**Batch: C3 Roll No.: 16010123217**

**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 |

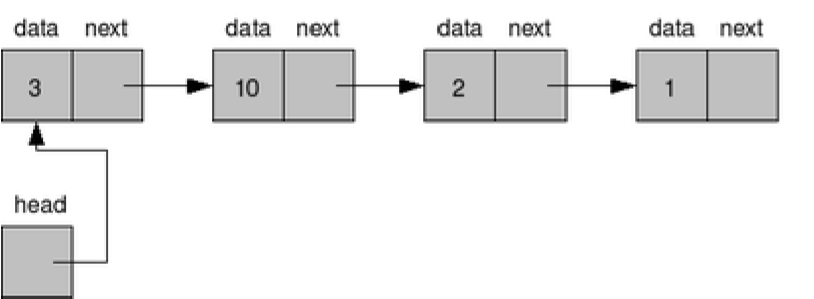
**Books/ Journals/ Websites referred:**

Lecture notes by snm mam

<https://www.geeksforgeeks.org/applications-of-linked-list-data-structure/>

**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:**

Struct NodeType{

ElementType Element;

struct NodeType \*Next;

}

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

1. Algorithm LLType CreateLinkedList()

//This Algorithm creates and returns an empty Linked List, pointed by a pointer -head

{

createNode(head);

head=NULL;

}

2. Algorithm LLType Insert(LLType Head, NodeType NewNode)

Case 1: insertion before the head

Newnode->next=Head;

Head=Newnode ;

Case 2: insertion after the last

Struct nodeType \*temp;

Temp=head;

While(temp->next!=NULL)

Temp=temp->next;

Temp->next = Newnode;

}

if (LL==Null) //inserting the first element

LL=Newnode; exit(0)

temp=LL, prev=null

while((newnode->data > temp->data) && temp!=Null)

{ prev=temp;

temp=temp->next;

}

if(temp==Null) //insertion in the end

Prev->next=Newnode;

if( temp=LL && newnode->data < temp->data)

{ Newnode ->next=temp; LL=Newnode }

Else // general case

Newnode->next =temp;

prev->next = Newnode

3. Algorithm LLType Insert(LLType Head, NodeType NewNode)

Case 3: insertion before some node (Key)

struct NodeType \*temp;

Temp=Head; Prev = Null;

While(temp->data!=key && temp!=Null)

prev=temp; temp=temp->next;

If(temp==Null) “Key not found”

Else

newnode->next=temp;

Prev->next=temp;

Case 4: insertion after some node(Key)

Struct nodeType \*temp;

While(temp->data!=key && temp!=Null)

temp=temp->next;

If(temp==Null) “Key not found”

Else

Newnode->next =temp->next;

Temp->next = Newnode;

}

3. Algorithm ElementType Delete(LLType Head, ElementType ele)

//This algorithm returns ElementType ele if it exists in the List, an error message otherwise. Temp and current are the a temproary nodes used in the delete process.

{ if (Head==NULL)

Print “Underflow”

exit;

Else

{0. search for element

1. element doesn’t exist in list

2. deletion in unsorted list

3. deletion in sorted list

}

4. Algorithm NodeType Search(LLType Head, ElementType Key)

//This algorithm returns NodeType node which contains the ‘keyvalue” being searched.

{ if (Head==NULL)

Print “element doesn’t exist”

exit;

Else

Transverse a pointer on linked list

If (Head->value == key)

Print Head;

}

**Program source code:**

*//16010123217*

#include <stdio.h>

#include <stdlib.h>

*// Define the structure for a node*

struct Node {

    int data;

    struct Node\* next;

};

void createList(struct Node\*\* head);

void insertAtStart(struct Node\*\* head, int data);

void insertAtEnd(struct Node\*\* head, int data);

