



EXPERIMENT - 4

Data Structures

Aim

Create a linked list and perform following operations:

- I. Insert a new node at specified position.
- II. Delete of a node with a specified. position
- III. Reversal of that linked list.

Syeda Reeha Quasar

14114902719

3C7

EXPERIMENT – 3

AIM: Create a linked list and perform following operations:

- i. Traversal in the linked list
- ii. Insert a new node at specified position.
- iii. Delete of a node with a specified. position.

THEORY

A linked list is a linear data structure where each element is a separate object. Linked list elements are not stored at contiguous location; the elements are linked using pointers. Each node of a list is made up of two items - the data and a reference to the next node. The last node has a reference to null.

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

PROGRAM 1

Traversal Program

Source code:

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

struct Node
{
    int data;
    struct Node* next;
```

```
};
```

```
void printList(struct Node* n)
```

```
{  
    while (n != NULL)  
    {  
        printf(" %d ", n -> data);  
        n = n -> next;  
    }  
}
```

```
int main()
```

```
{  
    // traversing a singly linked list  
  
    // my information  
    printf("\n\n Name - Syeda Reeha Quasar \n Roll No. - 14114802719 \n Group - 3C7 \n\n");  
  
    // declaring and initiallizing the nodes as null  
    struct Node* head = NULL;  
    struct Node* second = NULL;  
    struct Node* third = NULL;  
    struct Node* fourth = NULL;  
    struct Node* fifth = NULL;  
  
    // allocating dynamic memory using malloc  
    head = (struct Node*)malloc(sizeof(struct Node));  
    second = (struct Node*)malloc(sizeof(struct Node));  
    third = (struct Node*)malloc(sizeof(struct Node));  
    fourth = (struct Node*)malloc(sizeof(struct Node));
```

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

```
// giving the data value in each node and assigning the next to another node in order to link
```

```
// head data value 1 and pointing to second
```

```
head -> data = 1;
```

```
head -> next = second;
```

```
// second data value 2 and pointing to third
```

```
second -> data = 2;
```

```
second -> next = third;
```

```
//third data value 3 and pointing to fourth
```

```
third -> data = 3;
```

```
third -> next = fourth;
```

```
//fourth data value 4 and pointing to fifth
```

```
fourth -> data = 4;
```

```
fourth -> next = fifth;
```

```
//fifth data value 5 and pointing to null i.e. is the end of the linked list
```

