# **DSA521S Group Project: Phonebook Application.**

Faculty of Computing & Informatics Department of Software Engineering:

Group Members & Student Numbers:

1. **Noa-Eli Singo (Group Leader): 224029630.**
2. **Frans Gottlieb: 221090991.**
3. **Twasikile Pwapwa: 221144765.**
4. **Rodrico Karingombe: 224019503.**
5. **Sibati Museta: 224050036.**

## **Section A: Algorithms Representation of Different Modules and Functions (Pseudocode/Flowchart):**

A Nambian Telecommunications company is looking for an efficient phonebook algorithm based on linear data structures. The key is to use simple, well known data structures and algorithms that can still deliver reasonably efficient performance for typical phonebook operations like insert, search, delete and update. Linear data structures such as arrays and linked lists are relatively simple to implement and understand, making them ideal for simpler systems.

### **Pseudocode:**

**1. Data Structure Declaration**

We will use a simple array structure to store contacts. Each contact will have a name and a phone number.

DECLARE PhoneBook : ARRAY[1:100] OF STRING

DECLARE TotalContacts : INTEGER

TotalContacts ← 0

**2. Insert Contact**

This procedure will allow inserting a new contact into the phonebook.

PROCEDURE InsertContact(Name : STRING, PhoneNumber : STRING)

    IF TotalContacts < LENGTH(PhoneBook) THEN

        TotalContacts ← TotalContacts + 1

        PhoneBook[TotalContacts] ← Name + "," + PhoneNumber

        OUTPUT "Contact inserted successfully."

    ELSE

        OUTPUT "Phonebook is full. Cannot insert more contacts."

    ENDIF

END PROCEDURE

**3. Search Contact**

This function will perform a linear search to find a contact by name. It returns the index of the contact or -1 if not found.

FUNCTION SearchContact(Name : STRING) RETURNS INTEGER

    FOR i ← 1 TO TotalContacts DO

        DECLARE Contact : STRING

        Contact ← PhoneBook[i]

        IF LEFT(Contact, FIND(Contact, ",") - 1) = Name THEN

            RETURN i

        ENDIF

    NEXT i

    RETURN -1  // Contact not found

END FUNCTION

**4. Display All Contacts**

This procedure will output all the contacts in the phonebook.

PROCEDURE DisplayContacts()

    IF TotalContacts = 0 THEN

        OUTPUT "Phonebook is empty."

    ELSE

        FOR i ← 1 TO TotalContacts DO

            OUTPUT PhoneBook[i]

        NEXT i

    ENDIF

END PROCEDURE

**5. Delete Contact**

This procedure deletes a contact from the phonebook based on the name provided.

PROCEDURE DeleteContact(Name : STRING)

    DECLARE Index : INTEGER

    Index ← SearchContact(Name)

    IF Index = -1 THEN

        OUTPUT "Contact not found."

    ELSE

        FOR i ← Index TO TotalContacts - 1 DO

            PhoneBook[i] ← PhoneBook[i + 1]

        NEXT i

        TotalContacts ← TotalContacts - 1

        OUTPUT "Contact deleted successfully."

    ENDIF

END PROCEDURE

**6. Update Contact**

This procedure updates the phone number of an existing contact.

PROCEDURE UpdateContact(Name : STRING, NewPhoneNumber : STRING)

    DECLARE Index : INTEGER

    Index ← SearchContact(Name)

    IF Index = -1 THEN

        OUTPUT "Contact not found."

    ELSE

        PhoneBook[Index] ← Name + "," + NewPhoneNumber

        OUTPUT "Contact updated successfully."

    ENDIF

END PROCEDURE

**7. Sort Contacts (Optional)**

This procedure sorts the phonebook in alphabetical order based on contact names using a simple bubble sort.

PROCEDURE SortContacts()

    DECLARE Temp : STRING

    FOR i ← 1 TO TotalContacts - 1 DO

        FOR j ← i + 1 TO TotalContacts DO

            IF PhoneBook[i] > PhoneBook[j] THEN

                Temp ← PhoneBook[i]

                PhoneBook[i] ← PhoneBook[j]

                PhoneBook[j] ← Temp

            ENDIF

        NEXT j

    NEXT i

    OUTPUT "Contacts sorted successfully."

