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

**Experiment No. 6**

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

|  |
| --- |
| **Title: Implementation of Linked List LL- doubly LL, circular LL, circular doubly LL** |

**Objective:** To understand the use of linked list as data structures for various application.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 2** | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

1. Fundamentals Of Data Structures In C – Ellis Horowitz, Satraj Sahni, Susan

Anderson-Fred

2. Data Structures Using C - Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein

3. <https://www.javatpoint.com/singly-linked-list>

4. <https://www.simplilearn.com/tutorials/data-structure-tutorial/types-of-linked-list>

1. <https://www.javatpoint.com/singly-linked-list>
2. <https://www.javatpoint.com/doubly-linked-list>
3. <https://www.javatpoint.com/circular-singly-linked-list>
4. <https://www.javatpoint.com/circular-doubly-linked-list>
5. https://www.geeksforgeeks.org/difference-between-singly-linked-list-and-doubly-linked-list/#:~:text=DLL%20nodes%20contains%203%20fields,or%20the%20next%20node%20link.

**Introduction:**

Linked List can be defined as collection of objects called nodes that are randomly stored in memory. A node contains two fields i.e., data stored at that particular address and the pointer which contains the address of the next node in memory. The last node of the list contains pointer to null.

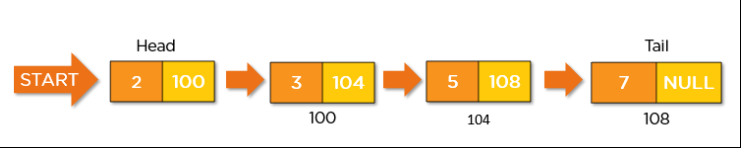
The linked list is not required to be present contiguously in memory. The node can reside anywhere in memory and can be linked together to form the linked list. This results in optimum utilization of memory and easy insertion and deletion of new nodes.

The list size is determined by the memory size and thus, does not need to be known in advance. Also, an empty node cannot be present in a linked list. Further, values of primitive types or of objects can be stored in a linked list.

**Types of linked list:**

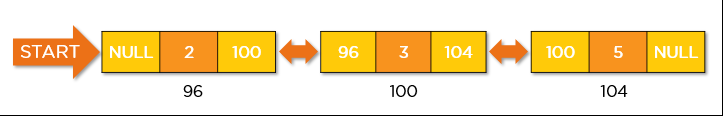
1. Singly Linked List

It is a unidirectional linked list. So, it can be traversed only in one direction, i.e., from head node to tail node. They are used to store a list of items that need to be accessed in order. It consists of one data field, which stores the data of the node, and one address field, which stores the address of the next pointer.



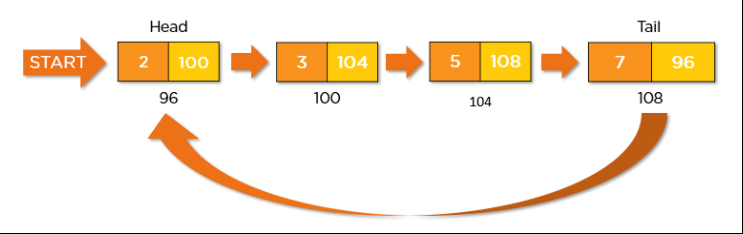
1. Doubly Linked List

It is a bi-directional linked list. So it can be traversed in both directions. A doubly linked list of singly linked lists is a data structure that consists of a set of singly linked lists, each of which is doubly linked. It is an ideal data structure for applications that required the ability to move both forward and backward through a list of data. It consists of three fields – one data field in the middle which stores the data of the node, one address field in the front which stores the address of the next node and one address field at the rear which stores address of the previous field.



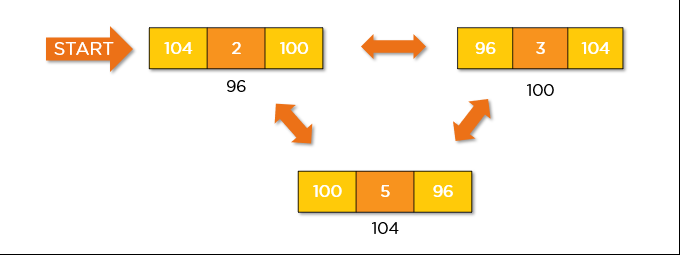
1. Circular Linked List

It is a unidirectional linked list. So, it can be traversed in one direction only. However, the last node in the linked list has an address field which contains the address of the first node of the linked list, i.e., the last node points to the first node. It is useful in applications which require looping through a set of data.