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

void insertAfterNode(struct Node\*\* head, int key, int data);

void deleteAtStart(struct Node\*\* head);

void deleteAtEnd(struct Node\*\* head);

void deleteSpecificNode(struct Node\*\* head, int key);

void traverseList(struct Node\* head);

void searchElement(struct Node\* head, int key);

void menu();

int main() {

    struct Node\* head = NULL;

    int choice, data, key;

    do {

        menu();

        printf("\nEnter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1:

                createList(&head);

                break;

            case 2:

                printf("Enter the element to insert at start: ");

                scanf("%d", &data);

                insertAtStart(&head, data);

                break;

            case 3:

                printf("Enter the element to insert at end: ");

                scanf("%d", &data);

                insertAtEnd(&head, data);

                break;

            case 4:

                printf("Enter the element to insert: ");

                scanf("%d", &data);

                printf("Enter the node before which to insert: ");

                scanf("%d", &key);

                insertBeforeNode(&head, key, data);

                break;

            case 5:

                printf("Enter the element to insert: ");

                scanf("%d", &data);

                printf("Enter the node after which to insert: ");

                scanf("%d", &key);

                insertAfterNode(&head, key, data);

                break;

            case 6:

                deleteAtStart(&head);

                break;

            case 7:

                deleteAtEnd(&head);

                break;

            case 8:

                printf("Enter the element to delete: ");

                scanf("%d", &key);

                deleteSpecificNode(&head, key);

                break;

            case 9:

                traverseList(head);

                break;

            case 10:

                printf("Enter the element to search: ");

                scanf("%d", &key);

                searchElement(head, key);

                break;

            case 11:

                printf("Exiting program.\n");

                break;

            default:

                printf("Invalid choice! Please try again.\n");

                break;

        }

    } while (choice != 11);

    return 0;

}

void menu() {

    printf("\n--- Linked List Operations ---\n");

    printf("1. Create Linked List\n");

    printf("2. Insert at Start\n");

    printf("3. Insert at End\n");

    printf("4. Insert Before a Node\n");

    printf("5. Insert After a Node\n");

    printf("6. Delete at Start\n");

    printf("7. Delete at End\n");

    printf("8. Delete Specific Node\n");

    printf("9. Traverse List\n");

    printf("10. Search Element\n");

    printf("11. Exit\n");

}

void createList(struct Node\*\* head) {

    int n, data;

    struct Node\* temp;

    printf("Enter the number of elements: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        printf("Enter element %d: ", i + 1);

        scanf("%d", &data);

        if (\*head == NULL) {

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

            (\*head)->data = data;

            (\*head)->next = NULL;

            temp = \*head;

        } else {

            temp->next = (struct Node\*)malloc(sizeof(struct Node));

            temp = temp->next;

            temp->data = data;

            temp->next = NULL;

        }

    }

    printf("Linked list created successfully.\n");

}

void insertAtStart(struct Node\*\* head, int data) {

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

    newNode->data = data;

    newNode->next = \*head;

    \*head = newNode;

    printf("Element inserted at start.\n");

}

void insertAtEnd(struct Node\*\* head, int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        struct Node\* temp = \*head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    printf("Element inserted at end.\n");

}

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

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

    newNode->data = data;

    if (\*head == NULL) {

        printf("List is empty. Cannot insert before node.\n");

        return;

    }

    if ((\*head)->data == key) {

        newNode->next = \*head;

        \*head = newNode;

        printf("Element inserted before node with key %d.\n", key);

        return;

    }

    struct Node\* temp = \*head;

    while (temp->next != NULL && temp->next->data != key) {

        temp = temp->next;

    }

    if (temp->next == NULL) {

        printf("Node with key %d not found.\n", key);

    } else {

        newNode->next = temp->next;

        temp->next = newNode;

        printf("Element inserted before node with key %d.\n", key);

    }

}

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

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

    newNode->data = data;

    struct Node\* temp = \*head;

    while (temp != NULL && temp->data != key) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Node with key %d not found.\n", key);

    } else {

        newNode->next = temp->next;

        temp->next = newNode;

        printf("Element inserted after node with key %d.\n", key);

    }

}

void deleteAtStart(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty. Cannot delete.\n");

        return;

    }

    struct Node\* temp = \*head;

    \*head = (\*head)->next;

    free(temp);

    printf("First node deleted.\n");

}

void deleteAtEnd(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty. Cannot delete.\n");

        return;

    }

    struct Node\* temp = \*head;

    if ((\*head)->next == NULL) {

        \*head = NULL;

        free(temp);

        printf("Last node deleted.\n");

        return;

    }

    while (temp->next->next != NULL) {

        temp = temp->next;

    }

    free(temp->next);

    temp->next = NULL;

    printf("Last node deleted.\n");