END PROCEDURE

**8. Analyze Search Efficiency**

This procedure counts the number of steps taken to find a contact, which represents the efficiency of the linear search algorithm.

PROCEDURE AnalyzeSearchEfficiency(Name : STRING)

    DECLARE steps : INTEGER

    steps ← 0

       FOR i ← 1 TO TotalContacts DO

        steps ← steps + 1

        IF LEFT(PhoneBook[i], FIND(PhoneBook[i], ",") - 1) = Name THEN

            OUTPUT "Contact found in ", steps, " steps."

            RETURN

        ENDIF

    NEXT i

    OUTPUT "Contact not found. Search took ", steps, " steps."

END PROCEDURE

**9. Main Program**

This is the main program that prompts the user to choose an option and calls the relevant procedures or functions.

PROCEDURE Main()

    DECLARE choice : INTEGER

    DECLARE name, phoneNumber : STRING

    REPEAT

        OUTPUT "1. Insert Contact"

        OUTPUT "2. Search Contact"

        OUTPUT "3. Display All Contacts"

        OUTPUT "4. Delete Contact"

        OUTPUT "5. Update Contact"

        OUTPUT "6. Sort Contacts"

        OUTPUT "7. Analyze Search Efficiency"

        OUTPUT "8. Exit"

        OUTPUT "Enter your choice: "

        INPUT choice

        IF choice = 1 THEN

            OUTPUT "Enter name: "

            INPUT name

            OUTPUT "Enter phone number: "

            INPUT phoneNumber

            CALL InsertContact(name, phoneNumber)

        ELSE IF choice = 2 THEN

            OUTPUT "Enter name to search: "

            INPUT name

            DECLARE index : INTEGER

            index ← SearchContact(name)

            IF index = -1 THEN

                OUTPUT "Contact not found."

            ELSE

                OUTPUT "Contact found: ", PhoneBook[index]

            ENDIF

        ELSE IF choice = 3 THEN

            CALL DisplayContacts()

        ELSE IF choice = 4 THEN

            OUTPUT "Enter name to delete: "

            INPUT name

            CALL DeleteContact(name)

        ELSE IF choice = 5 THEN

            OUTPUT "Enter name to update: "

            INPUT name

            OUTPUT "Enter new phone number: "

            INPUT phoneNumber

            CALL UpdateContact(name, phoneNumber)

        ELSE IF choice = 6 THEN

            CALL SortContacts()

        ELSE IF choice = 7 THEN

            OUTPUT "Enter name to analyze search: "

            INPUT name

            CALL AnalyzeSearchEfficiency(name)

        ELSE IF choice = 8 THEN

            OUTPUT "Exiting application."

        ENDIF

    UNTIL choice = 8

END PROCEDURE

// Start the program

CALL Main()

### **Section B: Practical Implementation of the program designed in Section A.**

Below is the simplified java code for the Phonebook application.

import java.util.Scanner;  
  
public class PhoneBookApp {  
  
 // Declare the phonebook and total contacts variable  
 private static String[] phoneBook = new String[100];  
 private static int totalContacts = 0;  
   
 // Main method  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int choice;  
 do {  
 System.out.println("1. Insert Contact");  
 System.out.println("2. Search Contact");  
 System.out.println("3. Display All Contacts");  
 System.out.println("4. Delete Contact");  
 System.out.println("5. Update Contact");  
 System.out.println("6. Sort Contacts");  
 System.out.println("7. Analyze Search Efficiency");  
 System.out.println("8. Exit");  
 System.out.print("Enter your choice: ");  
 choice = scanner.nextInt();  
 scanner.nextLine(); // clear the buffer  
  
