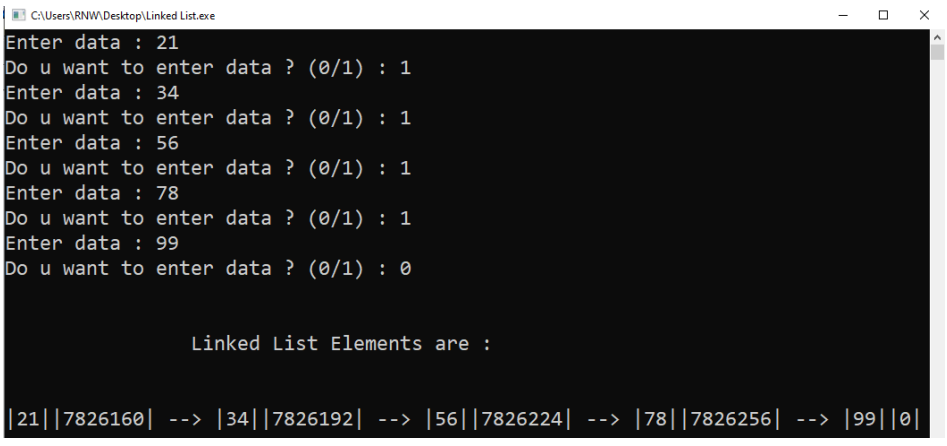
VIDHYADEEP UNIVERSITY
Holy Flame Of Knowledge

VIDHYADEEP UNIVERSITY INSTITUTE OF B.Sc. IT & BCA			
NAME :-			
SUBJECT :-		**ENROLLMENT :-**	
SUBMIT DATE :-		**DEPARTMENT :-**	
SR NO	PROBLEMS	DATE	SIGN
1	WAP TO SINGLY LINKED LIST.		
2	WAP TO DOUBLY LINKED LIST.		
3	WAP TO CIRCULAR LINKED LIST.		
4	WAP TO CIRCULAR DOUBLY LINKED LIST.		

❖ PROGRAM 1: WAP TO SINGLY LINKED LIST.

```
#include<stdio.h>
struct node{
	int data;
	struct node *next;
};
//Implementation of Linked List
int main(){
	struct node *head=0, *newnode, *tmp;
	int ch;
		do{
		printf("Do u want to enter data ? (0/1) : ");
		scanf("%d",&ch);

			if(ch==1){
				newnode=(struct node *) malloc(sizeof(struct node));

				printf("Enter data : ");
				scanf("%d",&newnode->data);
					newnode->next=0;

					if(head==0){
						head=tmp=newnode;
					}
					else{
						tmp->next=newnode;
						tmp=newnode;
					}
			}
	}while(ch!=0);
	//Travertion of Linked List
	tmp=head;
	printf("\n\n\t\tLinked List Elements are : \n\n\n");
	while(tmp!=0){
		printf("|%d||%u| --> ",tmp->data,tmp->next);
		tmp=tmp->next;
	}
}
```

➢ **Output:**



❖ PROGRAM 2: WAP TO DOUBLY LINKED LIST.

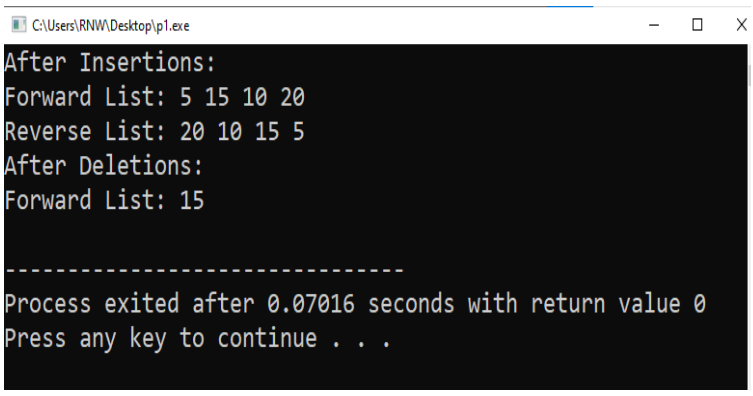
```
#include <stdio.h>
#include <stdlib.h>

// defining a node
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

// Function to create a new node with given value as data
Node* createNode(int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

// Function to insert a node at the beginning
void insertAtBeginning(Node** head, int data)
{
    // creating new node
    Node* newNode = createNode(data);

    // check if DLL is empty
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
}

// Function to insert a node at the end
void insertAtEnd(Node** head, int data)
{
    // creating new node
    Node* newNode = createNode(data);

    // check if DLL is empty
    if (*head == NULL) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
```

