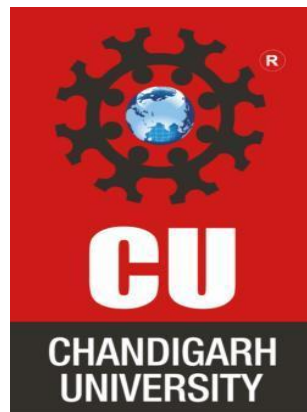


**DEPARTMENT OF
UNIVERSITY INSTITUTE OF COMPUTING
CHANDIGARH UNIVERSITY**



**PROJECT FILE
SUBJECT NAME: DATA STRUCTURE
SUBJECT CODE: 24CAP-152**

Submitted by:

Name : Sumit Kumar

UID : 24BCA10516

Section : 24BCA-7B

Submitted To:

Name : Mrs Monika Choudhary

Signature:

Acknowledgements

I would like to express my heartfelt gratitude to all those who helped me in the completion of this project.

First and foremost, I am deeply grateful to **Ms. Monika Choudhary**, whose guidance, feedback, and continuous encouragement were invaluable throughout the project. Her expertise and insights have been instrumental in shaping the direction and outcome of this work.

I also extend my thanks to Chandigarh University/UIC for providing the necessary resources and a conducive environment to carry out this project.

Lastly, I would like to thank my teacher and peers for their unwavering support and understanding, without which this project would not have been possible.

Name: Sumit Kumar

UID: 24BCA10516

Date: 17/04/2025

PROJECT FILE

Aim: The Contact List Manager

Aim of the Project

The aim of this project is to **create a simple Contact Manager** using **C programming** that allows users to **add, search, edit, and delete contacts** efficiently.

This project is designed to help users **store and manage contact details** dynamically using a **Doubly Linked List**, ensuring smooth **insertion, deletion, and navigation** of contacts. It provides a **user-friendly menu-driven interface**, making contact management simple and efficient.

This project also enhances **understanding of data structures**, particularly **linked lists, memory allocation, and CRUD operations** in C programming.

Introduction

The **Contact List Manager** is a simple C program that helps users **store, search, update, and delete contacts** easily. It uses a **Doubly Linked List**, allowing contacts to be added and removed without wasting memory.

Unlike an array, which has a fixed size, a **linked list grows dynamically**, making it more efficient for managing contacts. This project is great for **learning data structures** and improving **C programming skills**.

Why Use a Doubly Linked List?

- Allows **bi-directional traversal** (forward & backward).
- Supports **efficient insertion and deletion** without shifting elements (unlike arrays).
- Provides **dynamic memory allocation** to handle contacts dynamically.

Why is it Useful?

- **Efficient Contact Management** – Quickly add, search, update, or delete contacts.
- **Dynamic Storage** – Unlike arrays, linked lists allow unlimited contacts without wasting memory.
- **Easy Navigation** – Move forward and backward through the contact list.
- **Practical Application** – Demonstrates real-world use of **Data Structures in C**.

How Does This Project Work?

1. The program stores contacts using a **Doubly Linked List**.
2. The user can **add contacts** by entering a **name and phone number**.
3. The user can **view all saved contacts** in the list.
4. A **search function** helps find a contact by name.
5. Users can **update** an existing contact's phone number.
6. A **delete function** removes unwanted contacts.
7. The **navigation feature** allows moving forward and backward through contacts.
8. The program runs **through a menu-driven interface**, making it user-friendly.

Objectives

1. **Implement a Contact Management System using Doubly Linked List** – Store and manage contacts efficiently using dynamic memory allocation.
2. **Develop a Menu-Driven Program** – Create an interactive console-based program for managing contacts.
3. **Implement File Handling** – Store contacts permanently in a file to retain data even after exiting the program.
4. **Allow CRUD Operations** – Enable users to **add, search, edit, and delete** contacts efficiently.
5. **Ensure a User-Friendly Interface** – Use a **simple and clear menu-driven system** to make navigation easy for users.

Software & Tools Used

- **Programming Language:** C
- **Compiler:** GCC, Turbo C, or Code::Blocks
- **Operating System:** Windows/Linux

System Requirements

- **Minimum RAM:** 4GB
- **C Compiler Installed:** GCC, Turbo C, or Code::Blocks
- **Free Disk Space:** At least 50MB

Features

Users can efficiently store and manage their contacts.

1. **Add a Contact** – Store a new contact with a name and phone number.
2. **View All Contacts** – Display the complete list of saved contacts.
3. **Search for a Contact** – Find a contact by entering their name.
4. **Update Contact Details** – Modify an existing contact's phone number.
5. **Delete a Contact** – Remove a contact from the list.
6. **Move Forward & Backward** – Navigate through the contact list easily.
7. **Exit the Program** – Close the application safely without losing data (if file storage is added).

