



Prepared By: Zian Rajeshkumar Surani
RA2311026050085 – AIML B



Data Structure & Algorithm

Singly and Doubly Linked List

Date

22 September, 2024





You are given a task of implementing a simple contact management system using a singly linked list. The system will manage contact names. Implement the following operations using a singly linked list and switch case. After every operation, display the current list of contacts. The operations to implement are:

- (i) Creation of the list: Allow the user to create a list of contact names by entering them one by one.
- (ii) Insertion of a new contact: Insert a new contact's name into a specific position in the list. The user should provide the name and the position at which it should be inserted.
- (iii) Deletion of a contact: Delete a contact's name from the list based on their position or name. Ask the user whether they want to delete by name or by position.
- (iv) Traversal of the list: Display all the contact names in the list in the current order.
- (v) Search for a contact: Search for a contact's name in the list and display whether or not the contact is found, along with their position if present.

Note: Github link is in the github logo icon





ANSWER



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

typedef struct Node {
    char name[50];
    struct Node* next;
} Node;

Node* createNode(char* name) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    strcpy(newNode->name, name);
    newNode->next = NULL;
    return newNode;
}

void createList(Node** head, int* size) {
    printf("Enter the number of contacts: ");
    scanf("%d", size);
    for (int i = 0; i < *size; i++) {
        char name[50];
        printf("Enter contact name %d: ", i + 1);
        scanf("%s", name);
        Node* newNode = createNode(name);
        if (*head == NULL) {
            *head = newNode;
        } else {
            Node* temp = *head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
}

void insertContact(Node** head, int position, char* name) {
    Node* newNode = createNode(name);
    if (position == 0) {
        newNode->next = *head;
        *head = newNode;
    } else {
        Node* temp = *head;
        for (int i = 0; i < position - 1; i++) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
}
```





```
void deleteContact(Node** head, char* name) {
    Node* temp = *head;
    Node* prev = NULL;
    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {
            if (prev == NULL) {
                *head = temp->next;
            } else {
                prev->next = temp->next;
            }
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
}

void displayList(Node* head) {
    Node* temp = head;
    printf("Contact list: ");
    while (temp != NULL) {
        printf("%s -> ", temp->name);
        temp = temp->next;
    }
    printf("NULL\n");
}

void searchContact(Node* head, char* name) {
    Node* temp = head;
    int position = 0;
    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {
            printf("%s found at position %d\n", name, position);
            return;
        }
        position++;
        temp = temp->next;
    }
    printf("%s not found\n", name);
}
```





```
int main() {
    Node* head = NULL;
    int size = 0;

    while (1) {
        printf("1. Create the list of contacts\n");
        printf("2. Insert a new contact\n");
        printf("3. Delete a contact\n");
        printf("4. Display contact list\n");
        printf("5. Search for a contact\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        int choice;
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&head, &size);
                displayList(head);
                break;
            case 2:
                char name[50];
                printf("Enter the contact's name to insert: ");
                scanf("%s", name);
                int position;
                printf("Enter the position (0-based index) to insert the contact: ");
                scanf("%d", &position);
                insertContact(&head, position, name);
                displayList(head);
                break;
            case 3:
                char deleteMethod;
                printf("Delete by name or position? (n/p): ");
                scanf(" %c", &deleteMethod);
                if (deleteMethod == 'n') {
                    char deleteName[50];
                    printf("Enter the contact's name to delete: ");
                    scanf("%s", deleteName);
                    deleteContact(&head, deleteName);
                    displayList(head);
                } else {
                    int deletePosition;
                    printf("Enter the position (0-based index) to delete the contact: ");
                    scanf("%d", &deletePosition);
                    // implement delete by position
                }
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                char searchName[50];
                printf("Enter the contact's name to search: ");
                scanf("%s", searchName);
                searchContact(head, searchName);
                break;
            case 6:
                printf("Exiting the program...\n");
                return 0;
        }
    }
    return 0;
}
```





OUTPUT

```
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 1
Enter the number of contacts: 2
Enter contact name 1: zian
Enter contact name 2: dishu
Contact list: zian -> dishu -> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 2
Enter the contact's name to insert: ritu
Enter the position (0-based index) to insert the contact: 1
Contact list: zian -> ritu -> dishu -> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 3
Delete by name or position? (n/p): n
Enter the contact's name to delete: dishu
Contact list: zian -> ritu -> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 4
Contact list: zian -> ritu -> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 5
Enter the contact's name to search: zian
```





