**Batch: \_B3\_\_\_\_\_ Roll No.: \_121\_\_\_\_\_\_\_\_\_**

**Experiment No.**

**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

**Expected Outcome of Experiment:**

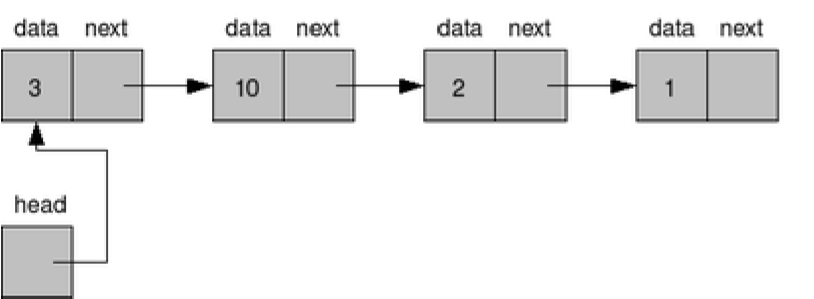
|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 1** | To understand the advantage of linked list over other structures like arrays in implementing the general linear list |

**Books/ Journals/ Websites referred:**

Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, “Data Structures Using C”, Pearson.

**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 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:**

Code for Basic Linked List:

#include<stdio.h>

#include<stdlib.h>

void create();

void insert();

void Delete();

void traverse();

void search();

int del\_el, i, ser\_el, flag;

struct Node

{

int data;

struct Node \*next;

};

struct Node \*start = NULL;

void main()

{

int ch;

do

{

printf("\nEnter:\n'1' to create a node.\n'2' to insert an element.\n'3' to delete a node.\n'4' to traverse through the linked list.\n'5' to search for an element.\n'6' to exit from the program.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

insert();

break;

case 3:

Delete();

break;

case 4:

traverse();

break;

case 5:

search();

break;

case 6:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("\nOnly '1', '2', '3', '4', '5' and '6' are allowed as inputs. Try again!");

break;

}

}while(ch!=6);

}

void create()

{

struct Node \*temp, \*ptr;

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

printf("\nEnter the element for the new node: ");

scanf("%d", &temp->data);

temp->next = NULL;

if(start==NULL)

{

start = temp;

}

else

{

ptr = start;

while(ptr->next!=NULL)

{

ptr = ptr -> next;

}

ptr -> next = temp;

}

}

void insert()

{

struct Node \*temp;

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

printf("\nEnter the element for the node to be inserted: ");

scanf("%d", &temp -> data);

temp -> next = NULL;

if(start==NULL)

{

start = temp;

}

else

{

temp -> next = start;

start = temp;

}

}

void Delete()

{

struct Node \*temp, \*ptr;

printf("\nEnter the location of the element to be deleted: ");

scanf("%d", &del\_el);

if(del\_el==0)

{

ptr = start;

start = start -> next;

printf("\nThe node deleted is: %d", ptr->data);

free(ptr);

}

else

{

ptr = start;

for(i = 0; i < del\_el; i++)

{

temp = ptr;

ptr = ptr -> next;

}

temp -> next = ptr -> next;

printf("\nThe node deleted is: %d", ptr->data);

free(ptr);

}

}

void traverse()

{

struct Node \*temp;

if(start==NULL)

{

printf("\nEmpty List!");

}

else

{

temp = start;

printf("\n");

while(temp != NULL)

{

printf("%d\t", temp -> data);

temp = temp -> next;

}

}

}

void search()

{

struct Node \*temp;

temp = start;

if(start==NULL)

{

printf("\nEmpty List!");

}

else

{

printf("\nEnter the element to be searched: ");

scanf("%d", &ser\_el);

while(temp!=NULL)

{

if(temp->data==ser\_el)

{

printf("\nElement found at location %d.", (i+1));

flag = 1;

break;

}

else

{

flag = 0;

}

temp = temp -> next;

i++;

}

if(flag==1)

printf("\nThe search was successful.");

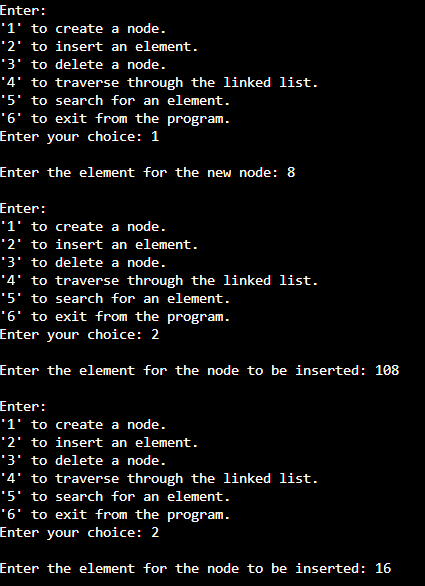
else

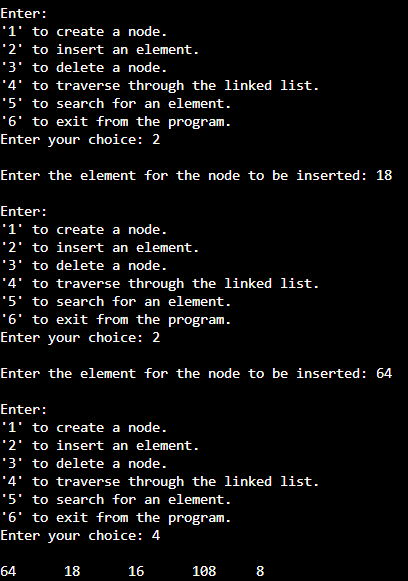
printf("\nThe search was unsuccessful.");

