Name: Aanshula Shetty

Branch: AIML

Roll No: 45

Subject: BAI

Experiment No 1

Objective: Implementation of Singly and Doubly Linked list program and its operations.

Software Used: VS code

Theory:

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. It consists of two parts: one is the data part and the other is the address part.

Types Of Linked List:

- Singly Linked List
- Doubly or Two Way Linked List
- Circular Linked List
- Circular Doubly Linked List

Operations to be performed:

- **traverse()/search:** To see the contents of the linked list, it is necessary to traverse the given linked list. The given traverse() function traverses and prints the content of the linked list.
- **insertAtFront():** This function simply inserts an element at the front/beginning of the linked list.
- **insertAtEnd():** This function inserts an element at the end of the linked list.
- **insertAtPosition():** This function inserts an element at a specified position in the linked list.
- **deleteFirst():** This function simply deletes an element from the front/beginning of the linked list.
- **deleteEnd():** This function simply deletes an element from the end of the linked list.
- **deletePosition():** This function deletes an element from a specified position in the linked list.

Single Linked List:

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way.

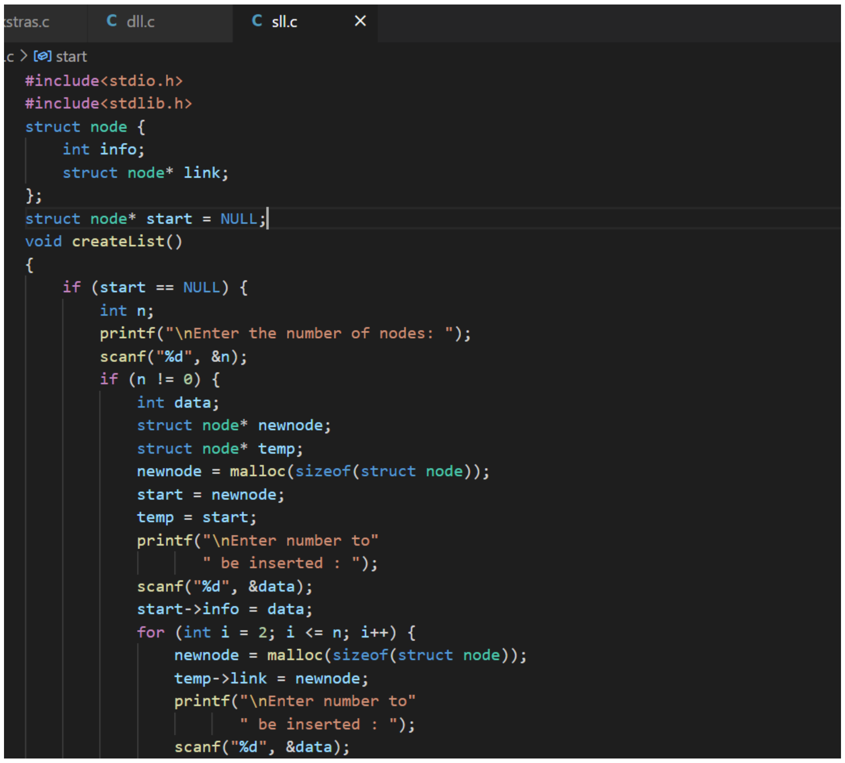
Advantages of Singly Linked List-

- It is very easier for the accessibility of a node in the forward direction.
- The insertion and deletion of a node are very easy.
- The Requirement will less memory when compared to doubly, circular or doubly circular linked list.
- The Singly linked list is the very easy data structure to implement.
- During the execution, we can allocate or deallocate memory easily.
- Insertion and deletion of elements don’t need the movement of all the elements when compared to an array.

Disadvantages of Singly Linked List-

- Accessing the preceding node of a current node is not possible as there is no backward traversal.
- The Accessing of a node is very time-consuming.

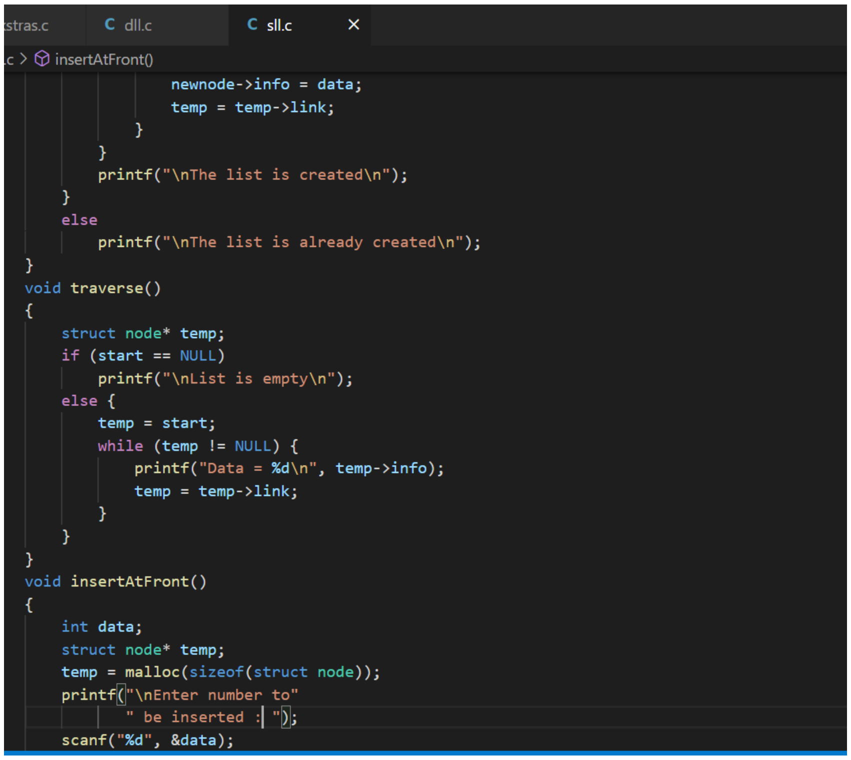
C program for single linked list:



stras.c   dll.c   sll.c

.c > start

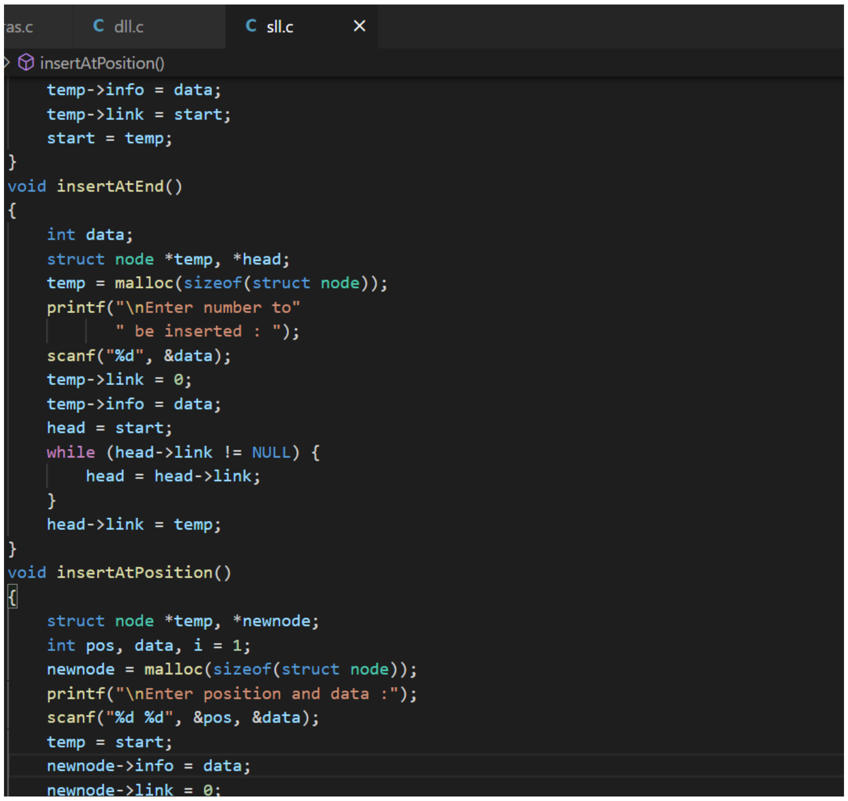
```c
#include<stdio.h>
#include<stdlib.h>
struct node {
    int info;
    struct node* link;
};
struct node* start = NULL;
void createList()
{
    if (start == NULL) {
        int n;
        printf("\nEnter the number of nodes: ");
        scanf("%d", &n);
        if (n != 0) {
            int data;
            struct node* newnode;
            struct node* temp;
            newnode = malloc(sizeof(struct node));
            start = newnode;
            temp = start;
            printf("\nEnter number to"
                   " be inserted : ");
            scanf("%d", &data);
            start->info = data;
            for (int i = 2; i <= n; i++) {
                newnode = malloc(sizeof(struct node));
                temp->link = newnode;
                printf("\nEnter number to"
                       " be inserted : ");
                scanf("%d", &data);
```



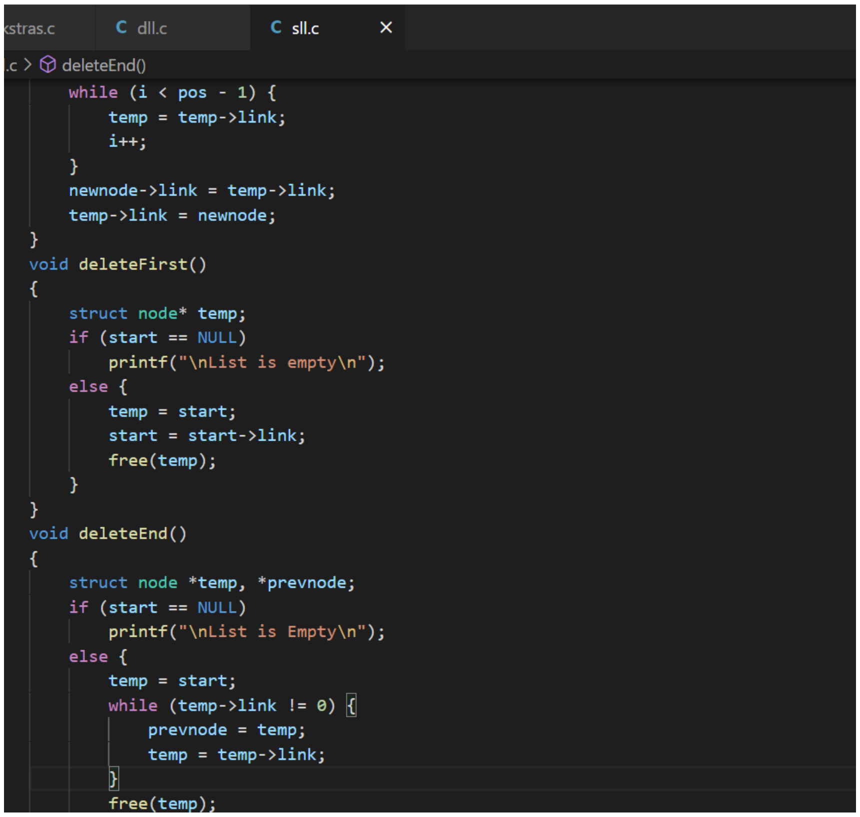
stras.c   dll.c   sll.c

.c > insertAtFront()

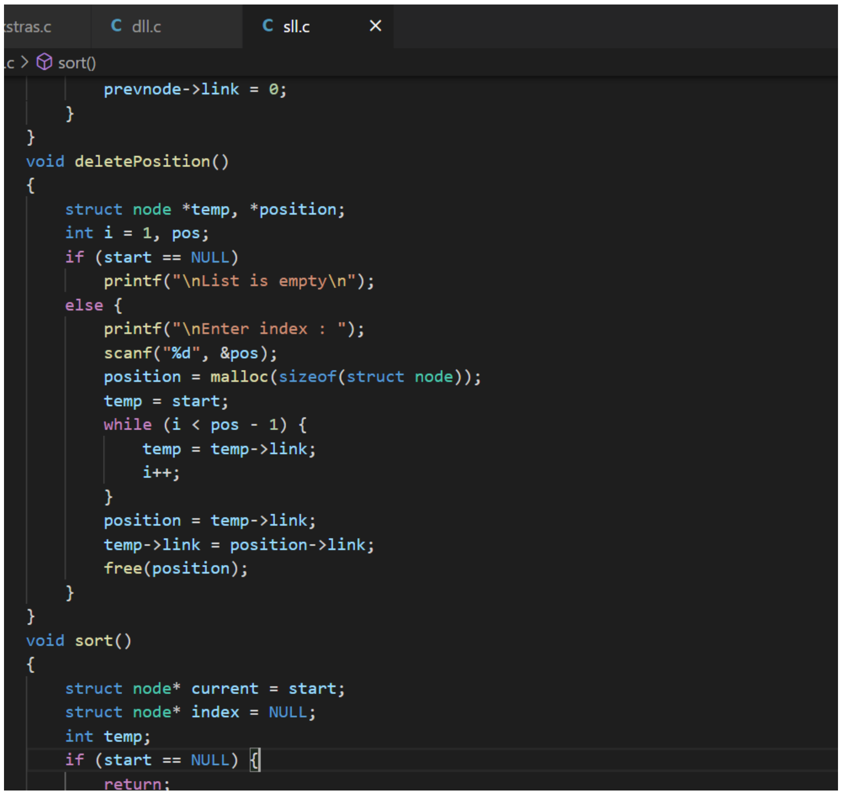
```c
                newnode->info = data;
                temp = temp->link;
            }
        }
        printf("\nThe list is created\n");
    }
    else
        printf("\nThe list is already created\n");
}
void traverse()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        while (temp != NULL) {
            printf("Data = %d\n", temp->info);
            temp = temp->link;
        }
    }
}
void insertAtFront()
{
    int data;
    struct node* temp;
    temp = malloc(sizeof(struct node));
    printf("\nEnter number to"
           " be inserted : ");
    scanf("%d", &data);
```



