Project Documentation

SN	Name	Student Number			
1	Erwin Johannes	223085227			
2	Undamuje Kahiiko	223052574			
3	Iileka Ndapewa Irja N	220060975			
4	Augustinus Mukoya Mbambi	224086529			
5	Haloloye paulus	224058525			
6	Martin Haindongo	222085770			

Namibian Telecommunications Company Phonebook Application

Namibian Telecommunications Company Phonebook Application

Project Documentation

Introduction

The Namibian Telecommunications Company is seeking an efficient and easy-to-understand phonebook application built using basic linear data structures. The goal is to provide a mobile phonebook that allows users to perform operations such as inserting, searching, deleting, and updating contact information. This project leverages simple data structures like arrays and linked lists to implement these operations. Additionally, the project enables students to explore more advanced techniques like tries and hash tables, potentially enhancing performance.

Core Operations/functions

The phonebook application is designed around the following basic operations:

Main functions

- **Insert**: Add a new contact to the phonebook, storing the contact's name and phone number.

- Search: Find a contact in the phonebook either by name or by phone number.

- Delete: Remove a contact from the phonebook.

- **Update:** Modify an existing contact's details, such as updating their phone number.

Functions and descriptions.

1. insertContact (Phonebook Module)

Purpose : Adds a new contact to the phonebook.
Input: Contact name and phone number.
Output: Contact inserted at the head of the list.
Process: Creates a new Contact object, updates the next pointer, and makes the new contact the head.

2. searchContact (Phonebook Module)

Purpose: Finds a contact by name.
Input: Contact name.
Output: Returns the Contact object if found, or null.
Process: Traverses the list from the head, comparing names until a match is found or the list ends.

Time Complexity: O(n) – Linear search through the list.

3. deleteContact (Phonebook Module)

Purpose: Removes a contact by name.
Input: Contact name.

Output: Deletes the contact and prints confirmation.
Process: Searches the list for the contact, adjusts pointers to remove it.

4. updateContact (Phonebook Module)

Purpose: Updates the phone number of a contact.
Input: Contact name and new phone number.
Output: Updates the phone number if the contact is found.
Process: Uses searchContact to find the contact, then updates the phone number.

5. displayContacts (Phonebook Module)

Purpose: Displays all contacts in the phonebook.
Input: None.
Output: Prints each contact's name and phone number.
Process: Traverses the contact list, printing each contact's details.

6. sortContacts (Phonebook Module)

Purpose: Sorts contacts alphabetically by name.
Input: None.
Output: Reorders contacts alphabetically.
Process: Applies a sorting algorithm to reorder the linked list based on contact names.

7. exit

Leave the application when selected

2.1 Data Structures Used

We have chosen to use the following data structures:

- **Linked List**: A sequential data structure where each element points to the next. It allows efficient insertions and deletions but requires traversal to locate elements.

3. **Implementation Details**

The following section describes the main modules and their respective functions.

3.1 Module

1. **Phonebook Module**: This module contains the main functionality for managing phonebook operations such as inserting, searching, deleting, and updating contacts.

3.2 Classes and Methods

Here’s how we can implement the core operations using Java:

```java
// Contact class representing a contact in the phonebook
class Contact {
    String name;
    String phoneNumber;
    Contact next;  // For linked list implementation

    public Contact(String name, String phoneNumber) {
        this.name = name;
        this.phoneNumber = phoneNumber;
        this.next = null;
    }
}

// Phonebook class to manage contacts
class Phonebook {
    private Contact head;
    // Insert a new contact
    public void insert(String name, String phoneNumber) {
        Contact newContact = new Contact(name, phoneNumber);
        if (head == null) {
            head = newContact;
        } else {
            Contact current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newContact;
        }
        System.out.println("Contact added: " + name);
    }

    // Search for a contact by name
    public Contact search(String name) {
        Contact current = head;
        while (current != null) {
            if (current.name.equals(name)) {
                System.out.println("Contact found: " + current.name + ", " +
current.phoneNumber);
                return current;
            }
            current = current.next;
        }
        System.out.println("Contact not found: " + name);
        return null;
    }

    // Delete a contact by name
    public void delete(String name) {
        if (head == null) {
            System.out.println("Phonebook is empty.");
            return;
        }

        if (head.name.equals(name)) {
            head = head.next;
            System.out.println("Contact deleted: " + name);
            return;
        }

        Contact current = head;
        while (current.next != null) {
            if (current.next.name.equals(name)) {
                current.next = current.next.next;
                System.out.println("Contact deleted: " + name);
                return;
            }
            current = current.next;
        }
```

