1. Write a program to construct a queue using Linked List with comments on each line.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

int main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

/\* Create an empty queue \*/

void create()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

printf("\n Queue size : %d", count);

}

/\* Enqueing the queue \*/

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

/\* Displaying the queue elements \*/

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

/\* Dequeing the queue \*/

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

/\* Returns the front element of queue \*/

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

/\* Display if queue is empty or not \*/

void empty()

{

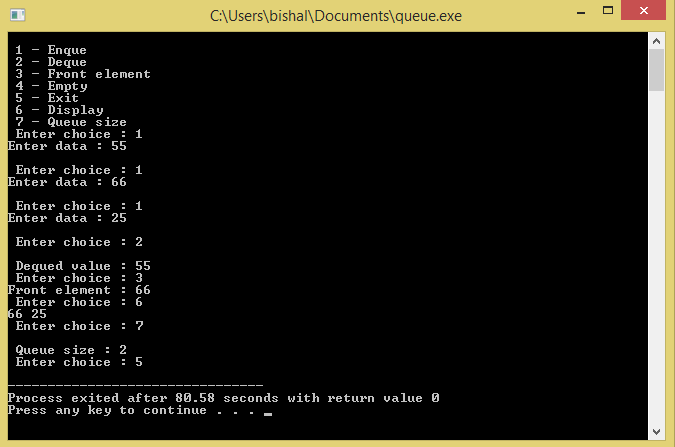
if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}



2. Write a program to construct a Circular Linked List with comments on each line.

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node \*next;

}node;

void insert(node \*pointer, int data)

{

node \*start = pointer;

/\* Iterate through the list till we encounter the last node.\*/

while(pointer->next!=start)

{

pointer = pointer -> next;

}

/\* Allocate memory for the new node and put data in it.\*/

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

pointer = pointer->next;

pointer->data = data;

pointer->next = start;

}

int find(node \*pointer, int key)

{

node \*start = pointer;

pointer = pointer -> next; //First node is dummy node.

/\* Iterate through the entire linked list and search for the key. \*/

while(pointer!=start)

{

if(pointer->data == key) //key is found.

{

return 1;

}

pointer = pointer -> next;//Search in the next node.

}

/\*Key is not found \*/

return 0;

}

void delet(node \*pointer, int data)

{

node \*start = pointer;

/\* Go to the node for which the node next to it has to be deleted \*/

while(pointer->next!=start && (pointer->next)->data != data)

{

pointer = pointer -> next;

}

if(pointer->next==start)

{

printf("Element %d is not present in the list\n",data);

return;

}

/\* Now pointer points to a node and the node next to it has to be removed \*/

node \*temp;

temp = pointer -> next;

/\*temp points to the node which has to be removed\*/

pointer->next = temp->next;

/\*We removed the node which is next to the pointer (which is also temp) \*/

free(temp);

/\* Beacuse we deleted the node, we no longer require the memory used for it .

free() will deallocate the memory.

\*/

return;

}

void print(node \*start,node \*pointer)

{

if(pointer==start)

{

return;

}

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

print(start,pointer->next);

}

int main()

{

/\* start always points to the first node of the linked list.

temp is used to point to the last node of the linked list.\*/

node \*start,\*temp;

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

temp = start;

temp -> next = start;

/\* Here in this code, we take the first node as a dummy node.

The first node does not contain data, but it used because to avoid handling special cases

in insert and delete functions.

\*/

printf("1. Insert\n");

printf("2. Delet\n");

printf("3. Print\n");

printf("4. Find\n");

while(1)

{

int query;

printf("Enter your choice:");

scanf("%d",&query);

if(query==1)

{

int data;

printf("Enter the data:");

scanf("%d",&data);

insert(start,data);

}

else if(query==2)

{

int data;

scanf("%d",&data);

delet(start,data);

}

else if(query==3)

{

printf("The list is ");

print(start,start->next);

printf("\n");

}

else if(query==4)

{

int data;

scanf("%d",&data);

int status = find(start,data);

if(status)

{

printf("Element Found\n");

}

else

{

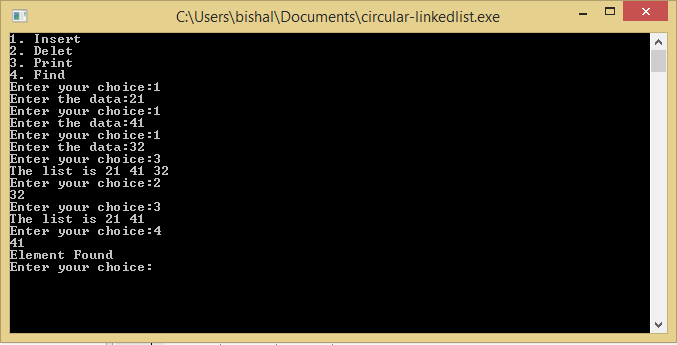
printf("Element Not Found\n");

}

}

}

}



3. Write a program to implement Stack as a circular list with comments on each line.

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node \*next;

}node;

void insert(node \*pointer, int data)

{

node \*start = pointer;

/\* Iterate through the list till we encounter the last node.\*/

while(pointer->next!=start)

{

pointer = pointer -> next;

}

/\* Allocate memory for the new node and put data in it.\*/

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

pointer = pointer->next;

pointer->data = data;

pointer->next = start;

}

int find(node \*pointer, int key)

{

node \*start = pointer;

pointer = pointer -> next; //First node is dummy node.