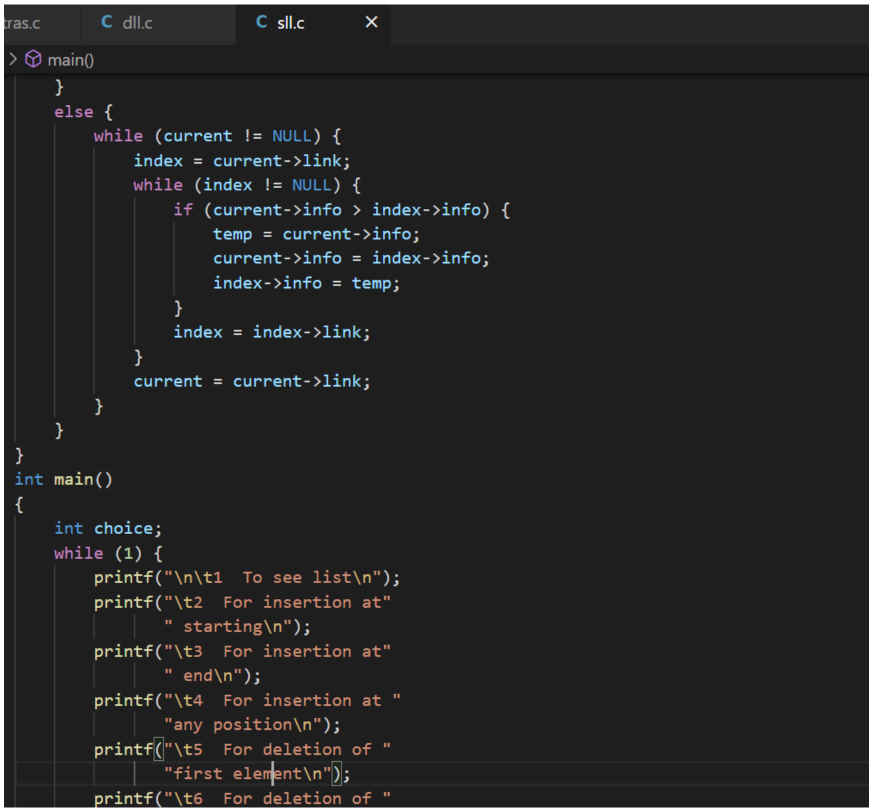
```
    temp->info = data;
    temp->link = start;
    start = temp;
}
void insertAtEnd()
{
    int data;
    struct node *temp, *head;
    temp = malloc(sizeof(struct node));
    printf("\nEnter number to"
           " be inserted : ");
    scanf("%d", &data);
    temp->link = 0;
    temp->info = data;
    head = start;
    while (head->link != NULL) {
        head = head->link;
    }
    head->link = temp;
}
void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));
    printf("\nEnter position and data :");
    scanf("%d %d", &pos, &data);
    temp = start;
    newnode->info = data;
    newnode->link = 0;
```



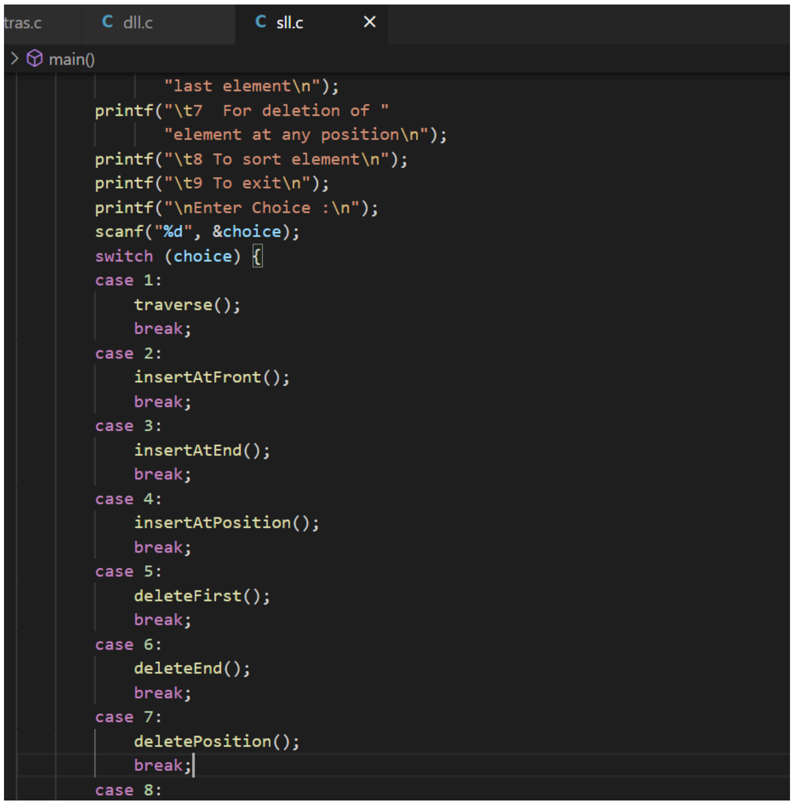
```
    while (i < pos - 1) {
        temp = temp->link;
        i++;
    }
    newnode->link = temp->link;
    temp->link = newnode;
}
void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->link;
        free(temp);
    }
}
void deleteEnd()
{
    struct node *temp, *prevnode;
    if (start == NULL)
        printf("\nList is Empty\n");
    else {
        temp = start;
        while (temp->link != 0) {
            prevnode = temp;
            temp = temp->link;
        }
        free(temp);
```



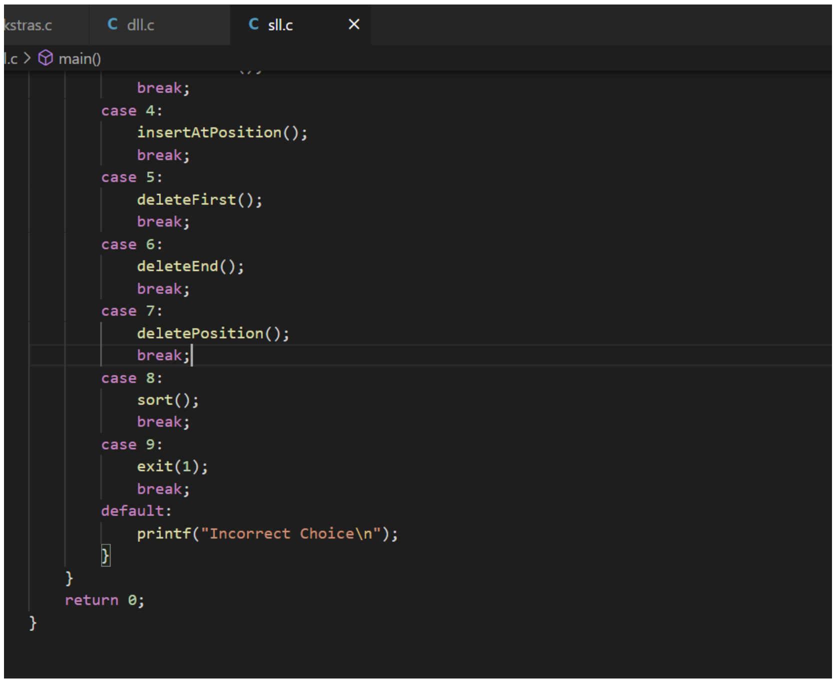
```c
        prevnode->link = 0;
    }
}
void deletePosition()
{
    struct node *temp, *position;
    int i = 1, pos;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        printf("\nEnter index : ");
        scanf("%d", &pos);
        position = malloc(sizeof(struct node));
        temp = start;
        while (i < pos - 1) {
            temp = temp->link;
            i++;
        }
        position = temp->link;
        temp->link = position->link;
        free(position);
    }
}
void sort()
{
    struct node* current = start;
    struct node* index = NULL;
    int temp;
    if (start == NULL) {
        return;
```



```c
    }
    else {
        while (current != NULL) {
            index = current->link;
            while (index != NULL) {
                if (current->info > index->info) {
                    temp = current->info;
                    current->info = index->info;
                    index->info = temp;
                }
                index = index->link;
            }
            current = current->link;
        }
    }
}
int main()
{
    int choice;
    while (1) {
        printf("\n\t1  To see list\n");
        printf("\t2  For insertion at"
               " starting\n");
        printf("\t3  For insertion at"
               " end\n");
        printf("\t4  For insertion at "
               "any position\n");
        printf("\t5  For deletion of "
               "first element\n");
        printf("\t6  For deletion of "
```