```
fifth -> data = 5;
```

```
fifth -> next = NULL;
```

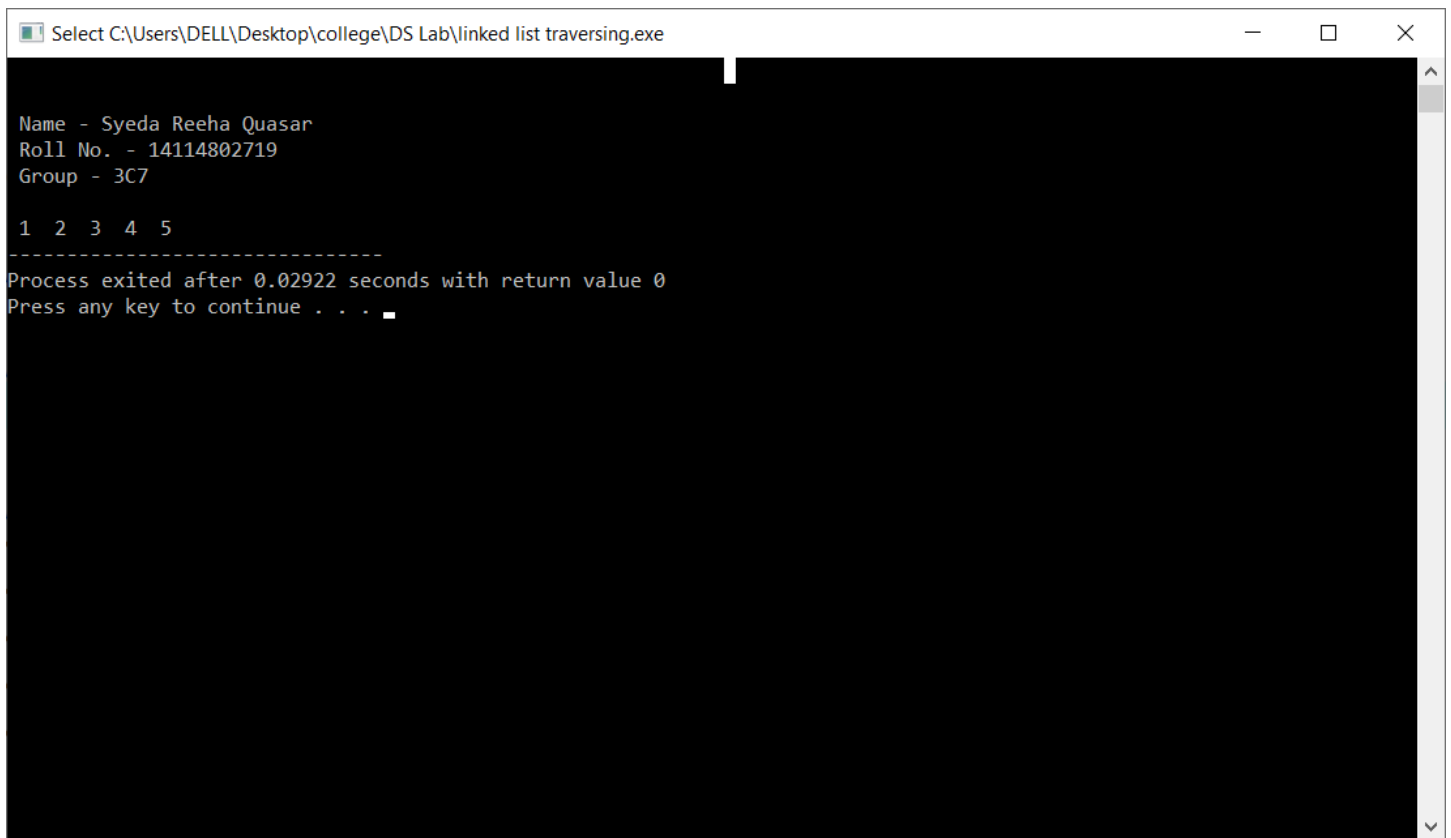
```
// printing the linked list (function is being called to do so)
```

```
printList(head);
```

```
return 0;
```

```
}
```

OUTPUT



The screenshot shows a Windows command prompt window with the title bar "Select C:\Users\DELL\Desktop\college\DS Lab\linked list traversing.exe". The window has a black background with white text. The output of the program is as follows:

```
Name - Syeda Reeha Quasar
Roll No. - 14114802719
Group - 3C7

1 2 3 4 5
-----
Process exited after 0.02922 seconds with return value 0
Press any key to continue . . .
```

INSERTION

Approach

You can add elements to either the beginning, middle or end of the linked list.

Add to the beginning

- Allocate memory for new node
- Store data
- Change next of new node to point to head
- Change head to point to recently created node

Add to the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

Add to the Middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

Source Code:

```
#include<stdio.h>

#include<stdlib.h>

struct Node{

    int data;
    struct Node* next;
};

void print(struct Node*n)
{
    while(n!=NULL)
    {
        printf(" %d",n->data);
        n=n->next;
    }
}

int main()
{
    // my information
    printf("\n\n Name - Syeda Reeha Quasar \n Roll No. - 14114802719 \n Group - 3C7 \n\n");

    // declaring and initiallizing the nodes as null
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;
    struct Node* fourth = NULL;
```

```
struct Node* fifth = NULL;
```

```
struct Node* newNode = NULL;
```

```
struct Node* startNode = NULL;
```

```
struct Node* midNode = NULL;
```

```
// allocating dynamic memory using malloc
```

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

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

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

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

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

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

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

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

