Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2023), B.Sc. in CSE (Day)

Course Title: Data Structures Lab
Course Code: CSE 106 Section: DC

Lab Project Name: Phonebook Management System.

Student Details

	Name	ID
1.	Shoaib Ahmed	212902019

Submission Date : 19/6/2023
Course Teacher's Name : Farhana Akter Sunny

[For Teachers use only: Don't Write Anything inside this box]

Lab Project Status

Marks: …… **Signature: .....**

Comments: ..... **Date: .....**

Table of Contents

Chapter 1

Introduction

1.1 Introduction

The Phonebook Management System is a user-friendly software application designed to efficiently organize and manage contact information. In today's fast-paced digital world, staying connected and having access to contact details is vital for effective communication. The Phonebook Management System aims to streamline the process of storing, retrieving, and updating contact information, eliminating the need for traditional paper-based phonebooks.

With this system, users can easily store an extensive database of contacts, including names, phone numbers, email addresses, and additional notes. It provides a centralized platform where users can conveniently search for specific contacts, add new entries, edit existing details, and remove outdated information. The software also offers various features, such as categorizing contacts into groups, setting reminders for important events, and backing up the data to ensure its security and availability.

Overall, the Phonebook Management System is a reliable and practical tool that brings order and convenience to the task of contact management. By embracing this digital solution, users can say goodbye to the hassle of flipping through pages in a traditional phonebook and embrace a modern, efficient approach to maintaining their contact information.

1.2 Design Goals/Objective

1. **Efficient Contact Management:** The system should enable users to efficiently manage their contacts by providing features such as adding new contacts, editing existing ones, deleting contacts etc. It should also allow for advanced search and filtering capabilities to quickly find specific contacts based on various criteria.
2. **User-Friendly Interface:** The system should have an intuitive and user-friendly interface that makes it easy for users to navigate, add, update, and search for contacts. A well-designed interface enhances user experience and increases productivity by minimizing the learning curve and reducing the time required to perform common tasks.
3. **Security and Privacy**: Protecting the privacy and security of contact information is crucial. The system should incorporate measures to ensure that only authorized individuals can access and modify the contact database. This includes implementing user authentication, role-based access control, and data encryption to safeguard sensitive information.

Chapter 2

Development/Implementation of the Project

2.1 Algorithm :

1. Start the program.
2. Display the current date and time.
3. Display the welcome message for the Phonebook Management System.
4. Declare a variable choice to store the user's menu choice
5. Repeat the following steps until the user chooses to exit (choice = 5):
    a. Display the menu options.
    b. Read the user's choice.
    c. Clear the input buffer.
    d. Use a switch statement to perform the action based on the user's choice:

    Case 1: Add Contact
    - Allocate memory for a new contact structure.
    - If memory allocation fails, display an error message and return.
    - Read the name and phone number of the contact from the user.
    - Remove the newline character from the input.
    - Set the next pointer of the new contact to the current phonebook.
    - Update the phonebook pointer to the new contact.
    - Display a success message.

    Case 2: Search Contact.
    - Read the name to search from the user.
    - Remove the newline character from the input.
    - Initialize a variable found to 0.
    - Iterate through the phonebook:
        - If a contact with a matching name is found, display the contact details and set found to 1.
    - If found is still 0, display a "contact not found" message.

    Case 3: Delete Contact
    - Read the name to delete from the user.
    - Remove the newline character from the input.
    - Initialize variables found to 0, current to the phonebook, and prev to NULL.
    - Iterate through the phonebook:
        - If a contact with a matching name is found:
            - If it's the first contact, update the phonebook pointer.
            - Otherwise, update the next pointer of the previous contact.

        - Free the memory of the current contact.
        - Set found to 1 and display a success message.
    - If found is still 0, display a "contact not found" message.

Case 4: Display Phonebook

- If the phonebook is empty, display an "empty phonebook" message.
- Otherwise, iterate through the phonebook and display the name and phone number of each contact.

Case 5: Exit

- Display a farewell message. e. Display a new line.

