****

**CHANDIGARH UNIVERSITY**

**DEPARTMENT: UIC**

Bachelors of Computer Application

Subject Name: Computer Programming

24CAH-101

PROJECT

Project name: **Simple Address Book**

Submitted by:

Members name:

* **DHARIKA (24BCD10089)**
* **TANYA SAXSENA (24BCD10084)**
* **PARAMJOT SINGH (24BCD10095)**

Submitted to: Mrs. Shuchi Sharma

**Project Description:**

An application for managing and storing contact information in an address book.

**Features of the project:**

* Contacts can be updated, deleted, and added.
* Use the search option to locate contacts based on additional parameters or by name.
* Contact details are stored through file processing. fundamental input data validation.

# 

# **Address-Book**

Address Book is a small application written in C language. It keeps track of names and telephone/mobile numbers and e-mail addresses. It is a console based application which uses standard I/O for adding and deleting contact names, phone numbers and e-mail addresses, searching names and associated numbers and email addresses, updating numbers and email addresses, and deleting contacts.

**ALGORITHM**

Step 1: Start Step 2: Read the file name Step 3: Read option:

1. Insert
2. Edit
3. Search
4. Delete
5. Display

INSERT

Step 1: Open the file in append mode Step 2: Read information from the user like name, email, phone number and city Step 3: Add spaces after each data upto 32 characters for each Step 4: Write the data into the file Step 5: Close the fileS

EDIT

Step 1: Open the file in read write mode Step 2: Store all the data in array of stuctures Step 3: Read the name which is to be edited Step 4: Read option to edit by: Name Email Phone City Step 4: Read the new data Step 5: Compare the new data with the name entered Step 6: If matched return the index value of the array in which the previous content is stored and repalce it by writing the new content to the file in that postion Step 7: Close the file

DELETE

Step 1: Open the file in read mode Step 2: Store all the datas in array of structures and close the file Step 3: Open the file in write mode Step 4: Read the name whose contact is to be deleted Step 5: Compare the name with all the array names Step 6: If the found skip that index and write the reamaining array elemnts back to the file Step 7: Close the file

SEARCH

Step 1: Open the file in read mode Step 2: Store all the datas in array of sturctures Step 3: Read the the option to search by: Name Email Phone City Step 4: Compare the read content with the content in the array Step 5: If found print index of the array and print the contents of the array with that index Step 6: Close the file

Code of the program

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NAME\_LENGTH 50

#define MAX\_PHONE\_LENGTH 15

#define MAX\_EMAIL\_LENGTH 50

// Structure to store a contact's information

typedef struct Contact {

char name[MAX\_NAME\_LENGTH];

char phone[MAX\_PHONE\_LENGTH];

char email[MAX\_EMAIL\_LENGTH];

struct Contact \*next; // Pointer to next contact (linked list)

} Contact;

// Function prototypes

Contact\* createContact();

void addContact(Contact \*\*head);

void displayContacts(Contact \*head);

void freeContacts(Contact \*head);

int main() {

Contact \*head = NULL; // Head of the linked list

int choice;

while (1) {

// Menu

printf("\nSimple Address Book (Linked List)\n");

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

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

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // Consume newline left by scanf

switch (choice) {

case 1:

// Add a contact

addContact(&head);

break;

case 2:

// Display contacts

if (head == NULL) {

printf("No contacts available!\n");

} else {

displayContacts(head);

}

break;

case 3:

// Exit program

freeContacts(head); // Free all allocated memory before exiting

printf("Exiting the Address Book. Goodbye!\n");

exit(0);

default:

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

}

}

return 0;

}

// Function to create and initialize a new contact

Contact\* createContact() {

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

if (newContact == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

printf("Enter name: ");

fgets(newContact->name, MAX\_NAME\_LENGTH, stdin);

newContact->name[strcspn(newContact->name, "\n")] = '\0'; // Remove newline

printf("Enter phone number: ");

fgets(newContact->phone, MAX\_PHONE\_LENGTH, stdin);

newContact->phone[strcspn(newContact->phone, "\n")] = '\0'; // Remove newline

printf("Enter email address: ");

fgets(newContact->email, MAX\_EMAIL\_LENGTH, stdin);

newContact->email[strcspn(newContact->email, "\n")] = '\0'; // Remove newline

newContact->next = NULL; // Initialize next to NULL

return newContact;

}

// Function to add a contact to the linked list

void addContact(Contact \*\*head) {

Contact \*newContact = createContact();

if (\*head == NULL) {

\*head = newContact; // If the list is empty, the new contact is the first one

} else {

Contact \*temp = \*head;

while (temp->next != NULL) {

temp = temp->next; // Traverse to the end of the list

}

temp->next = newContact; // Add new contact at the end

}

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

}

// Function to display all contacts in the linked list

void displayContacts(Contact \*head) {

Contact \*temp = head;

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

while (temp != NULL) {

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

printf("Phone: %s\n", temp->phone);

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

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

temp = temp->next;

