Programming Assignment #3
Facebook Friends Portal

By: Ayush Gaggar
CS 146, Section 4

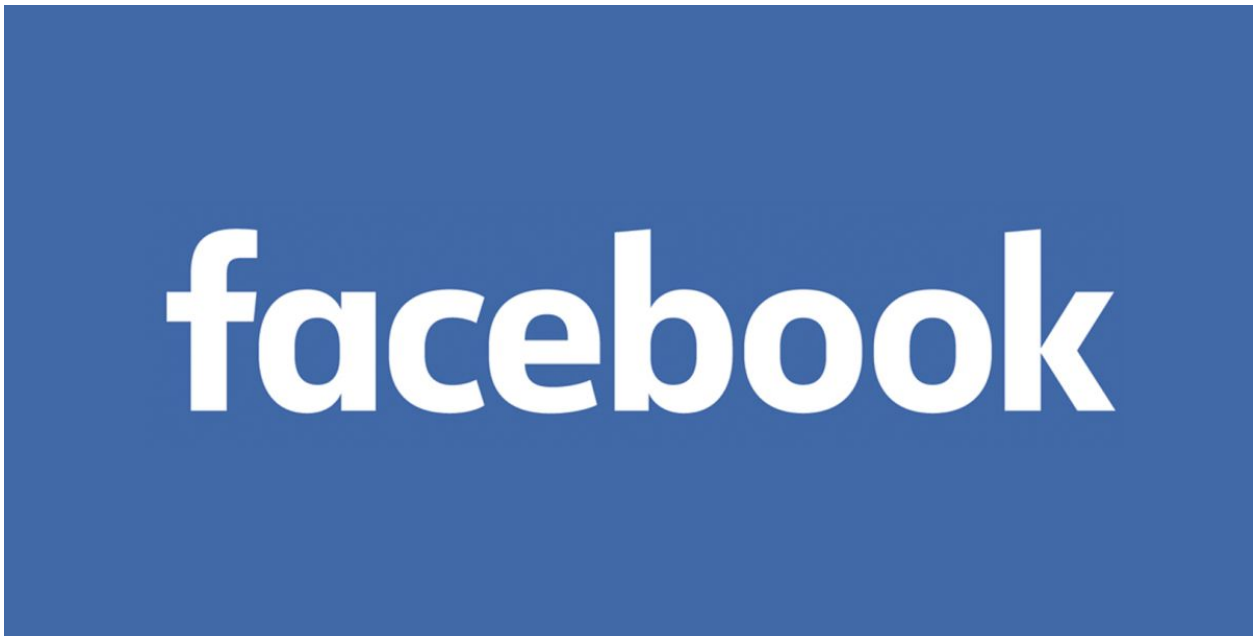
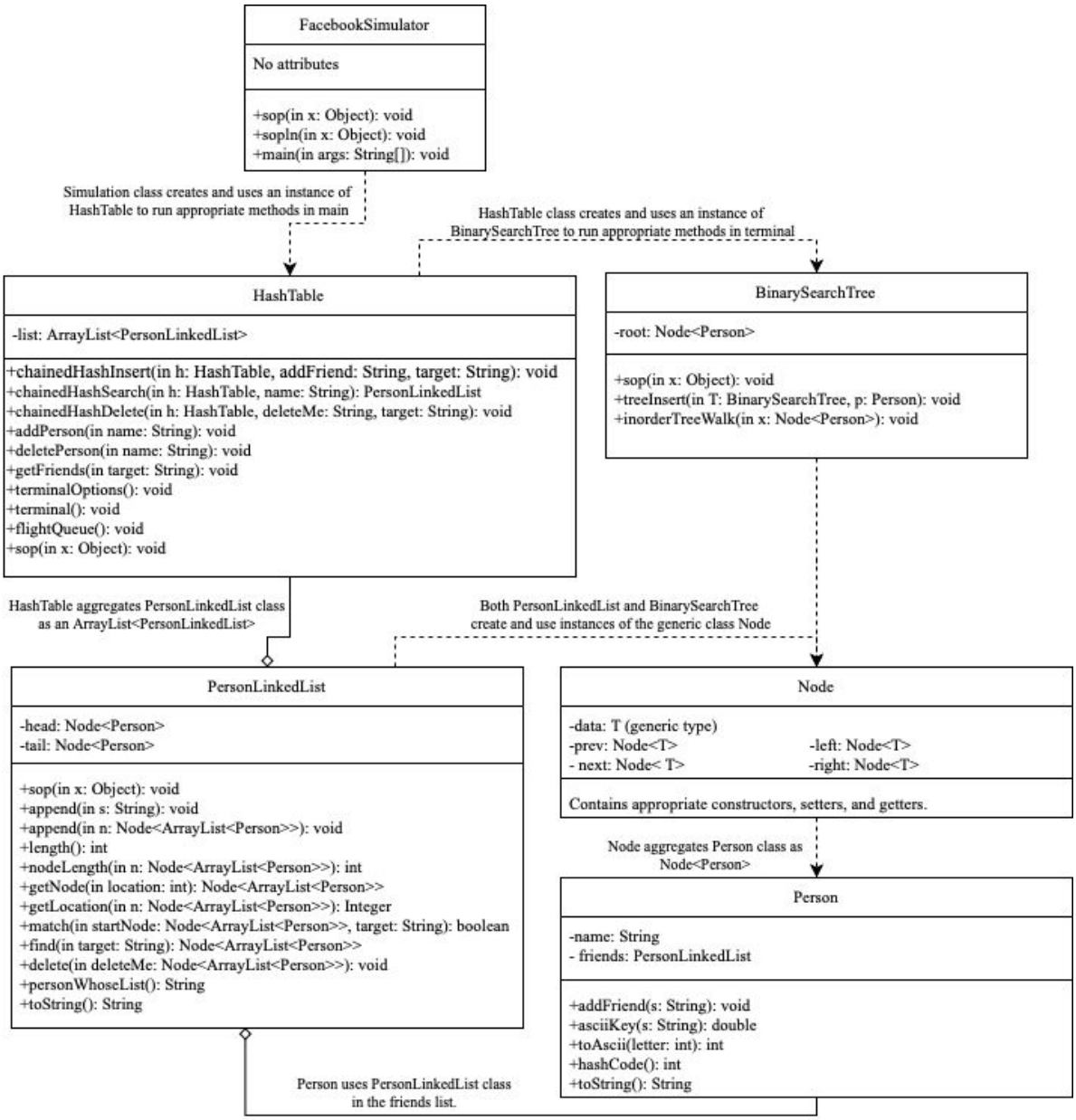


Table Of Contents

Design and Implementation

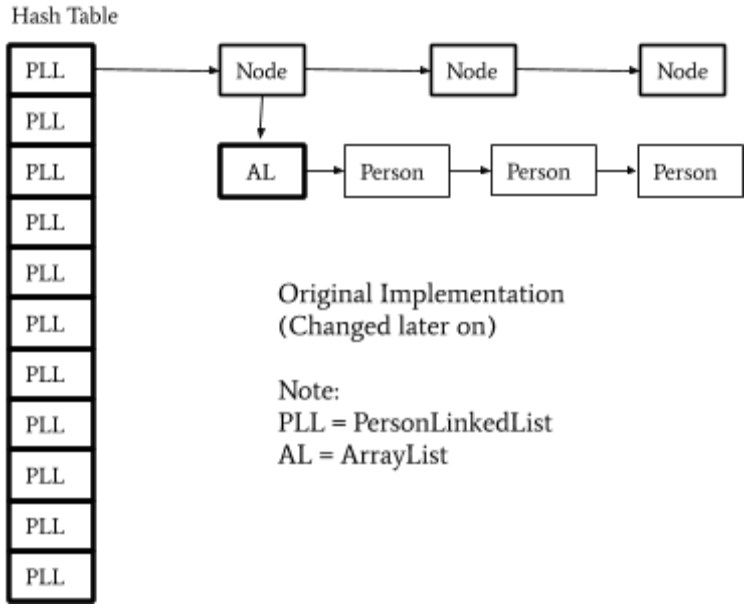
Below is a UML diagram detailing the 6 classes I used, as well as the variables and methods contained within each class. It's important to note that although FacebookSimulator is a class, its only purpose is to create, initialize, and run methods in main so that the user can interact with the simulator.



The methods within the HashTable class are the most crucial parts to the entire programming assignment, so it's critical that appropriate data structures were implemented and called upon by these methods. HashTable essentially calls or uses instances of every other class (besides FacebookSimulator), and so it is crucial that the data structures used or

classes aggregated by said data structures are robust. For the HashTable, I chose to use an ArrayList aggregating objects of type PersonLinkedList (ArrayList<PersonLinkedList>) as the data structure the HashTable methods will use. HashTable doesn't need any other specific functionality besides the simple flexibility provided by Arraylist to get the PersonLinkedList located at a specific index.

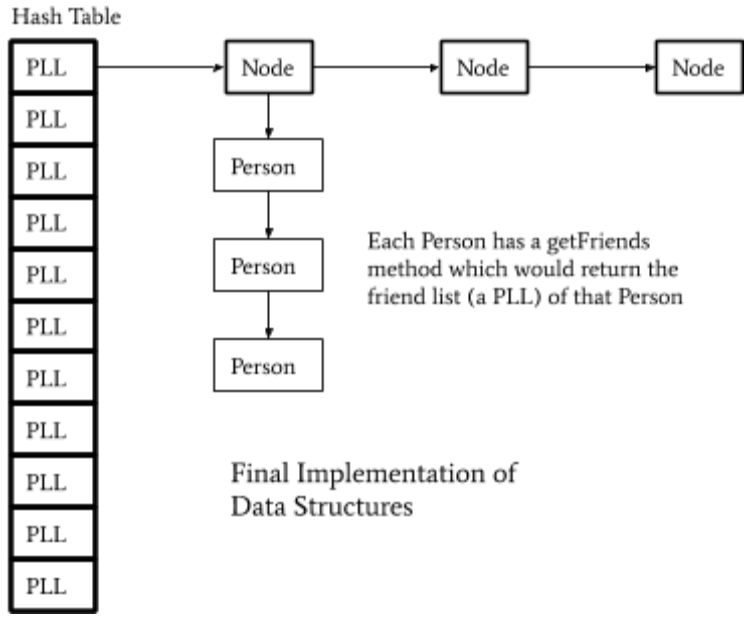
PersonLinkedList needed to be made up of objects that knew the elements that came after and before it; although I could have added left and right methods in the Person class, I decided to make a generic Node class as it could aggregate any object from data structures to Persons. My initial implementation was to have Node aggregate ArrayList<Person>, in that each Node would be the list of friends for that Person, with the first name in the ArrayList the name of the Person who's friend list it was. For visualization purposes, imagine a column representing the Hashtable, with each row of the column containing Node<ArrayList<Person>>; each column of that row would again hold an ArrayList representing the friend list of that Person.



