Batch: A3 **Roll No.: 16010121045**

Experiment No. 6

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

Title: Implementation of Linked List

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:

Introduction:

Define Linked List

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Types of linked list:

- Singly linked lists
- Doubly linked lists
- Circular linked lists

- Circular doubly linked lists

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

Singly Linked List

Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, set **newNode→next** = **NULL** and **head** = **newNode**.
- **Step 4 -** If it is **Not Empty** then, set **newNode→next** = **head** and **head** = **newNode**.

Inserting At End of the list

We can use the following steps to insert a new node at end of the single linked list...

- **Step 1 -** Create a **newNode** with given value and **newNode → next** as **NULL**.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**).
- **Step 3 -** If it is **Empty** then, set **head** = **newNode**.
- **Step 4 -** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5 -** Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
- **Step 6 -** Set **temp → next** = **newNode**.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, set **newNode** → **next** = **NULL** and **head** = **newNode**.
- **Step 4 -** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5 -** Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1** → **data** is equal to **location**, here location is the node value after which we want to insert the newNode).
- **Step 6 -** Every time check whether **temp** is reached to last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.
- **Step 7 -** Finally, Set '**newNode** → **next** = **temp** → **next**' and '**temp** → **next** = **newNode**'

Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Check whether list is having only one node (**temp** → **next** == **NULL**)
- **Step 5 -** If it is **TRUE** then set **head** = **NULL** and delete **temp** (Setting **Empty** list conditions)
- **Step 6 -** If it is **FALSE** then set **head** = **temp** → **next**, and delete **temp**.

Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

- **Step 3 -** If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2'** and initialize '**temp1**' with **head**.
- **Step 4 -** Check whether list has only one Node (**temp1 → next** == **NULL**)
- **Step 5 -** If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)
- **Step 6 -** If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next** == **NULL**)
- **Step 7 -** Finally, Set **temp2 → next** = **NULL** and delete **temp1**.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize '**temp1**' with **head**.
- **Step 4 -** Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the '**temp1**' to its next node.
- **Step 5 -** If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.
- **Step 6 -** If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
- **Step 7 -** If list has only one node and that is the node to be deleted, then set **head** = **NULL** and delete **temp1** (**free(temp1)**).
- **Step 8 -** If list contains multiple nodes, then check whether **temp1** is the first node in the list (**temp1 == head**).
- **Step 9 -** If **temp1** is the first node then move the **head** to the next node (**head = head → next**) and delete **temp1**.
- **Step 10 -** If **temp1** is not first node then check whether it is last node in the list (**temp1 → next == NULL**).
- **Step 11 -** If **temp1** is last node then set **temp2 → next** = **NULL** and delete **temp1** (**free(temp1)**).
- **Step 12 -** If **temp1** is not first node and not last node then set **temp2 → next** = **temp1 → next** and delete **temp1** (**free(temp1)**).

Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!!'** and terminate the function.

- **Step 3 -** If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Keep displaying **temp → data** with an arrow (**--->**) until **temp** reaches to the last node
- **Step 5 -** Finally display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

Doubly Linked List

Insert At Beginning

1. Start
2. Input the DATA to be inserted
3. Create a new node.
4. NewNode → Data = DATA NewNode →Lpoint =NULL
5. IF START IS NULL NewNode→ Rpoint = NULL
6. Else NewNode → Rpoint = START START→Lpoint = NewNode
7. START =NewNode
8. Stop

ii. Insertion at location:

1. Start
2. Input the DATA and POS
3. Initialize TEMP = START; i = 0
4. Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
5. TEMP = TEMP → RPoint; i = i +1
6. If (TEMP not equal to NULL) and (i equal to POS)

(a) Create a New Node

(b) NewNode → DATA = DATA

(c) NewNode → RPoint = TEMP → RPoint

(d) NewNode → LPoint = TEMP

(e) (TEMP → RPoint) → LPoint = NewNode

1. (f) TEMP → RPoint = New Node
2. Else

(a) Display "Position NOT found"

