

# **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**

The Phonebook Application is a simple contact management system that allows users to add, view, search, update, and delete contacts. Each contact consists of a name and a phone number. The application also includes functionality for checking duplicate contacts and sorting them alphabetically.

## **Section A**

### **Data Structures used**

DefaultListModel: This data structure is used to store and manage the list of contacts. It provides methods for adding, removing, and updating contacts. The contacts are stored as strings, each containing a name and phone number in the format "Name: <name>, Phone: <phone>".

ArrayList: This is used temporarily in the sorting process to hold the contacts, sort them alphabetically, and then reload them into the DefaultListModel.

## **How the Application Works**

### **Add Contact:**

The user inputs a name and phone number. The application checks for empty fields and duplicate contacts. If valid, the contact is added to the DefaultListModel, sorted alphabetically, and the fields are cleared.

### **View Contacts:**

The user can view all stored contacts. If no contacts exist, a message is displayed indicating that the contact list is empty.

### **Search Contacts:**

The user enters a search term (name or phone number). The application performs search through the contacts and displays matching results. If no match is found, a message is displayed.

### **Delete Contact:**

The user selects a contact from the list and confirms the deletion. The selected contact is then removed from the DefaultListModel.

### **Update Contact:**

The user selects a contact and is prompted to edit the name or phone number. The application checks for duplicates before updating the contact information.

### **Check for Duplicates:**

When adding or updating a contact, the application checks the DefaultListModel to ensure no contact has the same name or phone number.

### **Sort Contacts:**

After adding or updating a contact, the application sorts the contacts alphabetically. This is done by copying the contacts to an ArrayList, sorting them, clearing the DefaultListModel, and re-adding the sorted contacts.

## **Pseudocode and Flowchart of each Function of the Phonebook**

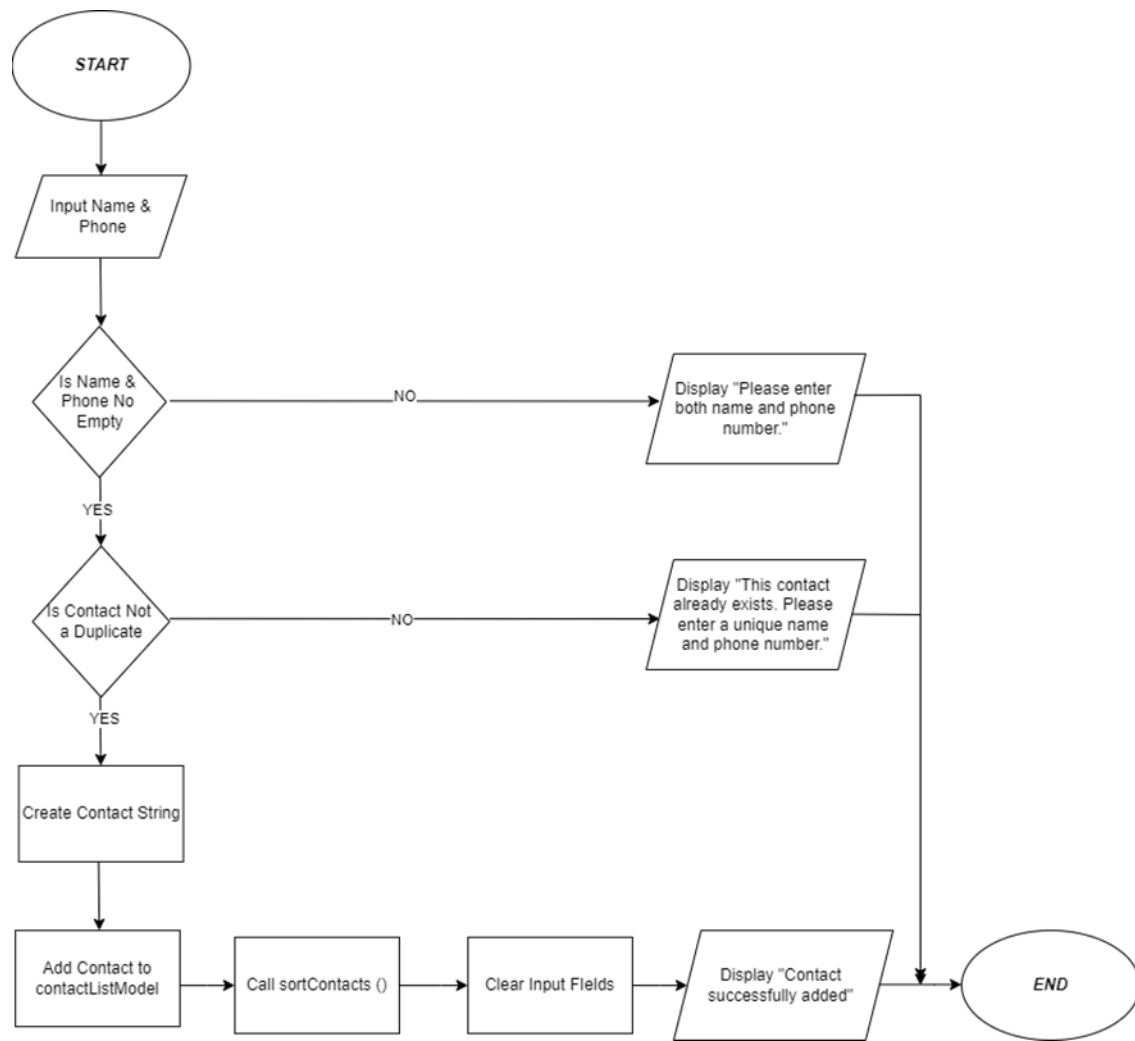
```
// 1.Add contact Algorithm
// FUNCTION to add a contact to the contact list
FUNCTION addContact(name, phone)
// Check if both name and phone number fields are filled
IF name IS NOT EMPTY AND phone IS NOT EMPTY
// Check if the contact is not a duplicate
IF NOT isDuplicateContact(name, phone, null, 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

