**Practical 7: Implementation of circular and doubly linked list**

Name: Sutariya Savankumar

Roll no: MA065

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

Code

*#include*<stdio.h>

*#include*<stdlib.h>

struct node{

    int value;

    struct node \*next;

};

typedef struct{

    struct node \*head;

    struct node \*tail;

} queue;

void enqueue(queue \**q*, int *value*){

*if*(*q*->tail == NULL && *q*->head == NULL){

*q*->head = (struct node \*)malloc(sizeof(struct node));

*q*->head->value = *value*;

*q*->head->next = *q*->head;

*q*->tail = *q*->head;

*return*;

    }

    struct node \*tmp = *q*->head;

*while*(tmp != *q*->tail){

        tmp=tmp->next;

    }

    tmp->next = (struct node \*)malloc(sizeof(struct node));

    tmp->next->value=*value*;

    tmp->next->next = *q*->head;

*q*->tail = tmp->next;

}

int dequeue(queue \**q*){

*if*(*q*->head == NULL || *q*->tail == NULL){

        printf("empty queue");

*return* 0;

    }

*if*(*q*->head == *q*->tail){

        int val = *q*->head->value;

        free(*q*->head);

*q*->head = NULL;

*q*->tail = NULL;

*return* val;

    }

    struct node \*tmp = *q*->head;

    int val = tmp->value;

*q*->head = *q*->head->next;

*q*->tail->next = *q*->head;

    free(tmp);

*return* val;

}

void display(queue \**q*){

*if*(*q*->head == NULL && *q*->tail == NULL){

        printf("Queue empty!!");

*return*;

    }

    struct node \*tmp = *q*->head;

    printf("\n");

*while*(tmp != *q*->tail){

        printf("(%p)|%d|%p| - ",tmp,tmp->value,tmp->next);

        tmp=tmp->next;

    }

    printf("(%p)|%d|%p| - ",tmp,tmp->value,tmp->next);

}

int main(){

    queue q;

    q.head = NULL;

    q.tail = NULL;

*// printf("%d %d",q.head->value,q.head->next->value);*

    enqueue(&q,5);

    enqueue(&q,6);

    enqueue(&q,9);

    display(&q);

    dequeue(&q);

    dequeue(&q);

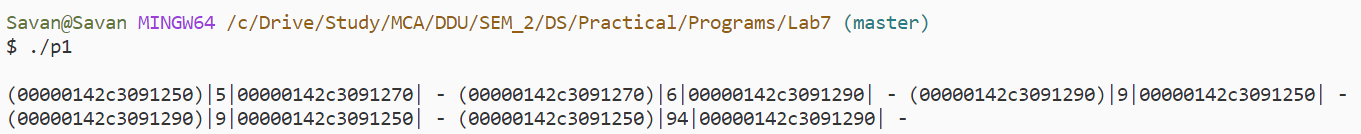
    enqueue(&q,94);

    display(&q);

*return* 0;

}

Output



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

Code

#include <stdio.h>

#include <stdlib.h>

struct node{

    int value;

    struct node \*next;

};

typedef struct{

    struct node \*head;

    struct node \*tail;

}llist;

void insertAtLast(llist \*l, int value){

    if(l->tail == NULL && l->head == NULL){

        l->head = (struct node \*)malloc(sizeof(struct node));

        l->head->value = value;

        l->head->next = l->head;

        l->tail = l->head;

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        tmp=tmp->next;

    }

    tmp->next = (struct node \*)malloc(sizeof(struct node));

    tmp->next->value=value;

    tmp->next->next = l->head;

    l->tail = tmp->next;

}

void insertAtFirst(llist \*l, int value){

    if(l->head == NULL && l->tail ==NULL){

        insertAtLast(l,value);

        return;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->next=l->head;

    new->value=value;

    l->head = new;

    l->tail->next = l->head;

}

void insertAtSpecific(llist \*l, int value, int pos){

    if((l->head == NULL && l->tail)|| pos == 0){

        insertAtFirst(l,value);

        return;

    }

    struct node \*tmp = l->head;

    int cnt=0;

    while(tmp != l->tail){

        if(cnt == pos-1){

            break;

        }

        tmp=tmp->next;

        cnt++;

    }

    if(tmp == l->tail){

        insertAtLast(l,value);

        return;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->value = value;

    new->next = tmp->next;

    tmp->next = new;

}

void deleteAtFirst(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->head;

    l->head = l->head->next;

    l->tail->next = l->head;

    free(tmp);

}

void deleteAtLast(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->head;

    while(tmp->next != l->tail){

        tmp = tmp->next;

    }

    free(l->tail);

    l->tail = tmp;

    l->tail->next = l->head;

}

void deleteAtSpecific(llist \*l, int pos){

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    if(pos == 0){

        deleteAtFirst(l);

        return;

    }

    struct node \*tmp = l->head;

    int cnt=0;

    while(tmp != l->tail){

        if(cnt == pos-1){

            break;