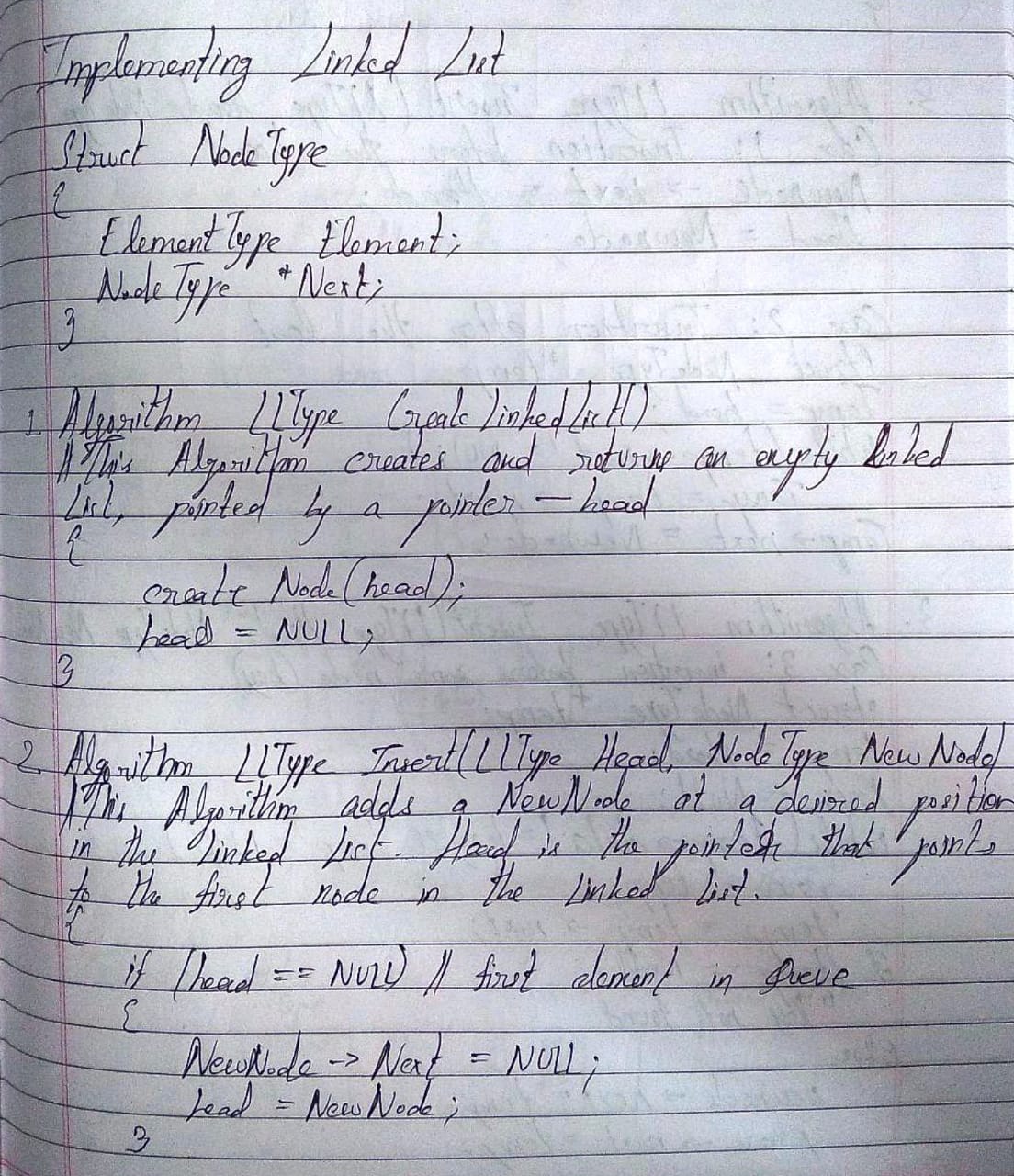


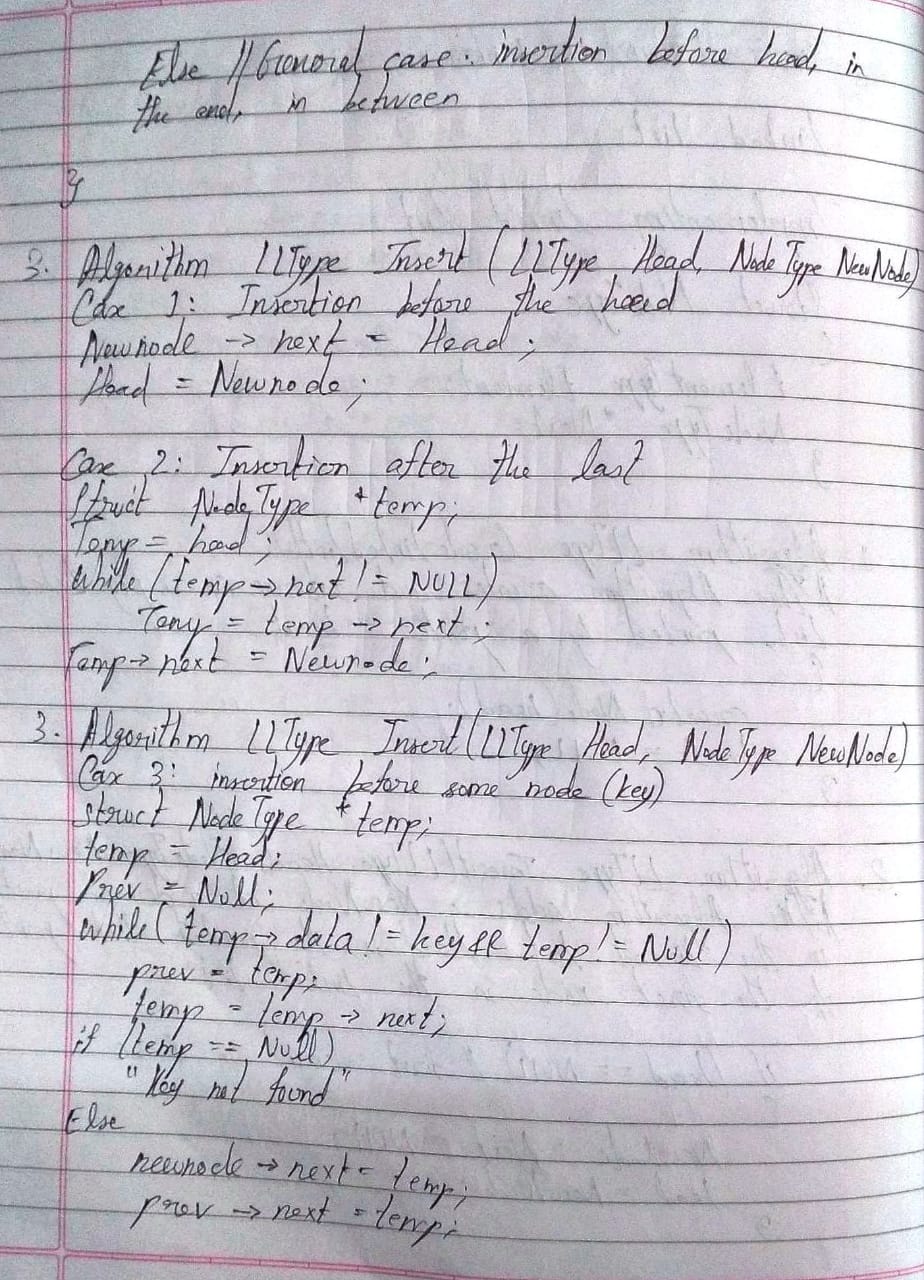
1. Circular Doubly Linked List

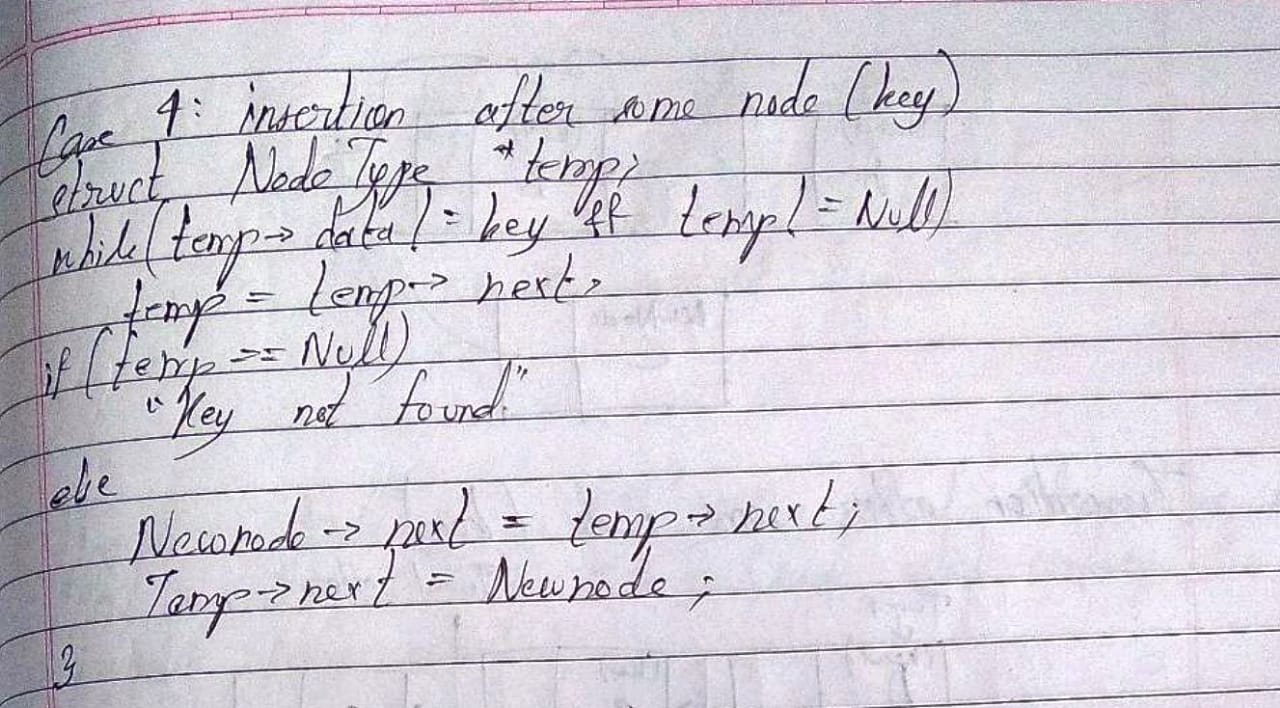
It is a mixture of a doubly linked list and a circular linked list. Like the doubly linked list, each node has an extra pointer which points to the previous node of the linked list. A Circular Doubly Linked List is a variation of the Doubly Linked List. In it, the first and last nodes a linked together, forming a circle. This allows for quick and easy traversal of the entire list without the need for special case handling at the beginning or end of the list. It combines the advantages of both Circular Linked List and Doubly Linked List and are used in applications which require the features of both the types of linked lists in a single linked list.