```

### Flowchart



```

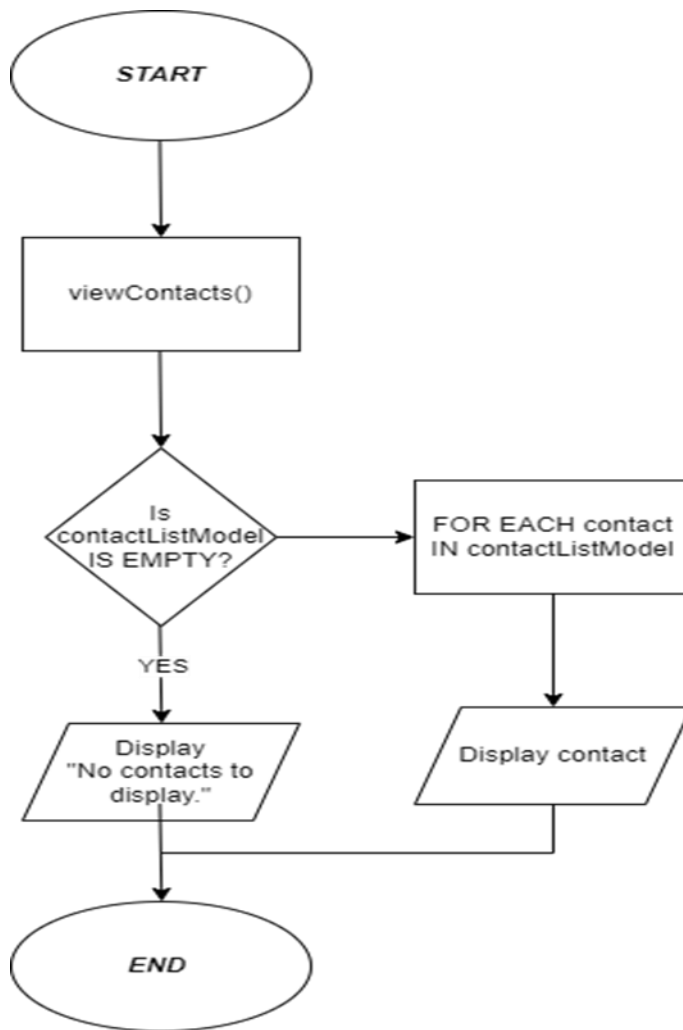
// 2.View Contacts Algorithm
FUNCTION viewContacts()
// Check if the contact list is empty
IF contactListModel IS EMPTY
// Notify the user that there are no contacts to display
DISPLAY "No contacts to display."
ELSE

