Data and Algorithms Group Project

Group Members

Rejoice Kaulumah 2240161135 (Group leader)

Mulanda Kuze 224070770

Nangolo Rauha 223102385

Benhard Amutse 224061887

Helvia Hashoongo 224016598

Phonebook Application

Introduction

The Phonebook Application allows users to add, view, search, update, and delete contacts. Contacts are stored in a dynamic queue, and the list is automatically sorted alphabetically by name upon adding or updating a contact. This ensures the contact list remains organized for easy search and display.

Section A

The application primarily uses a Dynamic Queue (implemented via a linked list) to manage contacts. Each contact is added to the end of the queue, and the first contact added is the first contact that can be removed (FIFO: First In, First Out). Temporary arrays are used to sort contacts alphabetically for better management and organization.

How the Application Works

Adding a Contact:

When a user adds a contact (name and phone number), the system checks if both fields are filled.
It then checks if the contact is unique (i.e., it doesn't already exist).
If the contact is unique, it's added to the end of the dynamic queue, and the contact list is re-sorted alphabetically.
If the contact already exists, the system alerts the user and requests a unique name and phone number.

Viewing Contacts:

The user can view all contacts in the phonebook.

The contact list is displayed in alphabetical order based on the names. If the contact list is empty, the system notifies the user that no contacts are available.

Searching for a Contact:

The user can search for a contact by typing in a name or phone number.
The search operation uses linear search to go through each contact in the queue and checks if the search term matches the contact's name or phone number (case-insensitive).
If a match is found, the contact is displayed; otherwise, the system informs the user that no contact was found.

Deleting a Contact:

The user can delete the first contact in the queue (since the queue follows FIFO behavior).
If no contacts are available, the system notifies the user. If the contact is deleted successfully, the system provides a confirmation message.

Updating a Contact:

The user can select a contact to update.
The system prompts the user to enter a new name or phone number (with the current name and phone number pre-filled as defaults).
It checks if the new name or phone number is unique (i.e., not a duplicate of any other contact). If valid, the contact is updated, and the list is re-sorted alphabetically.

Sorting Contacts:

After each contact addition or update, the contact list is sorted alphabetically by name.
The sorting process uses a temporary ArrayList to hold all contacts, sort them alphabetically, and then reinsert them into the queue in the correct order.

