**PROJECT REPORT--------------------------**

**CONTACT LIST MANAGEMENT SYSTEM**

**-----------------------------------------------**

**Submitted by : [Riya Kumari]**

**UID : [24BCA10178]**

**Course : [BCA]**

**Semester : [2ND]**

**Institute : [UIC]**

**Date : [20TH APRIL 2025]**

**Submitted to: Mr.Parwan Singh**

**Signature-----------------------**

**Important points:**

** Acknowledgment**

** Abstract**

** Problem Definition**

** Objective**

** Source code**

** System Requirements**

** Flowchart**

** System Design**

** Functional Description**

** Output Overview**

** Challenges Faced**

** Future Scope**

** Conclusion**

** References**

**Acknowledgment**

I would like to express my sincere gratitude to all those who supported me throughout the development of the Contact List Management System. This project would not have been possible without the invaluable guidance, encouragement, and support of my faculty members who consistently provided insightful suggestions and valuable feedback. I am also grateful to my peers for their constant motivation and for offering help whenever I encountered challenges. Special thanks go to my family for their moral support and unwavering belief in my capabilities. Their encouragement served as a source of strength and motivation during every phase of this project. This project has been a significant learning experience and an important milestone in my academic journey, and I am thankful to all who contributed to making it a success.

**Abstract**

The Contact List Management System is a command-line based application developed in the C programming language, aimed at providing users with a structured and effective way to manage personal or professional contact information. The system enables users to perform essential contact management operations such as adding, viewing, searching, and deleting contact records. These operations are performed using simple command-line interfaces, and data persistence is achieved through the use of file handling techniques. The entire system is developed using procedural programming principles, employing structures to organize contact data and standard file input/output operations to read and write contact records. The simplicity and effectiveness of the system make it suitable for users seeking a basic yet reliable contact management solution. Additionally, the project serves as an educational tool that demonstrates the practical implementation of core programming concepts such as structures, file handling, and user interaction through menu-driven programming.

**Problem Definition**

Managing contacts manually, whether through physical notebooks or basic digital files, often leads to disorganization, inefficiency, and errors. Searching for a specific contact, updating outdated information, or deleting obsolete entries becomes a tedious and error-prone task in such environments. Additionally, traditional methods offer little to no support for data backup and recovery, making them vulnerable to data loss. As the volume of contact information grows, the challenges of managing this data also increase. The need for an efficient, user-friendly, and scalable solution becomes evident. This project addresses this problem by developing a contact management system that allows users to perform contact-related operations through a structured interface. The system leverages programming techniques and file management to overcome the limitations of manual record-keeping and provide a more reliable and efficient way to store and manage contact information.

**Objective**

The primary objective of the Contact List Management System is to develop an efficient software solution that simplifies the management of contact information. The system is designed to provide a menu-driven interface that allows users to add new contacts, view all existing contacts, search for a specific contact by name, and delete contacts that are no longer needed. By using file handling, the system ensures that contact data is stored persistently, even after the program is closed. The objectives also include designing a program that is easy to understand, operate, and maintain. The system should be capable of handling multiple records, ensuring data integrity and accuracy. Furthermore, the project aims to strengthen the developer’s understanding of structured programming, user input handling, and the application of real-world file management operations using the C programming language.

**SOURCE CODE:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Define struct**

**struct Contact {**

**char name[50];**

**char phone[15];**

**char email[50];**

**};**

**// Function declarations**

**void addContact();**

**void viewContacts();**

**void searchContact();**

**void deleteContact();**

**int main() {**

**int choice;**

**while (1) {**

**printf("\nContact Management System\n");**

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

**printf("2. View Contacts\n");**

**printf("3. Search Contact\n");**

**printf("4. Delete Contact\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**getchar(); // Consume leftover newline**

