# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

# Jnana Sangama, Belagavi - 590018



### **MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING**

*Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution*

(*A Unit of Rajalaxmi Education Trust®, Mangalore - 575001*)

Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi

*Badaga Mijar, Moodabidri-574225, Karnataka*

2022-23



Department of Information Science &Engineering

**Mini project**

***On***

**“PHONE BOOK MANAGEMENT SYSTEM”**

**By**

**AKHILESH K 4MT21IS003**

**RAO ASHLESHA PADMANANDA 4MT21IS031**

**SAMAN MOHAMMED SHERIF 4MT21IS034**

**SAMARTH Y R 4MT21IS035**

**VINOD K S 4MT21IS060**

### 

**Abstract**

The Phone Book Management System in C is a versatile and user-centric software solution designed for efficient contact information organization and retrieval. It offers a straightforward interface that simplifies the management of contacts, providing users with easy-to-use tools for adding, updating, deleting, and searching for contacts. Whether for personal or business purposes, this system ensures the integrity and accessibility of contact data. Users can effortlessly maintain their contact lists, enhancing productivity and communication. With its user-friendly design and essential features, the Phone Book Management System in C serves as a valuable tool for anyone looking to streamline contact management tasks effectively.

**Introduction**

In today's fast-paced world, managing contact information efficiently is essential for both individuals and businesses. A Phone Book Management System in C is a practical and useful software application designed to help users store, retrieve, and manage their contacts seamlessly. This project aims to provide a user-friendly interface for storing, editing, and searching contact information, making it a valuable tool for personal and professional use.

**Background :**

The primary purpose of a Phone Book Management Project is to provide a digital solution for managing and organizing contact information. Users can add, view, edit, and delete contacts within the phone book.

**Objectives :**

* ***Contact Management***: The primary objective is to create a system for managing contact information, including names, phone numbers, email addresses, and other relevant details.
* ***User-Friendly Interface***: Design a user-friendly interface that allows users to easily add, edit, delete, and search for contacts within the phone book.
* ***Data Storage***: Implement a data storage mechanism to store contact information persistently, ensuring that data is not lost when the program is closed and reopened.
* ***Search Functionality***: Provide efficient search functionality to allow users to find contacts by name, phone number, or other criteria quickly.
* ***Data Validation***: Implement data validation to ensure that users enter valid and consistent contact information.
* ***Error Handling***: Implement robust error-handling mechanisms to handle unexpected situations and provide helpful error messages to users.
* ***Maintenance and Updates***: Plan for future maintenance and updates to address bug fixes, security vulnerabilities, and feature enhancements.

**Technologies Used**

**C Programming Language**: The code is written in the C programming language, which is a widely used and popular programming language for system-level and application-level programming.

**Standard C Libraries**: The code utilizes several standard C libraries, including:

**<stdio.h>:** For input and output operations like printf and scanf.

**<stdlib.h>:** For memory allocation functions like malloc and free.

**<string.h>:** For string manipulation functions like strcpy, strtok, and strstr.

**<ctype.h>**: For character classification and manipulation functions like isdigit.

**File I/O**: The code uses file input and output operations for saving and loading contact data to/from a file. It relies on standard C functions for file handling.

**Dynamic Memory Allocation**: The code dynamically allocates memory for contact structures using malloc. This allows it to manage a linked list of contacts efficiently.

**Data Structures**: The code employs a linked list data structure to store and manage contact information. This data structure is implemented using a custom-defined structure (struct Contact) and pointers.

**User Input**: The code takes user input for various operations using scanf to read user choices and contact information.

**Conditional Statements and Loops**: It uses conditional statements (e.g., if, switch) and loops (e.g., do-while, while) for control flow and decision-making.

**System Architecture**

**Front-End** : The front-end of the system is implemented using the console or command-line interface, where users interact with the system by entering commands and providing input.

**Back-End :** The back-end of the system consists of C programming logic that performs various operations users to add, update, delete, and search for contacts,It also handles file handling to store and retrieve data.

**Database :** The system uses files to act as databases for storing information about medicines, suppliers, and customers. Each entity has a separate file for data storage.

**Project Modules :**

**Module 1-Interface (UI) Module:**

- Responsible for handling user input and displaying information to the user.

- It can use functions like `printf` and `scanf` to interact with the user.

**Module 2-Contact Data Structure Module:**

- Define a structure to represent a contact with fields like name, phone number, email, etc.