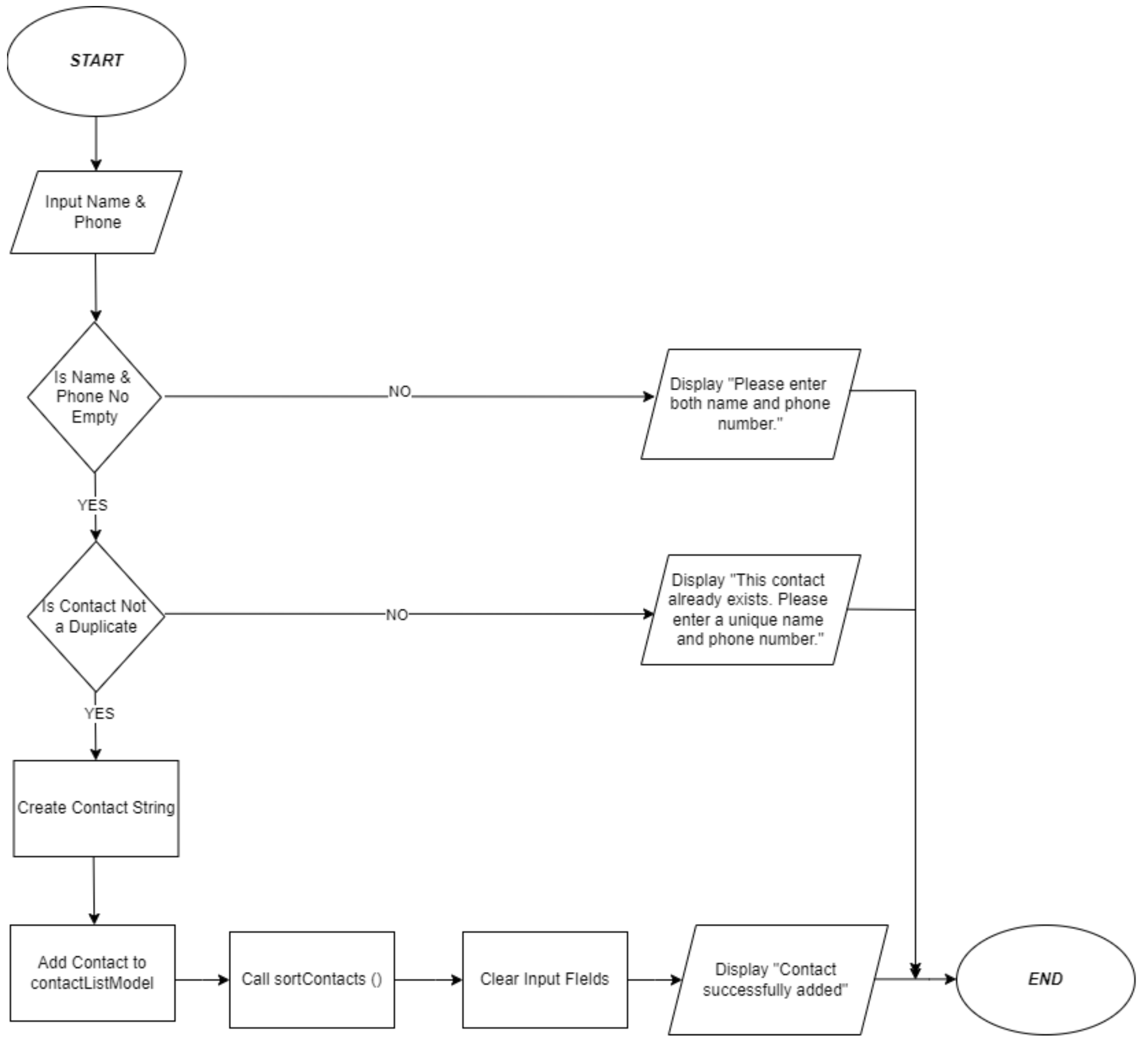
Pseudocode and Flowchart of each Function of the Phonebook

```
// 1.Add contact  Algorithm
// FUNCTION to add a contact to the contact list
FUNCTION addContact(name, phone)
   // Check if both name and phone number fields are filled
   IF name IS NOT EMPTY AND phone IS NOT EMPTY
      // Check if the contact is not a duplicate
      IF NOT isDuplicateContact(name, phone, null, null)
         // Create a contact string combining name and phone number
         contact = "Name: " + name + ", Phone: " + phone
         // Add the new contact to the contact list
```

```
            ADD contact TO contactListModel
            // Sort the contact list alphabetically
            CALL sortContacts()
            // Clear the input fields after successfully adding the contact
            CLEAR nameField AND phoneField
            // Notify the user that the contact was added successfully
            DISPLAY "Contact successfully added."
        ELSE
            // If the contact already exists, notify the user
            DISPLAY "This contact already exists. Please enter a unique name and phone number."
        END IF
    ELSE
        // If either field is empty, notify the user
        DISPLAY "Please enter both name and phone number."
    END IF
END FUNCTION
```

- Data Structure: Dynamic Queue (used to store contacts).

Flowchart :