```

```

// 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
Flowchart

```



```

// 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)

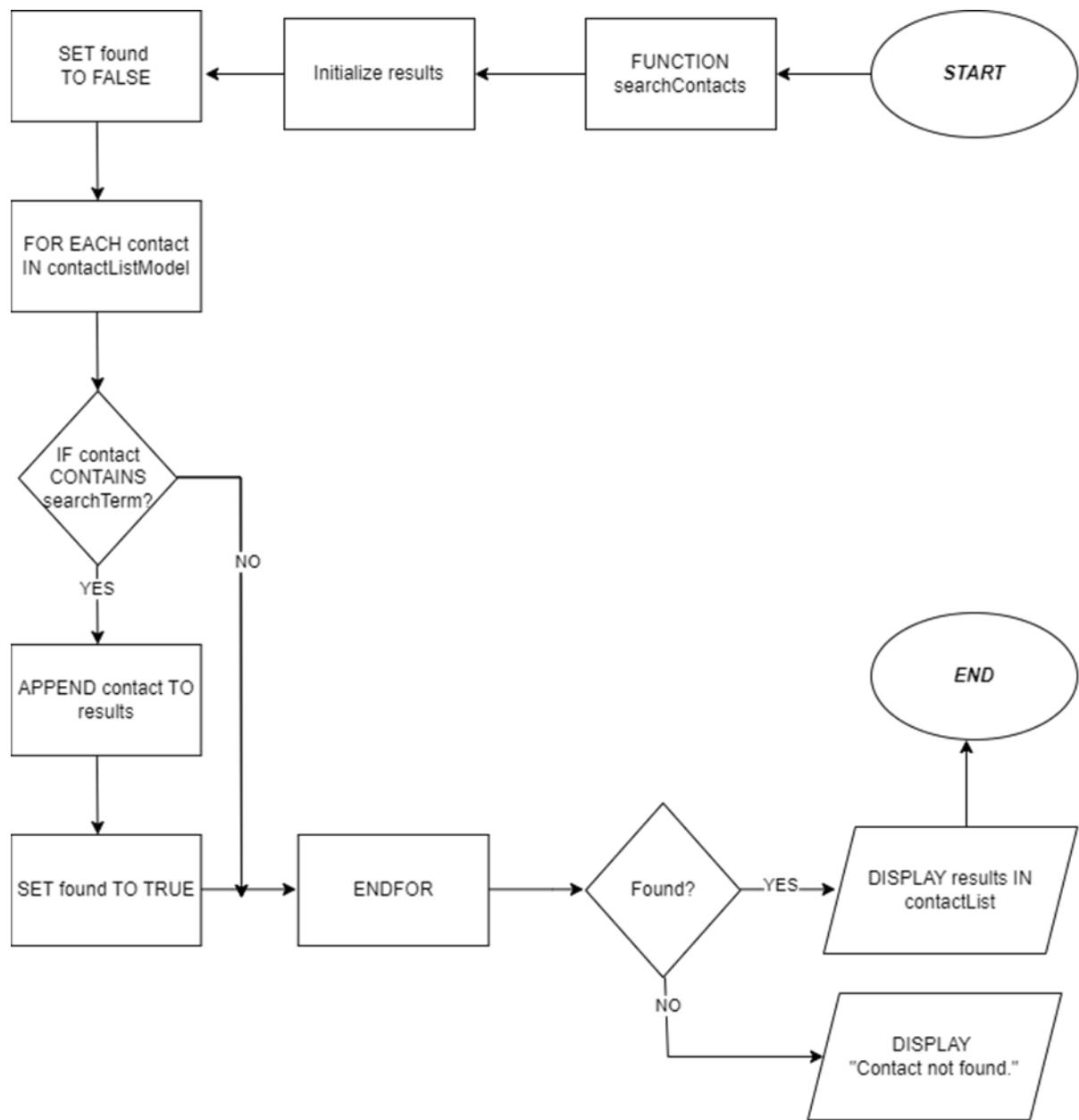
```

```
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
```