- Functions for creating, updating, and deleting contacts.

**Module 3-File I/O Module:**

- Functions for reading contacts from a file into memory and saving contacts from memory to a file.

- This module can use file handling functions like `fopen`, `fread`, and `fwrite`.

**Module 4:Search Module:**

- Implement functions to search for contacts based on different criteria (e.g., name, phone number).

- The search module can use algorithms like linear search or binary search, depending on how contacts are stored.

**Module 5-Add/Modify/Delete Module:**

- Implement functions to add new contacts, modify existing contacts, and delete contacts from the phone book.

**Module 6-Sorting Module:**

- Implement sorting algorithms (e.g., bubble sort, quicksort) to sort the contacts in various ways (e.g., by name, by phone number).

**Module 7-Display Module:**

- Functions for displaying contacts in a user-friendly format.

- Options to display all contacts or a specific contact.

**Module 8-Menu Navigation Module:**

- Handle the menu system and user navigation through the program.

- Allow the user to choose different actions like adding a contact, searching, sorting, etc.

**Module 9-Validation Module:**

- Implement functions to validate user inputs, such as ensuring that phone numbers have the correct format or that email addresses are valid.

**Module 10-Memory Management Module:**

- Handle dynamic memory allocation and deallocation for the contact data structures.

- Avoid memory leaks and manage memory efficiently.

**Module 11-Error Handling Module:**

- Manage errors gracefully by providing appropriate error messages and handling exceptional situations.

**Module 12-Main Program Module**:

- The main function that ties everything together. It initializes the program, presents the user interface, and calls the various modules' functions based on user input.

**Design and Implementation**

**Front-End Design :**

• The front-end design is based on a command-line interface, where users can select options by entering numbers corresponding to their desired actions.

**Back-End Design :**

• The back-end logic is implemented in the C programming language.

• Each module has its functions for adding, displaying, modifying, and deleting contacts.

• File handling functions are used to perform operations on data files.

**Database Design :**

• Data for contacts and users are stored in separate files.

• Each file contains contacts in a structured format

**Features and Functionality :**

* Contact Management: Add, edit, and delete contacts.
* Search: Search for contacts by name or phone number.
* Display: Show a list of all contacts.
* Sorting: Organize contacts alphabetically.
* Data Storage: Save and load contacts from a file.
* User Interface: Provide a user-friendly command-line interface.
* Error Handling: Handle invalid inputs and file errors.
* Data Validation: Ensure data integrity and security.
* Data Import/Export: Import and export contacts to/from external files.
* Data Backup: Create backups of the phone book data.

**Testing**

**Unit Testing** :

• Each module is tested individually to ensure that it performs its functions correctly. • Test cases are created to cover various scenarios.

**Integration Testing :**

• Modules are integrated to test the overall functionality of the system. • Data flow between modules is tested.

**User Acceptance Testing :**

• The system is tested by end-users to ensure it meets their requirements and expectations.

• Any feedback or issues raised by users are addressed and resolved.

**Challenges Faced**

• Implementing file handling for data storage and retrieval.

• Ensuring data consistency and accuracy.

• Handling errors and exceptions gracefully.

• Future Enhancements

• Adding features for generating reports and statistics.

• Implementing security measures to protect data.

**Conclusion**

In conclusion, the Phone Book Management Project in C has successfully addressed the need for an efficient and user-friendly system to manage contacts. This project aimed to create a reliable and robust solution for storing, organizing, and retrieving contact information. Through the course of development, several significant accomplishments and observations have been made.

**CODE AND OUTPUT:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

struct Contact {

char name[50];

char phoneNumber[15];

char email[100];

struct Contact\* next;

};

void adminMenu();

void userMenu();

void addContact();

void searchContact();

void displayContacts();

void updateContact();

void deleteContact();

void saveContactsToFile();

void loadContactsFromFile();

struct Contact\* head = NULL;

int main() {

int choice;

int isAdmin = 0;

printf("Phone Book Management System\n");

printf("Are you an admin? (1 for Admin, 0 for User): ");

scanf("%d", &isAdmin);

loadContactsFromFile();

if (isAdmin) {

adminMenu();

} else {

userMenu();

}

struct Contact\* current = head;

while (current) {

struct Contact\* next = current->next;

free(current);

current = next;

}

return 0;