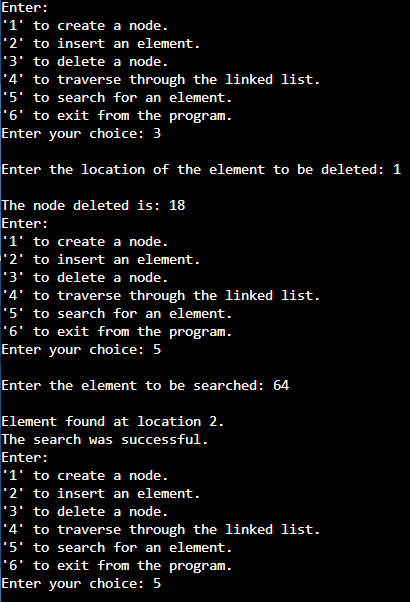
}

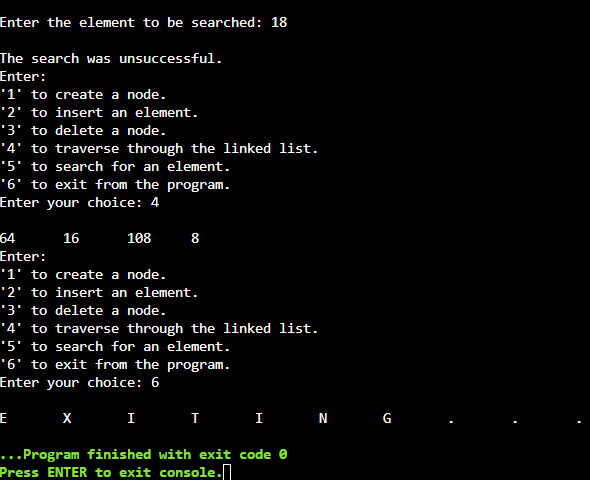
}

Output:









Code for Application-based Linked List:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void admission();

void cancellation();

void display();

int compare(char[], char[]);

struct Node

{

char name[100];

struct Node \*next;

};

struct Node \*start = NULL;

void main()

{

int ch;

do

{

printf("\nEnter:\n'1' to apply for student admission.\n'2' to cancel the student's admission.\n'3' to display the list of admitted students.\n'4' to exit from the program.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

admission();

break;

case 2:

cancellation();

break;

case 3:

display();

break;

case 4:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("Please enter '1', '2' and '3' only. Try again!");

}

}while(ch!=4);

}

void admission()

{

struct Node \*temp;

float marks;

char cert;

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

printf("\nEnter the name of the student: ");

scanf("%s", temp -> name);

printf("\nEnter the marks obtained in Grade XII: ");

scanf("%f", &marks);

printf("\nDo you have necessary documents and certificates?: ");

scanf("%s", &cert);

temp -> next = NULL;

if(marks < 35)

{

printf("\nWe would be glad to accept your ward, provided that\nhe should have secured at least 35 percent in his Class XII examinations.");

}

else

{

if(cert == 'N')

{

printf("\nWe would be glad to accept your ward, provided that\nall the necessary documents and certificates were present.");

}

else

{

if(start == NULL)

{

start = temp;

}

else

{

temp -> next = start;

start = temp;

}

}

}

}

void cancellation()

{

struct Node \*temp;

char adm\_cancel[100];

if(start==NULL)

{

printf("\nEmpty List!");

}

else

{

printf("\nEnter the name of the student whose admission you wish to cancel: ");

scanf("%s", adm\_cancel);

while(temp!=NULL)

{

if(compare(temp -> name, adm\_cancel) == 0)

{

printf("\nAdmission cancelled.");

free(temp);

//break;

}

else

temp = temp -> next;

}

}

}

void display()

{

struct Node \*temp;

temp = start;

printf("\n");

if(start==NULL)

{

printf("\nEmpty List!");

}

else

{

while(temp!=NULL)

{

printf("%s\t", temp -> name);

temp = temp -> next;

}

}

}

int compare(char a[], char b[])

{

int flag = 0, i = 0;

while(a[i] != '\0' && b[i] != '\0')

{

if(a[i] != b[i])

{

flag = 1;

break;

}

i++;

}

if(flag == 0)

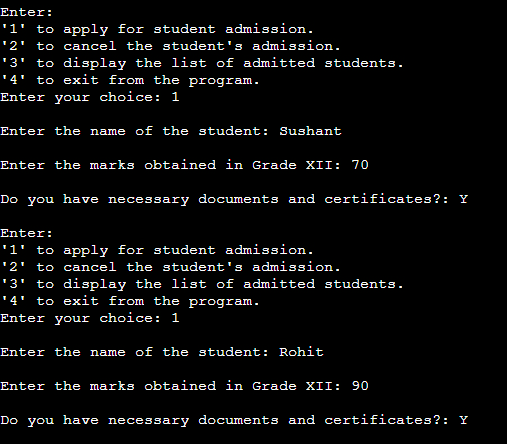
return 0;

