Batch: B2	Roll No.: 16010124107
Experiment / assignment / tutorial No. 2	
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:

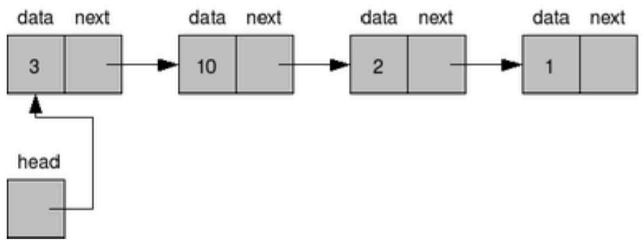
CO	Outcome
1	Comprehend the different data structures used in problem solving

Books/ Journals/ Websites referred:
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

https://ds1-iiith.vlabs.ac.in/exp/poly-arithmetic/index.html

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 are 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

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

1. Start
2. Create a struct for Linked list’s node
3. Create the first node called head = nullptr
4. Create function to append to list
5. Allocate memory and link new nodes
6. Create a function called insertAt to insert value at position pos
7. Determine if the position is valid, then run a loop to find last position pointed to
8. Rearrange pointers
9. Create a function to delete elements from a linked list

10. Check if the list is empty
11. Linearly look for the value.
12. When found, make the previous pointer skip this and point to next element
13. Store the skipped value in a temporary variable, after rearranging the pointer, delete this
14. Create a function to search by value.
15. Repeat step 11. If found, print result. If temp==null, that means the list does not contain that value, print result.
16. Create function to traverse the list.
17. Repeat step 10 and 11. Print all numbers in step 11.
18. Create a menu driven program in main() that calls these functions as and when required.
19. Stop.

Program source code:

```
#include <iostream>

using namespace std;


struct Node{

int data;

Node* next;

};


Node* head = NULL;

void append(int v)

{

   Node* newnode = new Node();

   newnode -> data = v;

   newnode -> next = NULL;


   if(head==NULL)
```