- 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**



// 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

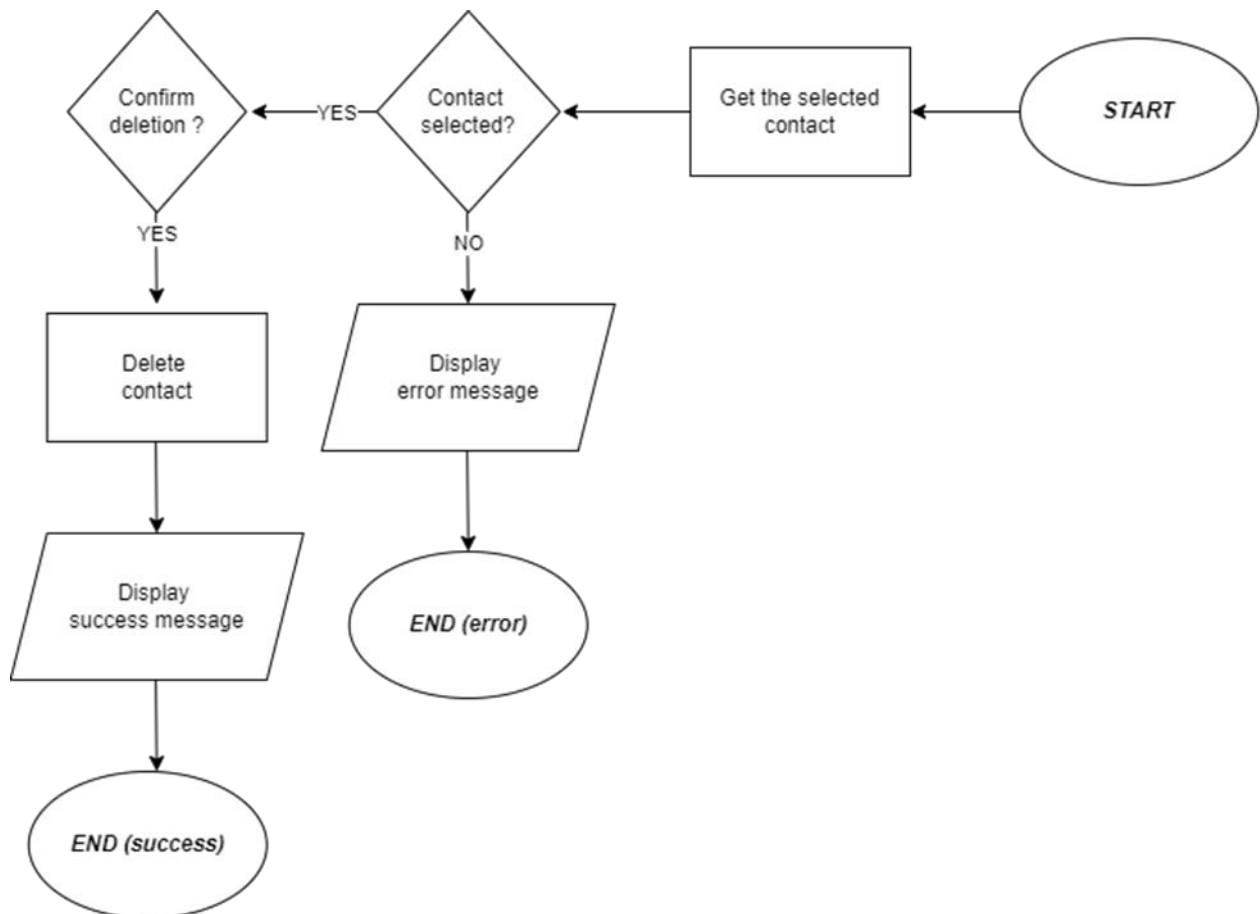
REMOVE selectedContact FROM contactListModel

```

// 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