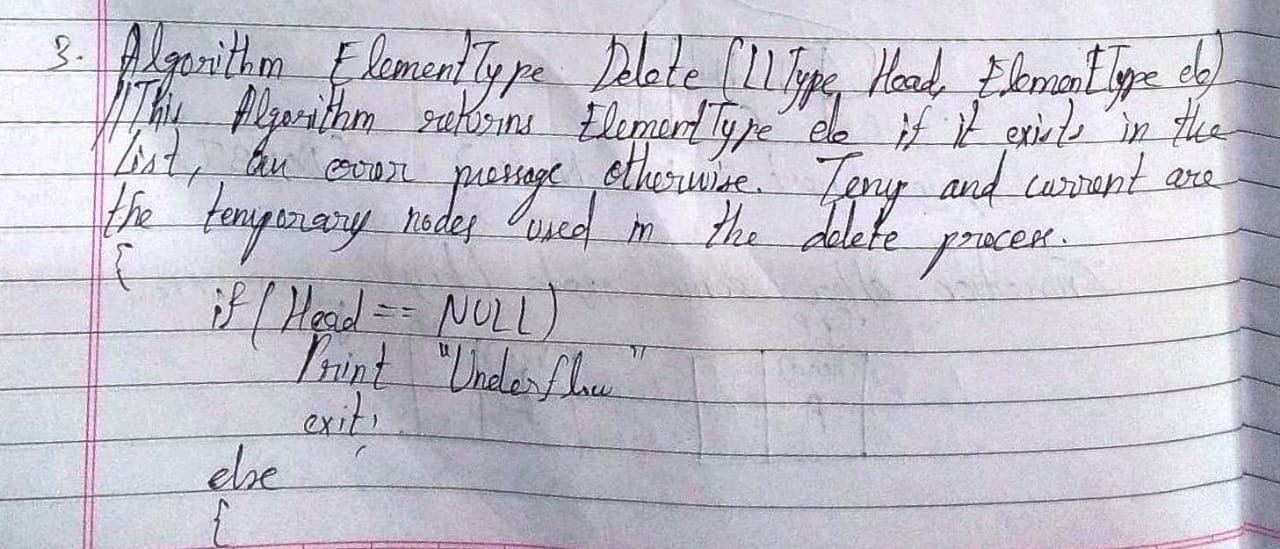


**Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:**

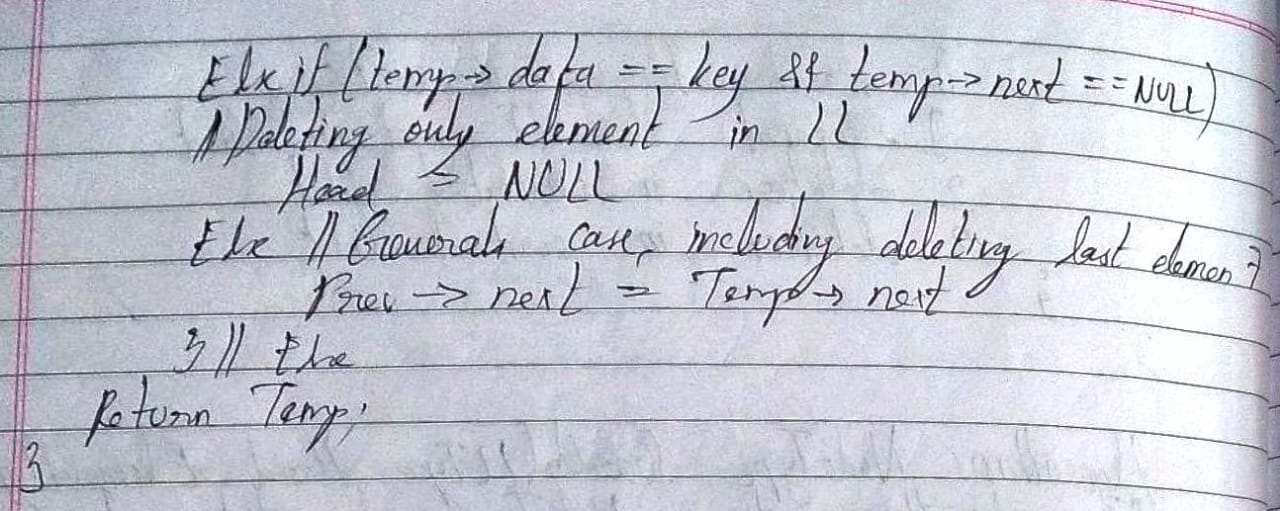


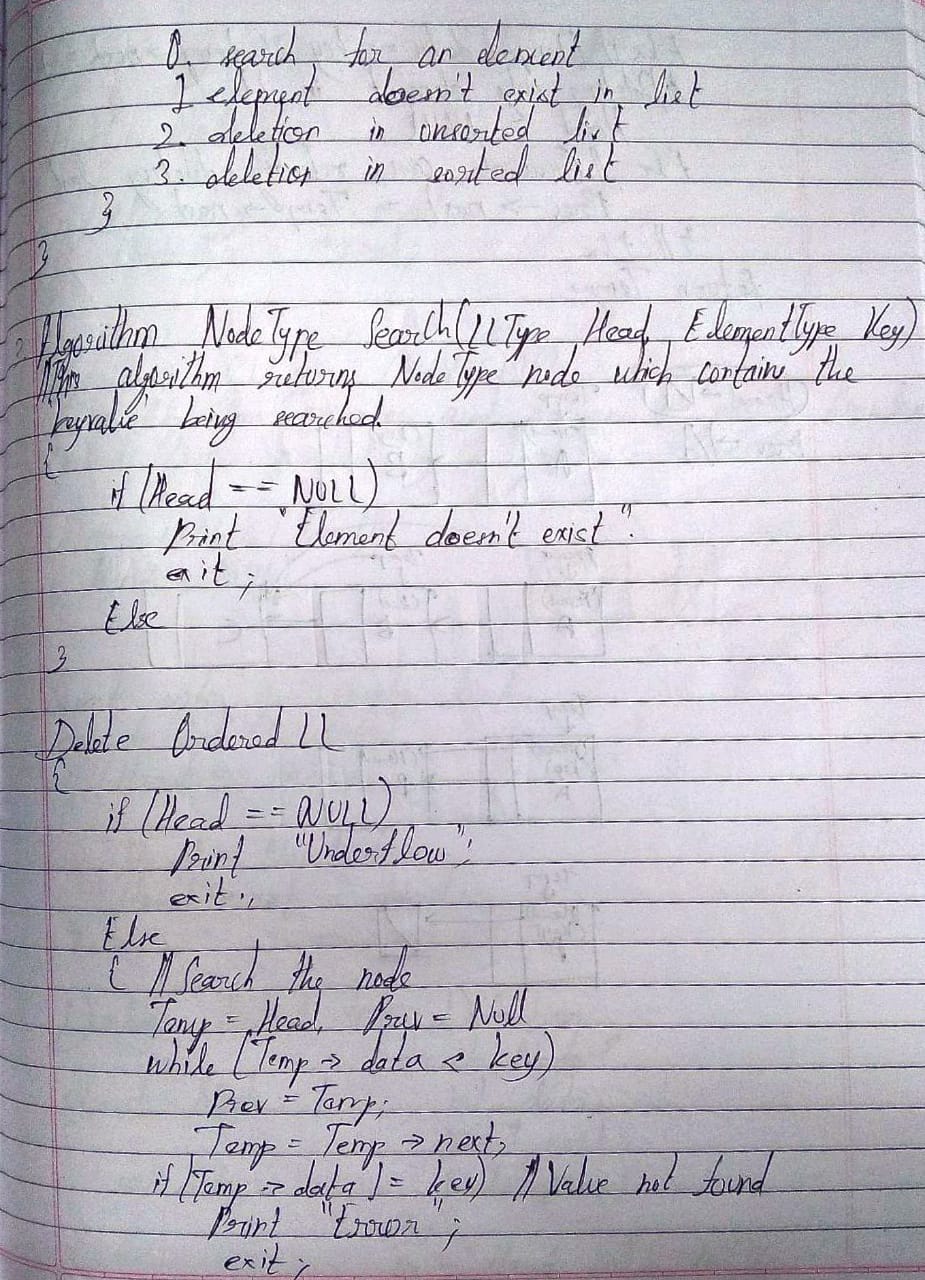