Project Scope

The **Contact List Manager** is a **mini-project** designed to help students learn:

1. **Doubly Linked Lists** – Efficiently storing and managing contacts.
2. **Dynamic Memory Allocation** – Managing memory efficiently using malloc() and free().
3. **CRUD Operations** – Performing Create, Read, Update, and Delete operations.
4. **User Interaction** – Developing a menu-driven system for contact management.
5. **Algorithm Development** – Implementing search, insert, and delete functions in C.

Project Implementation

Modules in the Project

1. Main Menu

- Displays options for the user to perform actions such as adding, searching, editing, deleting, moving through contacts, displaying all contacts, and exiting.

2. Add Contact

- Stores a new contact in the **doubly linked list**.
- Takes **name** and **phone number** as input.
- Inserts the contact at the **beginning** of the list.

3. Search Contact

- Finds a contact by **name**.
- If found, displays the contact's details.
- If not found, shows an error message.

4. Edit Contact

- Searches for a contact by **name**.
- If found, allows the user to **update the phone number**.

5. Delete Contact

- Finds a contact by **name** and removes it.
- Ensures correct re-linking of nodes before freeing memory.

6. Display Contacts

- Displays **all saved contacts** in the list.
- If the list is empty, prints a **No contacts available** message.

7. Move Forward

- Traverses the contact list **from head to tail**.
- Displays each contact's details.

8. Move Backward

- Moves to the **last contact** first.
- Traverses backward to display contacts in **reverse order**.

9. Exit

- Terminates the program with a **goodbye message**.

Code Explanation

`void addContact()`

- Creates a new contact using `malloc()`.
- Takes input for name and phone.
- Inserts the contact at the beginning of the doubly linked list.

`void displayContacts()`

- Traverses the entire doubly linked list and prints all contacts.
- If the list is empty, it displays a No contacts available message.

`void searchContact()`

- Takes a name as input.
- Iterates through the linked list to find a match.

- If found, displays the contact details; otherwise, shows a Contact Not Found message.

void editContact()

- Searches for a contact by name.
- If found, allows the user to update the phone number.

void deleteContact()

- Finds a contact by name.
- Adjusts the previous and next pointers to remove the contact.
- Frees the memory allocated for the contact.

void moveForward()

- Starts from the head of the list and moves to the end.
- Displays each contact in order.

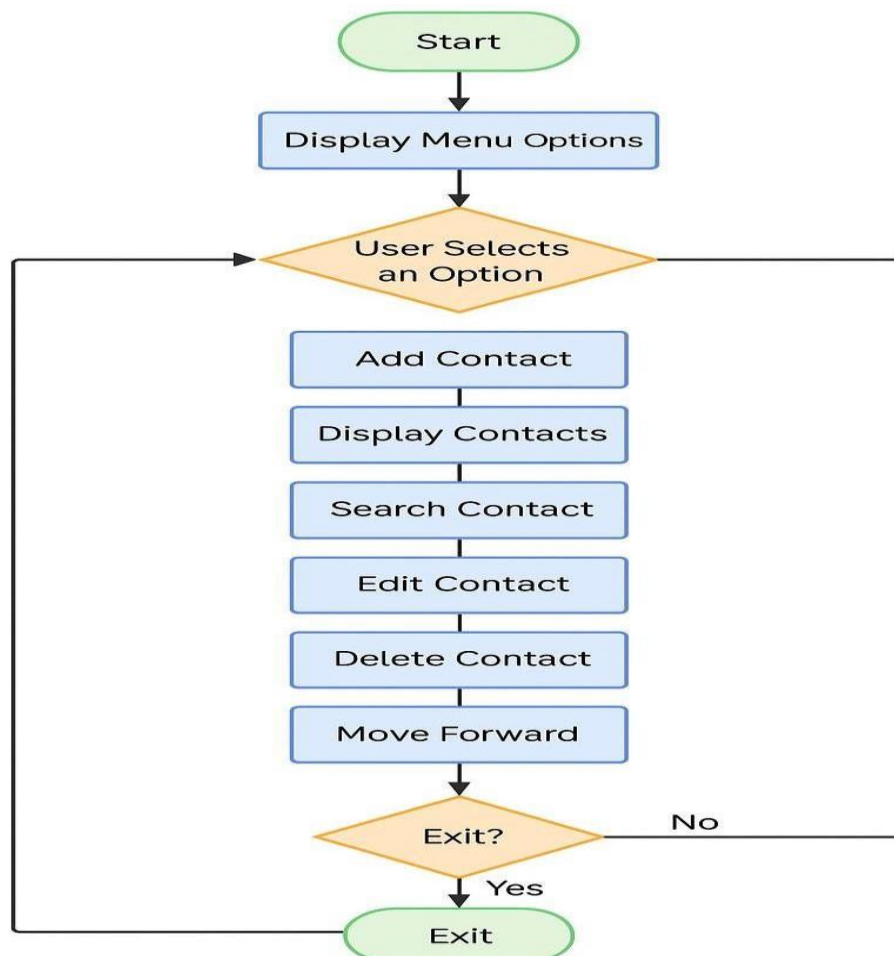
void moveBackward()

- Moves to the last contact first.
- Traverses backward through the doubly linked list to display contacts in reverse order.

