Jesaya Barnabas	219096910
Tadeus Kalola	224092057
Tuyeimo Nangobe	224096303
Rainer Coetzer	223001481
Delicia Damases	223128856

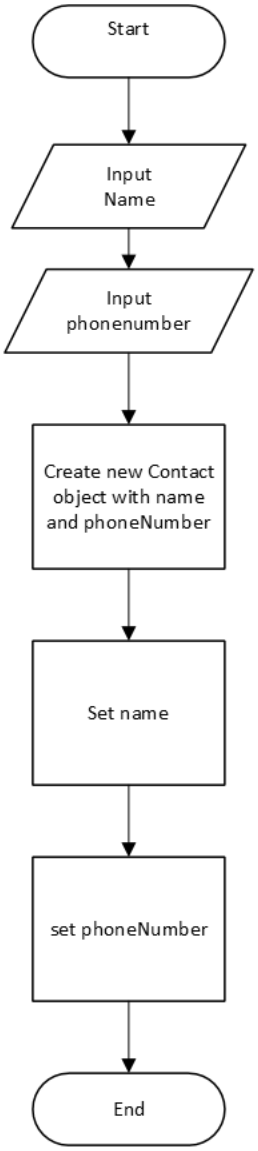
Contents

Modules

Contact

In a traditional phonebook a contact has two major components of interest to the user. The name and the phone number. To build this basic structure, we lay down the foundation of the phonebook the contact class that stores key information. Having this separate class enables easier expansion with additional information of our contact such as the address and or email address.

Representing this in a flowchart:



```
CLASS Contact
  ATTRIBUTE name
  ATTRIBUTE phoneNumber

  FUNCTION Contact (name, phoneNumber)
    SET this.name = name
    SET this.phoneNumber = phoneNumber
  END FUNCTION
END CLASS
```

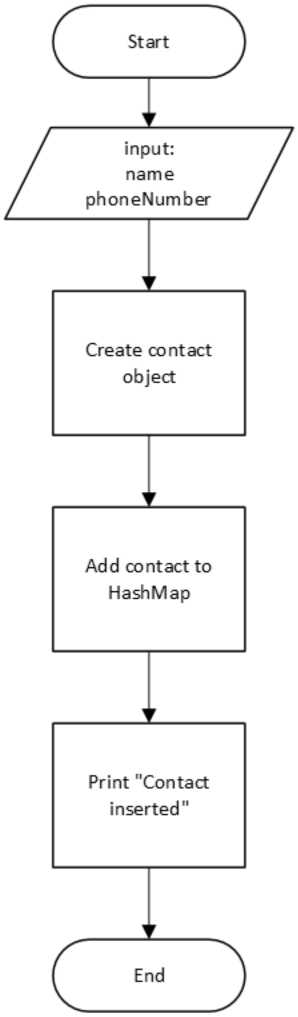
Phone book

A Vital component to our application it not only stores contacts but performs key functions (insert, delete update, delete search and display). These functions should be efficient throughout, as speed defines our modern age. The data structure we choose to implement in this application is crucial to the efficiency of the application. Linked list are dynamic and efficient in terms of insertion and deletion, there memory usage can be more expensive as they have a value and node addresses however, they fail in one major category that being access time or search time, which crucial in a phone book. Arrays have fast access times; however, they aren't dynamic and are inefficient in terms of insertion or deletion, and this is crucial as the phonebook grows over time. The best data structure therefore in the context of this project is a HashTable which is dynamic, has the same efficiency in terms of insertion and deletion as linked list and the same access and search time as arrays.

Next, we will need to implement our key operations using a HashMap, but first we will need to represent our functions as flowcharts and pseudocode. Then we can build our code from there.