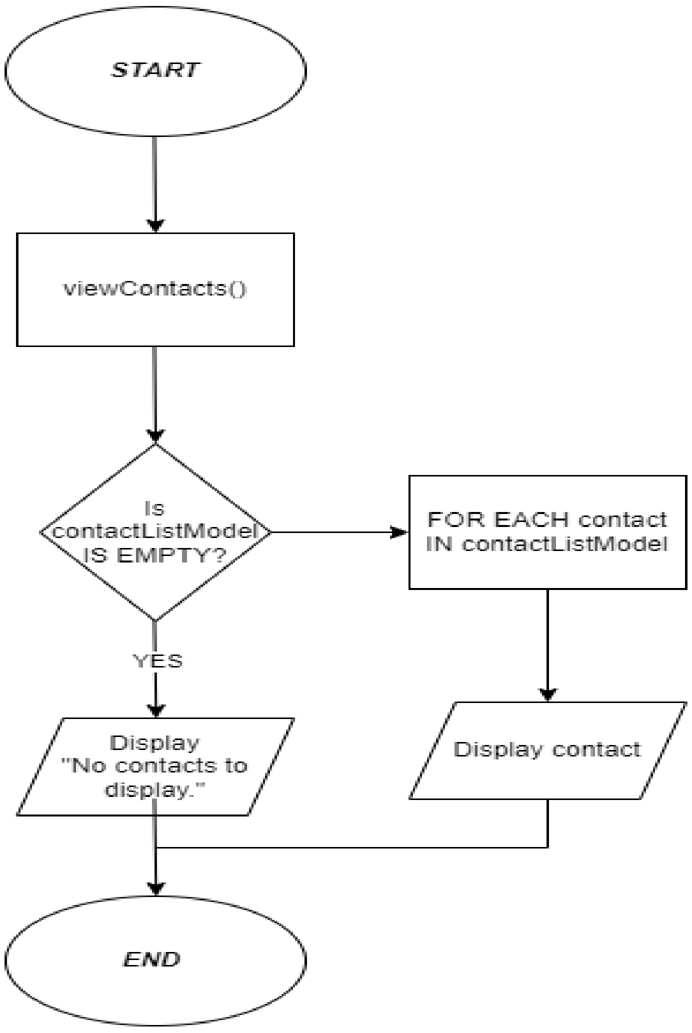

START
Input Name & Phone
Is Name & Phone No Empty
NO
Display "Please enter both name and phone number."
YES
Is Contact Not a Duplicate
NO
Display "This contact already exists. Please enter a unique name and phone number."
YES
Create Contact String
Add Contact to contactListModel
Call sortContacts ()
Clear Input Fields
Display "Contact successfully added"
END

// 2.View Contacts Algorithm

```
FUNCTION viewContacts()
   // Check if the contact list is empty
   IF contactListModel IS EMPTY
      // Notify the user that there are no contacts to display
      DISPLAY "No contacts to display."
   ELSE
      // Iterate through each contact in the contact list
      FOR EACH contact IN contactListModel
         // Display each contact's details
         DISPLAY contact
      END FOR
   END IF
END FUNCTION
```

- Data Structure: Dynamic Queue.

Flowchart:


START
viewContacts()
Is contactListModel IS EMPTY?
YES
Display "No contacts to display."