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

// Function to insert a node at a specified position
void insertAtPosition(Node** head, int data, int position)
{           int i;
    if (position < 1) {
        printf("Position should be >= 1.\n");
        return;
    }

    // if we are inserting at head
    if (position == 1) {
        insertAtBeginning(head, data);
        return;
    }
    Node* newNode = createNode(data);
    Node* temp = *head;
    for (i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf(
            "Position greater than the number of nodes.\n");
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}

// Function to delete a node from the beginning
void deleteAtBeginning(Node** head)
{
    // checking if the DLL is empty
    if (*head == NULL) {
        printf("The list is already empty.\n");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    free(temp);    }
```

```
// Function to delete a node from the end
void deleteAtEnd(Node** head)
{
    // checking if DLL is empty
    if (*head == NULL) {
        printf("The list is already empty.\n");
        return;
    }

    Node* temp = *head;
    if (temp->next == NULL) {
        *head = NULL;
        free(temp);
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->prev->next = NULL;
    free(temp);
}

// Function to delete a node from a specified position
void deleteAtPosition(Node** head, int position)
{           int i;
    if (*head == NULL) {
        printf("The list is already empty.\n");
        return;
    }
    Node* temp = *head;
    if (position == 1) {
        deleteAtBeginning(head);
        return;
    }
    for (i = 1; temp != NULL && i < position; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position is greater than the number of "
               "nodes.\n");
        return;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
}
```

```
// Function to print the list in forward direction
void printListForward(Node* head)
{
    Node* temp = head;
    printf("Forward List: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to print the list in reverse direction
void printListReverse(Node* head)
{
    Node* temp = head;
    if (temp == NULL) {
        printf("The list is empty.\n");
        return;
    }
    // Move to the end of the list
    while (temp->next != NULL) {
        temp = temp->next;
    }
    // Traverse backwards
    printf("Reverse List: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}

int main()
{    Node* head = NULL;                    // Demonstrating various operations

    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtBeginning(&head, 5);
    insertAtPosition(&head, 15, 2);         // List: 5 15 10 20

    printf("After Insertions:\n");
    printListForward(head);
    printListReverse(head);

    deleteAtBeginning(&head);               // List: 15 10 20
    deleteAtEnd(&head);                     // List: 15 10
    deleteAtPosition(&head, 2);             // List: 15

    printf("After Deletions:\n");
    printListForward(head);
    return 0;                           }
```

➢ **Output:**



❖ PROGRAM 3: WAP TO CIRCULAR LINKED LIST.

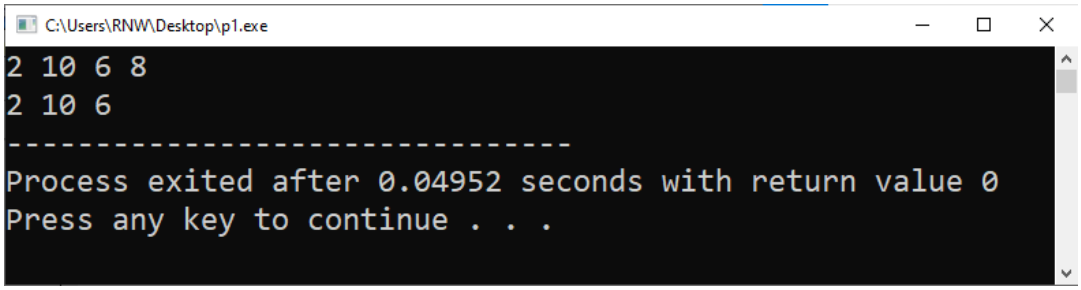
```
// C code to perform circular linked list operations

#include <stdio.h>
#include <stdlib.h>

struct Node {
  int data;
  struct Node* next;
};

struct Node* addToEmpty(struct Node* last, int data) {
  if (last != NULL) return last;

  // allocate memory to the new node
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // assign data to the new node
  newNode->data = data;

  // assign last to newNode
  last = newNode;

  // create link to iteself
  last->next = last;

  return last;
}

// add node to the front
struct Node* addFront(struct Node* last, int data) {
  // check if the list is empty
  if (last == NULL) return addToEmpty(last, data);