}

void adminMenu() {

int choice;

do {

printf("\nAdmin Menu\n");

printf("1. Add Contact\n");

printf("2. Search Contact\n");

printf("3. Display Contacts\n");

printf("4. Update Contact\n");

printf("5. Delete Contact\n");

printf("6. Save Contacts to File\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addContact();

break;

case 2:

searchContact();

break;

case 3:

displayContacts();

break;

case 4:

updateContact();

break;

case 5:

deleteContact();

break;

case 6:

saveContactsToFile();

break;

case 7:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

break;

}

} while (choice != 7);

}

void userMenu() {

int choice;

do {

printf("\nUser Menu\n");

printf("1. Search Contact\n");

printf("2. Display Contacts\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

searchContact();

break;

case 2:

displayContacts();

break;

case 3:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

break;

}

} while (choice != 3);

}

void addContact() {

struct Contact\* newContact = (struct Contact\*)malloc(sizeof(struct Contact));

if (newContact == NULL) {

printf("Memory allocation failed. Unable to add contact.\n");

return;

}

printf("Enter Name: ");

scanf(" %[^\n]", newContact->name);

char phoneNumber[15];

int isValidPhoneNumber = 0;

do {

printf("Enter Phone Number (up to 10 digits, no letters): ");

scanf(" %[^\n]", phoneNumber);

isValidPhoneNumber = 1;

for (int i = 0; phoneNumber[i] != '\0'; i++) {

if (!isdigit(phoneNumber[i])) {

isValidPhoneNumber = 0;

break;

}

}

if (!isValidPhoneNumber || strlen(phoneNumber) > 10) {

printf("Invalid phone number. Please enter a valid phone number with up to 10 digits and no letters.\n");

}

} while (!isValidPhoneNumber || strlen(phoneNumber) > 10);

strcpy(newContact->phoneNumber, phoneNumber);

char email[100];

int isValidEmail = 0;

do {

printf("Enter Email (must contain '@gmail.com'): ");

scanf(" %[^\n]", email);

isValidEmail = (strstr(email, "@gmail.com") != NULL);

if (!isValidEmail) {

printf("Invalid email. Please enter an email containing '@gmail.com'.\n");

}

} while (!isValidEmail);

strcpy(newContact->email, email);

newContact->next = NULL;

if (head == NULL) {

head = newContact;

} else {

struct Contact\* current = head;

while (current->next) {

current = current->next;

}

current->next = newContact;

}

printf("Contact added successfully.\n");

}

void searchContact() {

if (head == NULL) {

printf("Phone book is empty. No contacts to search.\n");

return;

}

char searchTerm[50];

printf("Enter Name or Phone Number to search: ");

scanf(" %[^\n]", searchTerm);

struct Contact\* current = head;

int found = 0;

while (current) {

if (strcmp(current->name, searchTerm) == 0 || strcmp(current->phoneNumber, searchTerm) == 0) {

printf("Contact found:\n");

printf("Name: %s\n", current->name);

printf("Phone Number: %s\n", current->phoneNumber);

printf("Email: %s\n", current->email);

found = 1;

break;

}

current = current->next;

}

if (!found) {

printf("Contact not found.\n");

}

}

void displayContacts() {

if (head == NULL) {

printf("Phone book is empty. No contacts to display.\n");

return;

}

printf("Contacts in the Phone Book:\n");

struct Contact\* current = head;

while (current) {

printf("Name: %s\n", current->name);

printf("Phone Number: %s\n", current->phoneNumber);

printf("Email: %s\n", current->email);

printf("-----------------------\n");

current = current->next;

}

}

void updateContact() {

if (head == NULL) {

printf("Phone book is empty. No contacts to update.\n");

return;

}

char searchTerm[50];

printf("Enter Name or Phone Number of the contact to update: ");

scanf(" %[^\n]", searchTerm);

struct Contact\* current = head;

int found = 0;

while (current) {

if (strcmp(current->name, searchTerm) == 0 || strcmp(current->phoneNumber, searchTerm) == 0) {

printf("Contact found:\n");

printf("1. Update Name\n");

printf("2. Update Phone Number\n");

printf("3. Update Email\n");

printf("4. Cancel\n");

printf("Enter your choice: ");

int updateChoice;

scanf("%d", &updateChoice);

switch (updateChoice) {

case 1:

printf("Enter updated Name: ");

scanf(" %[^\n]", current->name);

break;

case 2:

// Validate and update phone number (similar to the addContact function)

char phoneNumber[15];

int isValidPhoneNumber = 0;

do {

printf("Enter updated Phone Number (up to 10 digits, no letters): ");

scanf(" %[^\n]", phoneNumber);

isValidPhoneNumber = 1;

for (int i = 0; phoneNumber[i] != '\0'; i++) {

if (!isdigit(phoneNumber[i])) {

isValidPhoneNumber = 0;

break;

}

}

if (!isValidPhoneNumber || strlen(phoneNumber) > 10) {

printf("Invalid phone number. Please enter a valid phone number with up to 10 digits and no letters.\n");

}

} while (!isValidPhoneNumber || strlen(phoneNumber) > 10);

strcpy(current->phoneNumber, phoneNumber);

break;

case 3:

// Validate and update email (similar to the addContact function)

char email[100];

int isValidEmail = 0;

do {

printf("Enter updated Email (must contain '@gmail.com'): ");

scanf(" %[^\n]", email);

isValidEmail = (strstr(email, "@gmail.com") != NULL);

if (!isValidEmail) {

printf("Invalid email. Please enter an email containing '@gmail.com'.\n");

}

} while (!isValidEmail);

strcpy(current->email, email);

break;

case 4:

printf("Update canceled.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

break;

}

printf("Contact updated successfully.\n");

found = 1;

break;

}

current = current->next;

}

if (!found) {

printf("Contact not found.\n");

}

}

void deleteContact() {

if (head == NULL) {

printf("Phone book is empty. No contacts to delete.\n");

return;

}

char searchTerm[50];

printf("Enter Name or Phone Number of the contact to delete: ");

scanf(" %[^\n]", searchTerm);

struct Contact\* current = head;

struct Contact\* previous = NULL;

int found = 0;

while (current) {

if (strcmp(current->name, searchTerm) == 0 || strcmp(current->phoneNumber, searchTerm) == 0) {

if (previous == NULL) {

head = current->next;

} else {

previous->next = current->next;

}

free(current);

printf("Contact deleted successfully.\n");

found = 1;

break;

}

previous = current;

current = current->next;

}

if (!found) {

printf("Contact not found.\n");

}

}

void saveContactsToFile() {

FILE\* file = fopen("phonebook.txt", "w");

if (file == NULL) {

printf("Error opening file for writing.\n");

return;

}

struct Contact\* current = head;

while (current) {

fprintf(file, "%s,%s,%s\n", current->name, current->phoneNumber, current->email);

current = current->next;

}

fclose(file);

printf("Contacts saved to file successfully.\n");

}

void loadContactsFromFile() {

FILE\* file = fopen("phonebook.txt", "r");

if (file == NULL) {

return;

}

char line[200];

while (fgets(line, sizeof(line), file)) {

char\* name = strtok(line, ",");

char\* phoneNumber = strtok(NULL, ",");

char\* email = strtok(NULL, "\n");

struct Contact\* newContact = (struct Contact\*)malloc(sizeof(struct Contact));

if (newContact == NULL) {

printf("Memory allocation failed. Unable to load contact.\n");

return;

}

strcpy(newContact->name, name);

strcpy(newContact->phoneNumber, phoneNumber);

strcpy(newContact->email, email);

newContact->next = NULL;

if (head == NULL) {

head = newContact;

} else {

struct Contact\* current = head;

while (current->next) {

current = current->next;

}

current->next = newContact;

}

}

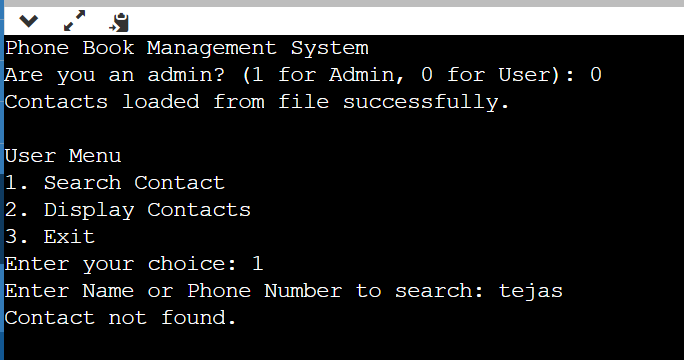
fclose(file);

printf("Contacts loaded from file successfully.\n");

