

# **Programming Assignment 3: Micro Version of Facebook**

Jen Luu

CS 146

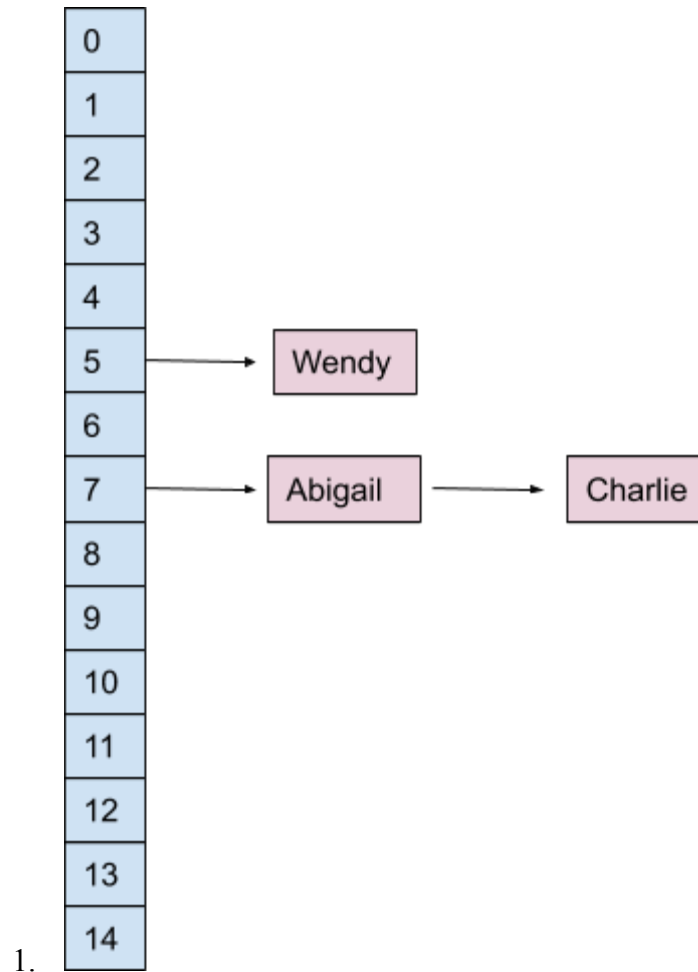
Professor Wu

May 14, 2019

## A. Design and Implementation

### a. Hash table

- i. Private global PersonNode arrays serve as the data structure for the hash tables. Each slot contains a PersonNode object, which contains a Person, its FriendLinkedList, and a pointer to the next PersonNode object. The arrays mimic a large-scale database. An array is an appropriate choice because it is fixed in length and the indexes are easily accessible. Therefore, inserting users into database is simple. The arrays had 15 slots to allow the maximum number of space possible to hold the 50 users. The hash table is similar to Bucket sort in how the elements must be distributed amongst the slots. The only difference is that the hash table does not care about order. As a result, users may be inserted without restrictions. However, collisions will most likely occur since the number of users is far more than the number of slots available. To resolve this issue, the hash table chains elements in the same slot together.
- ii. Illustration:



***b. Hash function: division method***

- i. There are two algorithms to determine where to insert users into the hash table, or database. One of the algorithms is to use the division method to create the hash function. The division method is as follows:
  1.  $h(k) = \text{key} \% m$
- ii. The key is the natural number equivalent of a person's name, and  $m$  is the size of the hash table. Although this method is fast and simple, one of its disadvantages is that it must avoid certain values of  $m$ , such as powers of 2, because the hash function will not consider all the bits of the key.

***c. Hash function: multiplication method***

- i. The other algorithm to decide where to insert users into the hash table is the multiplication method. It goes as follows:
  1.  $h(k) = m (kA \% 1)$
- ii. The same guidelines apply for the key and  $m$ . Let  $A = (5 - 1) / 2$  since it is a good approximation. The advantage of this method is that the value of  $m$  is not critical; however, it is slower than the division method.

***d. FriendLinkedList***

- i. Each Person object is created with a String and a FriendLinkedList. The FriendLinkedList acts a list that records that person's friends, similar to the real version of Facebook. The FriendLinkedList contains PersonNodes that contains both a Person object and a pointer to the next PersonNode.