6. Free the memory allocated for the phonebook.
7. End the program.

2.2 Implementation:

C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <time.h>

struct Contact {
    char name[50];
    char phone[15];
    struct Contact* next;
};

struct Contact* phonebook = NULL;

void addContact() {
    struct Contact* newContact = (struct Contact*)malloc(sizeof(struct Contact));
    if (newContact == NULL) {
        printf("\n\t\t\t Failed to allocate memory for new contact.\n");
        return;
    }

    printf("\n\t\t\t Enter name: ");
    fgets(newContact->name, sizeof(newContact->name), stdin);
    newContact->name[strcspn(newContact->name, "\n")] = '\0';

    printf("\n\t\t\t Enter phone number: ");
    fgets(newContact->phone, sizeof(newContact->phone), stdin);
    newContact->phone[strcspn(newContact->phone, "\n")] = '\0';
```

```
    newContact->next = phonebook;
    phonebook = newContact;

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

void searchContact() {
    char searchName[50];
    printf("\n\t\t\t Enter name to search: ");
    fgets(searchName, sizeof(searchName), stdin);
    searchName[strcspn(searchName, "\n")] = '\0';

    struct Contact* current = phonebook;
    int found = 0;

    while (current != NULL) {
        if (strcmp(current->name, searchName) == 0) {
            printf("\n\t\t\t Name: %s\n\t\t\tPhone: %s\n", current->name, current->phone);
            found = 1;
            break;
        }
        current = current->next;
    }

    if (!found) {
        printf("\n\t\t\t Contact not found.\n");
    }
}

void deleteContact() {
    char deleteName[50];

    printf("\n\t\t\t Enter name to delete: ");
    fgets(deleteName, sizeof(deleteName), stdin);
    deleteName[strcspn(deleteName, "\n")] = '\0';

    struct Contact* current = phonebook;
    struct Contact* prev = NULL;
    int found = 0;

    while (current != NULL) {
        if (strcmp(current->name, deleteName) == 0) {
            if (prev == NULL) {
                phonebook = current->next;
            } else {
```

```
            prev->next = current->next;
        }
        free(current);
        found = 1;
        printf("\n\t\t\t Contact deleted successfully.\n");
        break;
    }
    prev = current;
    current = current->next;
  }

  if (!found) {
    printf("\n\t\t\t Contact not found.\n");
  }
}

void displayPhonebook() {
  if (phonebook == NULL) {
    printf("\n\t\t\t Phonebook is empty.\n");
    return;
  }

  printf("\n\t\t\t Contact list:\n");
  struct Contact* current = phonebook;

  while (current != NULL) {
    printf("\n\t\t\t Name: %s\n\t\t\t Phone: %s\n", current->name, current->phone);
    current = current->next;
  }
}

void freePhonebook() {
  struct Contact* current = phonebook;
  struct Contact* temp;

  while (current != NULL) {
    temp = current;
    current = current->next;
    free(temp);
  }
}
int main() {

  system("color 3F");
  time_t now;
  time(&now);
```

```
    printf("\n");
    printf("\t\t\t\t\t    %s",ctime(&now));
    printf("\n\t\t\t\t  ======== Welcome to Phonebook Management System ========\n");

    int choice;

    do {
        printf("\n\n\t\t\t\t  ============ Phonebook Management System ============");
        printf("\n\t\t 1. Add Contact\n");
        printf("\n\t\t 2. Search Contact\n");
        printf("\n\t\t 3. Delete Contact\n");
        printf("\n\t\t 4. Display Phonebook\n");
        printf("\n\t\t 5. Exit\n");
        printf("\n\t\t Enter your choice: ");

        scanf("%d", &choice);
        getchar(); // Clear input buffer

        switch (choice) {
            case 1:
                addContact();
                break;
            case 2:
                searchContact();
                break;
            case 3:
                deleteContact();
                break;
            case 4:
                displayPhonebook();
                break;
            case 5:
                printf("\n\t\t\t Thanks for using my phonebook.");
                break;
            default:
                printf("\n\t\t\t Invalid choice. Please try again.\n");
        }

        printf("\n");
    } while (choice != 5);

