# Assignment 3(Linked List)

1. Implementation of Singly Linked List :

CODE :

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

int size = 0;

struct Node \*initializer(int data)

{

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

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

void insertFirst(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        return;

    }

    newNode->next = \*head;

    \*head = newNode;

}

void insertLast(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        return;

    }

    struct Node \*temp = \*head;

    while (temp->next != NULL)

    {

        temp=temp->next;

    }

    temp->next = newNode;

}

void insertMiddle(struct Node \*\*head, int data, int index)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        return;

    }

    if (index == 0)

    {

        newNode->next = \*head;

        \*head = newNode;

        return;

    }

    struct Node \*p1 = \*head;

    struct Node \*p2 = (\*head)->next;

    while (--index)

    {

        p1 = p1->next;

        p2 = p2->next;

    }

    p1->next = newNode;

    newNode->next = p2;

}

void printLL(struct Node \*head)

{

    if (head == NULL)

    {

        printf("The Linked list is empty!!\n");

        return;

    }

    struct Node \*temp = head;

    while (temp != NULL)

    {

        printf("%d-->",temp->data);

        temp=temp->next;

    }

    printf("NULL\n");

}

int deleteLast(struct Node \*\*head)

{

    if (\*head == NULL)

    {

        printf("The linked list is empty, nothing to delete.\n");

        return -1;

    }

    struct Node \*lastSec = \*head;

    struct Node \*last = (\*head)->next;

    while (last->next != NULL)

    {

        lastSec = lastSec->next;

        last = last->next;

    }

    lastSec->next = NULL;

    int pop = last->data;

    free(last);

    return pop;

}

struct Node \*copyLL(struct Node \*head)

{

    if (head == NULL)

    {

        return NULL;

    }

    struct Node \*head2 = NULL;

    struct Node \*temp = head;

    while (temp != NULL)

    {

        insertLast(&head2, temp->data);

        temp = temp->next;

    }

    return head2;

}

int main(int argc, char const \*argv[])

{

    struct Node \*head = NULL, \*head2 = NULL;

    int data, ch;

    printf("Welcome to Linked List!!\n");

    printf("1 for inserting at first.\n");

    printf("2 for inserting at last.\n");

    printf("3 for print linked list.\n");

    printf("4 for inserting at index(head has 0 index).\n");

    printf("5 to print size of linked list.\n");

    printf("6 for deleting last node.\n");

    printf("7 for copying linked list.\n");

    printf("0 to exit.\n");

    while(1)

    {

        printf("\nEnter operation to perform : ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter data to insert : ");

            scanf("%d", &data);

            insertFirst(&head,data);

            break;

        case 2:

            printf("Enter data to insert : ");

            scanf("%d", &data);

            insertLast(&head,data);

            break;

        case 3:

            printf("Your Linked List is : \n");

            printLL(head);

            break;

        case 4:

            printf("Enter data to insert : ");

            scanf("%d", &data);

            int index;

            printf("Enter index to insert : ");

            scanf("%d",&index);

            insertMiddle(&head,data,index);

            break;

        case 5:

            printf("The size of list is : %d\n",size);

            break;

        case 6:

            printf("%d deleted from linked list.\n",deleteLast(&head));

            printLL(head);

            break;

        case 7:

            head2 = copyLL(head);

            printf("The copied linked list is :\n");

            printLL(head2);

            break;

        case 0:

            printf("Exited Successfully!!\n");

            return 0;

        default:

            printf("Error, Try again!!\n");

            break;