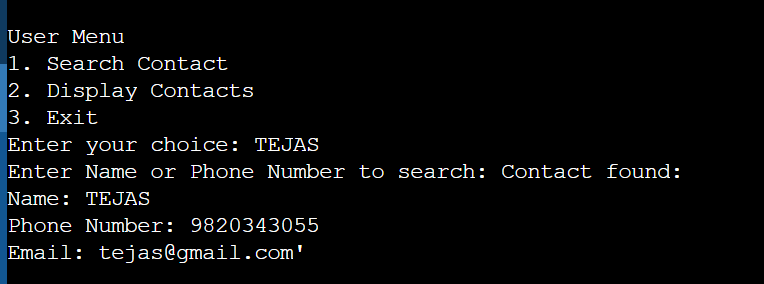
}

**OUTPUT/EXECUTION:**

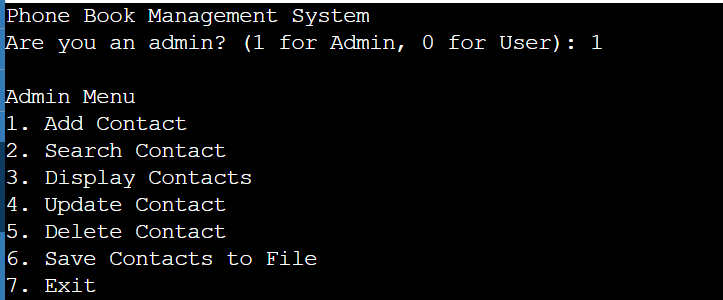
1)User Logged IN and unsuccessful contact search



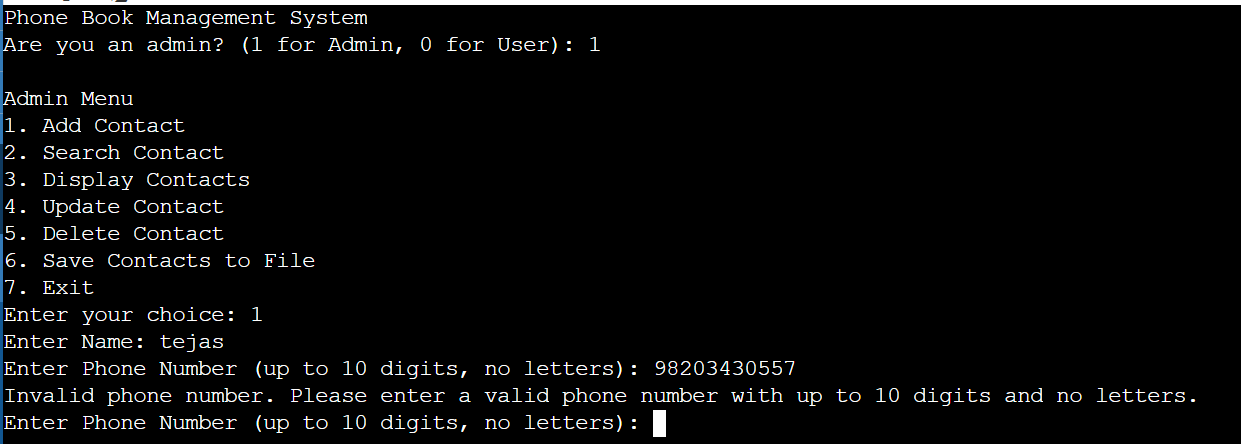
2)User Logged In and Successful contact search

