**DSPLAB Assignment – 3**

2. Split a Circular Linked List into two halves

Source Code:

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int data;

struct Node \* next;

}Node;

typedef struct CircularLinkedList

{

struct Node \* head;

}CLL;

Node \*LinkNode(int data, Node \*first)

{

Node \*m = (Node \*) malloc(sizeof(Node));

if (m == NULL)

{

return NULL;

}

m->data = data;

m->next = first;

return m;

}

CLL \* getCLL()

{

CLL \* m = (CLL \* ) malloc(sizeof(CLL));

if (m == NULL)

{

return NULL;

}

m->head = NULL;

return m;

}

void insert(CLL \* m)

{ int value;

scanf("%d",&value);

Node \* node = LinkNode(value, m->head);

if ((m->head == NULL))

{

m->head = node;

node->next = m->head;

}

else

{

Node \* temp = m->head;

while (temp->next != m->head)

{

temp = temp->next;

}

temp->next = node;

}

}

void display(CLL \* m)

{

if ((m->head == NULL))

{

printf("Empty Linked List");

}

else

{

printf("\nLinked List Element :");

Node \* temp = m->head;

while (temp != NULL)

{

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

temp = temp->next;

if ((temp == m->head))

{

return;

}

}

printf("Not Circular Linked List");

}

}

CLL \* splitCLL(CLL \* m)

{

CLL \* o = getCLL();

if ((m->head == NULL))

{

printf("Empty Linked List");

}

else

{

Node \* temp = m->head;

Node \* middle = NULL;

while (temp != NULL && temp->next != m->head &&

temp->next->next != m->head)

{

if ((middle == NULL))

{

middle = temp->next;

}

else

{

middle = middle->next;

}

temp = temp->next->next;

}

if ((middle == NULL))

{

if ((temp->next != m->head))

{

o->head = temp->next;

o->head->next = o->head;

m->head->next = temp;

}

}

else

{

o->head = middle->next;

middle->next = m->head;

if ((temp->next == m->head))

{

temp->next = o->head;

}

else if ((temp->next->next == m->head))

{

temp->next->next = o->head;

}

}

}

return o;

}

int main()

{

int n;

CLL \* c1 = getCLL();

while(1)

{

printf("\n1.Insert number to Circular linked list:");

printf("\n2.Display Two Halves of the Circular Linked List and Exit from program:");

printf("\nEnter your Choice:\n");

scanf("%d",&n);

if(n==1){

printf("\nEnter number to Circular linked list:");

insert(c1);

}

else if(n==2)

{

printf("Before spliting: ");

display(c1);

printf("\nAfter spliting: \n");

CLL \* c2 = splitCLL(c1);

printf("\nCircular Linked List1: ");

display(c1);

printf("\nCircular Linked List2: ");

display(c2);

exit(0);

}

else

{

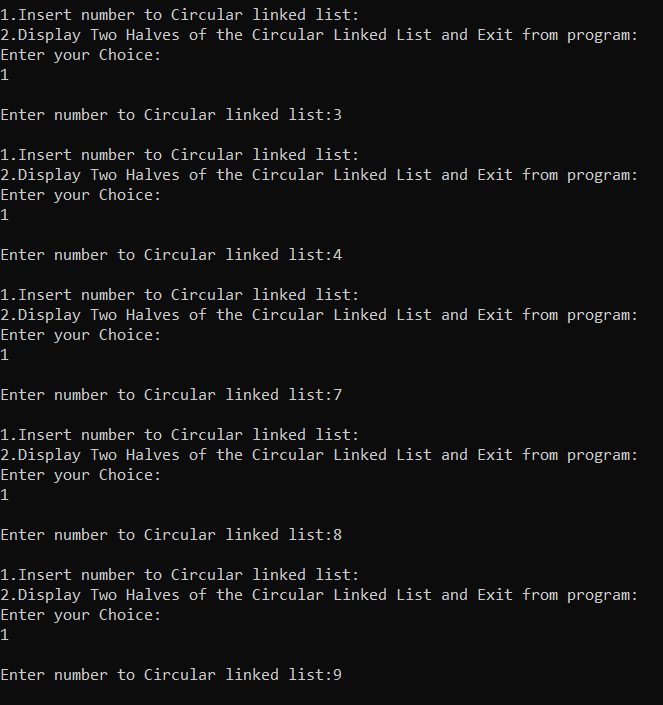
printf("\nPlease select valid choice");