**B. Classes and methods**

***a. Person***

- i. **public person(String n, FriendLinkedList f)**
  1. This method creates a Person object with a name and a FriendLinkedList. In addition, it converts the name into a natural number.
- ii. **public void makeFriend(Person p)**
  1. A Person object is added into the desired Person's FriendLinkedList.
- iii. **public void unFriend(Person p)**
  1. A Person object is removed from the specified Person's FriendLinkedList.
- iv. **public boolean findFind(Person p)**
  1. This method returns true or false depending on whether or not the Person exists in the database.
- v. **public void getFriendList()**

1. This method prints out the desired Person's FriendLinkedList.
- vi. **public int getKey()**
  1. The Person's natural number value is returned.
- vii. **public int convertToKey(String name)**
  1. Given a name, the method takes out any spaces if there are any and return the length of the concatenated String.
- b. **PersonNode (This class is modeled after the CharNode class in CS 46B)**
  - i. **public PersonNode(Person person)**
    1. This method creates a PersonNode and sets the global Person variable to the one in the argument.
  - ii. **public void setNext(PersonNode next)**
    1. This method sets the global variable next pointer for a PersonNode to the one in the argument.
  - iii. **public PersonNode getNext()**
    1. The global PersonNode variable next is returned.
  - iv. **public Person getPerson()**
    1. A Person object is returned.
- c. **FriendLinkedList (This class is modeled after the CharLinked List class in CS 46B)**
  - i. **public void insertAtHead(Person person)**
    1. A new PersonNode is inserted at the head of a FriendLinkedList. It creates a new PersonNode given the Person object and first checks if the list is empty. If it is, then the head and tail both point to the new PersonNode. The next pointer of it is also set to null.
    2. However, if the list is not empty then the next pointer of the new PersonNode is initially set to the head. The head points to the new PersonNode to complete insertion.
  - ii. **public void deletePerson(Person person)**
    1. This method searches for a Person object in the FriendLinkedList at the head and deletes it. It first checks if the list is empty, and if it is then an error message will print.
    2. If the list only has one element, which is the head, then the next pointer will increment. Since deletion will make the list empty, the next pointer will indirectly point to null.
    3. If the list has more than one element, then the method will begin searching at the head since it is the only accessible element in the singly linked list. As long as the PersonNode that is being checked is not null, the PersonNode will increment until a match is found.

Afterwards, the current PersonNode will reset its next pointer to the next next element.

**iii. public boolean findPerson(Person person)**

1. The boolean will return true when a match is found for the target Person object. As usual, begin the search at the head of the FriendLinkedList. If the list is empty, then an error message will print. If it is not, then the method will go through each PersonNode in the list and compare the Person's name until the target is found. If no such match is found, then the boolean will return false.

**iv. public void printList()**

1. This is a helper method that prints the name of the Person objects in the FriendLinkedList. Each index is named "Slot" followed by the index number. If the index is empty, then the method will just print a slash "/" or the names of the Person objects otherwise.

**d. DivHash**

- i. The class contains a global private PersonNode[] variable that is initialized to size 15. This is the hash table that acts as a database to hold all the users.

**ii. public int divHashFunction(Person person)**

1. The algorithm follows the division method, which extracts the Person's natural number conversion as the key. The method uses the key to execute the function  $h(k) = k \% 15$ , which returns an int. The int specifies which slot the Person will be inserted into the hash table.

**iii. public void register(Person person)**

1. This method creates a new PersonNode and obtains the Person object's key by using the divHashFunction. It simulates the process of registering someone who created an account and holds the information in a database.
2. It goes through two cases. Case 1 checks if the hash table is currently empty. If it is, then the PersonNode will be inserted as the current head and its next point will be set to null.
3. Case 2 handles insertion when the slot is already occupied. Similar to inserting an element in a FriendLinkedList, the PersonNode will be inserted at the head of the slot in the hash table. A temporary current PersonNode will be obtained and the new PersonNode will set its next pointer to it. Afterwards, the head will point to the new PersonNode to complete insertion.