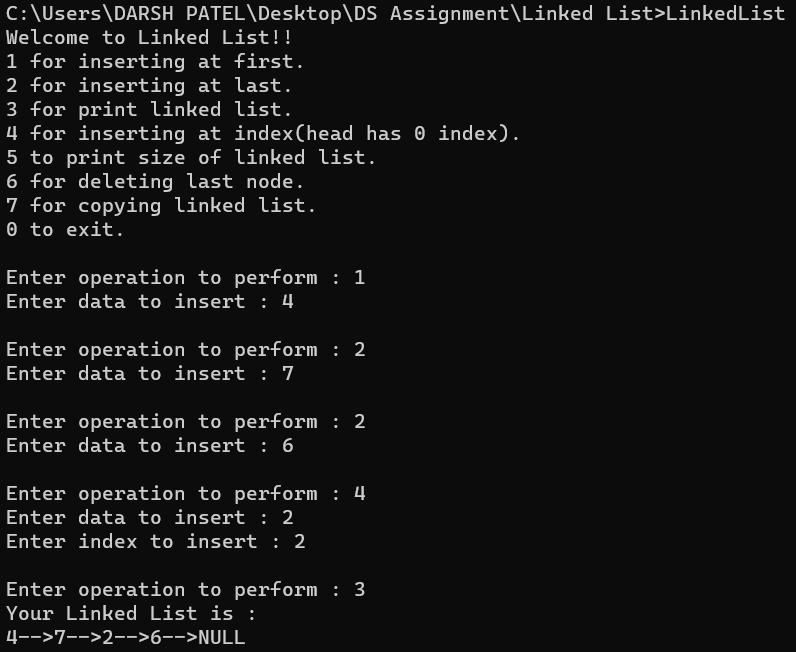
        }

    }

    return 0;

}

OUTPUT :



1. Implementation of Doubly Linked List :

CODE :

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    int data;

    struct Node \*prev;

    struct Node \*next;

};

int size = 0;

struct Node \*initializer(int data)

{

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

    newNode->data = data;

    newNode->prev = NULL;

    newNode->next = NULL;

    return newNode;

}

void insertFirst(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        return;

    }

    newNode->next = \*head;

    (\*head)->prev = newNode;

    \*head = newNode;

}

void insertLast(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        return;

    }

    struct Node \*temp = \*head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->prev = temp;

}

void insertAtIndex(struct Node \*\*head, int data, int index)

{

    if (index < 0 || index > size)

    {

        printf("Invalid index for insertion.\n");

        return;

    }

    if (index == 0)

    {

        insertFirst(head, data);

        return;

    }

    if (index == size)

    {

        insertLast(head, data);

        return;

    }

    struct Node \*newNode = initializer(data);

    struct Node \*temp = \*head;

    for (int i = 0; i < index - 1; i++)

    {

        temp = temp->next;

    }

    newNode->next = temp->next;

    newNode->prev = temp;

    temp->next->prev = newNode;

    temp->next = newNode;

    size++;

}

void deleteFirst(struct Node \*\*head)

{

    if (\*head == NULL)

    {

        printf("List is empty, cannot delete.\n");

        return;

    }

    struct Node \*temp = \*head;

    \*head = temp->next;

    if (\*head != NULL)

    {

        (\*head)->prev = NULL;

    }

    free(temp);

    size--;

}

void deleteLast(struct Node \*\*head)

{

    if (\*head == NULL)

    {

        printf("List is empty, cannot delete.\n");

        return;

    }

    if ((\*head)->next == NULL)

    {

        free(\*head);

        \*head = NULL;

        size--;

        return;

    }

    struct Node \*temp = \*head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->prev->next = NULL;

    free(temp);

    size--;

}

void printLL(struct Node \*head)

{

    if (head == NULL)

    {

        printf("The Linked list is empty!!\n");

        return;

    }

    struct Node \*temp = head;

    printf("NULL<--");

    while (temp != NULL)

    {

        printf("%d<-->", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

int main(int argc, char const \*argv[])

{

    struct Node \*head = NULL;

    int data;

    int ch;

    printf("Welcome to Doubly Linked List!!\n");

    printf("1 for inserting at first.\n");

    printf("2 for inserting at last.\n");

    printf("3 for print linked list.\n");

    printf("4 for peeking head node.\n");

    printf("5 to print size of linked list.\n");

    printf("6 for inserting at index.\n");

    printf("0 to exit.\n");

loop:

    printf("\nEnter operation to perform : ");

    scanf("%d", &ch);

    switch (ch)

    {

    case 1:

        printf("Enter data to insert : ");

        scanf("%d", &data);

        insertFirst(&head, data);

        goto loop;

    case 2:

        printf("Enter data to insert : ");

        scanf("%d", &data);

        insertLast(&head, data);

        goto loop;

    case 3:

        printf("Your Linked List is : \n");

        printLL(head);

        goto loop;

    case 4:

        if (head)

        {

            printf("The head node contains : %d\n", head->data);

        }

        else

        {

            printf("The list is empty.\n");

        }

        goto loop;

    case 5:

        printf("The size of list is : %d\n", size);

        goto loop;

    case 6:

        printf("Enter data to insert : ");

        scanf("%d", &data);

        int index;

        printf("Enter index to insert : ");

        scanf("%d",&index);

        insertAtIndex(&head, data, index);

        goto loop;

    case 0:

        printf("Exited Successfully!!\n");

        break;

    default:

        printf("Error, Try again!!\n");