```

Flowchart



```

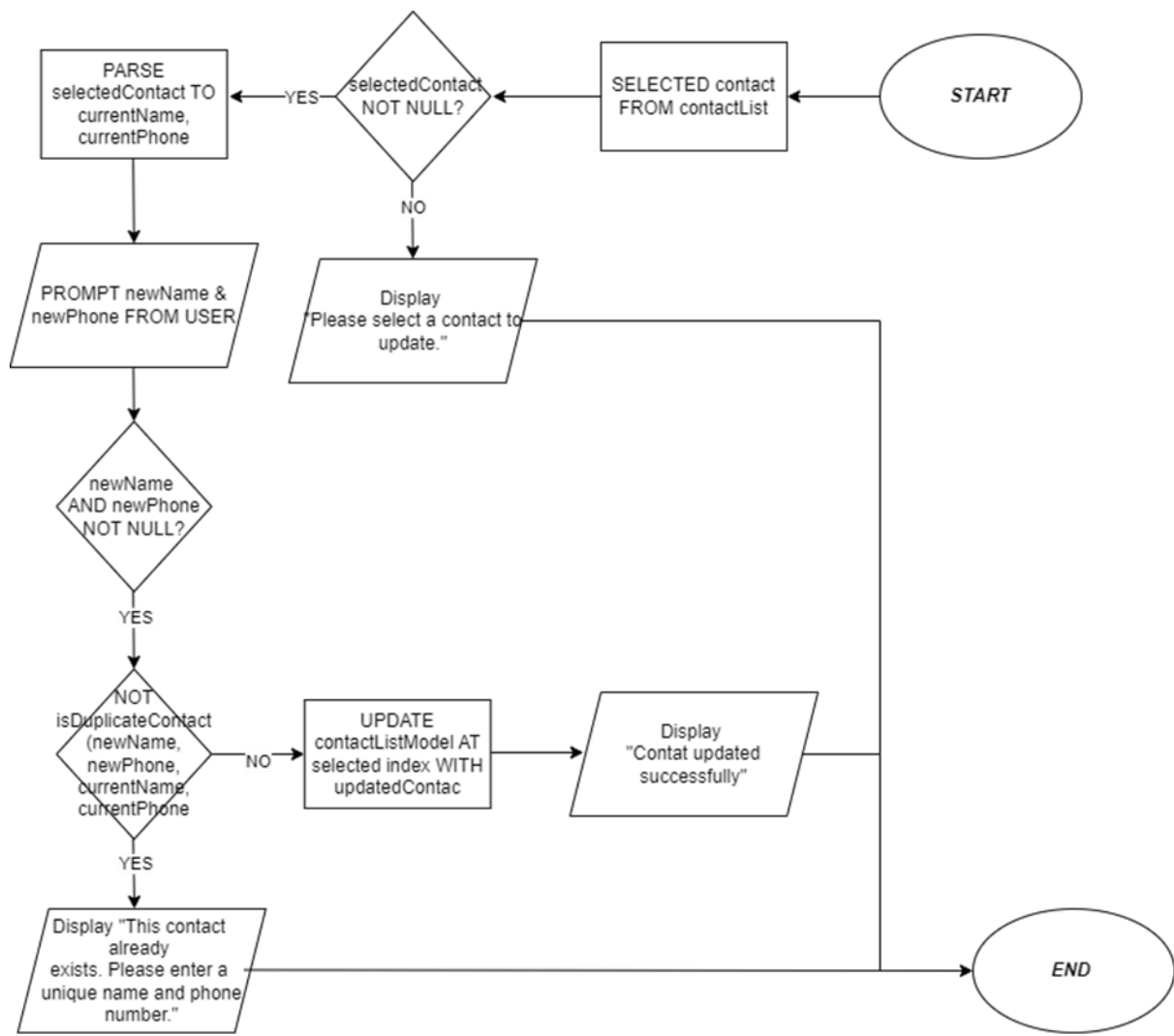
// 5.Update contacts Algorithm
// FUNCTION to update a selected contact
FUNCTION updateContact()
// Get the contact selected by the user
SET selectedContact TO SELECTED contact FROM contactList
// Check if a contact is selected
IF selectedContact IS NOT NULL
// Parse the selected contact to extract its 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 currentPhone