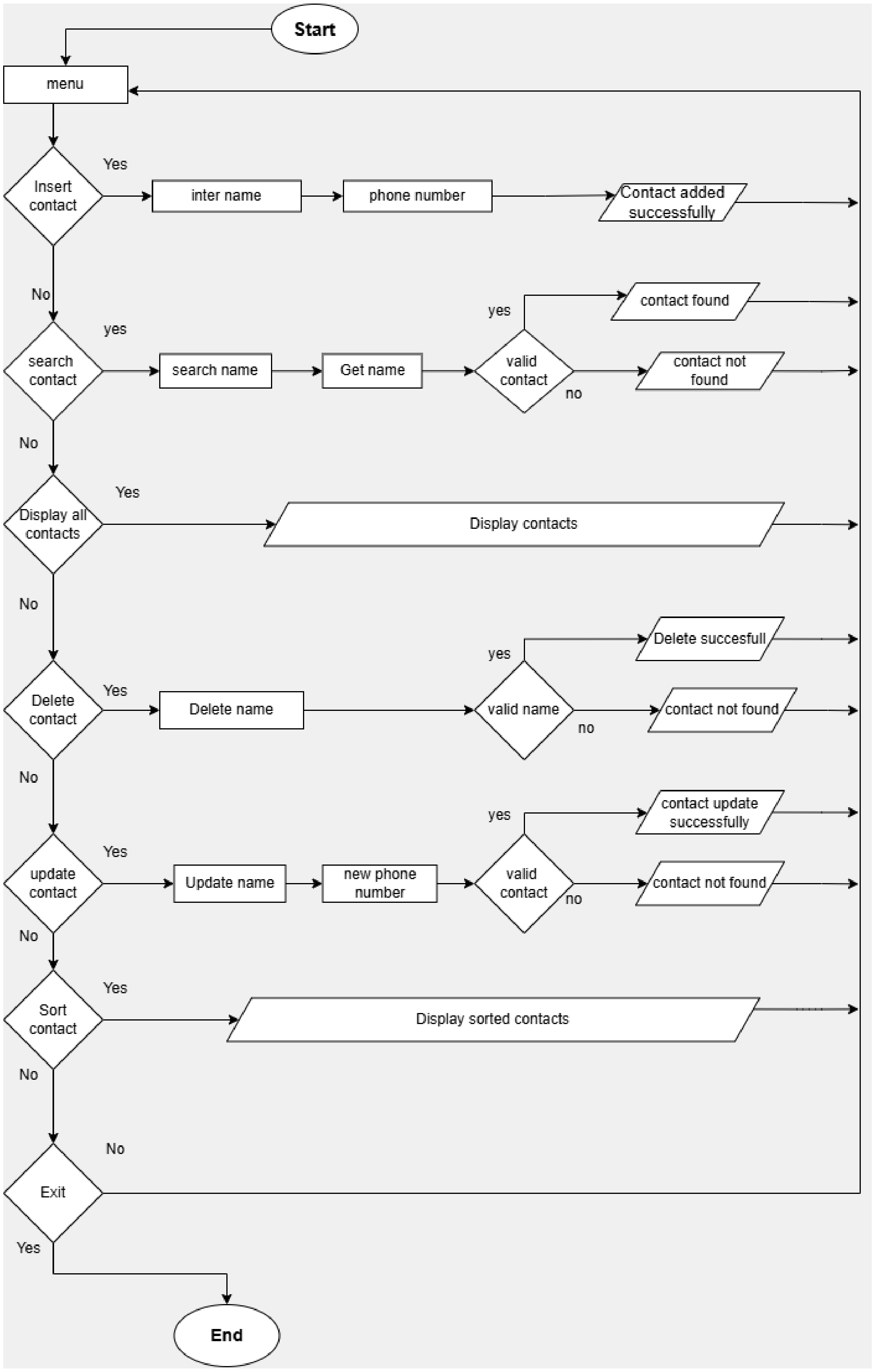
```
        System.out.println("Contact not found: " + name);
    }

    // Update an existing contact's phone number
    public void update(String name, String newPhoneNumber) {
        Contact contact = search(name);
        if (contact != null) {
            contact.phoneNumber = newPhoneNumber;
            System.out.println("Contact updated: " + name + " -> " + newPhoneNumber);
        }
    }
}
```

