DSA ASSIGNMENT 3

1. 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.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Node {            //creating structure

   char name[100];

   struct Node* next;      //declaring a pointer to the next node in the singly linked list

};
```

```
struct Node* head = NULL;    //declaring and initiating the head node as NULL

void displayContacts() {            // function to display the contacts
   struct Node* current = head;
   if (current == NULL) {                    // to check if the list is empty
      printf("Contact list is empty.\n");
      return;
   }
   while (current != NULL) {             // to diaplay the elements in the contacts lsit
      printf("%s -> ", current->name);
      current = current->next;
   }
   printf("NULL\n");
}

int main() {                          //main function
   int choice, position, count, i;
   char name[100];

   while (1) {                                  //to choose what operation is gonna be done
      printf("\n1. 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: ");
```

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

switch (choice) {                        //switch case
   case 1:                               // Create the list
      printf("Enter the number of contacts: ");
      scanf("%d", &count);
      for (i = 0; i < count; i++) {
         printf("Enter contact name %d: ", i + 1);
         scanf("%s", name);
         struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
         strcpy(newNode->name, name);
         newNode->next = NULL;

         if (head == NULL) {
            head = newNode;                // First node
         } else {
            struct Node* current = head;
            while (current->next != NULL) {
               current = current->next;              // move to the end of the list
            }
            current->next = newNode;                 // insert at the end
         }
      }
      displayContacts();
      break;

   case 2:                               // Inserting a new contact
      printf("Enter the contact's name to insert: ");
```

```
    scanf("%s", name);
    printf("Enter the position (0-based index) to insert the contact: ");
    scanf("%d", &position);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->name, name);
    newNode->next = NULL;

    if (position == 0) {
        newNode->next = head;
        head = newNode;
    } else {
        struct Node* current = head;
        for (i = 0; i < position - 1 && current != NULL; i++) {
            current = current->next;
        }
        if (current == NULL) {
            printf("Position out of bounds!\n");
            free(newNode);
        } else {
            newNode->next = current->next;
            current->next = newNode;
        }
    }
    displayContacts();
    break;

case 3:                                 //  Delete a contact
    printf("Delete by name or position? (n/p): ");
```

```
char delChoice;
scanf(" %c", &delChoice);
if (delChoice == 'n') {
    printf("Enter the contact's name to delete: ");
    scanf("%s", name);
    struct Node* current = head;
    struct Node* prev = NULL;
    while (current != NULL && strcmp(current->name, name) != 0) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        printf("Contact not found!\n");
    } else {
        if (prev == NULL) {
            head = current->next;
        } else {
            prev->next = current->next;
        }
        free(current);
    }
} else if (delChoice == 'p') {
    printf("Enter the position (0-based index) to delete the contact: ");
    scanf("%d", &position);
    struct Node* current = head;
    struct Node* prev = NULL;
    for (i = 0; i < position && current != NULL; i++) {
        prev = current;
```

```
            current = current->next;
        }
        if (current == NULL) {
            printf("Position out of bounds!\n");
        } else {
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
        }
    }
    displayContacts();
    break;

case 4:                         // displaying contact list
    displayContacts();
    break;

case 5:                                 // searching for a contact
    printf("Enter the contact's name to search: ");
    scanf("%s", name);
    struct Node* current = head;
    position = 0;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            printf("%s found at position %d\n", name, position);
```

```
                break;
            }
            current = current->next;
            position++;
        }
        if (current == NULL) {
            printf("%s not found.\n", name);
        }
        break;

    case 6:                         // exiting from the program
        printf("Exiting the program...\n");
        return 0;

    default:
        printf("Invalid choice! Please try again.\n");
    }
  }

  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: 3
Enter contact name 1: anish
Enter contact name 2: aravind
Enter contact name 3: haris
anish -> aravind -> haris -> 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: alaguraja
Enter the position (0-based index) to insert the contact:
3
anish -> aravind -> haris -> alaguraja -> 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
anish -> aravind -> alaguraja -> 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: alaguraja
alaguraja found at position 2

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...


...Program finished with exit code 0
Press ENTER to exit console.
```

2. 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.