Insert contact

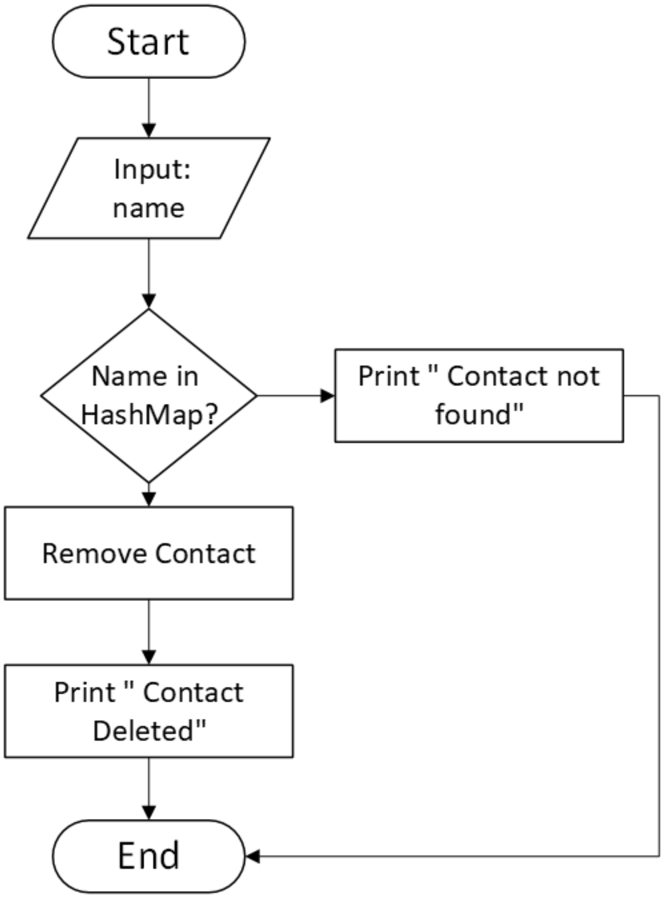
Insert a contact with a name and phoneNumber



```
FUNCTION insertContact(phonebook, name, phoneNumber)
    CREATE new Contact(name, phoneNumber)
    ADD Contact to phonebook (e.g., in HashMap with name as key)
    PRINT "Contact inserted: " + name
END FUNCTION
```

Delete contact

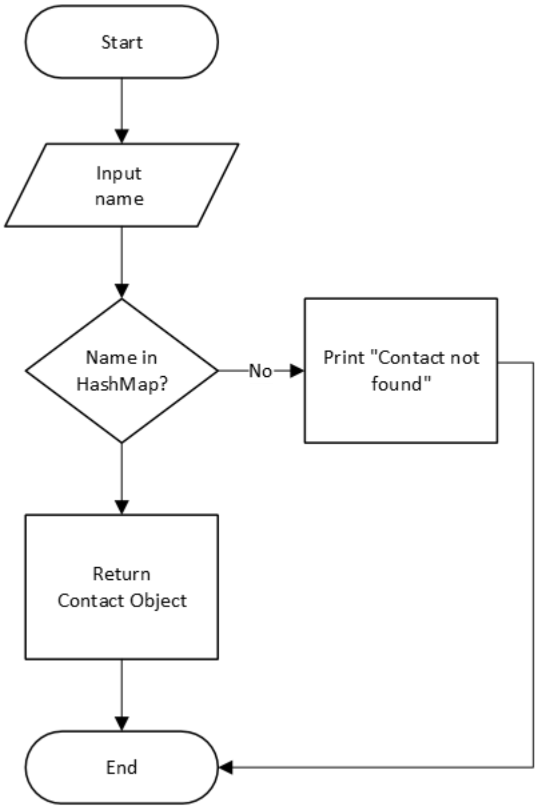
Delete a contact from the hashmap.



```
FUNCTION deleteContact(phonebook, name)
  IF name EXISTS in phonebook
    REMOVE Contact from phonebook
    PRINT "Contact deleted: " + name
  ELSE
    PRINT "Contact not found."
  END IF
END FUNCTION
```