```
// giving the data value in each node and assigning the next to another node in order to link
```

```
// head data value 1 and pointing to second
```

```
head->data = 1;
```

```
head->next = second;
```

```
// second data value 2 and pointing to third
```

```
second->data = 2;
```

```
second->next = third;
```

```
//third data value 3 and pointing to fourth
```

```
third->data = 3;
```

```
third->next = fourth;
```

```
//fourth data value 4 and pointing to fifth
```

```
fourth->data = 4;
```


fourth->next = fifth;

//fifth data value 5 and pointing to null i.e. is the end of the linked list

fifth->data = 5;

fifth->next = NULL;

// insertion in the middle

printf("\nEnter node to be inserted at the middle \n");

int n3,pos;

scanf("%d", &n3);

printf("Enter the position \n");

scanf("%d", &pos);

midNode->data =n3;

// temp node declaration and storing value of head in it.

struct Node *temp=head;

// travelling untill position in temp

for(int i=2;i<pos;i++)

{

if(temp->next!=NULL)

temp=temp->next;

}

// linking the midnode

midNode->next=temp->next;

temp->next=midNode;

// printing the mid node inseted list

printf("Updated link list \n");

```
print(head);  
printf("\n\n");
```

```
// insertion at the end  
printf("\nEnter node to be inserted at last\n");  
int num;  
scanf("%d", &num);
```

```
//linking  
newNode-> data =num;  
newNode->next = NULL;  
fifth-> next = newNode;
```

```
// printing new linked list with new node inserted at the end  
printf("Updated link list\n");  
print(head);  
printf("\n\n");
```

```
// insertion at the beginning  
printf("\nEnter node to be inserted at start\n");  
int n2;  
scanf("%d", &n2);
```

```
//linking  
startNode-> data =n2;  
startNode->next = head;
```

```
head = startNode;
```

```
// printing the new linked list with new node in the beginning
```

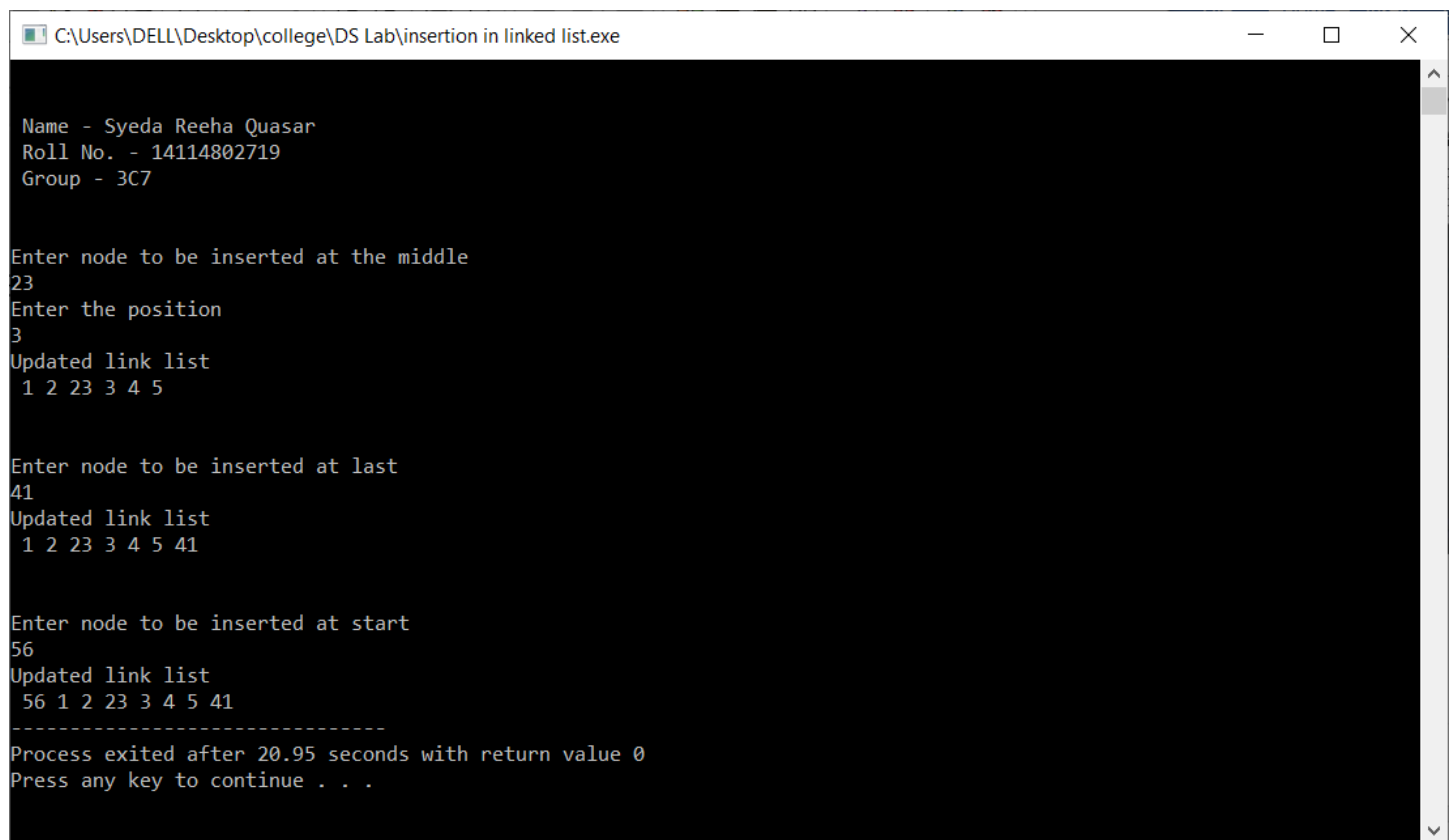
```
printf("Updated link list \n");
```

