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; //defining pointer

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

int frontelement();

void enq(int data);

void deq();

void display();

void create();

void queuesize();

int count = 0;

int main()

{

int no, ch, e; //defning some needed parameters

printf("\n 1. Enque"); // display line for Enque statement

printf("\n 2. Deque"); // display line for Deque statement

printf("\n 3. Front element"); // display line for Front element statement

printf("\n 4. Exit"); // display line for Exit statement

printf("\n 5. Display"); // display line for Display statement

create();

while (1)

{

// Function calling for choice statement

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1: //switch case for entering data of choice option

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2: //switch case for Deque choice

deq();

break;

case 3: //switch case for frontelement choice

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: //switch case for exit choice

exit(0);

case 5: //switch case for display choice for display of the queue

display();

break;

default: // for implement of correct chice

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

break;

}

}

}

//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 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;

}



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

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

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

struct node \* createNode(int data) {

struct node \*newnode;

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

newnode->data = data;

newnode->next = NULL;

}

/\*

\* create dummy head and tail to make

\* insertion and deletion operation simple.

\* While processing data in our circular

\* linked list, skip these dummies.

\*/

void createDummies() {

head = createNode(0);

tail = createNode(0);

head->next = tail;

tail->next = head;

}

/\* insert data next to dummy head \*/

void circularListInsertion(int data) {

struct node \*newnode, \*temp;

newnode = createNode(data);

temp = head->next;

head->next = newnode;

newnode->next = temp;

}

/\* Delete the node that has the given data \*/

void DeleteInCircularList(int data) {

struct node \*temp0, \*temp;

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

printf("Circular Queue/list is empty\n");

}

temp0 = head;

temp = head->next;

while (temp != tail) {

if (temp->data == data) {

temp0->next = temp->next;

free(temp);

break;

}

temp0 = temp;

temp = temp->next;

}

return;

}

/\*

\* travese the circular linked list for

\* the given no of times.

\*/

void display(int count) {

int n = 0;

struct node \*tmp = head;

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

printf("Circular linked list is empty\n");

return;

}

while (1) {

/\* skip the data in dummy head and tail \*/

if (tmp == head || tmp == tail) {

if (tmp == tail) {

n++;

printf("\n");

if (n == count)

break;

} else {

tmp = tmp->next;

continue;

}

} else {

printf("%-3d", tmp->data);

}

tmp = tmp->next;

}

return;

}

int main() {

int data, ch, count;

createDummies();

while (1) {

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter ur choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the data to insert:");

scanf("%d", &data);

circularListInsertion(data);

break;

case 2:

printf("Enter the data to delete:");

scanf("%d", &data);

DeleteInCircularList(data);

break;

case 3:

printf("No of time you want traverse:");

scanf("%d", &count);

display(count);

break;

case 4:

exit(0);

default:

printf("Please enter correct option\n");

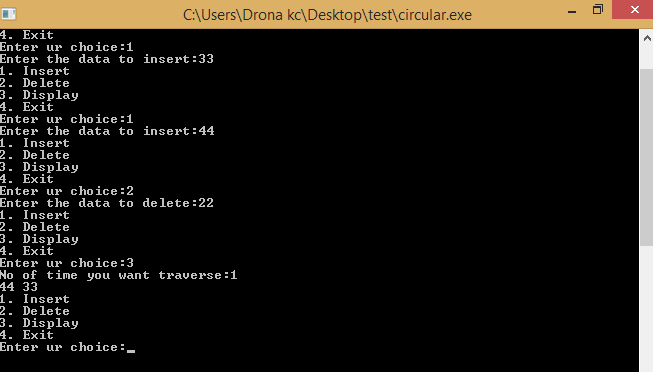
break;

}

}

return 0;

}



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

#include <stdlib.h>

#include<stdio.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

// Decleration of functions

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

int main()

{

int no, ch, e; // Intiliation of parameters

//Displaying the statement that are done on stack

printf("\n 1. Push");

printf("\n 2. Pop");

printf("\n 3. Top");

printf("\n 4. Empty");

printf("\n 5. Exit");

printf("\n 6. Dipslay");

printf("\n 7. Stack Count");

create();