Search contact

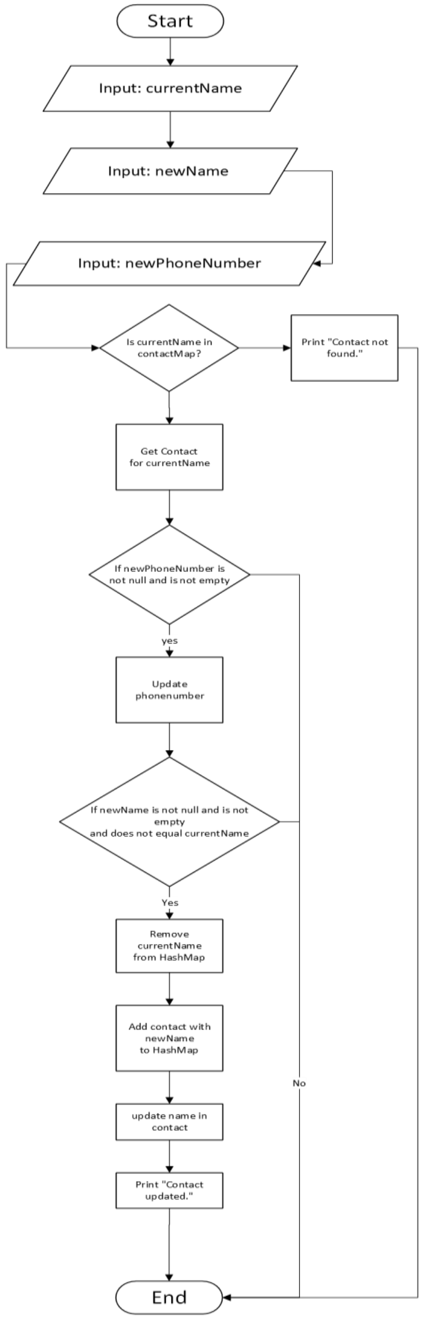
Search for a contact in the HashMap.



```
FUNCTION searchContact(phonebook, name)
  IF name EXISTS in phonebook
    RETURN Contact object
  ELSE
    PRINT "Contact not found."
    RETURN null
  END IF
END FUNCTION
```

Updating contact

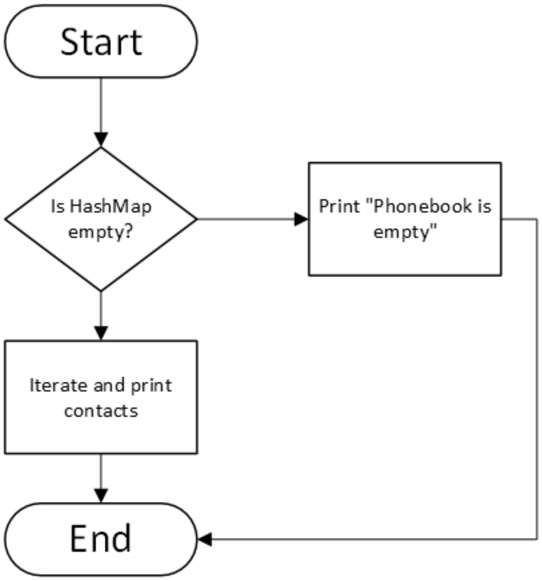
Update a contact name or phone number.


Start
Input: currentName
Input: newName
Input: newPhoneNumber
Is currentName in contactMap?
Print "Contact not found."
Get Contact for currentName
If newPhoneNumber is not null and is not empty
yes
Update phonenumber
If newName is not null and is not empty and does not equal currentName
Yes
Remove currentName from HashMap
Add contact with newName to HashMap
update name in contact
Print "Contact updated."
No
End

```
FUNCTION updateContact(phonebook, currentName, newName, newPhoneNumber)

   IF currentName EXISTS in phonebook THEN

    contact = phonebook.get(currentName)

    IF newPhoneNumber IS NOT null AND newPhoneNumber IS NOT empty THEN
      SET contact.phoneNumber = newPhoneNumber
    END IF

    IF newName IS NOT null AND newName IS NOT empty AND newName IS NOT equal to
currentName THEN

      phonebook.remove(currentName)
      phonebook.add(newName, contact)
      SET contact.currentName = newName

    END IF

    PRINT "Contact updated."

  ELSE

    PRINT "Contact not found."
  END IF
END FUNCTION
```