FOR EACH contact IN contactListModel
Display contact
END

// 3.Search contacts algorithm

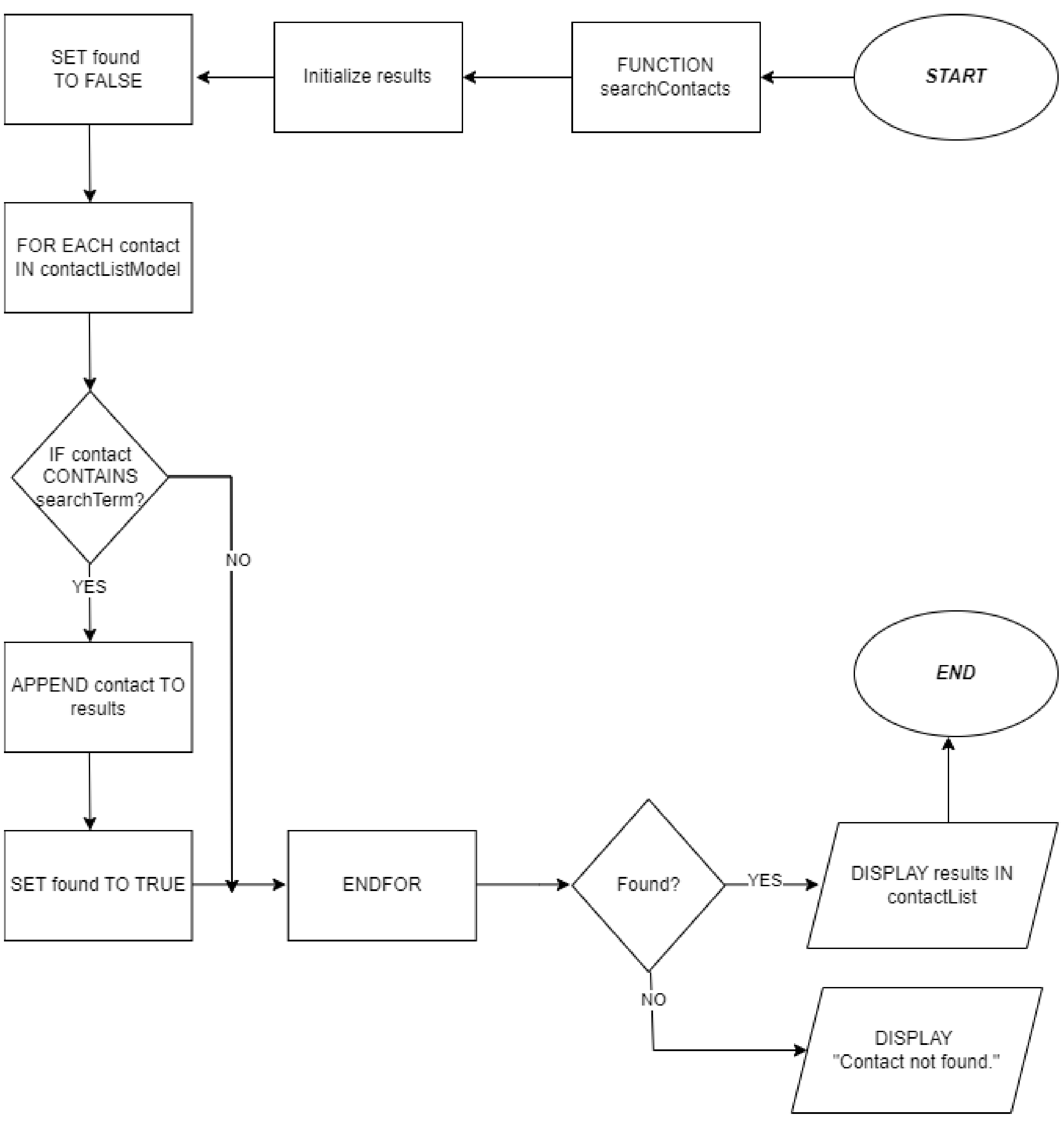
```
// FUNCTION to search for a contact

FUNCTION searchContacts(searchTerm)
  // Initialize an empty string to store the search results
  SET results TO EMPTY STRING
  // Set a boolean variable to track if any contacts are found
  SET found TO FALSE\


  // Loop through each contact in the contact list
  FOR EACH contact IN contactListModel
    // Check if the contact contains the search term (case-insensitive)
    IF contact CONTAINS searchTerm (case-insensitive)
      // Append the found contact to the results
      APPEND contact TO results
      // Mark that at least one contact was found
      SET found TO TRUE
    END IF
  END FOR

  // If a contact was found, display the results
  IF found
    DISPLAY results IN contactList
  ELSE
    // If no contact was found, notify the user
    DISPLAY "Contact not found."
  END IF
END FUNCTION
```