As my program evolved, the benefits of a generic Node class quickly became apparent, as BinarySearchTree also aggregated Node<Person>s, and needed each Node to know its parent, left child, and right child. Thus, it was a good idea for Person class to be separate, and for Node to just aggregate Person class whenever needed.

The methods in PersonLinkedList needed to be able to support the basic functionality of a doubly linked LinkedList, which at its core, are the append and delete methods. My final implementation used the PersonLinkedList class both for the LinkedLists in the hash table,

as well as the LinkedList of friends, instead of using an ArrayList to hold each friend. Each Node contained just a single Person, and that Person had a method which could call its friend list (a PersonLinkedList).



Final Implementation of Data Structures

The implementation of the overall HashTable involved abstract thinking in multiple dimensions, as each slot in the HashTable contained a PersonLinkedList, each PersonLinkedList contained several Nodes, each Node contained several Persons, and finally each Person contained methods to retrieve the Person's name and the Person's friend list (which was again a PersonLinkedList!). Having several layers of data structure made for a more robust program, but served as a unique challenge when programming this assignment.

Description of Classes and Methods

Class 1: Person

Each name in the directory is of type Person, just aggregated within a Node.

- 2 private variables: String name, PersonLinkedList friends
- Constructors:
    - Takes in arguments: String name
        - Initializes name, and initializes friends by creating a new, empty PersonLinkedList
- Methods: all methods are public unless otherwise specified
    - Setters and getters for each private variable (4 total)
    - addFriend: void, no return type
        - Method which appends a String s to the end of the friends PersonLinkedList
    - asciiKey: Returns double
        - Returns the weighted sum of its characters as a double, as specified by the division method.
        - Used by hashCode() method
    - toAscii: Returns int
        - Converts the given letter of this Person's name to its Ascii value. Used by BST class.
    - hashCode: Returns int
        - Returns the int value of the Person's hashCode, as specified by the division method. Calls asciiKey multiple times.
        - Uses Person's name to calculate its hashcode (int between 0 and 10 inclusive).
    - toString: Returns String
        - Returns the Person's name instead of Person's object hashcode.

Class 2: Node (Generic class, can aggregate Generic T)

Generic Node class, containing only getters, setters, and one constructor. Used by Binary Search Tree and PersonLinkedList in order to aggregate ArrayList<Person>.

- 5 private variables: T data, Node<T>prev, Node<T>next, Node<T> left, Node<T> right, Node<T> parent
- Constructors:
    - Takes in arguments: T data (generic)
        - Initializes data
- Methods: all methods are public unless otherwise specified
    - Setters and getters for each private variable (12 total)

Class 3: PersonLinkedList
Custom LinkedList class, where each Link is a Node<Person>.

- 2 private variables: Node<Person> head, Node<Person> tail
- Constructors:
    - No-args constructor
    - Takes in String s as argument
        - Calls overloaded method append on s
- Methods: all methods are public unless otherwise specified
    - Getters only for each private variable (2 total)
    - append(String s): void, no return type
        - This append method takes in a String, creates a new Node<Person>, and then calls the next append method.
    - append(Node<Person> n): void, no return type
        - This method takes in a Node<ArrayList<Person>>, and Appends n to the tail of this list.
        - Checks corner cases before appending.
    - length: Returns int
        - Returns an int equal to the length of this PersonLinkedList.
    - getNode: Node<Person>
        - Returns the Node<Person> located at the specified location in this PersonLinkedList
    - find: Node<Person>
        - If this PersonLinkedList contains a Node whose name is the target, returns that Node. If the target appears multiple times in this list, returns the first occurrence. If the target is not in this list, returns null.
        - Calls match method multiple times.
    - match: boolean
        - Helper method used by find, returns true if the node startNode's Person matches the target string.
    - delete: void, no return type
        - Deletes a Node<ArrayList<Person>> from this PersonLinkedList.
    - toString: String
        - returns the names of every Node in the list,
        - along with each Node's friend list.

Class 4: BinarySearchTree
Custom Binary Search Tree class, used to sort friends alphabetically.

- 1 private variables: Node<Person> root
- Constructors:

    - No explicit constructors
- Methods: all methods are public unless otherwise specified
    - Getters and setters for each private variable (2 total)
    - treeInsert: void, no return type
        - TreeInsert Method, following pseudocode from textbook.
        - Takes in a Person p and creates a Node<Person> z to add to BinarySearchTree T.
        - y is always the parent of x (trailing pointer) as x moves down the tree, as compared by the indicated letter of x's name and z's name. Once x becomes null, z's parent is y, as y is the parent of where z would go. z goes to the right or left of y depending on the Ascii comparison of the indicated letter of x's name and y's name.
    - inorderTreeWalk: void, no return type
        - Inorder-Tree-Walk method, programmed as specified by pseudocode. Iterates through the tree: first prints the left node, then the root, then the right node, as it moves up the tree to the root. Because of how tree-insert is specified, this traversal prints out all the Nodes in alphabetical order.
    - sop: void, no return type
        - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.print

Class 5: HashTable

Custom HashTable class with appropriate methods to meet all functional requirements.

- 1 private variable: ArrayList<PersonLinkedList> list
- Constructors:
    - No-args constructors, which initializes each slot in the HashTable (11 slots total) as new PersonLinkedList
- Methods: all methods are public unless otherwise specified
    - Getters only for each private variable (1 total)
    - chainedHashInsert: void, no return type
        - ChainedHashInsert modifies the pseudocode to include two String parameters.
        - It first checks to make sure the target String is in the directory; if not, the method does nothing. It then checks to see if addFriend is in the directory. If it is not, it adds addFriend as a new user.
        - The for loop then checks to make sure target and addFriend are not already friends. The next line adds addFriend to target's friend list.

Finally, it adds target to addFriend's friend list (as friendships must be mutual).

- chainedHashSearch: Returns PersonLinkedList
    - ChainedHashSearch follows the pseudocode. It calls the find method in PersonLinkedList class to determine whether the String name is in the directory. If not, return null. If it is, then return the friend PersonLinkedList of String name.
- chainedHashDelete: void, no return type
    - ChainedHashDelete modifies the pseudocode to include two String parameters.
    - It first checks to make sure the target String is in the directory; if not, the method does nothing because a Person not in the directory obviously cannot be deleted.
    - It then checks to see if addFriend is in the directory. If it is not, it adds addFriend as a new user.
    - It then iterates through target's friend list to find String deleteMe. If deleteMe is found, it removes Person deleteMe from the target's friend list.
    - Finally, it calls getFriends() method to print target's new friend list.
- addPerson: void, no return type
    - A method not part of functional requirements.
    - Adds a new Person into the directory of Persons held by the HashTable, if Person name does not already exist.
- deletePerson: void, no return type
    - A method not part of functional requirements.
    - Deletes a given Person from the directory of Persons held by the HashTable, if Person name does not already exist.
- getFriends: void, no return type
    - A Helper method which prints out the Friend List of target
- getHashLocation: int
    - Returns the hashtable slot number where String key would be located.
- terminalOptions: void, no return type
    - Calls method sopln to print the options the user can select from
- terminal: void, no return type
    - Directs the user to choose one of the options from terminalOptions, and responds according to the option by calling the correct methods.
    - The print statements embedded within terminalOptions and within this method use method sop, and allow for a seamless and easy

interface for the user to interact with when testing HashTable and BinarySearchTree functionalities.

- sop: void, no return type
    - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.print
- sopln: void, no return type
    - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.println

Class 6: FacebookSimulator

- The FacebookSimulator class is used as a runner class to run the necessary methods from all classes, and simulates the Facebook Portal this assignment replicates.
- No variables, and no explicit constructor calls
- Methods:
    - sop: void, no return type
        - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.print
    - sopln: void, no return type
        - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.println
    - hardcodeAdd: void, no return type
        - Adds the 50 given names to the given HashTable T
    - main: void, no return type
        - The main method is where the HashTable object is created and where the necessary methods are called from.
        - First, main creates an empty HashTable T, which calls hardcodeAdd and stores the 50 names in their correct locations within T.
        - It then iterates through each PersonLinkedList, and adds a random number (0-4) of friends to each Person using HashTable's chainedHashInsert method. The friends being added are also randomly chosen using the random number generator.
        - Main then "boots up" the terminal, and allows users to interact with the simulator, by calling the terminal method in the HashTable class.
        - This application terminates whenever the terminal method terminates, based on user interaction with the Facebook Simulator.

Demonstration of Meeting Functional Requirements

NOTE: For the purpose of conserving space and testing efficiently, examples of screenshot outputs often use a smaller sample size to demonstrate meeting requirements. Every method also works on larger sample sizes, as well as for corner cases.

