Table of Contents

Introduction

This report presents the development of a **Phone Directory Application** implemented using **C++**. The application uses a **doubly linked list** to store contact information, allowing for efficient addition, deletion, and searching of contacts. Additionally, the program features an **undo functionality** that allows users to reverse their last actions, as well as a **priority queue** to track and display the most frequently accessed contacts.

The project demonstrates practical application of data structures such as **doubly linked lists**, **stacks**, and **unordered maps** for creating a functional and efficient contact management system.

Objectives

The main objectives of the Phone Directory Application are:

- To design a **contact management system** using **doubly linked lists**.
- To implement basic features such as adding, deleting, updating, and sorting contacts.
- To provide a **search** function to find contacts based on names or phone numbers.
- To track **most accessed contacts** using a **priority queue**.
- To implement an **undo feature** to allow the user to reverse the last action performed.

Functional Requirements

The application supports the following features:

Add Contact:

- Users can add contacts to the directory.
- Contacts are added in **sorted order** based on the contact name.

```
************************************************
 *** PHONE DIRECTORY MENU ***
************************************************
 1 - View Contacts
 2 - Search For a Contact
 3 - Add Contact
 4 - Edit Contact
 5 - Delete Contact
 6 - Sort Contacts
 7 - Undo Last Action
 8 - View Most Accessed Contacts
 0 - Exit

How can I assist you? (0-8): 3

Enter name: Sidra
Enter phone number: 0989129980

*** Contact added successfully! ***
```

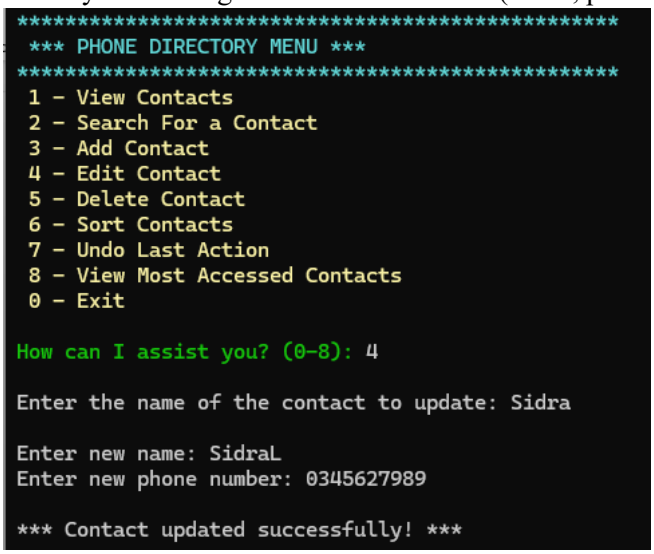
Delete Contact:

- Users can delete contacts from the directory.
- The action can be **undone** by the user.

```
 5 - Delete Contact
 6 - Sort Contacts
 7 - Undo Last Action
 8 - View Most Accessed Contacts
 0 - Exit

How can I assist you? (0-8): 5

Enter the name of the contact to delete: Sidra

*** Contact deleted successfully! ***
```

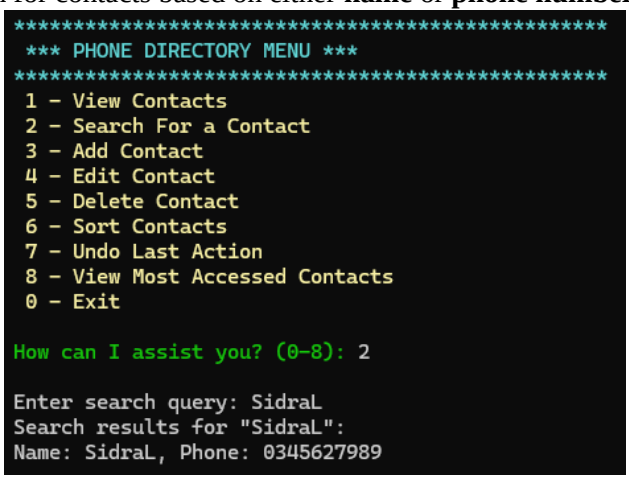
Update Contact:

- Allows users to modify an existing contact's information (name, phone number).



