Batch: D3	**Roll No.: 16010123294**
Experiment / assignment / tutorial No. 3	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of basic Linked List – creation, insertion, deletion, traversal, searching an element

Objective: To understand the advantage of linked list over other structures like arrays in implementing the general linear list

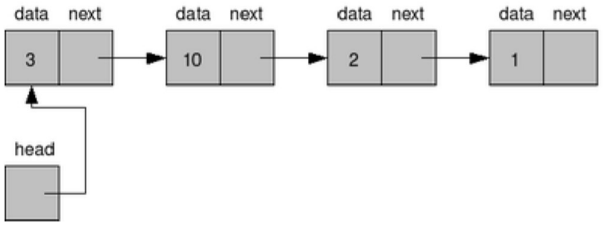
Expected Outcome of Experiment:

CO	Outcome
1	Comprehend the different data structures used in problem solving

Books/ Journals/ Websites referred:

Introduction:

A linear list is a list where each element has a unique successor. There are four common operations associated with a linear list: insertion, deletion, retrieval, and traversal. Linear list can be divided into two categories: general list and restricted list. In general list the data can be inserted or deleted without any restriction whereas in restricted list there is restrictions for these operations. Linked list and arrays are commonly used to implement general linear list. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.



A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This Structure that contains elements and pointers to the next structure is called a Node.

Related Theory: -

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers

Linked List ADT:

Data Structure

Node Structure:

- **Data**: Stores the actual value (e.g., an integer, string, etc.).
- **Link/Pointer**: Stores the address of the next node in the list.

Linked List Structure:

- **Head/Start**: A pointer that references the first node in the list. If the list is empty, the head is NULL.

Basic Operations

- **Create List**:
  - Initialize the head of the list to NULL.
- **Insert Element**:
  - **At the Start**: Insert a new node at the beginning of the list.
  - **At the End**: Insert a new node at the end of the list.
  - **At a Specific Position**: Insert a new node at a specified position in the list.
- **Delete Element**:
  - **From the Start**: Remove the first node from the list.
  - **From the End**: Remove the last node from the list.
  - **From a Specific Position**: Remove the node at a specified position in the list.
- **Traverse List**:
  - Visit each node in the list, starting from the head, and perform an operation (e.g., printing the node's data).
- **Search Element**:
  - Find a node containing a specific value. If found, return its position or a reference to the node; otherwise, indicate that the element is not present.
- **Check if Empty**:

- Deermine whether the linked list is empty (i.e., whether the head is NULL).

- **Get Length**:
  - Calculate and return the number of nodes in the list.

Key Characteristics

- **Dynamic Size**: The size of a linked list can grow or shrink dynamically, unlike arrays with fixed sizes.
- **Non-contiguous Memory Allocation**: Nodes are not stored contiguously in memory; they are linked using pointers.
- **Efficient Insertion/Deletion**: Insertion and deletion operations can be performed efficiently, particularly at the beginning or end of the list.

Types of Linked Lists

- **Singly Linked List**: Each node has a single pointer pointing to the next node.
- **Doubly Linked List**: Each node has two pointers: one pointing to the next node and one to the previous node.
- **Circular Linked List**: The last node points back to the first node, forming a circle.

Algorithm for creation, insertion, deletion, traversal and searching an element in linked list:

Initialize:

- Define a struct node containing info (data) and link (pointer to the next node).
- Initialize start pointer as NULL.

Create Node Function:

- Input: data (integer).
- Create a new node with the given data.
- If the list is empty, set start to this new node.
- Otherwise, traverse to the end of the list and link the new node.

Traverse Function:

- Initialize a temporary pointer to start.
- If the list is empty, print "The linked list is empty."
- Otherwise, traverse the list and print each node's data followed by -> until reaching NULL.

Search Function:

- Input: n (element to search).
- Initialize a temporary pointer to start.
- If the list is empty, print "The linked list is empty."
- Traverse the list:
    - If the element is found, print "Element Found" and return.
    - If the element is not found, print "Not Found."

Insert Function:

- Input: data (element to insert).
- Prompt the user to choose the position for insertion: Start, End, or Specific Position.
- Based on the user's choice:
    - **Start**: Insert the new node at the beginning.
    - **End**: Insert the new node at the end.
    - **Specific Position**: Insert the new node at the specified position.