**iv. public void chainedHashInsert(Person original, Person friend)**

1. The purpose of this method is to locate a Person object in the hash table, access FriendLinkedList, and insert a Person object in that. This is to simulate the process of making friends on Facebook and recording that action via a friend list.
  2. To begin, the method will obtain the index value of the original Person object. A temporary current PersonNode is created to access that slot. If the slot is empty, a null error message will print. If it is not, then the current PersonNode will traverse through elements in the slot to find the original Person. A match is found when the name of the current PersonNode is the same as the original Person. Once a match is found, the current PersonNode (now the original PersonNode) will insert the friend Person into that FriendLinkedList through a helper method from the Person class. The original Person's list will be printed as well.
- v. **public void chainedHashDelete(Person original, Person friend)**
1. The purpose of this method is to locate a Person object in the hash table, access FriendLinkedList, and remove Person object from that. This is to simulate the process of unfriending people on Facebook and recording that action via a friend list.
  2. This method follows the same procedure as chainedHashInsert(), but it will remove the friend Person from the original Person's list. The original Person's list will be printed afterwards.
- vi. **public void chainedHashSearch(Person person)**
1. The purpose of this method is to locate a Person object in the hash table and confirm it exists.
  2. This method follows the same procedure as chainedHashInsert(), but the list will not be altered. The friend list of the target Person will be printed to show its current friends.
- vii. **public boolean searchList(PersonNode current, Person original, Person friend)**
1. This is a helper method that traverses through the original Person's FriendLinkedList and returns true if the friend Person is found. The current PersonNode acts as a temporary aid to go through the hash table. The method will return false if the friend Person is not found.
- viii. **public void checkTwoFriends(String one, String two)**
1. The purpose of this method is to look at two Persons' FriendLinkedList to see one appears in the other's list. This mimics the process of ensuring two Persons are friends.

2. Two Person objects will be made from the two String inputs. Each Person's slot index will be obtained to locate where it is in the database
3. Step 1 will require a temporary current PersonNode to go to the index slot of the original Person and find it. The helper method searchList() will be used, and it will return true if a match is found.
4. Step 2 will do execute the same procedure, but for the friend Person.
5. If both search results return true, then the method will print "Yes" to confirm that both Persons are friends. The method will print "No" otherwise.

**ix. public static void printHashTable()**

1. This method prints the contents of the hash table. It will show the elements that are chained.

**e. MultHash**

- i. The class contains a global private PersonNode[] variable that is initialized to size 15. This is the hash table that acts as a database to hold all the users. The purpose of this class is to offer an alternative method for hashing based on the multiplication method.
- ii. All but one method are identical to the methods in the class DivHash.

**iii. public int multHashFunction(Person p)**

1. Given a Person, the method will extract its natural number value and use that as its key. The size of the hash table, 15, will be m. A is a constant that is represented by  $(5 - 1) / 2$  according to Knuth's suggestion.
2. The method will use the following formula to return an int:  $h(k) = m(kA \% 1)$ . This number signifies the index slot of where to insert the Person into the hash table.

**f. MiniFacebook**

- i. This is a class that runs the simulator.

**C. Self-testing screenshots**

- a. *Storing 50 distinct users:*

```

Slot 0: /
Slot 1: /
Slot 2: /
Slot 3: Mia --> Ava --> /
Slot 4: Aria --> Ella --> Emma --> Noah --> Liam --> /
Slot 5: Riley --> Chloe --> Grace --> Sofia --> Avery --> Emily --> Wyatt --> David --> Henry --> Aiden --> Ethan --> Lucas --> Jacob --> Mason --> Logan --> James --> /
Slot 6: Camila --> Harper --> Evelyn --> Amelia --> Sophia --> Olivia --> Carter --> Samuel --> Joseph --> Daniel --> Oliver --> Elijah --> /
Slot 7: Madison --> Abigail --> Jackson --> Matthew --> Michael --> William --> /
Slot 8: Penelope --> Victoria --> Scarlett --> Isabella --> Benjamin --> /
Slot 9: Elizabeth --> Charlotte --> Sebastian --> Alexander --> /
Slot 10: /
Slot 11: /
Slot 12: /
Slot 13: /
Slot 14: /

```

## b. Division method

### i. Inserting a person:

Hello user! Welcome to Minibook.