Code:

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

struct Node {                    //creating structure
   char name[100];
   struct Node* prev;            //creating pointer nodes
   struct Node* next;
};

struct Node* head = NULL;                //initiating nodes as null
struct Node* tail = NULL;

void displayContacts() {                 //function to display contacts
   struct Node* current = head;
   printf("Contact list (forward): ");
   while (current != NULL) {                // printing the list in forward manner
      current = current->next;
   }
   printf("NULL\n");

   current = tail;
   printf("Contact list (backward): ");          //pirnting list in backward manner
   while (current != NULL) {
      printf("%s <-> ", current->name);
      current = current->prev;
   }
   printf("NULL\n");
```

```
}

void createList(int count) {          //function to create list
   for (int i = 0; i < count; i++) {
      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      printf("Enter contact name %d: ", i + 1);
      scanf("%s", newNode->name);
      newNode->prev = tail;
      newNode->next = NULL;

      if (tail != NULL) {
         tail->next = newNode;
      } else {
         head = newNode; // First node
      }
      tail = newNode; // Update tail
   }
   displayContacts();
}

void insertContact(char* name, int position) {                    // function to insert an element
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   strcpy(newNode->name, name);
   newNode->prev = NULL;
   newNode->next = NULL;

   if (position == 0) {                    // Insert at head
      newNode->next = head;
```

```
        if (head != NULL) {
            head->prev = newNode;
        }
        head = newNode;
        if (tail == NULL) {                         // First insertion
            tail = newNode;
        }
    } else {
        struct Node* current = head;
        for (int i = 0; i < position - 1 && current != NULL; i++) {
            current = current->next;
        }
        if (current != NULL) {
            newNode->next = current->next;
            newNode->prev = current;
            current->next = newNode;
            if (newNode->next != NULL) {
                newNode->next->prev = newNode;
            } else {
                tail = newNode;                     // Update tail if inserted at the end
            }
        } else {
            printf("Position out of bounds!\n");
            free(newNode);
        }
    }
    displayContacts();
}
```

```
void deleteContact(char* name, int position, int byName) {
    struct Node* current;
    if (byName) {
        current = head;
        while (current != NULL && strcmp(current->name, name) != 0) {
            current = current->next;
        }
    } else {
        current = head;
        for (int i = 0; i < position && current != NULL; i++) {
            current = current->next;
        }
    }

    if (current != NULL) {
        if (current->prev != NULL) {
            current->prev->next = current->next;
        } else {
            head = current->next;              // updating head if needed
        }
        if (current->next != NULL) {
            current->next->prev = current->prev;
        } else {
            tail = current->prev;              // updating tail if needed
        }
        free(current);
        displayContacts();
```

```
    } else {
        printf("Contact not found!\n");
    }
}

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

int main() {            //main function
    int choice;
    char name[100];
    int position, count;

    while (1) {                     //choosing what operation to be done
        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: ");
scanf("%d", &choice);

switch (choice) {                    //switch case
   case 1:                        //creating the list
      printf("Enter the number of contacts: ");
      scanf("%d", &count);
      createList(count);
      break;
   case 2:                  //inserting an element
      printf("Enter the contact's name to insert: ");
      scanf("%s", name);
      printf("Enter the position (0-based index) to insert the contact: ");
      scanf("%d", &position);
      insertContact(name, position);
      break;
   case 3:                //delete contact with name or position
      printf("Delete by name or position? (n/p): ");
      char option;
      scanf(" %c", &option);
      if (option == 'n') {
         printf("Enter the name to delete: ");
         scanf("%s", name);
         deleteContact(name, -1, 1);
      } else {
```

```
            printf("Enter the position (0-based index) to delete the contact: ");
            scanf("%d", &position);
            deleteContact(NULL, position, 0);
        }
        break;
    case 4:
        displayContacts();
        break;
    case 5:                         //search contact
        printf("Enter the contact's name to search: ");
        scanf("%s", name);
        searchContact(name);
        break;
    case 6:                             //exit progrmam
        printf("Exiting the program...\n");
        return 0;
    default:                            //default message if there is any invalid choice is given
        printf("Invalid choice! Please try again.\n");
    }
  }
}
```

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: 3
Enter contact name 1: anish
Enter contact name 2: aravind
Enter contact name 3: haris
Contact list (forward): anish <-> aravind <-> haris <-> NULL
Contact list (backward): haris <-> aravind <-> anish <-> 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: alaguraja
Enter the position (0-based index) to insert the contact: 2
Contact list (forward): anish <-> aravind <-> alaguraja <-> haris <-> NULL
Contact list (backward): haris <-> alaguraja <-> aravind <-> anish <-> 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 name to delete: haris
Contact list (forward): anish <-> aravind <-> alagura
Contact list (backward): alaguraja <-> aravind <-> ar
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: haris
haris not found
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: aravind
aravind 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...


...Program finished with exit code 0
Press ENTER to exit console.
```

By ,

ANISH IDHAYAN I,

https://github.com/AnishIdhayan-_1412/DSA-_ASSIGNMENT-_3/tree/main

AIML-_B-_II YEAR,

RA2311026050078.