## MINI PROJECT

**On**

(Phone Dictionary Using Linked Lists)

## In

**Data Structures and Algorithms Lab**

# BACHELOR OF TECHNOLOGY

**IN**

# Artificial Intelligence and Machine Learning

SUBMITTED BY

# Name Bontha Mallikarjun Reddy

# (PRN - 23070126026)



## Under the Supervision of

**Ms. Suman Tanwar, Assistant Professor**

**Ms. Hema Karade, Assistant Professor**

## SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE – 412115

**(A CONSTITUENT OF SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)) 2023-24**

1. **Problem Statement:**

**Statement**: Develop a dynamic contact management system that efficiently stores, retrieves, and manages contact information.

**Explanation**:

* Traditional static data structures like arrays require fixed memory allocation, limiting data management flexibility.
* Using a linked list allows the program to handle a variable number of contacts without predetermined limits, making it a flexible solution for contact storage and retrieval.
* **Goal**: Create a menu-driven phone dictionary that enables users to add, search, and display contacts dynamically.

1. **Motivation**:  
   **Real-World Application**: Contact management is essential for organizing personal and professional information in mobile devices and online applications.

**Educational Purpose**: To gain hands-on experience with linked lists and understand their applications for dynamic memory allocation and efficient data storage.

**Challenge Addressed**: To overcome the limitations of static arrays by leveraging the advantages of linked lists for efficient insertion and deletion operations.

1. **Objective:**

**Primary Objectives**:

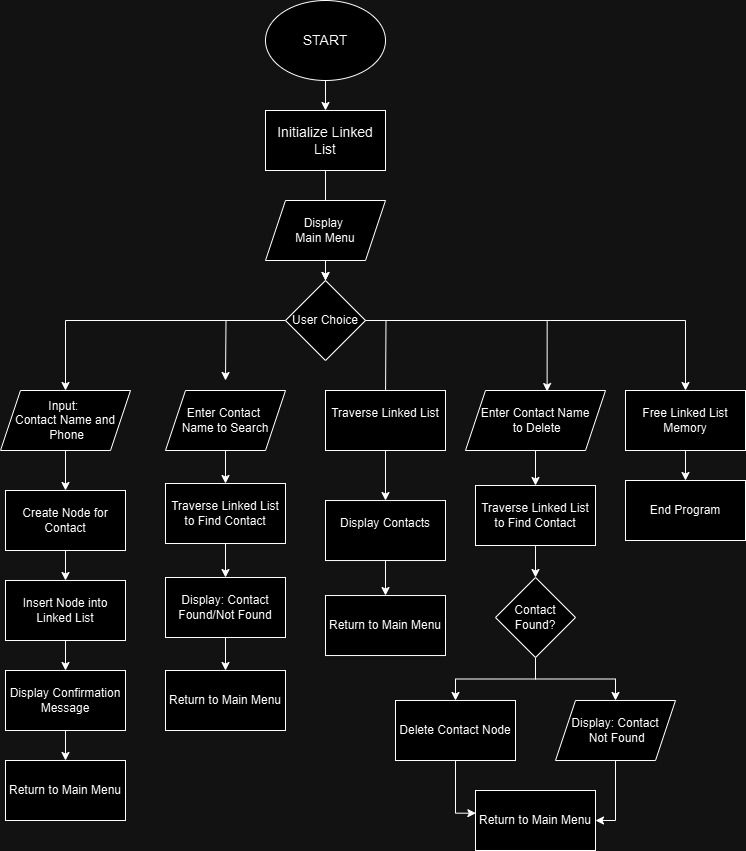
1. Develop a contact management system using linked lists in C.
2. Implement basic functionalities: adding a contact, searching for a contact, and displaying all contacts.
3. Design an intuitive, menu-driven interface for easy navigation and user interaction.

**Secondary Objectives**:

* Ensure proper memory management through allocation and deallocation.
* Lay the foundation for additional features like delete and update functions for contacts.

1. **Methodology of Implementation**

* **Flowchart**

****

* **Explanation of Flow Chart**

**1. Start Program**

* **Purpose**: This step marks the beginning of the program's execution. It initializes necessary variables and data structures, specifically an empty linked list, to store contact nodes.