Enter '1' to register users in the MicroFacebook database. Max: 50 users

Enter '2' to record a person as a new friend.

Enter '3' to remove a person from a friend list.

Enter '4' to search a person's friend list.

Enter '5' to see if two people are friends with each other.

Enter '6' to log out.

1

Please enter in the name of who you wish to register.

Sally

Press 'd' or 'm' to see the database from the division or multiplication method.

Enter 'x' if you wish to move on.

d

Slot 0: /

Slot 1: /

Slot 2: /

Slot 3: /

Slot 4: /

Slot 5: Sally --> /

Slot 6: /

Slot 7: /

Slot 8: /

Slot 9: /

Slot 10: /

Slot 11: /

Slot 12: /

Slot 13: /

1. Slot 14: /

### ii. Recording a friend:

Enter '1' to register users in the MicroFacebook database. Max: 50 users

Enter '2' to record a person as a new friend.

Enter '3' to remove a person from a friend list.

Enter '4' to search a person's friend list.

Enter '5' to see if two people are friends with each other.

Enter '6' to log out.

2

Please specify who you are.

Wendy

Please specify who you want to add as a friend

Wilson

Please type 'd' or 'm' to specify which database to perform this action in.

d

Wendy's friends:

Wilson, Abigail,

Wilson's friends:

Wendy,

1.



iii. Removing a friend:

```
Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

3
Please specify who you are.
Wendy
Please specify who you want to unfriend.
Wilson
Please type 'd' or 'm' to specify which database to perform this action in.
d
Wendy's friends:
Abigail,
Wilson's friends:
```

1.

iv. Searching a person's friend list:

```
Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

4
Please enter in the name of who you wish to search in the database
Wendy
Please type 'd' or 'm' to specify which database to search in.
d
Wendy is in the database.
Wendy's friends:
Winona, Maxwell, Webber, Wilson, Abigail,
```

1.

v. Checking if two people are friends with each other:

```
Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

5
Please enter in the name of the first person
Wendy
Please enter in the name of the second person
Abigail
Please type 'd' or 'm' to specify which database to perform this action in.
d
Yes.
```

1.

c. *Multiplication method*

i. Inserting a person:

Hello user! Welcome to Minibook.

Enter '1' to register users in the MicroFacebook database. Max: 50 users  
Enter '2' to record a person as a new friend.  
Enter '3' to remove a person from a friend list.  
Enter '4' to search a person's friend list.  
Enter '5' to see if two people are friends with each other.  
Enter '6' to log out.

1

Please enter in the name of who you wish to register.

Sally

Press 'd' or 'm' to see the database from the division or multiplication method.  
Enter 'x' if you wish to move on.

m

Slot 0: /  
Slot 1: Sally --> /  
Slot 2: /  
Slot 3: /  
Slot 4: /  
Slot 5: /  
Slot 6: /  
Slot 7: /  
Slot 8: /  
Slot 9: /  
Slot 10: /  
Slot 11: /  
Slot 12: /  
Slot 13: /  
Slot 14: /

1.

ii. Recording a friend:

Enter '1' to register users in the MicroFacebook database. Max: 50 users  
Enter '2' to record a person as a new friend.  
Enter '3' to remove a person from a friend list.  
Enter '4' to search a person's friend list.  
Enter '5' to see if two people are friends with each other.  
Enter '6' to log out.

2

Please specify who you are.

Sally

Please specify who you want to add as a friend

Kitty

Please type 'd' or 'm' to specify which database to perform this action in.

m

Kitty's friends:  
Kitty,  
Sally's friends:  
Kitty,  
Kitty's friends:  
Sally, Kitty,  
Sally's friends:  
Sally, Kitty,

1.

iii. Removing a friend:

```

Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

```

```

3
Please specify who you are.
Sally
Please specify who you want to unfriend.
Kitty
Please type 'd' or 'm' to specify which database to perform this action in.
m
Sally's friends:
Sally, Kitty,
Kitty's friends:
Kitty,

```

1.

iv. Searching a person's friend list:

```

Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

```

```

4
Please enter in the name of who you wish to search in the database
Sally
Please type 'd' or 'm' to specify which database to search in.
m
Sally is in the database.
Sally's friends:
Sally, Kitty,

```

1.

v. Checking if two people are friends with each other:

```

Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.

