Experiment 1.4

Student Name: Ruchika Raj **UID:20BCS9285**
Branch: CSE **Section/Group: 615-B**
Semester: 5TH **Date of Performance:06/09/22**
Subject Name: DAA LAB

1. Aim/Overview of the practical

Code to Insert and Delete an element at the beginning and at end in Doubly and

Circular Linked List

2. Task to be done/ Which logistics used:

Task: Insert and delete a node from the doubly and circular linked list Logic

Used:

For insertion in doubly linked list.

At beginning:

The next pointer of the new Node is referenced to the head node and its previous pointer is referenced to NULL. The previous pointer of the head node is referenced to the new Node. The new Node is then made as the head node.

At the end:

Make the next pointer of the last node to point to the new Node. The next pointer of the new Node is referenced to NULL and its previous pointer is made to point to the last node. Then, the new Node is made as the last node.

For deletion in doubly linked list.

At beginning:

Copy the head node in some temporary node. Make, the second node as the head node. The prev pointer of the head node is referenced to NULL. Delete the temporary node **At the end:**

Copy the last node to a temporary node. Shift the last node to the second last position. Make, the last node's next pointer as NULL. Delete the temporary node.

For insertion in circular linked list.

At beginning:

Store the address of the current first node in the new Node (i.e. pointing the new Node to the current first node). Point the last node to new Node (i.e making new Node as head) **At the end:**

store the address of the head node to next of new Node (making new Node the last node). Point the current last node to new Node. Make new Node as the last node.

Platform Used : Online Compiler

3. Algorithm/Flowchart (For programming based labs)**:**

1. **Doubly Linked List :**

INSERTION:

a) **At the beginning**

Algorithm:

1. Start the program.

2. Allocate the space for the new node in the memory. This will be done by using the following statement.

3. Check whether the list is empty or not. The list is empty if the condition head == NULL holds. In that case, the node will be inserted as the only node of the list and therefore the prev and the next pointer of the node will point to NULL and the head pointer will point to this node.

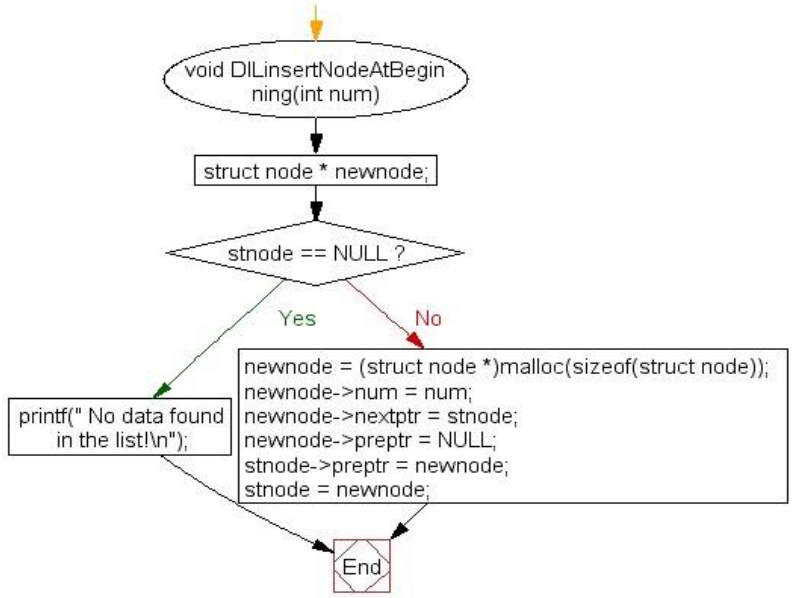
4. In the second scenario, the condition head == NULL become false and the node will be inserted in beginning. The next pointer of the node will point to the existing head pointer of the node. The prev pointer of the existing head will point to the new node being inserted.

5. Since, the node being inserted is the first node of the list and therefore it must contain NULL in its prev pointer. Hence assign null to its previous part and make the head point to this node.

6. Print the linked list with the added node.

7. End the program.

Flow Chart:



Pseudo Code:

- Step 1: BEGIN
- Step 2: IF ptr = NULL Write OVERFLOW Go to Step 9 [END OF IF]
- Step 3: SET NEW_NODE = ptr
- Step 4: SET ptr = ptr -> NEXT
- Step 5: SET NEW_NODE -> DATA = VAL
- Step 6: SET NEW_NODE -> PREV = NULL
- Step 7: SET NEW_NODE -> NEXT = START
- Step 8: SET head -> PREV = NEW_NODE
- Step 9: SET head = NEW_NODE
- Step 10: EXIT

b) At end

Algorithm:

1) Start the program.

