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

// Program to Implement Queue Data Structure using Linked List

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

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

void enq(int data); //Defining function for enquing

void deq(); //Defining function for dequing

void display(); //Defininf function for displaying the elements

void create(); // Defining function for creating a node

int count = 0;

int main()

{

int no, ch, e;

//Printing menu

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Exit");

printf("\n 4 - Display");

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:

exit(0);

case 4:

display();

break;

default:

printf("Invalid choice ");

break;

}

}

return 0;

}

// Create an empty queue

void create()

{

front = rear = NULL;

}

// 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 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--;

}

1. 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 0;

}

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;

scanf("%d",&query);

if(query==1)

{

int 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");

}

}

}

}

1. Write a program to implement queue as a circular list with comments on each line.

#include<stdio.h>

#include<conio.h>

typedef struct cqueue

{

int info;

struct cqueue \*next;

}node;

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

void insert();

void delet();

void main()

{

int ch;

do

{

clrscr();

printf("\npress 1 to insert the element : ");

printf("\npress 2 to delete the element : ");

printf("\npress 4 to exit from main : ");

printf("\nEnter choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delet();

break;

case 3:

exit(1);

default:

printf("\nInvalid choice :");

}

getch();

}while(1);

}

void insert()

{

node \*newnode;

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

printf("\nEnter the node value : ");

scanf("%d",&newnode->info);

newnode->next=NULL;

if(rear==NULL)

front=rear=newnode;

else

{

rear->next=newnode;

rear=newnode;

}

rear->next=front;

}

void delet()

{

temp=front;

if(front==NULL)

printf("\nUnderflow :");

else

{

if(front==rear)

{

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

front=rear=NULL;

}

else

{

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

front=front->next;

rear->next=front;

}

temp->next=NULL;

free(temp);

}

}

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

#include <stdio.h>

#include <stdlib.h>

struct node

{

struct node \*prev;

int n;

struct node \*next;

}\*h,\*temp,\*temp1,\*temp2,\*temp4;

void insert1();

void insert2();

void insert3();

void disp();

void delet();

int count = 0;

void main()

{

int ch;

h = NULL;

temp = temp1 = NULL;

printf("\n 1 - Insert at beginning");

printf("\n 2 - Insert at end");

printf("\n 3 - Insert at position i");

printf("\n 4 - Delete at i");

printf("\n 5 - Display");

printf("\n 6- End");

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert1();

break;

case 2:

insert2();

break;

case 3:

insert3();

break;

case 4:

delet();

break;

case 5:

disp();

break;

case 6:

exit(0);

default:

printf("\n Wrong choice menu");

}

}

}

/\* TO create an empty node \*/

void create()

{

int data;

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

temp->prev = NULL;

temp->next = NULL;

printf("\n Enter value to node : ");

scanf("%d", &data);

temp->n = data;

count++;

}

/\* TO insert at beginning \*/

void insert1()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp->next = h;

h->prev = temp;

h = temp;

}

}

/\* To insert at end \*/

void insert2()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp1->next = temp;

temp->prev = temp1;

temp1 = temp;

}

}

/\* To insert at any position \*/

void insert3()

{

int pos, i = 2;

printf("\n Enter position to be inserted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Position out of range to insert");

return;

}

if ((h == NULL) && (pos != 1))

{

printf("\n Empty list cannot insert other than 1st position");

return;

}

if ((h == NULL) && (pos == 1))

{

create();

h = temp;

temp1 = h;

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

create();

temp->prev = temp2;

temp->next = temp2->next;

temp2->next->prev = temp;

temp2->next = temp;

}

}

/\* To delete an element \*/

void delet()

{

int i = 1, pos;

printf("\n Enter position to be deleted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Error : Position out of range to delete");

return;

}

if (h == NULL)

{

printf("\n Error : Empty list no elements to delete");

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

if (i == 1)

{

if (temp2->next == NULL)

{

printf("Node deleted from list");

free(temp2);

temp2 = h = NULL;

return;

}

}

if (temp2->next == NULL)

{

temp2->prev->next = NULL;

free(temp2);

printf("Node deleted from list");

return;

}

temp2->next->prev = temp2->prev;

if (i != 1)

temp2->prev->next = temp2->next;

if (i == 1)

h = temp2->next;

printf("\n Node deleted");

free(temp2);

}

count--;

}

/\* Traverse from beginning \*/

void disp()

{

temp2 = h;

if (temp2 == NULL)

{

printf("List empty to display \n");

return;

}

printf("\n Linked list elements : ");

while (temp2->next != NULL)

{

printf(" %d ", temp2->n);

temp2 = temp2->next;

}

printf(" %d ", temp2->n);

}