```

```

5
Please enter in the name of the first person
Sally
Please enter in the name of the second person
Kitty
Please type 'd' or 'm' to specify which database to perform this action in.
m
Yes.

```

1.

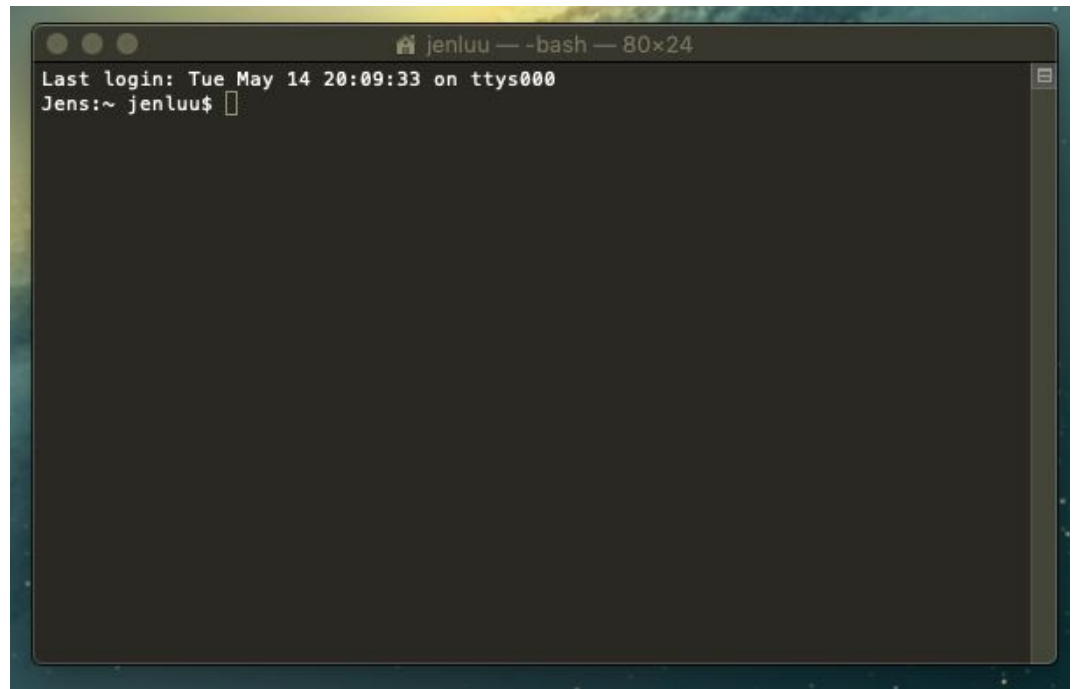
## D. Procedure for running the program

a. Unzip the folder "PA3-Section 7-Luu." The following files should be present:

- i. PA3-Jennifer-Luu.jar
- ii. PA3-Report-Section 7-Luu.pdf
- iii. DivHash.java
- iv. FriendLinkedList.java
- v. MiniFacebook.java
- vi. Person.java