Functional Requirements as stated in Programming Assignment #3:

1. Create a HashTable (size between 10 and 15)
   The HashTable class has one variable called list, which is an ArrayList aggregating PersonLinkedLists. Although we could have used LinkedLists for each slot in the hash table, using the PersonLinkedLists class I made allowed for greater flexibility as I knew exactly which methods I could call, and what they did. The constructor of HashTable constructs a HashTable of size 11, and initializes a new PersonLinkedList for each slot in the HashTable. A screenshot of the constructor is shown below. A screenshot where all 50 hardcoded people and a random number of friends are placed into their corresponding hash slots is shown under Functional Requirement #2, and further demonstrates how there are 11 total slots (numbered 0-10).

```
//Constructor: Initializes each of the 11 slots of the HashTable as a new
//PersonLinkedList.
public HashTable() {
    for(int i = 0; i < 11; i++) {
        list.add(new PersonLinkedList());
    }
}
```

2. Hard code add the below 50 distinct names into your hashtable:
   Benjamin, Alexander, Samuel, Sebastian, Sophia, Charlotte, Elizabeth, Scarlett, Victoria, James, Daniel, Joseph, Emily, Chloe, Camilia, Liam, Lucas, Wyatt, Harper, Ella, Matthew, Emma, Aria, Jacob, Evelyn, Grace, Logan, Jackson, Carter, Ava, Madison, William, Mason, Michael, Aiden, Henry, Elijah, Oliver, Olivia, Mia, Avery, David, Isabella, Penelope, Noah, Abigail, Sofia, Riley, Ethan, Amelia

   The screenshots below shows a sample of each of the 50 names, as well as each name's friend list, added to their corresponding position in the HashTable based on each name's hashcode. The hashcode method is based off the Division method, and is explained in further detail in Functional Requirement #3. The screenshots below also again prove Functional Requirement #1, in that there are 11 Hash Slots total.
   Note: Friends are randomly generated for each Person.

```
Hash Slot #0
Benjamin: Logan, Madison, Sofia, Sophia, Harper, Oliver,
Alexander: Wyatt, Lucas, Matthew, David, Sophia, Jacob, Ava,
Samuel: Grace, Noah, Riley, James, Harper, Elijah,
Sebastian: Madison, Mason, Daniel, Ethan,
Sophia: Benjamin, Daniel, Alexander, Grace, Charlotte, Harper, Mason,
Charlotte: Lucas, Sophia, Jackson, Matthew,
Elizabeth: Harper, Elijah, William, Mia,
Scarlett: Matthew, Carter, Evelyn, Henry, Mia,
Victoria: Evelyn, Henry, Ava,

Hash Slot #1
James: Samuel, Amelia, Grace, Carter, Isabella,
Daniel: Sebastian, Sophia, Mia, Amelia, Jacob, Aria, Wyatt, Evelyn,
Joseph: Abigail, Emma, Chloe, Madison, Ella, Ethan,
Emily: Mason, Riley, Grace, Penelope, Liam, Avery, Noah,
Chloe: Joseph, Wyatt, Lucas, Aria, Avery, Camilia, Harper, Emma, Mason,
Camilia: Ethan, Matthew, Wyatt, Chloe, Ava, Elijah, Avery,

Hash Slot #2
Liam: Riley, Emily, Aiden, Ella, Ava, Madison, Isabella,
Lucas: Alexander, Charlotte, Chloe, Henry, Matthew, Oliver, Jacob, Carter,
Wyatt: Alexander, Chloe, Camilia, Elijah, Noah, Daniel, Aria, David,
Harper: Elizabeth, Sophia, Samuel, Benjamin, Chloe, Logan,
Ella: Liam, Joseph, Matthew, Jacob, Aiden, Aria, Evelyn, Henry, Amelia,

Hash Slot #3
Matthew: Alexander, Charlotte, Scarlett, Camilia, Lucas, Ella, Amelia, Aria, Abigail,
Emma: Joseph, Amelia, Evelyn, Chloe, Aria, William, Abigail,
Aria: Daniel, Chloe, Wyatt, Matthew, Emma, Abigail, Amelia, Ella, Avery, Michael,

Hash Slot #4
Jacob: Daniel, Lucas, Ella, Carter, Michael, Alexander, Amelia, Henry, Avery, Ethan,
Evelyn: Scarlett, Victoria, Emma, Ella, Daniel, Ethan, Penelope,
Grace: Samuel, Sophia, James, Emily, Mia, Ethan, Mason, Jackson, Michael, Henry, Abigail,

Hash Slot #5
Logan: Benjamin, Harper, Riley, Abigail, David, Jackson,
Jackson: Charlotte, Mia, Sofia, Grace, Logan, Penelope, Noah,
Carter: Scarlett, James, Jacob, Lucas, Madison, Mason, Penelope, Noah, Abigail,
Ava: Victoria, Alexander, Camilia, Liam, Madison, Oliver, Olivia, Noah,
Madison: Benjamin, Sebastian, Joseph, Carter, Ava, Henry, Liam, Amelia,

Hash Slot #6
William: Elizabeth, Olivia, Emma, Amelia, Riley, Aiden, Elijah,
Mason: Sebastian, Emily, Grace, Carter, Chloe, Sophia, Elijah, Abigail, Sofia,
Michael: Jacob, Aria, Mia, Grace, Oliver, Riley, Ethan,
Aiden: Liam, Ella, Avery, William, Ethan, Penelope, Isabella, Amelia,
Henry: Victoria, Lucas, Madison, Scarlett, Grace, Jacob, Ella,

Hash Slot #7
Elijah: Elizabeth, Wyatt, Mason, Camilia, Samuel, Penelope, William, Mia,
Oliver: Lucas, David, Michael, Ava, Benjamin, Sofia,
Olivia: William, Penelope, Ava, Sofia, Avery, David,
Mia: Elizabeth, Daniel, Grace, Jackson, Michael, Elijah, Penelope, Scarlett, Riley,
Avery: Chloe, Aria, Aiden, Olivia, Emily, Camilia, Jacob, Isabella,

Hash Slot #8
David: Alexander, Logan, Oliver, Wyatt, Amelia, Olivia, Sofia,
Isabella: James, Aiden, Avery, Abigail, Liam,
Penelope: Emily, Evelyn, Aiden, Elijah, Olivia, Mia, Carter, Jackson, Amelia,
```

```
Hash Slot #9
Noah: Samuel, Wyatt, Emily, Carter, Ava, Jackson,
Abigail: Joseph, Matthew, Aria, Logan, Isabella, Mason, Carter, Grace, Emma,
Sofia: Benjamin, Jackson, Olivia, Oliver, David, Mason, Ethan, Riley, Amelia,
Riley: Samuel, Emily, Liam, Logan, William, Mia, Amelia, Michael, Sofia,

Hash Slot #10
Ethan: Sebastian, Camilia, Evelyn, Grace, Aiden, Sofia, Joseph, Michael, Jacob,
Amelia: James, Daniel, Matthew, Emma, Aria, Jacob, Madison, William, David, Penelope, Riley, Sofia, Ella, Aiden,
```

3. Design and implement your own Division Method Hash Function

A screenshot of the two methods is shown below. The hashcode method takes the name of the Person and uses the asciiKey method to convert each string to its weighted Ascii value.

The asciiKey method takes a String s and uses the weighted division method from the textbook to convert the String to a natural number. It iterates through the string character by character. For each character, the method casts the character to an int (automatically converting it to its ASCII value), and then multiplying it by a power of 128. Initial characters get a larger exponent of 128, and as you iterate through characters, the exponent to 128 will decrement to 0. The method returns the sum of the weighted ASCII values of each character.

The hashCode method then takes this value, divides it by the size of the hashtable, and returns the remainder. Since the Ascii sums are larger than Integer.MAX_VALUE, I had to use doubles to store the sum. The mod operator in Java does not work with doubles. A workaround for this was to take the floor of the AsciiSum when divided by 11, multiply that number by 11, and then subtract that from the original. This process will always give the remainder between 0 and 10 of the AsciiSum when divided by 11.

```
//Returns the weighted sum of its characters as a double. Used by hashCode() method
public double asciiKey(String s) {
    double sum = 0;
    double ascii = 0;
    for (int i = 0; i < s.length(); i++) {
        ascii = (int) s.charAt(i);
        ascii = (ascii * Math.pow(128, (s.length() - 1 - i)));
        sum = sum + ascii;
    }
    return sum;
}

//Returns the int value of the Person's hashCode, as specified by the division method.
//Uses Person's name to calculate its hashcode.
public int hashCode() {
    int result = 0;
    result = (int) (this.asciiKey(this.getName()) - Math.floor(this.asciiKey(this.getName())/11)*11);
    return result;
}
```

In the screenshot from Functional Requirement 2, an example for Person Ethan would be:
E - (69)*128^4 = 1.852 x 10^10. Sum = 1.852 x 10^10
t - (116)*128^3 = 243,269,632. Sum = 1.852 x 10^10 + 243269632 = 1.8765 x 10^10
h - (104)*128^2 = 1,703,936. Sum = 1.8765 x 10^10 + 1703936 = 1.8767 x 10^10
a - (97)*128^1 = 12,416. Sum = 1.8767 x 10^10 + 12416 = 1.8767 x 10^10
n - (110)*128^0 = 110. Sum = 1.8767 x 10^10 + 110 = 1.8767 x 10^10
Result = 1.8767 x 10^10 - Floor[1.8767 x 10^10/11]*11 = 10
So Ethan's hashCode would be 10, and so would be stored in slot 10 in the hashTable.

4. Define a Person class, with variable's name and a list of friends.
   My Person class had a private variable called name and a private PersonLinkedList called friends, with setters and getters. To create a new Person, all that was needed was the Person's name; a new PersonLinkedList was created within the constructor block which would contain Node<Person> which were friends of this Person.

```
private String name;
private PersonLinkedList friends;

public Person (String name) {
    this.setName(name);
    this.setFriends(new PersonLinkedList());
}
```

5. Store friends of a Person a LinkedList of your design.
   I created a class called PersonLinkedList in order to create a doubly linked LinkedList of my design. The most important methods in any LinkedList are append (to add elements to the tail) and delete (to delete elements from the list). In addition to these two methods, I also had: a find method to return the element with a specified String name, a length method which returned the size of the list, and a getNode method which returned the element at a specified index. The list also needs to know its head and tail.