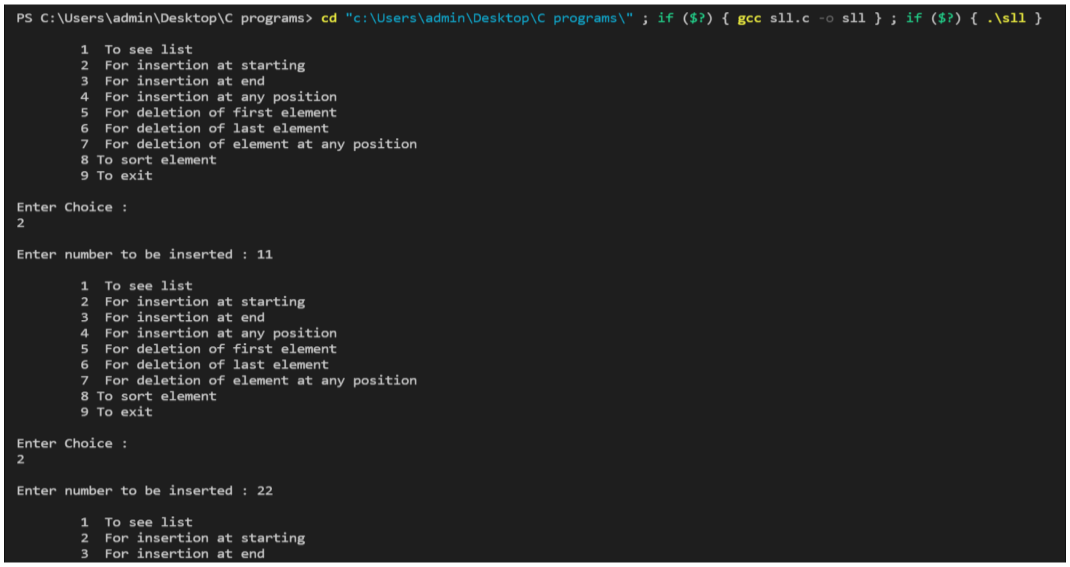


```c
            "last element\n");
    printf("\t7  For deletion of "
            "element at any position\n");
    printf("\t8 To sort element\n");
    printf("\t9 To exit\n");
    printf("\nEnter Choice :\n");
    scanf("%d", &choice);
    switch (choice) {
    case 1:
        traverse();
        break;
    case 2:
        insertAtFront();
        break;
    case 3:
        insertAtEnd();
        break;
    case 4:
        insertAtPosition();
        break;
    case 5:
        deleteFirst();
        break;
    case 6:
        deleteEnd();
        break;
    case 7:
        deletePosition();
        break;
    case 8:
```

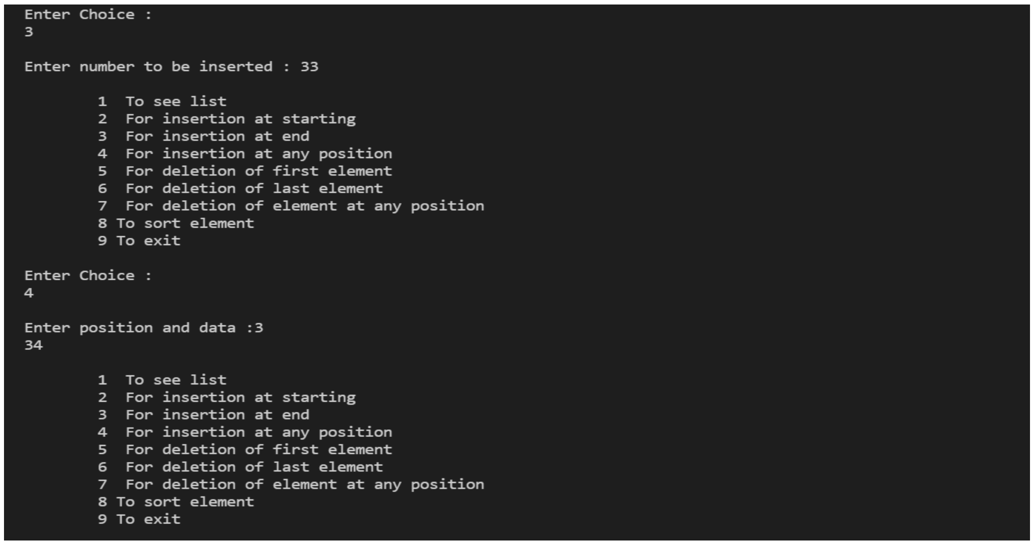


```c
            break;
        case 4:
            insertAtPosition();
            break;
        case 5:
            deleteFirst();
            break;
        case 6:
            deleteEnd();
            break;
        case 7:
            deletePosition();
            break;
        case 8:
            sort();
            break;
        case 9:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
    return 0;
}
```

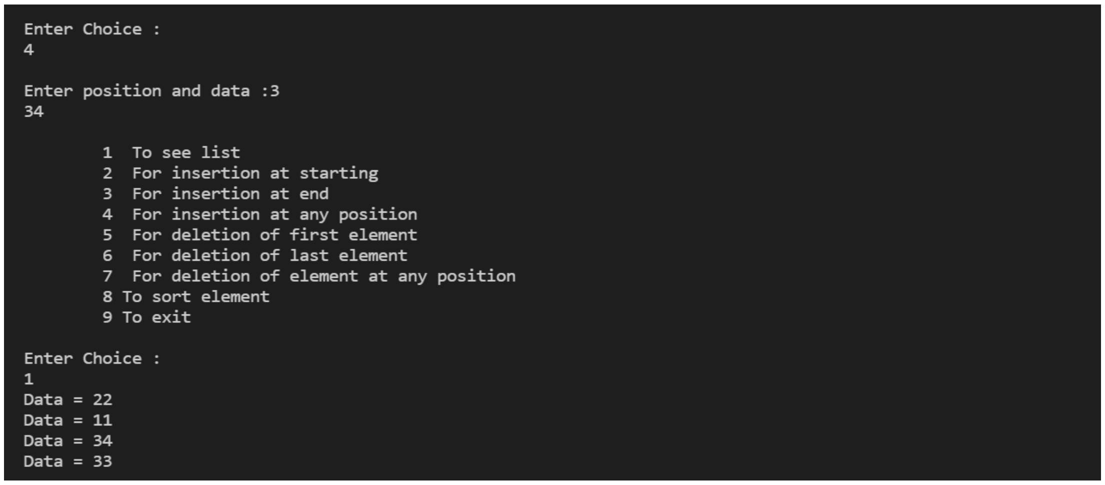
Output:



```
PS C:\Users\admin\Desktop\C programs> cd "c:\Users\admin\Desktop\C programs\" ; if ($?) { gcc sll.c -o sll } ; if ($?) { .\sll }

        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8 To sort element
        9 To exit

Enter Choice :
2

Enter number to be inserted : 11

        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8 To sort element
        9 To exit

Enter Choice :
2

Enter number to be inserted : 22

        1  To see list
        2  For insertion at starting
        3  For insertion at end
```



```
Enter Choice :
3

Enter number to be inserted : 33

        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8 To sort element
        9 To exit

Enter Choice :
4

Enter position and data :3
34

        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8 To sort element
        9 To exit
```



```
Enter Choice :
4

Enter position and data :3
34

        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8 To sort element
        9 To exit

Enter Choice :
1
Data = 22
Data = 11
Data = 34
Data = 33
```