2) Allocate the memory for the new node. Make the pointer ptr point to the new node being inserted.

3) Check whether the list is empty or not. The list is empty if the condition head == NULL holds. In that case, the node will be inserted as the only node of the list and therefore the prev and the next pointer of the node will point to NULL and the head pointer will point to this node.

4) In the second scenario, the condition head == NULL become false. The new node will be inserted as the last node of the list. For this purpose, we have to traverse the whole list in order to reach the last node of the list. Initialize the pointer temp to head and traverse the list by using this pointer

5) The pointer temp point to the last node at the end of this while loop. Now, we just need to make a few pointer adjustments to insert the new node ptr to the list. First, make the next pointer of temp point to the new node being inserted i.e. ptr.

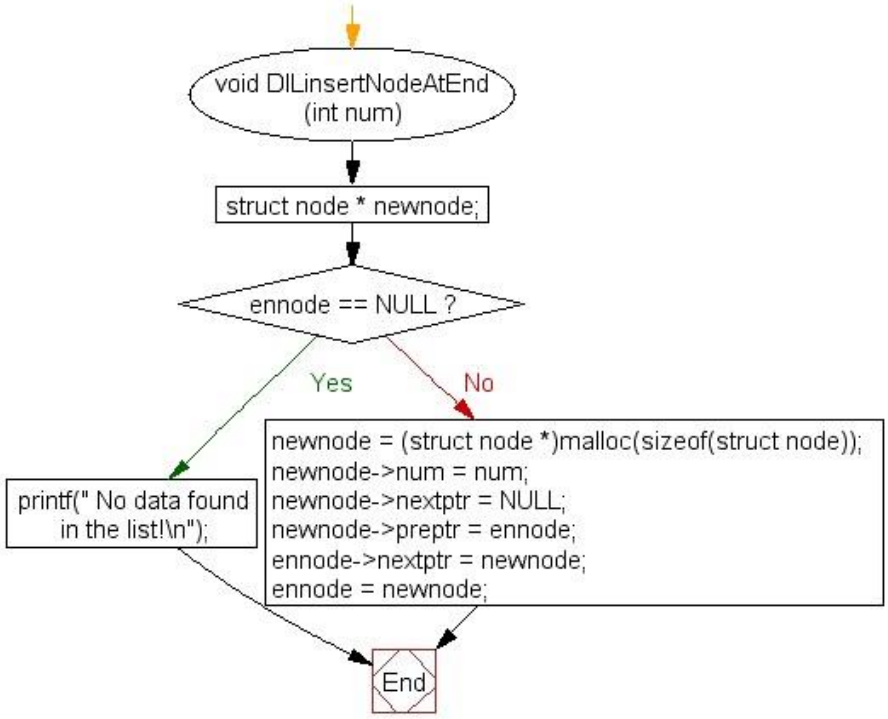
6) Make the previous pointer of the node ptr point to the existing last node of the list i.e. temp.

7) Make the next pointer of the node ptr point to the null as it will be the new last node of the list.

8) Print the linked list with the added node.

9) End the program.

Flow Chart:



Pseudo Code:

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

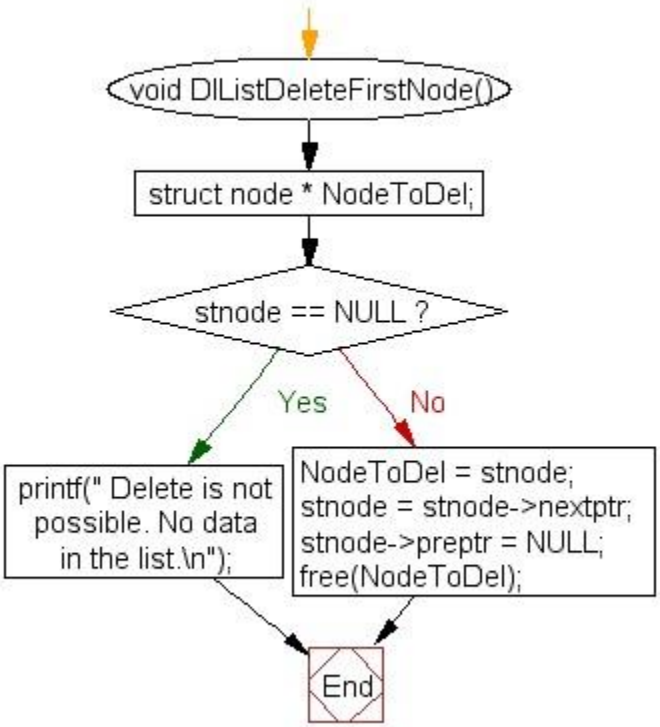
DELETION :