}

}

// Function to free all allocated memory for contacts

void freeContacts(Contact \*head) {

Contact \*temp;

while (head != NULL) {

temp = head;

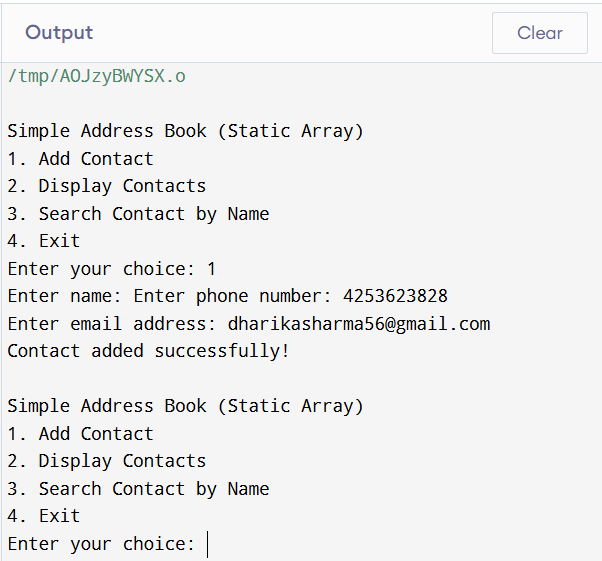
head = head->next;

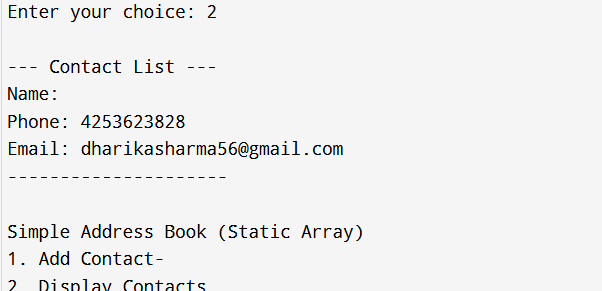
free(temp); // Free the memory allocated for each contact

}

}

**Output**





**Summary and Conclusion**

### Summary:

The project on a \*\*Simple Address Book\*\* in C programming involves creating a software application that allows users to store, search, update, and delete contact information such as names, phone numbers, and email addresses. The primary goal of the project is to practice basic programming concepts such as data structures (arrays or linked lists), input/output handling, and user interface design within a console application.

In this project:

- Users can input contact details such as name, phone number, and email.

- The program provides functionality to display the stored contacts, search for specific contacts, update existing contact details, and delete contacts.

- The project may also involve implementing file handling to store contact data persistently, ensuring that the data is saved even after the program is closed.

- Simple error handling is incorporated to ensure that invalid input is avoided, and the program runs smoothly.

### Conclusion:

The Simple Address Book project in C programming provides an excellent opportunity to apply fundamental programming concepts like arrays, functions, and file operations in a practical setting. By building such a project, developers improve their ability to manage user inputs, manipulate data, and manage memory efficiently. Moreover, it introduces key elements of software design such as modularity, error handling, and user interaction.

This project also lays the foundation for more complex applications, where data structures like linked lists or databases could be used for more advanced contact management. By understanding the implementation details in a basic address book, developers can move on to creating larger, more feature-rich applications while maintaining good programming practices.

### Learning Objectives:

The \*\*Simple Address Book\*\* project in C programming provides a hands-on approach to learning several core programming concepts. The key learning objectives are:

1. \*\*Understanding Data Structures\*\*:

- Learn to use arrays (or linked lists) for storing multiple data entries (contacts).

- Understand how to access, modify, and iterate over data in a structured way.

2. \*\*File Handling in C\*\*:

- Implement file operations (such as reading from and writing to a file) to store contact data persistently.

- Learn how to open, close, read, and write data to a file using standard C file handling functions.

3. \*\*Handling User Input/Output\*\*:

- Practice accepting user inputs from the console and handling different data types (e.g., strings, integers).

- Develop the ability to format and display data effectively to the user in a readable manner.

4. \*\*Function Decomposition and Modularity\*\*:

- Understand the importance of breaking down a program into smaller, reusable functions.

- Practice designing modular code for tasks such as adding, searching, updating, and deleting contacts.

5. \*\*Memory Management and Error Handling\*\*:

- Learn how to handle memory allocation (using `malloc`/`free`) in dynamic data structures (if applicable).

- Implement basic error handling techniques to validate user inputs and manage edge cases (e.g., invalid input, out-of-bounds array access).

6. \*\*Debugging and Testing\*\*:

- Develop debugging skills by testing different parts of the code to ensure correct functionality.

- Learn how to identify and resolve issues related to user input, memory allocation, and data integrity.

7. \*\*Basic User Interface Design\*\*:

- Gain experience designing simple text-based user interfaces (UI) that allow users to interact with the program.

8. \*\*Data Persistence and Program Persistence\*\*:

- Learn how to persist user data between program runs, ensuring the program doesn't lose contact data when closed.

Through these learning objectives, you will gain practical experience in both the core aspects of C programming and the broader field of software development.