vii. PersonNode.java

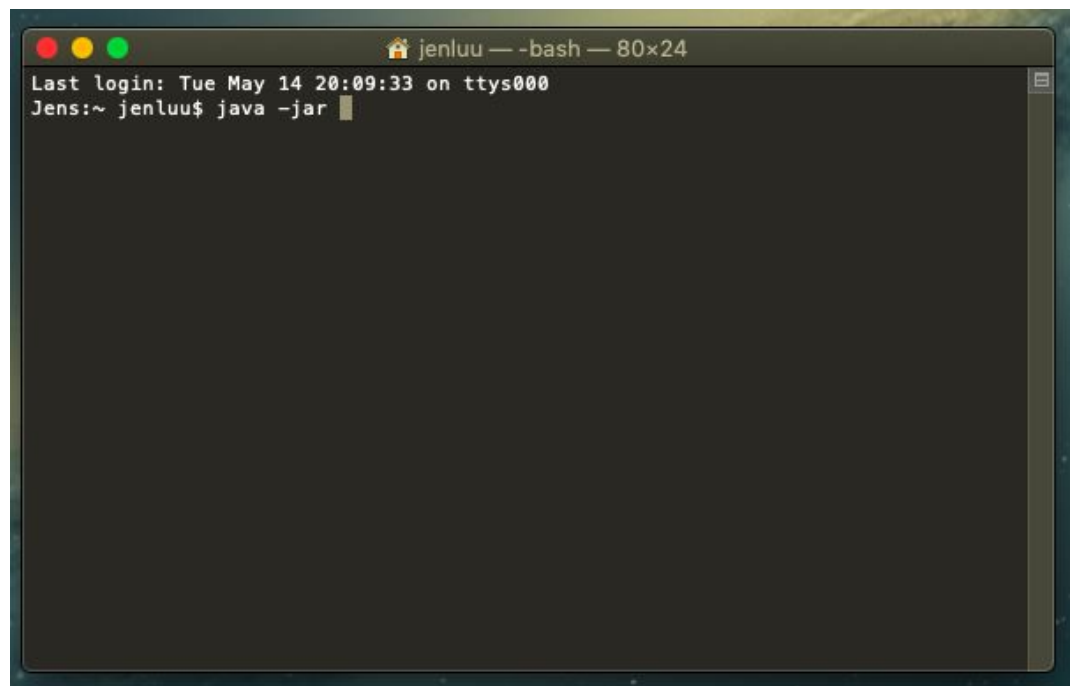
b. Open the command terminal.



```
jenluu — -bash — 80x24
Last login: Tue May 14 20:09:33 on ttys000
Jens:~ jenluu$
```

i.

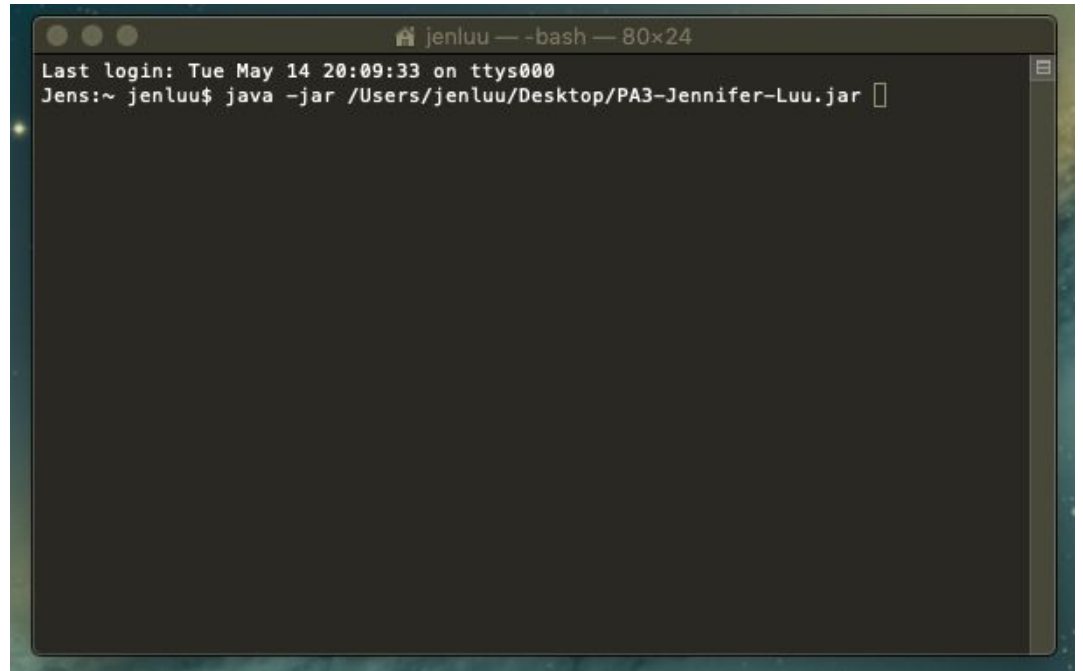
c. Type in “java -jar ”



```
jenluu — -bash — 80x24
Last login: Tue May 14 20:09:33 on ttys000
Jens:~ jenluu$ java -jar
```

i.

d. Drag the jar file into the command terminal.