    // Free memory before exiting
    freePhonebook();

    return 0;
}
```

Chapter 3

Performance Evaluation

3.1 Simulation Procedure

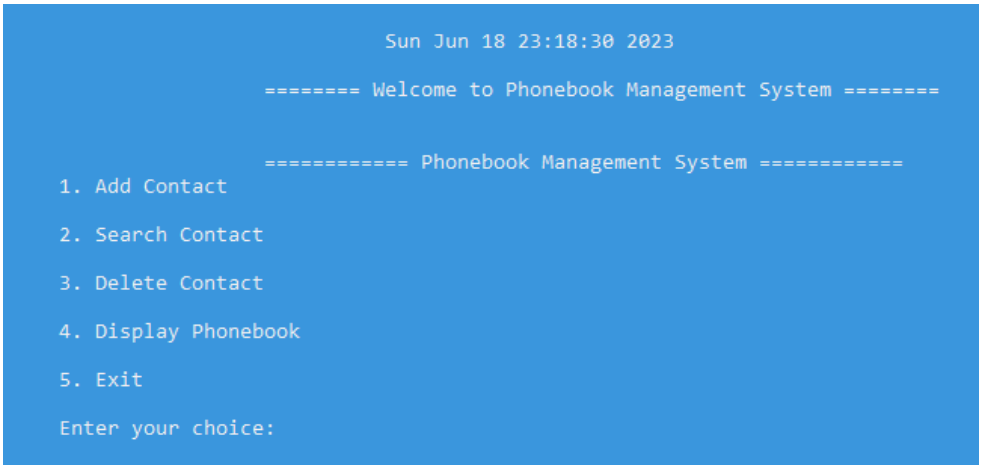

Fig 3.1: System Starting Menu

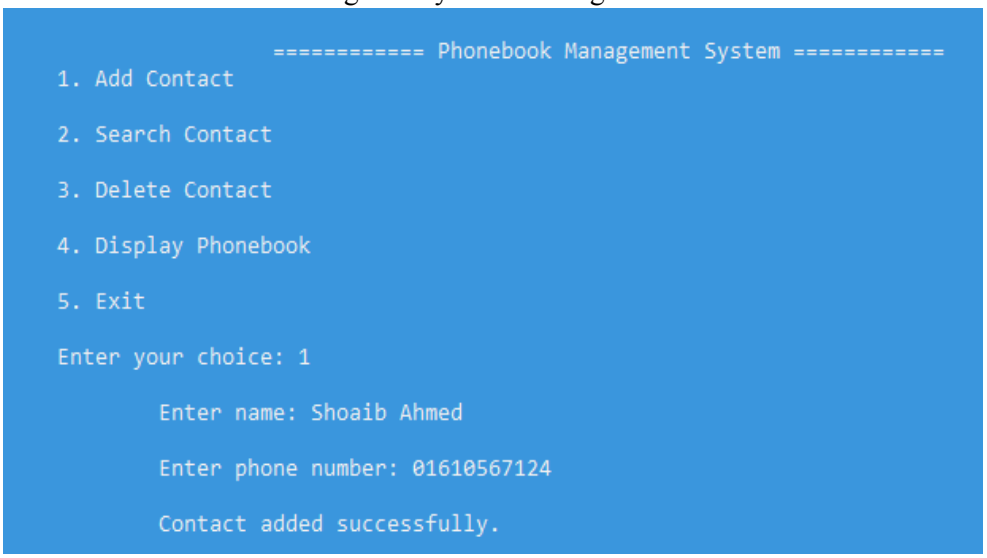

Fig 3.2: Adding a Contact

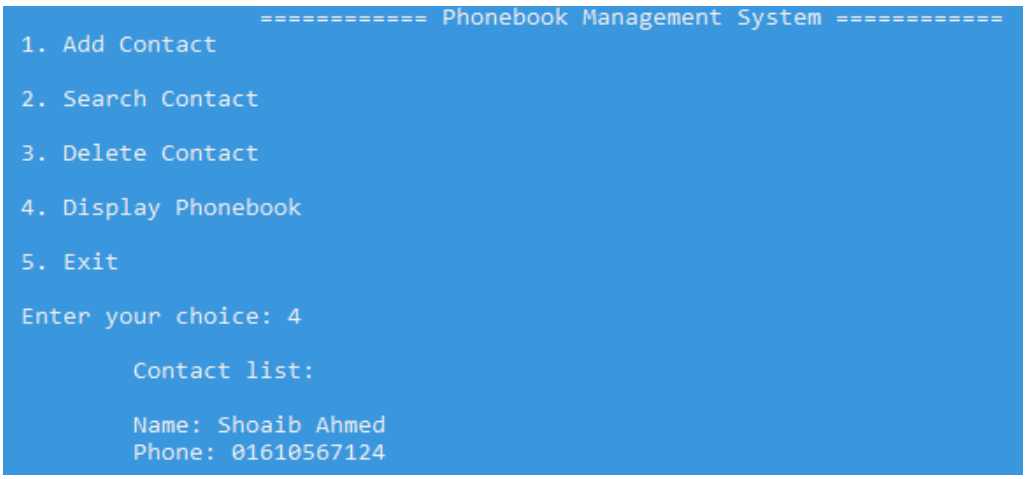

Fig 3.3: Display the contacts

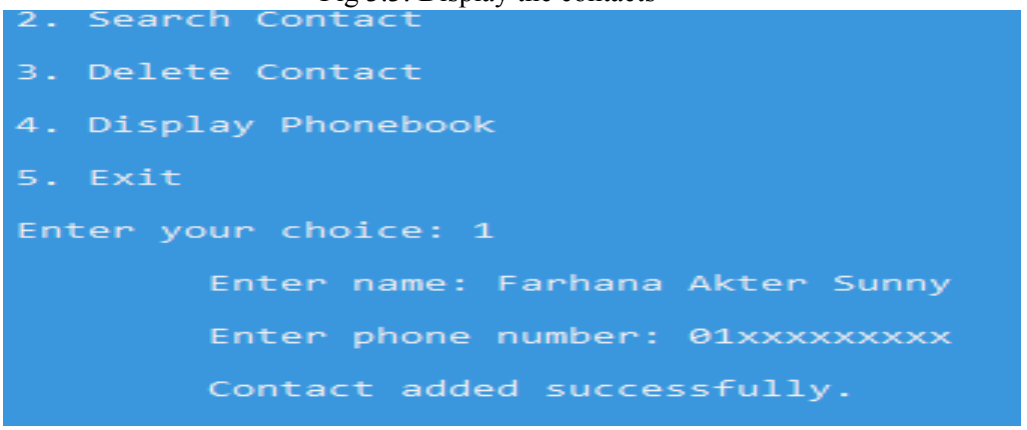

Fig 3.4: Adding another contact

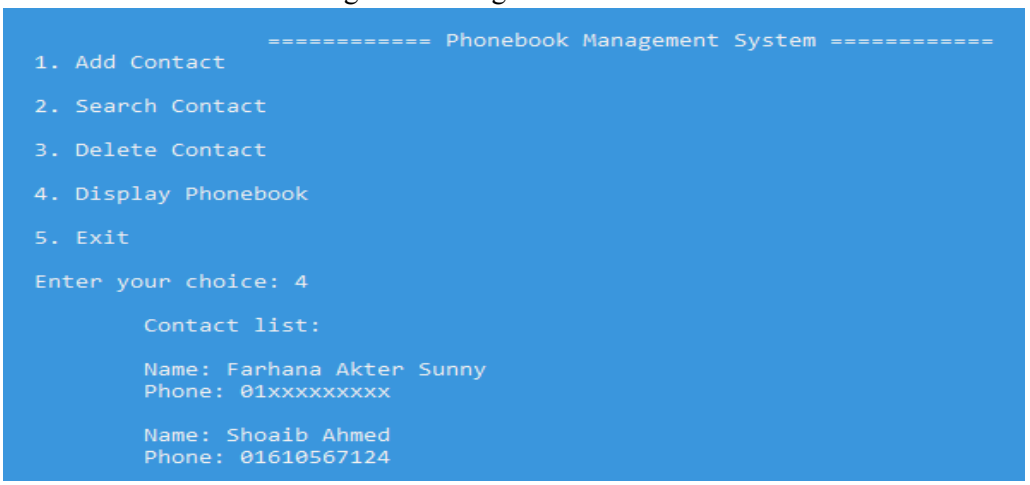

Fig 3.5: Display again after adding another number

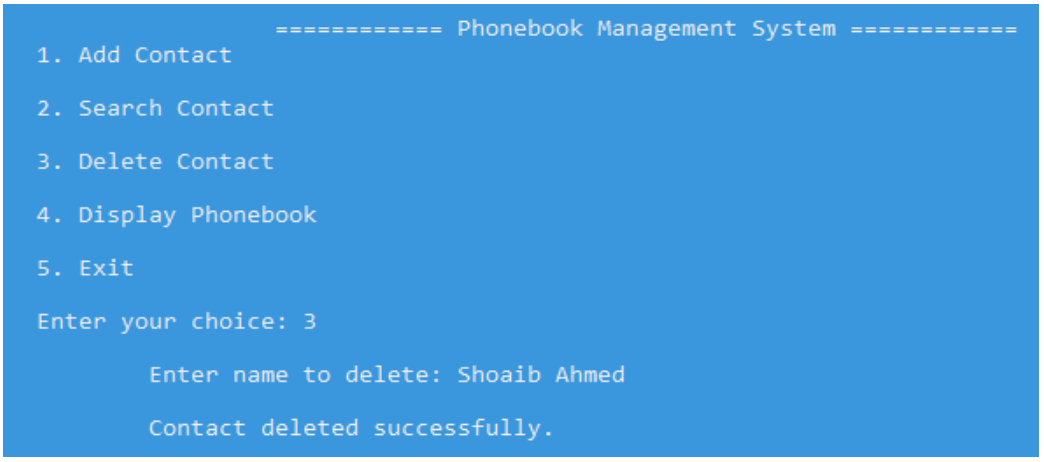

Fig 3.6: Deleting a contact

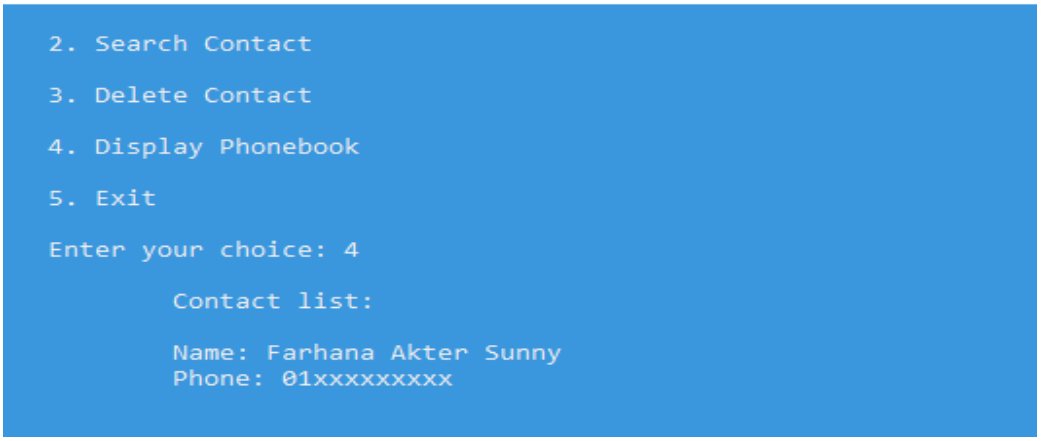

Fig 3.7: Display contact list after deleting one contact.

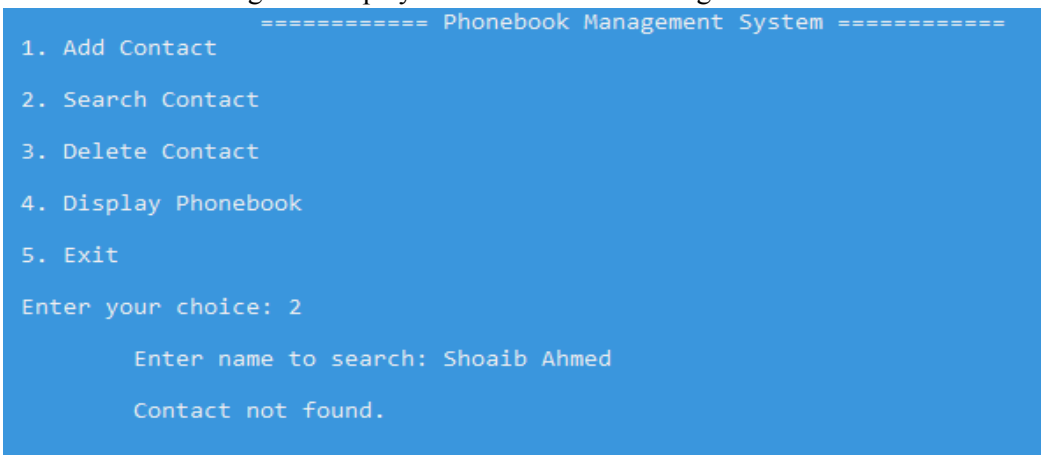

Fig 3.8: Searching a contact (not found)

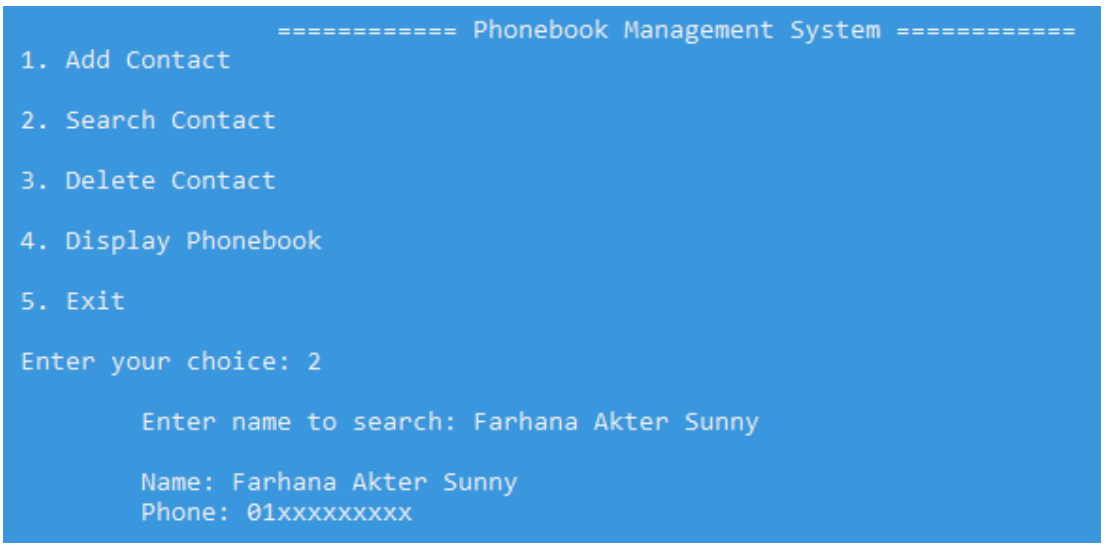

Fig 3.9: Search Contact (contact found)

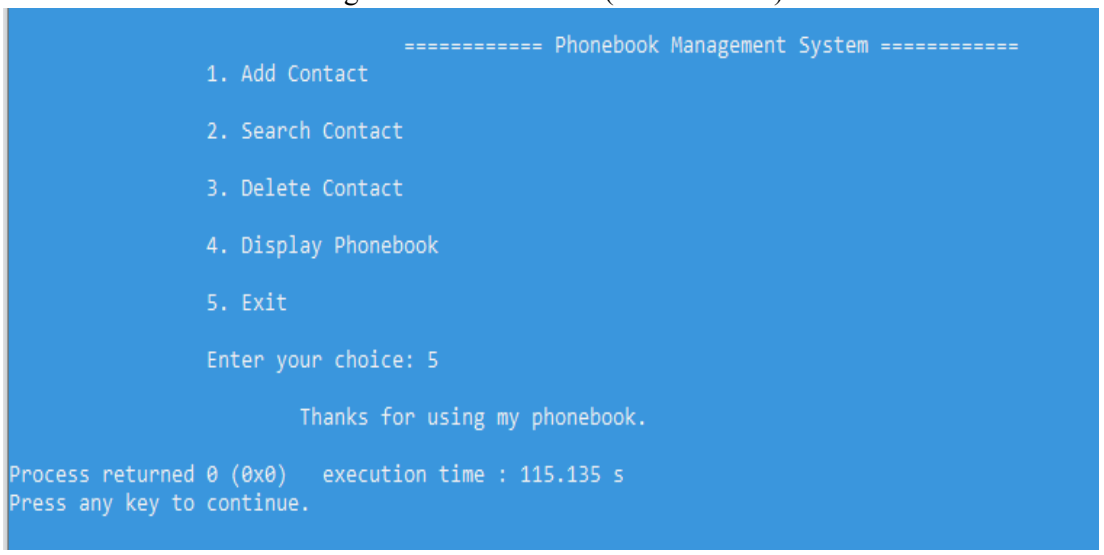

Fig 3.10: Closing the program.

3.2 Results and Discussions

3.2.1 Results

The implementation of the Phonebook Management System has led to improved efficiency in contact management . So anyone can quickly add any number, remove any number and he can easily see the full contact list at a glance.

3.2.2 Analysis and Outcome

This project is a simple Phonebook Management System implemented in C programming language. It allows users to perform various operations on a phonebook, such as adding contacts, searching for contacts, deleting contacts, and displaying the phonebook.

The program uses a linked list data structure to store the contacts. Each contact is represented by a struct Contact which contains fields for name, phone number, and a pointer to the next contact in the list. The main phonebook is represented by a pointer to the first contact in the list.

The program provides a menu-driven interface for users to choose different operations.

Here's a breakdown of each operation:

1. Add Contact: This function prompts the user to enter a name and phone number for a new contact. It dynamically allocates memory for a new contact, assigns the entered values, and adds it to the beginning of the phonebook linked list.
2. Search Contact: Users can search for a contact by entering a name. The program traverses the phonebook linked list and compares the entered name with each contact's name. If a match is found, it displays the contact's name and phone number.
3. Delete Contact: Users can delete a contact by entering the name of the contact to be deleted. The program searches for the contact in the phonebook linked list, updates the pointers accordingly, and frees the memory occupied by the deleted contact.
4. Display Phonebook: This function displays all the contacts in the phonebook. It traverses the linked list and prints the name and phone number of each contact.

The program continues to display the menu until the user chooses to exit (option 5). It also prints the current date and time at the start of the program.

At the end of the program, there is a freePhonebook() function that frees the memory occupied by all the contacts in the phonebook before exiting.

Chapter 4

Conclusion

4.1 Introduction

A phonebook management system project in C is a program that helps users store and manage contact information. It allows users to add new contacts, delete existing ones, search for specific contacts, and update contact details. The program typically uses data structures like arrays or linked lists to store the contacts, and functions are implemented to perform various operations on the contacts. Users can interact with the program through a command-line interface, where they can input commands to perform different tasks related to managing their phonebook. Overall, the project aims to provide a simple and efficient way to organize and handle contact information.

4.1 Practical Implications

Implementing a phonebook management system project can have several practical implications. Here are some of them:

1. **Efficient organization of contact information:** A phonebook management system allows users to store and manage their contacts in a structured manner. It provides a systematic way to store names, phone numbers, addresses, and other relevant details. This ensures that contact information is organized and easily accessible whenever needed.