```
print(head);
```

```
return 0;
```

```
}
```

OUTPUT



```
C:\Users\DELL\Desktop\college\DS Lab\insertion in linked list.exe

Name - Syeda Reeha Quasar
Roll No. - 14114802719
Group - 3C7

Enter node to be inserted at the middle
23
Enter the position
3
Updated link list
1 2 23 3 4 5

Enter node to be inserted at last
41
Updated link list
1 2 23 3 4 5 41

Enter node to be inserted at start
56
Updated link list
56 1 2 23 3 4 5 41

-----
Process exited after 20.95 seconds with return value 0
Press any key to continue . . .
```

DELETION

Approach

You can delete either from the beginning, end or from a particular position.

Delete from beginning

- Point head to the second node

Delete from end

- Traverse to second last element
- Change its next pointer to null

Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node{

    int data;
    struct Node* next;
};
void print(struct Node*n)
{
    while(n!=NULL)
```

```
{
    printf(" %d",n->data);

    n=n->next;

}

}

int main()
{

    // my information

    printf("\n\n Name - Syeda Reeha Quasar \n Roll No. - 14114802719 \n Group - 3C7 \n\n");


    // declaring and initiallizing the nodes as null

    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;
    struct Node* fourth = NULL;
    struct Node* fifth = NULL;
    struct Node* temp = NULL;


    // allocating dynamic memory using malloc

    head=(struct Node*)malloc(sizeof(struct Node));
    second=(struct Node*)malloc(sizeof(struct Node));
    third=(struct Node*)malloc(sizeof(struct Node));
    fourth=(struct Node*)malloc(sizeof(struct Node));
    fifth=(struct Node*)malloc(sizeof(struct Node));
    temp =(struct Node*)malloc(sizeof(struct Node));


    // giving the data value in each node and assigning the next to another node in order to link
    // head data value 1 and pointing to second
    head->data = 1;
    head->next = second;
```

```
// second data value 2 and pointing to third
```

```
second->data = 2;
```

```
second->next = third;
```

```
//third data value 3 and pointing to fourth
```

```
third->data = 3;
```

```
third->next = fourth;
```

```
//fourth data value 4 and pointing to fifth
```

```
fourth->data = 4;
```

```
fourth->next = fifth;
```

```
//fifth data value 5 and pointing to null i.e. is the end of the linked list
```

```
fifth->data = 5;
```

```
fifth->next = NULL;
```

```
// printing the linked list
```

```
print(head);
```

```
int num;
```

```
//deletion of node at desired node
```

```
//taking value of node to be deleted
```

```
printf("\nEnter node to be deleted at any other position\n");
```