A terminal window titled "jenluu — -bash — 80x24". The prompt is "Jens:~ jenluu\$". The user has entered the command "java -jar /Users/jenluu/Desktop/PA3-Jennifer-Luu.jar". The output shows the last login time and the command being executed.

```
jenluu — -bash — 80x24
Last login: Tue May 14 20:09:33 on ttys000
Jens:~ jenluu$ java -jar /Users/jenluu/Desktop/PA3-Jennifer-Luu.jar
```

- i.
- e. Press “Enter.”



A terminal window titled "jenluu — java -jar ~/Desktop/PA3-Jennifer-Luu.jar — 80x24". The prompt is "Jens:~ jenluu\$". The user has entered the command "java -jar /Users/jenluu/Desktop/PA3-Jennifer-Luu.jar". The output shows the last login time, the command being executed, and a welcome message. The program then displays a menu of options for the user to choose from.

```
jenluu — java -jar ~/Desktop/PA3-Jennifer-Luu.jar — 80x24
Last login: Tue May 14 20:09:33 on ttys000
Jens:~ jenluu$ java -jar /Users/jenluu/Desktop/PA3-Jennifer-Luu.jar
Hello user! Welcome to Minibook.

Enter '1' to register users in the MicroFacebook database. Max: 50 users
Enter '2' to record a person as a new friend.
Enter '3' to remove a person from a friend list.
Enter '4' to search a person's friend list.
Enter '5' to see if two people are friends with each other.
Enter '6' to log out.
```

- i.
- f. Follow the instructions on the program. It is important to begin by entering all the names that will be used for later operations.
  - i. For example, the user must register the name “Anna” and “Rose” if he/she wishes to record them as friends via option 2.

## E. Problems encountered

- a. *Understanding the relationship between the hash table and Linked List*

- i. At first, I could not understand the correlation between the two data structures. Originally, I misunderstood the prompt and thought that everything was stored in just one of the data structures. If I could store the Person into the hash table, then I could not see the need for the Linked List. When deleting and inserting friends, I thought all of it pertained to the hash table and I would have to modify it. However, this was wrong once I spoke to multiple people. Asking for clarification from peers and the professor because it prevented me from misinterpreting the instructions and building something completely irrelevant. After I received explanations about what the expectations of the project should be, I rephrased them in my own words to confirm.
- ii. The hash table is a database that contains all the users, which is similar to a database that holds all the accounts. Each element in the hash table is a PersonNode. Within the PersonNode is a Person object and pointers to the next PersonNode (for chaining). Within each Person is a name and a Linked List that holds the friends.
- iii. After fully understanding the instructions, my thought process began much clearer.

*b. Forgetting LinkedList properties*

- i. It was a challenge to create a LinkedList from scratch because the last time I took CS 46B was in spring 2018, which was a year. As a result, although I had learned what a LinkedList was I had trouble recalling its properties. To solve this problem, I looked at my old CS 46B homework on LinkedLists and read through the code line by line. I did not move on until I understood each line, and I researched supplemental materials to look at as well. The content began to register with me again when I realized that in a singly linked list, the only available element is the head. Each node contains a pointer to the next reference. Therefore, the only way to access each node and its information was to traverse starting from the head. A for loop would not work because there is no inherent way to obtain the size of the Linked List. Therefore, setting a while loop to iterate until the current node is null was an appropriate method to go through all the elements.

## **F. Lessons learned**

*a. Good programming comes with experience*

- i. To some, having free reign in programming is a creative advantage due to the numerous amounts of solutions. However, I struggle a lot when I do not have structure because I am not very experienced in programming. It was especially difficult when I wasn't sure of what I could and couldn't

do. In addition, remembering the fine details of syntax was challenging. More often than not, I had to refer to several APIs to refresh myself on many coding concepts. Although I felt frustrated and stressed at times, learning from trial and error was one of the most effective ways for me to retain information. Struggling is okay because it is part of the learning process.

*b. Asking for help is crucial*

- i. It is bad practice to assume what the instructions mean. If there is confusion about the expectations of what the project should do, then ask a question early on. This will clear up any misunderstandings and prevent the person from building something that fails to meet the requirements. Even if people are mostly clear about the instructions, it is still useful to clarify and confirm to reinforce concepts. In fact, re-explaining something to another person is also a way to learn. By teaching to other people, the person shows a clear grasp of the subject.