  // allocate memory to the new node
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // add data to the node
  newNode->data = data;

  // store the address of the current first node in the newNode
  newNode->next = last->next;

  // make newNode as head
  last->next = newNode;

  return last;
}

// add node to the end
struct Node* addEnd(struct Node* last, int data) {
```

```
  // check if the node is empty
  if (last == NULL) return addToEmpty(last, data);

  // allocate memory to the new node
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // add data to the node
  newNode->data = data;

  // store the address of the head node to next of newNode
  newNode->next = last->next;

  // point the current last node to the newNode
  last->next = newNode;

  // make newNode as the last node
  last = newNode;

  return last;
}

// insert node after a specific node
struct Node* addAfter(struct Node* last, int data, int item) {
  // check if the list is empty
  if (last == NULL) return NULL;

  struct Node *newNode, *p;

  p = last->next;
  do {
  // if the item is found, place newNode after it
  if (p->data == item) {
    // allocate memory to the new node
    newNode = (struct Node*)malloc(sizeof(struct Node));

    // add data to the node
    newNode->data = data;

    // make the next of the current node as the next of newNode
    newNode->next = p->next;

    // put newNode to the next of p
    p->next = newNode;

    // if p is the last node, make newNode as the last node
    if (p == last) last = newNode;
    return last;
  }

  p = p->next;
  } while (p != last->next);
```

```
  printf("\nThe given node is not present in the list");
  return last;
}

// delete a node
void deleteNode(struct Node** last, int key) {
  // if linked list is empty
  if (*last == NULL) return;

  // if the list contains only a single node
  if ((*last)->data == key && (*last)->next == *last) {
  free(*last);
  *last = NULL;
  return;
  }

  struct Node *temp = *last, *d;

  // if last is to be deleted
  if ((*last)->data == key) {
  // find the node before the last node
  while (temp->next != *last) temp = temp->next;

  // point temp node to the next of last i.e. first node
  temp->next = (*last)->next;
  free(*last);
  *last = temp;
  }

  // travel to the node to be deleted
  while (temp->next != *last && temp->next->data != key) {
  temp = temp->next;
  }

  // if node to be deleted was found
  if (temp->next->data == key) {
  d = temp->next;
  temp->next = d->next;
  free(d);
  }
}

void traverse(struct Node* last) {
  struct Node* p;

  if (last == NULL) {
  printf("The list is empty");
  return;
  }

  p = last->next;
```

```
  do {
  printf("%d ", p->data);
  p = p->next;

  } while (p != last->next);
}

int main() {
  struct Node* last = NULL;

  last = addToEmpty(last, 6);
  last = addEnd(last, 8);
  last = addFront(last, 2);

  last = addAfter(last, 10, 2);

  traverse(last);

  deleteNode(&last, 8);

  printf("\n");

  traverse(last);

  return 0;
}
```

➢ **Output:**



❖ PROGRAM 4: WAP TO CIRCULAR DOUBLY LINKED LIST.

```
// C code of insert node at begin in
// doubly Circular linked list.

#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};


struct Node* createNode(int x);

// Function to insert a node at the
// beginning of the doubly circular linked list
struct Node* insertAtBeginning(struct Node* head, int newData) {

    struct Node* newNode = createNode(newData);

    if (!head) {
        newNode->next = newNode->prev = newNode;
        head = newNode;
    } else {

        // List is not empty
        struct Node* last = head->prev;

        // Insert new node
        newNode->next = head;
        newNode->prev = last;
        head->prev = newNode;
        last->next = newNode;

        // Update head
        head = newNode;
    }

    return head;
}

void printList(struct Node* head) {
    if (!head) return;
    struct Node* curr = head;
    do {
        printf("%d ", curr->data);
        curr = curr->next;
    } while (curr != head);
    printf("\n");
```

```
}

struct Node* createNode(int x) {
    struct Node* newNode =
      (struct Node*)malloc(sizeof(struct Node));
    newNode->data = x;
    newNode->next = newNode->prev = NULL;
    return newNode;
}

int main(){

    // Linked List : 10<->20<->30
    struct Node* head = createNode(10);
    head->next = createNode(20);
    head->next->prev = head;
    head->next->next = createNode(30);
    head->next->next->prev = head->next;
    head->next->next->next = head;
    head->prev = head->next->next;


    head = insertAtBeginning(head, 5);
    printList(head);

    return 0;
}
```

- **Output:**