1. **Write a program to implement Enqueue and Dequeue operations of circular queue using circular link list.**

**Code :**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int data;

struct Node\* next;

}node;

node \*front=NULL,\*rear=NULL;

void enqueue(int info){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

if(rear==NULL){

front=rear=newnode;

newnode->next=front;

}else{

//temp->next=rear;

rear->next=newnode;

rear=newnode;

rear->next=front;

}

}

void display(){

if(front==NULL)

printf("\nQueue Empty");

else{

node \*temp=front;

printf("Queue : ");

do

{

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

temp=temp->next;

}while(temp!=front);

printf("\n");

}

}

void dequeue(){

if(front==NULL){

printf("Queue is Empty\n");

}else{

// node \*temp=rear;

printf(" \n Dequqe element :%d\n ",front->data );

rear->next=front->next;

front=front->next;

}

}

int main(){

enqueue(5);

enqueue(1);

enqueue(4);

enqueue(2);

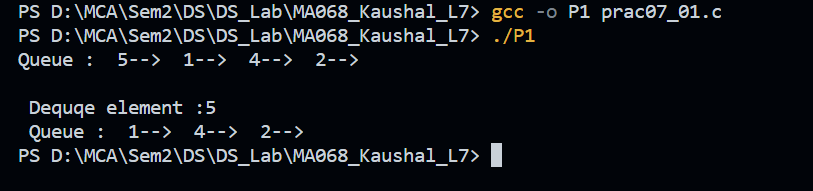
display();

dequeue();

display();

return 0;

}

**Output:**

1. **Write a program for all operations of a circular singly linked list.** 
   1. **Inserting Node – as First Node, at specific location, as Last Node**
   2. **Deleting Node – at First, at Last, specific node**
   3. **Display List**

**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int data;

struct Node\* next;

}node;

node \*head=NULL,\*tail=NULL;

void insertLast(int info){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

if(tail==NULL){

head=tail=newnode;

newnode->next=head;

}else{

tail->next=newnode;

tail=newnode;

tail->next=head;

}

}

void insertFirst(int info){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

if(tail==NULL){

head=tail=newnode;

// newnode->next=head;

}else{

newnode->next=head;

head=newnode;

tail->next=newnode;

}

}

void insertLoc(int info,int loc){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

node \*temp=head;

int count=2;

while(count!=loc){

temp=temp->next;

count++;

}

newnode->next=temp->next;

temp->next=newnode;

}

void deleteFirst(){

if(head==NULL){

printf("Queue is Empty\n");

}else{

printf(" \n Dequqe element :%d\n ",head->data );

tail->next=head->next;

head=tail->next;

}

}

void deleteLast(){

if(head==NULL){

printf("Queue is Empty\n");

}else{

node \*temp=head;

printf(" \n Dequqe element :%d\n ",tail->data );

while(temp->next!=tail){

temp=temp->next;

}

tail=temp;

temp->next=head;

}

}

void deleteLoc(int loc){

int count=2;

if(loc==1){

deleteFirst();

}else{

node \*temp=head;

while(loc!=count){

temp=temp->next;

count++;

}

printf(" \n Dequqe element :%d\n ",temp->next->data );

temp->next=temp->next->next;

}

}

void display(){

if(head==NULL)

printf("\nQueue Empty");

else{

node \*temp=head;

printf("Queue : ");

do

{

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

temp=temp->next;

}while(temp!=head);

printf("\n");

}

}

int main(){

insertLast(5);

insertLast(1);

insertLast(7);

display();

insertFirst(20);

display();

insertLoc(68,3);

display();

insertLoc(18,2);

display();

deleteFirst();

display();

deleteLast();

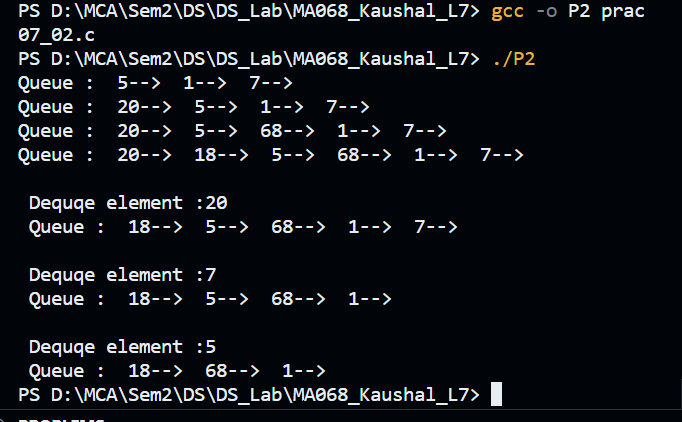
display();

deleteLoc(2);

display();

return 0;