   I didn't want elements in the PersonLinkedList to be Person's, because then each Person would need to know its "next" and "previous" Person as well. Instead, I created a generic Node class (which BinarySearchTree could then use as well) with the appropriate methods, and aggregated Persons to Node<Person>. Thus, each element (including the head and tail) were of type Node<Person>. A screenshot of the append method is shown below, as it is the most crucial method to the LinkedList. To view the other methods, please open the .java files from the zip folder (see Installation Procedure for more info).

```java
public class PersonLinkedList {
    //Custom LinkedList class, where each Link is a Node<ArrayList<Person>>.

    private Node<Person> head;  // Empty if head and
    private Node<Person> tail;  // tail are null

    //No-args constructor
    public PersonLinkedList() {

    }

    //Constructor to create a new PersonLinkedList. Calls the overloaded,
    //appropriate append method.
    public PersonLinkedList(String s) {
        append(s);
    }

    //This append method takes in a String, creates a new Node<ArrayList<Person>>, and
    //calls the next append method.
    public void append(String s) {
        append(new Node<Person>(new Person(s)));
    }

    // This method takes in a Node<ArrayList<Person>>, and Appends n to the tail of this list.
    public void append(Node<Person> n) {

        // Corner case: empty list.
        //The Corner Case means there are no people in that index of the hash table.
        if (tail == null) {
            n.setPrev(null);
            n.setNext(null);
            head = n;
            tail = n;
        }

        else {
            tail.setNext(n);
            n.setPrev(tail);
            tail = n;
            n.setNext(null);
        }
    }
```

6. Create a hash table which indexes People by name
   This requirement is a continuation of Functional Requirement #2 and #3. The program calls each Person's hashCode method on its name, and uses that hashCode value to slot the Person to its correct slot in the HashTable. The screenshot belows the output, with each Person's hashcode shown in parentheses before the list of their friends. As you can see, all hashcode's of a given number are slotted to that number in the hashtable.

```
Hash Slot #0
Benjamin(0): Aiden, William, Mason, Victoria, Harper, Isabella,
Alexander(0): Madison, Ava, Logan, Carter, Evelyn, Isabella,
Samuel(0): Wyatt, Ella, Lucas, Evelyn, Amelia,
Sebastian(0): Mason, Henry, Michael, Aiden, Harper, Aria, Amelia,
Sophia(0): Sophia, Sophia, Victoria, Elizabeth, Harper,
Charlotte(0): Madison, Ava, Jackson, Victoria, Aria, Evelyn, Isabella, Amelia,
Elizabeth(0): Sophia, Ella, Liam, Lucas, Victoria, Harper,
Scarlett(0): Isabella, David, Harper, Aria, Amelia,
Victoria(0): Sophia, Elizabeth, Charlotte, Victoria, Victoria, Benjamin, Aria, Evelyn,

Hash Slot #1
James(1): Jackson, Ava, Madison, Logan, Aiden, Oliver, Noah,
Daniel(1): Harper, Liam, Lucas, Wyatt, Oliver, Olivia,
Joseph(1): Abigail, Riley, Noah, Sofia, Liam,
Emily(1): Amelia, Ethan, Aiden, Olivia,
Chloe(1): Avery, Mia, Liam, Aiden, Oliver,
Camilia(1): Aria, Emma, Matthew, Liam, Olivia, Noah,
```

```
Hash Slot #2
Liam(2): Elizabeth, Daniel, Joseph, Camilia, Chloe, Carter, Sofia,
Lucas(2): Samuel, Elizabeth, Daniel, Penelope, David, Isabella, Matthew, Sofia,
Wyatt(2): Samuel, Daniel, Evelyn, Jacob, Matthew, Carter,
Harper(2): Daniel, Sophia, Sebastian, Elizabeth, Benjamin, Scarlett, Matthew, Carter, Sofia,
Ella(2): Samuel, Elizabeth, Ethan, Amelia, Matthew,

Hash Slot #3
Matthew(3): Camilia, Harper, Ella, Wyatt, Lucas, Ava, Mason, Henry, Riley,
Emma(3): Camilia, Abigail, Noah, Ava, Mason, Henry, Riley,
Aria(3): Camilia, Sebastian, Charlotte, Victoria, Scarlett, Ava, Mason, Henry, Riley,

Hash Slot #4
Jacob(4): Wyatt, William, Aiden, Henry, Madison, Penelope,
Evelyn(4): Wyatt, Charlotte, Samuel, Alexander, Victoria, Madison, Penelope,
Grace(4): David, Isabella, Penelope,

Hash Slot #5
Logan(5): Alexander, James, Elijah, Avery, Mia,
Jackson(5): Charlotte, James, Penelope, David, Elijah,
Carter(5): Alexander, Harper, Liam, Wyatt, Elijah,
Ava(5): Alexander, Charlotte, James, Aria, Emma, Matthew,
Madison(5): Alexander, Charlotte, James, Evelyn, Jacob, Elijah,

Hash Slot #6
William(6): Benjamin, Jacob, David, Penelope, Mia, Avery,
Mason(6): Benjamin, Sebastian, Aria, Matthew, Emma, Avery,
Michael(6): Sebastian, Isabella, Penelope, Mia,
Aiden(6): Benjamin, Sebastian, Jacob, Emily, Chloe, James, Mia, Avery,
Henry(6): Sebastian, Jacob, Aria, Emma, Matthew, Mia, Avery,

Hash Slot #7
Elijah(7): Logan, Carter, Jackson, Madison,
Oliver(7): Chloe, Daniel, James, Ethan,
Olivia(7): Emily, Camilia, Daniel,
Mia(7): Chloe, Logan, William, Michael, Henry, Aiden, Ethan,
Avery(7): Chloe, Logan, Mason, Aiden, William, Henry,

Hash Slot #8
David(8): Scarlett, Lucas, Grace, Jackson, William, David, David, Isabella,
Isabella(8): Scarlett, Lucas, Grace, Michael, David, Charlotte, Benjamin, Alexander, Abigail,
Penelope(8): Lucas, Grace, Jackson, William, Michael, Evelyn, Jacob, Abigail,

Hash Slot #9
Noah(9): Joseph, Emma, James, Camilia,
Abigail(9): Joseph, Emma, Penelope, Isabella,
Sofia(9): Joseph, Liam, Lucas, Harper,
Riley(9): Joseph, Matthew, Aria, Emma,

Hash Slot #10
Ethan(10): Emily, Ella, Oliver, Mia,
Amelia(10): Emily, Ella, Sebastian, Scarlett, Samuel, Charlotte,
```

7. Using Chained-Hash-Insert, record a person as a new friend.
   The chainedHashInsert method in this program modifies the pseudocode form the textbook to include 2 string parameters: one for the friend to add and one for the person to add the friend to. The method first checks to make sure the target String is in the directory; if not, the method does nothing. It then checks to see if addFriend is in the directory. If it is not, it adds addFriend as a new user. The for loop checks to make sure target and addFriend are not already friends. The next line adds addFriend to target's friend list. Finally, it adds target to addFriend's friend list (as friendships must be mutual). The screenshots below show both the method, and an example of chainedHashInsert being called on a given Person (both if the Person is in the hashtable already, and if the Person is not yet in the hash table).

```java
public void chainedHashInsert(HashTable h, String addFriend, String target) {
    int index = this.getHashLocation(target);
    Node<Person> n = list.get(index).find(target);
    if (n == null) {
        sopln("Person: " + target + " was not in directory.");
        return;
    }
    if (list.get(this.getHashLocation(addFriend)).find(addFriend) == null) {
        this.addPerson(addFriend);
        sopln("Person: " + addFriend + " was first created in directory before friending "
                + "with " + target + ".");
    }
    for (int i = 0; i < n.getData().getFriends().length(); i++) {
        Person p = n.getData().getFriends().getNode(i).getData();
        if (p.getName().equalsIgnoreCase(addFriend)) {
            //sopln(target + " and " + addFriend + " are already friends.");
            return;
        }
    }
    n.getData().getFriends().append(addFriend);
    list.get(getHashLocation(addFriend)).find(addFriend).getData().getFriends().append(target);
}
```

```
1
Please type in the name of the Person you'd like to use Friend methods for.
Wyatt
Person Wyatt has been selected

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
4
Person Wyatt has been selected
Person Wyatt's friends are:
Scarlett, Mia, Elijah, Oliver, Ava, Noah,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
2
Person Wyatt has been selected
Please type in the name of the friend you'd like to add to Wyatt.
Ethan
Person Wyatt's new friend list is:
Scarlett, Mia, Elijah, Oliver, Ava, Noah, Ethan,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
2
Person Wyatt has been selected
Please type in the name of the friend you'd like to add to Wyatt.
Ayush
Person: Ayush was first created in directory before friending with Wyatt.
Person Wyatt's new friend list is:
Scarlett, Mia, Elijah, Oliver, Ava, Noah, Ethan, Ayush,
```

8. Using Chained-Hash-Delete, remove a person from the friend list.
   The chainedHashDelete method in this program modifies the pseudocode form the textbook to include 2 string parameters: one for the friend to delete and one for the person to delete the friend from friend list. The method first checks to make sure the target String is in the directory; if not, the method does nothing because a Person not in the directory obviously cannot be deleted. It then iterates through target's friend friend list to find String deleteMe. If deleteMe is found, it removes Person deleteMe from the target's friend list. Finally, it calls getFriends() method to print target's new friend list. The screenshots below show both the method, and an example of chainedHashDelete being called on a given Person in 3 cases: if the Person deleteMe is in target's friend list, if deleteMe is not in the target's friend list, and if deleteMe does not exist in the hash table.

```
public void chainedHashDelete(HashTable h, String deleteMe, String target) {
    int index = this.getHashLocation(target);
    Node<Person> n = list.get(index).find(target);
    if (n == null) {
        sopln("Person: " + target + " was not found and so could not be deleted.");
        return;
    }
    Node<Person> p;
    boolean found = false;
    for (int i = 0; i < n.getData().getFriends().length(); i++) {
        p = n.getData().getFriends().getNode(i);
            if (p.getData().getName().equalsIgnoreCase(deleteMe)) {
                sopln("Person " + deleteMe + " was deleted from " + target + "'s friend list.");
                n.getData().getFriends().delete(p);
                found = true;
            }
    }
    if (found == true) {
        sopln(target + "'s new Friend List is: ");
        this.getFriends(target);
    }
    else
        sopln(target + " is not friends with " + deleteMe);
}
```

```
Please type in the name of the Person you'd like to use Friend methods for.
Penelope
Person Penelope has been selected

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
4
Person Penelope has been selected
Person Penelope's friends are:
Avery, Emily, James, Chloe,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
3
Person Penelope has been selected
Please type in the name of the friend you'd like to delete from Penelope's list.
Ayush
Penelope is not friends with Ayush
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
3
Person Penelope has been selected
Please type in the name of the friend you'd like to delete from Penelope's list.
Benjamin
Penelope is not friends with Benjamin

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
3
Person Penelope has been selected
Please type in the name of the friend you'd like to delete from Penelope's list.
Chloe
Person Chloe was deleted from Penelope's friend list.
Penelope's new Friend List is:
Avery, Emily, James,
```