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

```
//deleting the node
```

```
struct Node *current = head;
```

```
while(current->next!=NULL)
```

```
{
```

```
        if(current->next->data==num)
        {
            temp=current->next;
            current->next=current->next->next;
            free(temp);
            break;
        }
        else
            current=current->next;
    }
```

// printing the desired linked list with deleted node

```
printf("\n\nUpdated link list \n\n");
```

```
print(head);
```

//deletion at beginning

```
temp=head;
```

```
head=head->next;
```

```
free(temp);
```

// printing the node after deleting first node

```
printf("\n\nUpdated link list after deletion at start\n\n");
```

```
print(head);
```

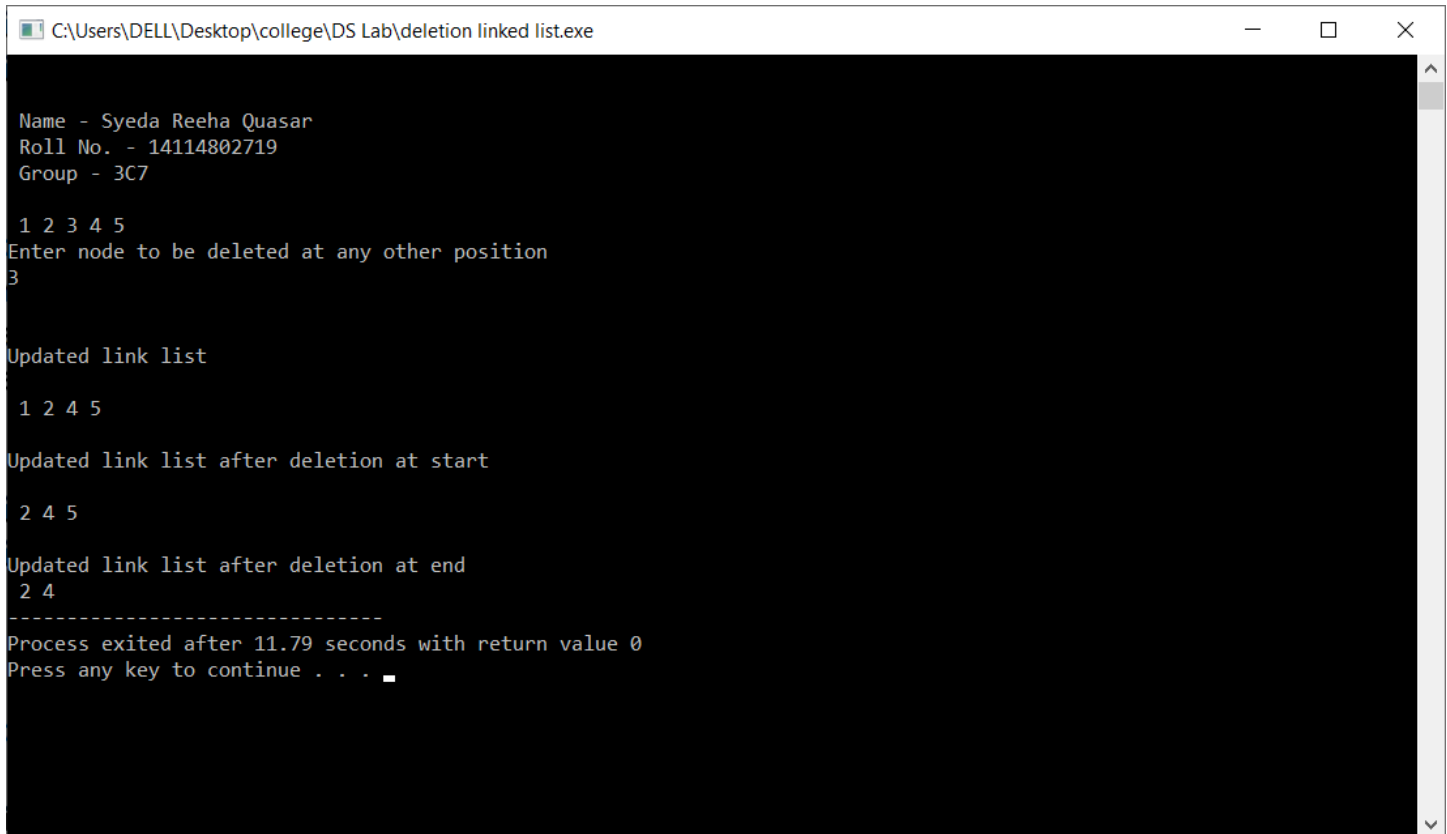
//deletion of the last node

```
while(current->next!=NULL)
{
    if(current->next->next==NULL)
    {
        temp=current->next;
        current->next=NULL;
        free(temp);
        break;
    }
    current=current->next;
}

// printing linked list after deleting end node
printf("\n\nUpdated link list after deletion at end \n");
print(head);

return 0;
}
```