Exit

- Terminates the program with a goodbye message.

Flowchart



Code Implementation

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

// Structure for Contact
typedef struct Contact
{
    char name[50];
    char phone[15];
    struct Contact *prev, *next;
} Contact;

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

// Function to add a new contact
void addContact()
{
    Contact *newContact = (Contact *)malloc(sizeof(Contact));
    printf("\nEnter Name: ");
    scanf(" %49[^\n]", newContact->name);
    printf("Enter Phone Number: ");
    scanf(" %14s", newContact->phone);

    newContact->prev = NULL;
    newContact->next = head;
    if (head != NULL)
    {
        head->prev = newContact;
    }
    head = newContact;
    printf("✔ Contact Added Successfully!\n");
}

// Function to display all contacts
void displayContacts()
{
    if (head == NULL)
```

```
{  
    printf("\n ✖ No contacts available.\n");  
    return;  
}  
Contact *temp = head;  
printf("\n ☞ Contact List:\n");  
while (temp != NULL)  
{  
    printf("Name: %s, Phone: %s\n", temp->name, temp->phone);  
    temp = temp->next;  
}  
}  
  
// Function to search for a contact  
void searchContact()  
{  
    if (head == NULL)  
    {  
        printf("\n ✖ No contacts available.\n");  
        return;  
    }  
    char searchName[50];  
    printf("\nEnter Name to Search: ");  
    scanf(" %49[^\n]", searchName);  
  
    Contact *temp = head;  
    while (temp != NULL)  
    {  
        if (strcmp(temp->name, searchName) == 0)  
        {  
            printf("☑ Contact Found: Name: %s, Phone: %s\n", temp->name, temp->phone);  
            return;  
        }  
        temp = temp->next;  
    }  
    printf("\n ✖ Contact Not Found.\n");  
}
```



```
// Function to delete a contact
void deleteContact()
{
    if (head == NULL)
    {
        printf("\n ✖ No contacts available to delete.\n");
        return;
    }
    char deleteName[50];
    printf("\nEnter Name to Delete: ");
    scanf(" %49[^\n]", deleteName);

    Contact *temp = head;
    while (temp != NULL)
    {
        if (strcmp(temp->name, deleteName) == 0)
        {
            if (temp->prev != NULL)
            {
                temp->prev->next = temp->next;
            }
            else
            {
                head = temp->next;
            }
            if (temp->next != NULL)
            {
                temp->next->prev = temp->prev;
            }
            free(temp);
            printf("\n ✔ Contact Deleted Successfully!\n");
            return;
        }
        temp = temp->next;
    }
    printf("\n ✖ Contact Not Found.\n");
}
```

```
// Function to edit a contact
void editContact()
{
    if (head == NULL)
    {
        printf("\n ✖ No contacts available to edit.\n");
        return;
    }
    char searchName[50];
    printf("\nEnter Name to Edit: ");
    scanf(" %49[^\n]", searchName);

    Contact *temp = head;
    while (temp != NULL)
    {
        if (strcmp(temp->name, searchName) == 0)
        {
            printf("Enter New Phone Number: ");
            scanf(" %14s", temp->phone);
            printf("\n ✔ Contact Updated Successfully!\n");
            return;
        }
        temp = temp->next;
    }
    printf("\n ✖ Contact Not Found.\n");
}

// Function to move forward through the contact list
void moveForward()
{
    if (head == NULL)
    {
        printf("\n ✖ No contacts available.\n");
        return;
    }
    Contact *temp = head;
    int count = 1;
    printf("\n ➡ Moving Forward:\n");
```

```
while (temp != NULL)
{
    printf("[%d] Name: %s, Phone: %s\n", count++, temp->name, temp->phone);
    temp = temp->next;
}

// Function to move backward through the contact list
void moveBackward()
{
    if (head == NULL)
    {
        printf("\n ✕ No contacts available.\n");
        return;
    }
    Contact *temp = head;