a) At the beginning

Algorithm:

1) Start the program.
2) If the node temp which we want to delete is NULL or the head is NULL, we will simply return.
3) Then we will check if temp is a head node or not.
4) If temp is the head node, then we will move next step.
5) We just need to copy the head pointer to pointer ptr and shift the head pointer to its next. Ptr = head;
   head = head → next;
6) Now make the prev of this new head node point to NULL.
7) Now free the pointer ptr by using the free function.
8) Print the linked list with the added node.
9) End the program.

Flowchart:



Pseudo Code:

- Step 1: BEGIN
- STEP 2: IF HEAD = NULL WRITE UNDERFLOW GOTO STEP 6
- STEP 3: SET PTR = HEAD
- STEP 4: SET HEAD = HEAD → NEXT
- STEP 5: SET HEAD → PREV = NULL
- STEP 6: FREE PTR
- STEP 7: EXIT

b) At end

Algorithm:

1) Start the program.

2) If the existed list is already empty then the condition head == NULL will become true and therefore the operation cannot be carried on.

3) If there is only one node in the list then the condition head → next == NULL become true. In this case, we just need to assign the head of the list to NULL and free head in order to completely delete the list.

4) Otherwise, just traverse the list to reach the last node of the list. This will be done by using the following statements.

```
ptr = head;
if(ptr->next != NULL)
{   ptr = ptr -> next;
}
```

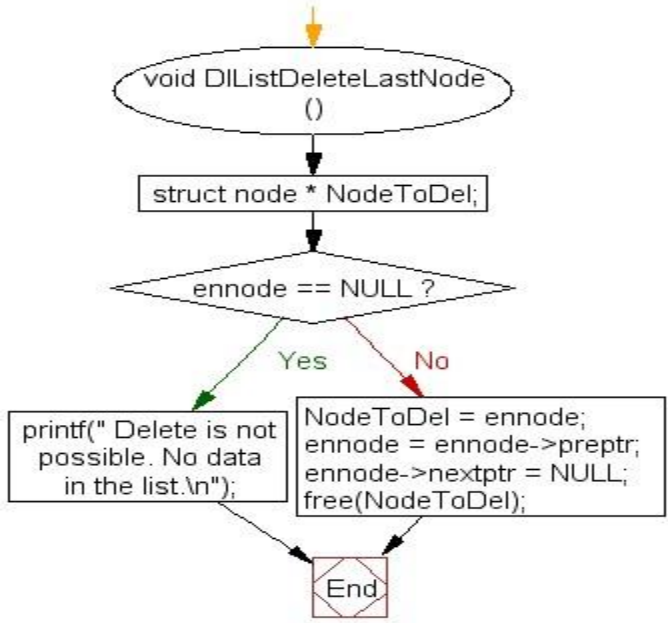
5) The ptr would point to the last node of the ist at the end of the for loop. Just make the next pointer of the previous node of ptr to NULL.

6) Free the pointer as this the node which is to be deleted. Command will be free(ptr).

7) Print the doubly linked list with the updated nodes.

8) End the program.

Flowchart:



Pseudo Code:

- Step 1: BEGIN
- Step 2: IF HEAD = NULL Write UNDERFLOW Go to Step 7 [END OF IF]
- Step 3: SET TEMP = HEAD
- Step 4: REPEAT STEP 4 WHILE TEMP->NEXT != NULL
- Step 5: SET TEMP = TEMP->NEXT [END OF LOOP]
- Step 6: SET TEMP ->PREV-> NEXT = NULL
- Step 7: FREE TEMP
- Step 8: EXIT

2. Circular Linked List

INSERTION :

a) At the beginning

Algorithm:

Start the program.