```
zian found at position 0
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 6
Exiting the program...
```





You are tasked with implementing a simple contact management system using a doubly linked list. The system will manage contact names. Implement the following operations using a doubly linked list and switch-case. After every operation, display the current list of contacts. The operations to implement are:

(i) Creation of the list: Allow the user to create a list of contact names by entering them one by one.

(ii) Insertion of a new contact:

Insert a new contact's name into a specific position in the list. The user should provide the name and the position at which it should be inserted.

(iii) Deletion of a contact:

Delete a contact's name from the list based on their position or name. Ask the user whether they want to delete by name or by position.

(iv) Traversal of the list (in both directions):

Display all the contact names in the list in the current order (forward traversal) and then display them in reverse order (backward traversal).

(v) Search for a contact:

Search for a contact's name in the list and display whether or not the contact is found, along with their position if present.



ANSWER



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

typedef struct Node {
    char name[50];
    struct Node* next;
    struct Node* prev;
} Node;

Node* createNode(char* name) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    strcpy(newNode->name, name);
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void createList(Node** head, Node** tail, int* size) {
    printf("Enter the number of contacts: ");
    scanf("%d", size);
    for (int i = 0; i < *size; i++) {
        char name[50];
        printf("Enter contact name %d: ", i + 1);
        scanf("%s", name);
        Node* newNode = createNode(name);
        if (*head == NULL) {
            *head = newNode;
            *tail = newNode;
        } else {
            newNode->prev = *tail;
            (*tail)->next = newNode;
            *tail = newNode;
        }
    }
}

void insertContact(Node** head, Node** tail, int position, char* name) {
    Node* newNode = createNode(name);
    if (position == 0) {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    } else {
        Node* temp = *head;
        for (int i = 0; i < position - 1; i++) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        newNode->prev = temp;
        temp->next = newNode;
        if (newNode->next != NULL) {
            newNode->next->prev = newNode;
        } else {
            *tail = newNode;
        }
    }
}
```





```
void deleteContact(Node** head, Node** tail, char* name) {
    Node* temp = *head;
    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {
            if (temp->prev != NULL) {
                temp->prev->next = temp->next;
            } else {
                *head = temp->next;
            }
            if (temp->next != NULL) {
                temp->next->prev = temp->prev;
            } else {
                *tail = temp->prev;
            }
            free(temp);
            return;
        }
        temp = temp->next;
    }
}

void displayListForward(Node* head) {
    Node* temp = head;
    printf("Contact list (forward): ");
    while (temp != NULL) {
        printf("%s <-> ", temp->name);
        temp = temp->next;
    }
    printf("NULL\n");
}

void displayListBackward(Node* tail) {
    Node* temp = tail;
    printf("Contact list (backward): ");
    while (temp != NULL) {
        printf("%s <-> ", temp->name);
        temp = temp->prev;
    }
    printf("NULL\n");
}

void searchContact(Node* head, char* name) {
    Node* temp = head;
    int position = 0;
    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {
            printf("%s found at position %d\n", name, position);
            return;
        }
        position++;
        temp = temp->next;
    }
    printf("%s not found\n", name);
}
```





```
int main() {
    Node* head = NULL;
    Node* tail = NULL;
    int size = 0;

    while (1) {
        printf("1. Create the list of contacts\n");
        printf("2. Insert a new contact\n");
        printf("3. Delete a contact\n");
        printf("4. Display contact list\n");
        printf("5. Search for a contact\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        int choice;
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&head, &tail, &size);
                displayListForward(head);
                displayListBackward(tail);
                break;
            case 2:
                char name[50];
                printf("Enter the contact's name to insert: ");
                scanf("%s", name);
                int position;
                printf("Enter the position (0-based index) to insert the contact: ");
                scanf("%d", &position);
                insertContact(&head, &tail, position, name);
                displayListForward(head);
                displayListBackward(tail);
                break;
            case 3:
                char deleteMethod;
                printf("Delete by name or position? (n/p): ");
                scanf(" %c", &deleteMethod);
                if (deleteMethod == 'n') {
                    char deleteName[50];
                    printf("Enter the contact's name to delete: ");
                    scanf("%s", deleteName);
                    deleteContact(&head, &tail, deleteName);
                    displayListForward(head);
                    displayListBackward(tail);
                } else {
                    int deletePosition;
                    printf("Enter the position (0-based index) to delete the contact: ");
                    scanf("%d", &deletePosition);
                    // implement delete by position
                    Node* temp = head;
                    for (int i = 0; i < deletePosition; i++) {
                        temp = temp->next;
                    }
                    if (temp->prev != NULL) {
                        temp->prev->next = temp->next;
                    } else {
                        head = temp->next;
                    }
                    if (temp->next != NULL) {
                        temp->next->prev = temp->prev;
                    } else {
                        tail = temp->prev;
                    }
                    free(temp);
                    displayListForward(head);
                    displayListBackward(tail);
                }
                break;
        }
    }
}
```





```
case 4:  
    displayListForward(head);  
    displayListBackward(tail);  
    break;  
case 5:  
    char searchName[50];  
    printf("Enter the contact's name to search: ");  
    scanf("%s", searchName);  
    searchContact(head, searchName);  
    break;  
case 6:  
    printf("Exiting the program...\\n");  
    return 0;  
}  
}  
return 0;
```

OUTPUT

```
1. Create the list of contacts  
2. Insert a new contact  
3. Delete a contact  
4. Display contact list  
5. Search for a contact  
6. Exit  
Enter your choice: 1  
Enter the number of contacts: 2  
Enter contact name 1: zian  
Enter contact name 2: jp  
Contact list (forward): zian <-> jp <-> NULL  
Contact list (backward): jp <-> zian <-> NULL
```





```
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 2
Enter the contact's name to insert: eshwar
Enter the position (0-based index) to insert the contact: 1
Contact list (forward): zian <-> eshwar <-> jp <-> NULL
Contact list (backward): jp <-> eshwar <-> zian <-> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 3
Delete by name or position? (n/p): n
Enter the contact's name to delete: jp
Contact list (forward): zian <-> eshwar <-> NULL
Contact list (backward): eshwar <-> zian <-> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 4
Contact list (forward): zian <-> eshwar <-> NULL
Contact list (backward): eshwar <-> zian <-> NULL
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 5
Enter the contact's name to search: eshwar
eshwar found at position 1
1. Create the list of contacts
2. Insert a new contact
3. Delete a contact
4. Display contact list
5. Search for a contact
6. Exit
Enter your choice: 6
Exiting the program...
```

