Practical no: 3

Objective: Write a program in C to create a doubly linked-list with following operations:-

- Insertion at beginning.
- Insertion at end
- Insertion at position
- Deletion at begin
- Deletion at end
- Deletion at end

Program Codes: Following is the code of this problems in C:-

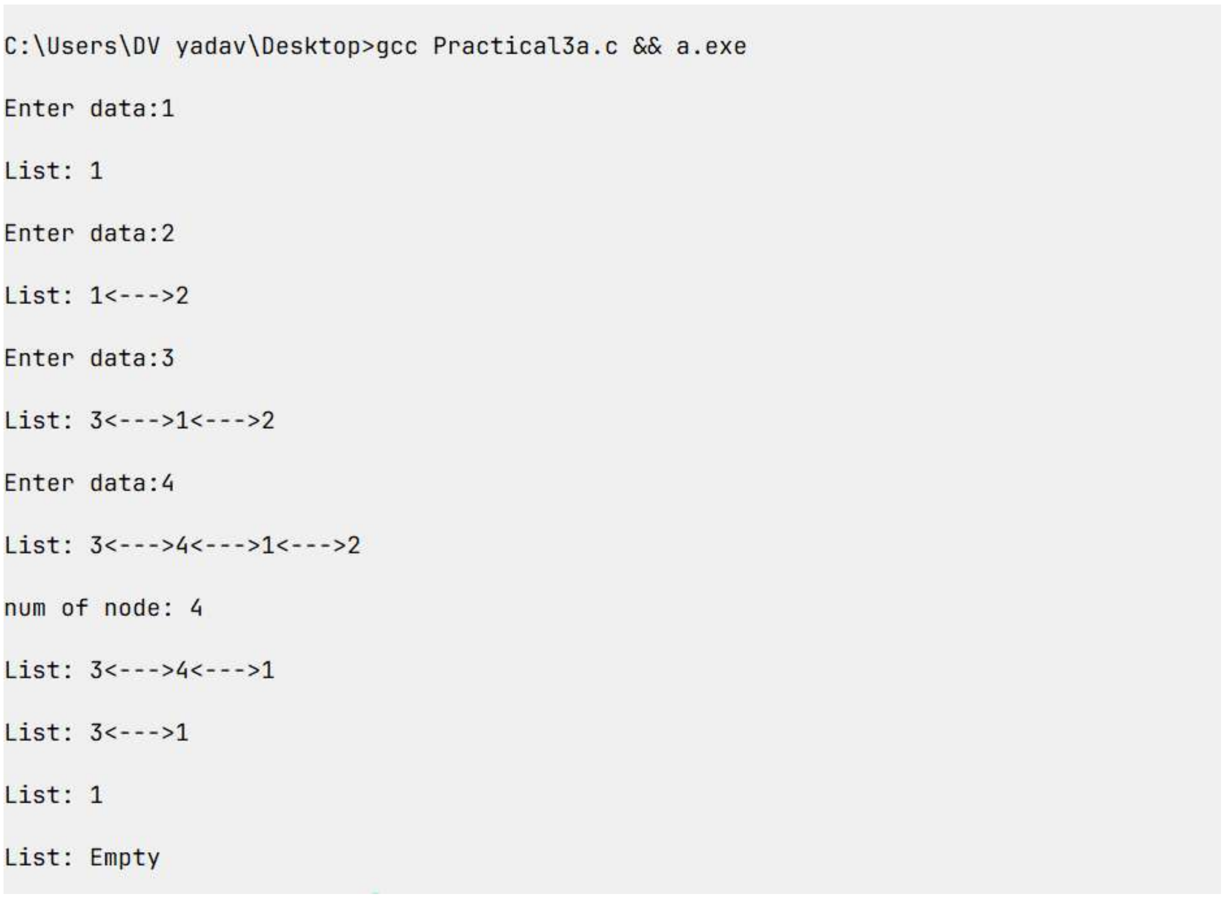
1. Practical3a.c

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

// use dnode instead of node
typedef struct dnode dnode;

// structure for doubly linked list node
struct dnode{
    int data;
    dnode * prev;
    dnode * next;
};

// function declarations
void display(dnode*);
int node_count(dnode*);
void insert_at_end(dnode**);
void insert_at_begin(dnode**);
void insert_at_pos(dnode**, int);
void del_at_begin(dnode**);
void del_at_end(dnode**);
void del_at_pos(dnode**, int);