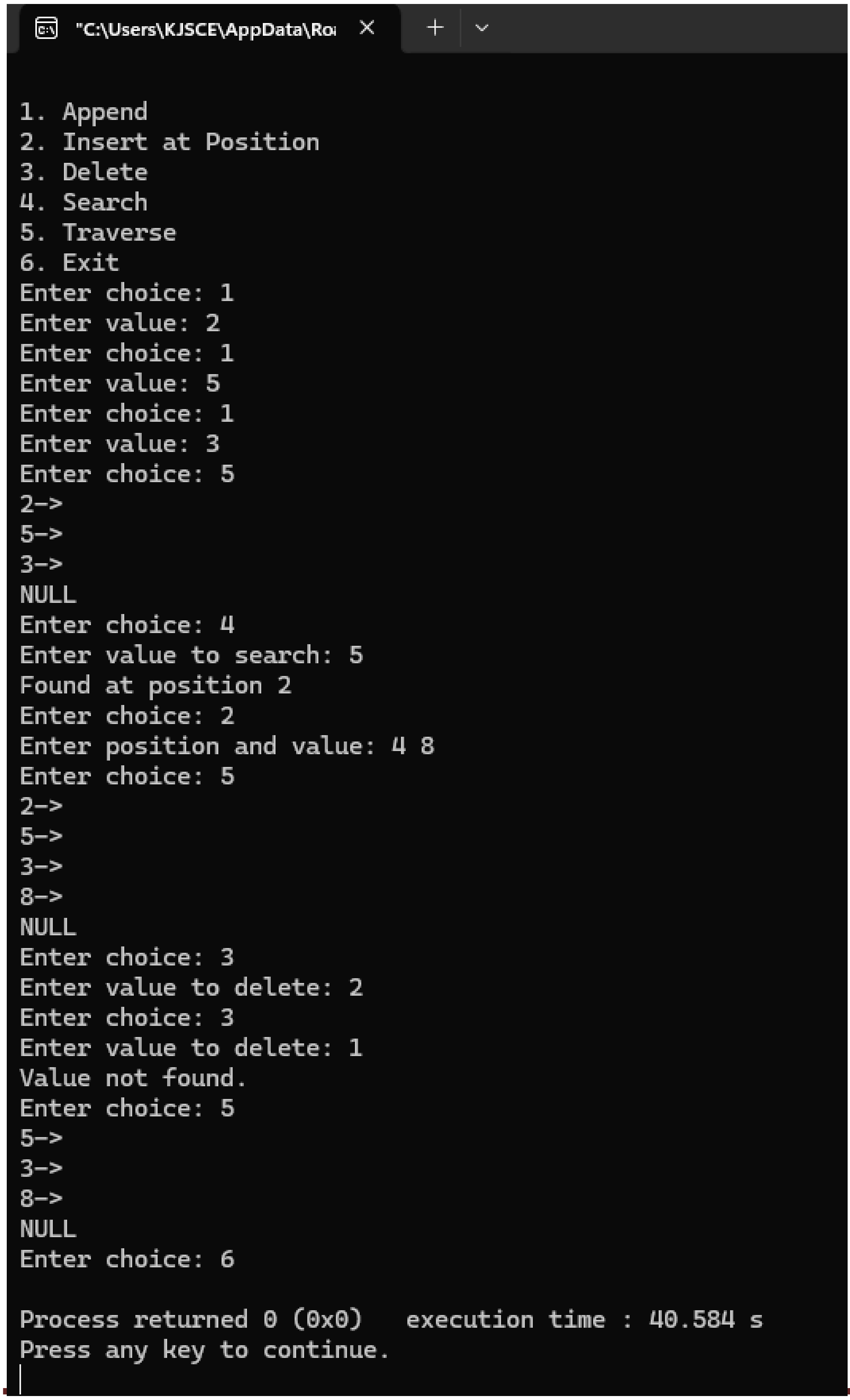
```
    {
        head = newnode;
        return;
    }

    Node* temp = head;
    while(temp->next!=NULL)
    {
        temp = temp->next;
    }
    temp->next = newnode;
}

void insertat(int pos, int value)
{
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = NULL;

    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
        //newNode becomes head
    }
    Node* temp = head;

    for (int i = 1; temp != NULL && i < pos - 1; i++)

       {

       temp = temp->next;

       }


    if (temp == NULL) {

       cout << "Position out of range.\n";

       return;

    }


    newNode->next = temp->next;

    temp->next = newNode;

}





void deleteNode(int value) {

    if (head == NULL) return; //nothing points


    if (head->data == value)

    {

       Node* toDelete = head;

       head = head->next;
    delete toDelete;

    return;

  //head becomes the next elemetn it pointed to

  //delete prev head

  }


  Node* temp = head;


  //store head temporarily

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

    temp = temp->next;

    //traverse till u find value

  }


  if (temp->next == NULL) //if u reach end and dont find value,

    {

    cout << "Value not found.\n";

    return;

  }


  Node* toDelete = temp->next; //else, next value is found. delete it.

  temp->next = temp->next->next; //temp skips that number and points to the next after it

  delete toDelete; //after rearranigng, delete the earlier node

}
```

```
void search(int value)

{

   Node* temp = head;

   int pos=1;

   while(temp!=NULL)

   {

      if(temp->data == value)

      {

         cout << "Found at position " << pos << endl;

         return;

      }

      pos++;

      temp = temp -> next;

   }

   cout << "Element not found\n";

}


void traverse()

{

   Node* temp = head;

   if(temp == NULL)

   {

      cout << "List is empty\n";

      return;

   }
```

```
    while(temp!=NULL)

    {

        cout << temp->data << "->"<< endl;

        temp = temp->next;

    }

    cout << "NULL\n";

}


int main()

{

    int choice;

                int pos,value;


        cout << "\n1. Append\n2. Insert at Position\n3. Delete\n4. Search\n5. Traverse\n6.
Exit\n";


        while(true)

        {

            cout << "Enter choice: ";

        cin >> choice;

switch (choice) {

            case 1:

                cout << "Enter value: ";

                cin >> value;

                append(value);
                break;


            case 2:

                cout << "Enter position and value: ";

                cin >> pos >> value;

                insertat(pos, value);

                break;


            case 3:

                cout << "Enter value to delete: ";

                cin >> value;

                deleteNode(value);

                break;


            case 4:

                cout << "Enter value to search: ";

                cin >> value;

                search(value);

                break;


            case 5:

                traverse();

                break;


            case 6:
```

```
            return 0;

          default:
            cout << "Invalid choice.\n";
        }
        }
      }
```

Output Screenshots:



```
1. Append
2. Insert at Position
3. Delete
4. Search
5. Traverse
6. Exit
Enter choice: 1
Enter value: 2
Enter choice: 1
Enter value: 5
Enter choice: 1
Enter value: 3
Enter choice: 5
2->
5->
3->
NULL
Enter choice: 4
Enter value to search: 5
Found at position 2
Enter choice: 2
Enter position and value: 4 8
Enter choice: 5
2->
5->
3->
8->
NULL
Enter choice: 3
Enter value to delete: 2
Enter choice: 3
Enter value to delete: 1
Value not found.
Enter choice: 5
5->
3->
8->
NULL
Enter choice: 6

Process returned 0 (0x0)   execution time : 40.584 s
Press any key to continue.
```

Conclusion:-

A linked list is a powerful data structure that helps us manage data dynamically. It allows for a chain based storage and makes insertion/deletion very quick and efficient. It has no fixed size and hence favours memory management.

Post lab questions:

1. Write the differences between linked list and linear array

Linear Array	Linked List
Fixed size	No fixed size
Contiguous	Non-contiguous
A lot of unused space is wasted	Space cannot be wasted
Insertion: O(n)	Insertion: O(1)
Read: O(1)	Read: O(n)

2. Name some applications which use a linked list.

- **Music/Video Playlists** – Next and previous songs are linked.
- **Web Browsers (History)** – Back and forward navigation uses linked lists.
- **Undo/Redo Functionality** – In editors like Word or Photoshop.
- **Dynamic Memory Allocation** – OS uses linked lists to manage free memory blocks.
- **Graph and Tree Implementations** – Adjacency lists in graphs for DSA and CP use linked lists.