4. **Time complexity**

- Insertion: O(n) in the worst case, as we must traverse to the end of the list.
- Search: O(n) since each contact must be checked sequentially.
- Deletion: O(n) because we must traverse the list to find the contact.
- Update : O(n), as it relies on searching for the contact first.

While not the most efficient, these operations are simple to implement and perform reasonably well for small datasets.

5. **FLAWCHART**


Start
menu
Insert contact
Yes
inter name
phone number
Contact added successfully
No
search contact
yes
search name
Get name
valid contact
yes
contact found
no
contact not found
No
Display all contacts
Yes
Display contacts
No
Delete contact
Yes
Delete name
valid name
yes
Delete succesfull
no
contact not found
No
update contact
Yes
Update name
new phone number
valid contact
yes
contact update successfully
no
contact not found
No
Sort contact
Yes
Display sorted contacts
No
Exit
No
Yes
End

6. PSEUDOCODE

```
START MENU:
 DISPLAY "1. Insert Contact"
 DISPLAY "2. Search Contact"
 DISPLAY "3. Display All Contacts"
 DISPLAY "4. Delete Contact"
 DISPLAY "5. Update Contact"
 DISPLAY "6. Sort Contacts"
DISPLAY "7. Exit"

GET userChoice
 IF userChoice == 1 THEN INSERT CONTACT DISPLAY "Enter name:"
GET name
DISPLAY "Enter phone number:"
GET phoneNumber
ADD contact (name, phoneNumber)
 DISPLAY "Contact added successfully."
GO TO MENU
 ELSE IF userChoice == 2 THEN
 SEARCH CONTACT DISPLAY "Enter name to search:"
GET searchName
 IF VALID CONTACT (searchName) THEN
DISPLAY "Contact found: " + GET_CONTACT (searchName)
ELSE DISPLAY "Contact not found."
GO TO MENU
ELSE IF userChoice == 3 THEN
```

```
DISPLAY ALL CONTACTS
 IF CONTACTS EXIST THEN DISPLAY ALL CONTACTS ELSE DISPLAY "No contacts to display."
 GO TO MENU
 ELSE IF userChoice == 4 THEN
 DELETE CONTACT DISPLAY "Enter name to delete:"
 GET deleteName IF VALID CONTACT (deleteName)
THEN DELETE contact(deleteName)
DISPLAY "Delete successful."
 ELSE DISPLAY "Contact not found."
GO TO MENU
ELSE IF userChoice == 5 THEN
 UPDATE CONTACT
DISPLAY "Enter name to update:"
 GET updateName
 IF VALID CONTACT (updateName) THEN
DISPLAY "Enter new phone number:"
GET newPhoneNumber
UPDATE contact (updateName, newPhoneNumber)
 DISPLAY "Contact updated successfully."
 ELSE DISPLAY "Contact not found."
GO TO MENU
ELSE IF userChoice == 6
THEN SORT CONTACTS
IF CONTACTS EXIST THEN
 DISPLAY "Sorted Contacts:
 " DISPLAY SORTED CONTACTS
 ELSE DISPLAY "No contacts to sort."
GO TO MENU
 ELSE IF userChoice == 7 THEN
EXIT ELSE END
```

Each contributor has played a role in coding, testing, and documenting the project.

6. Conclusion

This phonebook application provides a straightforward and efficient solution for managing contacts using basic linear data structures. The use of linked lists ensures ease of implementation while allowing room for future enhancements with more advanced data structures.