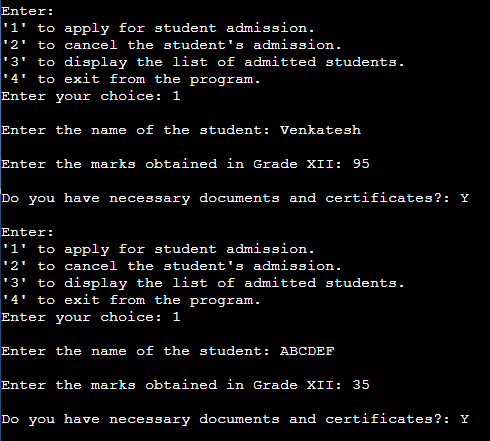
else

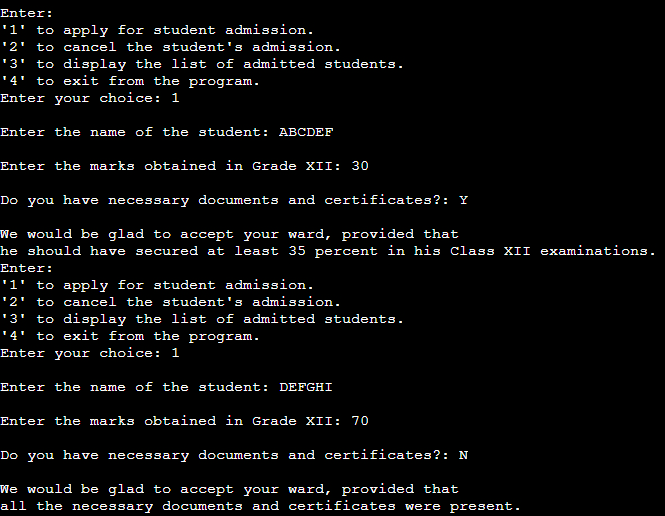
return 1;

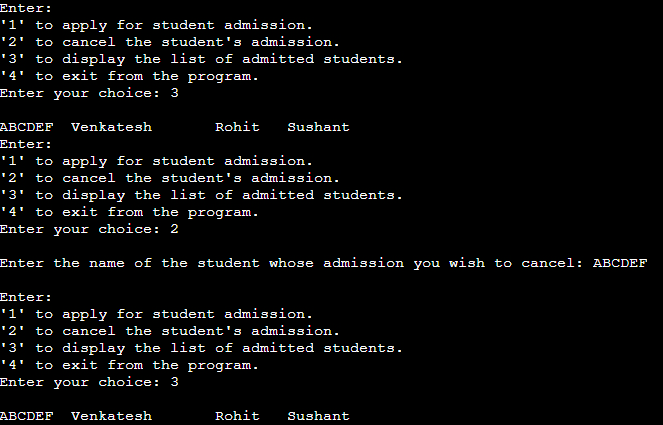
}

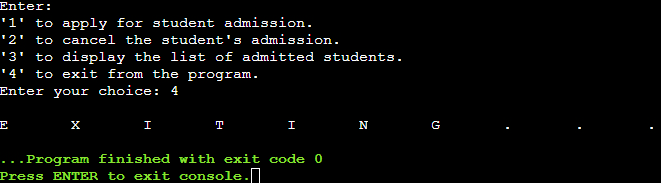
Output:



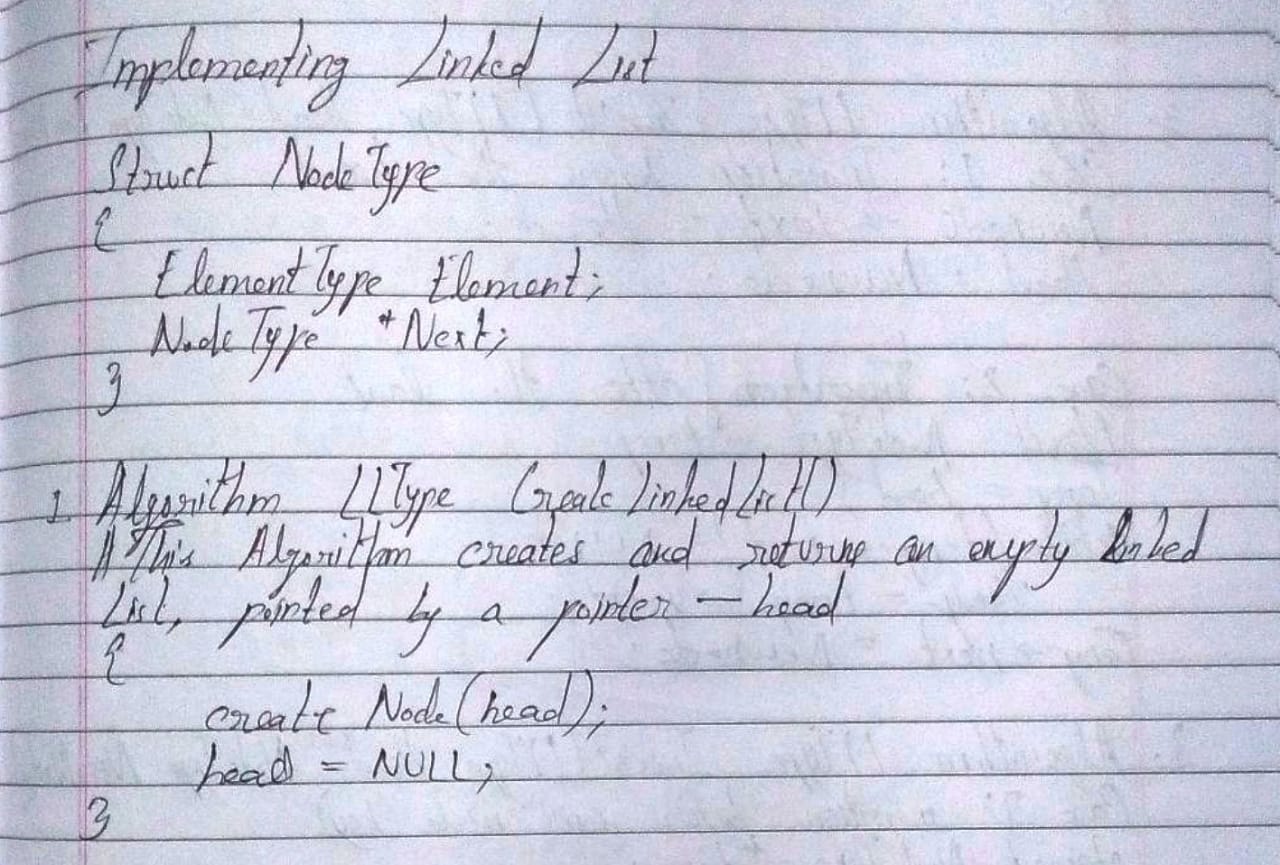


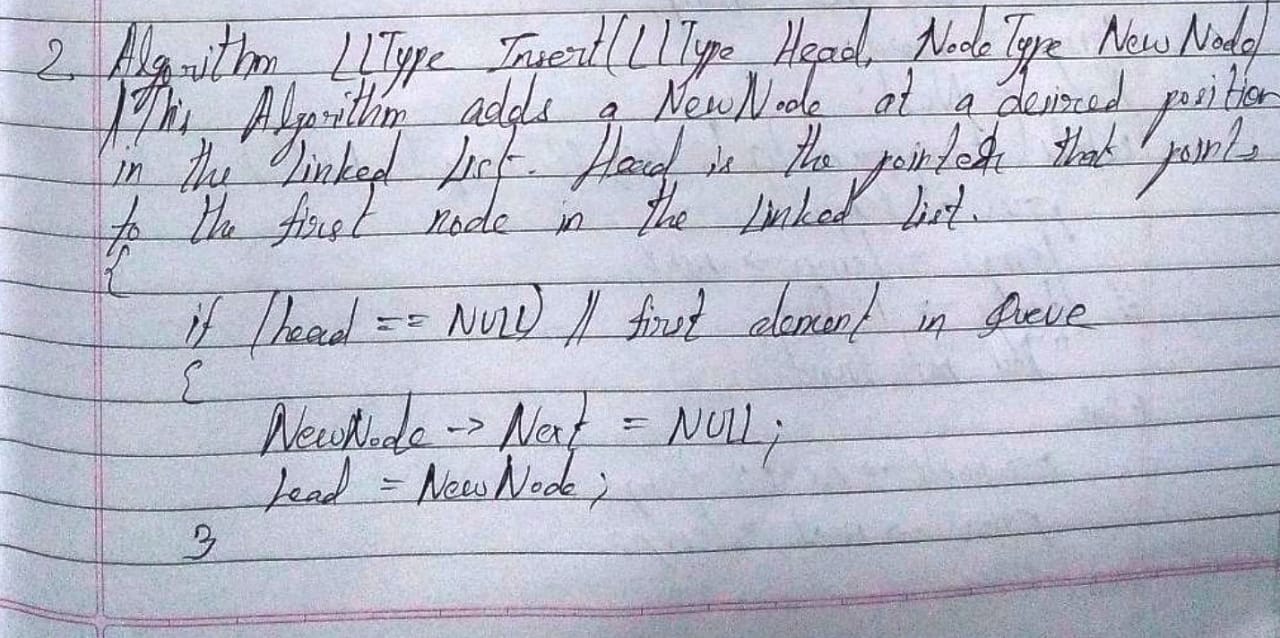