1. Stop

iii. Insert at End

1. Start
2. Input DATA to be inserted
3. Create a NewNode
4. NewNode → DATA = DATA
5. NewNode → RPoint = NULL
6. If (SATRT equal to NULL)

a. START = NewNode

b. NewNode → LPoint=NULL

1. Else

a. TEMP = START

b. While (TEMP → Next not equal to NULL)

i. TEMP = TEMP → Next

c. TEMP → RPoint = NewNode

d. NewNode → LPoint = TEMP

1. Stop

iv. Forward Traversal

1. Start
2. If (START is equal to NULL)

a) Display "The list is Empty"

b) Stop

1. Initialize TEMP = START
2. Repeat the step 5 and 6 until (TEMP == NULL)
3. Display "TEMP → DATA"
4. TEMP = TEMP → Next
5. Stop

v. Backward Traversal

1. Start
2. If (START is equal to NULL)
3. Display "The list is Empty"

4. Stop
5. Initialize TEMP = TAIL
6. Repeat the step 5 and 6 until (TEMP == NULL)
7. Display "TEMP → DATA"
8. TEMP = TEMP → Prev
9. Stop

Circular Linked List

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list...

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, set **head** = **newNode** and **newNode→next** = **head** .
- **Step 4 -** If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with '**head**'.
- **Step 5 -** Keep moving the '**temp**' to its next node until it reaches to the last node (until '**temp** → **next** == **head**').
- **Step 6 -** Set '**newNode** → **next** =**head**', '**head** = **newNode**' and '**temp** → **next** = **head**'.

Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**).
- **Step 3 -** If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.
- **Step 4 -** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5 -** Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp** → **next** == **head**).
- **Step 6 -** Set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the circular linked list...

- **Step 1 -** Create a **newNode** with given value.

- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.
- **Step 4 -** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5 -** Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1** → **data** is equal to **location**, here location is the node value after which we want to insert the newNode).
- **Step 6 -** Every time check whether **temp** is reached to the last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.
- **Step 7 -** If **temp** is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).
- **Step 8 -** If **temp** is last node then set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.
- **Step 8 -** If **temp** is not last node then set **newNode** → **next** = **temp** → **next** and **temp** → **next** = **newNode**.

Deletion

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the circular linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize both '**temp1**' and '**temp2**' with **head**.
- **Step 4 -** Check whether list is having only one node (**temp1** → **next** == **head**)
- **Step 5 -** If it is **TRUE** then set **head** = **NULL** and delete **temp1** (Setting **Empty** list conditions)
- **Step 6 -** If it is **FALSE** move the **temp1** until it reaches to the last node. (until **temp1** → **next** == **head**)
- **Step 7 -** Then set **head** = **temp2** → **next**, **temp1** → **next** = **head** and delete **temp2**.

Deleting from End of the list

We can use the following steps to delete a node from end of the circular linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2'** and initialize '**temp1**' with **head**.
- **Step 4 -** Check whether list has only one Node (**temp1** → **next** == **head**)
- **Step 5 -** If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate from the function. (Setting **Empty** list condition)
- **Step 6 -** If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until **temp1** reaches to the last node in the list. (until **temp1** → **next** == **head**)
- **Step 7 -** Set **temp2** → **next** = **head** and delete **temp1**.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the circular linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize '**temp1**' with **head**.
- **Step 4 -** Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the '**temp1**' to its next node.
- **Step 5 -** If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.
- **Step 6 -** If it is reached to the exact node which we want to delete, then check whether list is having only one node (**temp1** → **next** == **head**)
- **Step 7 -** If list has only one node and that is the node to be deleted then set **head** = **NULL** and delete **temp1** (**free(temp1)**).
- **Step 8 -** If list contains multiple nodes then check whether **temp1** is the first node in the list (**temp1 == head**).
- **Step 9 -** If **temp1** is the first node then set **temp2** = **head** and keep moving **temp2** to its next node until **temp2** reaches to the last node. Then set **head = head** → **next**, **temp2** → **nex**t = **head** and delete **temp1**.
- **Step 10 -** If **temp1** is not first node then check whether it is last node in the list (**temp1** → **next == head**).
- **Step 1 1-** If **temp1** is last node then set **temp2** → **next** = **head** and delete **temp1** (**free(temp1)**).

