Batch:_B2_ Roll No.:_16010122151

Experiment No. 2

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of different operations on 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

NAME:SMIT PATIL
ROLL NO. :16010122139
BATCH:B2

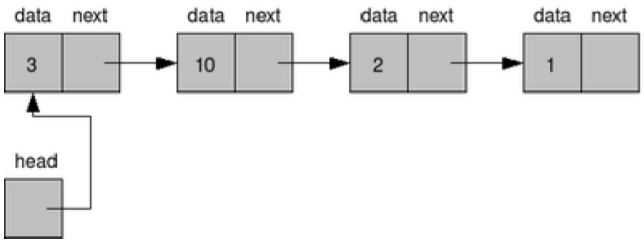
Expected Outcome of Experiment:

CO	Outcome
CO 1	To understand the advantage of linked list over other structures like arrays in implementing the general linear list

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 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 arecommonly 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:

abstract typedef linklist {

preconditions: linklist must not be empty

postcondition: makes a dynamic list where insertions and deletions can occur at any node.

}

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

- Creation of a Linked List (start):This function creates the initial node of the linked list and returns a pointer to it.
- Insertion at the Beginning (insertionbeginning): This function inserts a new node at the beginning of the linked list.
- Insertion at the End (insertionend): This function inserts a new node at the end of the linked list.
- Insertion at a Specified Index (insertionatindex): This function inserts a new node at the specified index of the linked list.
- Deletion from the Beginning (deletionbeginning) :This function deletes the first node from the linked list.
- Deletion from the End (deletionend): This function deletes the last node from the linked list.
- Deletion from a Specified Index (deletionatindex): This function deletes the node at the specified index from the linked list.
- Search for an Element (search) : This function searches for a given element in the linked list.
- Traversal (traversal): This function displays (traverses) the elements of the linked list.
- Main Function (main): The main function is where the user can interact with the linked list operations using a menu-driven approach.

Implementation of an application using linked lists:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node* next;
};
struct node*start(){
   int val;
   printf("\nEnter the value for linked list:");
   scanf("%d",&val);
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   temp->data=val;
   temp->next=NULL;
   return temp;
}
void traversal(struct node* head){
   printf("\n\n");
   while(head!=NULL){
      printf("%d ->",head->data);
      head=head->next;
   }
   printf("NULL");
   printf("\n");
}
void search(struct node* head){
   int val;
   printf("\nEnter the element to be searched:");
   scanf("%d",&val);
   while(head!=NULL){
```

```
        if(head->data==val){
            printf("\nThe element is found\n");
            return ;
        }
        head=head->next;
    }
    printf("The element was not found\n");
}
struct node* insertionbegining(struct node* head){
    int val;
    printf("\nEnter the element to be inserted at beginning:");
    scanf("%d",&val);
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data=val;
    temp->next=head;
    return temp;
}
struct node* insertionend(struct node* head){
    struct node* ptr=head;
    int val;
    printf("\nEnter the element to be inserted\n");
    scanf("%d",&val);
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data=val;
    while(head->next!=NULL){
        head=head->next;
    }
    temp->next=head->next;
    head->next=temp;
    return ptr;
}
struct node* insertionatindex(struct node* head){
    struct node* ptr=head;
```

```
    int val,index;
    printf("\nEnter the element to be inserted and its index:");
    scanf("%d",&val);
    scanf("%d",&index);
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data=val;
    for(int i=0;i<index-1;i++){
        head=head->next;
    }
    temp->next=head->next;
    head->next=temp;
    return ptr;
}
struct node* deletionbegining(struct node* head){
    if(head==NULL){
        printf("\nThe list is empty\n");
        return head;
    }
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp=head;
    head=head->next;
    free(temp);
    return head;
}
struct node* deletionend(struct node* head){
    if(head==NULL){
        printf("\nThe list is empty\n");
        return head;
    }
    struct node* ptr=head;
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp=head;
    head=head->next;
```