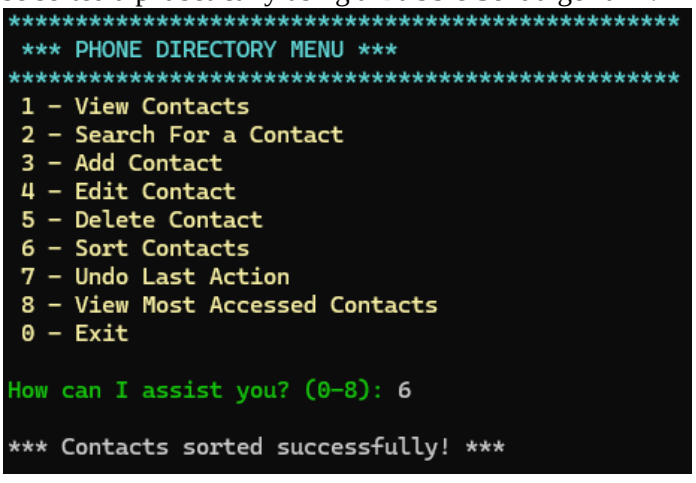
Search Contact:

- Users can search for contacts based on either **name** or **phone number**.



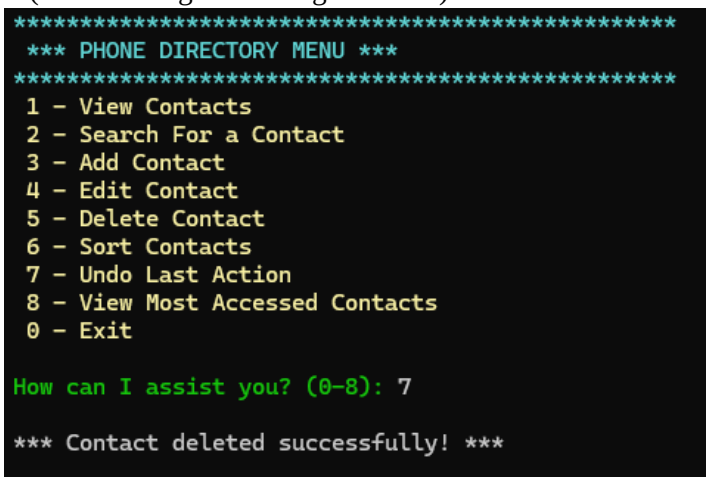
Sort Contacts:

- Contacts can be sorted alphabetically using a **Bubble Sort** algorithm.



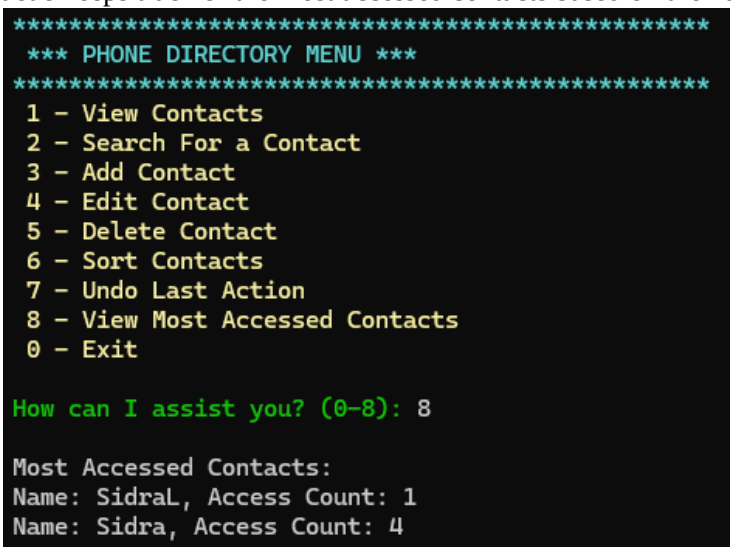
Undo Last Action:

- The last action (either adding or deleting a contact) can be undone.



Most Accessed Contacts:

- A priority queue keeps track of the most accessed contacts based on the number of searches.



Data Structures Used

Doubly Linked List:

- Each contact is represented by a node in a **doubly linked list**, which allows easy insertion, deletion, and traversal.
- Each node has:
  - name: Contact's name.
  - phoneNumber: Contact's phone number.
  - prev: Pointer to the previous node.
  - next: Pointer to the next node.

Stack:

- A stack is used to store **undo actions**. Whenever a contact is added or deleted, the action is pushed onto the stack to allow users to reverse it.

Priority Queue:

- The **unordered map** keeps track of contact access counts. The contacts with the highest access counts are stored in the map and can be displayed when requested.

System Design

The design of the system revolves around managing contact information efficiently. The core components include:

Contact Node Structure:

- A **Contact** node stores the information of each contact, as well as pointers to the next and previous contacts.