**switch (choice) {**

**case 1: addContact(); break;**

**case 2: viewContacts(); break;**

**case 3: searchContact(); break;**

**case 4: deleteContact(); break;**

**case 5: exit(0);**

**default: printf("Invalid choice. Try again.\n");**

**}**

**}**

**return 0;**

**}**

**void removeNewline(char\* str) {**

**str[strcspn(str, "\n")] = 0;**

**}**

**void addContact() {**

**FILE \*fp = fopen("contacts.txt", "ab");**

**struct Contact c;**

**printf("Enter name: ");**

**fgets(c.name, sizeof(c.name), stdin);**

**removeNewline(c.name);**

**printf("Enter phone: ");**

**fgets(c.phone, sizeof(c.phone), stdin);**

**removeNewline(c.phone);**

**printf("Enter email: ");**

**fgets(c.email, sizeof(c.email), stdin);**

**removeNewline(c.email);**

**fwrite(&c, sizeof(struct Contact), 1, fp);**

**fclose(fp);**

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

**}**

**void viewContacts() {**

**FILE \*fp = fopen("contacts.txt", "rb");**

**struct Contact c;**

**if (fp == NULL) {**

**printf("No contacts found.\n");**

**return;**

**}**

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

**while (fread(&c, sizeof(struct Contact), 1, fp)) {**

**printf("Name: %s\nPhone: %s\nEmail: %s\n\n", c.name, c.phone, c.email);**

**}**

**fclose(fp);**

**}**

**void searchContact() {**

**FILE \*fp = fopen("contacts.txt", "rb");**

**struct Contact c;**

**char search[50];**

**int found = 0;**

**if (fp == NULL) {**

**printf("No contacts found.\n");**

**return;**

**}**

**printf("Enter name to search: ");**

**fgets(search, sizeof(search), stdin);**

**removeNewline(search);**

**while (fread(&c, sizeof(struct Contact), 1, fp)) {**

**if (strcmp(c.name, search) == 0) {**

**printf("Contact found:\nName: %s\nPhone: %s\nEmail: %s\n", c.name, c.phone, c.email);**

**found = 1;**

**break;**

**}**

**}**

**if (!found) {**

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

**}**

**fclose(fp);**

**}**

**void deleteContact() {**

**FILE \*fp = fopen("contacts.txt", "rb");**

**FILE \*temp = fopen("temp.txt", "wb");**

**struct Contact c;**

**char name[50];**

**int found = 0;**

**if (fp == NULL) {**

**printf("No contacts found.\n");**

**return;**

**}**

**printf("Enter name to delete: ");**

**fgets(name, sizeof(name), stdin);**

**removeNewline(name);**

**while (fread(&c, sizeof(struct Contact), 1, fp)) {**

**if (strcmp(c.name, name) != 0) {**

**fwrite(&c, sizeof(struct Contact), 1, temp);**

**} else {**

**found = 1;**

**}**

**}**

**fclose(fp);**

**fclose(temp);**

**remove("contacts.txt");**

**rename("temp.txt", "contacts.txt");**

**if (found)**

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