    // Move to the last contact
    while (temp->next != NULL)
    {
        temp = temp->next;
    }

    int count = 1;
    printf("\n ← Moving Backward:\n");
    while (temp != NULL)
    {
        printf("[%d] Name: %s, Phone: %s\n", count++, temp->name, temp->phone);
        temp = temp->prev;
    }
}

// Function to display menu
void menu()
{
    int choice;
    while (1)
    {
```

```
printf("\n🔗 Contact Manager\n");
printf("1. Add Contact\n");
printf("2. Display Contacts\n");
printf("3. Search Contact\n");
printf("4. Edit Contact\n");
printf("5. Delete Contact\n");
printf("6. Move Forward\n");
printf("7. Move Backward\n");
printf("8. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);


switch (choice)
{
    case 1: addContact(); break;
    case 2: displayContacts(); break;
    case 3: searchContact(); break;
    case 4: editContact(); break;
    case 5: deleteContact(); break;
    case 6: moveForward(); break;
    case 7: moveBackward(); break;
    case 8: printf("👋 Exiting Contact Manager. Goodbye!\n"); return;
    default: printf("❌ Invalid choice. Please try again.\n");
}
}
}

int main()
{
    menu();
    return 0;
}
```

Output

Output

Clear


 Contact Manager

1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit

Enter your choice: 1


Enter Name: Ritesh

Enter Phone Number: 8709624574

 Contact Added Successfully!

Output

Clear


 Contact Manager

1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit

Enter your choice: 1

Enter Name: Uday

Enter Phone Number: 9015155535

 Contact Added Successfully!

Output


Clear

 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 1

Enter Name: Nitin
Enter Phone Number: 7837925739
 Contact Added Successfully!

Output


Clear


 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 2

 Contact List:
Name: Nitin, Phone: 7837925739
Name: Uday, Phone: 9015155535
Name: Ritesh, Phone: 8709624574

Output

Clear

 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 3

Enter Name to Search: Ritesh
 Contact Found: Name: Ritesh, Phone: 8709624574

Output

Clear

```
📞 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 4

Enter Name to Edit: Ritesh
Enter New Phone Number: 8709624574
✅ Contact Updated Successfully!
```

Output

Clear

```
📞 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 6

➡ Moving Forward:
[1] Name: Nitin, Phone: 7837925739
[2] Name: Uday, Phone: 9015155535
[3] Name: Ritesh, Phone: 8709624574
```

Output

Clear


```
📞 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 7

➡ Moving Backward:
[1] Name: Ritesh, Phone: 8709624574
[2] Name: Uday, Phone: 9015155535
[3] Name: Nitin, Phone: 7837925739
```

Output


Clear


 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 5



Enter Name to Delete: Nitin
 Contact Deleted Successfully!

Output

Clear

 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 2

 Contact List:
Name: Uday, Phone: 9015155535
Name: Ritesh, Phone: 8709624574

 Contact Manager
1. Add Contact
2. Display Contacts
3. Search Contact
4. Edit Contact
5. Delete Contact
6. Move Forward
7. Move Backward
8. Exit
Enter your choice: 8
 Exiting Contact Manager. Goodbye!

Limitations & Future Enhancements

Limitations:

1. **No GUI, Only Console-Based** – The application runs in a terminal, making it less user-friendly.
2. **No Cloud Synchronization** – Contacts are lost after exiting the program since there's no online storage.
3. **Limited Search Functionality** – Users can only search by full name, not partial names or phone numbers.

Future Enhancements:

1. **Convert to a GUI-Based Application** – Develop a user-friendly interface using **Python (Tkinter/PyQt)** or **Java (Swing/JavaFX)**.
2. **Add Import/Export Feature** – Allow users to **save and load contacts** using **CSV or JSON** files.
3. **Sync with Online Services** – Integrate with **Google Contacts or iCloud** for cloud synchronization.

Conclusion

Through this project, I learned how to implement a **doubly linked list** for managing contacts efficiently. It enhanced my understanding of **dynamic memory allocation, file handling, and user interaction** in C programming. Debugging issues like **memory leaks and input handling** improved my problem-solving skills.

This project demonstrates the **practical application of data structures** in real-world scenarios. Although it's a **basic console-based** contact manager, it can be extended with **GUI, cloud integration, and database support** to create a fully functional contact management system.

References

Here are five useful websites for C programming references and resources:

1. GeeksforGeeks:

<https://www.geeksforgeeks.org/c-language-introduction/?ref=shm>

2. W3schools:

https://www.w3schools.com/c/c_intro.php

3. javapoints:

<https://www.javatpoint.com/c-programming-language-tutorial>

4. Learn c:

https://www.learn-c.org/#google_vignette

5. Codecademy:

<https://www.codecademy.com/learn/paths/c>