Contact List Management:

- Contacts are stored in a **doubly linked list**. New contacts are inserted in sorted order, ensuring that the directory remains organized.

Undo Mechanism:

- A **stack** is used to keep track of user actions, allowing the reversal of the last operation (add or delete).

Most Accessed Contacts:

- The **priority queue** stores contact names and their corresponding access counts. The application increments access counts every time a contact is searched.

Code Implementation

The implementation of the Phone Directory Application involves several key functions:

Contact Node Structure:

The Contact structure holds each contact's information and pointers to the previous and next nodes.

```
struct Contact {
   string name;
   string phoneNumber;
   Contact* prev;
   Contact* next;

   Contact(string n, string p) : name(n), phoneNumber(p), prev(nullptr), next(nullptr) {}
};
```

Adding a Contact:

New contacts are added to the list while maintaining the alphabetical order.

```
void addContact(Contact*& head, string name, string phone, ContactPriorityQueue& pq) {
   Contact* newContact = new Contact(name, phone);

   if (!head || head->name > name) {
      newContact->next = head;
      if (head) head->prev = newContact;
      head = newContact;
      undoStack.push("Add " + name);
      return;
```

```
    }

    Contact* current = head;
    while (current->next && current->next->name < name) {
        current = current->next;
    }

    newContact->next = current->next;
    if (current->next) current->next->prev = newContact;
    current->next = newContact;
    newContact->prev = current;
    undoStack.push("Add " + name);
    pq.incrementAccess(name);
}
```

Undo Feature:

The undo stack allows the user to revert the last performed action (add or delete).

```
void undoAction(Contact*& head, ContactPriorityQueue& pq) {
    if (undoStack.empty()) {
        cout << "\nNo actions to undo.\n";
        return;
    }

    string lastAction = undoStack.top();
    undoStack.pop();
    if (lastAction.find("Add") != string::npos) {
        string name = lastAction.substr(4); // Extract name
        deleteContact(head, name, pq); // Undo adding a contact
    }
    else if (lastAction.find("Delete") != string::npos) {
        string name = lastAction.substr(7); // Extract name
        cout << "\nUndo for 'Delete' action is not fully implemented.\n";
    }
}
```

Search and Most Accessed Contacts:

The access count is tracked using an unordered map, and users can view the most accessed contacts.

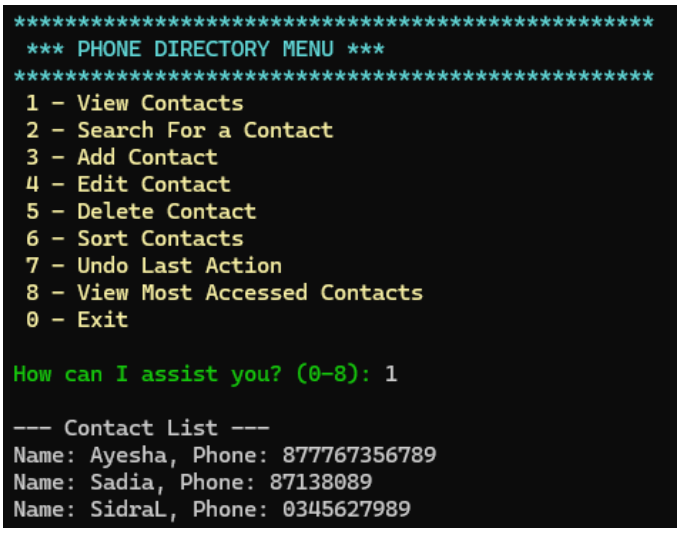
```
void searchContact(Contact* head, string query, ContactPriorityQueue& pq) {
    if (!head) {
        cout << "\nNo contacts available.\n";
        return;
    }
    Contact* current = head;
    bool found = false;
    while (current) {
        if (current->name.find(query) != string::npos || current->phoneNumber.find(query) != string::npos) {
            cout << "Found: " << current->name << " - " << current->phoneNumber << endl;
            found = true;
            pq.incrementAccess(current->name);
        }
        current = current->next;
    }
    if (!found) {
        cout << "\nNo matching contacts found.\n";
    }
}
```

Testing and Results

Menu Interaction:

- The menu is displayed for the user to select an option (view contacts, add, delete, search, etc.).
- The program responds to user inputs by performing the requested operations and displaying results.

Sample Output:



Conclusion

The Phone Directory Application is a fully functional program that allows users to manage their contacts efficiently. The system utilizes **doubly linked lists** for storing and managing contacts, **stacks** for undo functionality, and **unordered maps** for tracking contact access.