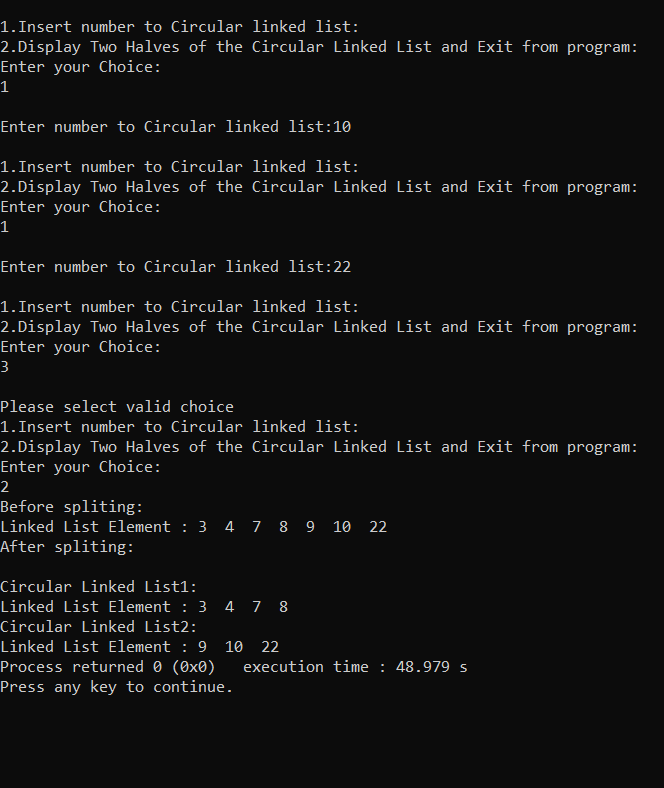
}

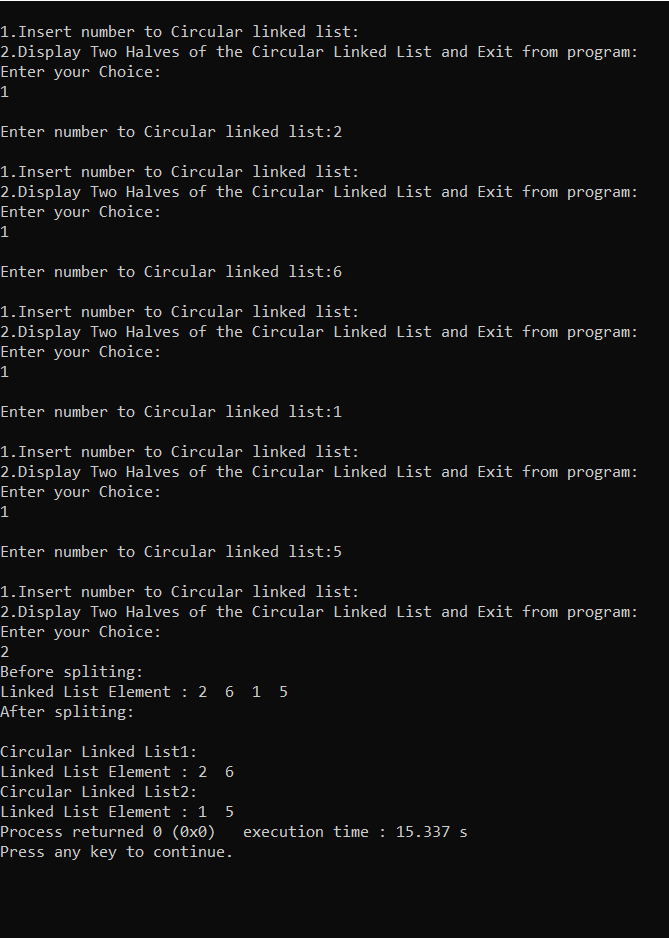
}

}

Output:







3. Write a source code on how to implement a priority queue

Source Code:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int arr[50];

char name[50][20];

int front=-1, rear=-1;

int size;

void insert()

{

if (rear >= size - 1)

{

printf("Queue overflow\n");

}

else{

char temp[20];

int marks;

printf("Enter the name : ");

scanf("%s",temp);

printf("Enter the Gate score ");

if(scanf("%d",&marks)==1){

if(marks<0 || marks >1000)

printf("Enter valid gate score between 0 to 1000 \n");

else{

if ((front == -1) && (rear == -1))

{

++front;

++rear;

strcpy(name[rear],temp);

arr[rear]=marks;

}

else{

int i,j;

for (i = 0; i <= rear; i++)

{

if (marks > arr[i])

{

for (j = rear + 1; j > i; j--)

{

arr[j] = arr[j - 1];

strcpy(name[j],name[j-1]);

}

arr[i] = marks;

strcpy(name[i],temp);

rear++;

return;

}

}

arr[i] = marks;

strcpy(name[i],temp);

rear++;

}

}

}

else{

printf("Invalid input. Only integers allowed\n");

exit(0);

}

}

}

void display()

{

if ((front == -1) && (rear == -1))

{

printf("Queue is empty\n");

}

else{

int i;

for (i=front;i <= rear; i++)

