**TITLE : PHONEBOOK APPLICATION**

TABLE OF CONTENTS

1. Summary  
   1.1. Brief Overview of the Project  
   1.2. Objectives
2. Introduction  
   2.1. Background and Context  
   2.2. Problem Statement  
   2.3. Project Objectives  
   2.4. Scope of the Project
3. System Requirements  
   3.1. Functional Requirements
4. Design and Implementation  
   4.1. Class Diagrams  
   4.2. Data Structure Implementations  
   4.3. Algorithms and Their Complexities  
   4.4. Code Snippets  
   4.5. Output Screenshots
5. Conclusion  
   5.1. Summary  
   5.2. Achievement of Objectives  
   5.3. Future Work and Recommendations
6. References

**SUMMARY**

### Brief Overview of the Project

The Phonebook Management System is a command-line application built to manage a list of contacts. It allows users to add, delete, edit, search, display, and sort contacts stored in a linked list. Additionally, the system supports an undo feature that restores the phonebook's previous state using a stack. The project demonstrates the use of Object-Oriented Programming (OOP) principles and fundamental data structures such as linked lists, stacks, and algorithms for sorting and searching.

### Objectives

The objectives of this project are as follows:

1. To implement a phonebook system that stores and manages contact information.
2. To demonstrate the application of OOP principles in software development.
3. To design and implement a functional and user-friendly interface.
4. To incorporate the use of common data structures (linked list, stack) for storing and manipulating data.
5. To implement key features such as adding, deleting, editing, and sorting contacts.
6. To introduce an undo operation to revert any changes made to the phonebook.

**INTRODUCTION**

### Background and Context

A phonebook application is essential for organizing and managing contacts. In today's digital world, having a well-maintained digital phonebook is essential for both personal and business purposes. The Phonebook Management System offers a simple yet effective solution for managing contact details.

### Problem Statement

Managing contact information can become cumbersome without a well-organized system. Users need a way to easily add, search, edit, delete, and sort contacts. Furthermore, users may wish to revert any changes made during their interactions with the phonebook, which requires an undo mechanism.

### Project Objectives

The primary objectives of this project are to:

1. Create a functional and interactive phonebook system.
2. Use a linked list to dynamically store contacts.
3. Implement the ability to undo changes using a stack-based history mechanism.
4. Incorporate sorting and searching functionalities.
5. Apply fundamental data structures and algorithms in the implementation.

### Scope of the Project

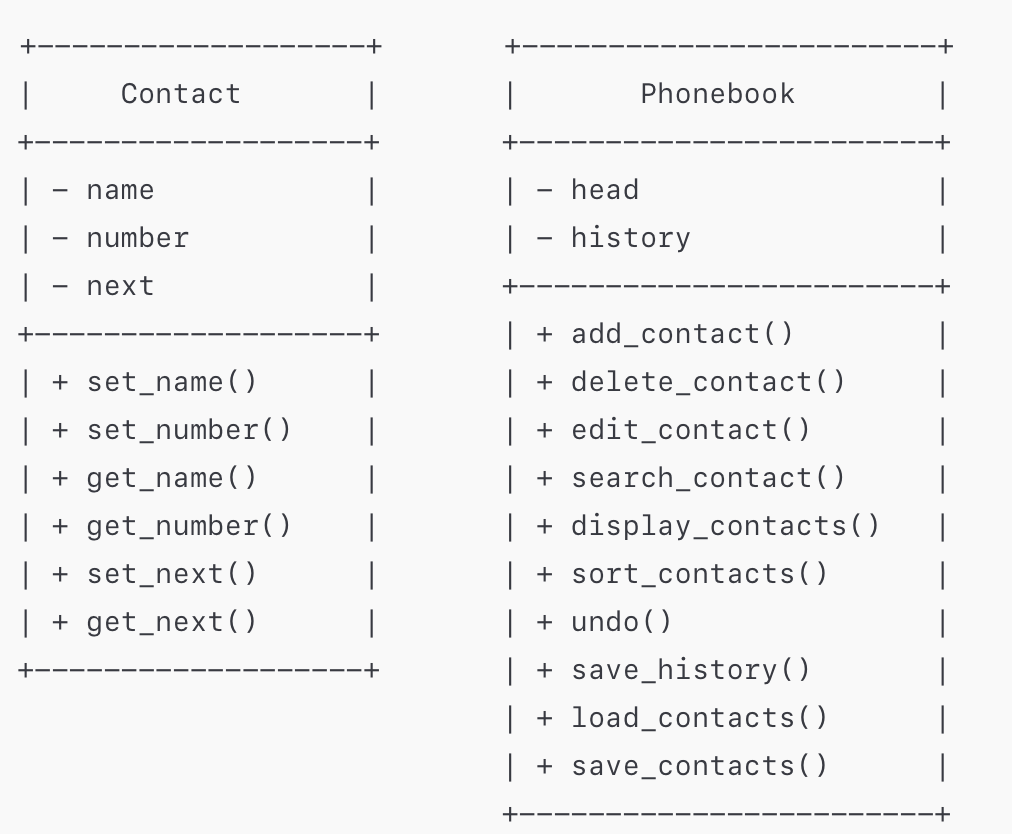
This project focuses on:

1. Implementing a command-line-based interface.
2. Developing the system with basic functionalities such as adding, deleting, editing, displaying, and sorting contacts.
3. Ensuring data integrity through an undo mechanism.
4. Supporting operations on a linked list of contacts.

**SYSTEM REQUIREMENTS**

### Functional Requirements

1. **Add Contact:** Allow users to add new contacts to the phonebook.
2. **Delete Contact:** Allow users to delete existing contacts from the phonebook.
3. **Edit Contact:** Allow users to update the contact information.
4. **Search Contact:** Allow users to search for a contact by name by using binary search algorithm.
5. **Sort Contacts:** Sort the contacts alphabetically by name by merge sort algorithm.
6. **Undo:** Undo the last modification to restore the previous state.
7. **File Handling:** Load contacts from a file when the system starts and save changes back to the file.
8. **Display Contacts**: The system must display all stored contacts with their names and phone numbers.

****

**DESIGN AND IMPLEMENTATION**

### Class Diagrams

The phonebook system involves two main classes: Contact and Phonebook class.

#### Contact Class:

* **Attributes:**
  + name: Stores the contact's name.
  + number: Stores the contact's phone number.
  + next: Points to the next contact in the list.
* **Methods:**
  + set\_name(), set\_number(): Modify contact details.
  + get\_name(), get\_number(): Retrieve contact details.
  + set\_next(), get\_next(): Manage the next contact in the list.

#### Phonebook Class:

* **Attributes:**
  + head: Points to the first contact in the list.
  + history: Stack to store the previous states of the phonebook for undo functionality.
* **Methods:**
  + add\_contact(),delete\_contact(),edit\_contact(), display\_contacts(), search\_contact(), sort\_contacts(), undo(),save\_contacts(),load\_contacts():Implement functionalities for managing the contacts.

### 

### 

### 

### Data Structure Implementations

1. **Linked List**: The contacts are stored in a linked list. Each contact is an object that contains information about the contact and a pointer to the next contact in the list.
2. **Stack**: A stack is used to save the state of the phonebook before any modifications. This allows users to undo actions by restoring the previous state.

**Algorithms and Their Complexities**

* **Merge Sort** for sorting the contact list:
  + Time complexity: **O(n log n)**
  + Space complexity: **O(n)**
* **Binary Search** for searching contacts:
  + Time complexity: **O(log n)** (for sorted lists)

**Code Snippets Demonstrating OOP and Data Structures**

class Contact {

private:

string name;

string number;

Contact\* next;

public:

Contact(string nameinput, string numberinput) {

name = nameinput;

number = numberinput;

next = NULL;

}

void set\_next(Contact\* next\_contact) {

next = next\_contact;

}

Contact\* get\_next() {

return next;

}

string get\_name() {

return name;

}

string get\_number() {

return number;