Delete Node Function:

- Prompt the user to choose the position for deletion: Start, End, or Specific Position.
- Based on the user's choice:
    - **Start**: Delete the first node.
    - **End**: Delete the last node.

- **Specific Position**: Delete the node at the specified position.

Main Loop:

- Display the menu with options:
  - Create a list
  - Display list
  - Search for an element
  - Insert an element
  - Delete an element
  - Exit

Prompt the user to enter their choice.

Execute the corresponding function based on the choice.

Exit Program

Program source code:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int info;
struct node*link;
};

struct node* start =NULL;


void createNode(int data){
struct node *q,*tmp;
tmp= (struct node*) malloc(sizeof(struct node));
tmp->info=data;
tmp->link=NULL;
if(start==NULL)
{
        start=tmp;
}
else
{
        q=start;
        while(q->link!=NULL)
                {
                q=q->link;
```

```
                    }
            q->link=tmp;
}
}

void traverse()
  {
     struct node* tmp = start;
     if (tmp == NULL)
  {
     printf("The linked list is empty.\n");
     return;
  }
  while (tmp)
  {
     printf("%d -> ", tmp->info);
     tmp = tmp->link;
  }
  printf("NULL\n");
}
void search(int n)
{
  struct node* tmp = start;
     if (tmp == NULL)
  {
     printf("The linked list is empty.\n");
```

```
    return;
  }
  while(tmp)
  {
    if( n == tmp->info)
      {
      printf("Element Found\n");
      return;
      }
      tmp = tmp->link;
  }
  if(tmp==NULL)
    {
      printf("Not Found\n");
    }

}
void insert(int data) {
  int choice, pos, i;
  struct node* tmp, * q;
  tmp = (struct node*)malloc(sizeof(struct node));
  tmp->info = data;
  tmp->link = NULL;

  printf("Insert at:\n Start (1)\n End (2)\n Specific Position (3)\n");
  scanf("%d", &choice);
```

```
switch (choice) {
case 1:
   tmp->link = start;
   start = tmp;
   break;
case 2:
   if (start == NULL) {
      start = tmp;
   }
   else {
      q = start;
      while (q->link != NULL) {
         q = q->link;
      }
      q->link = tmp;
   }
   break;
case 3:
   printf("Enter position to insert (starting from 0): ");
   scanf("%d", &pos);
   if (pos == 0) {
      tmp->link = start;
      start = tmp;
   }
   else {
```

```
            q = start;
            for (i = 0; i < pos - 1 && q != NULL; i++) {
               q = q->link;
            }
            if (q == NULL) {
               printf("Position out of range.\n");
               free(tmp);
            }
            else {
               tmp->link = q->link;
               q->link = tmp;
            }
         }
         break;
      default:
         printf("Invalid choice.\n");
         free(tmp);
         break;
      }
}
void deleteNode() {
   int choice, pos, i;
   struct node* tmp, * q;

   printf("Delete from: 1. Start 2. End 3. Specific Position\n");
   scanf("%d", &choice);
```

```
switch (choice) {
case 1:
   if (start == NULL) {
      printf("List is already empty.\n");
   }
   else {
      tmp = start;
      start = start->link;
      free(tmp);
      printf("Node deleted from start.\n");
   }
   break;
case 2:
   if (start == NULL) {
      printf("List is already empty.\n");
   }
   else if (start->link == NULL) {
      free(start);
      start = NULL;
      printf("Last node deleted.\n");
   }
   else {
      q = start;
      while (q->link->link != NULL) {
         q = q->link;
```

```
        }
        tmp = q->link;
        q->link = NULL;
        free(tmp);
        printf("Node deleted from end.\n");
      }
      break;
    case 3:
      printf("Enter position to delete (starting from 0): ");
      scanf("%d", &pos);
      if (pos == 0) {
        if (start != NULL) {
          tmp = start;
          start = start->link;
          free(tmp);
          printf("Node deleted from position %d.\n", pos);
        }
        else {
          printf("List is already empty.\n");
        }
      }
      else {
        q = start;
        for (i = 0; i < pos - 1 && q != NULL; i++) {
          q = q->link;
        }
```