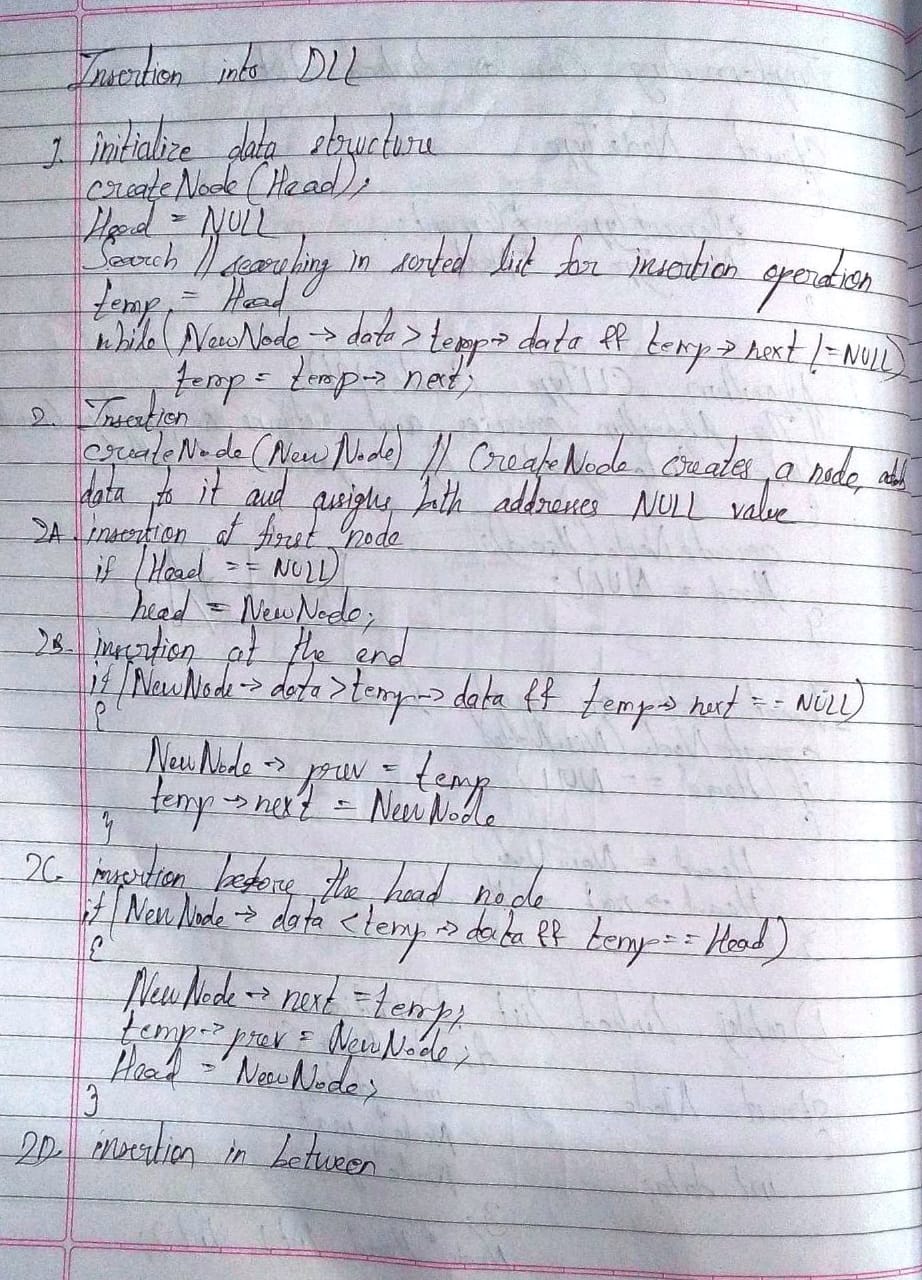


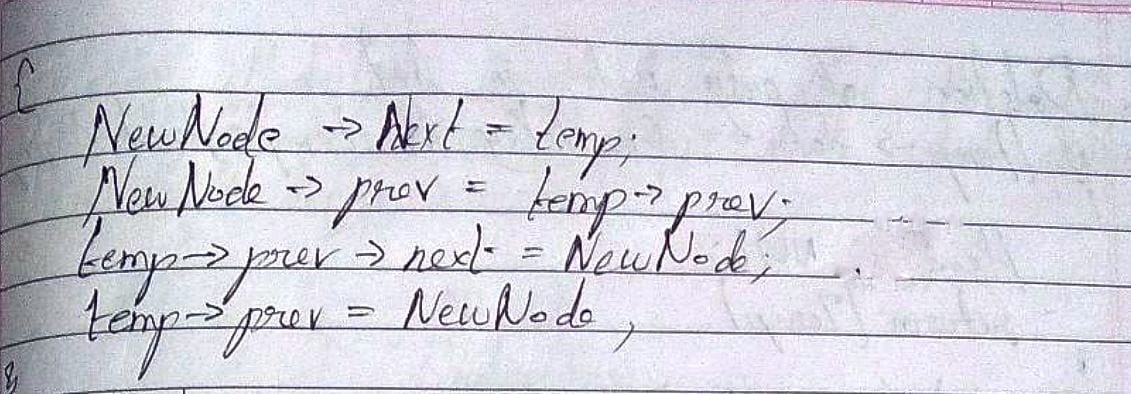


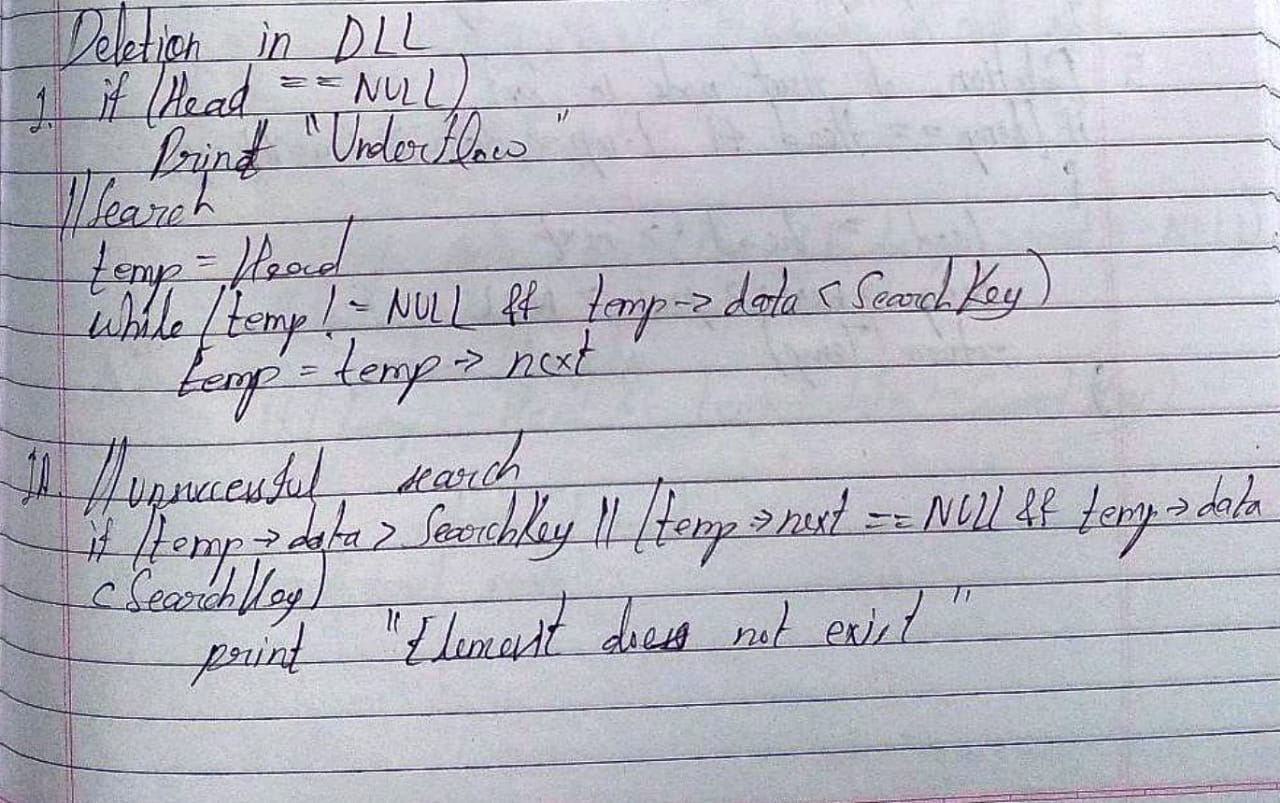
(the picture below comes after the next picture as per the order)