1) Firstly, allocate the memory space for the new node.

2) In the first scenario, the condition head == NULL will be true. Since, the list in which, we are inserting the node is a circular singly linked list, therefore the only node of the list (which is just inserted into the list) will point to itself only. We also need to make the head pointer point to this node.

3) In the second scenario, the condition head == NULL will become false which means that the list contains at least one node. In this case, we need to traverse the list in order to reach the last node of the list.

4) At the end of the loop, the pointer temp would point to the last node of the list. Since, in a circular singly linked list, the last node of the list contains a pointer to the first node of the list.

5) Therefore, we need to make the next pointer of the last node point to the head node of the list and the new node which is being inserted into the list will be the new head node of the list therefore the next pointer of temp will point to the new node ptr.

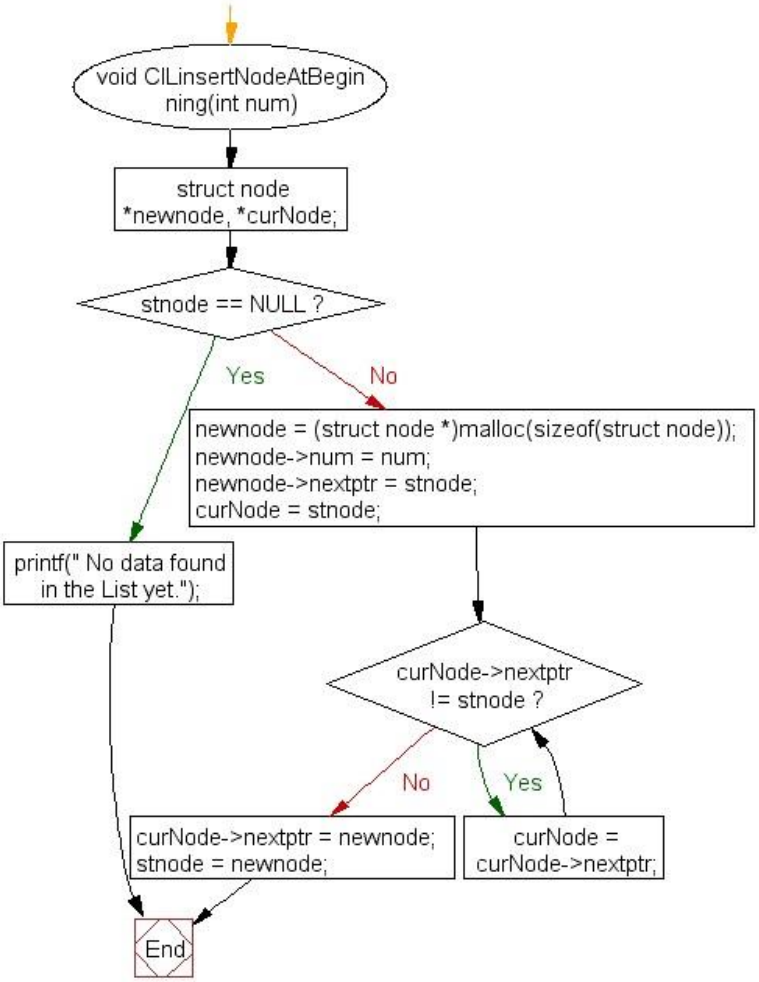
6) The next pointer of temp will point to the existing head node of the list.

7) Now, make the new node ptr, the new head node of the circular singly linked list.

8) Print the updated list.

9) End the program.

Flowchart:



Pseudo Code:

Step 1: IF PTR = NULL
Write OVERFLOW
Go to Step 11
[END OF IF]

Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET TEMP = HEAD
Step 6: Repeat Step 8 while TEMP -> NEXT != HEAD

Step 7: SET TEMP = TEMP -> NEXT
[END OF LOOP]

Step 8: SET NEW_NODE -> NEXT = HEAD
Step 9: SET TEMP → NEXT = NEW_NODE
Step 10: SET HEAD = NEW_NODE
Step 11: EXIT

b) At end

Algorithm:

1) Start the program.

2) Firstly, allocate the memory space for the new node

3) In the first scenario, the condition head == NULL will be true. Since, the list in which, we are inserting the node is a circular singly linked list, therefore the only node of the list (which is just inserted into the list) will point to itself only. We also need to make the head pointer point to this node. This will be done by using the following statements.