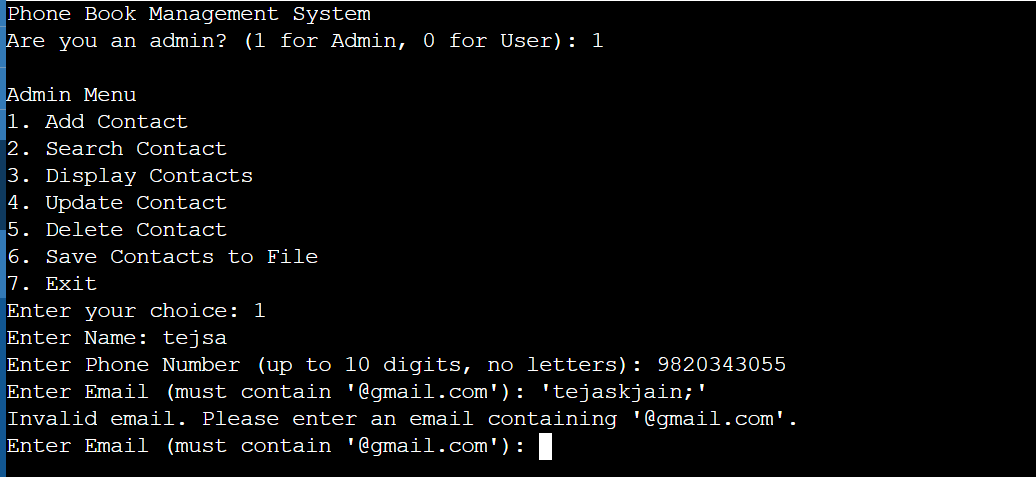


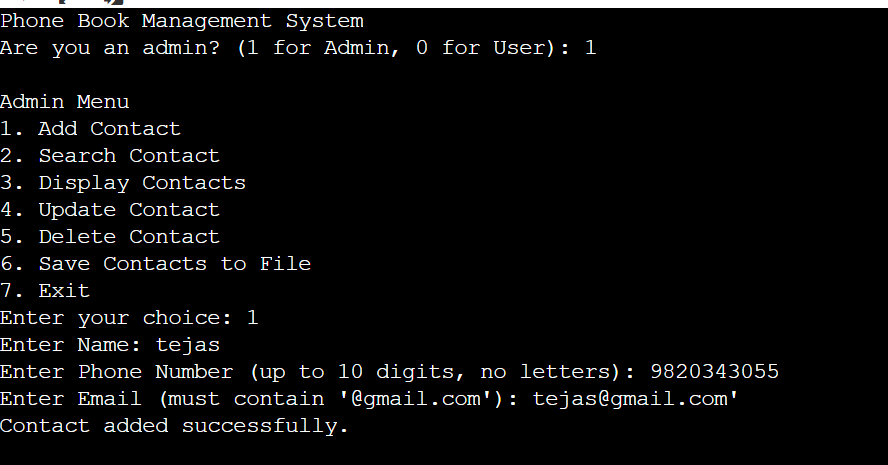
3)Admin Login

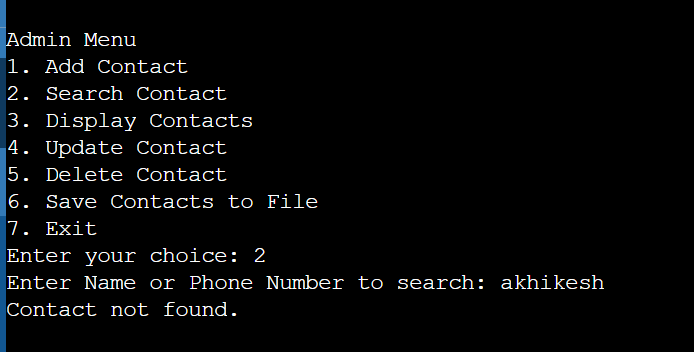


4)Add contact option with invalid phone number

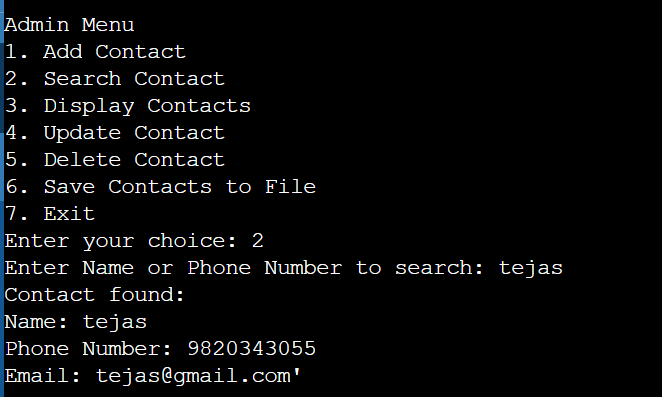


5)Add contact option with invalid email id

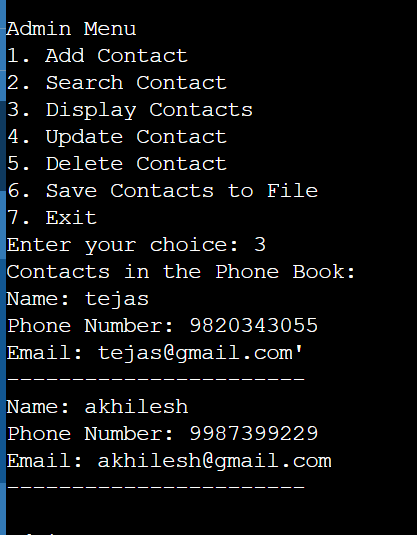
6)Successful Add contact

7)Unsuccessful Search contact

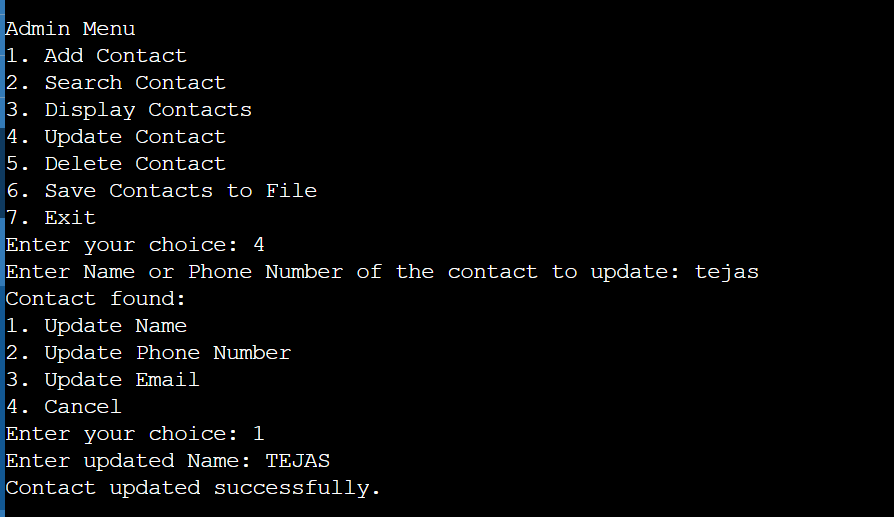