**2. Initialize Linked List**

* **Details**: A linked list is created with a head pointer initialized to NULL, indicating no contacts stored at the start.

**3. Display Main Menu**

* **Explanation**: The program presents a menu to the user, listing available options:
  + **1**. Add Contact
  + **2**. Search Contact
  + **3**. Display All Contacts
  + **4**. Delete Contact
  + **5**. Exit Program
* **Purpose**: This menu allows the user to navigate the different features of the phone dictionary.

**4. The User Chooses an Option**

* **Explanation**: The program waits for the user to select an option by entering a number between 1 and 5.
* **Decision Point**: The program directs the user to the corresponding functionality based on the user's choice.

1. **Hardware/Software Used**

* **Hardware:** Standard PC or laptop.
* **Software:**
  + **IDE:** VS Code
  + **Programming Language:** C (standard C libraries for I/O and memory management).
  + **Operating System:** Windows

1. **Data Set description**

 Note: No dataset integration in this project (if future data persistence is required, contacts could be stored in a text file).

 CurrentDesign: The user enters Contacts manually during runtime, ensuring flexibility and user-defined data without a predefined dataset.

1. **Executable Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define the Contact struct

typedef struct {

    char name[50];

    char phone[15];

} Contact;

// Define the Node struct for the linked list

typedef struct Node {

    Contact contact;

    struct Node\* next;

} Node;

// Head pointer for the linked list

Node\* head = NULL;

// Function to create a new node with a contact

Node\* createNode(char name[], char phone[]) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (!newNode) {

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

        exit(1);

    }

    strcpy(newNode->contact.name, name);

    strcpy(newNode->contact.phone, phone);

    newNode->next = NULL;

    return newNode;

}

// Function to add a contact to the linked list

void addContact() {

    char name[50];

    char phone[15];

    printf("Enter contact name: ");

    scanf("%s", name);

    printf("Enter phone number: ");

    scanf("%s", phone);

    Node\* newNode = createNode(name, phone);

    newNode->next = head;

    head = newNode;

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

}

// Function to search for a contact by name

void searchContact() {

    if (head == NULL) {

        printf("No contacts to search.\n");

        return;

    }

    char name[50];

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

    scanf("%s", name);

    Node\* current = head;

    while (current != NULL) {

        if (strcmp(current->contact.name, name) == 0) {

            printf("Contact found: %s - %s\n", current->contact.name, current->contact.phone);

            return;

        }

        current = current->next;

    }

    printf("Contact not found.\n");

}

// Function to display all contacts

void listContacts() {

    if (head == NULL) {

        printf("No contacts to display.\n");

        return;

    }

    printf("\nPhone Book Contacts:\n");

    Node\* current = head;

    int index = 1;

    while (current != NULL) {

        printf("%d. %s - %s\n", index++, current->contact.name, current->contact.phone);

        current = current->next;

    }

}

// Function to delete a contact by name

void deleteContact() {

    if (head == NULL) {

        printf("No contacts to delete.\n");

        return;

    }

    char name[50];

    printf("Enter the name of the contact to delete: ");

    scanf("%s", name);

    Node\* current = head;

    Node\* previous = NULL;

    // Traverse the list to find the contact

    while (current != NULL && strcmp(current->contact.name, name) != 0) {

        previous = current;

        current = current->next;

    }

    if (current == NULL) {

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

        return;

    }

    // Remove the node from the list

    if (previous == NULL) {

        head = current->next; // The head is the contact to be deleted

    } else {

        previous->next = current->next;

    }

    free(current);

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

}

// Function to free all memory used by the linked list

void freeMemory() {

    Node\* current = head;

    while (current != NULL) {

        Node\* temp = current;

        current = current->next;

        free(temp);

    }

}

// Main function with a menu-driven interface

int main() {

    int choice;

    do {

        printf("\nPhone Dictionary Menu:\n");

        printf("1. Add Contact\n");

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

        printf("3. List All Contacts\n");