- Data Structure: Dynamic Queue - Uses Linear Search algorithm to search for contacts.

- Efficiency of the linear search

The efficiency of the linear search is O(n) where n is the number of contacts in the queue. It goes through each contact one by one until the desired contact is found or all contacts are checked. While simple to implement, the linear search may become inefficient for large datasets

Flowchart:

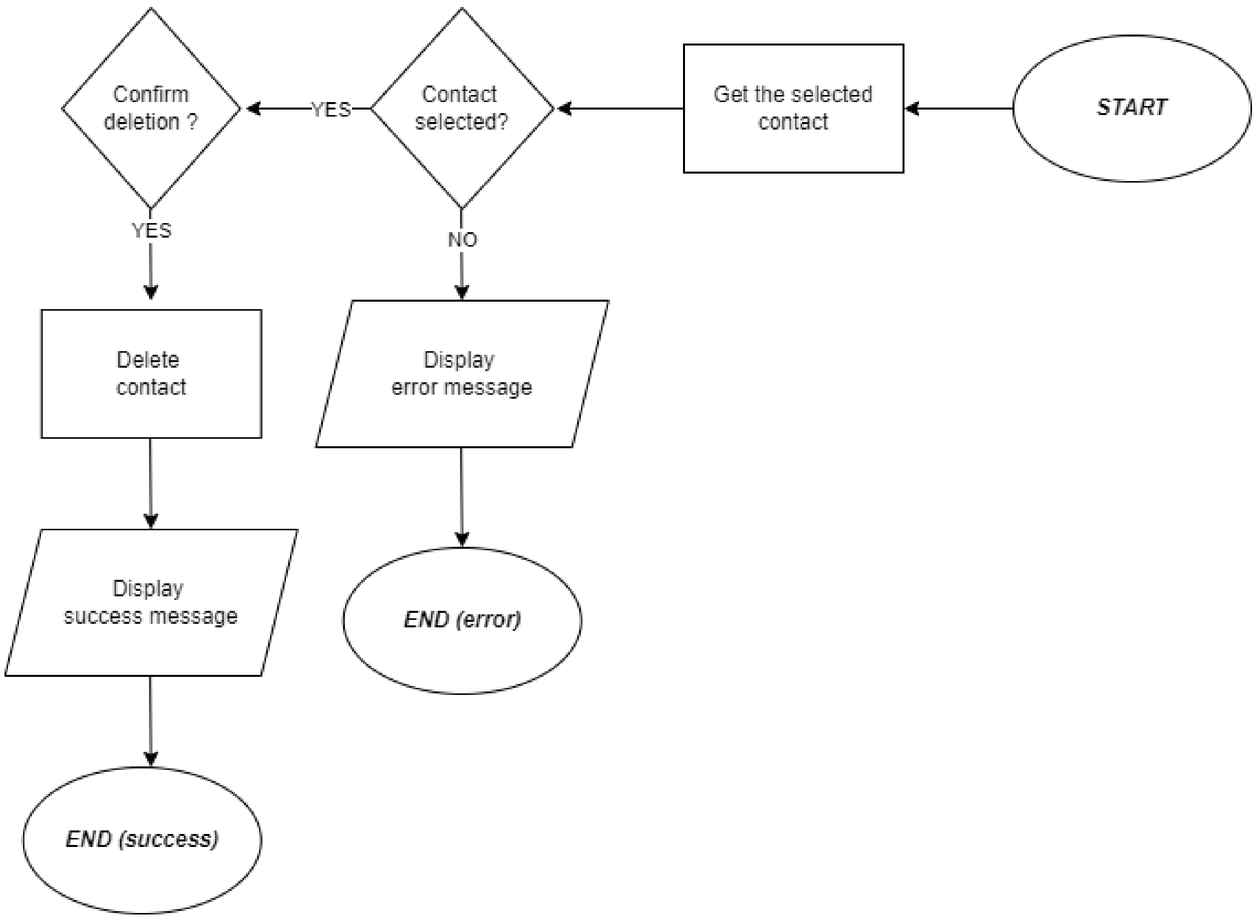

START
FUNCTION searchContacts
Initialize results
SET found TO FALSE
FOR EACH contact IN contactListModel
IF contact CONTAINS searchTerm?
YES
NO
APPEND contact TO results
SET found TO TRUE
ENDFOR
Found?
YES
NO
DISPLAY results IN contactList
DISPLAY "Contact not found."
END

```
// 4.Delete contacts Algorithm
// FUNCTION to delete a selected contact
FUNCTION deleteContact()
  // Get the contact selected by the user
  SET selectedContact TO SELECTED contact FROM contactList

  // Check if a contact is selected
  IF selectedContact IS NOT NULL
    // Prompt the user for confirmation before deleting
    PROMPT USER TO CONFIRM "Are you sure you want to delete the selected contact?"
    // If the user confirms deletion
    IF USER CONFIRMS
```

```
            // Remove the selected contact from the contact list
            REMOVE selectedContact FROM contactListModel
            // Notify the user that the contact was deleted
            DISPLAY "Contact deleted successfully."
        END IF
    ELSE
        // If no contact is selected, notify the user
        DISPLAY "Please select a contact to delete."
    END IF
END FUNCTION
```

- Data Structure: Dynamic Queue (FIFO: first contact added is the first contact deleted).