OUTPUT



```
C:\Users\DELL\Desktop\college\DS Lab\deletion linked list.exe

Name - Syeda Reeha Quasar
Roll No. - 14114802719
Group - 3C7

1 2 3 4 5
Enter node to be deleted at any other position
3

Updated link list

1 2 4 5

Updated link list after deletion at start

2 4 5

Updated link list after deletion at end

2 4
-----
Process exited after 11.79 seconds with return value 0
Press any key to continue . . .
```

Viva Questions

Q1. What is singly linked list with example?

Ans1.

A linked list is a dynamic data structure where each element (called a node) is made up of two items: the data and a reference (or pointer), which points to the next node. A linked list is a collection of nodes where each node is connected to the next node through a pointer. The first node is called a head and if the list is empty then the value of head is NULL.

For a real-world analogy of linked list, you can think of conga line, a special kind of dance in which people line up behind each other with hands on the shoulders of the person in front. Each dancer represents a data element, while their hands serve as the pointers or links to the next element.

Q2. What is a disadvantage of singly linked list?

Ans2.

1. It requires more space as pointers are also stored with information.
2. Different amount of time is required to access each element.
3. If we have to go to a particular element then we have to go through all those elements that come before that element.
4. We cannot traverse it from last & only from the beginning.
5. It is not easy to sort the elements stored in the linear linked list.

Q3. What is the complexity of searching for a particular element in a singly linked list?

Ans3.

To search element, you have to traverse through the entire list, hence complexity is $O(n)$.

Q4. Why do we use linked list?

Ans4.

Linked lists are linear data structures that hold data in individual objects called nodes. Linked lists are often used because of their efficient insertion and deletion. They can be used to implement stacks, queues, and other abstract data types.

Q5. Why are linked lists better than arrays?

Ans-5.

Link list better than arrays because of the linked list elements can be added to it indefinitely, while an array will eventually get filled or have to be resized (a costly operation that isn't always possible). Elements are also easily removed from a linked list whereas removing elements from an array leaves empty spaces that are a waste of computer memory.

Q6. What are the operations of singly linked list?

Ans-6.

Basic Operations on Linked List

- **Traversal:** To traverse all the nodes one after another.
- **Insertion:** To add a node at the given position.
- **Deletion:** To delete a node.
- **Searching:** To search an element(s) by value.
- **Updating:** To update a node

Q7. What is the costly operation in singly linked list?

Ans-7.

Operations following random access of elements are least efficient and hence, are costly operations in singly linked list.