}

void set\_name(const string& nameinput) {

name = nameinput;

}

void set\_number(const string& numberinput) {

number = numberinput;

}

};

class Phonebook {

private:

Contact\* head;

stack<Contact\*> history;

void save\_history() {

Contact\* temp = head;

Contact\* prev = NULL;

Contact\* new\_head = NULL;

while (temp != NULL) {

Contact\* new\_contact = new Contact(temp->get\_name(), temp->get\_number());

if (prev == NULL) {

new\_head = new\_contact;

} else {

prev->set\_next(new\_contact);

}

prev = new\_contact;

temp = temp->get\_next();

}

history.push(new\_head);

}

Contact\* merge(Contact\* left, Contact\* right, bool by\_name) {

if (!left) return right;

if (!right) return left;

if (by\_name) {

if (left->get\_name() < right->get\_name()) {

left->set\_next(merge(left->get\_next(), right, by\_name));

return left;

} else {

right->set\_next(merge(left, right->get\_next(), by\_name));

return right;

}

} else {

if (left->get\_number() < right->get\_number()) {

left->set\_next(merge(left->get\_next(), right, by\_name));

return left;

} else {

right->set\_next(merge(left, right->get\_next(), by\_name));

return right;

}

}

}

Contact\* merge\_sort(Contact\* node, bool by\_name) {

if (!node || !node->get\_next()) return node;

Contact\* middle = get\_middle(node);

Contact\* left = node;

Contact\* right = middle->get\_next();

middle->set\_next(NULL);

left = merge\_sort(left, by\_name);

right = merge\_sort(right, by\_name);

return merge(left, right, by\_name);

}

Contact\* get\_middle(Contact\* node) {

if (!node) return node;

Contact\* slow = node;

Contact\* fast = node;

while (fast->get\_next() && fast->get\_next()->get\_next()) {

slow = slow->get\_next();

fast = fast->get\_next()->get\_next();

}

return slow;

}

void load\_contacts() {

ifstream infile("contacts.txt");

string name, number;

while (getline(infile, name) && getline(infile, number)) {

Contact\* new\_contact = new Contact(name, number);

if (head == NULL) {

head = new\_contact;

} else {

Contact\* temp = head;

while (temp->get\_next() != NULL) {

temp = temp->get\_next();

}

temp->set\_next(new\_contact);

}

}

infile.close();

}

void save\_contacts() {

ofstream outfile("contacts.txt");

Contact\* temp = head;

while (temp != NULL) {

outfile << temp->get\_name() << endl;

outfile << temp->get\_number() << endl;

temp = temp->get\_next();

}

outfile.close();

}

Contact\* binary\_search(Contact\* start, string target, bool by\_name) {

Contact\* left = start;

Contact\* right = NULL;

while (left != right) {

Contact\* middle = get\_middle(left);

if (!middle) return NULL;

if (by\_name) {

if (middle->get\_name() == target) {

return middle;

} else if (middle->get\_name() < target) {

left = middle->get\_next();

} else {

right = middle;

}

} else {

if (middle->get\_number() == target) {

return middle;

} else if (middle->get\_number() < target) {

left = middle->get\_next();

} else {

right = middle;

}

}

}

return NULL;

}

public:

Phonebook() {

head = NULL;

load\_contacts();

}

bool is\_digit(char c) {

return (c >= '0' && c <= '9');

}

bool is\_valid\_number(const string& number) {

if (number.length() != 10) return false;

for (int i = 0; i < number.length(); ++i) {

if (!is\_digit(number[i])) return false;

}

return true;

}

void add\_contact() {

string name, number;

cout << "Enter name: ";

cin >> name;

do {

cout << "Enter phone number (10 digits): ";

cin >> number;

if (!is\_valid\_number(number)) {

cout << "Invalid number. Please enter a valid 10-digit number." << endl;

}

} while (!is\_valid\_number(number));

save\_history();

Contact\* new\_contact = new Contact(name, number);

if (head == NULL) {

head = new\_contact;

} else {

Contact\* temp = head;

while (temp->get\_next() != NULL) {

temp = temp->get\_next();