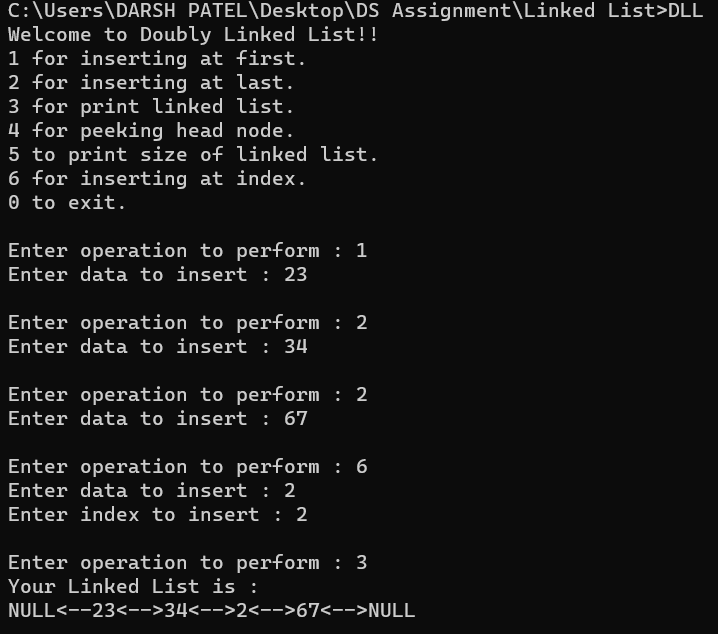
        goto loop;

    }

    return 0;

}

OUTPUT :



1. Implementation of Circular Doubly Linked List :

CODE :

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    int data;

    struct Node \*prev;

    struct Node \*next;

};

int size = 0;

struct Node \*initializer(int data)

{

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

    newNode->data = data;

    newNode->prev = NULL;

    newNode->next = NULL;

    return newNode;

}

void insertFirst(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        newNode->next = newNode;

        newNode->prev = newNode;

        return;

    }

    newNode->next = \*head;

    newNode->prev = (\*head)->prev;

    (\*head)->prev->next = newNode;

    (\*head)->prev = newNode;

    \*head = newNode;

}

void insertLast(struct Node \*\*head, int data)

{

    struct Node \*newNode = initializer(data);

    size++;

    if (\*head == NULL)

    {

        \*head = newNode;

        newNode->next = newNode;

        newNode->prev = newNode;

        return;

    }

    newNode->prev = (\*head)->prev;

    newNode->next = \*head;

    (\*head)->prev->next = newNode;

    (\*head)->prev = newNode;

}

void deleteFirst(struct Node \*\*head)

{

    if (\*head == NULL)

    {

        printf("List is empty, cannot delete.\n");

        return;

    }

    struct Node \*temp = \*head;

    if ((\*head)->next == \*head)

    {

        \*head = NULL;

    }

    else

    {

        \*head = temp->next;

        (\*head)->prev = temp->prev;

        temp->prev->next = \*head;

    }

    free(temp);

    size--;

}

void deleteLast(struct Node \*\*head)

{

    if (\*head == NULL)

    {

        printf("List is empty, cannot delete.\n");

        return;

    }

    struct Node \*temp = (\*head)->prev;

    if ((\*head)->next == \*head)

    {

        \*head = NULL;

    }

    else

    {

        temp->prev->next = \*head;

        (\*head)->prev = temp->prev;

    }

    free(temp);

    size--;

}

void printLL(struct Node \*head)

{

    if (head == NULL)

    {

        printf("The Linked list is empty!!\n");

        return;

    }

    struct Node \*temp = head;

    do

    {

        printf("%d<-->", temp->data);

        temp = temp->next;

    } while (temp != head);

    printf("... (circular)\n");

}

int main(int argc, char const \*argv[])

{

    struct Node \*head = NULL;

    int data;

    int ch;

    printf("Welcome to Circular Doubly Linked List!!\n");

    printf("1 for inserting at first.\n");

    printf("2 for inserting at last.\n");

    printf("3 for print linked list.\n");

    printf("4 for peeking head node.\n");

    printf("5 to print size of linked list.\n");

    printf("6 for delete first.\n");

    printf("7 for delete last.\n");

    printf("0 to exit.\n");

loop:

    printf("\nEnter operation to perform : ");

    scanf("%d", &ch);

    switch (ch)

    {

    case 1:

        printf("Enter data to insert : ");

        scanf("%d", &data);

        insertFirst(&head, data);

        goto loop;

    case 2:

        printf("Enter data to insert : ");

        scanf("%d", &data);

        insertLast(&head, data);

        goto loop;

    case 3:

        printf("Your Linked List is : \n");

        printLL(head);

        goto loop;

    case 4:

        if (head)