Doubly Linked List:

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

Following are advantages/disadvantages of doubly linked list over singly linked list.
Advantages over singly linked list
1) A DLL can be traversed in both forward and backward direction.
2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
3) We can quickly insert a new node before a given node.
In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

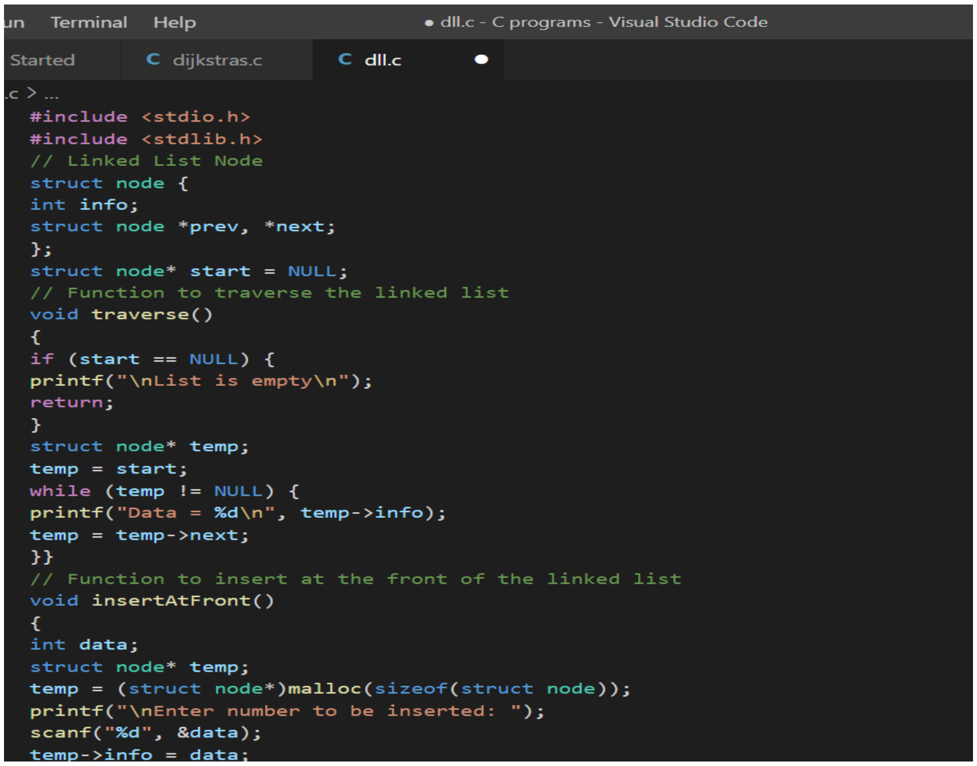
Disadvantages over singly linked list
1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though (See this and this).
2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Applications of Doubly linked list:-

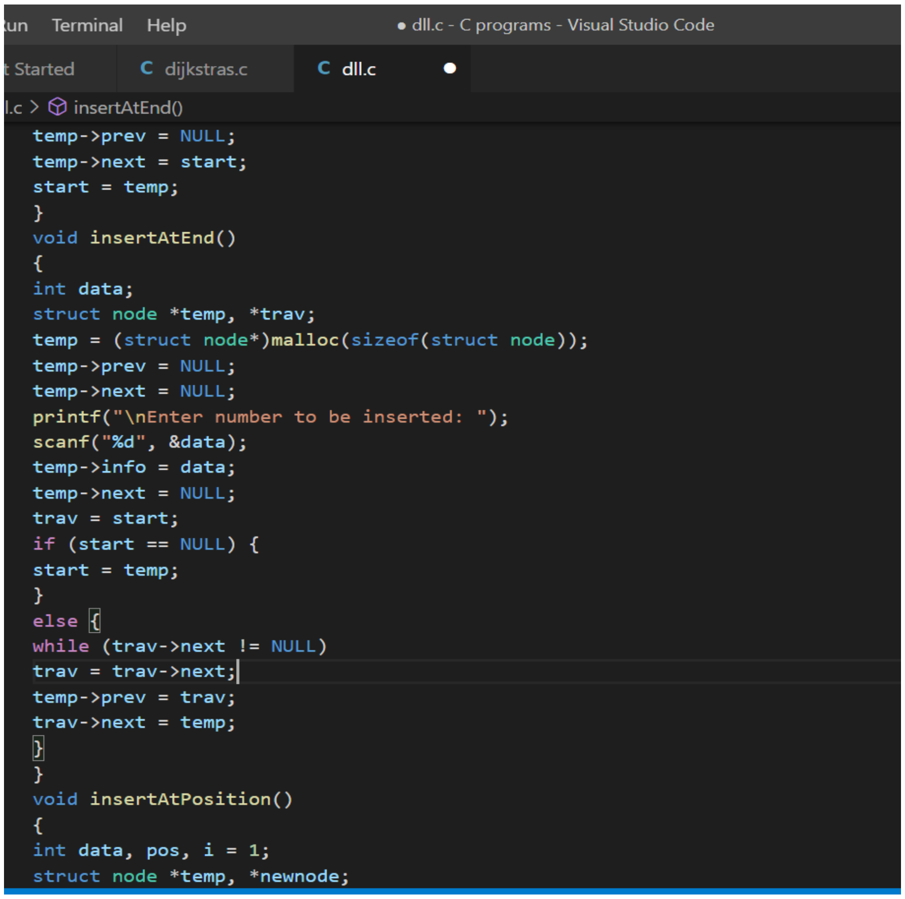
There are various application of doubly linked list in the real world. Some of them can be listed as:

- Doubly linked list can be used in navigation systems where both front and back navigation is required.
- It is used by browsers to implement backward and forward navigation of visited web pages i.e. **back** and **forward** button.
- It is also used by various application to implement **Undo** and **Redo** functionality.
- It can also be used to represent deck of cards in games.
- It is also used to represent various states of a game.

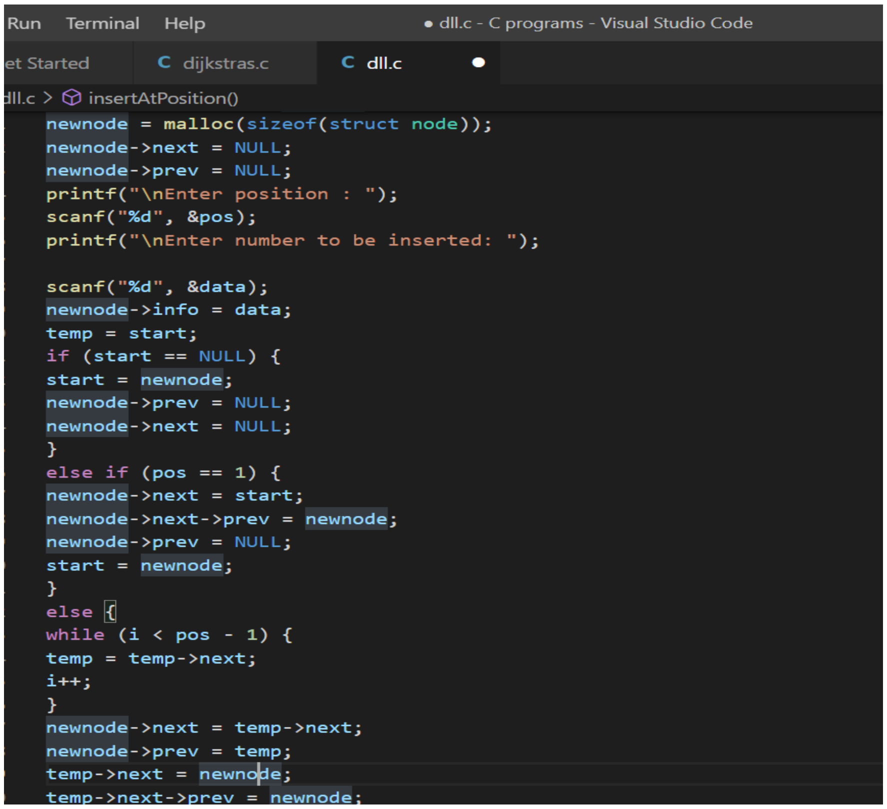
C program for Doubly Linked list:



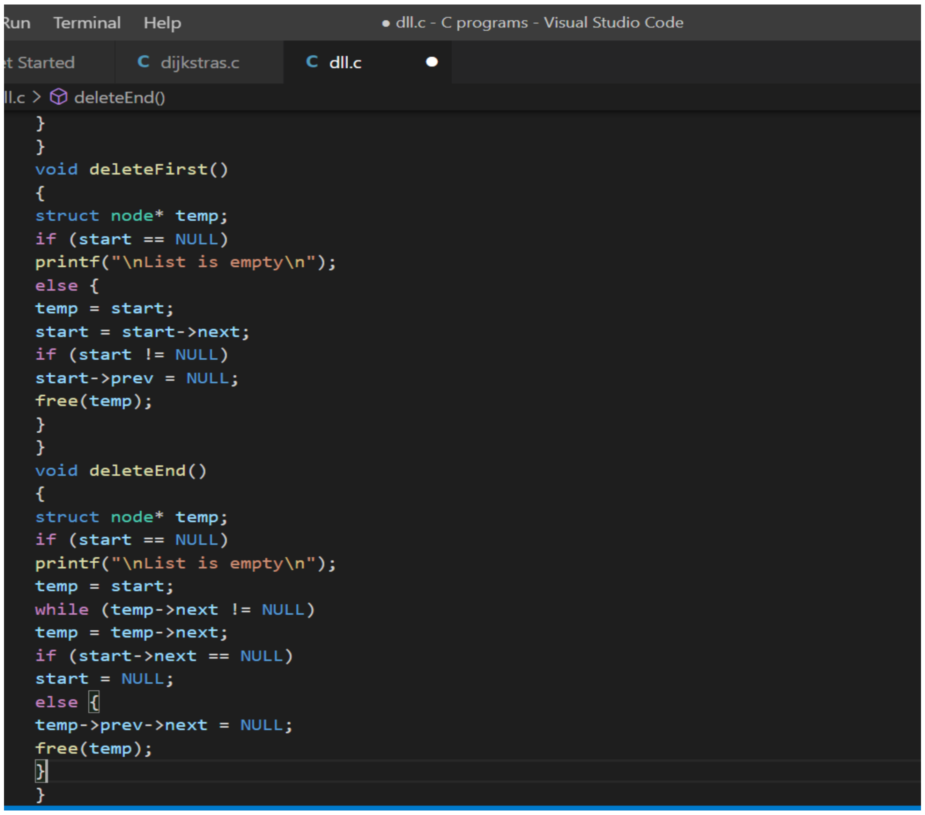
```c
#include <stdio.h>
#include <stdlib.h>
// Linked List Node
struct node {
int info;
struct node *prev, *next;
};
struct node* start = NULL;
// Function to traverse the linked list
void traverse()
{
if (start == NULL) {
printf("\nList is empty\n");
return;
}
struct node* temp;
temp = start;
while (temp != NULL) {
printf("Data = %d\n", temp->info);
temp = temp->next;
}}
// Function to insert at the front of the linked list
void insertAtFront()
{
int data;
struct node* temp;
temp = (struct node*)malloc(sizeof(struct node));
printf("\nEnter number to be inserted: ");
scanf("%d", &data);
temp->info = data;
```



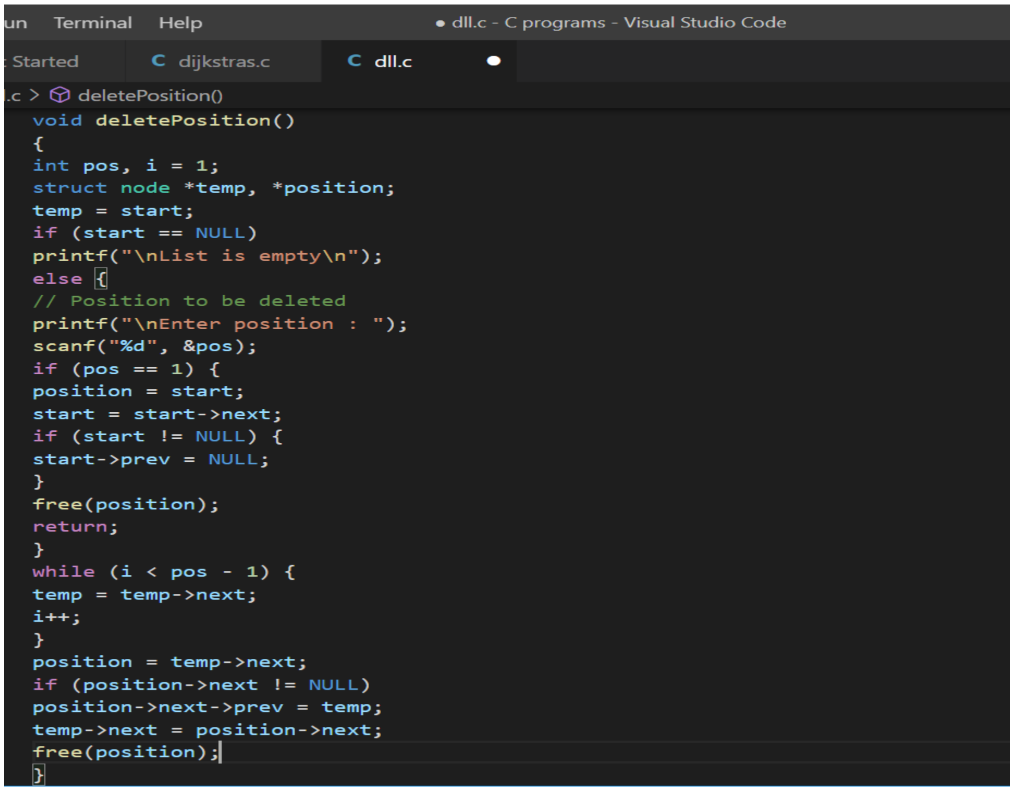
```c
temp->prev = NULL;
temp->next = start;
start = temp;
}
void insertAtEnd()
{
int data;
struct node *temp, *trav;
temp = (struct node*)malloc(sizeof(struct node));
temp->prev = NULL;
temp->next = NULL;
printf("\nEnter number to be inserted: ");
scanf("%d", &data);
temp->info = data;
temp->next = NULL;
trav = start;
if (start == NULL) {
start = temp;
}
else {
while (trav->next != NULL)
trav = trav->next;
temp->prev = trav;
trav->next = temp;
}
}
void insertAtPosition()
{
int data, pos, i = 1;
struct node *temp, *newnode;
```


Run Terminal Help
● dll.c - C programs - Visual Studio Code
et Started
dijkstras.c
dll.c
dll.c > insertAtPosition()