 switch (choice) {  
 case 1:  
 System.out.print("Enter name: ");  
 String name = scanner.nextLine();  
 System.out.print("Enter phone number: ");  
 String phoneNumber = scanner.nextLine();  
 insertContact(name, phoneNumber);  
 break;  
 case 2:  
 System.out.print("Enter name to search: ");  
 name = scanner.nextLine();  
 int index = searchContact(name);  
 if (index == -1) {  
 System.out.println("Contact not found.");  
 } else {  
 System.out.println("Contact found: " + phoneBook[index]);  
 }  
 break;  
 case 3:  
 displayContacts();  
 break;  
 case 4:  
 System.out.print("Enter name to delete: ");  
 name = scanner.nextLine();  
 deleteContact(name);  
 break;  
 case 5:  
 System.out.print("Enter name to update: ");  
 name = scanner.nextLine();  
 System.out.print("Enter new phone number: ");  
 phoneNumber = scanner.nextLine();  
 updateContact(name, phoneNumber);  
 break;  
 case 6:  
 sortContacts();  
 break;  
 case 7:  
 System.out.print("Enter name to analyze search: ");  
 name = scanner.nextLine();  
 analyzeSearchEfficiency(name);  
 break;  
 case 8:  
 System.out.println("Exiting application.");  
 break;  
 default:  
 System.out.println("Invalid choice. Please try again.");  
 }  
 } while (choice != 8);  
 scanner.close();  
 }  
  
 // Insert a new contact into the phonebook  
 public static void insertContact(String name, String phoneNumber) {  
 if (totalContacts < phoneBook.length) {  
 phoneBook[totalContacts] = name + "," + phoneNumber;  
 totalContacts++;  
 System.out.println("Contact inserted successfully.");  
 } else {  
 System.out.println("Phonebook is full. Cannot insert more contacts.");  
 }  
 }  
  
 // Search for a contact by name  
 public static int searchContact(String name) {  
 for (int i = 0; i < totalContacts; i++) {  
 String contact = phoneBook[i];  
 String contactName = contact.split(",")[0]; // Extract name from "name,phoneNumber"  
 if (contactName.equalsIgnoreCase(name)) {  
 return i;  
 }  
 }  
 return -1; // Contact not found  
 }  
  
 // Display all contacts  
 public static void displayContacts() {  
 if (totalContacts == 0) {  
 System.out.println("Phonebook is empty.");  
 } else {  
 for (int i = 0; i < totalContacts; i++) {  
 System.out.println(phoneBook[i]);  
 }  
 }  
 }  
  
 // Delete a contact by name  
 public static void deleteContact(String name) {  
 int index = searchContact(name);  
 if (index == -1) {  
 System.out.println("Contact not found.");  
 } else {  
 // Shift all contacts after the deleted contact  
 for (int i = index; i < totalContacts - 1; i++) {  
 phoneBook[i] = phoneBook[i + 1];  
 }  
 totalContacts--; // Reduce totalContacts count  
 System.out.println("Contact deleted successfully.");  
 }  
 }  
  
 // Update a contact's phone number  
 public static void updateContact(String name, String newPhoneNumber) {  
 int index = searchContact(name);  
 if (index == -1) {  
 System.out.println("Contact not found.");  
 } else {  
 phoneBook[index] = name + "," + newPhoneNumber; // Update the contact  
 System.out.println("Contact updated successfully.");  
 }  
 }  
  
 // Sort contacts alphabetically by name using bubble sort  
 public static void sortContacts() {  
 for (int i = 0; i < totalContacts - 1; i++) {  
 for (int j = i + 1; j < totalContacts; j++) {  
 String[] contact1 = phoneBook[i].split(",");  
 String[] contact2 = phoneBook[j].split(",");  
 if (contact1[0].compareToIgnoreCase(contact2[0]) > 0) {  
 // Swap contacts  
 String temp = phoneBook[i];  
 phoneBook[i] = phoneBook[j];  
 phoneBook[j] = temp;  
 }  
 }  
 }  
 System.out.println("Contacts sorted successfully.");  
 }  
  
 // Analyze search efficiency by counting the number of steps  
 public static void analyzeSearchEfficiency(String name) {  
 int steps = 0;  
 for (int i = 0; i < totalContacts; i++) {  
 steps++;  
 String contactName = phoneBook[i].split(",")[0]; // Extract name from "name,phoneNumber"  
 if (contactName.equalsIgnoreCase(name)) {  
 System.out.println("Contact found in " + steps + " steps.");  
 return;  
 }  
 }  
 System.out.println("Contact not found. Search took " + steps + " steps.");  
 }  
}