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

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                addContact();

                break;

            case 2:

                searchContact();

                break;

            case 3:

                listContacts();

                break;

            case 4:

                deleteContact();

                break;

            case 5:

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

                freeMemory();

                break;

            default:

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

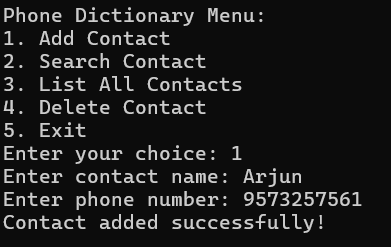
        }

    } while (choice != 5);

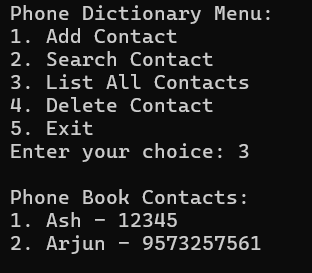
    return 0;

}

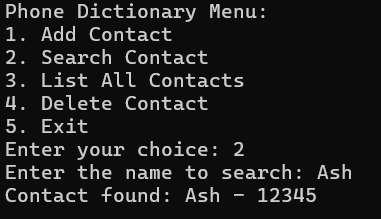
1. **Result Analysis with Output Screen Shot  
   Add Contact**

****

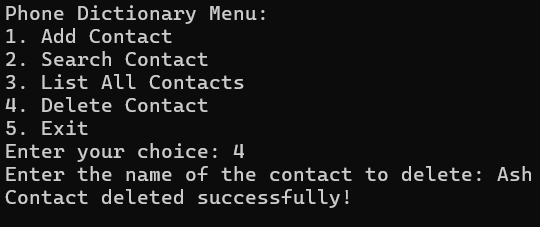
**List All Contacts**

****

**Search Contacts**

****

**Delete Contact**

****

**Analysis**:

* Successful addition, search, and display of contacts confirm the program's functionality.
* Efficient memory usage observed, as each new contact only allocates necessary memory, with no fixed limits.

1. **Learning Outcome**

This project has led to several valuable insights:

**Data Structures**:

* Practical experience with linked lists, including creating nodes, linking them dynamically, and traversing lists.

**Programming Skills**:

* Enhanced understanding of C programming, specifically pointers, memory allocation (malloc), and freeing memory (free).

**Software Development**:

* Developed a user-friendly, menu-driven interface that handles basic user input and interaction.
* Gained insight into creating a structured, modular program that follows clean code principles and efficient memory management.

1. **Real-world Applications**

## Telecommunications:

## Telecom companies can utilize similar data structures to manage and search large volumes of customer contact information. Linked lists can help efficiently handle dynamic lists of contacts that may change frequently as customers are added or removed.

## Healthcare Systems:

## Hospitals and clinics can use linked lists for patient records management, where records need frequent updating or removal. Linked lists allow for flexible insertion and deletion without reorganizing data, making it ideal for dynamic patient management systems.

## Customer Relationship Management (CRM):

## CRM systems can benefit from linked list structures to store and update customer data, such as contact details and interaction history. The dynamic nature of linked lists allows CRM applications to manage large, changing datasets without performance degradation.

## Conclusion

## The Phone Dictionary Using Linked Lists project demonstrates the effectiveness of linked lists in managing contact information dynamically. By allowing easy insertion, deletion, and search functionalities, linked lists provide an efficient structure for handling data that frequently changes in size. The menu-driven program is user-friendly and seamlessly enables basic contact management tasks, showcasing the practicality of linked lists in real-world applications.

## Through this project, we gained a deeper understanding of:

## Data Structures: The advantages of linked lists over arrays for dynamic data storage.

## Memory Management: Efficient memory use with malloc and free functions in C.

## User Interface Design: Designing a simple, intuitive console-based interface.

## This project sets a solid foundation for exploring more advanced data structures, such as trees and hash tables, which could enhance search and organization efficiency. It also highlights the importance of linked lists in applications where data changes frequently and needs efficient, on-the-fly management.

## References

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language*. Prentice Hall.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). The MIT Press.