- **Step 12 -** If **temp1** is not first node and not last node then set **temp2 → next** = **temp1 → next** and delete **temp1** (**free(temp1)**).

Displaying a circular Linked List

We can use the following steps to display the elements of a circular linked list...

- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty**, then display **'List is Empty!!!'** and terminate the function.
- **Step 3 -** If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Keep displaying **temp → data** with an arrow (**--->**) until **temp** reaches to the last node
- **Step 5 -** Finally display **temp → data** with arrow pointing to **head → data**.

Doubly circular linked list

Inserting a new node at the beginning

```
Step 1: IF AVAIL = NULL
        Write OVERFLOW
        Go to Step 13
    [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7: SET PTR = PTR -> NEXT
    [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

Inserting a Node at the End

```
Step 1: IF AVAIL = NULL
        Write OVERFLOW
        Go to Step 12
    [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE > NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
```

Step 8: SET PTR = PTR -> NEXT
[END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: SET START -> PREV = NEW_NODE
Step 12: EXIT

Deleting the First Node

Step 1: IF START = NULL
Write UNDERFLOW
Go to Step 8
[END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4: SET PTR = PTR -> NEXT
[END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: SET START -> NEXT -> PREV = PTR
Step 7: FREE START
Step 8: SET START = PTR -> NEXT

Deleting the Last Node

Algorithm to delete the last node**
Step 1: IF START = NULL
Write UNDERFLOW
Go to Step 8
[END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4: SET PTR = PTR -> NEXT
[END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = START
Step 6: SET START -> PREV = PTR -> PREV

Step 7: FREE PTR
Step 8: EXIT

Implementation of an application using linked list:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// doubly linked list
struct node
{
  int data;
  struct node *next;
  struct node *prev;
};

struct node *front = NULL;
int isEmpty()
{
  if (front == NULL)
  {
    return 1;
  }
  return 0;
}
void insertend()
{
  int new_data;
  printf("Enter the data to be inserted: ");
  scanf("%d", &new_data);
  struct node *newnode = malloc(sizeof(struct node));
  struct node *ptr;
  newnode->data = new_data;
```

```
 newnode->prev = NULL;
 newnode->next = NULL;
 if (isEmpty() == 1)
 {
  front = newnode;
 }
 else
 {
  ptr = front;
  while (ptr->next != NULL)
  {
   ptr = ptr->next;
  }
  ptr->next = newnode;
  newnode->prev = ptr;
  newnode->next = NULL;
 }
}

void insertbegin()
{
 int new_data;
 printf("Enter the data to be inserted:");
 scanf("%d", &new_data);
 struct node *newnode = malloc(sizeof(struct node));
 newnode->data = new_data;
 newnode->prev = NULL;
 newnode->next = NULL;
 if (isEmpty() == 1)
```

```
 {
   front = newnode;
 }
 else
 {
   front->prev = newnode;
   newnode->next = front;
   front = newnode;
 }
}

void deletebegindoubly()
{
 struct node *temp;
 temp = front;
 temp = temp->next;
 if (isEmpty() == 1)
 {
   printf("The list is empty");
 }
 else
 {
   temp->prev = NULL;
 }
 free(temp);
}
void deleteenddoubly()
{
 struct node *temp;
```

```
  temp = front;
  while (temp->next != NULL)
  {
    temp = temp->next;
  }
  temp->next = NULL;
  temp->prev = NULL;
  free(temp);
}
void displaydoubly()
{
  struct node *ptr;
  if (front == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    ptr = front;
    printf("The list is: ");
    while (ptr != NULL)
    {
      printf("%d\t", ptr->data);
      ptr = ptr->next;
    }
  }
}
void searchdoub()
{
```

```
  int c;
  printf("Enter the element you want to search in the linked list:");
  scanf("%d", &c);