Display all contacts

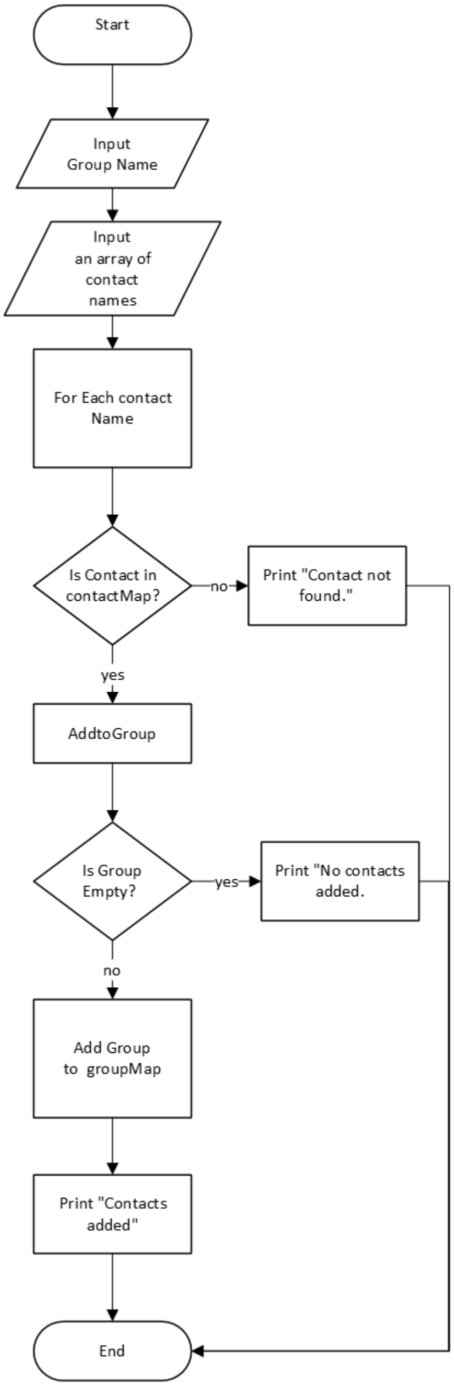
Displaying all contacts in the Hashmap



```
FUNCTION displayAllContacts(phonebook)
  IF phonebook IS EMPTY
    PRINT "Phonebook is empty."
  ELSE
    FOR each contact IN phonebook
      PRINT "Name: " + contact.name + ", Phone Number: " + contact.phoneNumber
    END FOR
  END IF
END FUNCTION
```

Group contacts

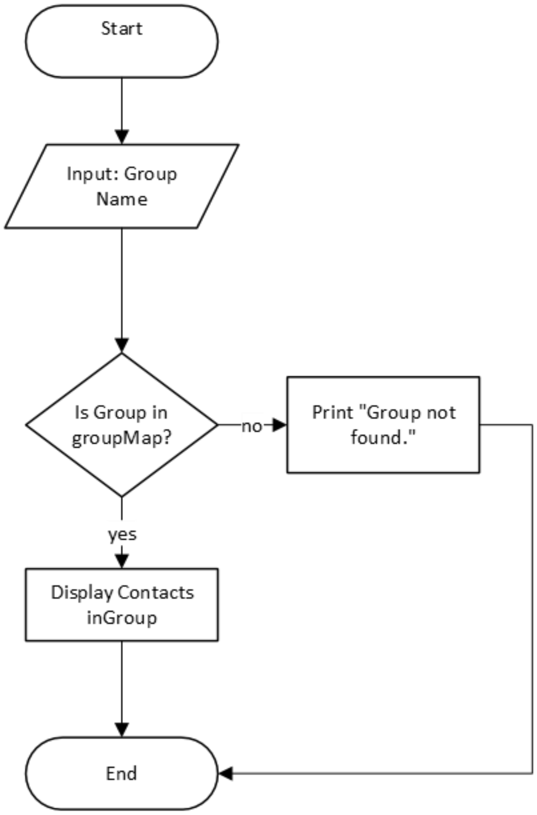
Group contacts under a single name making organization better.



```
FUNCTION groupContacts(phonebook, groupName, contactNames)
  INITIALIZE groupContacts as an empty HashMap

  FOR EACH name IN contactNames
    IF name EXISTS in phonebook.contactMap
      ADD name and phonebook.contactMap[name] TO groupContacts
    ELSE
      PRINT "Contact not found: " + name
    END IF
  END FOR

  IF groupContacts IS NOT empty
    ADD groupName and groupContacts TO phonebook.groupMap
    PRINT "Contacts added to group: " + groupName
  ELSE
    PRINT "No contacts added to group."
  END IF
END FUNCTION
```