        }

        tmp=tmp->next;

        cnt++;

    }

    if(tmp == l->tail){

        deleteAtLast(l);

        return;

    }

    struct node \*tmp2 = tmp->next;

    tmp->next = tmp->next->next;

    free(tmp2);

}

void display(llist \*l){

    printf("\n");

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        printf("%d ",tmp->value);

        tmp = tmp->next;

    }

    printf("%d ",tmp->value);

}

int main(){

    llist l;

    l.head = NULL;

    l.tail = NULL;

    insertAtLast(&l,1);

    insertAtLast(&l,2);

    insertAtLast(&l,3);

    display(&l);

    insertAtFirst(&l,0);

    display(&l);

    insertAtSpecific(&l,4,2);

    display(&l);

    deleteAtFirst(&l);

    display(&l);

    deleteAtLast(&l);

    display(&l);

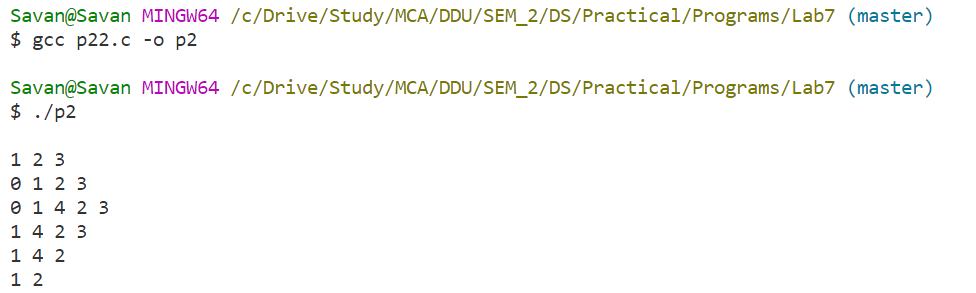
    deleteAtSpecific(&l,1);

    display(&l);

    return 0;

}

Output



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

Code

#include <stdio.h>

#include <stdlib.h>

struct node{

    int value;

    struct node \*next;

    struct node \*prev;

};

typedef struct{

    struct node \*head;

    struct node \*tail;

}llist;

void insertAtLast(llist \*l, int value){

    if(l->tail == NULL && l->head == NULL){

        l->head = (struct node \*)malloc(sizeof(struct node));

        l->head->value = value;

        l->head->next = l->head;

        l->head->prev = l->head;

        l->tail = l->head;

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        tmp=tmp->next;

    }

    tmp->next = (struct node \*)malloc(sizeof(struct node));

    tmp->next->value=value;

    tmp->next->next = l->head;

    tmp->next->prev = tmp;

    l->tail = tmp->next;

    l->head->prev = l->tail;

}

void insertAtFirst(llist \*l, int value){

    if(l->head == NULL && l->tail ==NULL){

        insertAtLast(l,value);

        return;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->next=l->head;

    new->value=value;

    new->prev = l->tail;

    l->head->prev = new;

    l->head = new;

    l->tail->next = l->head;

}

void insertAtSpecific(llist \*l, int value, int pos){

    if((l->head == NULL && l->tail)|| pos == 0){

        insertAtFirst(l,value);

        return;

    }

    struct node \*tmp = l->head;

    int cnt=0;

    while(tmp != l->tail){

        if(cnt == pos-1){

            break;

        }

        tmp=tmp->next;

        cnt++;

    }

    if(tmp == l->tail){

        insertAtLast(l,value);

        return;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->value = value;

    new->next = tmp->next;

    new->prev = tmp;

    tmp->next->prev = new;

    tmp->next = new;

}

void deleteAtFirst(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->head;

    l->head = l->head->next;

    l->head->prev = l->tail;

    l->tail->next = l->head;

    free(tmp);

}

void deleteAtLast(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        tmp=tmp->next;

    }

    tmp->prev->next = l->head;

    l->head->prev = tmp->prev;

    l->tail = tmp->prev;

    free(tmp);

}

void deleteAtSpecific(llist \*l, int pos){

    if(l->head == NULL && l->tail == NULL){

        return;

    }

    if(pos == 0){

        deleteAtFirst(l);

        return;

    }

    struct node \*tmp = l->head;

    int cnt=0;

    while(tmp != l->tail){

        if(cnt == pos-1){

            break;

        }

        tmp=tmp->next;

        cnt++;

    }

    if(tmp == l->tail){

        deleteAtLast(l);

        return;

    }

    tmp->next->next->prev = tmp;

    tmp->next = tmp->next->next;

    free(tmp->next->prev);

}

void display(llist \*l){

    printf("\n");

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        printf("%d ",tmp->value);

        tmp=tmp->next;

    }

    printf("%d ",tmp->value);