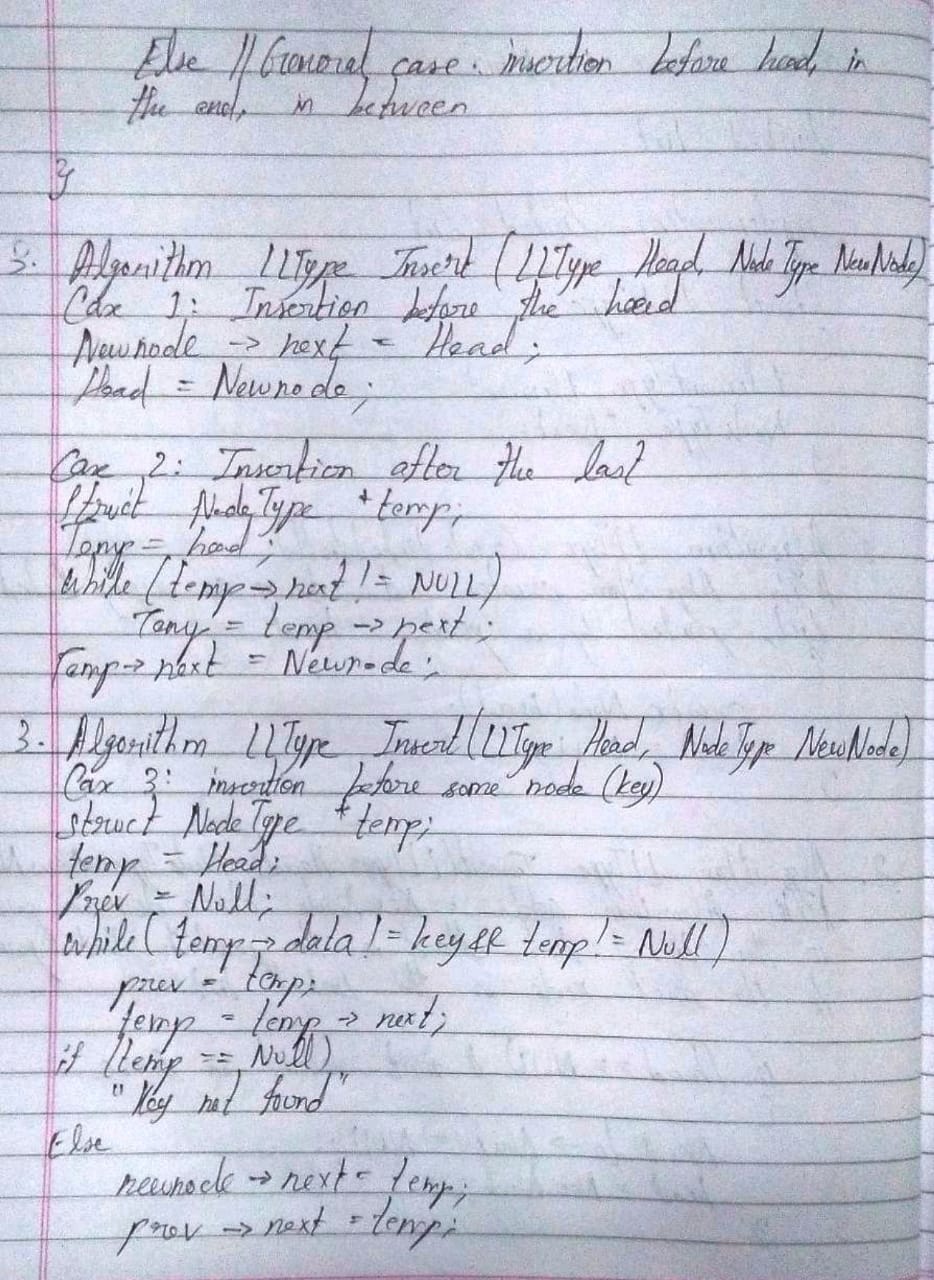


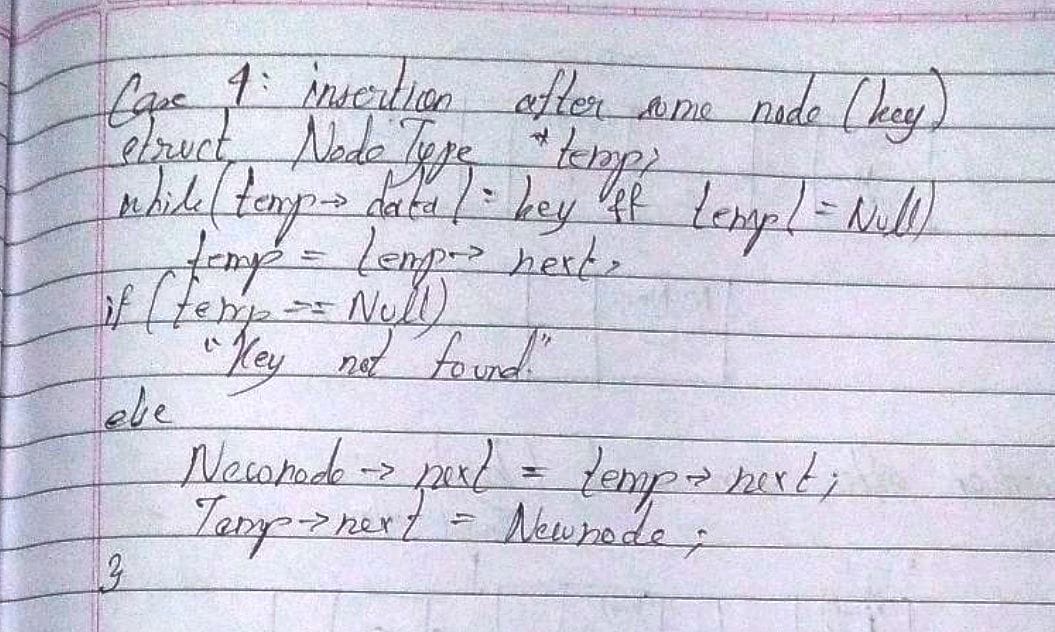