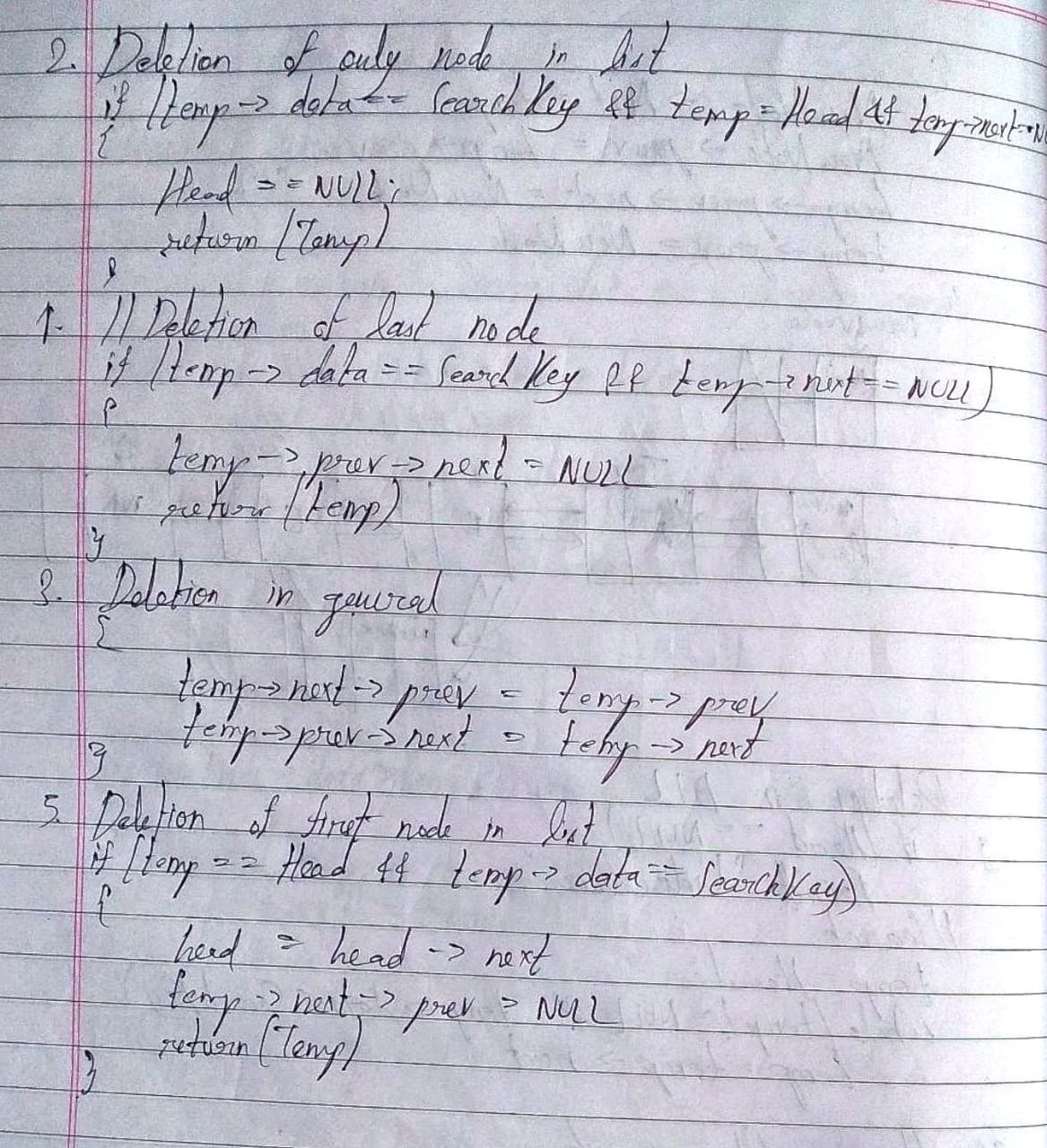


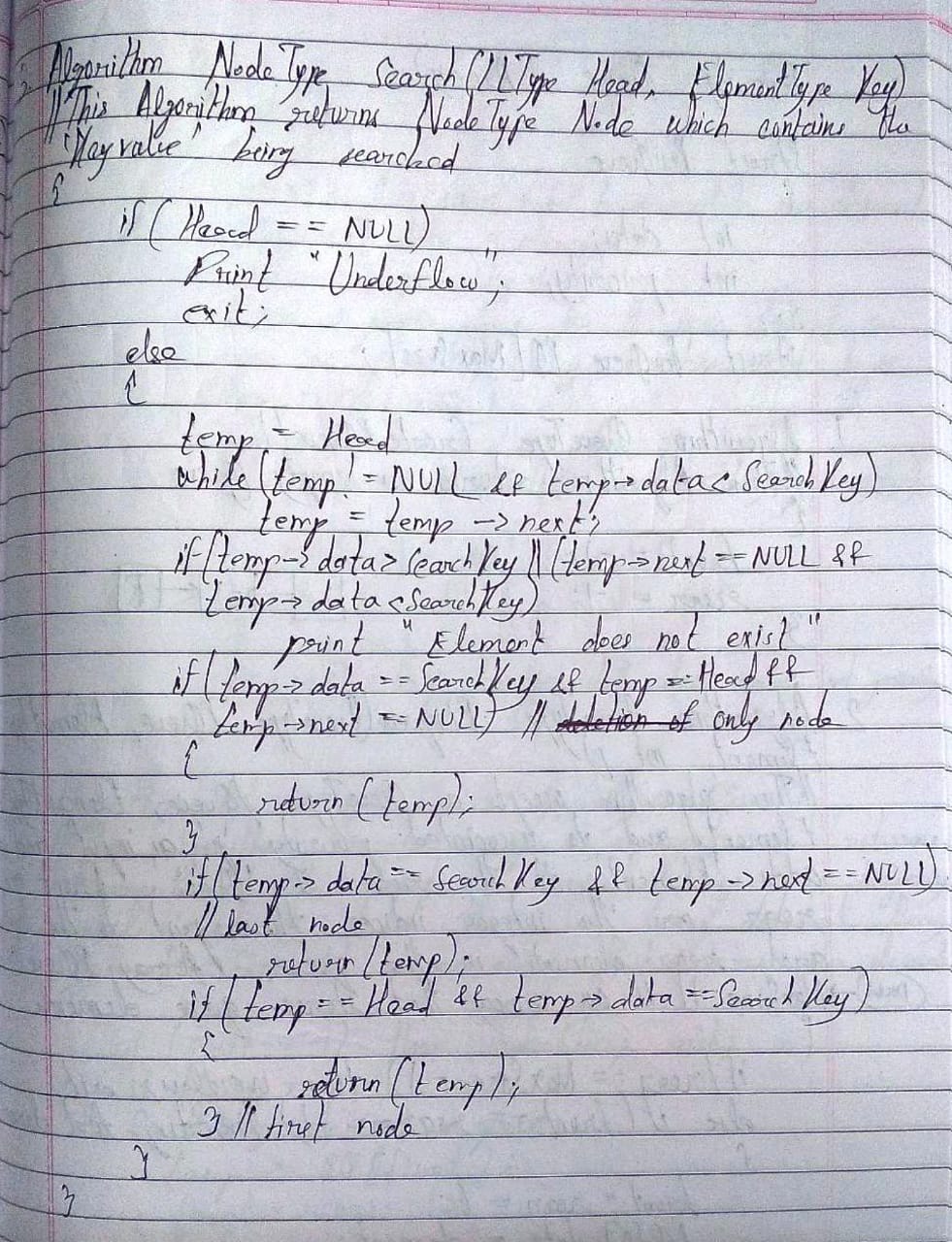












**Implementation of an application using linked list:**

**Singly Linked List:**

Code:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*head;

void insert();

void delete();

void display();

void search();

void main()

{

int ch;

printf("\nSINGLY\tLINKED\tLIST\tIMPLEMENTATION\n");

do

{

printf("\nEnter:\n'1' to insert a node.\n'2' to delete a node.\n'3' to display the list.\n'4' to search for a node.\n'5' to exit.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

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

break;

default:

printf("\nPlease enter '1', '2', '3', '4' or '5' only and try again!");

}

}while(ch!=5);

}

void insert()

{

struct Node \*temp, \*ptr;

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

int pos, locn;

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

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

temp -> next = NULL;

printf("\nEnter:\n'1' to insert at beginning.\n'2' to insert at end.\n'3' to insert in between.\nEnter your choice: ");

scanf("%d", &pos);

if(temp == NULL)

{

printf("\nMemory allocation has been unsuccessful.");

}

else if(pos == 1)

{

temp -> next = head;

head = temp;

printf("\nInsertion Successful!");

}

else if(pos == 2)

{

if(head == NULL)

{

head = temp;

temp -> next = NULL;

printf("\nInsertion Successful!");

}

else

{

ptr = head;

while(ptr -> next != NULL)

{

ptr = ptr -> next;

}

ptr -> next = temp;

temp -> next = NULL;

printf("\nInsertion Successful!");

}

}

else if(pos == 3)

{

printf("\nEnter the location for insertion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr = ptr -> next;

if(ptr == NULL)

{

printf("\nInsertion Unsuccessful.");

return;

}

}

temp -> next = ptr -> next;

ptr -> next = temp;

printf("\nInsertion Successful!");

}

}

void delete()

{

struct Node \*ptr, \*ptr1;

int pos, locn;

printf("\nEnter:\n'1' to delete at beginning.\n'2' to delete at end.\n'3' to delete in between.\nEnter your choice: ");

scanf("%d", &pos);

if(head == NULL)

{

printf("\nUnderflow!");

}

if(pos == 1)

{

ptr = head;

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

head = head -> next;

free(ptr);

}

else if(pos == 2)

{

if(head -> next == NULL)

{

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

head = NULL;

free(head);

}

else

{

ptr = head;

while(ptr -> next != NULL)

{

ptr1 = ptr;

ptr = ptr -> next;

}

ptr1 -> next = NULL;

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

free(ptr);

}

}

else if(pos == 3)

{

printf("\nEnter the location of deletion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr1 = ptr;

ptr = ptr -> next;

if(ptr == NULL)

{

printf("\nDeletion Unsuccessful.");

return;

}

}

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

ptr1 -> next = ptr -> next;

free(ptr);

}

}

void search()

{

struct Node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n");

}

}

}

void display()

{

struct Node \*ptr;

ptr = head;

if(head == NULL)

{

printf("\nUnderflow!");

}

else

{

while(ptr != NULL)

{

printf("%d\n", ptr -> data);

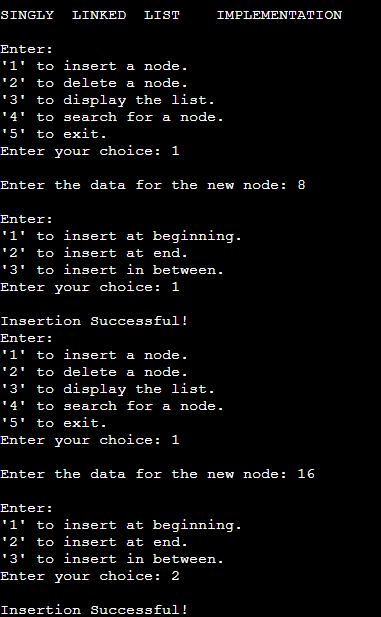
ptr = ptr -> next;