}

temp->set\_next(new\_contact);

}

save\_contacts();

cout << "Contact added successfully." << endl;

}

void display\_contacts() {

if (head == NULL) {

cout << "Phonebook is empty." << endl;

return;

}

Contact\* temp = head;

cout << "Phonebook Contacts:" << endl;

while (temp != NULL) {

cout << "Name: " << temp->get\_name() << ", Phone Number: " << temp->get\_number() << endl;

temp = temp->get\_next();

}

}

void search\_contact\_by\_name() {

if (head == NULL) {

cout << "Phonebook is empty." << endl;

return;

}

string name;

cout << "Enter name to search: ";

cin >> name;

head = merge\_sort(head, true);

Contact\* result = binary\_search(head, name, true);

if (result) {

cout << "Phone Number of " << name << " is " << result->get\_number() << endl;

} else {

cout << "Contact not found." << endl;

}

}

void search\_contact\_by\_number() {

if (head == NULL) {

cout << "Phonebook is empty." << endl;

return;

}

string number;

cout << "Enter phone number to search: ";

cin >> number;

head = merge\_sort(head, false);

Contact\* result = binary\_search(head, number, false);

if (result) {

cout << "Contact found: " << result->get\_name() << ", Phone Number: " << result->get\_number() << endl;

} else {

cout << "Contact not found." << endl;

}

}

void delete\_contact() {

string name;

cout << "Enter name to delete: ";

cin >> name;

if (head == NULL) {

cout << "Phonebook is empty." << endl;

return;

}

save\_history();

if (head->get\_name() == name) {

Contact\* temp = head;

head = head->get\_next();

delete temp;

save\_contacts();

cout << "Contact deleted successfully." << endl;

return;

}

Contact\* current = head;

while (current->get\_next() != NULL && current->get\_next()->get\_name() != name) {

current = current->get\_next();

}

if (current->get\_next() != NULL) {

Contact\* temp = current->get\_next();

current->set\_next(temp->get\_next());

delete temp;

save\_contacts();

cout << "Contact deleted successfully." << endl;

} else {

cout << "Contact not found." << endl;

}

}

void edit\_contact() {

string name;

cout << "Enter the name of the contact you want to edit: ";

cin >> name;

Contact\* temp = head;

while (temp != NULL) {

if (temp->get\_name() == name) {

string new\_name, new\_number;

cout << "Enter new name: ";

cin >> new\_name;

do {

cout << "Enter new phone number (10 digits): ";

cin >> new\_number;

} while (!is\_valid\_number(new\_number));

save\_history();

temp->set\_name(new\_name);

temp->set\_number(new\_number);

save\_contacts();

cout << "Contact updated successfully." << endl;

return;

}

temp = temp->get\_next();

}

cout << "Contact not found." << endl;

}

void sort\_contacts() {

if (head == NULL || head->get\_next() == NULL) {

cout << "No contacts to sort." << endl;

return;

}

save\_history();

int choice;

cout << "Sort By:\n";

cout << "1. Name\n";

cout << "2. Number\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

head = merge\_sort(head, true);

cout << "Contacts sorted by Name." << endl;

} else if (choice == 2) {

head = merge\_sort(head, false);

cout << "Contacts sorted by Number." << endl;

} else {

cout << "Invalid choice." << endl;

return;

}

save\_contacts();

}

void empty\_phonebook() {

if (head == NULL) {

cout << "Phonebook is already empty." << endl;

return;

}

save\_history();

while (head != NULL) {

Contact\* temp = head;

head = head->get\_next();

delete temp;

}

save\_contacts();

cout << "Phonebook emptied successfully." << endl;

}

void undo() {

if (history.empty()) {

cout << "No action to undo." << endl;

return;

}

head = history.top();

history.pop();

save\_contacts();

cout << "Undo successful. Previous state restored." << endl;

}

~Phonebook() {

while (head != NULL) {

Contact\* temp = head;

head = head->get\_next();

delete temp;