}

**Output:**

1. **Write a program for all operations of doubly linked list**
   1. **Inserting Node – as First Node, at specific location, as Last Node**
   2. **Deleting Node – at First, at Last, specific node**
   3. **Display List**

**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

struct Node\* pre;

int data;

struct Node\* next;

}node;

node \*head=NULL,\*tail=NULL;

void insertAtLast(int info){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

newnode->pre=NULL;

if(head==NULL ){

head=tail=newnode;

}else{

tail->next=newnode;

newnode->pre=tail;

tail=newnode;

}

}

void insertAtFirst(int info){

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

newnode->pre=NULL;

if(head==NULL){

head=tail=newnode;

}else{

newnode->next=head;

//newnode->pre=head;

head=newnode;

}

}

void insertLoc(int info,int loc){

loc--;

if(loc==1){

insertAtFirst(info);

}else{

node \*newnode=(node\*)malloc(sizeof(node));

newnode->data=info;

newnode->next=NULL;

newnode->pre=NULL;

node \*temp=head;

while(--loc){

temp=temp->next;

}

newnode->next=temp->next;

newnode->pre=temp;

temp->next=newnode;

temp->next->pre=newnode;

}

}

void deleteLast(){

if(tail==NULL){

printf("\nLinked List is Empty...");

}else{

tail=tail->pre;

tail->next=NULL;

}

}

void deleteFirst(){

if(head==NULL){

printf("\nLinked List is Empty...");

}else if(head->next==NULL){

head=NULL;

}

else{

head=head->next;

head->pre=head;

}

}

void deletePosition(int pos)

{

int i=1;

node \*temp, \*position;

temp=head;

if(head==NULL)

{

printf("List is empty\n");

}

else

{

if(pos==1)

{

deleteFirst();

return;

}

while(i<pos-1)

{

temp=temp->next;

i++;

}

position=temp->next;

if(position->next!=NULL)

{

position->next->pre=temp;

}

temp->next=position->next;

free(position);

}

}

void display(){

if(head==NULL)

printf("\nQueue Empty");

else{

node \*temp=head;

printf("Doubly Linked List : ");

while(temp!=NULL)

{

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

temp=temp->next;

}

printf("\n\n");

}

}

int main(){

insertAtLast(10);

display();

insertAtLast(20);

display();

insertAtLast(30);

display();

insertAtFirst(-10);

display();

insertAtFirst(-20);

display();

insertLoc(45,3);

display();

deleteLast();

display();

deleteLast();

display();

deleteFirst();

display();

deleteFirst();

display();

deletePosition(2);

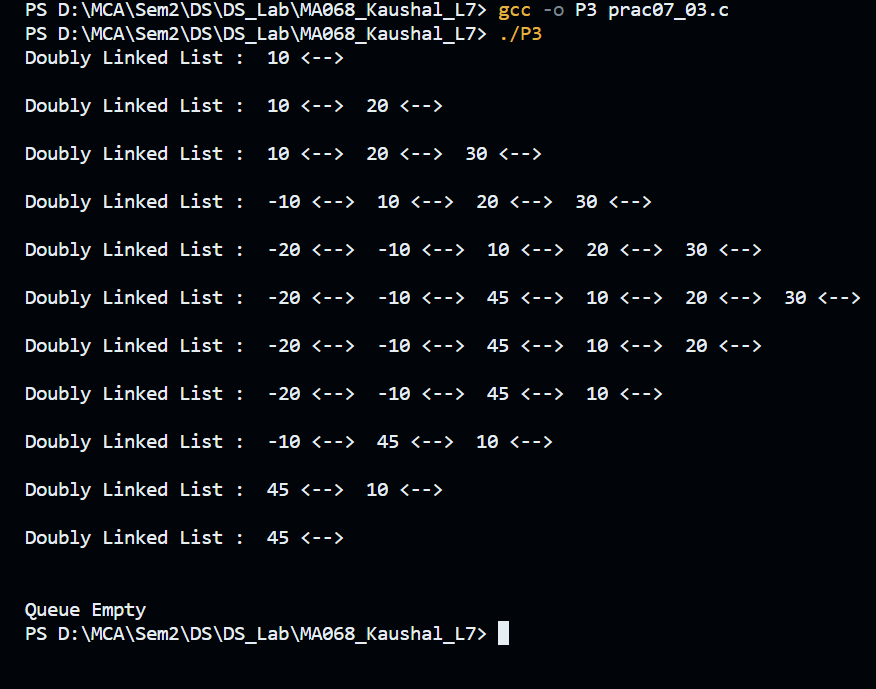
display();

deletePosition(1);

display();

return 0;