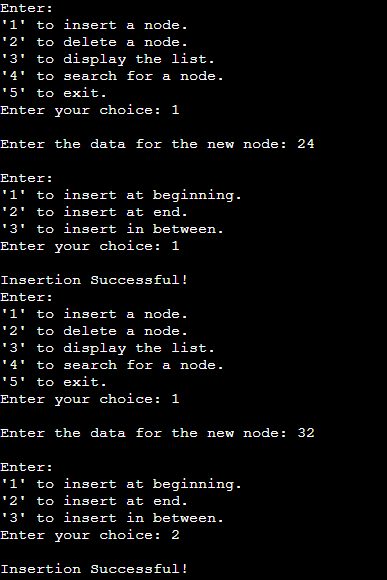
}

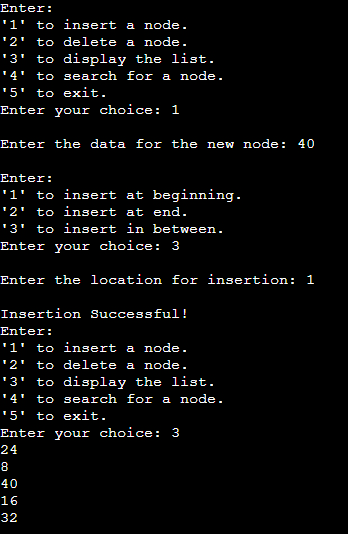
}

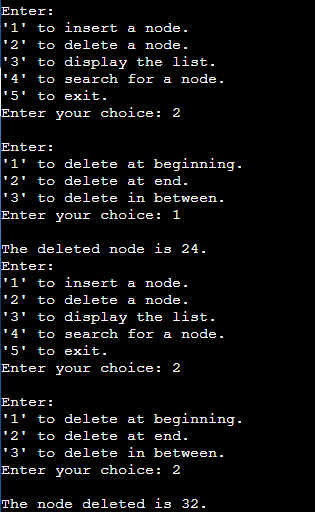
}

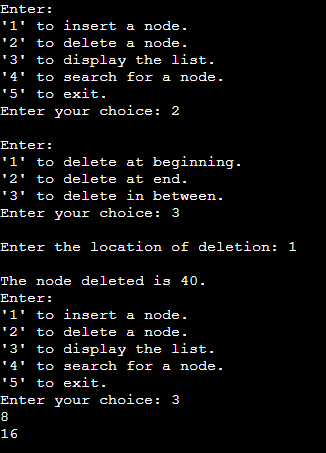
Output:

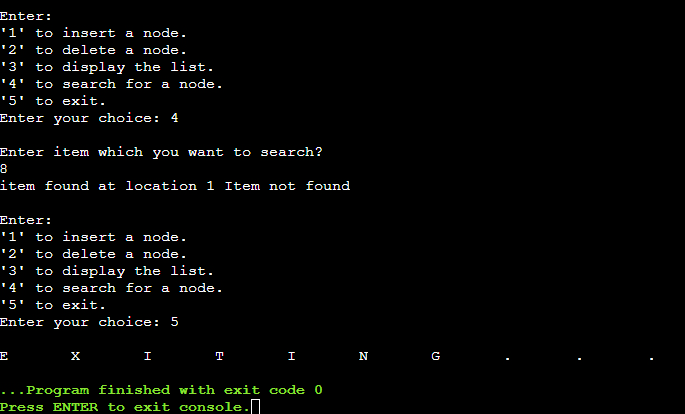












**Doubly Linked List:**

Code:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

};

struct Node \*head;

void insert();

void delete();

void display();

void search();

void main()

{

int ch;

printf("\nDOUBLY\tLINKED\tLIST\tIMPLEMENTATION\n");

do

{

printf("\nEnter:\n'1' to insert a node.\n'2' to delete a node.\n'3' to display the list.\n'4' to search for a node.\n'5' to exit.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

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

break;

default:

printf("\nPlease enter '1', '2', '3', '4' or '5' only and try again!");

}

}while(ch!=5);

}

void insert()

{

struct Node \*temp, \*ptr;

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

int pos, locn;

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

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

temp -> next = NULL;

temp -> prev = NULL;

printf("\nEnter:\n'1' to insert at beginning.\n'2' to insert at end.\n'3' to insert in between.\nEnter your choice: ");

scanf("%d", &pos);

if(temp == NULL)

{

printf("\nMemory allocation has been unsuccessful.");

}

else if(pos == 1)

{

if(head == NULL)

{

temp -> next = NULL;

temp -> prev = NULL;

head = temp;

}

else

{

temp -> prev = NULL;

temp -> next = head;

head -> prev = temp;

head = temp;

}

printf("\nInsertion Successful!");

}

else if(pos == 2)

{

if(head == NULL)

{

temp -> prev = NULL;

temp -> next = NULL;

head = temp;

}

else

{

ptr = head;

while(ptr -> next != NULL)

{

ptr = ptr -> next;

}

ptr -> next = temp;

temp -> prev = ptr;

temp -> next = NULL;

}

printf("\nInsertion Successful!");

}

else if(pos == 3)

{

printf("\nEnter the location for insertion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr = ptr -> next;

if(ptr == NULL)

{

printf("\nInsertion Unsuccessful.");

return;

}

}

temp -> next = ptr -> next;

temp -> prev = ptr;

ptr -> next = temp;

ptr -> next -> prev = temp;

printf("\nInsertion Successful!");

}

}

void delete()

{

struct Node \*ptr, \*ptr1;

int pos, locn;

printf("\nEnter:\n'1' to delete at beginning.\n'2' to delete at end.\n'3' to delete in between.\nEnter your choice: ");

scanf("%d", &pos);

if(head == NULL)

{

printf("\nUnderflow!");

}

if(pos == 1)

{

if(head -> next == NULL)

{

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

free(head);

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

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

free(ptr);

}

}

else if(pos == 2)

{

if(head -> next == NULL)

{

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

free(head);

}

else

{

ptr = head;

while(ptr -> next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev = NULL;

ptr -> next = NULL;

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

free(ptr);

}

}

else if(pos == 3)

{

int deldata;

printf("\nEnter the data after which the node should be deleted: ");

scanf("%d", &deldata);

ptr = head;

while(ptr -> data != deldata)

ptr = ptr -> next;

if(ptr -> next == NULL)

printf("\nDeletion Unsuccessful.");

else if(ptr -> next -> next == NULL)

ptr -> next = NULL;

else

{

ptr1 = ptr -> next;

ptr -> next = ptr1 -> next;

ptr1 -> next -> prev = ptr;

printf("\nThe deleted node is %d.", ptr1 -> data);

free(ptr1);

}

}

}

void search()

{

struct Node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nUnderflow!");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n");

}

}

}

void display()

{

struct Node \*ptr;

ptr = head;

if(head == NULL)

{

printf("\nUnderflow!");

}

else

{

while(ptr != NULL)

{

printf("%d\n", ptr -> data);

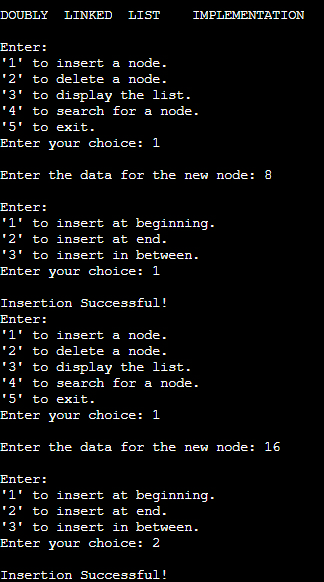
ptr = ptr -> next;

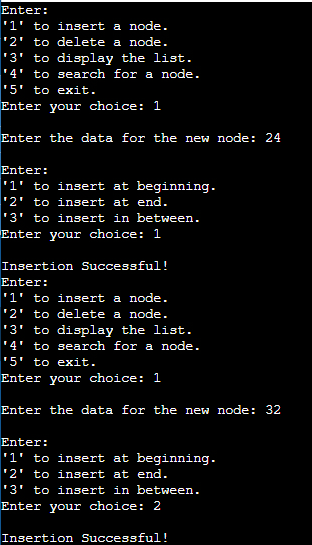
}

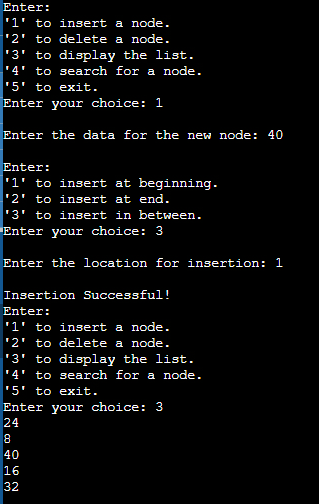
}

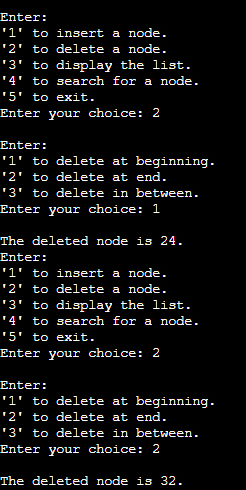
}

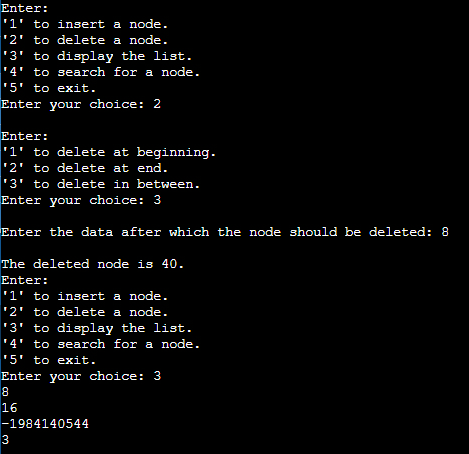
Output:

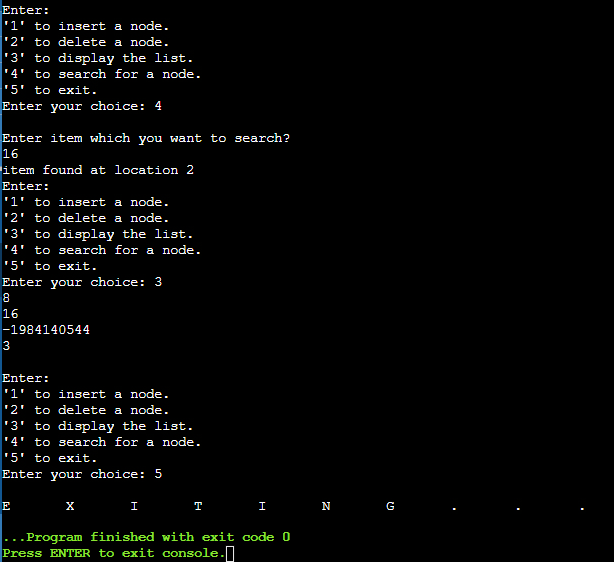












**Circular Linked List:**

Code:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*head;

void insert();

void delete();

void display();

void search();

void main()

{

int ch;

printf("\nCIRCULAR\tLINKED\tLIST\tIMPLEMENTATION\n");

do

{

printf("\nEnter:\n'1' to insert a node.\n'2' to delete a node.\n'3' to display the list.\n'4' to search for a node.\n'5' to exit.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

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

break;

default:

printf("\nPlease enter '1', '2', '3', '4' or '5' only and try again!");

}

}while(ch!=5);

}

void insert()

{

struct Node \*temp, \*ptr;

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

int pos, locn;

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

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

temp -> next = NULL;

printf("\nEnter:\n'1' to insert at beginning.\n'2' to insert at end.\n'3' to insert in between.\nEnter your choice: ");

scanf("%d", &pos);

if(temp == NULL)

{

printf("\nMemory allocation has been unsuccessful.");

}

else if(pos == 1)

{

if(head == NULL)

{

head = temp;

temp -> next = head;

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

temp -> next = head;

ptr -> next = temp;

head = temp;

}

printf("\nInsertion Successful!");

}

else if(pos == 2)

{

if(head == NULL)

{

head = temp;

temp -> next = head;

printf("\nInsertion Successful!");

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> next = temp;

temp -> next = head;

printf("\nInsertion Successful!");

}

}

else if(pos == 3)

{

printf("\nEnter the location for insertion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr = ptr -> next;

if(ptr == head)

{

printf("\nInsertion Unsuccessful.");

return;

}

}

temp -> next = ptr -> next;

ptr -> next = temp;

printf("\nInsertion Successful!");

}

}

void delete()

{

struct Node \*ptr, \*ptr1;

int pos, locn;

printf("\nEnter:\n'1' to delete at beginning.\n'2' to delete at end.\n'3' to delete in between.\nEnter your choice: ");

scanf("%d", &pos);

if(head == NULL)

{

printf("\nUnderflow!");

}

if(pos == 1)

{

if(head -> next == head)

{

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

free(head);

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> next = head -> next;

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

free(head);

head = ptr -> next;

}

}

else if(pos == 2)

{

if(head -> next == head)

{

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

head = NULL;

free(head);

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr1 = ptr;

ptr = ptr -> next;

}

ptr1 -> next = ptr -> next;

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

free(ptr);

}

}

else if(pos == 3)

{

printf("\nEnter the location of deletion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr1 = ptr;

ptr = ptr -> next;

if(ptr == head)

{

printf("\nDeletion Unsuccessful.");

return;

}

}

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

ptr1 -> next = ptr -> next;

free(ptr);

}

}

void search()

{

struct Node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nUnderflow!");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head -> data == item)

{

printf("item found at location %d ",i+1);

flag=0;

}

else

{

while (ptr -> next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n");

}

}

}

}

void display()

{

struct Node \*ptr;

ptr = head;

if(head == NULL)

{

printf("\nUnderflow!");

}

else

{

while(ptr -> next != head)

{

printf("%d\n", ptr -> data);

ptr = ptr -> next;

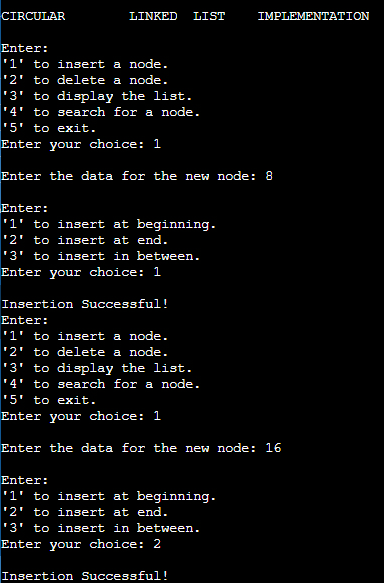
}

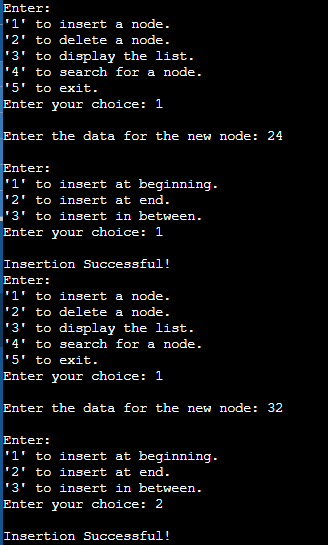
printf("%d\n", ptr -> data);

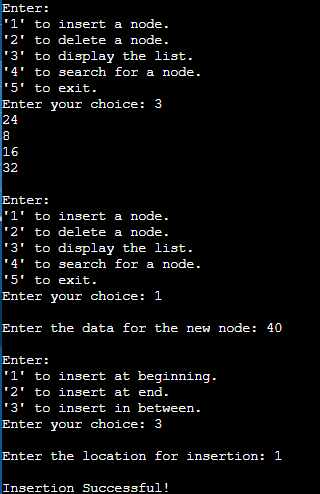
}

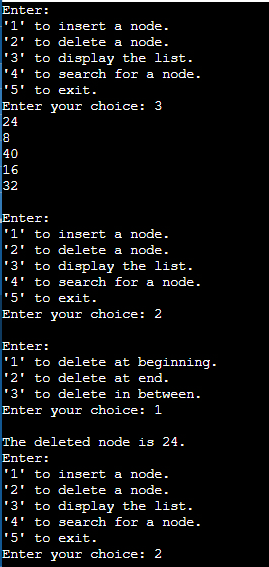
}

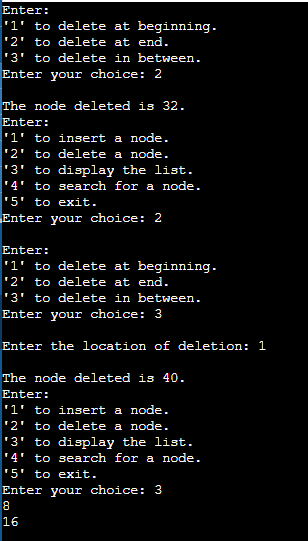
Output:

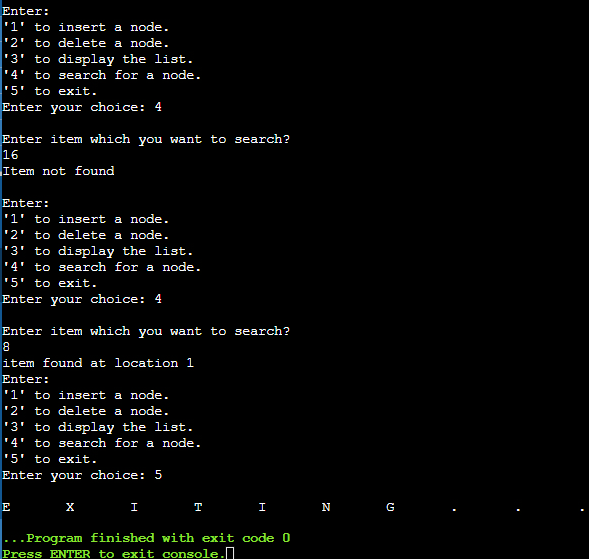












**Circular Doubly Linked List:**

Code:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*prev;

struct Node \*next;