```

```
// 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**





//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

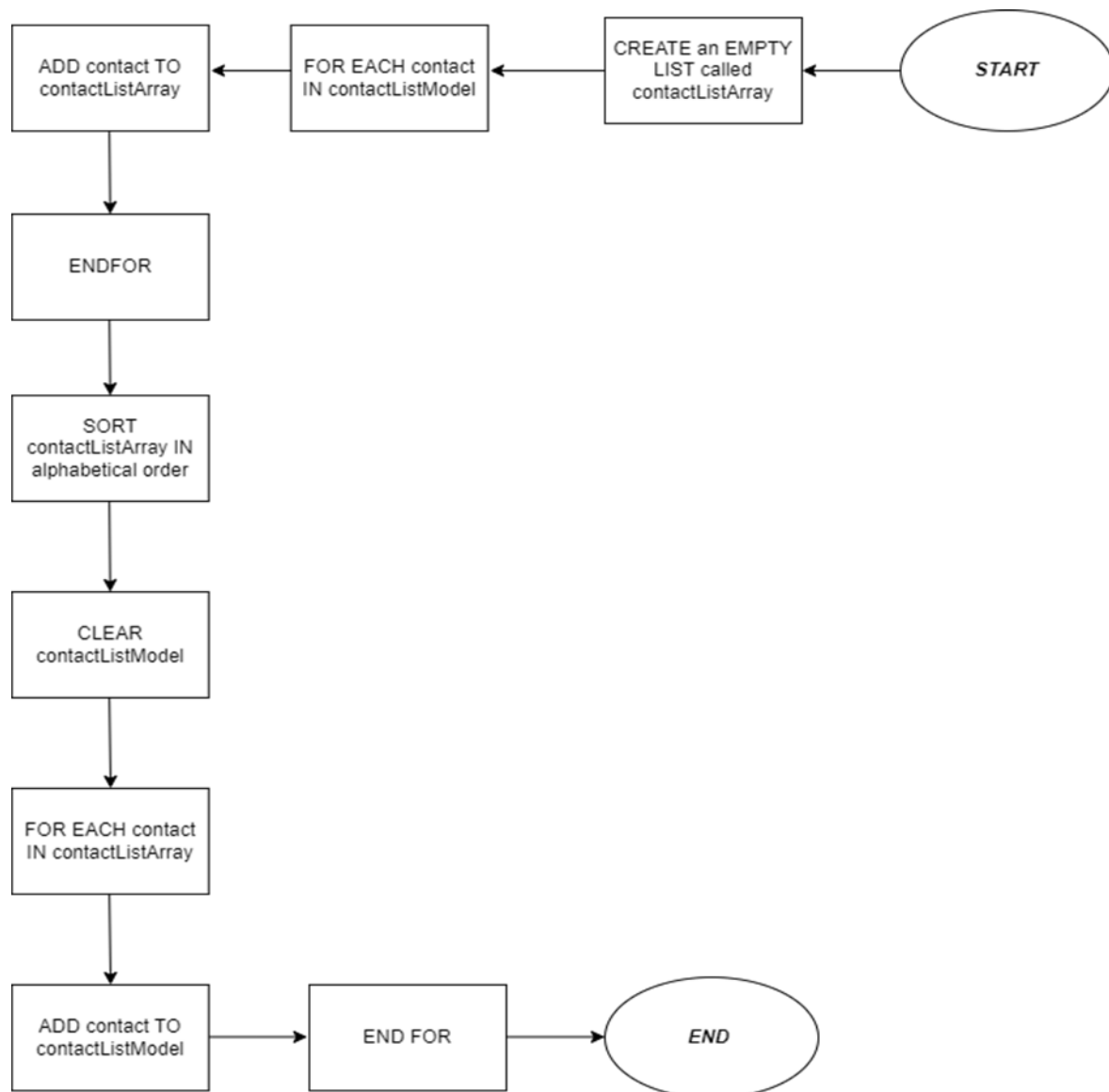
// Loop through the sorted contacts in the temporary list

FOR EACH contact IN contactListArray

// Add each sorted contact back to the contact list

ADD contact TO contactListModel

END FOR  
END FUNCTION



//7. Check for Duplicate Contact

// FUNCTION to check if a contact is a duplicate

FUNCTION isDuplicateContact(name, phone, originalName, originalPhone)

// Loop through each contact in the contact list

FOR EACH contact IN contactListModel

// Parse the contact to extract the name and phone number

PARSE contact TO existingName, existingPhone

// If the original name and phone number are provided, skip this contact

IF originalName IS NOT NULL AND existingName EQUALS originalName AND existingPhone

EQUALS originalPhone

CONTINUE (skip this contact)

```

        END IF

        // Check if the existing contact's name or phone number matches the new name or phone
        number
        IF existingName EQUALS name (case-insensitive) OR existingPhone EQUALS phone
            // Return true, indicating a duplicate was found
            RETURN TRUE (duplicate found)
        END IF
    END FOR

    // If no duplicates were found, return false
    RETURN FALSE (no duplicates found)
END FUNCTION

```

## **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);
}

7.
private boolean isDuplicateContact(String name, String phone, String currentName, String
currentPhone) {
    for (int i = 0; i < contactListModel.size(); i++) {
        String contact = contactListModel.get(i);
        String[] parts = contact.split(", ");
        String existingName = parts[0].substring(6);
        String existingPhone = parts[1].substring(7);

        // Skip checking if this is the current contact being updated
        if (existingName.equalsIgnoreCase(currentName) && existingPhone.equals(currentPhone)) {
            continue;
        }

        // Check for duplicates
        if (existingName.equalsIgnoreCase(name) || existingPhone.equals(phone)) {
            return true;
        }
    }
}

```

```
    }  
  }  
  return false;  
}
```