Search group

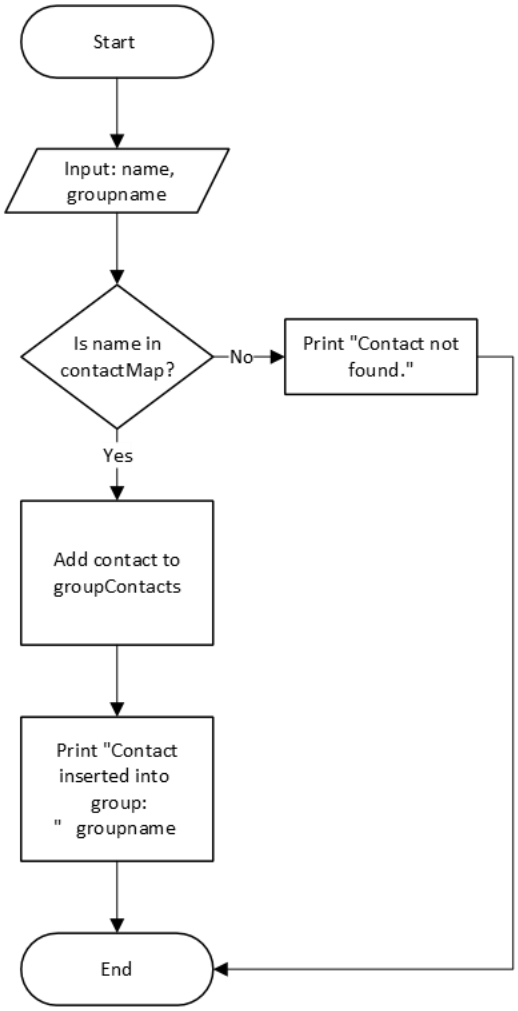
Search for a group.



```
FUNCTION searchContactsByGroup(phonebook, groupName)
  IF groupName EXISTS in phonebook.groupMap
    INITIALIZE groupContacts = phonebook.groupMap[groupName]
    PRINT "Contacts in group: " + groupName

    FOR EACH contact IN groupContacts
      PRINT "Name: " + contact.name + ", Phone Number: " + contact.phoneNumber
    END FOR
  ELSE
    PRINT "Group not found: " + groupName
  END IF
END FUNCTION
```

Insert into group

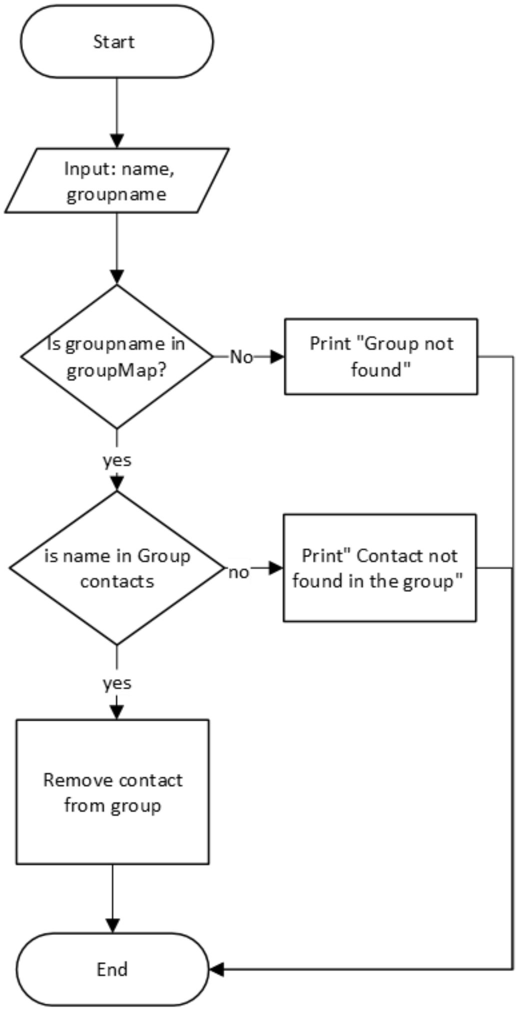
Insert a contact into a group.