}

int main(){

    llist l;

    l.head = NULL;

    l.tail = NULL;

    insertAtLast(&l,1);

    insertAtLast(&l,2);

    display(&l);

    insertAtFirst(&l,3);

    display(&l);

    insertAtSpecific(&l,4,2);

    display(&l);

    deleteAtFirst(&l);

    display(&l);

    deleteAtLast(&l);

    display(&l);

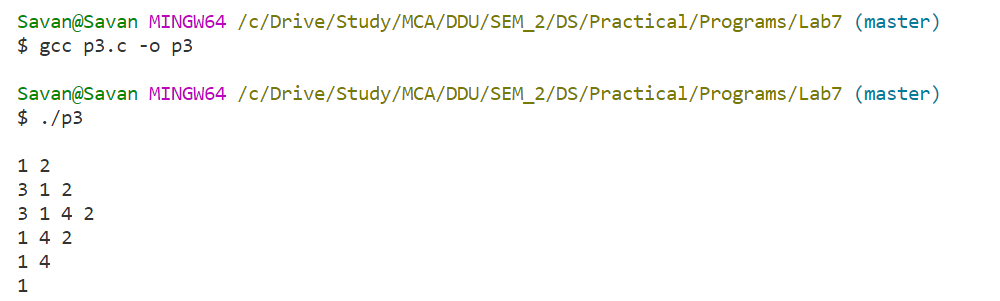
    deleteAtSpecific(&l,2);

    display(&l);

    return 0;

}

Output



1. **Write a program for all operations of circular 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>

struct node{

    int value;

    struct node \*next;

    struct node \*prev;

};

typedef struct{

    struct node \*head;

    struct node \*tail;

}llist;

void insertAtLast(llist \*l, int value){

    if(l->tail == NULL && l->head == NULL){

        l->head = (struct node \*)malloc(sizeof(struct node));

        l->head->value = value;

        l->head->next = l->head;

        l->head->prev = l->head;

        l->tail = l->head;

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        tmp=tmp->next;

    }

    tmp->next = (struct node \*)malloc(sizeof(struct node));

    tmp->next->value=value;

    tmp->next->next = l->head;

    tmp->next->prev = tmp;

    l->tail = tmp->next;

    l->head->prev = l->tail;

}

void insertAtFirst(llist \*l, int value){

    if(l->head == NULL && l->tail ==NULL){

        insertAtLast(l,value);

        return;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->next=l->head;

    new->value=value;

    new->prev = l->tail;

    l->head->prev = new;

    l->head = new;

    l->tail->next = l->head;

}

void insertAtSpecific(llist \*l, int value, int pos){

    if((l->head == NULL && l->tail)|| pos == 0){

        insertAtFirst(l,value);

        return;

    }

    struct node \*tmp = l->head;

    int i=0;

    while(i<pos-1){

        tmp=tmp->next;

        i++;

    }

    struct node \*new = (struct node \*)malloc(sizeof(struct node));

    new->value = value;

    new->next = tmp->next;

    new->prev = tmp;

    tmp->next->prev = new;

    tmp->next = new;

}

void deleteAtFirst(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->head;

    l->head = l->head->next;

    l->head->prev = l->tail;

    l->tail->next = l->head;

    free(tmp);

}

void deleteAtLast(llist \*l){

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    if(l->head == l->tail){

        free(l->head);

        l->head = NULL;

        l->tail = NULL;

        return;

    }

    struct node \*tmp = l->tail;

    l->tail = l->tail->prev;

    l->tail->next = l->head;

    l->head->prev = l->tail;

    free(tmp);

}

void deleteAtSpecific(llist \*l, int pos){

    if(l->head == NULL && l->tail == NULL){

        return;

    }

    if(pos == 0){

        deleteAtFirst(l);

        return;

    }

    struct node \*tmp = l->head;

    int cnt=0;

    while(tmp != l->tail){

        if(cnt == pos-1){

            break;

        }

        tmp=tmp->next;

        cnt++;

    }

    if(tmp == l->tail){

        deleteAtLast(l);

        return;

    }

    tmp->next->next->prev = tmp;

    tmp->next = tmp->next->next;

    free(tmp->next->prev);

}

void display(llist \*l){

    printf("\n");

    if(l->head == NULL && l->tail == NULL){

        printf("List is empty");

        return;

    }

    struct node \*tmp = l->head;

    while(tmp != l->tail){

        printf("%d ",tmp->value);

        tmp=tmp->next;

    }

    printf("%d ",tmp->value);

}

int main(){

    llist l;

    l.head = NULL;

    l.tail = NULL;

    insertAtLast(&l,1);

    insertAtLast(&l,2);

    display(&l);

    insertAtFirst(&l,3);

    display(&l);

    insertAtSpecific(&l,4,2);

    display(&l);

    deleteAtFirst(&l);

    display(&l);

    deleteAtLast(&l);

    display(&l);

    deleteAtSpecific(&l,2);

    display(&l);

    return 0;

}

Output