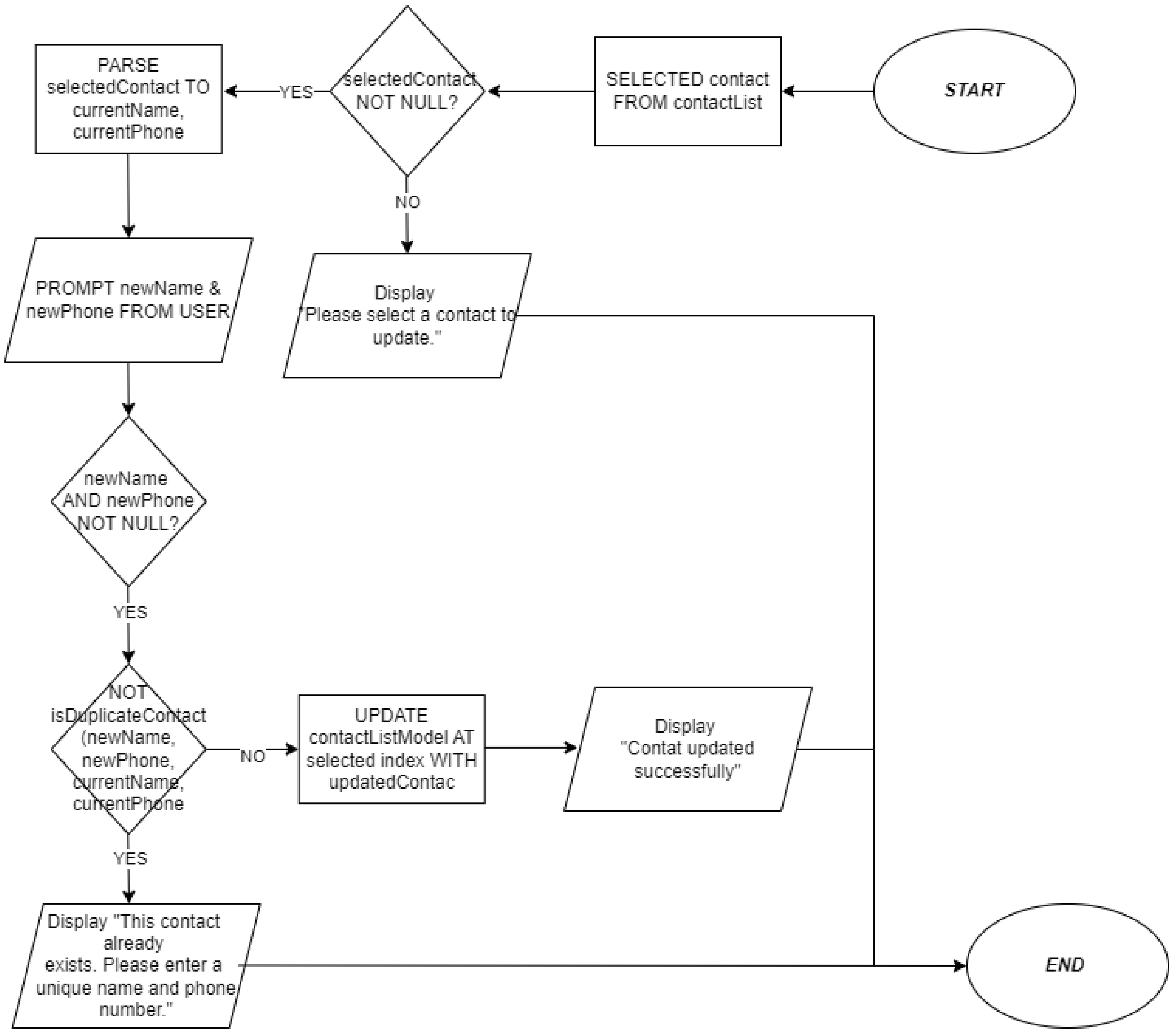
Flowchart:



// 5.Update contacts Algorithm

```
// FUNCTION to update a selected contact
FUNCTION updateContact()
   // Get the contact selected by the user
   SET selectedContact TO SELECTED contact FROM contactList

   // Check if a contact is selected
   IF selectedContact IS NOT NULL
      // Parse the selected contact to extract its current name and phone number
      PARSE selectedContact TO currentName, currentPhone

      // Prompt the user for a new name, using the current name as default
      PROMPT newName FROM USER WITH DEFAULT currentName
      // Prompt the user for a new phone number, using the current phone number as default
      PROMPT newPhone FROM USER WITH DEFAULT currentPhoned

      // Check if the new name and phone number are not empty
      IF newName IS NOT NULL AND newPhone IS NOT NULL
         // Check if the updated contact information is not a duplicate
         IF NOT isDuplicateContact(newName, newPhone, currentName, currentPhone)
            // Create a new string combining the updated name and phone number
            updatedContact = "Name: " + newName + ", Phone: " + newPhone
            // Update the contact at the selected index with the new information
            UPDATE contactListModel AT selected index WITH updatedContact
            // Notify the user that the contact was updated successfully
            DISPLAY "Contact updated successfully."
         ELSE
            // If the updated contact is a duplicate, notify the user
            DISPLAY "This contact already exists. Please enter a unique name and phone number."
         END IF
      END IF
   ELSE
      // If no contact is selected, notify the user
      DISPLAY "Please select a contact to update."
   END IF
END FUNCTION
```

Flowchart:



START
SELECTED contact FROM contactList
selectedContact NOT NULL?
YES
NO
PARSE selectedContact TO currentName, currentPhone
Display "Please select a contact to update."
PROMPT newName & newPhone FROM USER
newName AND newPhone NOT NULL?
YES
NOT isDuplicateContact (newName, newPhone, currentName, currentPhone
NO
UPDATE contactListModel AT selected index WITH updatedContac
Display "Contat updated successfully"
YES
Display "This contact already exists. Please enter a unique name and phone number."
END