4) In the second scenario, the condition head == NULL will become false which means that the list contains at least one node. In this case, we need to traverse the list in order to reach the last node of the list.

5) At the end of the loop, the pointer temp would point to the last node of the list. Since, the new node which is being inserted into the list will be the new last node of the list.

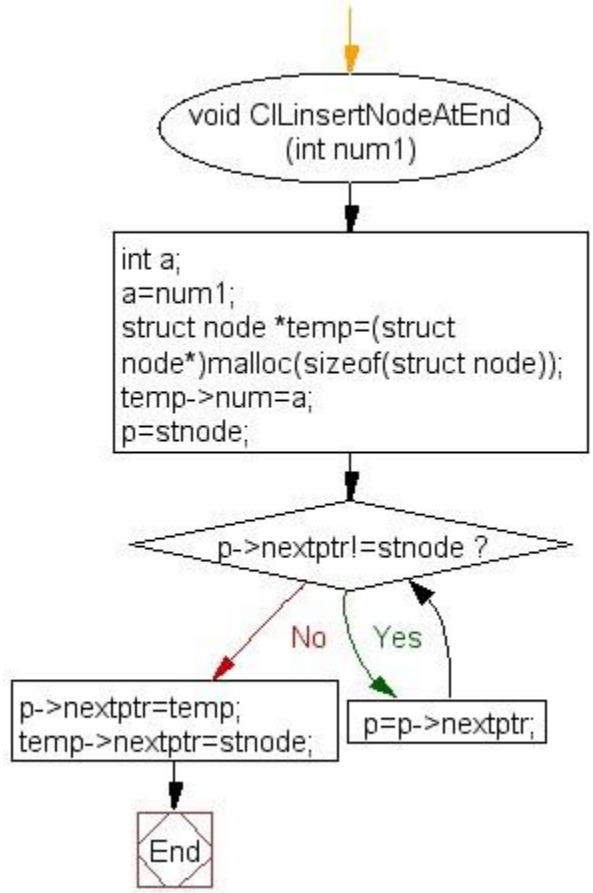
6) Therefore, the existing last node i.e. temp must point to the new node ptr.

7) The new last node of the list i.e. ptr will point to the head node of the list.

8) Print the updated list.

9) End the program.

Flowchart:



Pseudo Code:

- Step 1: IF PTR = NULL
  Write OVERFLOW
  Go to Step 1

[END OF IF]

- Step 2: SET NEW_NODE = PTR
- Step 3: SET PTR = PTR -> NEXT
- Step 4: SET NEW_NODE -> DATA = VAL
- Step 5: SET NEW_NODE -> NEXT = HEAD
- Step 6: SET TEMP = HEAD
- Step 7: Repeat Step 8 while TEMP -> NEXT != HEAD
- Step 8: SET TEMP = TEMP -> NEXT
  [END OF LOOP]

- Step 9: SET TEMP -> NEXT = NEW_NODE
- Step 10: EXIT

DELETION :

a) At the beginning

Algorithm:

1) Start the program.

2) Firstly, we will check the number of node in the list and proceed further accordingly.

3) If the list is empty then the condition head == NULL will become true, in this case, we just need to print underflow on the screen and make exit.

4) If the list contains single node then, the condition head → next == head will become true. In this case, we need to delete the entire list and make the head pointer free.

5) If the list contains more than one node then, in that case, we need to traverse the list by using the pointer ptr to reach the last node of the list.

6) At the end of the loop, the pointer ptr point to the last node of the list. Since, the last node of the list points to the head node of the list. Therefore, this will be changed as now, the last node of the list will point to the next of the head node.