}

**Output:**

1. **Write a program for all operations of doubly linked list**
   1. **Inserting Node – as First Node, at specific location, as Last Node**
   2. **Deleting Node – at First, at Last, specific node**
   3. **Display List**

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Circular doubly linked list

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

struct Node\* head = NULL;

void insertFirst(int data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = newNode;

newNode->prev = newNode;

return;

}

struct Node\* last = head->prev;

newNode->next = head;

head->prev = newNode;

newNode->prev = last;

last->next = newNode;

head = newNode;

}

void insertLast(int data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = newNode;

newNode->prev = newNode;

return;

}

struct Node\* last = head->prev;

newNode->prev = last;

newNode->next = head;

last->next = newNode;

head->prev = newNode;

}

void insertAt(int data, int position) {

if (head == NULL || position == 1) {

insertFirst(data);

return;

}

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

struct Node\* current = head;

for (int i = 1; i < position - 1; i++) {

if (current == NULL) {

printf("Invalid position!\n");

return;

}

current = current->next;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != NULL) {

current->next->prev = newNode;

}

current->next = newNode;

}

void deleteFirst() {

if (head == NULL) {

printf("List is empty, deletion not possible.\n");

return;

}

struct Node\* temp = head;

if (head->next == head) {

head = NULL;

}

else {

head->next->prev = head->prev;

head->prev->next = head->next;

head = head->next;

}

free(temp);

}

void deleteLast() {

if (head == NULL) {

printf("List is empty, deletion not possible.\n");

return;

}

struct Node\* temp = head;

// If there is only one node

if ((head)->next == head) {

head = NULL;

}

else {

while (temp->next != head) {

temp = temp->next;

}

temp->prev->next = head;

(head)->prev = temp->prev;

}

free(temp);

}

void deleteAt( int position) {

if (head == NULL) {

printf("List is empty, deletion not possible.\n");

return;

}

if (position == 1) {

deleteFirst(head);

return;

}

struct Node\* temp = head;

int i;

for (i = 1; i < position && temp->next != head; i++) {

temp = temp->next;

}

if (i != position) {

printf("Invalid position, deletion not possible.\n");

return;

}

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

free(temp);

}

void displayList() {

if (head == NULL) {

printf("List is empty!\n");

return;

}

struct Node\* current = head;

do {

printf("%d ", current->data);

current = current->next;

} while (current != head);

printf("\n");

}

int main() {

insertFirst(10);

printf("List: ");

displayList();

insertLast( 20);

printf("List: ");

displayList();

insertAt(30, 2);

printf("List: ");

displayList();

insertAt( 40, 1);

printf("List: ");

displayList();

insertAt( 50, 6);

printf("List: ");

displayList();

deleteFirst();

printf("List: ");

displayList();

deleteLast();

printf("List: ");

displayList();

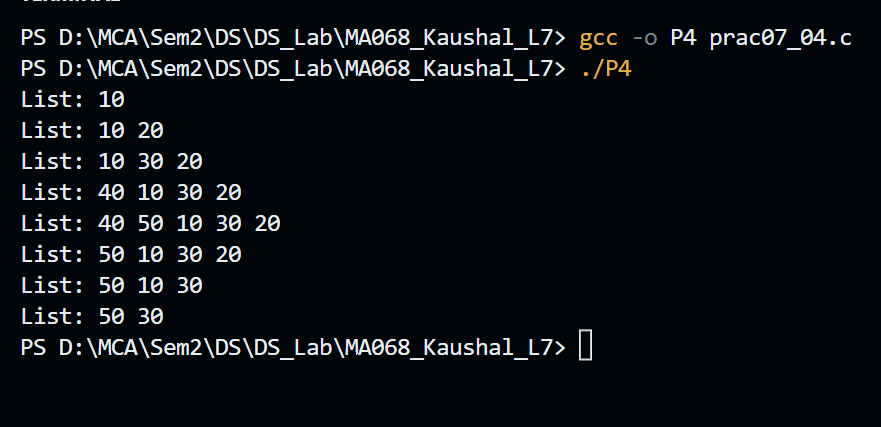
deleteAt( 2);

printf("List: ");

displayList();

return 0;

}

**Output:**