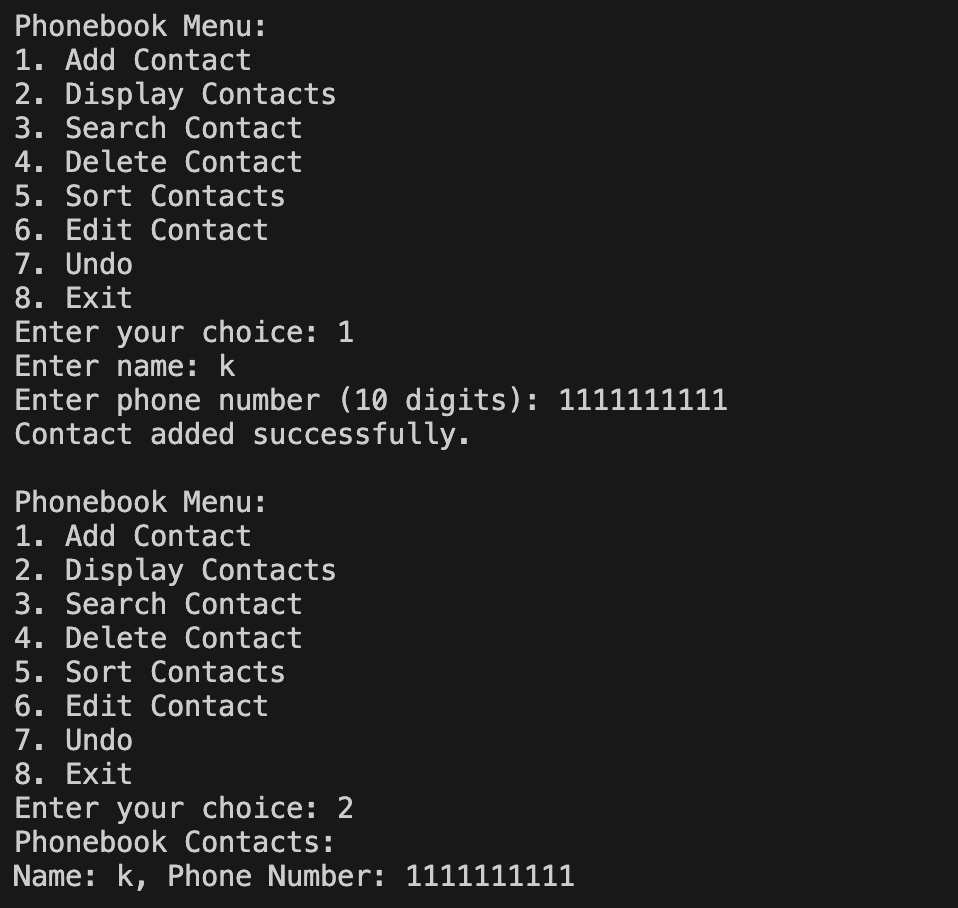
}

}

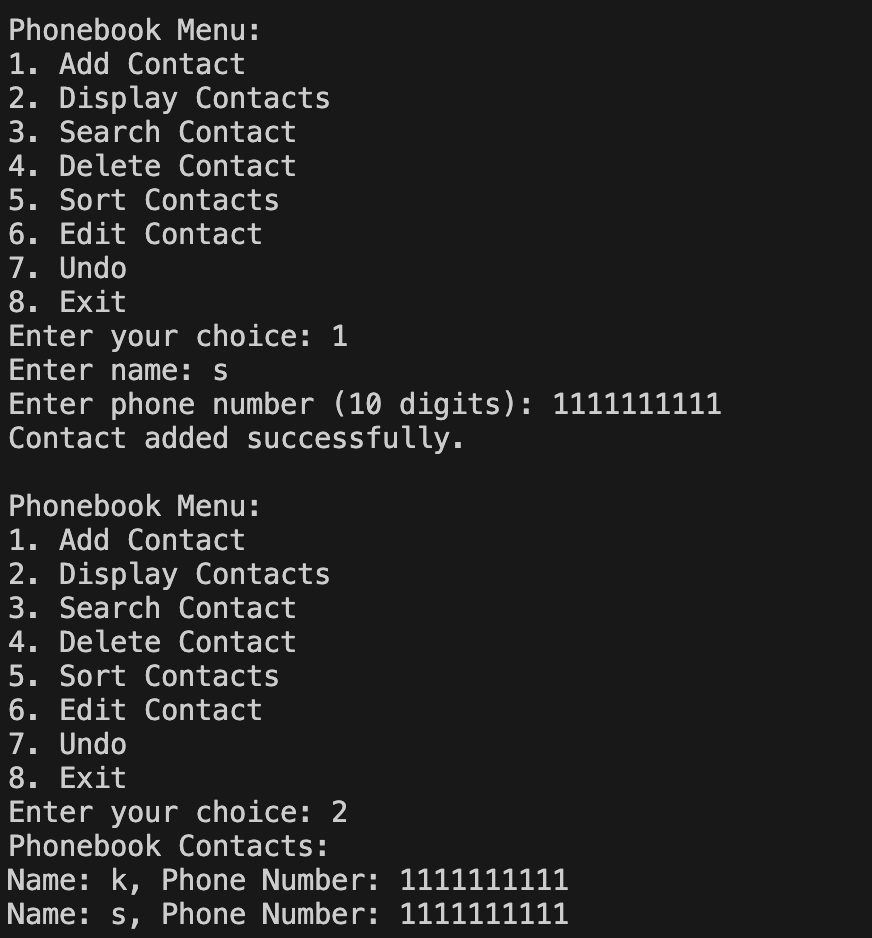
};

**Output screen shots