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;

void 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()

{

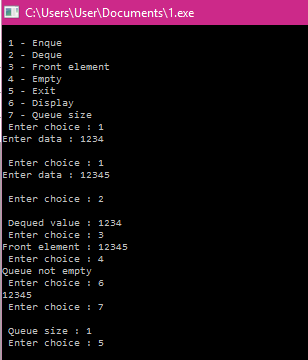
if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}



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 delete(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;

scanf("%d",&query);

if(query==1)

{

int data;

scanf("%d",&data);

insert(start,data);

}

else if(query==2)

{

int data;

scanf("%d",&data);

delete(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

{

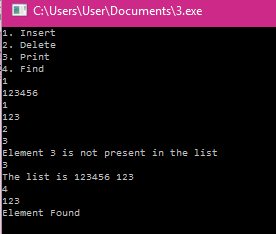
printf("Element Not Found\n");

}

}

}

}



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

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

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

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

void main()

{

int no, ch, e;

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");

printf("\n 8 - Destroy stack");

create();

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;

case 8:

destroy();

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);

}

/\* Destroy entire stack \*/

void destroy()

{

top1 = top;

while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;

top1 = top1->ptr;

}

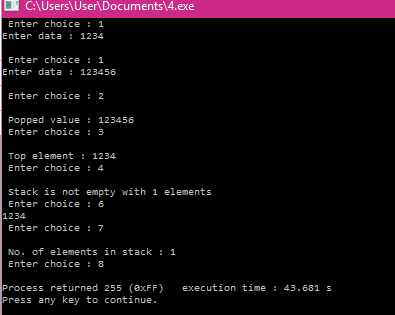
free(top1);

top = NULL;

printf("\n All stack elements destroyed");

count = 0;

}



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

scanf("%d",&query);

if(query==1)

{

int data;

scanf("%d",&data);

insert(start,data);

}

else if(query==2)

{

int data;

scanf("%d",&data);

delete(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

{

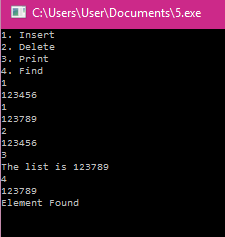
printf("Element Not Found\n");

}

}

}

}



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

#include <stdio.h>

#include <stdlib.h>

struct node

{

int val;

struct node \*next;

struct node \*prev;

};

typedef struct node n;

n\* create\_node(int);

void add\_node();

void insert\_at\_first();

void insert\_at\_end();

void insert\_at\_position();

void delete\_node\_position();

void sort\_list();

void update();

void search();

void display\_from\_beg();

void display\_in\_rev();

n \*new, \*ptr, \*prev;

n \*first = NULL, \*last = NULL;

int number = 0;

void main()

{

int ch;

printf("\n linked list\n");

printf("1.insert at beginning \n 2.insert at end\n 3.insert at position\n4.sort linked list\n 5.delete node at position\n 6.updatenodevalue\n7.search element \n8.displaylist from beginning\n9.display list from end\n10.exit ");

while (1)

{

printf("\n enter your choice:");

scanf("%d", &ch);

switch (ch)

{

case 1 :

insert\_at\_first();

break;

case 2 :

insert\_at\_end();

break;

case 3 :

insert\_at\_position();

break;

case 4 :

sort\_list();

break;

case 5 :

delete\_node\_position();

break;

case 6 :

update();

break;

case 7 :

search();

break;

case 8 :

display\_from\_beg();

break;

case 9 :

display\_in\_rev();

break;

case 10 :

exit(0);

case 11 :

add\_node();

break;

default:

printf("\ninvalid choice");

}

}

}

/\*

\*MEMORY ALLOCATED FOR NODE DYNAMICALLY

\*/

n\* create\_node(int info)

{

number++;

new = (n \*)malloc(sizeof(n));

new->val = info;

new->next = NULL;

new->prev = NULL;

return new;

}

/\*

\*ADDS NEW NODE

\*/

void add\_node()

{

int info;

printf("\nenter the value you would like to add:");

scanf("%d", &info);

new = create\_node(info);

if (first == last && first == NULL)

{

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

last->next = new;

new->prev = last;

last = new;

last->next = first;

first->prev = last;

}

}

/\*

\*INSERTS ELEMENT AT FIRST

\*/

void insert\_at\_first()

{

int info;

printf("\nenter the value to be inserted at first:");

scanf("%d",&info);

new = create\_node(info);

if (first == last && first == NULL)

{

printf("\ninitially it is empty linked list later insertion is done");

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

new->next = first;

first->prev = new;

first = new;

first->prev = last;

last->next = first;

printf("\n the value is inserted at begining");

}

}

/\*

\*INSERTS ELEMNET AT END

\*/

void insert\_at\_end()