/\* Iterate through the entire linked list and search for the key. \*/

while(pointer!=start)

{

if(pointer->data == key) //key is found.

{

return 1;

}

pointer = pointer -> next;//Search in the next node.

}

/\*Key is not found \*/

return 0;

}

void delet(node \*pointer, int data)

{

node \*start = pointer;

/\* Go to the node for which the node next to it has to be deleted \*/

while(pointer->next!=start && (pointer->next)->data != data)

{

pointer = pointer -> next;

}

if(pointer->next==start)

{

printf("Element %d is not present in the list\n",data);

return;

}

/\* Now pointer points to a node and the node next to it has to be removed \*/

node \*temp;

temp = pointer -> next;

/\*temp points to the node which has to be removed\*/

pointer->next = temp->next;

/\*We removed the node which is next to the pointer (which is also temp) \*/

free(temp);

/\* Beacuse we deleted the node, we no longer require the memory used for it .

free() will deallocate the memory.

\*/

return;

}

void print(node \*start,node \*pointer)

{

if(pointer==start)

{

return;

}

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

print(start,pointer->next);

}

int main()

{

/\* start always points to the first node of the linked list.

temp is used to point to the last node of the linked list.\*/

node \*start,\*temp;

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

temp = start;

temp -> next = start;

/\* Here in this code, we take the first node as a dummy node.

The first node does not contain data, but it used because to avoid handling special cases

in insert and delete functions.

\*/

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Print\n");

printf("4. Find\n");

while(1)

{

int query;

printf("Enter your choice:");

scanf("%d",&query);

if(query==1)

{

int data;

printf("Enter data:");

scanf("%d",&data);

insert(start,data);

}

else if(query==2)

{

int data;

scanf("%d",&data);

delet(start,data);

}

else if(query==3)

{

printf("The list is ");

print(start,start->next);

printf("\n");

}

else if(query==4)

{

int data;

scanf("%d",&data);

int status = find(start,data);

if(status)

{

printf("Element Found\n");

}

else

{

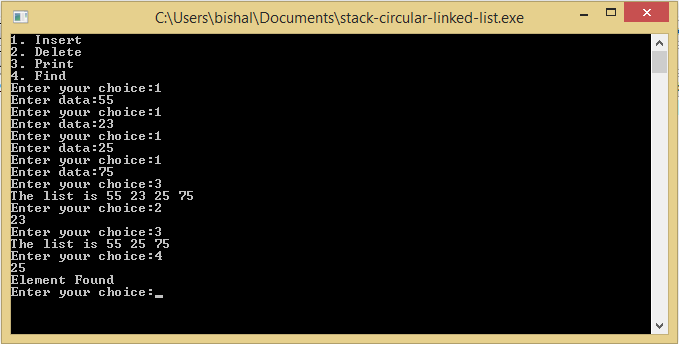
printf("Element Not Found\n");

}

}

}

}



5. Write a program to implement Doubly Linked List with comments on each line.

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

}node;

void insert(node \*pointer, int data)

{

/\* Iterate through the list till we encounter the last node.\*/

while(pointer->next!=NULL)

{

pointer = pointer -> next;

}

/\* Allocate memory for the new node and put data in it.\*/

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

(pointer->next)->prev = pointer;

pointer = pointer->next;

pointer->data = data;

pointer->next = NULL;

}

int find(node \*pointer, int key)

{

pointer = pointer -> next; //First node is dummy node.

/\* Iterate through the entire linked list and search for the key. \*/

while(pointer!=NULL)

{

if(pointer->data == key) //key is found.

{

return 1;

}

pointer = pointer -> next;//Search in the next node.

}

/\*Key is not found \*/

return 0;

}

void delet(node \*pointer, int data)

{

/\* Go to the node for which the node next to it has to be deleted \*/

while(pointer->next!=NULL && (pointer->next)->data != data)

{

pointer = pointer -> next;

}

if(pointer->next==NULL)

{

printf("Element %d is not present in the list\n",data);

return;

}

/\* Now pointer points to a node and the node next to it has to be removed \*/

node \*temp;

temp = pointer -> next;

/\*temp points to the node which has to be removed\*/

pointer->next = temp->next;

temp->prev = pointer;

/\*We removed the node which is next to the pointer (which is also temp) \*/

free(temp);

/\* Beacuse we deleted the node, we no longer require the memory used for it .

free() will deallocate the memory.