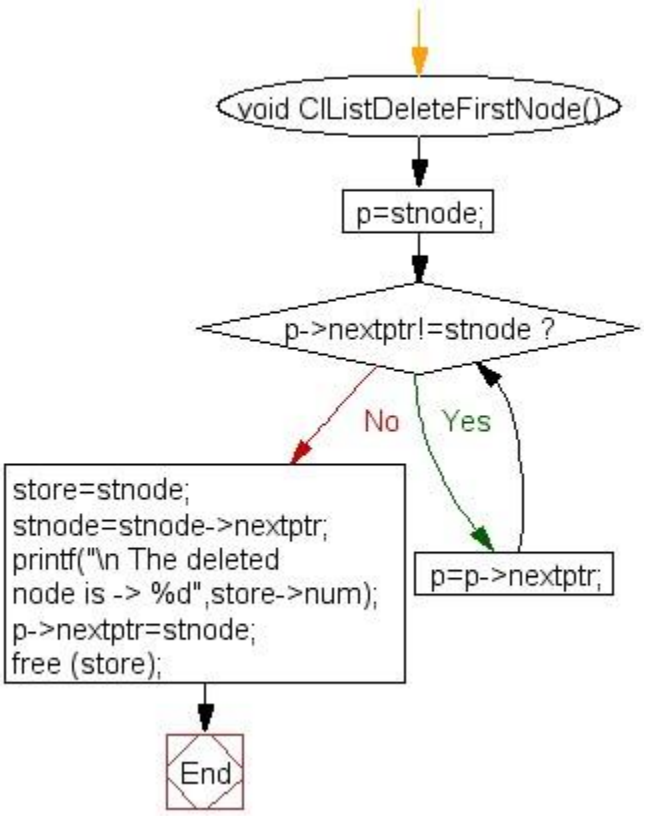
7) Now, free the head pointer by using the free() method.

8) Make the node pointed by the next of the last node, the new head of the list.

9) Print the updated list.

10) End the program.

Flowchart:



Pseudo Code:

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

b) At end

Algorithm:

1) Start the program.

2) Firstly, we will check the number of node in the list and proceed further accordingly.

3) If the list is empty then the condition head == NULL will become true, in this case, we just need to print underflow on the screen and make exit.

4) If the list contains single node then, the condition head → next == head will become true. In this case, we need to delete the entire list and make the head pointer free.

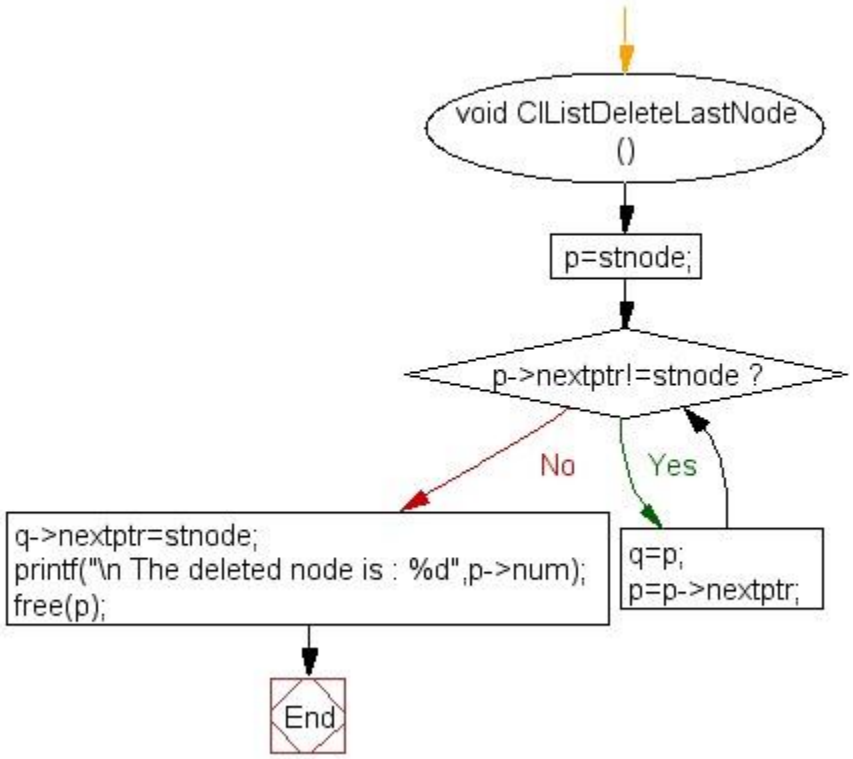
5) If the list contains more than one element, then in order to delete the last element, we need to reach the last node. We also need to keep track of the second last node of the list. For this purpose, the two pointers ptr and preptr are defined.

6) now, we need to make just one more pointer adjustment. We need to make the next pointer of preptr point to the next of ptr (i.e. head) and then make pointer ptr free.

7) Print the updated list.

8) End the program.