{

int info;

printf("\nenter the value that has to be inserted at last:");

scanf("%d", &info);

new = create\_node(info);

if (first == last && first == NULL)

{

printf("\ninitially the list is empty and now new node is inserted but at first");

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

last->next = new;

new->prev = last;

last = new;

first->prev = last;

last->next = first;

}

}

/\*

\*INSERTS THE ELEMENT AT GIVEN POSITION

\*/

void insert\_at\_position()

{

int info, pos, len = 0, i;

n \*prevnode;

printf("\n enter the value that you would like to insert:");

scanf("%d", &info);

printf("\n enter the position where you have to enter:");

scanf("%d", &pos);

new = create\_node(info);

if (first == last && first == NULL)

{

if (pos == 1)

{

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

printf("\n empty linked list you cant insert at that particular position");

}

else

{

if (number < pos)

printf("\n node cant be inserted as position is exceeding the linkedlist length");

else

{

for (ptr = first, i = 1;i <= number;i++)

{

prevnode = ptr;

ptr = ptr->next;

if (i == pos-1)

{

prevnode->next = new;

new->prev = prevnode;

new->next = ptr;

ptr->prev = new;

printf("\ninserted at position %d succesfully", pos);

break;

}

}

}

}

}

/\*

\*SORTING IS DONE OF ONLY NUMBERS NOT LINKS

\*/

void sort\_list()

{

n \*temp;

int tempval, i, j;

if (first == last && first == NULL)

printf("\nlinked list is empty no elements to sort");

else

{

for (ptr = first,i = 0;i < number;ptr = ptr->next,i++)

{

for (temp = ptr->next,j=i;j<number;j++)

{

if (ptr->val > temp->val)

{

tempval = ptr->val;

ptr->val = temp->val;

temp->val = tempval;

}

}

}

for (ptr = first, i = 0;i < number;ptr = ptr->next,i++)

printf("\n%d", ptr->val);

}

}

/\*

\*DELETION IS DONE

\*/

void delete\_node\_position()

{

int pos, count = 0, i;

n \*temp, \*prevnode;

printf("\n enter the position which u wanted to delete:");

scanf("%d", &pos);

if (first == last && first == NULL)

printf("\n empty linked list you cant delete");

else

{

if (number < pos)

printf("\n node cant be deleted at position as it is exceeding the linkedlist length");

else

{

for (ptr = first,i = 1;i <= number;i++)

{

prevnode = ptr;

ptr = ptr->next;

if (pos == 1)

{

number--;

last->next = prevnode->next;

ptr->prev = prevnode->prev;

first = ptr;

printf("%d is deleted", prevnode->val);

free(prevnode);

break;

}

else if (i == pos - 1)

{

number--;

prevnode->next = ptr->next;

ptr->next->prev = prevnode;

printf("%d is deleted", ptr->val);

free(ptr);

break;

}

}

}

}

}

/\*

\*UPDATION IS DONE FRO GIVEN OLD VAL

\*/

void update()

{

int oldval, newval, i, f = 0;

printf("\n enter the value old value:");

scanf("%d", &oldval);

printf("\n enter the value new value:");

scanf("%d", &newval);

if (first == last && first == NULL)

printf("\n list is empty no elemnts for updation");

else

{

for (ptr = first, i = 0;i < number;ptr = ptr->next,i++)

{

if (ptr->val == oldval)

{

ptr->val = newval;

printf("value is updated to %d", ptr->val);

f = 1;

}

}

if (f == 0)

printf("\n no such old value to be get updated");

}

}

/\*

\*SEARCHING USING SINGLE KEY

\*/

void search()

{

int count = 0, key, i, f = 0;

printf("\nenter the value to be searched:");

scanf("%d", &key);

if (first == last && first == NULL)

printf("\nlist is empty no elemnets in list to search");

else

{

for (ptr = first,i = 0;i < number;i++,ptr = ptr->next)

{

count++;

if (ptr->val == key)

{

printf("\n the value is found at position at %d", count);

f = 1;

}

}

if (f == 0)

printf("\n the value is not found in linkedlist");

}

}

/\*

\*DISPLAYING IN BEGINNING

\*/

void display\_from\_beg()

{

int i;

if (first == last && first == NULL)

printf("\nlist is empty no elemnts to print");

else

{

printf("\n%d number of nodes are there", number);

for (ptr = first, i = 0;i < number;i++,ptr = ptr->next)

printf("\n %d", ptr->val);

}

}

/\*

\* DISPLAYING IN REVERSE

\*/

void display\_in\_rev()

{

int i;

if (first == last && first == NULL)

printf("\nlist is empty there are no elments");

else

{

for (ptr = last, i = 0;i < number;i++,ptr = ptr->prev)

{

printf("\n%d", ptr->val);

}

}

}