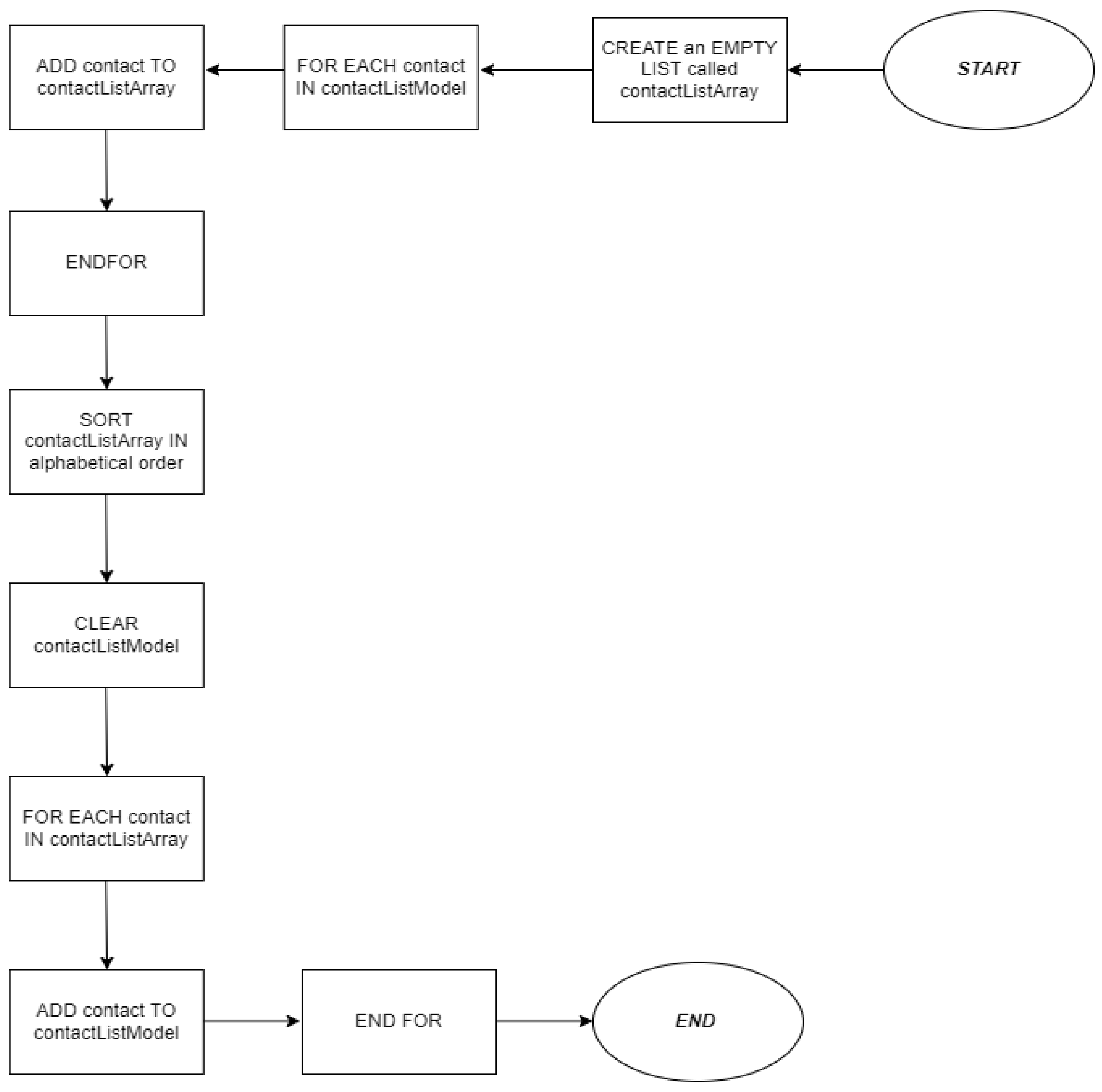
//6. **Sort contacts Algorithm**

```
// FUNCTION to sort contacts alphabetically
FUNCTION sortContacts()
  // Create an empty list to hold contacts temporarily
  CREATE an EMPTY LIST called contactListArray

  // Loop through each contact in the contact list
  FOR EACH contact IN contactListModel
    // Add each contact to the temporary list
    ADD contact TO contactListArray
  END FOR

  // Sort the contacts in the temporary list in alphabetical order
  SORT contactListArray IN alphabetical order
```

```
    // Clear the contact list before re-adding sorted contacts
    CLEAR contactListModel
    // Loop through the sorted contacts in the temporary list
    FOR EACH contact IN contactListArray
        // Add each sorted contact back to the contact list
        ADD contact TO contactListModel
    END FOR
END FUNCTION
```

- Data Structure: ArrayList (used temporarily to hold and sort contacts).