}

void deleteSpecificNode(struct Node\*\* head, int key) {

    if (\*head == NULL) {

        printf("List is empty. Cannot delete.\n");

        return;

    }

    struct Node\* temp = \*head;

    if ((\*head)->data == key) {

        \*head = (\*head)->next;

        free(temp);

        printf("Node with key %d deleted.\n", key);

        return;

    }

    while (temp->next != NULL && temp->next->data != key) {

        temp = temp->next;

    }

    if (temp->next == NULL) {

        printf("Node with key %d not found.\n", key);

    } else {

        struct Node\* toDelete = temp->next;

        temp->next = temp->next->next;

        free(toDelete);

        printf("Node with key %d deleted.\n", key);

    }

}

void traverseList(struct Node\* head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = head;

    printf("Linked list elements: ");

    while (temp != NULL) {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

void searchElement(struct Node\* head, int key) {

    struct Node\* temp = head;

    int position = 1;

    while (temp != NULL) {

        if (temp->data == key) {

            printf("Element %d found at position %d.\n", key, position);

            return;

        }

        temp = temp->next;

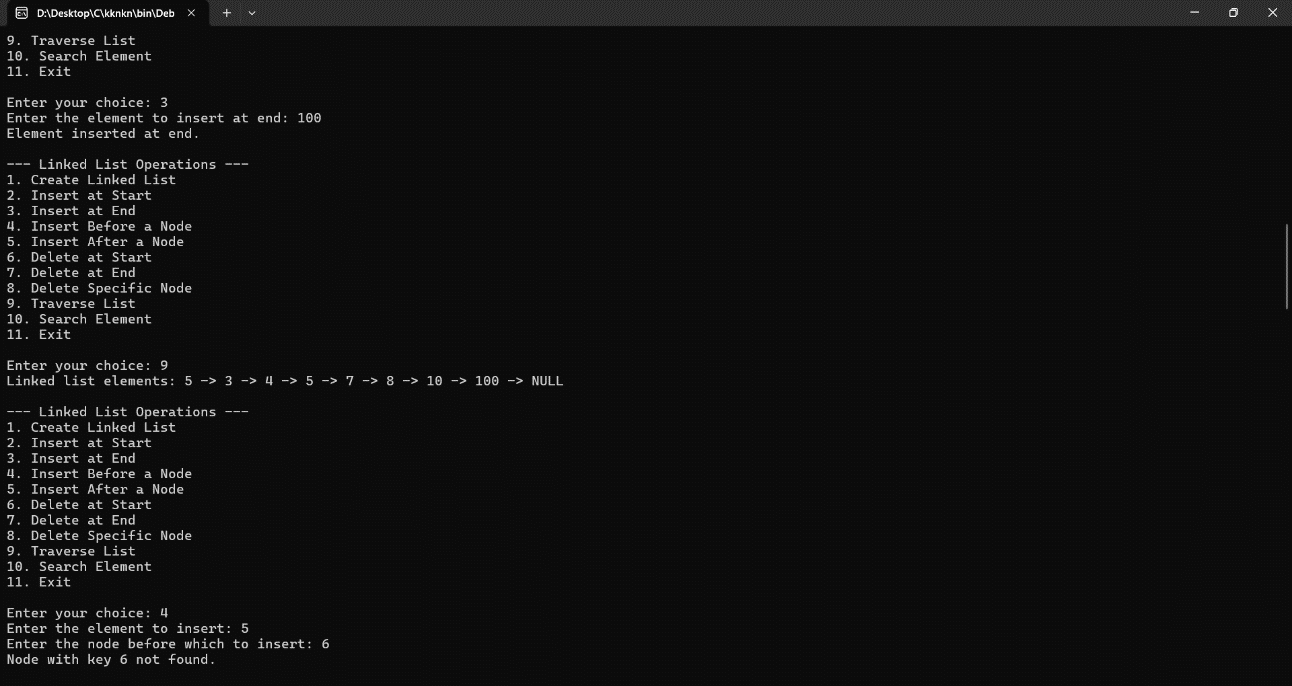
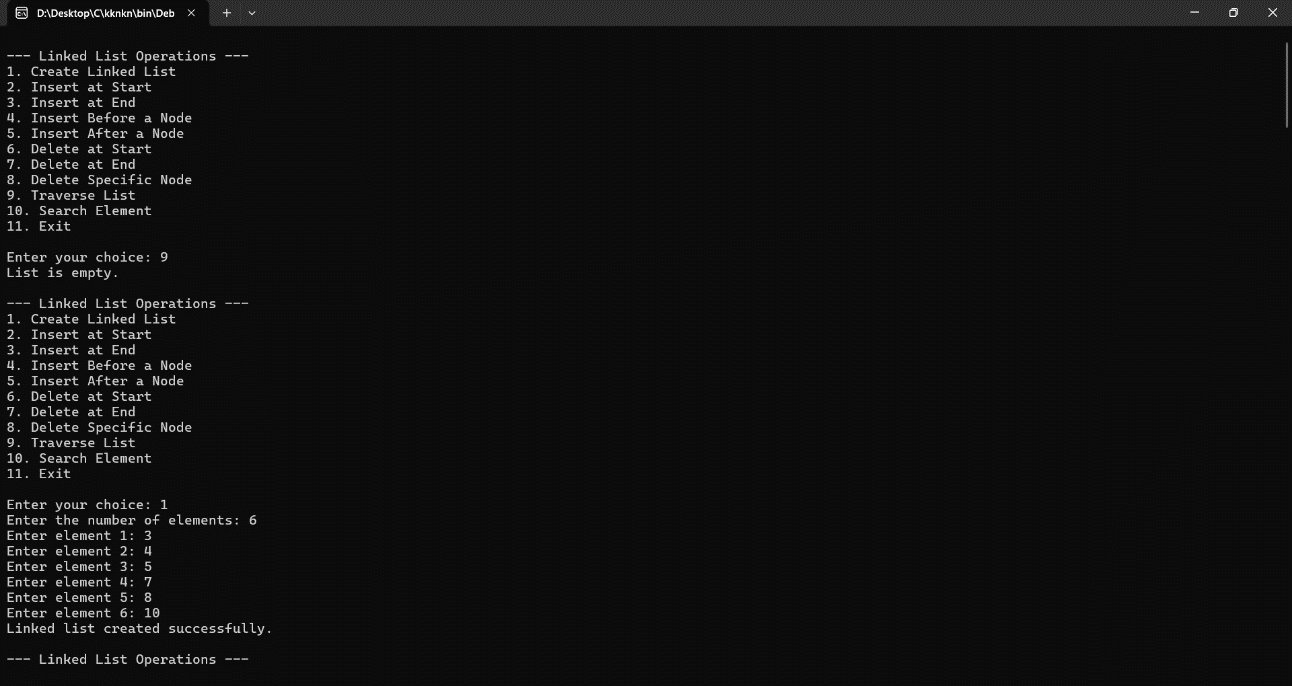
        position++;