  struct node *p;
  p = front;
  int i = 1;
  while (p->data != c)
  {
    p = p->next;
    i++;
  }
  if (p->data == c)
  {
    printf("The position of the element %d in the list is: %d", p->data, i);
  }
  else
  {
    printf("The element is not in the list");
  }
}
// Circular linked list
struct Node
{
  int data1;
  struct Node *next;

} * list;

void insertbegincircular()
```

```
{
  int a;
  printf("Enter the data to be inserted: ");
  scanf("%d", &a);
  struct Node *p;
  struct Node *newnode1 = malloc(sizeof(struct Node));
  newnode1->data1 = a;
  if (list == NULL)
  {
    list = newnode1;
    newnode1->next = list;
  }
  else
  {
    p = list;
    while (p->next != list)
    {
      p = p->next;
    }
    p->next = newnode1;
    newnode1->next = list;
    list = newnode1;
  }
}
void inserAtendcircular()
{
  int b;
  struct Node *p, *q;
  printf("Enter the element you want to enter in the list:");
```

```
  scanf("%d", &b);
  p = (struct Node *)malloc(sizeof(struct Node));
  p->data1 = b;
  q = list;
  if (q == NULL)
  {
    list = p;
  }
  else
  {
    while (q->next != list)
    {
      q = q->next;
    }
    q->next = p;
  }
  p->next = list;
}
void deletebegincircular()
{
  struct Node *p;
  if (list == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    p = list;
    while (p->next != list)
```

```
    {
      p = p->next;
    }
    p->next = list->next;
    free(list);
    list = p->next;
  }
}
void deleteendcircular()
{
  struct Node *p, *q;
  if (list == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    p = list;
    while (p->next != list)
    {
      q = p;
      p = p->next;
    }
    q->next = p->next;
    free(p);
  }
}
void displaycircular()
{
```

```
  struct Node *p;
  if (list == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    p = list;
    printf("The list is: ");
    while (p->next != list)
    {
      printf("%d ", p->data1);
      p = p->next;
    }
    printf("%d", p->data1);
  }
}
void searchcirc()
{
  int h;
  printf("Enter the element you want to search in the linked list:");
  scanf("%d", &h);
  struct Node *p;
  p = list;
  int i = 1;
  while (p->data1 != h)
  {
    p = p->next;
    i++;
```

```
  }
  if (p->data1 == h)
  {
    printf("The position of the element %d in the list is: %d", p->data1, i);
  }
  else
  {
    printf("The element is not in the list");
  }
}

// Circular Doubly Linked List
struct Node1
{
  int data2;
  struct Node1 *next;
  struct Node1 *prev;

} * list1;
void insertbegincirdoub()
{
  int a;
  struct Node1 *newnode2 = malloc(sizeof(struct Node1));
  struct Node1 *ptr;
  printf("Enter the data to be inserted: ");
  scanf("%d", &a);
  newnode2->data2 = a;
  if (list1 == NULL)
  {
```

```
    list1 = newnode2;
    newnode2->next = list1;
  }
  else
  {
    ptr = list1;
    while (ptr->next != list1)
    {
      ptr = ptr->next;
    }
    newnode2->prev = ptr;
    ptr->next = newnode2;
    newnode2->next = list1;
    list1->prev = newnode2;
    list1 = newnode2;
  }
}
void insertAtendcirdoub()
{
  int b;
  struct Node1 *p, *q;
  printf("Enter the element you want to enter in the list:");
  scanf("%d", &b);
  p = (struct Node1 *)malloc(sizeof(struct Node1));
  p->data2 = b;
  if (list1 == NULL)
  {
    list1 = p;
  }
```

```
 else
 {
   q = list1;
   while (q->next != list1)
   {
     q = q->next;
   }
   q->next = p;
   p->prev = q;
   p->next = list1;
   list1->prev = p;
 }
}
void deletebegincirdoub()
{
 struct Node1 *p, *temp;
 if (list1 == NULL)
 {
   printf("The list is empty");
 }
 else
 {
   p = list1;
   while (p->next != list1)
   {
     p = p->next;
   }
   p->next = list1->next;
   temp = list1;
```

```
    list1 = list1->next;
    list1->prev = p;
    free(temp);
  }
}
void deleteendcirdoub()
{
  struct Node1 *p, *q;
  if (list1 == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    p = list1;