9. Using Chained-Hash-Search, search a person to list his/her friend list ChainedHashSearch follows the pseudocode. It calls the find method in class PersonLinkedList to determine whether the String name is in the directory. If not, return null. If it is, then return the PersonLinkedList friend of String name. Many of the screenshots of the terminal output use this method to print the selected Person's friend list, to demonstrate that those Functional Requirements work as well. Below is a screenshot showing ChainedHashSearch's functionality for 2 different Persons.

```
public PersonLinkedList chainedHashSearch(HashTable h, String name) {
    int index = this.getHashLocation(name);
    if (list.get(index).find(name) == null) {
        sopln("Person: " + name + " was not found.");
        return null;
    }
    return list.get(index).find(name).getData().getFriends();
}
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
4
Person Penelope has been selected
Person Penelope's friends are:
Avery, Emily, James,
```

```
4
Person Ethan has been selected
Person Ethan's friends are:
Sebastian, Scarlett, Chloe, Camilia, Liam, Olivia, Mia, Elijah, Avery, Oliver, Amelia,
```

10. Enter two Person's names to verify if they are <u>both</u> friends with each other.

The screenshot below shows the code which checks to see if target (if target exists) is friends with Person with name “friend”. The code calls chainedHashSearch on both target and friend, ensuring neither friend list is null. It then iterates through each list to ensure that both person1 and person2 Friend List contains the other String. Only if they are **both** friends with each other will the program say that they are both friends. Otherwise, the program will print the no statement.

```
if (input == 5) {
    if (target == null)
        sopln("Please select option 1 first.");
    else {
        sopln("Person " + target + " has been selected");
        sopln("Please type in the name of the friend to see if " + target +
                " is friends with them.");
        String friend = br.readLine();
        PersonLinkedList person1 = this.chainedHashSearch(this, target);
        PersonLinkedList person2 = this.chainedHashSearch(this, friend);
        boolean friends = false;
        if (person1 != null && person2 != null) {
            for (int i = 0; i < person1.length(); i++) {
                Person p = person1.getNode(i).getData();
                if (p.getName().equalsIgnoreCase(friend))
                    friends = true;
            }
            if (friends == true) {
                friends = false;
                for (int i = 0; i < person2.length(); i++) {
                    Person p = person2.getNode(i).getData();
                    if (p.getName().equalsIgnoreCase(target))
                        friends = true;
                }
            }
        }
        if (friends == true)
            sopln("Yes, " + target + " and " + friend + " are both friends");
        else
            sopln("N0, " + target + " and " + friend + " are not both friends");
    }
}
```

Continuing with previous screenshots, I will show that Penelope is friends with Avery but not with Chloe, as Chloe was previously removed from Penelope’s friend list. Likewise, Avery is friends with Penelope, but Chloe is not friends with Penelope. Checking to see if a Person is friends with a Person that does not exist will return a statement saying that they are not friends.

```
Person Penelope has been selected
Please type in the name of the friend to see if Penelope is friends with them.
Chloe
N0, Penelope and Chloe are not both friends

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
5
Person Penelope has been selected
Please type in the name of the friend to see if Penelope is friends with them.
Avery
Yes, Penelope and Avery are both friends
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
1
Please type in the name of the Person you'd like to use Friend methods for.
Chloe
Person Chloe has been selected

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
5
Person Chloe has been selected
Please type in the name of the friend to see if Chloe is friends with them.
Penelope
N0, Chloe and Penelope are not both friends
```

```
5
Person Chloe has been selected
Please type in the name of the friend to see if Chloe is friends with them.
Ayush
Person: Ayush was not found.
N0, Chloe and Ayush are not both friends
```

11. Use BST Pseudocode to sort and list a Person's friend list in alphabetical order
    In order to sort using a Binary Search Tree, the only needed methods are treeInsert and inorderTreeWalk. I created a BinarySearchTree class and created a BST only if the user selected option 6 in the terminal, the option that sorts a Person's friend list. The program creates a BST and adds all the elements Node<Person> from the target's friend list into the BST. As long as the BST is not empty (meaning as long as target does have friends), the BST will then sort by calling the inorderTreeWalk method.

    The BST methods follow the pseudocode from the textbook. TreeInsert takes in a Node<Person> z to add to BinarySearchTree T. Node<Person> x and y start as the root and null respectively. Y is always the parent of x as x moves down the tree, as compared by the indicated letter of x's name and z's name. Once x becomes null, z's parent is y, as y is the parent of where z would go. Z goes to the right or left of y depending on the Ascii comparison of the indicated letter of x's name and y's name. The struggle in sorting by the next letter if the current letter was a bit of a challenge, and is further discussed in the Problems Encountered Section.

    Below are screenshots of the code, as well as the output when sorting a given Person's friend list.

```
if (input == 6) {
    if (target == null)
        sopln("Please select option 1 first.");
    else {
        sopln("Person " + target + " has been selected");
        int index = this.getHashLocation(target);
        PersonLinkedList p = list.get(index).find(target).getData().getFriends();
        BinarySearchTree bt = new BinarySearchTree();

        for (int i = 0; i < p.length(); i++) {
            bt.treeInsert(bt, p.getNode(i));
        }
        sop(target + "'s Friend List sorted alphabetically: ");
        if (bt.getRoot() == null)
            sopln(target + " currently has no friends.");
        else
            bt.inorderTreeWalk(bt.getRoot());
        sopln("");
    }
}
```

```
public void treeInsert(BinarySearchTree T, Person p) {
    Node<Person> z = new Node<Person>(p);
    Node<Person> y = null;
    Node<Person> x = T.getRoot();
    int letter = 0;

    while (x != null && letter < (z.getData().getName().length() - 1)) {
        y = x;
        if (z.getData().toAscii(letter) == x.getData().toAscii(letter))
            letter++;
        if (z.getData().toAscii(letter) < x.getData().toAscii(letter)) {
            letter = 0;
            x = x.getLeft();
        }
        else {
            letter = 0;
            x = x.getRight();
        }
    }
    if (letter == (z.getData().getName().length() - 1))
        x = x.getRight();

    letter = 0;
    boolean sameLetter = true;
    z.setParent(y);
    if (y == null)
        T.setRoot(z);
    else {
        while (sameLetter == true) {
            if (z.getData().toAscii(letter) == y.getData().toAscii(letter))
                letter++;
            if (z.getData().toAscii(letter) < y.getData().toAscii(letter)) {
                y.setLeft(z);
                sameLetter = false;
            }
            else {
                y.setRight(z);
                sameLetter = false;
            }
        }
    }
}

/*
 * Inorder-Tree-Walk method as specified by pseudocode.
 * First prints the left node, then the root, then the right node, as it
 * moves up the tree to the root. Because of how tree-insert is specified, this
 * traversal prints out all the Nodes in alphabetical order
 */
public void inorderTreeWalk(Node<Person> x) {
    if (x != null) {
        this.inorderTreeWalk(x.getLeft());
        sop(x.getData() + ", ");
        this.inorderTreeWalk(x.getRight());
    }
}
```

```
1
Please type in the name of the Person you'd like to use Friend methods for.
Noah
Person Noah has been selected

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
4
Person Noah has been selected
Person Noah's friends are:
Alexander, Victoria, Jackson, Elijah, Olivia, Mia, Matthew, Evelyn, Grace, Ethan,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
6
Person Noah has been selected
Noah's Friend List sorted alphabetically: Alexander, Elijah, Ethan, Evelyn, Grace, Jackson, Matthew, Mia, Olivia, Victoria,
```