{

printf("%s %d\n",name[i],arr[i]);

}

}

}

void Delete()

{

int i;

int data=0;

if ((front==-1) && (rear==-1))

{

printf("Queue underflow\n");

}

else{

printf("%s %d\n",name[0],arr[0]);

for (i=0; i < rear; i++)

{

arr[i] = arr[i + 1];

strcpy(name[i],name[i+1]);

}

rear--;

if (rear == -1)

front = -1;

}

}

void main()

{

int choice=0;

printf("Enter the size of Queue: ");

if(scanf("%d",&size) == 1){

while (choice!=4)

{

printf("\n1.Insert \n2.Delete \n3.Display \n4.Exit: \nEnter your choice :");

if(scanf("%d", &choice)==1){

if(choice==1)

insert();

else if(choice== 2)

Delete();

else if(choice== 3)

display();

else if(choice==4)

break;

else

printf("Enter valid choice\n");

}

else{

printf("Invalid input. Only integers allowed\n");

exit(0);

}

}

}

else{

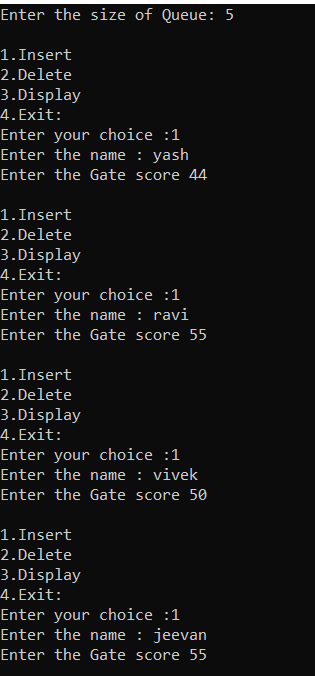
printf("invalid input. Only integers allowed\n");

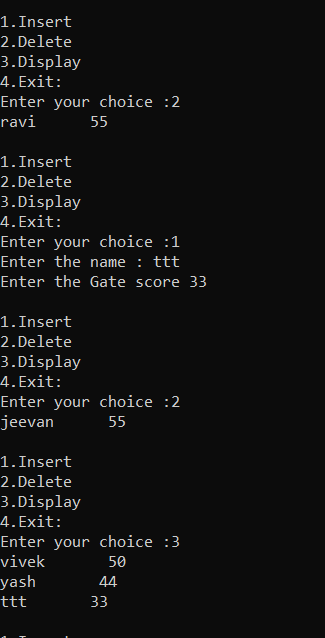
exit(0);

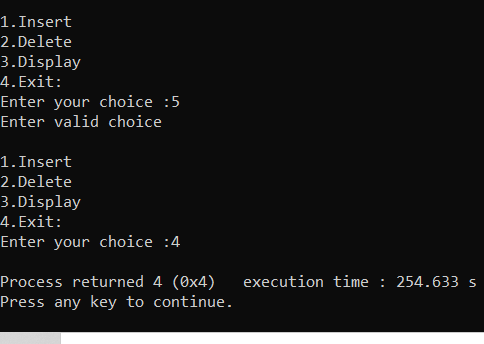
}

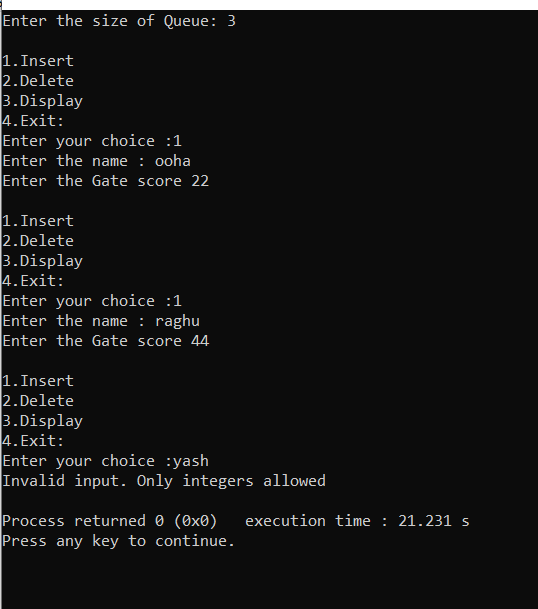
}

Output:









1. Write a source code how to implement efficiently multiple queues in a single array

Source Code:

#include <stdio.h>

#define MAX 20

void insert\_que1(int \*,int \*,int \*,int);

void insert\_que2(int \*,int \*,int \*,int);

void delete\_que1(int \*,int \*,int \*,int \*);

void delete\_que2(int \*,int \*,int \*,int \*);

void display\_que1(int \*,int \*,int \*);

void display\_que2(int \*,int \*,int \*);

int c1=0;

int c2=0;

void main()