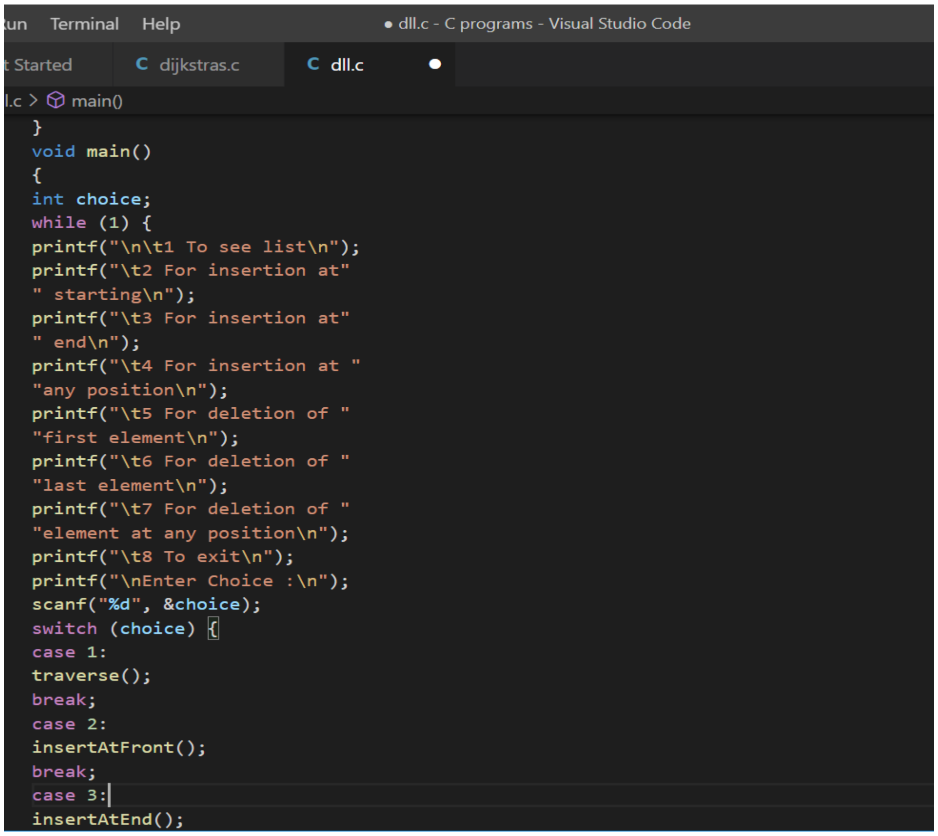
```
newnode = malloc(sizeof(struct node));
newnode->next = NULL;
newnode->prev = NULL;
printf("\nEnter position : ");
scanf("%d", &pos);
printf("\nEnter number to be inserted: ");

scanf("%d", &data);
newnode->info = data;
temp = start;
if (start == NULL) {
start = newnode;
newnode->prev = NULL;
newnode->next = NULL;
}
else if (pos == 1) {
newnode->next = start;
newnode->next->prev = newnode;
newnode->prev = NULL;
start = newnode;
}
else {
while (i < pos - 1) {
temp = temp->next;
i++;
}
newnode->next = temp->next;
newnode->prev = temp;
temp->next = newnode;
temp->next->prev = newnode;
```



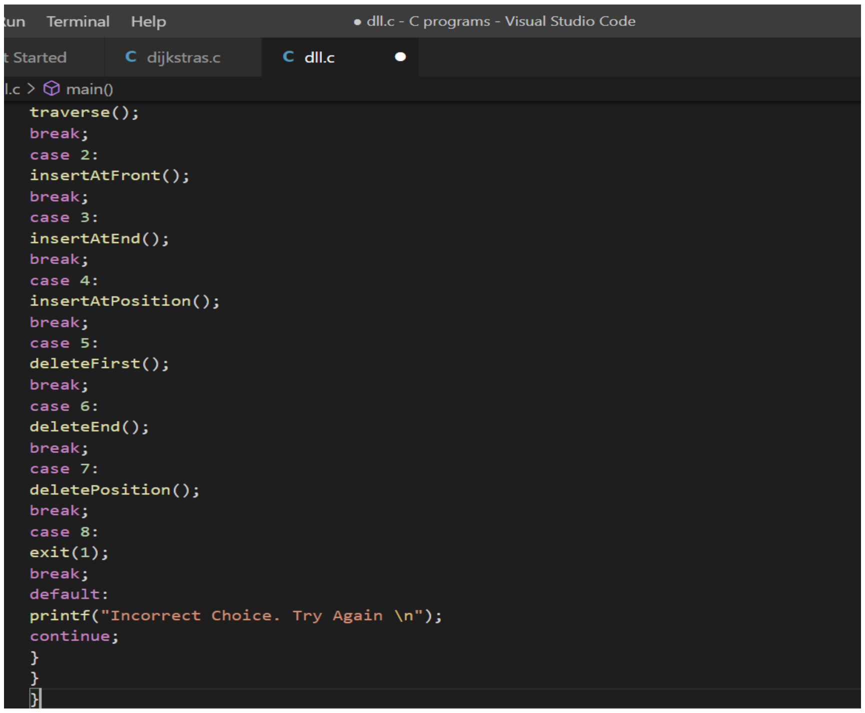
```
}
}
void deleteFirst()
{
struct node* temp;
if (start == NULL)
printf("\nList is empty\n");
else {
temp = start;
start = start->next;
if (start != NULL)
start->prev = NULL;
free(temp);
}
}
void deleteEnd()
{
struct node* temp;
if (start == NULL)
printf("\nList is empty\n");
temp = start;
while (temp->next != NULL)
temp = temp->next;
if (start->next == NULL)
start = NULL;
else {
temp->prev->next = NULL;
free(temp);
}
}
```



```c
void deletePosition()
{
int pos, i = 1;
struct node *temp, *position;
temp = start;
if (start == NULL)
printf("\nList is empty\n");
else {
// Position to be deleted
printf("\nEnter position : ");
scanf("%d", &pos);
if (pos == 1) {
position = start;
start = start->next;
if (start != NULL) {
start->prev = NULL;
}
free(position);
return;
}
while (i < pos - 1) {
temp = temp->next;
i++;
}
position = temp->next;
if (position->next != NULL)
position->next->prev = temp;
temp->next = position->next;
free(position);
}
```