Overall Design Implementation, demonstrating all functional requirements and potential corner cases for this Programming Assignment.
(Note that screenshots are broken up to fit on the page, yet are still in order; the terminal output is one continuous, smooth output that the user can interact with.)

```
Welcome to Facebook's Portal for Friends!
Remember to Type 1 to select a Person to use Friend methods for, or some methods will not work.
Alternatively, Type 9 to print all available Persons and their friends.
Enjoy!


Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
9
Hash Slot #0
Benjamin: Aria, Madison, Ava, Charlotte, Lucas, Ethan,
Alexander: Matthew, Ethan, Madison, Oliver, Sofia,
Samuel: Charlotte, Aria, Matthew, Madison, Lucas, Mason, Mia,
Sebastian: Henry, Matthew, Lucas, Elizabeth, Ella, Evelyn,
Sophia: Ethan, Riley, Camilia, Sofia, William,
Charlotte: Samuel, Ella, Benjamin, Amelia, Henry, Victoria, Wyatt, Olivia, Mia,
Elizabeth: Evelyn, Lucas, Sebastian, Sofia, James, Aria, Amelia,
Scarlett: Oliver, Mia, Wyatt, Michael, Jacob, Riley,
Victoria: Charlotte, Sofia, Chloe, Riley,

Hash Slot #1
James: David, Riley, Jacob, Elizabeth, Jackson, Michael,
Daniel: Abigail, Grace, William, Riley,
Joseph: Sofia, Henry, David, Mason, Matthew, Elijah,
Emily: Matthew, Sofia, Ella, Logan, Chloe, Olivia,
Chloe: Victoria, Emily, Sofia, Isabella, Abigail, Camilia, Avery,
Camilia: Sophia, Chloe, Oliver, Penelope, Ethan, Madison, Mason,

Hash Slot #2
Liam: Riley, David, Aria, William, Matthew, Evelyn,
Lucas: Sebastian, Elizabeth, Benjamin, Aiden, Samuel, Ethan, Avery,
Wyatt: Scarlett, Grace, Isabella, Madison, Charlotte, Noah,
Harper: Matthew, Aria, Michael,
Ella: Charlotte, Emily, Sebastian, Noah, Henry, Evelyn, Grace, Aiden, Abigail,
```

```
Hash Slot #3
Matthew: Alexander, Samuel, Sebastian, Emily, Harper, Aria, Liam, Oliver, Joseph, Carter, Penelope, Sofia,
Emma: Noah, Abigail, Amelia, Madison, Aiden,
Aria: Benjamin, Samuel, Liam, Harper, Matthew, Aiden, Amelia, Elizabeth, Jacob, Oliver, Penelope,

Hash Slot #4
Jacob: James, Aria, Henry, Scarlett, Ethan, Jackson, Michael,
Evelyn: Elizabeth, Ella, Oliver, Liam, Sebastian, Grace, Logan, Aiden,
Grace: Daniel, Wyatt, Ella, Evelyn, Sofia, Ethan, David,

Hash Slot #5
Logan: Emily, Mia, Mason, Ethan, Evelyn, Ava,
Jackson: Jacob, James, David, Sofia, Oliver, Olivia,
Carter: Mia, Matthew, Sofia, William, Ava, Abigail,
Ava: Benjamin, Carter, Logan, Olivia, Ethan, Aiden,
Madison: Benjamin, Alexander, Samuel, Wyatt, Mia, Mason, Emma, Camilia, William, Penelope, Abigail,

Hash Slot #6
William: Daniel, Liam, Carter, Ethan, Sophia, Madison, Henry, Mia, Sofia,
Mason: Joseph, Logan, Madison, Oliver, Samuel, Camilia, Michael,
Michael: Scarlett, Harper, James, Mason, Mia, Jacob,
Aiden: Lucas, Aria, Ella, Emma, Evelyn, Ava,
Henry: Sebastian, Charlotte, Joseph, Ella, Jacob, William, Penelope,

Hash Slot #7
Elijah: Amelia, Joseph, Isabella,
Oliver: Alexander, Scarlett, Camilia, Matthew, Evelyn, Mason, Avery, Aria, Isabella, Jackson,
Olivia: Ava, Emily, Noah, Charlotte, Jackson, Avery,
Mia: Scarlett, Logan, Carter, Madison, Michael, Charlotte, Samuel, William, Noah, Amelia,
Avery: Oliver, Riley, Olivia, Lucas, Chloe, Isabella,

Hash Slot #8
David: James, Joseph, Liam, Jackson, Grace, Isabella,
Isabella: Chloe, Wyatt, Oliver, Penelope, Elijah, Avery, David,
Penelope: Camilia, Henry, Isabella, Matthew, Abigail, Madison, Aria,

Hash Slot #9
Noah: Ella, Emma, Olivia, Mia, Sofia, Ethan, Wyatt,
Abigail: Daniel, Chloe, Emma, Penelope, Ella, Madison, Carter,
Sofia: Sophia, Elizabeth, Victoria, Joseph, Emily, Chloe, Grace, Jackson, Carter, Noah, Alexander, Amelia, Matthew, William,
Riley: Sophia, James, Liam, Avery, Victoria, Daniel, Scarlett, Ethan,

Hash Slot #10
Ethan: Alexander, Sophia, Camilia, Lucas, Jacob, Grace, Logan, Ava, William, Noah, Riley, Benjamin, Amelia,
Amelia: Charlotte, Emma, Aria, Elijah, Sofia, Mia, Elizabeth, Ethan,


Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
1
Please type in the name of the Person you'd like to use Friend methods for.
Wyatt
Person Wyatt has been selected
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
2
Person Wyatt has been selected
Please type in the name of the friend you'd like to add to Wyatt.
Scarlett
Person Wyatt's new friend list is:
Scarlett, Grace, Isabella, Madison, Charlotte, Noah,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
2
```

```
2
Person Wyatt has been selected
Please type in the name of the friend you'd like to add to Wyatt.
Ethan
Person Wyatt's new friend list is:
Scarlett, Grace, Isabella, Madison, Charlotte, Noah, Ethan,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
2
Person Wyatt has been selected
Please type in the name of the friend you'd like to add to Wyatt.
Ayush
Person: Ayush was first created in directory before friending with Wyatt.
Person Wyatt's new friend list is:
Scarlett, Grace, Isabella, Madison, Charlotte, Noah, Ethan, Ayush,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
3
Person Wyatt has been selected
Please type in the name of the friend you'd like to delete from Wyatt's list.
Ethan
Person Ethan was deleted from Wyatt's friend list.
Wyatt's new Friend List is:
Scarlett, Grace, Isabella, Madison, Charlotte, Noah, Ayush,
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
4
Person Wyatt has been selected
Person Wyatt's friends are:
Scarlett, Grace, Isabella, Madison, Charlotte, Noah, Ayush,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
5
Person Wyatt has been selected
Please type in the name of the friend to see if Wyatt is friends with them.
Ethan
N0, Wyatt and Ethan are not both friends
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
5
Person Wyatt has been selected
Please type in the name of the friend to see if Wyatt is friends with them.
Scarlett
Yes, Wyatt and Scarlett are both friends

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
6
Person Wyatt has been selected
Wyatt's Friend List sorted alphabetically: Ayush, Charlotte, Grace, Isabella, Madison, Noah, Scarlett,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
7
Please type in the name of the person you'd like to add to the directory.
Mark
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
7
Please type in the name of the person you'd like to add to the directory.
Wyatt
Person Wyatt already exist.

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
8
Please type in the name of the person you'd like to delete from the directory.
Wyatt
Person Wyatt was deleted from the directory
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
9
Hash Slot #0
Benjamin: Aria, Madison, Ava, Charlotte, Lucas, Ethan,
Alexander: Matthew, Ethan, Madison, Oliver, Sofia,
Samuel: Charlotte, Aria, Matthew, Madison, Lucas, Mason, Mia,
Sebastian: Henry, Matthew, Lucas, Elizabeth, Ella, Evelyn,
Sophia: Ethan, Riley, Camilia, Sofia, William,
Charlotte: Samuel, Ella, Benjamin, Amelia, Henry, Victoria, Wyatt, Olivia, Mia,
Elizabeth: Evelyn, Lucas, Sebastian, Sofia, James, Aria, Amelia,
Scarlett: Oliver, Mia, Wyatt, Michael, Jacob, Riley,
Victoria: Charlotte, Sofia, Chloe, Riley,

Hash Slot #1
James: David, Riley, Jacob, Elizabeth, Jackson, Michael,
Daniel: Abigail, Grace, William, Riley,
Joseph: Sofia, Henry, David, Mason, Matthew, Elijah,
Emily: Matthew, Sofia, Ella, Logan, Chloe, Olivia,
Chloe: Victoria, Emily, Sofia, Isabella, Abigail, Camilia, Avery,
Camilia: Sophia, Chloe, Oliver, Penelope, Ethan, Madison, Mason,

Hash Slot #2
Liam: Riley, David, Aria, William, Matthew, Evelyn,
Lucas: Sebastian, Elizabeth, Benjamin, Aiden, Samuel, Ethan, Avery,
Harper: Matthew, Aria, Michael,
Ella: Charlotte, Emily, Sebastian, Noah, Henry, Evelyn, Grace, Aiden, Abigail,

Hash Slot #3
Matthew: Alexander, Samuel, Sebastian, Emily, Harper, Aria, Liam, Oliver, Joseph, Carter, Penelope, Sofia,
Emma: Noah, Abigail, Amelia, Madison, Aiden,
Aria: Benjamin, Samuel, Liam, Harper, Matthew, Aiden, Amelia, Elizabeth, Jacob, Oliver, Penelope,

Hash Slot #4
Jacob: James, Aria, Henry, Scarlett, Ethan, Jackson, Michael,
Evelyn: Elizabeth, Ella, Oliver, Liam, Sebastian, Grace, Logan, Aiden,
Grace: Daniel, Wyatt, Ella, Evelyn, Sofia, Ethan, David,
Mark:

Hash Slot #5
Logan: Emily, Mia, Mason, Ethan, Evelyn, Ava,
Jackson: Jacob, James, David, Sofia, Oliver, Olivia,
Carter: Mia, Matthew, Sofia, William, Ava, Abigail,
Ava: Benjamin, Carter, Logan, Olivia, Ethan, Aiden,
Madison: Benjamin, Alexander, Samuel, Wyatt, Mia, Mason, Emma, Camilia, William, Penelope, Abigail,

Hash Slot #6
William: Daniel, Liam, Carter, Ethan, Sophia, Madison, Henry, Mia, Sofia,
Mason: Joseph, Logan, Madison, Oliver, Samuel, Camilia, Michael,
Michael: Scarlett, Harper, James, Mason, Mia, Jacob,
Aiden: Lucas, Aria, Ella, Emma, Evelyn, Ava,
Henry: Sebastian, Charlotte, Joseph, Ella, Jacob, William, Penelope,
Ayush: Wyatt,
```

```
Hash Slot #7
Elijah: Amelia, Joseph, Isabella,
Oliver: Alexander, Scarlett, Camilia, Matthew, Evelyn, Mason, Avery, Aria, Isabella, Jackson,
Olivia: Ava, Emily, Noah, Charlotte, Jackson, Avery,
Mia: Scarlett, Logan, Carter, Madison, Michael, Charlotte, Samuel, William, Noah, Amelia,
Avery: Oliver, Riley, Olivia, Lucas, Chloe, Isabella,

Hash Slot #8
David: James, Joseph, Liam, Jackson, Grace, Isabella,
Isabella: Chloe, Wyatt, Oliver, Penelope, Elijah, Avery, David,
Penelope: Camilia, Henry, Isabella, Matthew, Abigail, Madison, Aria,

Hash Slot #9
Noah: Ella, Emma, Olivia, Mia, Sofia, Ethan, Wyatt,
Abigail: Daniel, Chloe, Emma, Penelope, Ella, Madison, Carter,
Sofia: Sophia, Elizabeth, Victoria, Joseph, Emily, Chloe, Grace, Jackson, Carter, Noah, Alexander, Amelia, Matthew, William,
Riley: Sophia, James, Liam, Avery, Victoria, Daniel, Scarlett, Ethan,

Hash Slot #10
Ethan: Alexander, Sophia, Camilia, Lucas, Jacob, Grace, Logan, Ava, William, Noah, Riley, Benjamin, Amelia, Wyatt,
Amelia: Charlotte, Emma, Aria, Elijah, Sofia, Mia, Elizabeth, Ethan,
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
6
Please select option 1 first.
```

```
Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
1
Please type in the name of the Person you'd like to use Friend methods for.
Ayush
Person Ayush has been selected

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
6
Person Ayush has been selected
Ayush's Friend List sorted alphabetically: Wyatt,

Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
10
```

Installation Procedure

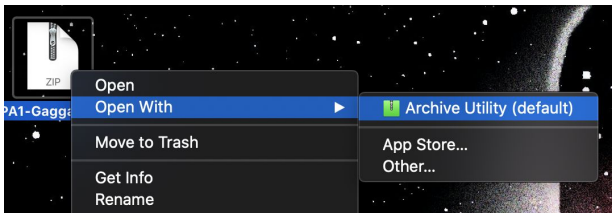
The following installation procedure is for Mac OSX. The procedure is probably very similar in Windows or Linux, but each step might be a little different. Note: A JDK must be pre-installed on the computer.