Flowchart:


START
CREATE an EMPTY LIST called contactListArray
FOR EACH contact IN contactListModel
ADD contact TO contactListArray
ENDFOR
SORT contactListArray IN alphabetical order
CLEAR contactListModel
FOR EACH contact IN contactListArray
ADD contact TO contactListModel
END FOR
END

The function of the Phonebook Application

The programming language used was Java

1. Add contacts

```
if (e.getSource() == addButton) {
   String name = nameField.getText().trim();
   String phone = phoneField.getText().trim();
   if (!name.isEmpty() && !phone.isEmpty()) {
      if (!isDuplicateContact(name, phone, "", "")) {
         String contact = "Name: " + name + ", Phone: " + phone;
         contactListModel.addElement(contact);
         sortContacts();
         nameField.setText("");
         phoneField.setText("");
         JOptionPane.showMessageDialog(frame, "Contact successfully added.");
      } else {
         JOptionPane.showMessageDialog(frame, "This contact already exists.");
      }
   } else {
      JOptionPane.showMessageDialog(frame, "Please enter both name and phone number.");
   }
}
```

2.View contacts

```
if (e.getSource() == viewButton) {
   if (contactListModel.isEmpty()) {
      JOptionPane.showMessageDialog(frame, "No contacts to display.");
   }
}
```

3. Delete contacts

```
if (e.getSource() == deleteButton) {
   String selectedContact = contactList.getSelectedValue();
   if (selectedContact != null) {
      int response = JOptionPane.showConfirmDialog(frame,
            "Are you sure you want to delete this contact?", "Confirm Delete",
            JOptionPane.YES_NO_OPTION);
      if (response == JOptionPane.YES_OPTION) {
         contactListModel.removeElement(selectedContact);
         JOptionPane.showMessageDialog(frame, "Contact deleted successfully.");
      }
   } else {
      JOptionPane.showMessageDialog(frame, "Please select a contact to delete.");
```

```
    }
}
```

4. Search contacts

```
private void searchContact(String search) {
    DefaultListModel<String> searchResults = new DefaultListModel<>();
    boolean found = false;
    for (int i = 0; i < contactListModel.size(); i++) {
        String contact = contactListModel.get(i);
        if (contact.toLowerCase().contains(search.toLowerCase())) {
            searchResults.addElement(contact);
            found = true;
        }
    }
    if (found) {
        contactList.setModel(searchResults);
    } else {
        JOptionPane.showMessageDialog(frame, "No matching contacts found.");
    }
}
```

5. Update contacts

```
if (e.getSource() == updateButton) {
    String selectedContact = contactList.getSelectedValue();
    if (selectedContact != null) {
        String[] parts = selectedContact.split(", ");
        String currentName = parts[0].substring(6);
        String currentPhone = parts[1].substring(7);

        String newName = JOptionPane.showInputDialog(frame, "Edit Name:", currentName);
        String newPhone = JOptionPane.showInputDialog(frame, "Edit Phone:", currentPhone);

        if (newName != null && newPhone != null && !newName.isEmpty() && !newPhone.isEmpty()) {
            if (!isDuplicateContact(newName, newPhone, currentName, currentPhone)) {
                String updatedContact = "Name: " + newName + ", Phone: " + newPhone;
                contactListModel.setElementAt(updatedContact, contactList.getSelectedIndex());
                JOptionPane.showMessageDialog(frame, "Contact updated successfully.");
            } else {
                JOptionPane.showMessageDialog(frame, "This contact already exists.");
            }
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Please select a contact to update.");
    }
}
```

6. Sort contact

```
private void sortContacts() {
    ArrayList<String> contactListArray = new ArrayList<>();
    for (int i = 0; i < contactListModel.size(); i++) {
        contactListArray.add(contactListModel.get(i));
    }
    Collections.sort(contactListArray);
    contactListModel.clear();
    contactListArray.forEach(contactListModel::addElement);
}
```