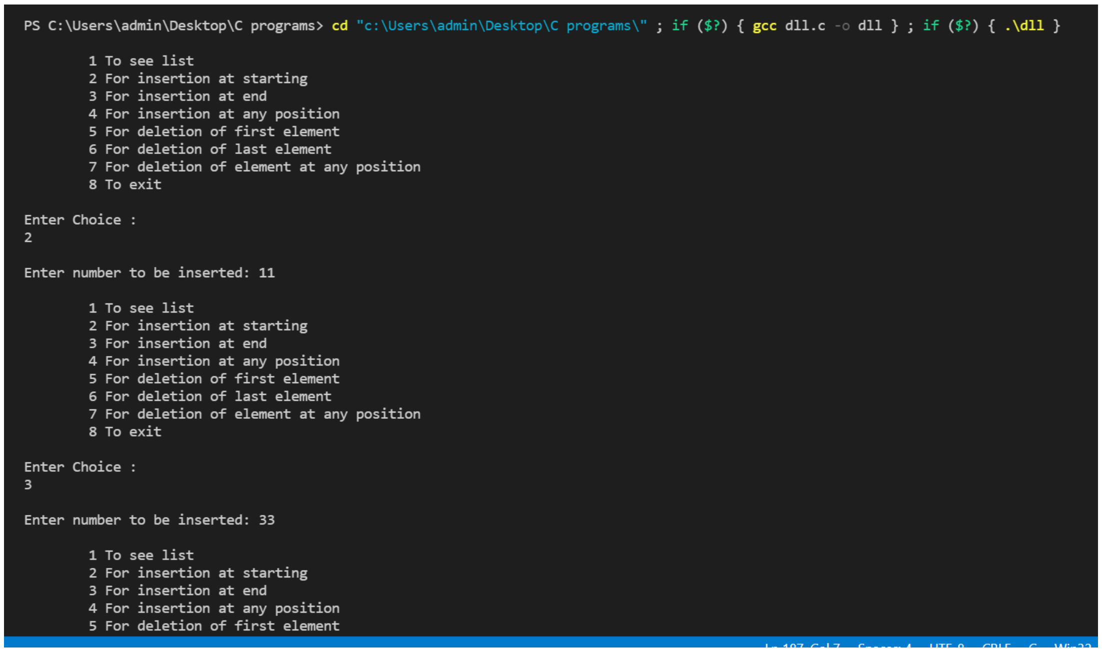


```c
}
void main()
{
int choice;
while (1) {
printf("\n\t1 To see list\n");
printf("\t2 For insertion at"
" starting\n");
printf("\t3 For insertion at"
" end\n");
printf("\t4 For insertion at "
"any position\n");
printf("\t5 For deletion of "
"first element\n");
printf("\t6 For deletion of "
"last element\n");
printf("\t7 For deletion of "
"element at any position\n");
printf("\t8 To exit\n");
printf("\nEnter Choice :\n");
scanf("%d", &choice);
switch (choice) {
case 1:
traverse();
break;
case 2:
insertAtFront();
break;
case 3:
insertAtEnd();
```

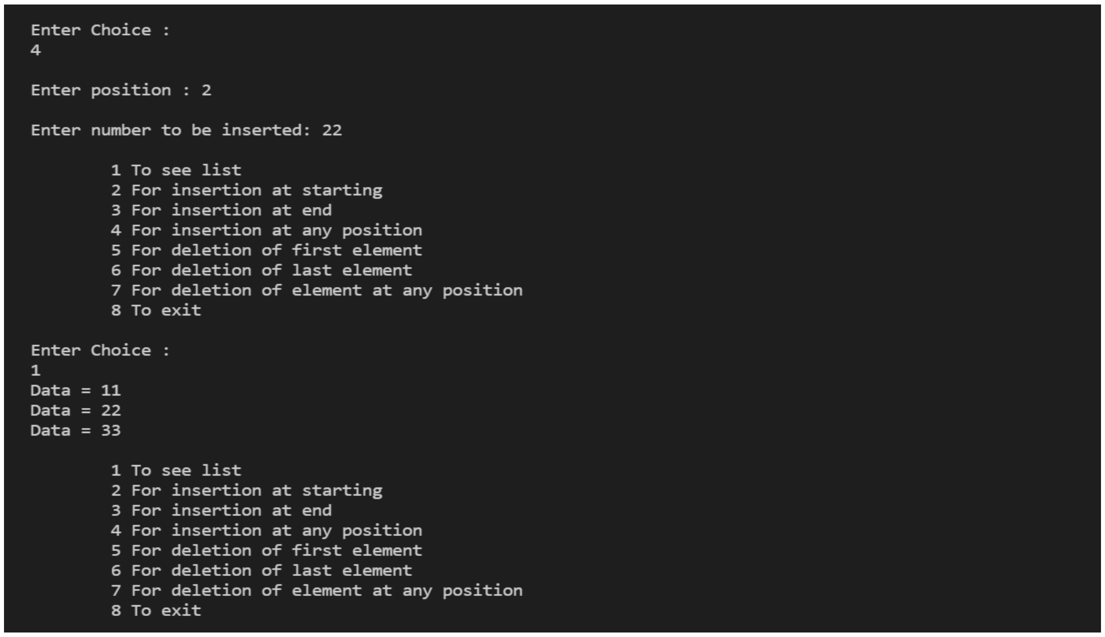


```c
traverse();
break;
case 2:
insertAtFront();
break;
case 3:
insertAtEnd();
break;
case 4:
insertAtPosition();
break;
case 5:
deleteFirst();
break;
case 6:
deleteEnd();
break;
case 7:
deletePosition();
break;
case 8:
exit(1);
break;
default:
printf("Incorrect Choice. Try Again \n");
continue;
}
}
}
```

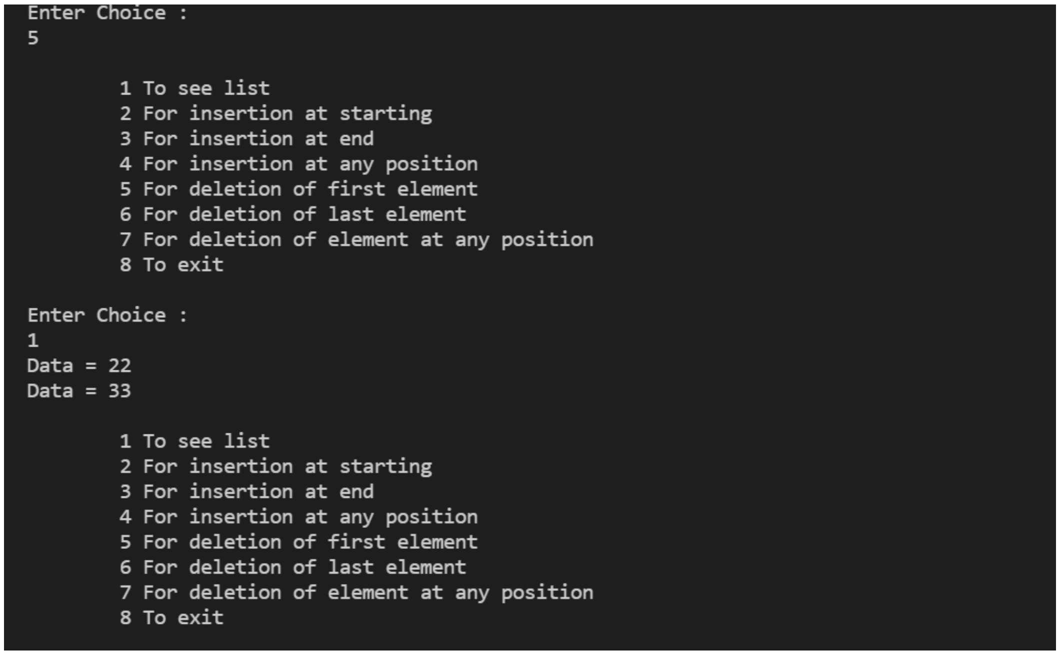
Output:



```
PS C:\Users\admin\Desktop\C programs> cd "c:\Users\admin\Desktop\C programs\" ; if ($?) { gcc dll.c -o dll } ; if ($?) { .\dll }

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit

Enter Choice :
2

Enter number to be inserted: 11

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit

Enter Choice :
3

Enter number to be inserted: 33

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
```



```
Enter Choice :
4

Enter position : 2

Enter number to be inserted: 22

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit

Enter Choice :
1
Data = 11
Data = 22
Data = 33

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit
```



```
Enter Choice :
5

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit

Enter Choice :
1
Data = 22
Data = 33

        1 To see list
        2 For insertion at starting
        3 For insertion at end
        4 For insertion at any position
        5 For deletion of first element
        6 For deletion of last element
        7 For deletion of element at any position
        8 To exit
```