**else**

**printf("Contact not found.\n");**

**}**

**OUTPUT:**

**1. Add Contact:**

**Enter name: Aarav Sharma**

**Enter phone: 9876543210**

**Enter email: aarav.sharma@email.com**

**Contact added successfully.**

**2. View Contacts:**

**--- Contact List ---**

**Name: Aarav Sharma**

**Phone: 9876543210**

**Email: aarav.sharma@email.com**

**3. Search Contact:**

**Enter name to search: Aarav Sharma**

**Contact found:**

**Name: Aarav Sharma**

**Phone: 9876543210**

**Email: aarav.sharma@email.com**

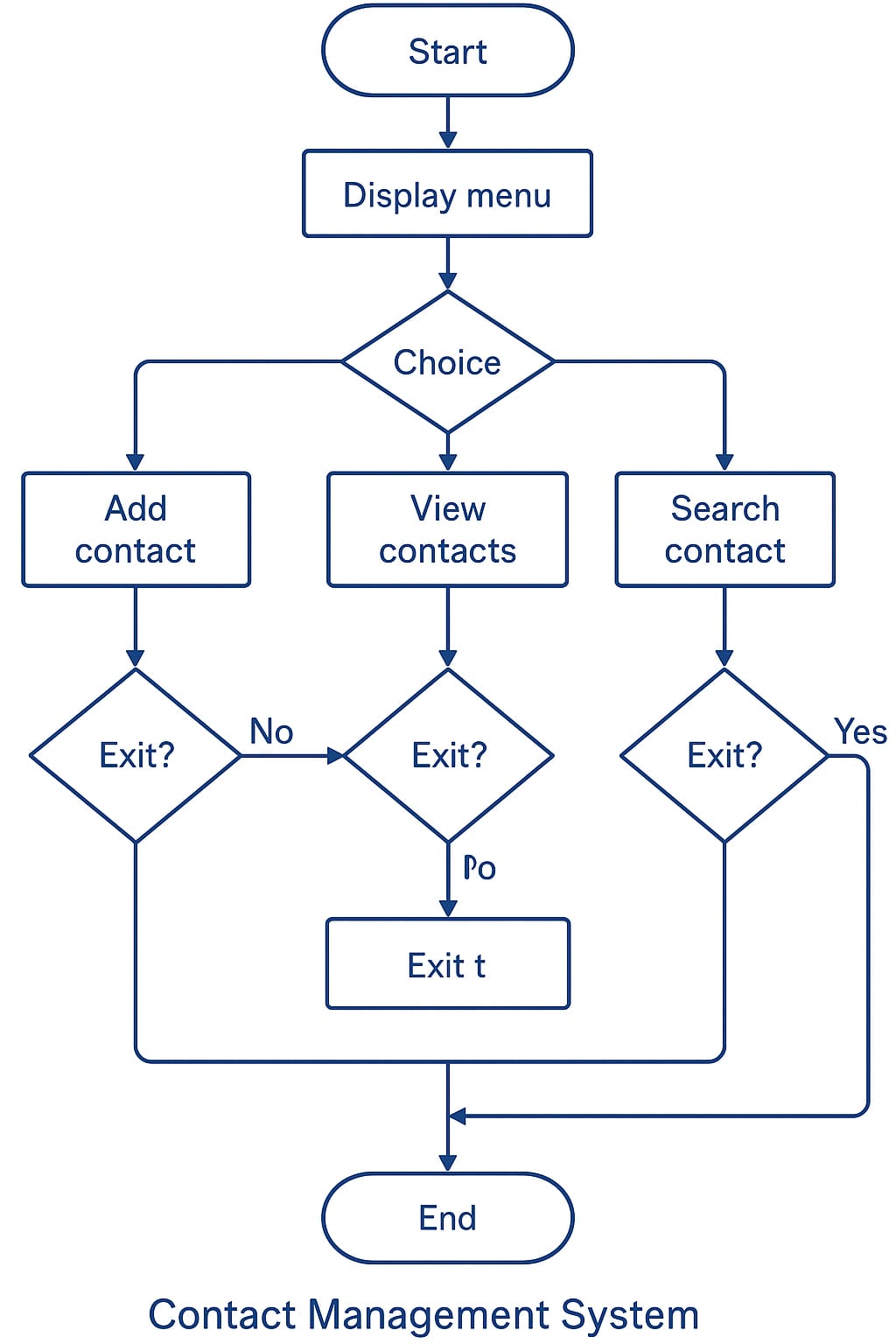
**4. Delete Contact:**

**Enter name to delete: Aarav Sharma**

**Contact deleted successfully.**

**5. Exit:**

**The program exits.**



**Flowchart**

**Below is the flowchart that represents the working logic of the Contact List Management System:**

**Start**

**|**

**v**

**Display Main Menu**

**|**

**v**

**Get User Choice**

**|**

**+--> [1] Add Contact -------------> Input Contact Details --> Save to File --> Return to Menu**

**|**

**+--> [2] View Contacts -----------> Read All from File --> Display --> Return to Menu**

**|**

**+--> [3] Search Contact ---------> Input Name --> Search in File --> Display Result --> Return to Menu**

**|**

**+--> [4] Delete Contact ---------> Input Name --> Copy All Except Match to Temp --> Replace File --> Return to Menu**

**|**

**+--> [5] Exit --------------------> Terminate Program**

**System Design**

The system is designed using a procedural approach in C programming, where user interaction is performed through a menu-driven interface. The design utilizes file handling to ensure that the contact information persists between sessions. The program organizes contact data in a struct called Contact, which contains three fields: name, phone, and email. This data is stored in a file named "contacts.txt". The main functionalities of the system include adding new contacts, viewing all contacts, searching for a specific contact by name, and deleting a contact from the list. The system employs simple command-line input and output to interact with the user, making it intuitive for basic operations.