Start
Input: name, groupname
Is name in contactMap?
No
Print "Contact not found."
Yes
Add contact to groupContacts
Print "Contact inserted into group: " groupname
End

```
FUNCTION insertIntoGroup(name, groupname)
  IF name EXISTS in contactMap
    ADD contact to groupContacts with key = name
    PRINT "Contact inserted into group: " + groupname
  ELSE
    PRINT "Contact not found."
  END IF
END FUNCTION
```

Delete from group

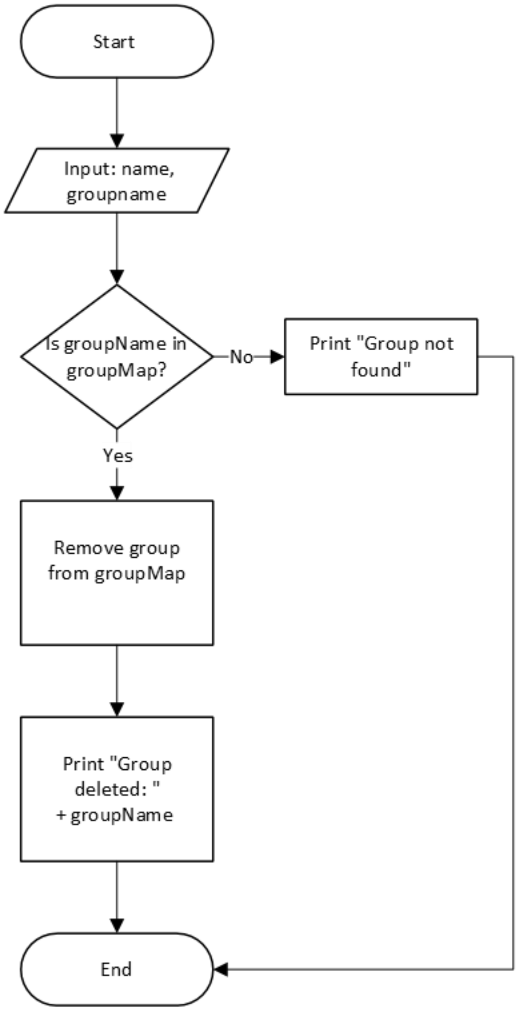
Delete a contact from a group


Start
Input: name, groupname
Is groupname in groupMap?
No
Print "Group not found"
yes
is name in Group contacts
no
Print" Contact not found in the group"
yes
Remove contact from group
End

```
FUNCTION deleteFromGroup(name, groupname)
  IF groupname EXISTS in groupMap
    IF name EXISTS in groupContacts (groupMap[groupname])
      REMOVE the contact from groupContacts
      PRINT "Contact removed from group: " + groupname
    ELSE
      PRINT "Contact not found in the group."
    END IF
  ELSE
    PRINT "Group not found."
  END IF
END FUNCTION
```

Delete group

Delete a group


Start
Input: name, groupname
Is groupName in groupMap?
No
Print "Group not found"
Yes
Remove group from groupMap
Print "Group deleted: " + groupName
End

```
FUNCTION deleteGroup(groupName)
  IF groupName EXISTS in groupMap
    REMOVE the group from groupMap
    PRINT "Group deleted: " + groupName
  ELSE
    PRINT "Group not found."
  END IF
END FUNCTION
```

Phone (Main)