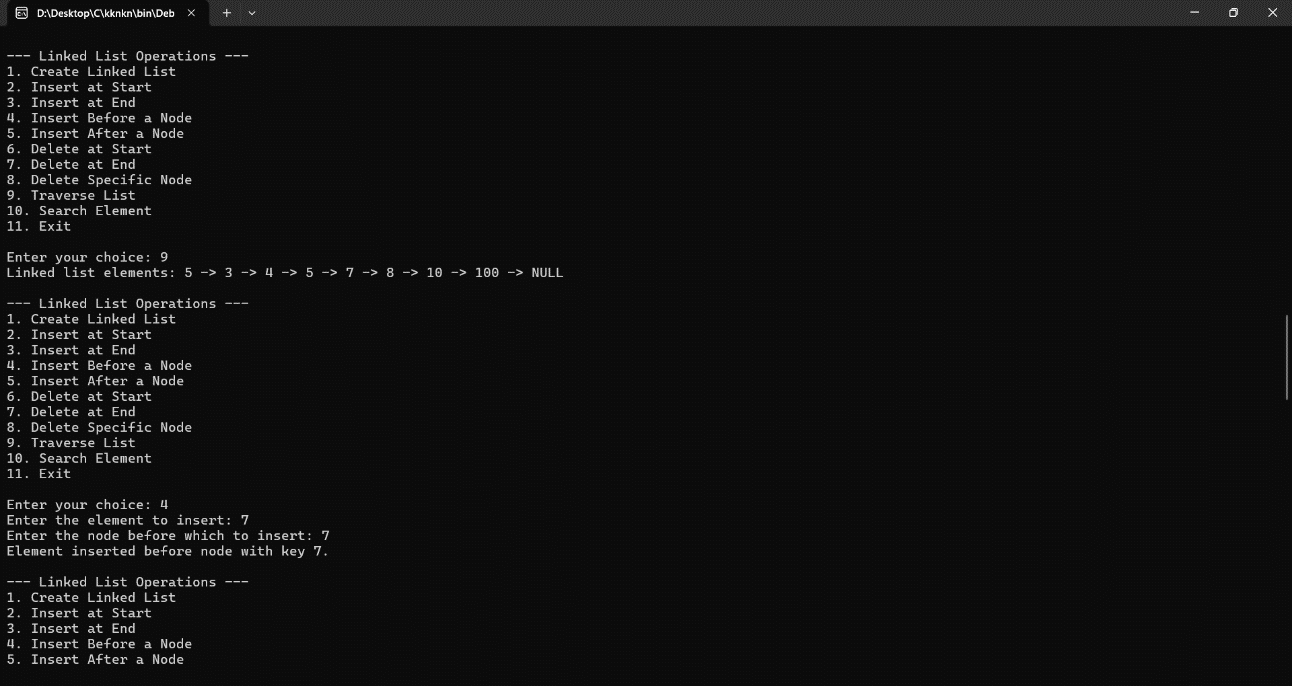
    }

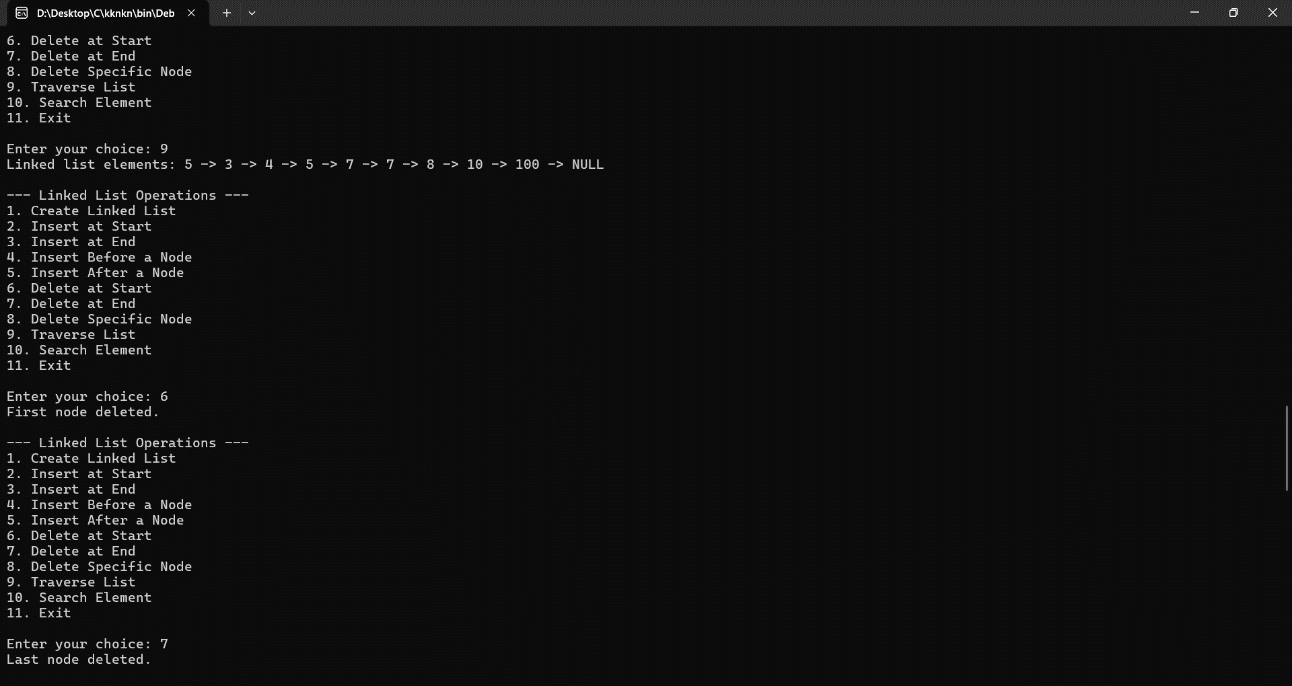
    printf("Element %d not found in the list.\n", key);