8)Successful search contact



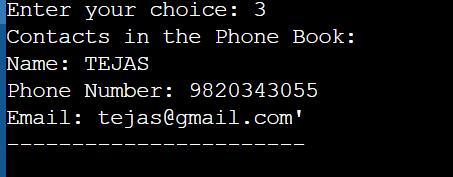
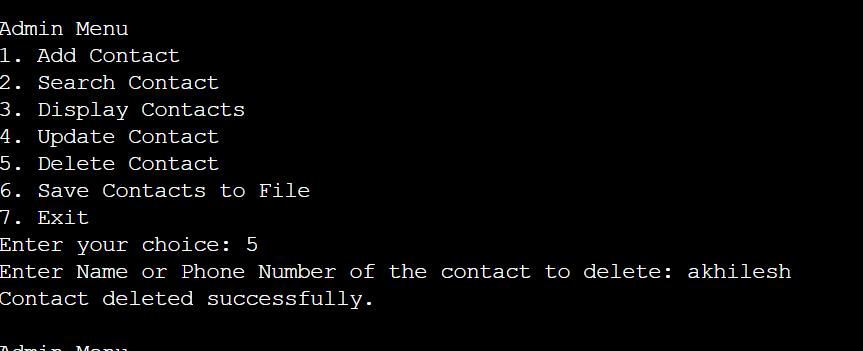
9)Display Contacts



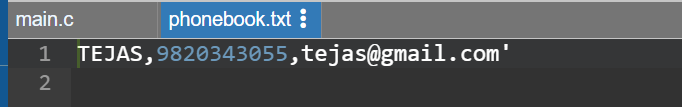
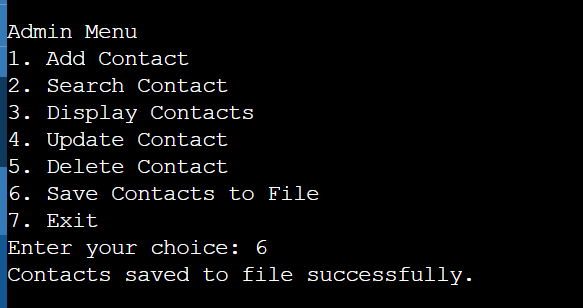
10) Update Contact



11)Delete Contact



12)Save contacts to File



**References**

🡪 <https://practice.geeksforgeeks.org>

🡪 <https://stackoverflow.com/>

🡪<https://miro.com/>

🡪<https://draw.io/>