackNode : struct
- top :StackNode
-length :int
+ Stack()
+ ~ Stack()
+ push(item type) : bool
+ peek():item type
+ pop():item type
+ peek():item type
+ isEmpty():bool
+ getLength():int


Customer
-cosNum : string
-name: string
-city: string
-state : string
-zip : string
-YTDsale : int
+ Customer()
+ Customer(string, string, string, string, string, int)
+setCusNum(string): void
+ setName(string) :void
+setCity(string): void
- setState(string): void
- setZip(string) : void
- setYTD(int) : void
- getCusNum() const : string
- getName() const : string
- getCity() const : string
- getState()const : string
- getZip()const : string
- getYTD() const :int