// traverse the data from donde
```

```c
void display(dnode* temp){
    printf("\nList: ");
    while(temp->next != NULL){
        printf("%d<--->",temp->data);
        temp=temp->next;
    }
    printf("%d\n",temp->data);
}

int node_count(dnode* temp){
    int count = 0;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    return count;
}

void insert_at_end(dnode** address_of_head){
    // create a new node
    dnode* new_node_ptr = (dnode*)malloc(sizeof(dnode));
    new_node_ptr->prev = NULL;
    new_node_ptr->next = NULL;
    printf("\nEnter data:");
    scanf("%d", & new_node_ptr->data);
    // if list is empty
    if(*address_of_head == NULL){
        *address_of_head = new_node_ptr;
        return;
    }
    // if list is not empty
    dnode* temp  = *address_of_head;
    // traverse till end
    while(temp->next != NULL){
        temp=temp->next;
    }
    // attach the new node
    temp->next = new_node_ptr;
    new_node_ptr->prev = temp;
}
```

```c
void insert_at_begin(dnode** address_of_head){
    // create new node
    dnode* new_node_ptr = (dnode*)malloc(sizeof(dnode));
    new_node_ptr->prev = NULL;
    new_node_ptr->next = NULL;
    printf("\nEnter data:");
    scanf("%d", & new_node_ptr->data);
    //attaching new node to front
    dnode* temp_head = *address_of_head;
    new_node_ptr->next = temp_head;
    temp_head->prev = new_node_ptr;
    //changing to new head
    *address_of_head = new_node_ptr;
}

void insert_at_pos(dnode** address_of_head, int pos){
    // invalid position handling
    if(pos < 1 || pos > node_count(*address_of_head)+1){
        printf("\nInvalid position. try again.\n");
        return;
    }

    // if position is first or last
    if(pos == 1){
        insert_at_begin(address_of_head);
        return;
    }else if(pos == node_count(* address_of_head)+1){
        insert_at_end(address_of_head);
        return;
    }

    // other than first or last
    dnode* temp = *address_of_head;
    // create new node
    dnode* new_node_ptr = (dnode*)malloc(sizeof(dnode));
    new_node_ptr->prev = NULL;
    new_node_ptr->next = NULL;
    printf("\nEnter data:");
    scanf("%d", & new_node_ptr->data);
    // traverse one before position
    for(int i = 1; i < pos -1 ; i++){
        temp = temp->next;
    }
    // attaching new node
    new_node_ptr->prev = temp;
    new_node_ptr->next = temp->next;
    temp->next->prev = new_node_ptr;
    temp->next = new_node_ptr;
}

void del_at_begin(dnode ** address_of_head){
    // detaching links
    dnode* temp = *address_of_head;
    *address_of_head = temp->next;
    (*address_of_head)->prev = NULL;
    temp->next = NULL;
    free(temp);
}

void del_at_end(dnode** address_of_head){
    // if only consist one node
    if(node_count(*address_of_head) == 1){
        free(*address_of_head);
        *address_of_head = NULL;
        return;
    }

    dnode* temp = *address_of_head;
    // traverse to the deleting node
    while(temp->next!= NULL){
        temp = temp->next;
    }
    // detaching links
    temp->prev->next = NULL;
    temp->prev = NULL;
    free(temp);
}

void del_at_pos(dnode** address_of_head, int pos){
    // handle invalid position
    if(pos < 1 || pos > node_count(*address_of_head)){
        printf("\nInvalid Position. Retry\n");
        return;
    }
    // delete at begin
    if(pos == 1){
        del_at_end(address_of_head);
        return;
    }else if(pos == node_count(*address_of_head)){
        del_at_end(address_of_head);
        return;
    }
    // other than first and last
    dnode* temp = *address_of_head;
    // traverse till one before the deleting node
    for(int i = 1; i < pos -1; i++){
        temp = temp->next;
    }
    // est. new links links
    temp->next = temp->next->next;
    temp = temp->next->prev; //pointer jumped on deleting node
    temp->next->prev = temp->prev;
    // removing links
    temp->next = NULL;
    temp->prev = NULL;
    free(temp);
}

void main(){
    dnode* head = NULL;

    insert_at_end(&head); display(head);
    insert_at_end(&head); display(head);
    insert_at_begin(&head); display(head);
    insert_at_pos(&head, 2); display(head);
    printf("\nnum of node: %d\n",node_count(head));

    del_at_end(&head); display(head);
    del_at_pos(&head, 2); display(head);
    del_at_begin(&head); display(head);
    del_at_pos(&head, 1); display(head);
    printf("\nnum of node: %d\n",node_count(head));
}
```

Output: Following is the output of the program:-

1. 

```
C:\Users\DV yadav\Desktop>gcc Practical3a.c && a.exe

Enter data:1

List: 1

Enter data:2

List: 1<--->2

Enter data:3

List: 3<--->1<--->2

Enter data:4

List: 3<--->4<--->1<--->2

num of node: 4

List: 3<--->4<--->1

List: 3<--->1

List: 1

List: Empty
```