};

struct Node \*head;

void insert();

void delete();

void display();

void search();

void main()

{

int ch;

printf("\nCIRCULAR\tDOUBLY\tLINKED\tLIST\tIMPLEMENTATION\n");

do

{

printf("\nEnter:\n'1' to insert a node.\n'2' to delete a node.\n'3' to display the list.\n'4' to search for a node.\n'5' to exit.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

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

break;

default:

printf("\nPlease enter '1', '2', '3', '4' or '5' only and try again!");

}

}while(ch!=5);

}

void insert()

{

struct Node \*temp, \*ptr;

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

int pos, locn;

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

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

temp -> next = NULL;

printf("\nEnter:\n'1' to insert at beginning.\n'2' to insert at end.\n'3' to insert in between.\nEnter your choice: ");

scanf("%d", &pos);

if(temp == NULL)

{

printf("\nMemory allocation has been unsuccessful.");

}

else if(pos == 1)

{

if(head == NULL)

{

head = temp;

temp -> prev = head;

temp -> next = head;

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> next = temp;

temp -> prev = ptr;

head -> prev = temp;

temp -> next = head;

head = temp;

}

printf("\nInsertion Successful!");

}

else if(pos == 2)

{

if(head == NULL)

{

head = temp;

temp -> prev = head;

temp -> next = head;

printf("\nInsertion Successful!");

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> next = temp;

temp -> prev = ptr;

temp -> next = head;

head -> prev = temp;

printf("\nInsertion Successful!");

}

}

else if(pos == 3)

{

printf("\nEnter the location for insertion: ");

scanf("%d", &locn);

ptr = head;

for(int i = 0; i < locn; i++)

{

ptr = ptr -> next;

if(ptr == head)

{

printf("\nInsertion Unsuccessful.");

return;

}

}

temp -> next = ptr -> next;

temp -> prev = ptr;

ptr -> next = temp;

ptr -> next -> prev = temp;

printf("\nInsertion Successful!");

}

}

void delete()

{

struct Node \*ptr, \*ptr1;

int pos, locn;

printf("\nEnter:\n'1' to delete at beginning.\n'2' to delete at end.\n'3' to delete in between.\nEnter your choice: ");

scanf("%d", &pos);

if(head == NULL)

{

printf("\nUnderflow!");

}

if(pos == 1)

{

if(head -> next == head)

{

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

free(head);

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> next = head -> next;

head -> next -> prev = ptr;

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

free(head);

head = ptr -> next;

}

}

else if(pos == 2)

{

if(head -> next == head)

{

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

free(head);

}

else

{

ptr = head;

while(ptr -> next != head)

{

ptr = ptr -> next;

}

ptr -> prev -> next = head;

head -> prev = ptr -> prev;

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

free(ptr);

}

}

else if(pos == 3)

{

int deldata;

printf("\nEnter the data after which the node should be deleted: ");

scanf("%d", &deldata);

ptr = head;

while(ptr -> data != deldata)

ptr = ptr -> next;

if(ptr -> next == head)

printf("\nDeletion Unsuccessful.");

else if(ptr -> next -> next == head)

ptr -> next = head;

else

{

ptr1 = ptr -> next;

ptr -> next = ptr1 -> next;

ptr1 -> next -> prev = ptr;

printf("\nThe deleted node is %d.", ptr1 -> data);

free(ptr1);

}

}

}

void search()

{

struct Node \*ptr;

int item, i=0, flag=1;

ptr = head;

if(ptr == NULL)

{

printf("\nUnderflow!");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head ->data == item)

{

printf("item found at location %d",i+1);

flag=0;

}

else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display()

{

struct Node \*ptr;

ptr = head;

if(head == NULL)

{

printf("\nUnderflow!");

}

else

{

while(ptr -> next != head)

{

printf("%d\n", ptr -> data);

ptr = ptr -> next;

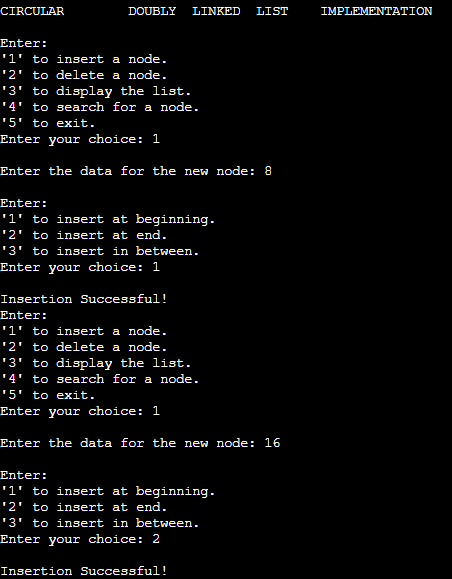
}

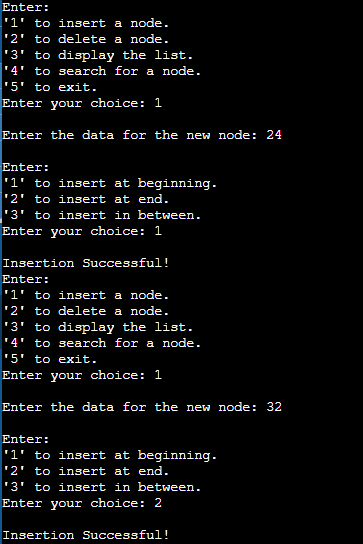
printf("%d\n", ptr -> data);

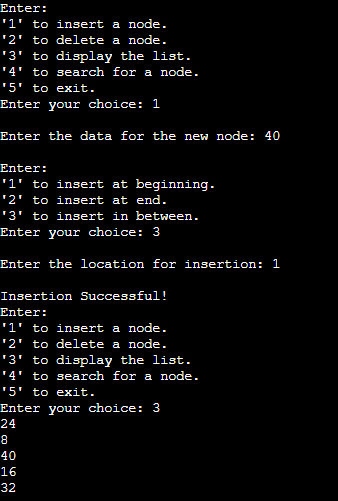
}

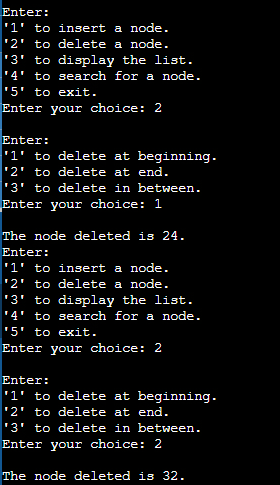
}

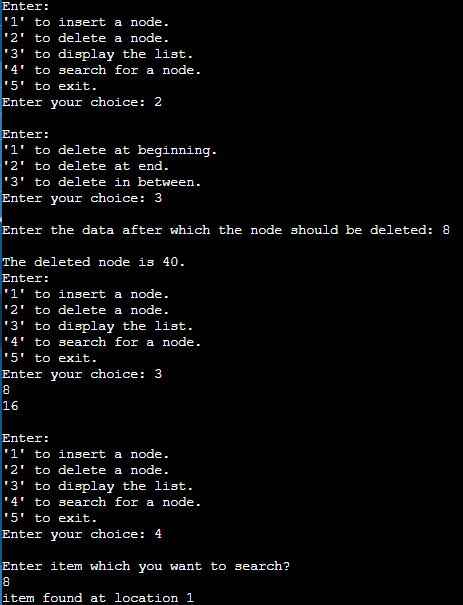
Output:

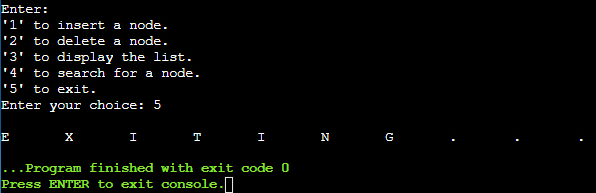












**Conclusion:-Post lab questions:**

1. Compare and contrast SLL and DLL

Ans.

|  |  |  |
| --- | --- | --- |
| Sr. No. | Singly Linked List | Doubly Linked List |
| 1. | It contains two fields – data field and next node address field. | It contains three fields – data field, previous node address field and next node address field. |
| 2. | It can be traversed in the forward direction only. | It can be traversed in the forward as well as reverse direction. |
| 3. | It occupies less memory than DLL. | It occupies more memory than SLL. |
| 4. | It is used for the execution of linear data structures like stacks. | It is used for the execution of non-linear data structures like trees. |