        {

            printf("The head node contains : %d\n", head->data);

        }

        else

        {

            printf("The list is empty.\n");

        }

        goto loop;

    case 5:

        printf("The size of list is : %d\n", size);

        goto loop;

    case 6:

        deleteFirst(&head);

        goto loop;

    case 7:

        deleteLast(&head);

        goto loop;

    case 0:

        printf("Exited Successfully!!\n");

        break;

    default:

        printf("Error, Try again!!\n");

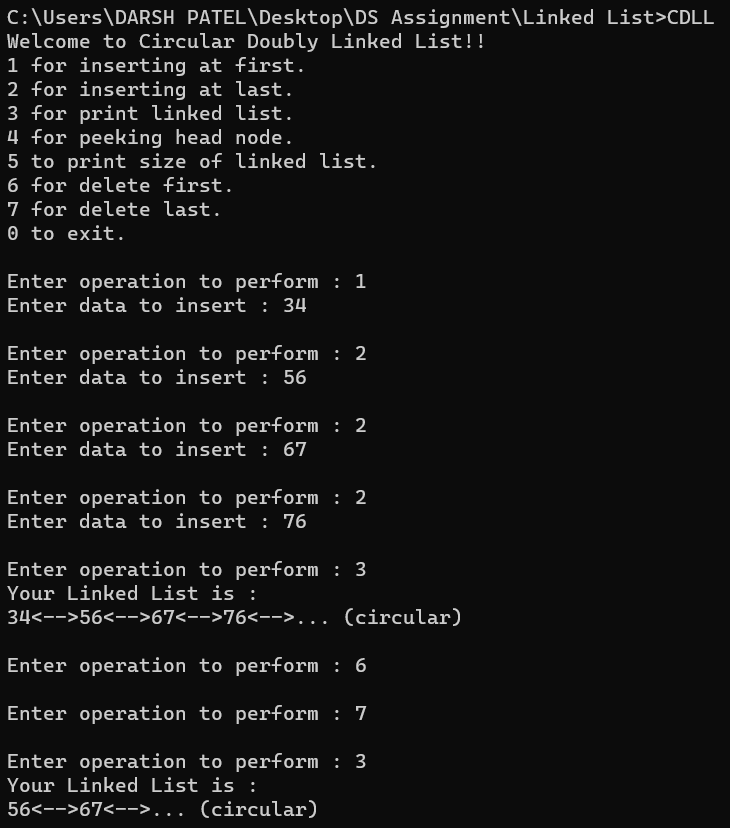
        goto loop;

    }

    return 0;

}

OUTPUT :



1. Implementation of Queue using Linked List :

CODE :

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

struct Node \*initializer(int data)

{

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

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

struct Queue

{

    struct Node \*front;

    struct Node \*rear;

}\*queue;

struct Queue \*constructor()

{

    struct Queue \*q = (struct Queue\*)malloc(sizeof(struct Queue));

    q->front = NULL;

    q->rear = NULL;

    return q;

}

int isEmpty()

{

    return queue->front == NULL && queue->rear == NULL;

}

void display()

{

    if (isEmpty())

    {

        printf("Queue Underflow, Nothing to display!!");

        return;

    }

    struct Node\* temp = queue->front;

    printf("Your queue is :\n");

    while (temp != NULL)

    {

        printf("%d->",temp->data);

        temp = temp->next;

    }

    printf("Null\n\n");

    free(temp);

}

void enqueue(int data)

{

    struct Node \*newNode = initializer(data);

    if (isEmpty())

    {

        queue->front = queue->rear = newNode;

        return;

    }

    queue->rear->next = newNode;

    queue->rear = newNode;

}

int dequeue()

{

    if (isEmpty())

    {

        printf("Queue Underflow, Nothing to dequeue!!");

        return -1;

    }

    struct Node \*temp = queue->front;

    int data = temp->data;

    queue->front = queue->front->next;

    free(temp);

    return data;

}

void main()

{

    int ch, data;

    queue = constructor();

    printf("Enter operation :\n");

    printf("1 to Enqueue.\n");

    printf("2 to Dequeue.\n");

    printf("0 to exit.\n");

    while (1)

    {

        scanf("%d",&ch);

        switch (ch)

        {

        case 1:

            printf("Enter data to enqueue : ");

            scanf("%d",&data);

            enqueue(data);

            display();

            break;

        case 2:

            printf("Dequeued element : %d\n",dequeue());

            display();

            break;

        case 0:

            printf("Succesfully exited\n");

            return;

        default:

            printf("Error Try again!!");

            break;

        }

    }

}

OUTPUT :