{

int que[MAX];

int front1,rear1,front2,rear2;

int value,choice,i;

front1 = rear1 = -1;

front2 = rear2 = 20;

for(i=0;i<MAX;i++)

que[i]=-1;

START:

do

{

printf("\n\tMULTIPLE QUEUE:\n");

display\_que1(que,&front1,&rear1);

display\_que2(que,&front2,&rear2);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n 1. insert to queue1\n");

printf("\n 2. insert to queue2\n");

printf("\n 3. delete to queue1\n");

printf("\n 4. delete to queue2\n");

printf("\n 5. Print elements in Queue1\n");

printf("\n 6. Print elements in Queue2\n");

printf("\n 7. Quit\n");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n\tENTER CHOICE HERE :");

scanf("%d",&choice);

switch(choice)

{

case 1 : printf("Enter the element to be inserted in front\n");

scanf("%d",&value);

insert\_que1(que,&front1,&rear1,value);

break;

case 2 : printf("Enter the element to be inserted in end\n");

scanf("%d",&value);

insert\_que2(que,&front2,&rear2,value);

break;

case 3 : delete\_que1(que,&front1,&rear1,&value);

if(value == -1)

printf("Deque\_2 is empty\n");

else

printf("The value deleted from front is %d\n",value);

break;

case 4 : delete\_que2(que,&front2,&rear2,&value);

if(value == -1)

printf("Deque\_2 is empty\n");

else

printf("The value deleted from front is %d\n",value);

break;

case 5 : display\_que1(que,&front1,&rear1);

break;

case 6 : display\_que2(que,&front2,&rear2);

break;

case 7 : printf("\npress any key to QUIT !!!\n");

goto EXIT;

default: goto START;

}

}while(1);

EXIT:

getch();

}

void insert\_que1(int que[],int \*front1,int \*rear1,int value)

{

int i,k;

if(\*front1 == 0 && \*rear1 == MAX-11)

{

printf("Deque\_1 is full.\n");

return;

}

if(\*rear1 == -1)

{

\*rear1=\*front1=0;

que[\*rear1]=value;

c1++;

return;

}

if(\*rear1 == MAX-11)

{

printf("shifting");

k=\*front1-1;

for(i=1;i<=c1;i++)

{

que[k]=que[k+1];

k++;

}

que[k]=value;

\*rear1=k;

(\*front1)++;

c1++;

}

else

{

(\*rear1)++;

que[\*rear1]=value;

c1++;

}

}

void delete\_que1(int que[],int \*front1,int \*rear1,int \*value)

{

if(\*front1 == -1)

{

printf("Deque\_1 is empty\n");

\*value=-1;

return;

}

\*value = que[\*front1];

que[\*front1]=-1;

if(\*front1 == \*rear1)

{

\*front1=-1;

\*rear1=-1;

\*value=-1;

c1--;

}

else

{

(\*front1)++;

c1--;

}

}

void display\_que1(int \*que,int \*front1,int\*rear1)

{

int i;

printf("\n QUEUE NO 1 :\t");

if(\*front1 == -1 || \*rear1 == -1)

printf("Deque\_1 is empty\n");

else

{ printf("front->");

for(i=\*front1;i<=\*rear1;i++)

printf(" %d",que[i]);

printf(" <-rear");

}

}

void insert\_que2(int que[],int \*front2,int \*rear2,int value)

{

int i,k;

if(\*front2 == MAX-1 && \*rear2 == MAX-10)

{

printf("Deque\_2 is full.\n");

return;

}

if(\*rear2 == 20)

{

\*rear2=\*front2=19;

que[\*rear2]=value;

c2++;

return;

}

if(\*rear2 == MAX-10)

{

printf("shifting");

k=\*front2-1;

for(i=1;i<=c2;i++)

{

que[k]=que[k-1];

k--;

}

que[k]=value;

\*rear2=k;

(\*front2)--;

c2++;

}

else

{

(\*rear2)--;

que[\*rear2]=value;

c2++;

}

}

void delete\_que2(int que[],int \*front2,int \*rear2,int \*value)

{

if(\*front2 == 20)

{

printf("Deque\_1 is empty\n");

\*value=-1;

return;

}

\*value = que[\*front2];

que[\*front2]=-1;

if(\*front2 == \*rear2)

{

\*front2=20;

\*rear2=20;

\*value=-1;

c2--;

}

else

{

(\*front2)--;

c2--;

}

}

void display\_que2(int \*que,int \*front2,int\*rear2)

{

int i;

printf("\n QUEUE NO 2 :\t");

if(\*front2 == 20 || \*rear2 == 20)

printf("Deque\_2 is empty\n");

else

{ printf("front->");

for(i=\*front2;i>=\*rear2;i--)

printf(" %d",que[i]);

printf(" <-rear");

}

}

Output:

