Report On

PHONEBOOK

Submitted in partial fulfilment of the requirements of the Course project in Semester III of Second Year Computer Engineering

By

Sambit Mazumder (Roll No. 34)

Supervisor
Prof. Aarti Puthran

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

(2023-24)

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled “Library Management System” is a Bonafide work of Sambit Mazumder (Roll No. 34) submitted to the University of Mumbai in partial fulfilment of the requirement for the Course project in semester III of Second Year Computer Engineering.

Supervisor

Prof. Aarti Puthran

Internal Examiner

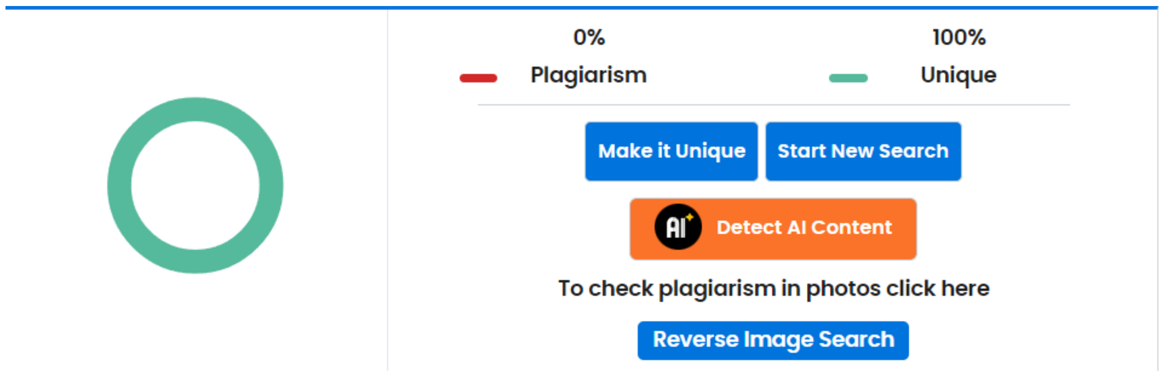
External Examiner

Dr Megha Trivedi
Head of Department

Dr. H.V. Vankudre
Principal

Index

1.Plagiarism:



2. Abstract:

A phone book, also known as an address book, is a repository for storing and organizing contact information, primarily comprising names, phone numbers, addresses, and email addresses. Its fundamental purpose is to facilitate efficient communication and provide a centralized location for essential contacts. Users can easily search, add, update, categorize, and manage contacts within the phone book, enabling streamlined access to vital information. In today's digital age, phone books have transitioned from traditional printed forms to electronic formats, integrating advanced features and enhanced accessibility for seamless communication and data management across various devices.

3. Problem Statement:

In today's digital era, efficiently managing and accessing a vast array of contacts across different communication channels is a growing challenge. Existing phone book applications often lack intuitive user interfaces, robust search capabilities, and seamless synchronization across devices, limiting user productivity and experience. Additionally, ensuring data privacy and security while providing effortless data management features is critical. This problem statement seeks to address these limitations by developing a comprehensive and user-friendly phone book application that optimizes contact organization, enhances search functionality, and prioritizes data privacy, thus providing a superior user experience and fostering effective communication in a dynamic, technology-driven landscape.

4. Module Description:

Phone Book Application Description:

1. Overview:

The phone book application is designed to provide a user-friendly interface for efficiently managing contacts. Utilizing the C programming language and a linked list data structure, it enables users to perform essential operations such as adding new contacts, deleting existing ones, modifying contact details, and searching for contacts based on names or numbers.

2. Data Storage using Linked List:

The heart of the application lies in the implementation of a linked list data structure to store contact information. Each node in the linked list represents a contact, containing fields for the contact's name, phone number, address, and any additional information. The linked list structure allows dynamic allocation and efficient management of contacts, accommodating the varying number of entries.

3. Functionalities:

- **Adding a Contact:**

Users can easily add a new contact by providing details such as the name, phone number, address, etc. The application then creates a new node in the linked list to store this information, effectively adding the contact to the phone book.

- **Deleting a Contact:**

Deleting a contact involves searching for the contact based on name or number and removing the corresponding node from the linked list. This operation ensures that unwanted or outdated contacts can be efficiently removed.

- **Modifying a Contact:**

Users have the ability to update and modify contact information, be it a change in phone number, address, or any other detail. The application facilitates locating the contact, modifying the necessary fields, and updating the linked list node accordingly.

- **Searching for a Contact:**

The application enables users to search for contacts based on either the contact's name or phone number. This search functionality is crucial for quick and convenient access to contact information within the phone book.

4. User Interaction:

The application provides a simple and intuitive user interface. Users can interact with the application through a command-line interface, allowing them to input their desired operations (add, delete, modify, search) and their respective contact details.

5. Flexibility and Scalability:

The use of a linked list data structure grants the application flexibility in managing contacts. It can efficiently handle a growing number of contacts without predefined size limitations, making it scalable and adaptable to various usage scenarios.

6. Future Enhancements:

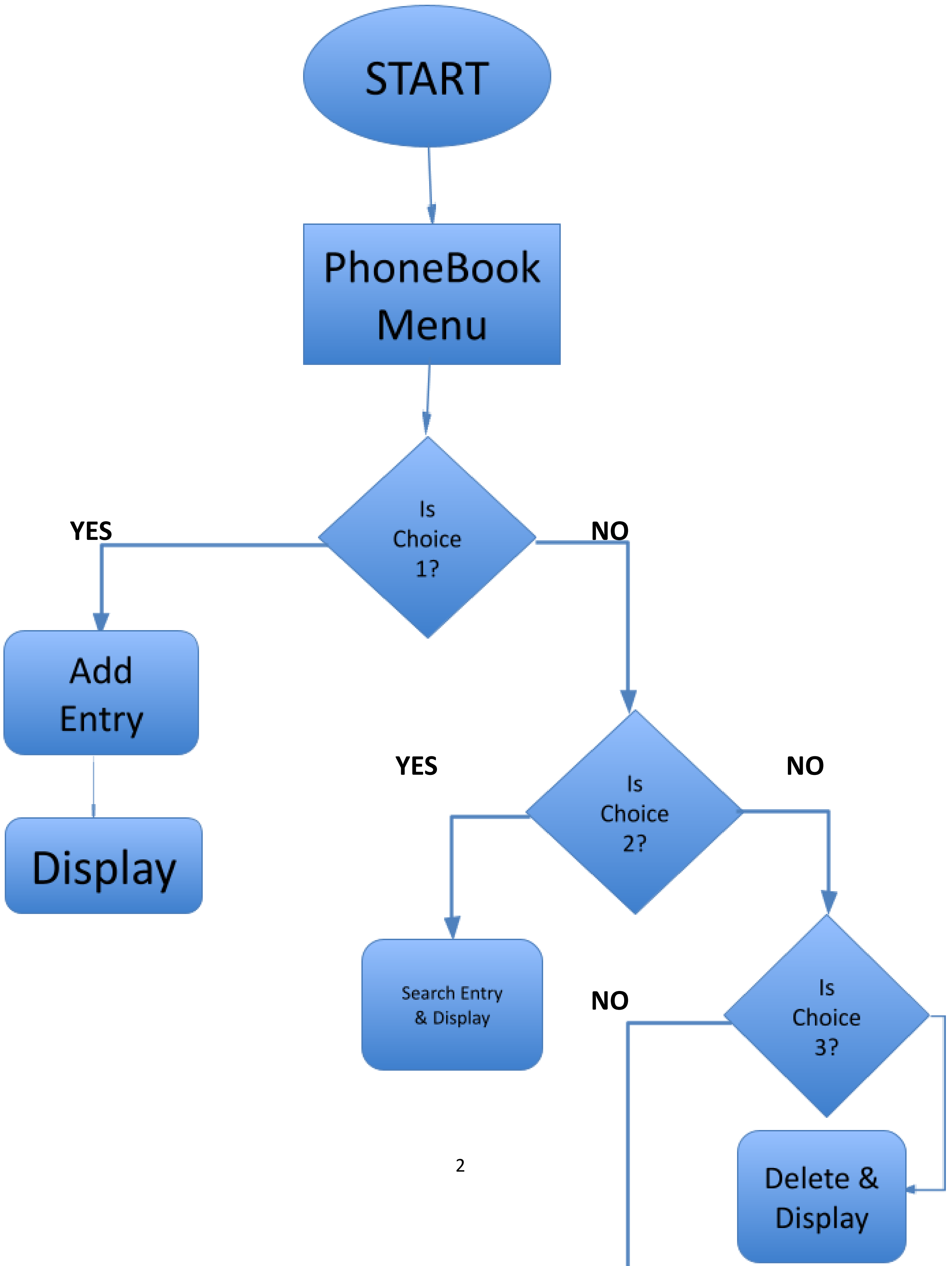
Potential future enhancements could include integrating a graphical user interface (GUI) for a more visually appealing and user-friendly experience.

Incorporating features like contact categorization, export/import functionality, and data encryption for improved user privacy and data security.

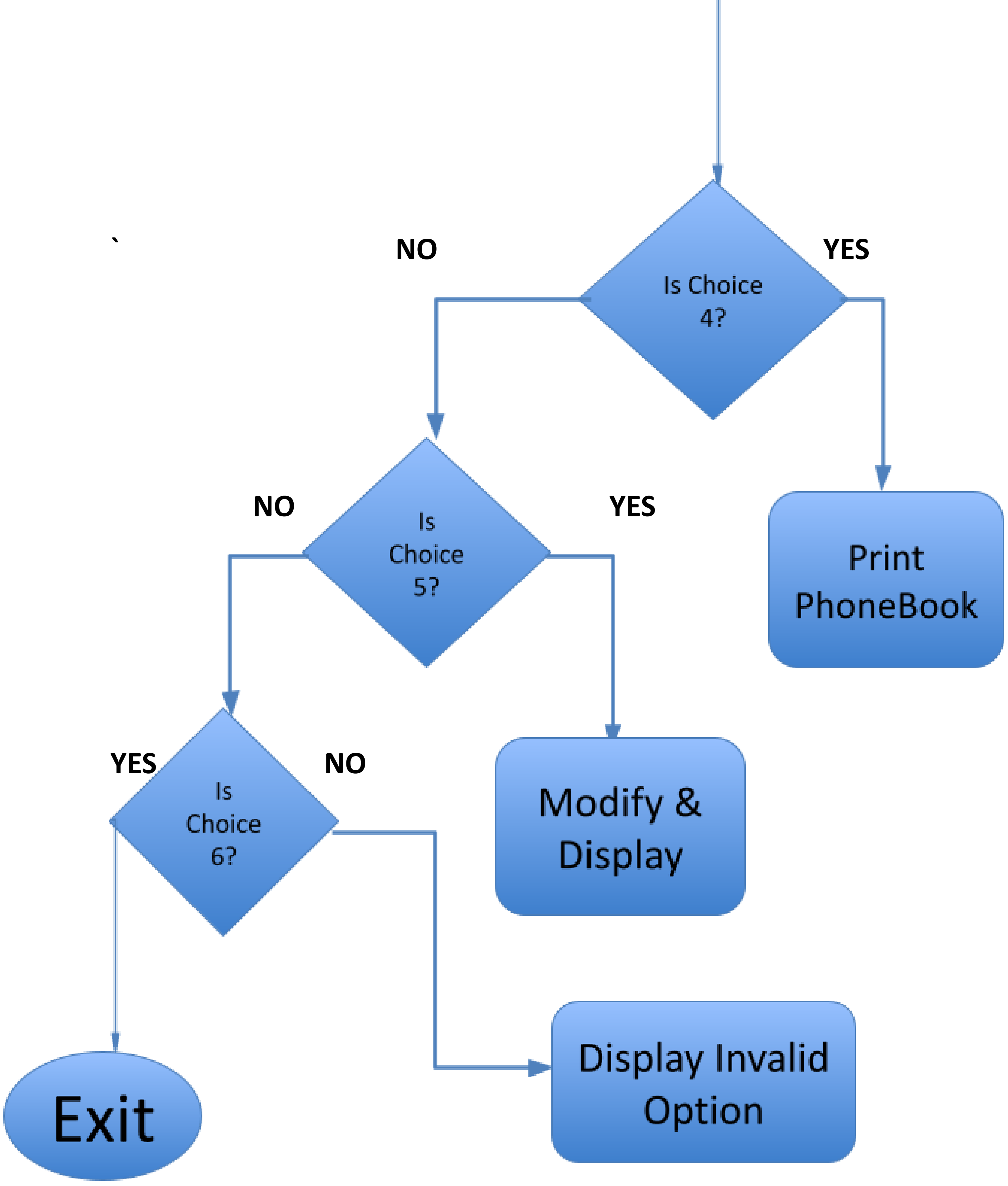
7. Conclusion:

The phone book application successfully leverages the power of C and linked list data structure to offer a robust solution for contact management. It empowers users to seamlessly organize and access their contacts, making communication more efficient and effective.

5. BLOCK DIAGRAM:



`


NO
Is Choice 4?
YES
Print PhoneBook
NO
Is Choice 5?
YES
Modify & Display
YES
Is Choice 6?
NO
Exit
Display Invalid Option

6. Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // For strcpy, strcasecmp, and strcmp

// Define a structure for a phonebook entry
struct PhonebookEntry
{
   char name[50];
   char phoneNumber[15];
   struct PhonebookEntry *next;
};

struct PhonebookEntry *head = NULL;

// Function to add a new entry to the phonebook
void addEntry(char name[], char phoneNumber[])
{
         struct PhonebookEntry *newEntry = (struct PhonebookEntry
*)malloc(sizeof(struct PhonebookEntry));

   if (newEntry == NULL)
   {
      printf("Memory allocation failed!\n");
      return;
   }

   // Use strcpy to copy the name and phoneNumber
   strcpy(newEntry->name, name);
   strcpy(newEntry->phoneNumber, phoneNumber);

   newEntry->next = head;
   head = newEntry;

   printf("Entry added successfully!\n");
}

// Function to search for an entry by name or phone number
(case-insensitive) and print the result
void searchEntry(char query[])
{
    struct PhonebookEntry *current = head;
    int found = 0;

    while (current != NULL)
    {
        //DO THE SEARCHING HERE
                          if    (strcasecmp(current->name,    query)    ==    0    ||
strcasecmp(current->phoneNumber, query) == 0)
        {
            // Case-insensitive comparison for both name and phone number
            printf("Name: %s\n", current->name);
            printf("Phone Number: %s\n", current->phoneNumber);
            found = 1;
        }
        //Increment the pointer
        current = current->next;
    }

    if (!found)
    {
        printf("Entry not found!\n");
    }
}

// Function to print all entries in the phonebook
void printPhonebook()
{
    struct PhonebookEntry *current = head;

    if (current == NULL)
    {
        printf("Phonebook is empty!\n");
        return;
    }

    printf("Phonebook entries:\n");
    while (current != NULL)
    {
        printf("Name: %s\n", current->name);
        printf("Phone Number: %s\n", current->phoneNumber);
        printf("----------------------------\n");
        current = current->next;
```

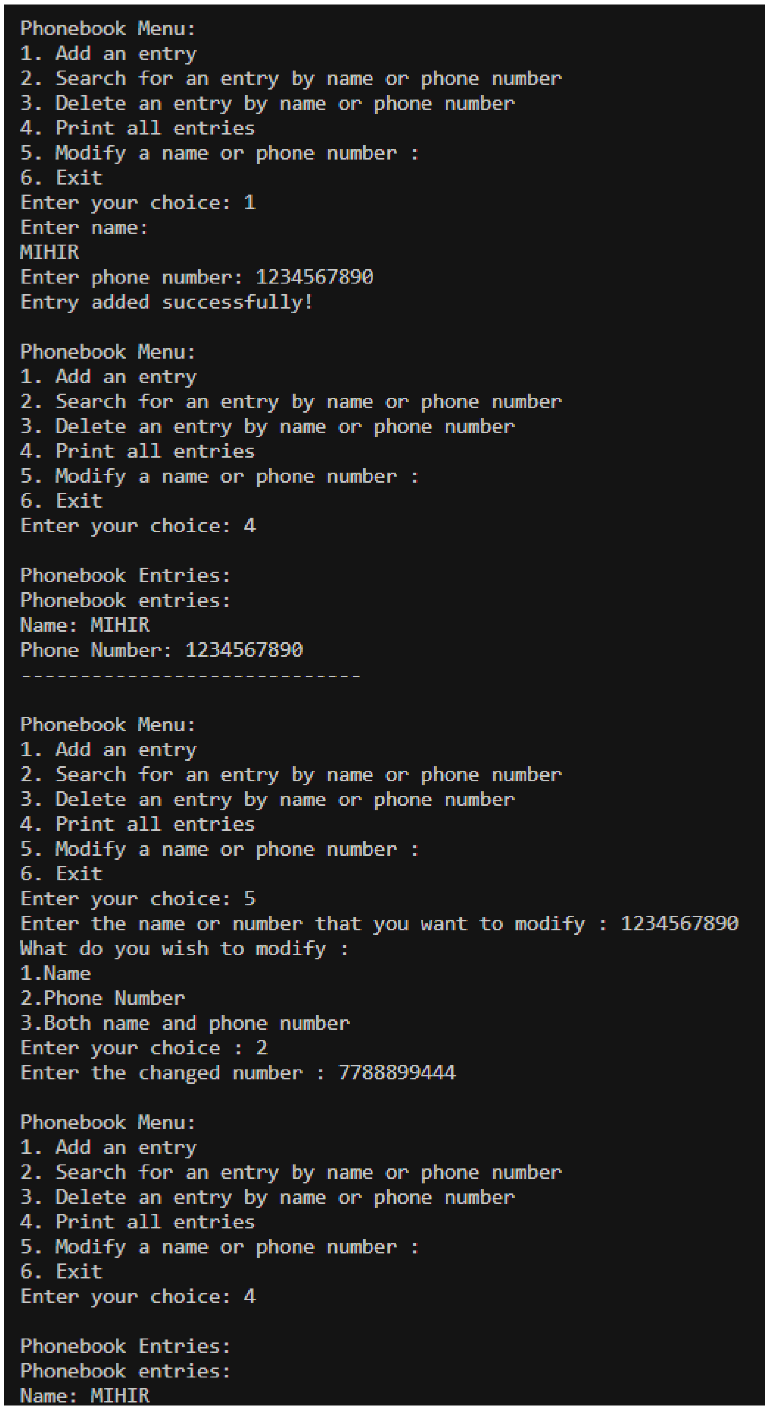
```
   }
}

// Function to delete an entry by name or phone number (case-insensitive)

void deleteEntry(
   char query[])
{
   struct PhonebookEntry *current = head;
   struct PhonebookEntry *prev = NULL;
   int found = 0;

   while (current != NULL)
   {
                        if    (strcasecmp(current->name,    query)    ==    0    ||
strcasecmp(current->phoneNumber, query) == 0)
      {
         // Case-insensitive comparison for both name and phone number
         printf("Deleted Entry:\n");
         printf("Name: %s\n", current->name);
         printf("Phone Number: %s\n", current->phoneNumber);

         if (prev == NULL)
         {
            head = current->next;
         }
         else
         {
            prev->next = current->next;
         }
         free(current);
         found = 1;
         break;
      }
      prev = current;
      current = current->next;
   }

   if (!found)
   {
      printf("Entry not found!\n");
   }
```

```
    else
    {
        printf("Entry deleted successfully!\n");
    }
}

//Modify an entry by name or phone number

void modify_contact(char query[50])
{
    struct PhonebookEntry * current = head;
    int found = 0;

    while (current != NULL)
    {
        //DO THE SEARCHING HERE
                    if (strcasecmp(current->name, query) == 0 ||
strcasecmp(current->phoneNumber, query) == 0)
        {
            int a = 0;
            char cn[50];
                    printf("What do you wish to modify :\n1.Name\n2.Phone
Number\n3.Both name and phone number\nEnter your choice : ");
            scanf("%d", &a);

            switch(a)
            {
                case 1 :
                {
                    printf("Enter the changed name : ");
                    scanf("%s", cn);
                    strcpy(current->name,cn);
                    break;
                }

                case 2 :
                {
                    printf("Enter the changed number : ");
                    scanf("%s", cn);
                    strcpy(current->phoneNumber,cn);
                    break;
                }
                case 3 :
                {
                    printf("Enter the changed name : ");
                    scanf("%s", cn);
                    strcpy(current->name,cn);
                    printf("Enter the changed number : ");
                    scanf("%s", cn);
                    strcpy(current->phoneNumber,cn);
                    break;
                }
            }
            found = 1;
        }
        //Increment the pointer
        current = current->next;
    }

    if (!found)
    {
        printf("Entry not found!\n");
    }
}

int main()
{

    int choice;
    char name[50], phoneNumber[15], query[50];

    while (1)
    {
        printf("\nPhonebook Menu:\n");
        printf("1. Add an entry\n");
        printf("2. Search for an entry by name or phone number\n");
        printf("3. Delete an entry by name or phone number\n");
        printf("4. Print all entries\n");
        printf("5. Modify a name or phone number : \n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

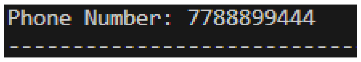
```
switch (choice)
{
case 1:
   printf("Enter name: ");
   scanf("%s", name);
   printf("Enter phone number: ");
   scanf("%s", phoneNumber);
   addEntry(name, phoneNumber);
   break;
case 2:
   printf("Enter name or phone number to search: ");
   scanf("%s", query);
   searchEntry(query);
   break;
case 3:
   printf("Enter name or phone number to delete: ");
   scanf("%s", query);
   deleteEntry(query);
   break;
case 4:
   // Print all entries
   printf("\nPhonebook Entries:\n");
   printPhonebook();
   break;

case 5:
   //Modify a name or phone number
   printf("Enter the name or number that you want to modify : ");
   scanf("%s", query);
   modify_contact(query);
   break;
case 6:
   // Free memory and exit
   while (head != NULL)
   {
      struct PhonebookEntry *temp = head;
      head = head->next;
      free(temp);
   }
   return 0;
default:
   printf("Invalid choice, please try again.\n");
```

```
        }
    }

    return 0;
}
```

7. Result:



```
Phonebook Menu:
1. Add an entry
2. Search for an entry by name or phone number
3. Delete an entry by name or phone number
4. Print all entries
5. Modify a name or phone number :
6. Exit
Enter your choice: 1
Enter name:
MIHIR
Enter phone number: 1234567890
Entry added successfully!

Phonebook Menu:
1. Add an entry
2. Search for an entry by name or phone number
3. Delete an entry by name or phone number
4. Print all entries
5. Modify a name or phone number :
6. Exit
Enter your choice: 4

Phonebook Entries:
Phonebook entries:
Name: MIHIR
Phone Number: 1234567890
-----------------------------

Phonebook Menu:
1. Add an entry
2. Search for an entry by name or phone number
3. Delete an entry by name or phone number
4. Print all entries
5. Modify a name or phone number :
6. Exit
Enter your choice: 5
Enter the name or number that you want to modify : 1234567890
What do you wish to modify :
1.Name
2.Phone Number
3.Both name and phone number
Enter your choice : 2
Enter the changed number : 7788899444

Phonebook Menu:
1. Add an entry
2. Search for an entry by name or phone number
3. Delete an entry by name or phone number
4. Print all entries
5. Modify a name or phone number :
6. Exit
Enter your choice: 4

Phonebook Entries:
Phonebook entries:
Name: MIHIR
```



```
Phone Number: 7788899444
-----------------------------
```

8. Conclusion:

In conclusion, the phone book application, built using C and a linked list, efficiently manages contacts by allowing users to add, delete, modify, and search for contacts. The use of a linked list ensures flexible data management, while the user-friendly interface enhances accessibility. Future improvements can enhance categorization, import/export, data encryption, and introduce a graphical user interface for an even better user experience.

9. Reference:

OpenAI, StackOverflow, GitHub, etc