Flowchart:



Pseudo Code:

- Step 1: IF HEAD = NULL
  Write UNDERFLOW

Go to Step 8
[END OF IF]

- Step 2: SET PTR = HEAD
- Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != HEAD
- Step 4: SET PREPTR = PTR
- Step 5: SET PTR = PTR -> NEXT
  [END OF LOOP]
- Step 6: SET PREPTR -> NEXT = HEAD
- Step 7: FREE PTR
- Step 8: EXIT

3. Steps for experiment/practical/Code:

For Insertion of a node in Doubly Linked List:

```
#include <bits/stdc++.h>
using namespace std;

class node { public:
        node* prev;
int data;       node*
next;

        node(int value)
        {
           // A constructor is called here
                prev = NULL;
        data = value;
next = NULL;
        }
};

void insert_at_head(node*& head, int value)
```

```
{

        node* n = new node(value);
        n->next = head;

        if (head != NULL) {
                head->prev = n;
        }

        head = n;
}

void insert_at_tail(node*& head, int value)
{

        if (head == NULL) {
                insert_at_head(head, value);
                return;
        }

        node* n = new node(value);
        node* temp = head;

        while (temp->next != NULL) {
                temp = temp->next;
        }
        temp->next = n;     n-
>prev = temp;
}

void display(node* head)
{
```

```
        node* temp = head;        while
(temp != NULL) {                  cout <<
temp->data << " --> ";
                temp = temp->next;
        }
        cout << "NULL" << endl;
}

// Driver code int
main()
{
        node* head
                = NULL;

        insert_at_tail(head, 11);
insert_at_tail(head, 22);   insert_at_tail(head, 33);
insert_at_tail(head, 44);   insert_at_tail(head, 55);
cout << "After insertion at tail: ";
        display(head);

        cout << "After insertion at head: ";
        insert_at_head(head, 10);

        display(head);
        return 0;
```

For Deletion of a node in doubly Linked List:

```
#include <iostream> using

namespace std;
```

```
//node structure
struct   Node   {
int data;
   Node* next;
   Node* prev;
};

class LinkedList {
private:
   Node* head;
public:
   LinkedList(){ head
     = NULL;
   }

   //Add new element at the end of the list
void push_back(int newElement) {
Node* newNode = new Node();
newNode->data = newElement;
newNode->next = NULL;       newNode-
```

```
>prev = NULL;       if(head == NULL) {
head = newNode;
    } else {
      Node* temp = head;
while(temp->next != NULL)
temp = temp->next;       temp->next
= newNode;       newNode->prev =
temp;
    }
  }

  //Delete first node of the list
void pop_front() { if(head !=
NULL) {
      Node* temp = head;
head = head->next;
free(temp);       if(head !=
NULL)       head->prev
= NULL;
    }
  }
```

```
   void pop_back() {
if(head != NULL) {
if(head->next == NULL) {
head = NULL;
       }
else {
         Node* temp = head;
while(temp->next->next != NULL)
temp = temp->next;          Node*
lastNode = temp->next;          temp-
>next = NULL;          free(lastNode);
}
     }
   }


   //display the content of the list
void PrintList() {       Node*
temp = head;       if(temp !=
NULL) {         cout<<"The list
contains: ";         while(temp !=
NULL) {          cout<<temp-
```

```
>data<<" ";          temp = temp-
>next;
       }
       cout<<endl;       } else {
cout<<"The list is empty.\n";
     }
   }
};

// test the code  int
main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
MyList.push_back(20);   MyList.push_back(30);
  MyList.push_back(40);
  MyList.PrintList();

  //Delete the first node
  MyList.pop_front();
  MyList.PrintList();
```

```
  //Delete the last node
  MyList.pop_back();
  MyList.PrintList();

  return 0;
}
```