**Functional Description**

The Contact List Management System allows users to manage their contact information efficiently. When the program starts, it displays a menu with several options, including adding, viewing, searching, and deleting contacts. The system uses file handling to read and write contact data to a file. Here is how each function works:

* **Add Contact**: Prompts the user to input the contact's name, phone number, and email address, then appends this data to the "contacts.txt" file.
* **View Contacts**: Reads all the contact records from the "contacts.txt" file and displays them on the screen.
* **Search Contact**: Accepts a name as input and searches through the file for a matching contact. If found, it displays the contact’s information; otherwise, it informs the user that the contact was not found.
* **Delete Contact**: Allows the user to delete a contact by entering the name of the contact they want to remove. It creates a temporary file with all the contacts except the one to be deleted and then replaces the original file with the updated one.

**Output Overview**

The output of the system consists mainly of displaying the contact details to the user. When a user views all contacts, the system reads and displays the contact information from the file. Each contact’s name, phone number, and email are displayed in a structured format. For search results, the system will either show the matching contact or inform the user that the contact was not found. When deleting a contact, the system confirms whether the operation was successful or if the contact could not be found. Finally, any error or invalid input is displayed with a corresponding message for the user.

**Challenges Faced**

Several challenges were encountered during the development of this system:

1. **File Handling Issues**: Initially, there were issues related to reading and writing to files, as the system had to handle various operations like appending, reading, and updating records without corrupting the file data.
2. **Memory Management**: Proper memory allocation and deallocation were essential, especially when dealing with dynamic user input and ensuring there were no memory leaks.
3. **Input Validation**: Ensuring that the user input was validated properly, particularly for contact details (e.g., ensuring that a phone number contains only digits), required additional logic and error-checking mechanisms.
4. **Deleting Records**: The process of deleting records required extra attention to ensure that the file content remained intact, and the contact data was only removed when necessary.

**Future Scope**

The system, as it stands, is functional for basic contact management tasks, but there is significant room for improvement. Some potential future enhancements include:

1. **Graphical User Interface (GUI)**: A GUI could be developed using libraries such as GTK or Qt to provide a more intuitive experience compared to the command line.
2. **Search and Sorting Features**: Adding more advanced search options (e.g., search by phone number or email) and allowing users to sort contacts alphabetically would enhance the system’s usability.
3. **Backup and Restore**: Implementing a backup and restore functionality would allow users to safeguard their contact information.
4. **Integration with External Platforms**: Future versions of the system could integrate with popular platforms such as Google Contacts or Microsoft Outlook for seamless contact synchronization.
5. **Security**: Incorporating encryption techniques to protect sensitive contact information from unauthorized access would add another layer of security.

**System Requirements**

The Contact List Management System has minimal system requirements, making it compatible with most basic computing environments. From a hardware perspective, a computer with an Intel Pentium processor or higher and at least 2 GB of RAM is sufficient to run the program smoothly. In terms of storage, a minimum of 10 MB of free disk space is required to store the executable files and data files. On the software side, the system requires an operating system such as Windows, Linux, or macOS that supports C compilers. GCC (GNU Compiler Collection) is recommended for compiling the program, though other compilers like Turbo C or TCC can also be used. A basic code editor such as Notepad++, Sublime Text, or any integrated development environment (IDE) like Code::Blocks or Dev-C++ is suitable for editing and compiling the code. No additional libraries or frameworks are required, which ensures ease of portability and deployment.

**Conclusion**

The Contact List Management System is a simple yet effective solution for managing contact information. The system successfully demonstrates the application of fundamental programming concepts such as file handling, structures, and procedural programming. It provides a solid foundation for future enhancements, including features like GUI support and advanced search functionalities. Despite its simplicity, the system addresses key problems related to manual contact management, offering a more organized and efficient approach to managing personal or professional contacts. This project serves as an excellent learning tool for understanding the practical use of file handling and user interaction in C programming.

**REFRENCES:**

** King, K. N. (2008). *C Programming: A Modern Approach*.**

** GeeksforGeeks. (n.d.). *File Handling in C*. Retrieved from https://www.geeksforgeeks.org/file-handling-c-using-fopen-fclose-fread-fwrite/**

** TutorialsPoint. (n.d.). *C Programming Language*. Retrieved from https://www.tutorialspoint.com/cprogramming/index.htm**

**Flowchart**