// Switch case for the statement

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:

if (top == NULL)

printf("No elements in stack");

else

{

e = topelement();

printf("\n Top element : %d", e);

}

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

stack\_count();

break;

default :

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

break;

}

}

}

// Create empty stack

void create()

{

top = NULL;

}

// Count stack elements

void stack\_count()

{

printf("\n No. of elements in stack : %d", count);

}

// Push data into stack

void push(int data)

{

if (top == NULL)

{

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

top->ptr = NULL;

top->info = data;

}

else

{

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

temp->ptr = top;

temp->info = data;

top = temp;

}

count++;

}

// Display stack elements

void display()

{

top1 = top;

if (top1 == NULL)

{

printf("Stack is empty");

return;

}

while (top1 != NULL)

{

printf("%d ", top1->info);

top1 = top1->ptr;

}

}

// Pop Operation on stack

void pop()

{

top1 = top;

if (top1 == NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

else

top1 = top1->ptr;

printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}

// Return top element

int topelement()

{

return(top->info);

}

// Check if stack is empty or not

void empty()

{

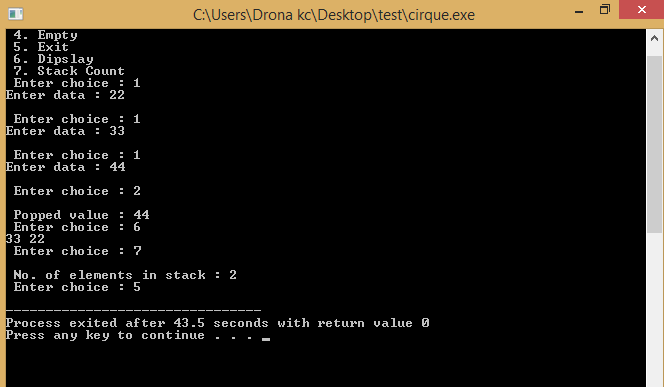
if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}



6. Write a program to implement Circular 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 traversebeg();

void traverseend(int);

void sort();

void search();

void delet();

int count = 0;

int 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 from beginning");

printf("\n 6 - Display from end");

printf("\n 7 - Search for element");

printf("\n 8 - Sort the list");

printf("\n 9 - Exit");

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:

traversebeg();

break;

case 6:

temp2 = h;

if (temp2 == NULL)

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

else

{

printf("\n Reverse order of linked list is : ");

traverseend(temp2->n);

}

break;

case 7:

search();

break;

case 8:

sort();

break;

case 9:

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; /\* Might not need this statement if i == 1 check \*/

if (i == 1)

h = temp2->next;

printf("\n Node deleted");

free(temp2);

}

count--;

}

// Traverse from beginning

void traversebeg()

{

temp2 = h;

if (temp2 == NULL)

{

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

return;

}

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

while (temp2->next != NULL)

{

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

temp2 = temp2->next;

}

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

}

// To traverse from end recursively

void traverseend(int i)

{

if (temp2 != NULL)

{

i = temp2->n;

temp2 = temp2->next;

traverseend(i);

printf(" %d ", i);

}

}

// To search for an element in the list

void search()

{

int data, count = 0;

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty to search for data");

return;

}

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

scanf("%d", &data);

while (temp2 != NULL)

{

if (temp2->n == data)

{

printf("\n Data found in %d position",count + 1);

return;

}

else

temp2 = temp2->next;

count++;

}

printf("\n Error : %d not found in list", data);

}

// To sort the linked list

void sort()

{

int i, j, x;

temp2 = h;

temp4 = h;

if (temp2 == NULL)

{

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

return;

}

for (temp2 = h; temp2 != NULL; temp2 = temp2->next)

{

for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)

{

if (temp2->n > temp4->n)

{

x = temp2->n;

temp2->n = temp4->n;

temp4->n = x;

}

}

}

traversebeg();

}