}

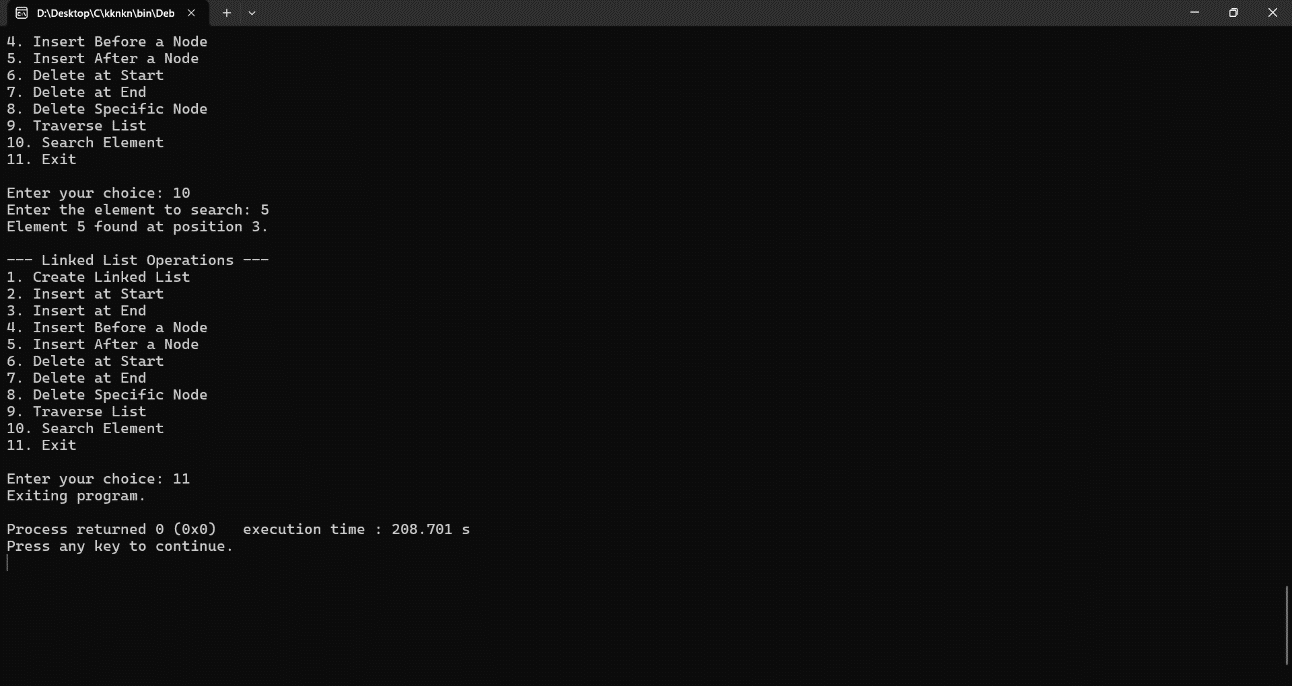
**Output Screenshots:**

****

****

****

****

****

**Conclusion:-**

**In this experiment we learnt about linked list and it’s uses in day to day life.**

**Also we discovered a new faster alternative to array in some case.**

**Post lab questions:**

1. Write the differences between linked list and linear array

Ans.

|  |  |
| --- | --- |
| Linked List | Linear Array |
| Linked lists are non-contiguous | Arrays are contguous |
| Uses more memory | Uses less memory |
| Dynamic memory allocation | Fixed in size |
| Memory is allocated at run time | Memory is allocated at compile time |
| To access an element we need to transverse the whole link list | Elements can be accessed easily. |
| Insertion and deletion operation is faster | Insertion and deletion takes time. |

2. Name some applications which uses linked list.

Ans.

Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.

File Systems- File systems use linked lists to represent the hierarchical structure of directories, where each directory or file is represented as a node in the list.

GPS navigation systems- Linked lists can be used to store and manage a list of locations and routes, allowing users to easily navigate to their desired destination.

Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.

Robotics- Linked lists can be used to implement control systems for robots, allowing them to navigate and interact with their environment.