with description**

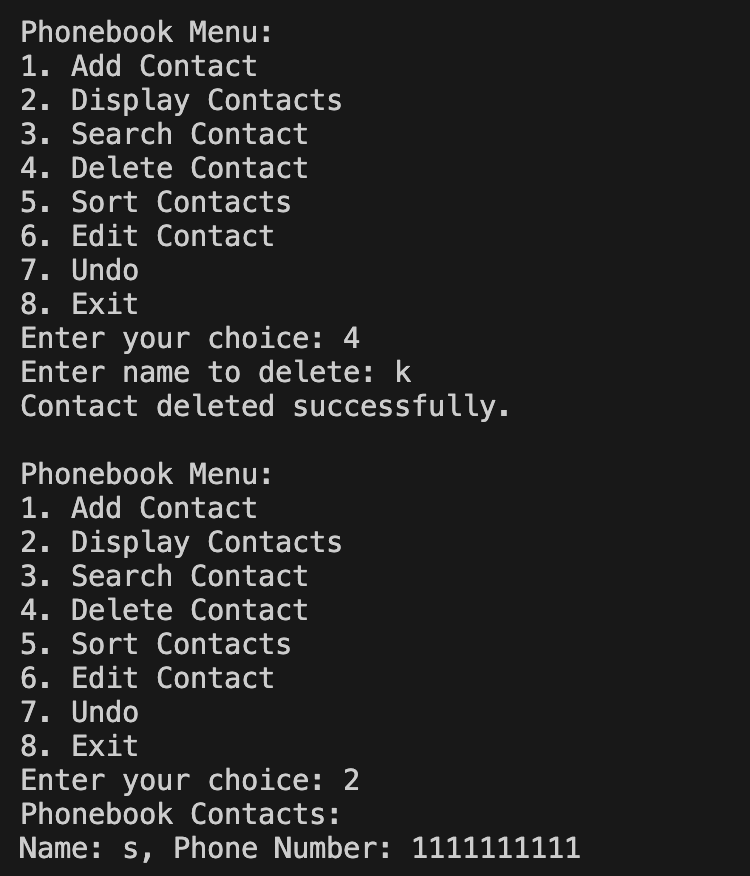
1. Added a contact named ‘k’ and displayed it:

****

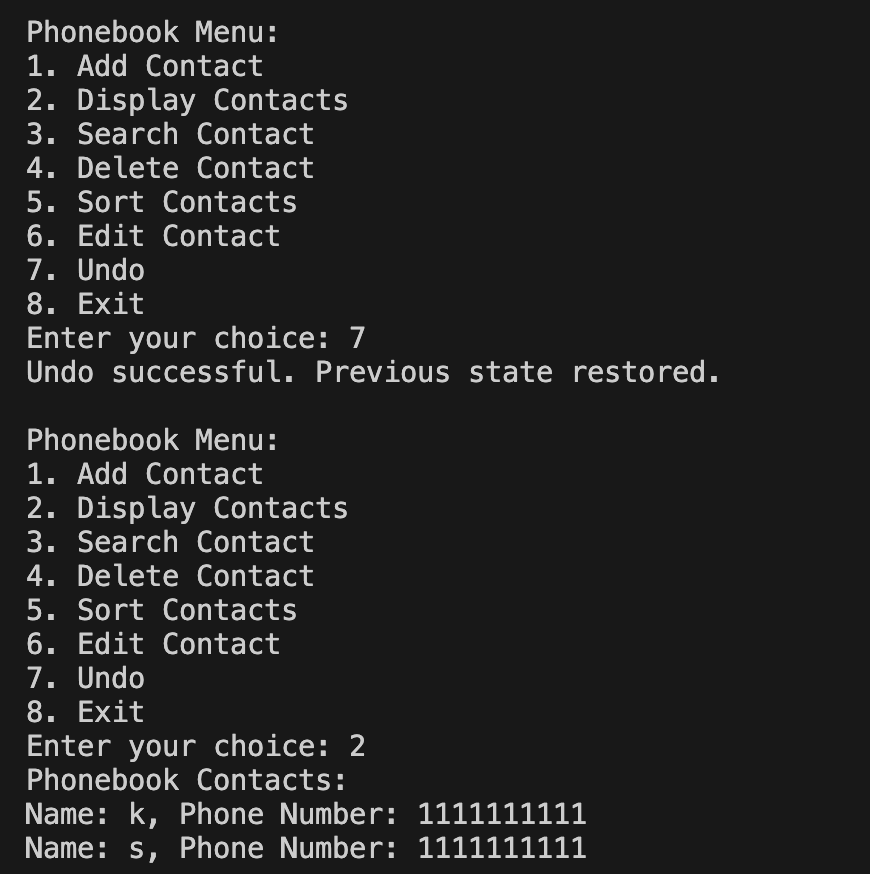
1. Added another contact named ‘s’ and displayed it:



1. Deletion of contact ‘k’:

****

1. Undo the deletion and now we again have 2 contacts:

****

1. Edit the contact with name ‘k’ and change it to ‘zz’:

## 

## 

## 6.Binary Search of a contact:

## 

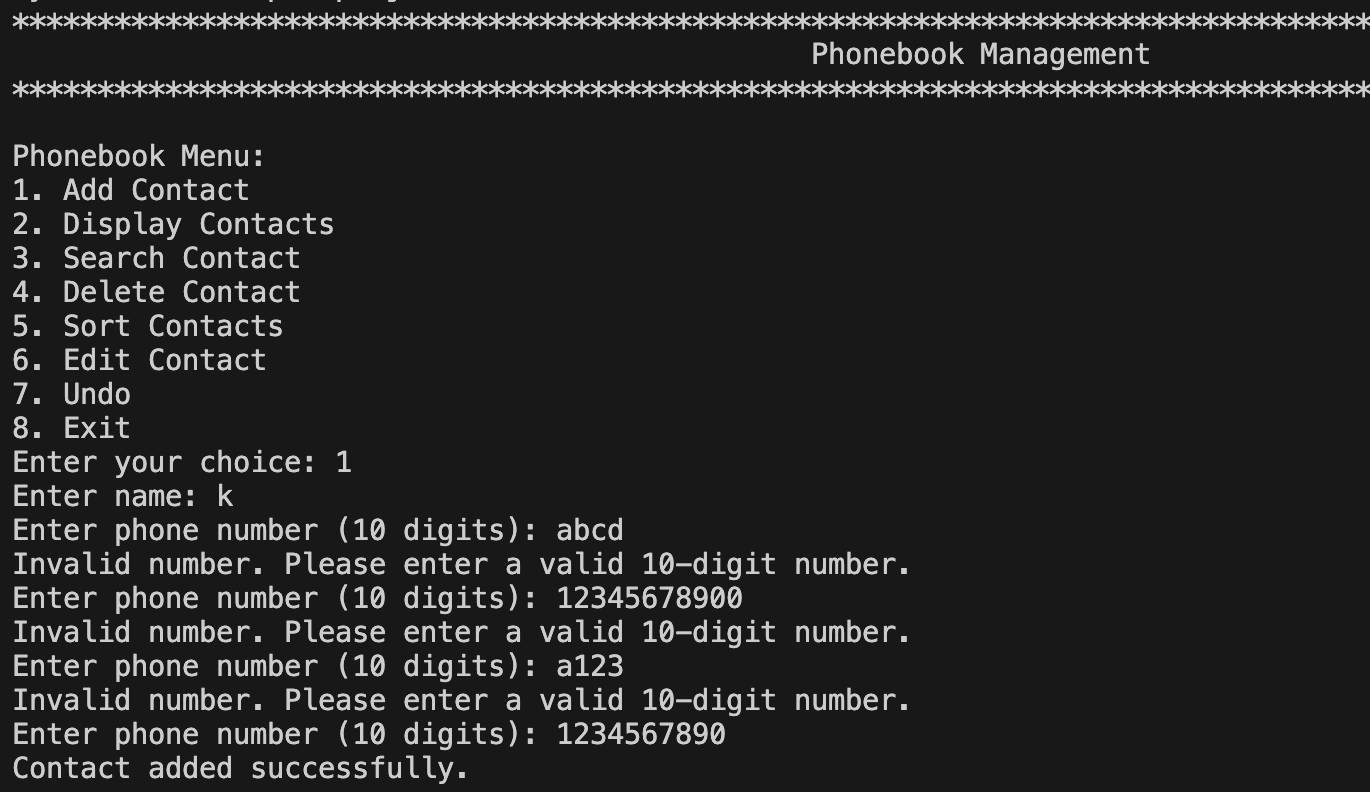
## 7.Merge sort:

## 

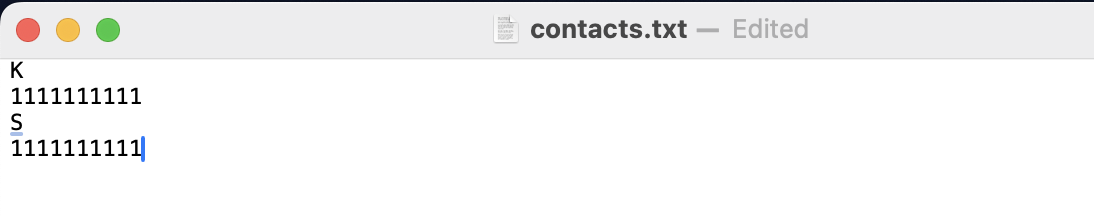
8.Exiting:

## 

9. Checks for valid phone number:



10.File saved the contacts:



**Conclusion**

### Summary

The Phonebook Management System is a C++ application that allows users to manage contact information using linked lists. Features such as undo and sorting make the system efficient and user-friendly. The use of file handling ensures that the contacts are saved between sessions.

### Achievement of Objectives

The project successfully met the outlined objectives, including the use of OOP principles and basic data structures such as linked lists and stacks. It provides a functional, user-friendly application with all specified features.Contacts are also saved to and loaded from a file .

### Future Work and Recommendations

Future improvements could include:

* Use more advanced data structures like balanced trees for faster search and insertion.
* Enhance the user interface for better user experience.

**References**

* <https://cplusplus.com/>
* <https://www.geeksforgeeks.org/>
* <https://www.w3schools.com/cpp/cpp_oop.asp>
* Lecture Slides: DS & SDF-II