\*/

return;

}

void print(node \*pointer)

{

if(pointer==NULL)

{

return;

}

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

print(pointer->next);

}

int main()

{

/\* start always points to the first node of the linked list.

temp is used to point to the last node of the linked list.\*/

node \*start,\*temp;

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

temp = start;

temp -> next = NULL;

temp -> prev = NULL;

/\* Here in this code, we take the first node as a dummy node.

The first node does not contain data, but it used because to avoid handling special cases

in insert and delete functions.

\*/

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Print\n");

printf("4. Find\n");

while(1)

{

int query;

printf("Enter choice:");

scanf("%d",&query);

if(query==1)

{

int data;

printf("Enter data:");

scanf("%d",&data);

insert(start,data);

}

else if(query==2)

{

int data;

scanf("%d",&data);

delet(start,data);

}

else if(query==3)

{

printf("The list is ");

print(start->next);

printf("\n");

}

else if(query==4)

{

int data;

scanf("%d",&data);

int status = find(start,data);

if(status)

{

printf("Element Found\n");

}

else

{

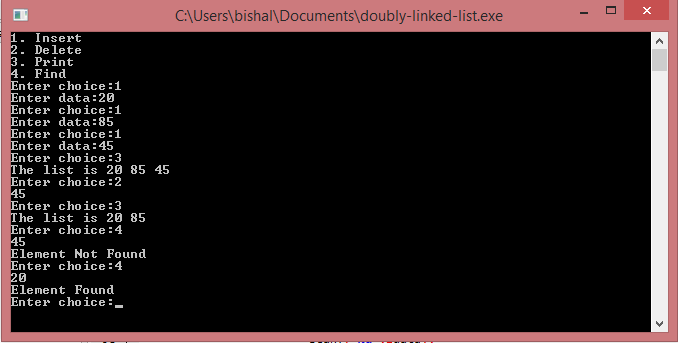
printf("Element Not Found\n");

}

}

}

}



6. Write a program to implement Circular Doubly Linked List with comments on each line.

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*prev, \*next;

};

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

struct node \*createNode(int);

void insertNode(int);

void deleteNode(int);

void display();

struct node\* createNode(int data) {

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

ptr->data = data;

ptr->prev = NULL;

ptr->next = NULL;

return (ptr);

}

/\* insertion in circular linked list \*/

void insertNode(int data) {

struct node \*temp, \*ptr = createNode(data);

/\* list is empty \*/

if (!head) {

head = ptr;

head->next = head;

head->prev = head;

tail = head;

return;

} else {

/\* only one node present in list \*/

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

temp = head;

ptr->next = temp;

ptr->prev = temp->prev;

temp->prev = ptr;

temp->next = ptr;

tail = ptr;

} else {

/\* do insertion next to head \*/

temp = head->next;

ptr->next = temp;

ptr->prev = temp->prev;

temp->prev->next = ptr;

temp->prev = ptr;

}

}

}

void deleteNode(int data) {

struct node \*ptr, \*temp = head;

/\* if list is not present \*/

if (head == NULL) {

printf("Data unavailable\n");

return;

} else if (temp->data == data) {

/\* deleting head node \*/

if (head == tail) {

temp->prev = NULL;

temp->next = NULL;

free(temp);

head = tail = NULL;

} else {

temp->next->prev = tail;

tail->next = temp->next;

head = temp->next;

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

free(temp);

}

} else {

while (temp->next != head && temp->data != data) {

ptr = temp;

temp = temp->next;

}

if (temp->next == head && temp->data != data) {

printf("Given data unvavailable in list\n");

return;

} else if (temp->next != head && temp->data == data) {

/\* deleting any node in between head & tail \*/

ptr->next = temp->next;

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

temp->next = NULL;

temp->prev = NULL;

free(temp);

printf("Data deleted successfully\n");

} else if (temp->next == head && temp->data == data) {

/\* deleting the tail node \*/

ptr->next = temp->next;

temp->next->prev = ptr;

tail = ptr;

free(temp);

printf("Data deleted successfully\n");

}

}

}

/\* traversing the list \*/

void display() {

struct node \*ptr = head;

int i = 0;

while (ptr) {

printf("%-3d\t", ptr->data);

if (ptr->next == head)

break;

ptr = ptr->next;

i++;

}

printf("\n");

}

int main() {

int ch, data;

while (1) {

printf("1.Insertion\t2.Deletion\n");

printf("3.Display\t4.Exit\n");

printf("Enter ur choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter data to insert:");

scanf("%d", &data);

insertNode(data);

break;

case 2:

printf("Enter data to delete:");

scanf("%d", &data);

deleteNode(data);

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("please enter right option\n");

break;

}

}

return 0;

}