Step 1: Download the file PA3_Gaggar_Ayush.zip

The process will be easier if you save the zip file to a convenient file location, i.e. Desktop, Downloads, etc. In this procedure, I downloaded and saved the zip file on my Desktop.

Step 2: Unzip PA3_Gaggar_Ayush.zip

Double click to unzip the file. If that doesn't work, right click, click open with, and click the default Archive Utility Opener. Once the folder is unzipped, the zip file can be deleted.



Step 3: Navigating the folder PA3_Gaggar_Ayush

Within the folder will be multiple documents, including this formal report, a folder called pa3_CS146 which contains the java classes themselves, and a jar executable file called pa3.jar

Step 4: Running the jar file on your computer

On Mac, open up Terminal. (If you cannot find it, go to the top right of your screen and click the search icon (the magnifying glass). Type Terminal and open the application).

Note: The below commands might be different if you saved the file at a different location.

Type the following into the terminal screen, and hit enter in between commands:

cd Desktop (OPTIONAL Command to see what files are on your Desktop: ls)
cd PA3-Gaggar_Ayush
java -jar pa3.jar

Upon hitting enter on that last command, the following should begin printing out:

```
ayushs-mbp:~ ayush$ cd Desktop
ayushs-mbp:Desktop ayush$ cd PA3_Gaggar_Ayush
ayushs-mbp:PA3_Gaggar_Ayush ayush$ java -jar pa3.jar
Welcome to Facebook's Portal for Friends!
Remember to Type 1 to select a Person to use Friend methods for, or some methods will not work.
Alternatively, Type 9 to print all available Persons and their friends.
Enjoy!


Please select and input an integer between the options below:
NOTE: 1 must first be selected in order to perform options 2-6 on the correct Person.
1. Select a person to perform methods to.
2. Type in the name of a person to add to the selected person's friend list. (uses Chained-Hash-Insert)
3. Type in the name of a person to delete from the selected person's friend list. (uses Chained-Hash-Delete)
4. Searches and prints each friend from the selected person's friend list. (uses Chained-Hash-Search)
5. Type in the name of the second person to verify if they and the selected person are friends. (Prints yes or no)
6. Sort the selected friend list alphabetically. (Uses BST)
7. Add a NEW Person to directory.
8. Delete an entire Person from directory.
9. Print all Persons along with the LinkedList of their friends.
10. Quit terminal.
```

At this point, the user is free to interact with the software as needed, and the software is fully installed. To re-run the program, simply input "java -jar pa3.jar" into Terminal again!

Problems Encountered During Implementation

- ChainedHashInsert checking for duplicates in the hash table

  One problem I kept running into was my implementation of ChainedHashInsert to add friends to a specific person. The regular pseudocode simply had one line of code - append person to the LinkedList at the specified slot. However, I continuously ran into NullPointerExceptions because of the various corner cases to consider.

  The first corner case is if the person you are trying to add a friend to does not exist. A simple if statement to check if n was in the hashtable was able to resolve this.

```
public void chainedHashInsert(HashTable h, String addFriend, String target) {
    int index = this.getHashLocation(target);
    Node<Person> n = list.get(index).find(target);
    if (n == null) {
        sopln("Person: " + target + " was not in directory.");
        return;
    }
```

  The second corner case is if the friend you are trying to add does not exist. At first, I was using the Person's friend list and checking to see if that was null to determine whether the friend existed or not. However, what I had originally overlooked was the fact that if friend does not exist, then the friend's friend list obviously would not exist either; in other words, the friend could exist and have 0 friends, and the if statement would say that user does not exist, whereas if the person did not exist, then it would not be able to call that person's friend list, causing the NullPointerException. The solution here was to determine whether finding the Person returns a null Node; if it does, ChainedHashInsert just calls the addPerson method to first add the Person to the Hashtable before continuing.

```
if (list.get(this.getHashLocation(addFriend)).find(addFriend) == null) {
    this.addPerson(addFriend);
    sopln("Person: " + addFriend + " was first created in directory before friending "
            + "with " + target + ".");
}
```

  The third corner case popped up as I was trying to randomly add friends to each Person, and uncovered a flaw in the code (more on this in the next Problem Encountered). ChainedHashInsert was not checking whether the Person was already friends with the friend before adding them to their friend list. A for statement iterating through each person in the target's friend list, and then checking to see whether the String friend and a Person's name in the friend list matched. A later addition I made was to also check to see if the String friend and the target's name matched, as I did not want the user to be able to add the friend as a friend of themselves. Finally, ChainedHashInsert added target as addFriend's friend, but also added addFriend as target's friend (because I wanted initial friendships to be mutual), by appending each String to the other's friend list.

```
    for (int i = 0; i < n.getData().getFriends().length(); i++) {
        Person p = n.getData().getFriends().getNode(i).getData();
        if (p.getName().equalsIgnoreCase(addFriend) || target.equalsIgnoreCase(addFriend)) {
            //sopln(target + " and " + addFriend + " are already friends.");
            return;
        }
    }
    n.getData().getFriends().append(addFriend);
    list.get(getHashLocation(addFriend)).find(addFriend).getData().getFriends().append(target);
}
```

Checking for these three corner cases added three blocks of code to the ChainedHashInsert method, but was effective in making ChainedHashInsert work properly.

- Adding random friends to each person
  Although adding random friends to each Person was not part of the functional requirements, I definitely thought that even in a micro version of Facebook, most Persons should still have some friends. In order to add a random Person to a given Person's friend list, my implementation was as follows: first, get a random PersonLinkedList from the hash table (random integer between 0 and 10, the size of the hash table); then, from the random PersonLinkedList, get a random Node<Person> (random integer between 0 and PersonLinkedList.length); finally, add the name of the random Node<Person> as a friend of the current Node<Person>'s name by calling ChainedHashInsert. Repeat this process 4 times for every Person (to give every Person 4 friends), and iterate through every Person by iterating through each Node<Person> in each PersonLinkedList from the hash table. The final implementation is shown below.

```
Random random = new Random();
for (int i = 0; i < 11; i++) {
    PersonLinkedList p = T.getList().get(i);
    for (int j = 0; j < p.length(); j++) {
        for (int friendsToGenerate = 0; friendsToGenerate < 4; friendsToGenerate++) {
            PersonLinkedList r = T.getList().get(random.nextInt(11));
            Node<Person> friend1 = p.getNode(j);
            Node<Person> friend2 = r.getNode(random.nextInt(r.length()));
            T.chainedHashInsert(T, friend1.getData().getName(), friend2.getData().getName());
        }
    }
}
```

At first, I was confused. Why did each Person have more than 4 friends if I was only adding 4 friends to each Person, and why did some Persons have more friends than others? After going through the code, I realized that the chainedHashInsert method made friendships mutual! This meant that, for example, if Ethan added Wyatt, Penelope, Joseph, and Elizabeth as friends, in each of Wyatt's, Penelope's, Joseph's, and Elizabeth's friend list, Ethan was also added as a friend. On top of that, when it was Wyatt's turn to add friends, he would add 4 more on top of Ethan, which meant he would have at least 5 friends, as would Penelope, Joseph, and Elizabeth.

This code wasn't always so clean. Remember how the third corner case in the Problem Encountered with ChainedHashInsert was found when I tried randomly adding friends? My initial solution was to go through each of friend1's and friend2's friend lists to make sure that they weren't already friends with each other inside the for loop. I created a boolean alreadyFriends set as false, which would iterate through friend1's friend list to find friend2; if it found friend2, it would set the boolean as true. If not, it would iterate through friend2's friend list to find friend1 and set alreadyFriends as true if it did. Finally, the program would only call ChainedHashInsert if alreadyFriends was false, and it it was true, it would decrement friendsToGenerate to give friend1 another chance at making a friend.

Very quickly I ran into problems. First, the chances that friend1 and friend2 were already friends was not negligible; especially for the last several Person's (who's friend lists were already quite long from mutual friendships), friendsToGenerate would decrement many times over, meaning the last several Persons could be adding 8 friends *on top of* how many friends they already had! Additionally, the runtime of iterating through both friend lists for all 50 Persons, although is small enough by itself, very quickly compounded to taking longer and longer time as friends were added.

As I was thinking about how to fix this error, going through each line of code to find a better implementation, I finally found it. What if I checked to see if friend1 and friend2 were friends within ChainedHashInsert? This was revolutionary for me! Although it seems a simple enough conclusion, it made my program exponentially better, as ChainedHashInsert could be called by other methods as well, and of course you would want to make sure two friends are not already friends before adding a friendship. Through this process, I was finally able to resolve the Problems of both adding friends to each Person, and fixing ChainedHashInsert to be as flexible as possible.

- Adding parameters to fit pseudocode
  Another challenge which was presented was using the Chained-Hash methods to add or delete friends from a given friends friend list. The pseudocode however, only stated one String (or key) for each of the three methods. Additionally, the pseudocode provided methods to add, search, or delete a Person entirely from the hashtable, whereas in this Programming Assignment, we simply want to add, search for, or delete a Person from a given Person's friend list.

The solution to this problem was twofold: first, the three Hash methods had to delete from a PersonLinkedList of friends rather than from the Hashtable itself (another advantage of using PersonLinkedList for both the Hashtable and the friend list of each Person); second, the Hash methods incorporated an additional parameter to locate which Person these methods should be acting upon.