For Insertion of a node in Circular Linked List:

```
#include <iostream> using
namespace std;

//node structure
struct Node {
int data;
    Node* next;
};

class LinkedList {
private:
    Node* head;
public:
```

```
LinkedList(){
head = NULL;
   }

   //Add new element at the start of the list
void push_front(int newElement) {
Node* newNode = new Node();
newNode->data = newElement;
newNode->next = NULL;       if(head ==
NULL) {        head = newNode;
newNode->next = head;
     } else {
       Node* temp = head;
while(temp->next != head)
temp = temp->next;        temp-
>next = newNode;
newNode->next = head;        head
= newNode;
     }
   }
```

```
   //Add new element at the end of the list
void push_back(int newElement) {
Node* newNode = new Node();
newNode->data = newElement;
newNode->next = NULL;       if(head ==
NULL) {        head = newNode;
newNode->next = head;
     } else {
       Node* temp = head;
while(temp->next != head)
temp = temp->next;        temp->next
= newNode;        newNode->next =
head;

     }
   }

   //display the content of the list
void PrintList() {       Node*
temp = head;       if(temp !=
NULL) {        cout<<"The list
contains: ";        while(true) {
```

```
cout<<temp->data<<" ";
temp = temp->next;
if(temp == head)              break;
        }
        cout<<endl;       } else {
cout<<"The list is empty.\n";
      }
    }
};

// test the code  int
main() {
  LinkedList MyList;

  //Add three elements at the start of the list.
  MyList.push_front(10);
MyList.push_front(20);
  MyList.push_front(30);
  MyList.PrintList();

  MyList.push_back(1);
MyList.push_back(2);
```

```
  MyList.push_back(3);
  MyList.PrintList();


  return 0;
}
```

For Deletion of a node in Circular Linked List:

```
#include <iostream> using
namespace std;


//node structure
struct Node {
int data;
    Node* next;
};


class LinkedList {
private:
    Node* head;
public:
LinkedList(){
head = NULL;
    }
```

```
   //Add new element at the end of the list
void push_back(int newElement) {
Node* newNode = new Node();
newNode->data = newElement;
newNode->next = NULL;       if(head ==
NULL) {         head = newNode;
newNode->next = head;
     } else {
       Node* temp = head;
while(temp->next != head)
temp = temp->next;        temp->next
= newNode;        newNode->next =
head;
     }
   }

   //Delete first node of the list
void pop_front() {       if(head
!= NULL) {        if(head-
>next == head) {          head
= NULL;
```

```
        } else {
          Node* temp = head;
Node* firstNode = head;
while(temp->next != head) {
temp = temp->next;
          }
          head = head->next;
temp->next = head;
free(firstNode);
        }
      }
    }

    void pop_back() {
if(head != NULL) {
if(head->next == head) {
head = NULL;
        } else {
          Node* temp = head;
while(temp->next->next != head)
temp = temp->next;          Node*
```

```
lastNode = temp->next;          temp-
>next = head;          free(lastNode);
        }
      }
    }

   //display the content of the list
void PrintList() {
      Node* temp = head;
      if(temp != NULL) {
cout<<"The list contains: ";
while(true) {
cout<<temp->data<<" ";
temp = temp->next;
if(temp == head)              break;
        }
        cout<<endl;       } else {
cout<<"The list is empty.\n";
      }
    }
};
```

```
// test the code  int
main() {
  LinkedList MyList;

  //Add four elements in the list.
  MyList.push_back(10);
MyList.push_back(20);   MyList.push_back(30);
  MyList.push_back(40);
  MyList.PrintList();

  //Delete the first node
  MyList.pop_front();
  MyList.PrintList();

  //Delete the last node
  MyList.pop_back();
  MyList.PrintList();

  return 0;
}
```

6. Result/Output/Writing Summary:

Insertion in doubly Linked list

```
Output

/tmp/Cmxu8UpcII.o
After insertion at tail: 11 --> 22 --> 33 --> 44 --> 55 --> NULL
After insertion at head: 10 --> 11 --> 22 --> 33 --> 44 --> 55 --> NULL
```

Deletion in doubly Linked list

```
Output

/tmp/Cmxu8UpcII.o
The list contains: 10 20 30 40
The list contains: 20 30 40
The list contains: 20 30
```

Insertion in Circular Linked list

```
Output

/tmp/Cmxu8UpcII.o
The list contains: 30 20 10
The list contains: 30 20 10 1 2 3
```

Deletion in Circular Linked list

```
Output

/tmp/Cmxu8UpcII.o
The list contains: 10 20 30 40
The list contains: 20 30 40
The list contains: 20 30
```

Learning outcomes (What I have learnt):

1. Learnt about the doubly linked list.
2. Learnt about the circular linked list.
3. Learnt about the insertion of a node at different position in doubly and circular linked list.
4. Learnt about the deletion of a node at different position in doubly and circular linked list.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			