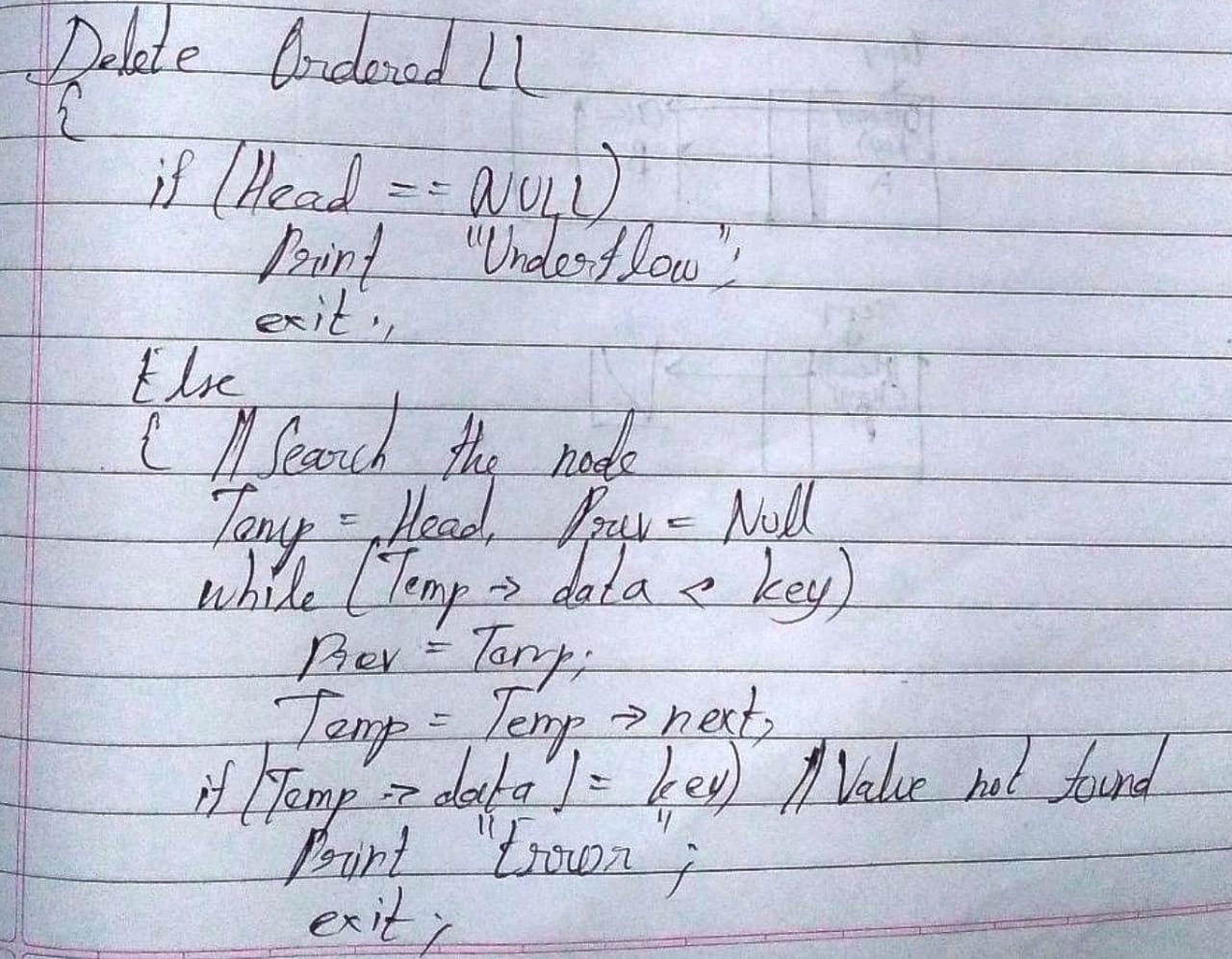
*Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:*