Both Chained-Hash-Insert and Chained-Hash-Delete take in an additional String parameter in order to locate the Person to which the respective method should operate on. After locating the target String, it then adds or deletes the additional String parameter, depending on the method. Screenshots of the final implementations of the two methods are shown below.

```
public void chainedHashInsert(HashTable h, String addFriend, String target) {
    int index = this.getHashLocation(target);
    Node<Person> n = list.get(index).find(target);
    if (n == null) {
        sopln("Person: " + target + " was not in directory.");
        return;
    }
    if (list.get(this.getHashLocation(addFriend)).find(addFriend) == null) {
        this.addPerson(addFriend);
        sopln("Person: " + addFriend + " was first created in directory before friending "
                + "with " + target + ".");
    }
    for (int i = 0; i < n.getData().getFriends().length(); i++) {
        Person p = n.getData().getFriends().getNode(i).getData();
        if (p.getName().equalsIgnoreCase(addFriend) || target.equalsIgnoreCase(addFriend)) {
            //sopln(target + " and " + addFriend + " are already friends.");
            return;
        }
    }
    n.getData().getFriends().append(addFriend);
    list.get(getHashLocation(addFriend)).find(addFriend).getData().getFriends().append(target);
}
```

I still used the original implementation of the Chained-Hash-Delete and Chained-Hash-Insert methods from the pseudocode in order to add an entirely new Person to the Hash Table, or to delete an entire Person from the HashTable directory. Although this was not technically a part of the functional requirements, I thought this would be a good functionality to add if the user wanted to further interact with the system. Screenshots of these two methods are shown below.

```
public void addPerson(String name) {
    int index = this.getHashLocation(name);
    Node<Person> n = list.get(index).find(name);
    if (n != null) {
        sopln("Person " + name + " already exists.");
        return;
    }
    else {
        list.get(index).append(name);
    }
}
```

```
public void deletePerson(String name) {
    int index = this.getHashLocation(name);
    Node<Person> n = list.get(index).find(name);
    if (n == null) {
        sopln("Person " + name + " does not exist.");
        return;
    }
    list.get(index).delete(n);
    sopln("Person " + name + " was deleted from the directory");
    for (int i = 0; i < 11; i++) {
        PersonLinkedList r = list.get(i);
        for (int j = 0; j < r.length(); j++) {
            PersonLinkedList s = r.getNode(j).getData().getFriends();
            for (int k = 0; k < s.length(); k++) {
                if (name.equalsIgnoreCase(s.getNode(k).getData().getName()) == true)
                    s.delete(s.getNode(k));
            }
        }
    }
    sopln("Person " + name + " was deleted from every Person's friend list.");
}
```

- Sorting BST after just the first letter
  Another prominent challenge I encountered was sorting friends of a particular person alphabetically using a Binary Search Tree. Rather than sorting friends based on their hash code keys, as the pseudocode suggests, this functional requirement calls to sort friends alphabetically. My initial approach was to first add all Node<Person> from the selected Person's friend list into a new Binary Search Tree, and compare the first letter of each Node<Person>'s name. As the pseudocode said, if the values were equal, always assign the node to the right node, and then sort them by calling Iterative Tree Walk. At first glance, this seemed to work. However, upon closer inspection, Persons with the same first letter were sorted seemingly arbitrarily, where sometimes Elizabeth would come before Emily, but then Joseph would come before James.

  In response to this, I first added a compareTo method to compare Persons, which would return the Person which would come before the other. The method either returned whichever String was longer, or compared by adding up the ASCII values of each String. Neither of these implementations consistently sorted alphabetically. I needed a way to compare Nodes based on a given letter, which would increment to the next letter if, when comparing, the current letter of the two Strings were the same.

  Initially, I had an integer called letter, which would increment whenever two Strings were the same and check again. I then modified the treeInsert method to include an int letter, which would then compare the Strings based on the given letter. Easy right? Not so fast... This method ran into issues quite early as the Node moved down the tree. As letter incremented, it would never be reset to compare the next String name from the first letter. As the Node moved down the tree, letter kept incrementing,

which meant Nodes were compared by some middle character and, at times, letter would be larger than the String length itself.

I removed the additional parameter from treeInsert and instead decided to add letter as a parameter every time the method started, as to reset letter to 0. I then added two while statements, one each when comparing the left and right nodes, which incremented letter whenever the current characters were the same. To exit the first while loop, node x must either be null (as the pseudocode says), or letter must have increased past the number of characters in the String itself. If it was the second case, outside the while loop, x was simply set to the x.getRight(). After the first while loop, letter was again reset to 0 in order to compare x with its parent, y. The second while loop included a boolean statement, and only exited the while loop if the character indicated by letter for the two Strings was not equal, and z was set as one of y's children. Finally, inorderTreeWalk was called to print the Nodes in their alphabetical order. This final implementation worked beautifully, and was able to resolve this problem. If this last paragraph seemed too technical or confusing, the screenshot of the code below should provide clarity.

```
public void treeInsert(BinarySearchTree T, Node<Person> z) {
    Node<Person> y = null;
    Node<Person> x = T.getRoot();
    int letter = 0;

    while (x != null && letter < (z.getData().getName().length() - 1)) {
        y = x;
        if (z.getData().toAscii(letter) == x.getData().toAscii(letter))
            letter++;
        if (z.getData().toAscii(letter) < x.getData().toAscii(letter)) {
            letter = 0;
            x = x.getLeft();
        }
        else {
            letter = 0;
            x = x.getRight();
        }
    }
    if (letter == (z.getData().getName().length() - 1))
        x = x.getRight();

    letter = 0;
    boolean sameLetter = true;
    z.setParent(y);
    if (y == null)
        T.setRoot(z);
    else {
        while (sameLetter == true) {
            if (z.getData().toAscii(letter) == y.getData().toAscii(letter))
                letter++;
            if (z.getData().toAscii(letter) < y.getData().toAscii(letter)) {
                y.setLeft(z);
                sameLetter = false;
            }
            else {
                y.setRight(z);
                sameLetter = false;
            }
        }
    }
}
```

Lesson Learned

This programming assignment was one of the most enjoyable programs I worked on this semester. There weren't many challenging concepts being addressed by this Assignment, as the pseudocode for BinarySearchTree and HashTables were both very straightforward. However, this project required creativity in thinking of how to implement these methods to properly function and complete this assignment. The assignment required a deeper analysis than simply blindly following the pseudocode, as many of the requirements addressed the friends of a certain Person, rather than just a Person in the HashTable itself.

When I first read the assignment description, I was daunted. Recreating Facebook?! Even a small part such as just manipulating friends and lists for given users seemed like such a monumental task, an assignment surely out of reach for an amateur programmer such as myself. However, I broke down the assignment into subtasks and analyzed how exactly could I accomplish a specific functional requirement. It was honestly an enjoyable experience to program such an advanced user interface, and elevated my confidence as a software engineer.

The most complex part of this assignment was layering and piecing together various data structures in order to accomplish this assignment. I experimented with various data structures, from the generic ArrayList and LinkedList classes, to aggregating Person classes inside of Node<Person>s, and then making a PersonLinkedList composed of these Nodes. It developed my fundamental knowledge on how these data structures are initially made and how to use them properly. It was critical to think multidimensionally for this assignment, because as a programmer, I'm delving into levels as I move from a Hash slot, to its PersonLinkedList, to a specific Node, to the Person inside that node, and finally to the PersonLinkedList of that Person's friends.

At first, I was able to make quick progress when programming the assignment, and made decent headway to program this version of Facebook. Only upon testing did I realize why this Assignment was more complex: the sheer amount of corner cases that have to be thought of and addressed. I backtracked and went through my code, thinking of it from every aspect. What if the List is empty? What if there's only one Node? What if statements do I need to add in case the selected Node is null or not found? It helped to logically think of all the possibilities that could potentially happen when programming the classes.

Some of the other errors I encountered helped strengthen my understanding of the basics of Java, and elevate my programming skill to where I am more confident about entering the workforce. During this project, I challenged myself to learn something new and widen my scope of how to use Java to its fullest extent. I learned more about Java Generics, and was

able to integrate what I learned from CS 46B, 151, and 146 in order to create a Generic Node class, which was used in PersonLinkedList and BinarySearchTree (and could be used in the future to aggregate even more objects). I also developed my knowledge of IO Streams, and used Input Reader and BufferedReader, and so I programmed the terminal method using IO streams I've grown comfortable using. I also incorporated several try catch blocks embedded within the Terminal to catch any Exceptions that may occur such that the program doesn't terminate upon encountering one.

Programming the terminal method helped me further User Interface (UI) design, and challenged me to think of the system as one that an average user would not be familiar with. Although this was a more basic interaction with the user, using simple System.out.print statements, it still helped me think from a developer's perspective. How do I guide the user to select the appropriate choice, and thus call the appropriate method to achieve the desired goal? How do I handle exceptions if the user inputs invalid arguments? How do I display selection choices in a clear, organized, and legible manner? As I learn about GUI methods in Object Oriented Design and how to build clean code that can be built upon, I made decisions to ensure a more robust and enjoyable experience in the future. After making a selection, the terminal often printed a follow-up statement to further assist the user in inputting a value compatible with the program. Although the terminal menu had 10 options, which often seemed repetitive when interacting with the software, it seemed like a helpful addition. I asked my sister, who is unfamiliar with any kind of programming, to navigate the software; with a little bit of prodding, she enjoyed manipulating the variables within the program. That gave me a huge sense of success and accomplishment as an emerging software engineer!

This programming assignment and class overall has really inspired me to learn more coding languages and further the depth of my knowledge in programming. There are factors to consider that I hadn't considered important when working with smaller sets of data. I find myself thinking, is this the most efficient way to solve this program? What's the runtime of this algorithm, and can I make it more efficient, even if it is just by 1%? I truly love logical problem solving, which is at the heart of Computer Science in general; logically approaching this assignment and tackling it in what I see as the best way possible really appealed to my logical nature. Developing this simulation inspired me to learn new skills and expand on my programming knowledge, in ways not possible through a textbook, classroom format; I believe that with more time, I would create a graphical user interface (GUI) for Facebook, in a format that would be similar to Facebook online! Further functionality could easily be added into this program, such as sorting all the People in the hashtable, deleting friendships mutually, breaking out of option choices midway, etc.

In the future, I'd love to work in the industry with the data structures knowledge I have right now. I'm even more eager to continue learning more advanced topics related to Bioinformatics and Artificial Intelligence; I hope to be able to combine my knowledge from these three fields in Computer Science with the pursuit of my Mechanical Engineering degree as well to work with robotics in the future, ideally in the aerospace or healthcare industries. I find myself constantly thinking about the vast possibilities that this crossover could provide, and how emerging, futuristic technology has the power to save countless lives and shape our standard of living. The first stepping stone to this would be gaining experience through an internship, a step that's proving difficult with today's current situation. I hope to someday take my passion and expertise towards founding a startup focused on improving current healthcare practices using advanced, efficient algorithms controlling robotics. I'm extremely fortunate that this programming assignment, in creating and simulating Facebook, was able to strengthen my grasp of data structures and algorithms while challenging me in an exciting and unique way!