```
            if (q == NULL || q->link == NULL) {
                printf("Position out of range.\n");
            }
            else {
                tmp = q->link;
                q->link = tmp->link;
                free(tmp);
                printf("Node deleted from position %d.\n", pos);
            }
        }
        break;
    default:
        printf("Invalid choice.\n");
        break;
    }
}




int main(){
    int choice,n,i,data;
    while(1){

    printf("Operations :\n");
    printf("1.Create a List \n");
```

```
printf("2.Display List \n");
printf("3.Search an element\n");
printf("4.Insert an Element\n");
printf("5.Delete an Element\n");
printf("6.Exit\n");
printf("\nEnter Your Choice : ");
scanf("%d",&choice);
switch (choice) {
case 1:
   printf("\nEnter number of Nodes: ");
   scanf("%d",&n);
   for(i=0;i<n;i++){
      printf("Enter Data to be Entered\n");
      scanf("%d",&data);
      createNode(data);
   }
   break;
case 2:
   traverse();
   break;
case 3:
   printf("Enter Element to be Search :\n");
   scanf("%d",&n);
   search(n);
   break;
case 4:
```

```
        printf("Enter data to be entered:\n");
        scanf("%d",&data);
        insert(data);
        break;
    case 5:
          deleteNode();
          break;
    case 6:
          printf("Exit the program.\n");
            return 0;
    default:
            printf("Invalid choice.\n");
            break;
}
   }


return 0;
}
```

Output Screenshots:

```
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 1

Enter number of Nodes: 5
Enter Data to be Entered
1
Enter Data to be Entered
2
Enter Data to be Entered
3
Enter Data to be Entered
4
Enter Data to be Entered
5
```

```
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 2
1 -> 2 -> 3 -> 4 -> 5 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit
```

```
Enter Your Choice : 3
Enter Element to be Search :
2
Element Found
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit
```

```
Element Found
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 4
Enter data to be entered:
6
Insert at:
 Start (1)
 End (2)
 Specific Position (3)
2
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 2
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
```

```
Enter Your Choice : 2
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 4
Enter data to be entered:
0
Insert at:
 Start (1)
 End (2)
 Specific Position (3)
1
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 2
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
```

```
Enter Your Choice : 2
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 5
Delete from: 1. Start 2. End 3. Specific Positi
3
Enter position to delete (starting from 0): 3
Node deleted from position 3.
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 2
0 -> 1 -> 2 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
```

```
0 -> 1 -> 2 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 5
Delete from: 1. Start 2. End 3. Specific Position
1
Node deleted from start.
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 2
1 -> 2 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit
```

```
Enter Your Choice : 2
1 -> 2 -> 4 -> 5 -> 6 -> NULL
Operations :
1.Create a List
2.Display List
3.Search an element
4.Insert an Element
5.Delete an Element
6.Exit

Enter Your Choice : 6
Exit the program.

Process returned 0 (0x0)   execution time : 90.023 s
Press any key to continue.
```

Conclusion:-

Learned to use singly linkedlist and made a menu driven program for the same . Made functions for the creation display insertion deletion and also exit functions.

Post lab questions:

1. Write the differences between linked list and linear array

Structure:

- **Array**: Fixed-size, contiguous block of memory.
- **Linked List**: Dynamic-size, elements are stored in nodes that may be scattered in memory, each node points to the next.

Size:

- **Array**: Size is defined at creation and cannot be changed.
- **Linked List**: Can grow or shrink dynamically as elements are added or removed.

Access Time:

- **Array**: O(1) time complexity for accessing elements by index.
- **Linked List**: O(n) time complexity for accessing elements, as traversal is required.

Memory Usage:

- **Array**: Requires contiguous memory allocation.
- **Linked List**: Requires extra memory for storing pointers but doesn't need contiguous memory.

Insertion/Deletion:

- **Array**: Insertion/deletion is expensive (O(n)) as elements need to be shifted.
- **Linked List**: Insertion/deletion is more efficient (O(1)) if the position is known, but searching takes O(n).

Flexibility:

- **Array**: Less flexible due to fixed size.
- **Linked List**: More flexible as size can be adjusted dynamically.

2. Name some applications which uses linked list.

Operating Systems: Process scheduling and memory management.

Web Browsers: Implementing forward and backward navigation in history.

Music Players: Managing playlists where songs are linked.

Undo Functionality: In text editors to track changes.

Dynamic Memory Allocation: Managing free blocks in memory.

Graphs: Representing adjacency lists.

Implementing Stacks/Queues: When dynamic size is required.