    while (p->next != list1)
    {
      q = p;
      p = p->next;
    }
    q->next = p->next;
    p->next->prev = p;
    free(p);
  }
}
void displaycirdoub()
{
  struct Node1 *p;
  if (list1 == NULL)
```

```
 {
   printf("The list is empty");
 }
 else
 {
   p = list1;
   printf("The list is:");
   while (p->next != list1)
   {
     printf("%d\t", p->data2);
     p = p->next;
   }
   printf("%d", p->data2);
 }
}
void searchcircdoub()
{
 int l;
 printf("Enter the element you want to search in the linked list:");
 scanf("%d", &l);
 struct Node1 *p;
 p = list1;
 int i = 1;
 while (p->data2 != l)
 {
   p = p->next;
   i++;
 }
 if (p->data2 == l)
```

```
 {
   printf("The position of the element %d in the list is: %d", p->data2, i);
 }
 else
 {
   printf("The element is not in the list");
 }
}
int main()
{
 int x, y, z, k;
 while (1)
 {
   printf("\nWhich linked list you want to use?\n1.Doubly Linked List\n2.Circular 
Linked Linked\n3.Circular Doubly Linked List\n4.Exit\nEnter the number in front the 
type of linked list to use the linked list:");
   scanf("%d", &x);
   switch (x)
   {
   case 1:
     while (y != 8)
     {
       printf("\n**************Doubly Linked List**************\n");
       printf("\n1.Insert at the begin\n2.Insert at the end\n3.Delete at the 
begin\n4.Delete at the end\n5.Traverse\n6.Search\n7.Exit\n");
       printf("Enter the number in front of the operation in Doubly Linked List:");
       scanf("%d", &y);
       if (y == 1)
       {
```

```
  insertbegin();
 }
 else if (y == 2)
 {
  insertend();
 }
 else if (y == 3)
 {
  deletebegindoubly();
 }
 else if (y == 4)
 {
  deleteenddoubly();
 }
 else if (y == 5)
 {
  displaydoubly();
 }
 else if (y == 6)
 {
  searchdoub();
 }
 else if (y == 7)
 {
  break;
 }
 else
 {
  printf("Invalid option");
```

```
      }
    }
    break;
  case 2:
    while (z != 8)
    {
      printf("\n**************Circular Linked List**************\n");
      printf("\n1.Insert at the begin\n2.Insert at the end\n3.Delete at the
begin\n4.Delete at the end\n5.Traverse\n6.Search\n7.Exit\n");
      printf("Enter the number in front of the operation in Circular Linked List:");
      scanf("%d", &z);
      if (z == 1)
      {
        insertbegincircular();
      }
      else if (z == 2)
      {
        inserAtendcircular();
      }
      else if (z == 3)
      {
        deletebegincircular();
      }
      else if (z == 4)
      {
        deleteendcircular();
      }
      else if (z == 5)
      {
```

```
          displaycircular();
        }
        else if (z == 6)
        {
          searchcirc();
        }
        else if (z == 7)
        {
          break;
        }
        else
        {
          printf("Invalid option");
        }
      }
      break;
    case 3:
      while (k != 8)
      {
        printf("\n**************Circular Doubly Linked List**************\n");
        printf("\n1.Insert at the begin\n2.Insert at the end\n3.Delete at the
begin\n4.Delete at the end\n5.Traverse\n6.Search\n7.Exit\n");
        printf("Enter the number in front of the operation in Circular Doubly Linked
List:");
        scanf("%d", &k);
        if (k == 1)
        {
          insertbegincirdoub();
        }
```

```
    else if (k == 2)
    {
     insertAtendcirdoub();
    }
    else if (k == 3)
    {
     deletebegincirdoub();
    }
    else if (k == 4)
    {
     deleteendcirdoub();
    }
    else if (k == 5)
    {
     displaycirdoub();
    }
    else if (k == 6)
    {
     searchcircdoub();
    }
    else if (k == 7)
    {
     break;
    }
    else
    {
     printf("Invalid option");
    }
  }
```

```
     break;
    case 4:
     printf("Exiting the program............");
     exit(1);
     break;
    default:
     printf("Invalid option");
     break;
   }
  }

  return 0;
}
```