Although not crucial to the functionality of the phonebook this class allows the user to interact and use the phonebook. It's a way to test and prove the functionality of the phonebook. It uses the command line to interact with it. A helper method is used to confirm the user's input. Although basic it serves its purpose in proving the functionality of the phonebook and makes it easy in the process.

```
FUNCTION main
  CREATE a phonebook
  INITIALIZE scanner for user input
  INITIALIZE operation as a variable to store user choices

  LOOP until user chooses to exit
    ASK user for the operation they want to perform (Insert, Search, Delete, Display, Update, Group Contacts, Search Group, Remove from Group, Insert into Group, Delete Group, Exit)

    IF user chooses to insert a contact
      PROMPT user for a name
      CONFIRM name input with the user
      IF confirmed, PROMPT user for a phone number
      CONFIRM phone number input with the user
      IF confirmed, ADD the contact to the phonebook

    IF user chooses to search for a contact
      PROMPT user for the name to search
      IF a name is given, SEARCH the phonebook
      IF contact is found, SHOW the contact details
      ELSE, SHOW that the contact was not found

    IF user chooses to delete a contact
      PROMPT user for the name to delete
      CONFIRM with the user if they are sure about deletion
      IF confirmed, DELETE the contact from the phonebook

    IF user chooses to display all contacts
      SHOW all contacts in the phonebook

    IF user chooses to update a contact
      PROMPT user for the name of the contact to update
      PROMPT user for new name or new phone number (allow skipping fields)
      CONFIRM the changes with the user
      IF confirmed, UPDATE the contact in the phonebook

    IF user chooses to group contacts
      PROMPT user for a group name
      PROMPT user for contact names to add to the group (allow multiple contacts)
      IF valid, ADD the contacts to the group

    IF user chooses to search for contacts in a group
      PROMPT user for the group name
      IF group exists, SHOW the contacts in the group
      ELSE, SHOW that the group was not found

    IF user chooses to remove a contact from a group
      PROMPT user for the group name
      PROMPT user for the contact name to remove from the group
      IF contact exists in group, REMOVE the contact from the group
      ELSE, SHOW that the contact or group was not found
```

```
    IF user chooses to insert a contact into a group
      PROMPT user for the group name
      PROMPT user for the contact name to add to the group
      IF the contact exists, INSERT the contact into the group

    IF user chooses to delete a group
      PROMPT user for the group name to delete
      IF group exists, DELETE the group
      ELSE, SHOW that the group was not found

    IF user chooses to exit
      EXIT the loop and close the phonebook

    HANDLE any errors in user input

  END LOOP
END FUNCTION

FUNCTION getInputWithConfirmation
  PROMPT user for a specific input (name, phone number, etc.)
  ASK for confirmation of the input
  IF confirmed, RETURN the input
  IF canceled, RETURN nothing
END FUNCTION
```

Search algorithm analysis

Data Structure

A HashMap is based on a hash table, it maps keys (contact names) to values (contact objects) using a hash function

A unique identifier is produced when a key (names of contacts) is hashed, and it can then be used to quickly locate a corresponding value (contact object) in an array-like structure called buckets.

Time complexity

Average case:

O(1) constant time

- Most cases search for a contact by name involves calculating hash code of the key (contact name) and then using it to directly access the correct buck in the hash table
- In the case of no collisions (mapping to the same bucket) the HashMap can retrieve a value in constant time
- This results in a average case of O(1) time which is highly efficient even for larger data sets, like that of a phonebook with thousands of contacts.

Worst case:

O(n) linear time/O(log n)

- When many keys hash to the same bucket it causes hash collisions resulting in degraded efficiency
- Multiple entries are stores in either a linked list or tree-like structure
- In this worst case, you would need to search through all the entries in the bucket which leads to a degraded time complexity, O(n).
- Java's HashMap however mitigates this. It switches to a balanced tree structure when the collision threshold is exceeded in a bucket. This reduces search time to O(log n).

Space Complexity

O(n)

n is the number of entries(contact) stored in the map. Key-value pairs consume space proportional to number of contacts.