

WAP to Implement doubly link list with primitive operations

I. Create a doubly linked list.

II. Insert a new node to the left of the node.

III. Delete the node based on a specific value

IV. Display the contents of the list

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node* prev;
```

```
    struct Node* next;
```

```
};
```

```
struct Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    if (newNode == NULL) {
```

```
        printf("Memory allocation failed\n");
```

```
        exit(EXIT_FAILURE);
```

```
    }
```

```
    newNode->data = data;
```

```
    newNode->prev = NULL;
```

```
    newNode->next = NULL;
```

```
    return newNode;
```

```
}
```

```

void insertAtBeginning(struct Node** head, int data) {

    struct Node* newNode = createNode(data);

    if (*head == NULL) {

        *head = newNode;

    } else {

        newNode->next = *head;

        (*head)->prev = newNode;

        *head = newNode;

    }

}

void insertBeforeNode(struct Node** head, int key, int data) {

    if (*head == NULL) {

        printf("List is empty\n");

        return;

    }

    struct Node* newNode = createNode(data);

    struct Node* current = *head;

    while (current) {

        if (current->data == key) {

            if (current->prev) {

                current->prev->next = newNode;

                newNode->prev = current->prev;

            } else {

```

```

        *head = newNode;
    }

    newNode->next = current;
    current->prev = newNode;
    return;
}

current = current->next;
}

printf("Key not found in the list\n");
}

void deleteNode(struct Node** head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = *head;
    int count = 1;

    while (current && count < pos) {
        current = current->next;
        count++;
    }
}

```

```
}
```

```
if (current == NULL) {  
    printf("Position %d is beyond the length of the list\n", pos);  
    return;  
}
```

```
if (current->prev) {  
    current->prev->next = current->next;  
} else {  
    *head = current->next;  
}
```

```
if (current->next) {  
    current->next->prev = current->prev;  
}
```

```
free(current);  
printf("Node at position %d deleted\n", pos);  
}
```

```
void displayList(struct Node* head) {  
    if (head == NULL) {  
        printf("List is empty\n");  
        return;  
    }
```

```

struct Node* current = head;

while (current) {

    printf("%d-> ", current->data);

    current = current->next;

}

printf("\n");
}

```

```

void freeList(struct Node* head) {

    struct Node* current = head;

    struct Node* nextNode;

    while (current) {

        nextNode = current->next;

        free(current);

        current = nextNode;

    }

}

```

```

int main() {

    struct Node* head = NULL;

    int ch, newData, pos, key;

    while (1) {

```

```
printf("\nMenu\n");

printf("1. Insert at the beginning\n");

printf("2. Insert before a node\n");

printf("3. Delete a node\n");

printf("4. Display list\n");

printf("5. Free doubly linked list and exit\n");

printf("Enter your choice: ");

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

```
switch (ch) {
```

```
    case 1:
```

```
        printf("Enter data to insert at the beginning: ");

        scanf("%d", &newData);

        insertAtBeginning(&head, newData);

        break;
```

```
    case 2:
```

```
        printf("Enter the value before which you want to insert: ");

        scanf("%d", &key);

        printf("Enter data to insert: ");

        scanf("%d", &newData);

        insertBeforeNode(&head, key, newData);

        break;
```

```
    case 3:
```

```
        printf("Enter the position you wish to delete: ");
```

```

        scanf("%d", &key);

        deleteNode(&head, key);

        break;

case 4:

    printf("Doubly linked list: ");

    displayList(head);

    break;

case 5:

    freeList(head);

    printf("Exiting the program\n");

    return 0;

default:

    printf("Invalid choice\n");

    }

    }

return 0;

}

```

OUTPUT:

```

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 3

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 3
Enter data to insert: 4

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 4-> 3-> 2-> 1->

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 2
Node at position 2 deleted

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 4-> 2-> 1->

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 4-> 2-> 1->

```