```
    while(head->next!=NULL){
        head=head->next;
        temp=temp->next;
    }
    temp->next=head->next;
    free(head);
    return ptr;
}
struct node* deletionatindex(struct node* head){
    if(head==NULL){
        printf("\nThe list is empty\n");
        return head;
    }
    struct node* ptr=head;
    int index;
    printf("\nEnter the index to be deleted:");
    scanf("%d",&index);
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp=head->next;
    for(int i=0;i<index-1;i++){
        temp=temp->next;
    }
    head->next=temp->next;
    free(temp);
    return ptr;
}
int main(){
    struct node* head=(struct node*)malloc(sizeof(struct node));
    head = start();
    printf("1 to insert at beginning\n");
    printf("2 to insert at end\n");
    printf("3 to insert at index\n");
    printf("4 to deletion at beginning\n");
```

```
printf("5 to deletion at end\n");
printf("6 to deletion at index\n");
printf("7 to search\n");
printf("8 to traverse(display)\n");
printf("0 to quit\n");
int ch=1;
while(ch!=0){

   printf("ENTER YOUR CHOICE:");
   scanf("%d",&ch);
   switch(ch)
   {
   case 1:
      head=insertionbegining(head);
      break;
   case 2:
      head=insertionend(head);
      break;
   case 3:
      head=insertionatindex(head);
      break;
   case 4:
      head=deletionbegining(head);
      break;
   case 5:
      head=deletionend(head);
      break;
   case 6:
      head=deletionatindex(head);
      break;
   case 7:
      search(head);
      break;
```

```
        case 8:
            traversal(head);
            break;
        case 0:
            printf("Exit");
            break;
        default:
            printf("\nPlease enter a valid choice\n");
            break;
        }
    }
    return 0;
}
```

```
Enter the value for linked list:25
1 to insert at beginning
2 to insert at end
3 to insert at index
4 to deletion at beginning
5 to deletion at end
6 to deletion at index
7 to search
8 to traverse(display)
0 to quit
ENTER YOUR CHOICE:1
Enter the element to be inserted at beginning:20
ENTER YOUR CHOICE:3
Enter the element to be inserted and its index:22 1
ENTER YOUR CHOICE:2
Enter the element to be inserted
90
ENTER YOUR CHOICE:8
20 ->22 ->25 ->90 ->NULL
ENTER YOUR CHOICE:5
```

```
Enter the element to be inserted and its index:22 1
ENTER YOUR CHOICE:2
Enter the element to be inserted
90
ENTER YOUR CHOICE:8
20 ->22 ->25 ->90 ->NULL
ENTER YOUR CHOICE:5
ENTER YOUR CHOICE:4
ENTER YOUR CHOICE:8
22 ->25 ->NULL
ENTER YOUR CHOICE:7
Enter the element to be searched:26
The element was not found
ENTER YOUR CHOICE:7
Enter the element to be searched:25
The element is found
ENTER YOUR CHOICE:8
22 ->25 ->NULL
ENTER YOUR CHOICE:0.
Exit
```


Conclusion:-

We learned linked list in this experiment. A linked list is a chain of nodes where each node in the list consists of two fields, a data field and a next address field. The data field holds the actual element on the list where the next address field contains the address of next node in the list. Hence, the next address field is simply a reference to the next node.

Post lab questions:

1. Write the differences between linked list and linear array

2. Name some applications which uses linked list.

1]

Sr. No.	LINEAR ARRAY	LINKED LIST
1.	An linear array is a consistent set of fixed number of data items.	A linked list is an ordered set of a variable number of data items.
2.	They are stored in contiguous memory locations.	They are not stored in contiguous memory locations.
3.	The memory allocation is done at compile time.	The memory allocation is done at run time.
4.	Fixed in size.	Dynamic in size.
5.	Less memory space as compared to linked lists.	More memory space.
6.	Isertion and deletion operations require more time to execute.	Insertion and deletion operations take less time.
7.	Accessing the elements is easier.	The whole linked list is to be traversed to access the elements.

2] Applications of linked list:

- Implementation of stacks and queues.

- Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.

- Dynamic memory allocation: We use a linked list of free blocks.

- Maintaining a directory of names.