2. **Quick and easy retrieval of contacts:** With a phonebook management system, users can search for specific contacts based on various criteria such as name, phone number, or address. This enables

quick retrieval of contacts, saving time and effort compared to manually searching through a physical phonebook.

3. **Addition, modification, and deletion of contacts:** The system allows users to add new contacts, modify existing ones, or delete unwanted contacts. This provides flexibility and ensures that the phonebook stays up to date with the latest information.
4. **Sorting and filtering capabilities:** A well-designed phonebook management system may offer sorting and filtering options to arrange contacts alphabetically, by phone number, or any other desired parameter. This helps users find contacts more easily and can be particularly useful when dealing with a large number of entries.
5. **Integration with other applications:** In some cases, a phonebook management system can be integrated with other applications or services. For example, it could provide an API that allows third-party applications to access and utilize the contact information. This integration enhances the functionality and usefulness of the system.
6. **Backup and restore functionality:** To prevent data loss, a phonebook management system may include backup and restore features. Regular backups ensure that contact information is safely stored and can be recovered in case of system failures or accidental deletions.
7. **Accessibility across multiple devices:** Depending on the implementation, a phonebook management system could support synchronization across multiple devices. This allows users to access their contact information from various devices such as smartphones, tablets, or computers, ensuring data consistency and convenience.
8. **Security and privacy considerations:** Since contact information can be sensitive, it's crucial to implement appropriate security measures in the phonebook management system. This may involve implementing user authentication, encryption, and access control to protect the data from unauthorized access or tampering.

Overall, the phonebook management system provides practical benefits by simplifying contact organization, improving retrieval speed, and offering various features to enhance usability and data management.

4.2 Scope of Future Work

This project provides a basic implementation of a phonebook management system. It can be expanded further by adding additional functionalities like editing contact details, sorting the phonebook, or saving the contacts to a file.

References

[1] Class Notes

[2] Google

[3] Quora

[4] Youtube