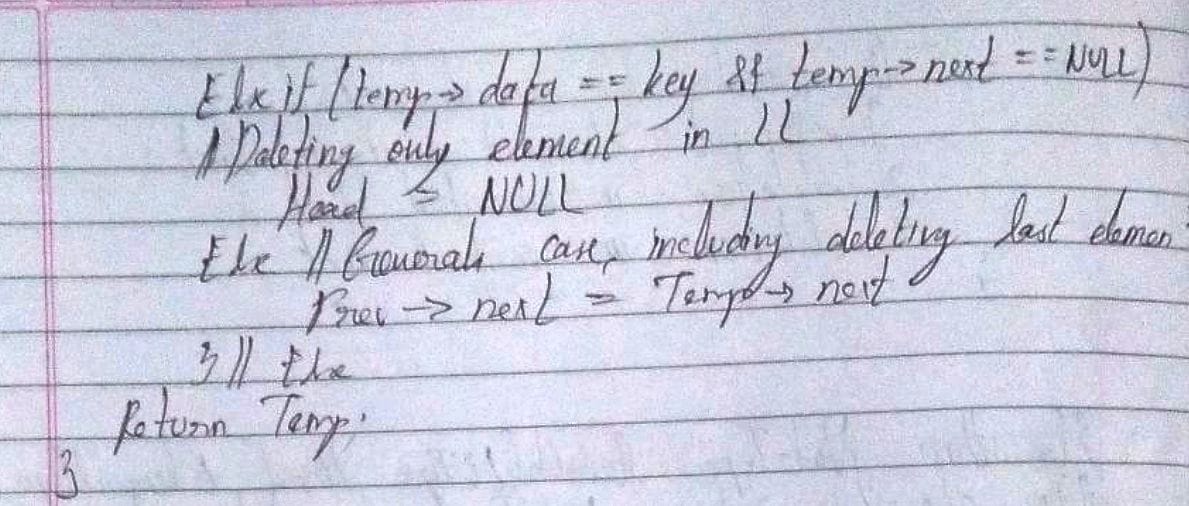


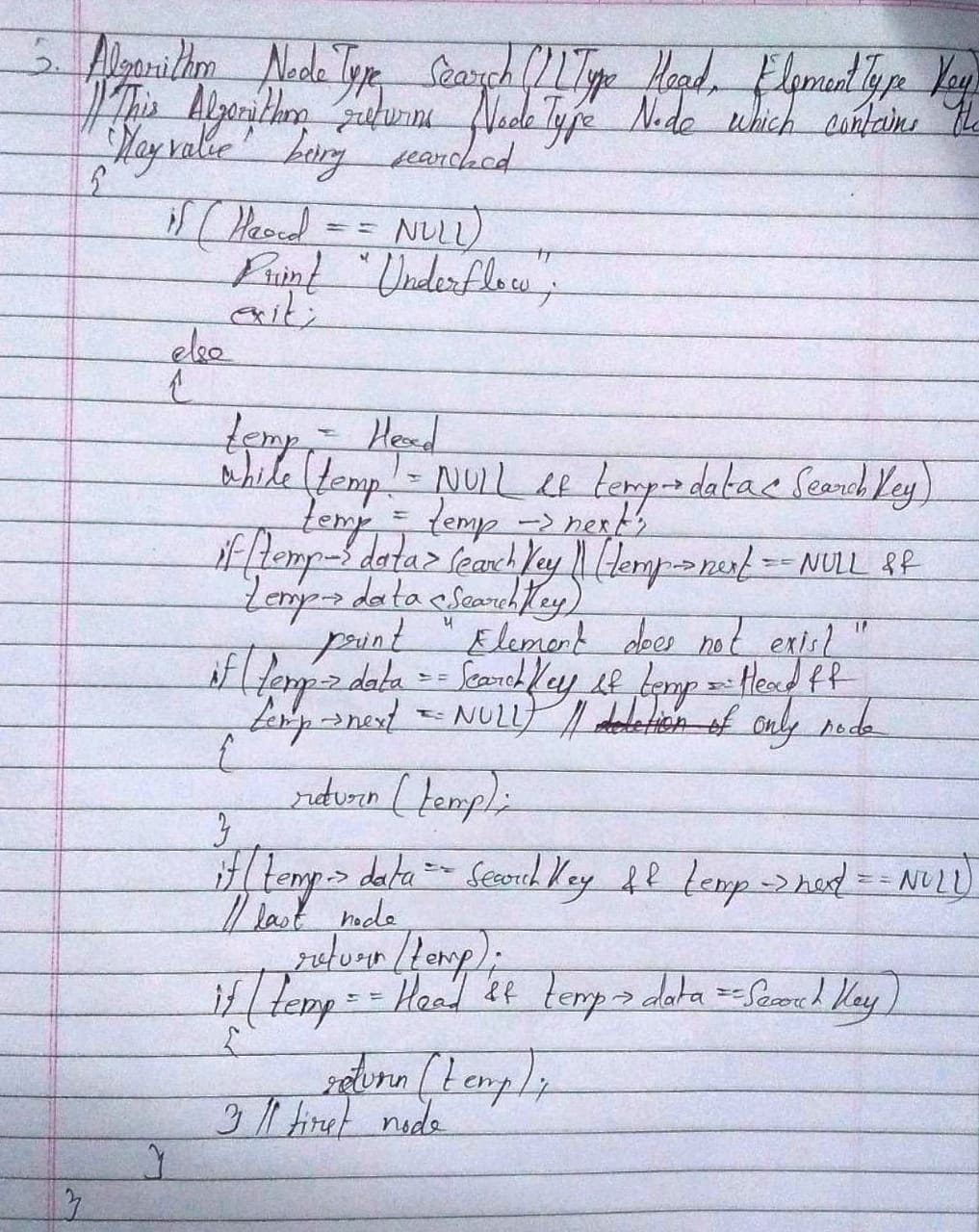


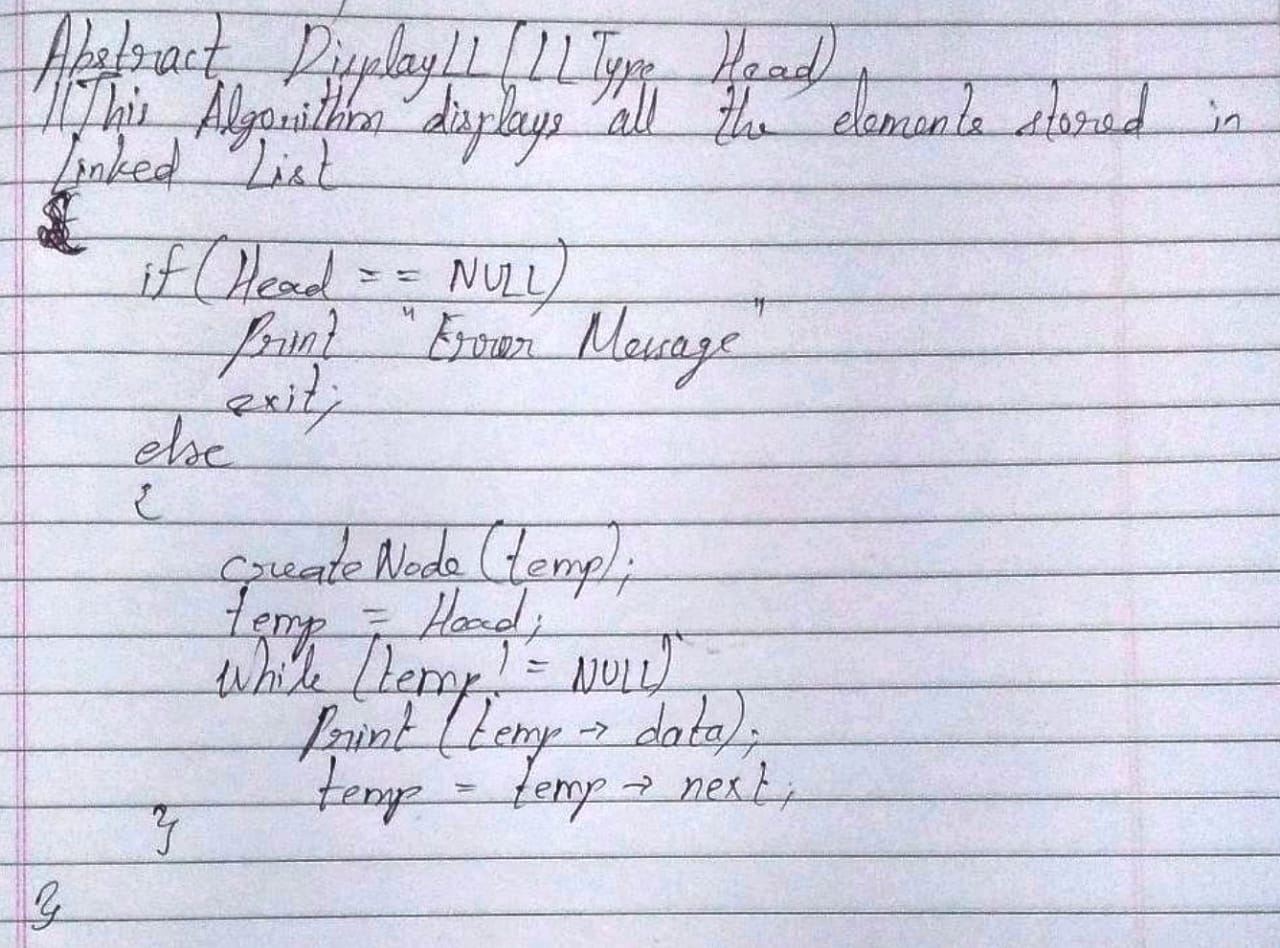












**Conclusion:-**

Thus, in this experiment, the application of Linked List has been demonstrated. Linked List has certain advantages as well as disadvantages over Arrays. However, when it comes to insertion and deletion operations, linked lists are certainly better than arrays. But, when it comes to traversing and searching, they are not as good as Arrays.

**Post lab questions:**

1. Write the differences between linked list and linear array

Ans.

|  |  |  |
| --- | --- | --- |
| Sr. No. | Linked List | Linear Array |
| 1. | Linked Lists are not stored in contiguous locations. | Arrays are stored in contiguous locations. |
| 2. | Their sizes can be changed during runtime as it is dynamic. | Their sizes cannot be changed during runtime as it is static. |
| 3. | The memory is allocated at run time. | The memory is allocated at compile time. |
| 4. | It uses more memory as it stores both data for a node as well as address for the next node. | It uses less memory as it stores only data for a node as it is known by default that the next node is located at the next memory location so there is no need to store the address of the next node. |
| 5. | In order to access an element, the list has to be traversed from the starting node upto that element. Thus, it is cumbersome. | In order to access an element, only the index of that element is required. So the element can be directly accessed. Thus, it is convenient. |
| 6. | Insertion and deletion processes are direct and hence faster. | Insertion and deletion processes require copying the entire array to a new array with the adjusted size. So, it is slower. |

1. Name some applications which uses linked list.

Ans. Some real-world applications of Linked List are as follows:

* 1. Image viewer – the previous and next images are linked. Thus, from a particular image, the previous and next images can be accessed by pressing the appropriate buttons.
  2. Previous and next page in a web browser – from a particular webpage, we can go to the previous page or the next page as the pages are linked as a linked list.
  3. Song player – from a particular song tract, we can play the previous or the next track as the song tracks are stored as linked lists.