Output:

```
Which linked list you want to use?
1.Doubly Linked List
2.Circular Linked Linked
3.Circular Doubly Linked List
4.Exit
Enter the number in front the type of linked list to use the linked list:1

**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:1
Enter the data to be inserted:1

**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:2
Enter the data to be inserted: 2
```

```
**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:1
Enter the data to be inserted:10

**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:5
The list is: 10 1       2
**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:6
Enter the element you want to search in the linked list:1
The position of the element 1 in the list is: 2
```

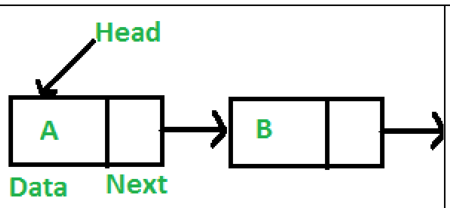
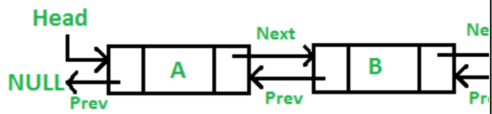
```
**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:4

**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:5
The list is: 10 1       2
**************Doubly Linked List**************

1.Insert at the begin
2.Insert at the end
3.Delete at the begin
4.Delete at the end
5.Traverse
6.Search
7.Exit
Enter the number in front of the operation in Doubly Linked List:7

Which linked list you want to use?
1.Doubly Linked List
2.Circular Linked Linked
3.Circular Doubly Linked List
4.Exit
```

Conclusion:- Hence we successfully implemented various type of Linked Lists.

Post lab questions:

1. Compare and contrast SLL and DLL

Singly linked list (SLL)	Doubly linked list (DLL)	
SLL nodes contains 2 field -data field and next link field.	DLL nodes contains 3 fields -data field, a previous link field and a next link field.	

	
In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.	In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).
The SLL occupies less memory than DLL as it has only 2 fields.	The DLL occupies more memory than SLL as it has 3 fields.
Complexity of insertion and deletion at a given position is O(n).	Complexity of insertion and deletion at a given position is O(n / 2) = O(n) because traversal can be made from start or from the end.
Complexity of deletion with a given node is O(n), because the previous node needs to be known, and traversal takes O(n)	Complexity of deletion with a given node is O(1) because the previous node can be accessed easily
We mostly prefer to use singly linked list for the execution of stacks.	We can use a doubly linked list to execute heaps and stacks, binary trees.
When we do not need to perform any searching operation and we want to save memory, we prefer a singly linked list.	In case of better implementation, while searching, we prefer to use doubly linked list.
A singly linked list consumes less memory as compared to the doubly linked list.	The doubly linked list consumes more memory as compared to the singly linked list.