Batch: A1	**Roll No.: 1601023012**
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

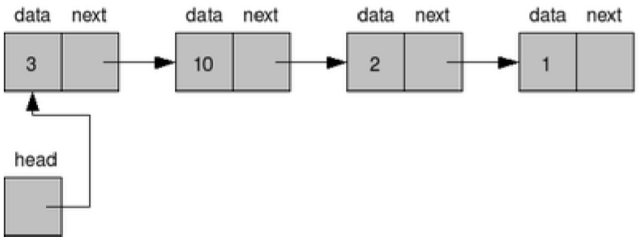
Objective: To understand the advantage of linked list over other structures like arrays in implementing the general linear list

Expected Outcome of Experiment:

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

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:
Linked List is an Abstract Data Type (ADT) that holds a collection of Nodes, the nodes can be accessed in a sequential way. Linked List doesn't provide a random access to a Node.

Value definition:
Abstract Typedef linked_list/node<<int number, struct linked_list *next>>
Condition: None

Operator Definition:

1. Abstract void beginsert<< >>
   Pre-condition: none
   Post-condition: New node of linked list are linked with it's next node; and tail as ptr thus, successively creating circular linked lists. Takes head node as the argument
2. Abstract void lastinsert<< >>
   Pre-condition: none
   Post-condition: Node of linked list are linked with it's tail as ptr thus and tail to its head

3. Abstract linked_list/node begin_delete<< >>
   Pre-condition: None
   Post-condition: The function then deletes the node aptly at the beginning.
4. Abstract linked_list/node last_delete<< >>
   Pre-condition: None
   Post-condition: The function then deletes the node aptly at the ending.
5. Abstract void display<< >>
   Pre-condition: The next node for tail node must point to head
   Post-condition: All nodes of the linked list is printed from head to tail; demonstrating that the linked list previously created is indeed, circualr linked
6. Abstract linked_list/node searching<<node *head>>
   Pre-condition: None
   Post-condition: Requests the user to enter the value of the data stored at a particular node which is to be searched. The function then searches the linked list to find the data. Returns the index of the node where the data is stored.

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

1. Initialization
   Define a Node structure with two members
   Initialize the head of the list to NULL

2. Create a New Node
   Allocate memory for a new node.
   Set the node's data to the given value.
   Initialize the node's next pointer to NULL.
   Return the new node.

3. Insert a Node
   Create a new node with the given data.
   If the list is empty or the new node's data is less than the head's data: Insert the new node at the beginning and update the head. Otherwise, traverse the list to find the correct position to insert the new node (maintaining sorted order).
   Insert the new node in its correct position.

4. Delete a Node
   If the list is empty, do nothing.
   If the head node's data matches the given data: Update the head to the next node and free the old head. Otherwise, traverse the list to find the node to be deleted.
   If the node is found, adjust the previous node's next pointer to skip the node to be deleted.
   Free the node to be deleted.

5. Search for a Node
   Traverse the list.

If a node with the given data is found, return 1
If the end of the list is reached without finding the data, return 0

6. Display the List
   Traverse the list.
   Print each node's data followed by a space.

7. Main Function
   Continuously prompt the user for a choice of operation:
   Perform the corresponding operation based on the user's choice.

Program source code:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

   int data;

   struct Node *next;

};

struct Node *head = NULL;

struct Node *newNode(int data) {

   struct Node *node = (struct Node *)malloc(sizeof(struct Node));

   node->data = data;

   node->next = NULL;

   return node;

}

void insert(int data) {

   struct Node *temp = newNode(data);

   if (head == NULL || data < head->data) {

      temp->next = head;

      head = temp;

      return;
```

```
    }
    struct Node *current = head;
    while (current->next != NULL && current->next->data < data) {
        current = current->next;
    }
    temp->next = current->next;
    current->next = temp;
}
void deleteNode(int data) {
    struct Node *temp = head;
    struct Node *prev = NULL;
    if (head != NULL && head->data == data) {
        head = head->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != data) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    prev->next = temp->next;
    free(temp);
}
int search(int data) {
    struct Node *temp = head;
```

```
    while (temp != NULL) {
        if (temp->data == data) {
            return 1;
        }
        temp = temp->next;
    }
    return 0;
}
void displayList() {
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int choice, data;
    while (1) {
        printf("Make a Choice:\n1) Insert\n2) Delete\n3) Search\n4) Display\n5) Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                insert(data);
```

```
            break;
        case 2:
            printf("Enter the data to delete: ");
            scanf("%d", &data);
            deleteNode(data);
            break;
        case 3:
            printf("Enter the data to search: ");
            scanf("%d", &data);
            if (search(data)) {
                printf("Data found in the list\n");
            } else {
                printf("Data not found in the list\n");
            }
            break;
        case 4:
            displayList();
            break;
        case 5:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice\n");
    }
}
return 0;
```

}

Output Screenshots:

```
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
1
Enter the data to insert: 23
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
1
Enter the data to insert: 75
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
1
Enter the data to insert: 567
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
1
Enter the data to insert: 678
```

```
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
1
Enter the data to insert: 12
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
3
Enter the data to search: 12
Data found in the list
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
2
Enter the data to delete: 23
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
3
Enter the data to search: 23
Data not found in the list
```

```
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
4
12 75 567 678
Make a Choice:
1) Insert
2) Delete
3) Search
4) Display
5) Exit
5
Exiting...


=== Code Execution Successful ===
```

Concusion:-

We have successfully completed this experiment and learned how to implement a basic linked list and wrote a program in which we created an ordered linked list having features 1) Insert 2) Delete 3) Search 4) Print 5) Exit

Post lab questions:

1. Write the differences between linked list and linear array

Linear Array	Linked List
An array is a collection of elementsof a similar data type.	A linked list is a collection of objects known as a node where nodeconsists of two parts, i.e., data and address.
Array works with a static memory. Here static memory means that the memory size is fixed and cannot be changed at the run time.	The Linked list works with dynamic memory. Here, dynamic memory means that the memory size canbe changed at the run time according to our requirements.
Array takes more time while performing any operation like insertion, deletion, etc.	Linked list takes less time while performing any operation like insertion, deletion, etc.
In the case of an array, memory is allocated at compile-time.	In the case of a linked list, memoryis allocated at run time.
Memory utilization is inefficient inan array	Memory utilization is efficient whenit comes to linked lists

2. Name some applications which uses linked list.

The Applications are as follows:

1. Implementation of stacks and queues.
2. Implementation of graphs: Adjacency list representation of graphs is most popular which is uses a linked list to store adjacent vertices.
3. Dynamic memory allocation: We use a linked list of free blocks.
4. Maintaining a directory of names.
5. Image viewer – Previous and next images are linked, hence can be accessed by next and previous button.
6. Useful for implementation of a queue. We can maintain a pointer to the last inserted node and the front can always be obtained as next of last