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

typedef struct node *nptr;

//struct
typedef struct node{
    struct node* prev;
    struct node* next;
    char* str;
} dll;

nptr insertAtBeginning(nptr head, char *str)
{
    nptr curr = head; //cursor
    nptr p = malloc(sizeof(dll)); //mallocing
    p->str = malloc(sizeof(char)*100);
    p->str = str; //adding string
    if(curr==NULL) //checking if head is null
    {
        head = p;
        p->next = p;
        p->prev = p;
        return(head); //assignment
    }
    else
    {
        p->next = curr;
        p->prev = curr->prev;
        (curr->prev)->next = p;
        curr->prev = p; //assignment
        return(p); //returning p because insert *before* head
    }
}

//Same logic as insertAtBeginning, except that this returns head
nptr insertAtEnd(nptr head, char *str)
{
    nptr curr = head;
    nptr p = malloc(sizeof(dll));
    p->str = malloc(sizeof(char)*100);
    p->str = str;
    if(curr==NULL)
    {
        head = p;
        p->next = p;
        p->prev = p;
        return(head);
    }
    else
    {
        p->next = curr;
        p->prev = curr->prev;
        (curr->prev)->next = p;
        curr->prev = p;
```

```
        return(head);
    }
}

nptr insertAfter(nptr head, char* pre, char* str)
{
    nptr curr; //cursor
    curr = head;
    if(head==NULL)
    {
        printf("Empty.\n");
        return head;
    }
    do
    {
        if(!(strcmp((curr->prev)->str, pre))) //checking if string
found at prev
        {
            nptr p = malloc(sizeof(dll));
            p->str = str;
            p->next = curr;
            p->prev = curr->prev;
            curr->prev = p;
            (p->prev)->next = p; //proper assignment done
            if (p==head->prev) //checking if head needs to be replaced
                return(p);
            return(head);
        }
        curr = curr->prev;
    }
    while(curr!=head); //loop till head is encountered twice
}

//Same logic as insertAfter except that the loop is reversed now and
assignments have been adjusted accordingly
nptr insertBefore(nptr head, char* nex, char* str)
{
    nptr curr;
    curr = head;
    if(head==NULL)
    {
        printf("Empty.\n");
        return head;
    }
    do
    {
        if(!(strcmp((curr->next)->str, nex)))
        {
            nptr p = malloc(sizeof(dll));
            p->str = str;
            p->next = curr->next;
            p->prev = curr;
            curr->next = p;
            (p->next)->prev = p;
            if (p->next==head)
                return(p);
            return(head);
```

```
        }
        curr = curr->next;
    }
    while(curr!=head);
}


nptr deleteNode(nptr head, char *str)
{
    if(head==NULL)
    {
        printf("Empty.\n");
        return head;
    }
    nptr curr; //Cursor
    curr = head->next; //note this: starting from head->next instead
of head
    do
    {
        if(!(strcmp(curr->str, str)))  //checking if str is in the
list
        {
            if(curr == head)
                return NULL;
            (curr->next)->prev = curr->prev;
            (curr->prev)->next = curr->next;
            free(curr);  //delete curr if str found
            return(head);
        }
        curr = curr->next;
    }
    while(curr!=head); //loop till head is encountered
    if(!(strcmp(curr->str, str))) //if control reaches here, curr is
at head. Check if str is at head.
    {
        (curr->next)->prev = curr->prev;
        (curr->prev)->next = curr->next;
        curr = curr->next; //make necessary adjustments
        free(head); //free head
        return(curr); //send curr back instead of head
    }
    printf("Element not found.\n");
    return head;
}

void search(nptr head, char* str)
{
    nptr curr;
    curr = head;
    int c = 0;
    if(head==NULL)
    {
        printf("Empty.\n");
        return;
    }
    do
    {
```

```
        if(!(strcmp(curr->str, str))) //check if element is found
        {
            printf("Element found. Index: %d\n", c);
            return;
        }
        c++;
        curr = curr->next;
    }
    while(curr!=head); //loop till head is encountered twice
    printf("Element not found!\n");
    return;
}

void display(nptr head)
{
    if(head == NULL)
    {
        printf("Empty.\n");
        return;
    }
    nptr temp = head;
    nptr curr = head;
    while(1) //looping till head if found twice
    {
        printf("%s<-->", curr->str);
        curr = curr->next;
        if(curr==temp)
        {
            printf("%s\n", curr->str);
            break; //breaking if head found second time
        }
    }
}

//main just checks input and executes appropriate functions (and asks
for arguments, if any)

int main(void)
{
    nptr head;
    while(1)
    {
        printf("Press 1 to insert at beginning, 2 to insert at end, 3
to insert before, 4 to insert after, 5 to delete node, 6 to search, 7
to display and 8 to exit.\n");
        int n;
        scanf("%d", &n);
        if(n==1)
        {
            printf("Enter Element: ");
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            insertAtBeginning(head, string);
        }
        else if(n==2)
        {
            printf("Enter Element: ");
```

```
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            insertAtEnd(head, string);
        }
        else if(n==3)
        {
            printf("Enter Element to Insert: ");
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            printf("Enter Element before which the string needs to be
inserted: ");
            char *before = malloc(sizeof(char)*100);
            scanf("%s", before);
            insertBefore(head, before, string);
        }
        else if(n==4)
        {
            printf("Enter Element to Insert: ");
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            printf("Enter Element after which the string needs to be
inserted: ");
            char *after = malloc(sizeof(char)*100);
            scanf("%s", after);
            insertAfter(head, after, string);
        }
        else if(n==5)
        {
            printf("Enter Element to Delete: ");
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            deleteNode(head, string);
        }
        else if(n==6)
        {
            printf("Enter Element to Search: ");
            char *string = malloc(sizeof(char)*100);
            scanf("%s", string);
            search(head, string);
        }
        else if(n==7)
            display(head);
        else if(n==8)
            exit(0);
        else
            printf("Invalid command. Please enter again.\n");
    }
}
```