**DS ASSIGNMENT 2**

Name: Vivian Vijay Ludrick

Branch : SE Comps A - Batch C

Roll No: 9914

Circular Queue Implementation

1. Without counter

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

*// creating a structure for the queue to store the float.*

typedef struct

{

int a[SIZE];

int front, rear;

} Queue;

*/\**

*adding elements to the queue*

*enqueue>> queue, element*

*if rear+1 == front -> queue full ; end*

*put rear =rear++%SIZE ; put in rear*

*if front == -1 -> front = 0*

*\*/*

void enqueue(Queue \**q*, int *ele*)

{

*// if the queue is full, then*

if ((*q*->rear + 1) % SIZE == *q*->front)

{

printf("\nQueue Full\n");

printf("----------------------------------------------------------------");

return;

}

*q*->rear = (*q*->rear + 1) % SIZE;

*q*->a[*q*->rear] = *ele*;

*// front should only be updated when size is -1 and when elements are dequeued*

if (*q*->front == -1)

{

*q*->front = 0;

}

printf("Element enqueued is %d\n", *ele*);

printf("----------------------------------------------------------------");

}

*/\**

*displays the elements of the queue*

*display>> queue*

*if rear< front-> print 0 to rear and then front to Size-1 ( more space)*

*(another logic)0 to SIZE -1 skip i>rear && <front(more time)*

*if front< rear -> print from front to rear*

*\*/*

void display(Queue *q*)

{

if (*q*.front == -1)

{

printf("\nQueue Empty");

return;

}

else if (*q*.rear < *q*.front)

{

*// front to end*

printf("\nElements are:\n");

for (int i = *q*.front; i < SIZE; i++)

{

printf("\t%d\n", *q*.a[i]);

}

*// 0 to rear*

for (int i = 0; i <= *q*.rear; i++)

{

printf("\t%d\n", *q*.a[i]);

}

}

else

{

printf("\nElements are:\n");

for (int i = *q*.front; i <= *q*.rear; i++)

{

printf("\t%d\n", *q*.a[i]);

}

}

printf("----------------------------------------------------------------");

}

*// remove the first element in the queue and returns it*

int dequeue(Queue \**q*)

{

if (*q*->front == -1)

{

printf("\nQueue Empty\n3");

printf("----------------------------------------------------------------");

return -1;

}

int ele = *q*->a[*q*->front];

if (*q*->front == *q*->rear)

{

*// this is done specifically for isEmpty() and dequeue() logic.*

*q*->front = *q*->rear = -1;

printf("The queue has been reset as there are no more elements in the queue.\n");

}

else

{

*q*->front = (*q*->front + 1) % SIZE;

}

return ele;

}

*/\**

*returns the size of the queue*

*if rear < front then Size - (front- rear)*

*else rear - front \*/*

int getSize(Queue \**q*)

{

if (*q*->front == -1)

{

printf("\nThe queue is empty\n");

printf("----------------------------------------------------------------");

return 0;

}

int size = *q*->rear < *q*->front ? SIZE - *q*->front + *q*->rear + 1 : *q*->rear - *q*->front + 1;

printf("\nSize is %d\n", size);

printf("----------------------------------------------------------------");

return size;

}

*// checks whether the size is empty or not*

void isEmpty(Queue *q*)

{

if (*q*.front == -1)

{

printf("\nQueue is empty\n");

}

else

{

printf("\nNot Empty\n");

}

printf("----------------------------------------------------------------");

}

*// check whether the queue is full or not*

void isFull(Queue *q*)

{

if ((*q*.rear + 1) % SIZE == *q*.front)

{

printf("\nFull\n");

}

else

{

printf("\nNot Full\n");

}

printf("----------------------------------------------------------------");

}

*// start of the main function*

int main()

{

Queue q1;

q1.front = q1.rear = -1; *// initialising an empty queue*

int choice, ele;

do

{

printf("\nMenu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Check Empty\n");

printf("5. Check Full\n");

printf("6. Get Size\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

printf("----------------------------------------------------------------");

switch (choice)

{

case 1:

printf("\nEnter element to enqueue: ");

scanf("%d", &*ele*);

enqueue(&*q1*, ele);

break;

case 2:

ele = dequeue(&*q1*);

if (ele != -1)

printf("\nRemoved element is: %d\n", ele);

printf("----------------------------------------------------------------");

break;

case 3:

display(q1);

break;

case 4:

isEmpty(q1);

break;

case 5:

isFull(q1);

break;

case 6:

getSize(&*q1*);

break;

case 7:

printf("\nExiting program...\n");

exit(0);

default:

printf("\nInvalid choice.\n");

printf("----------------------------------------------------------------");

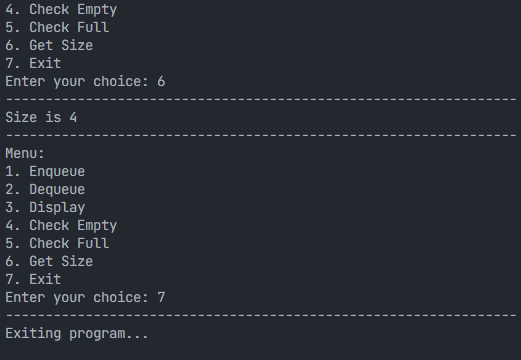
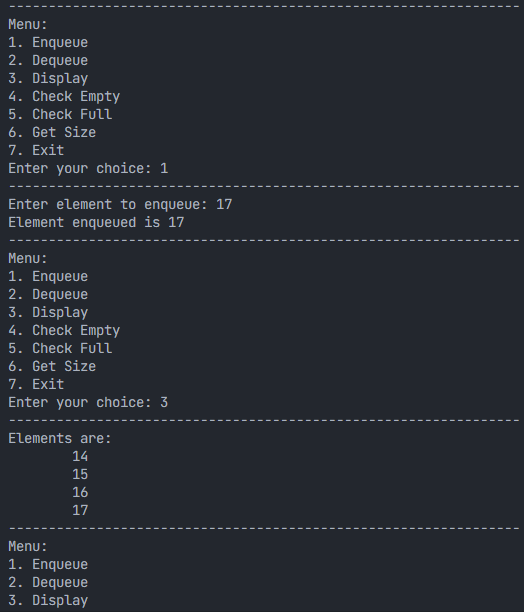
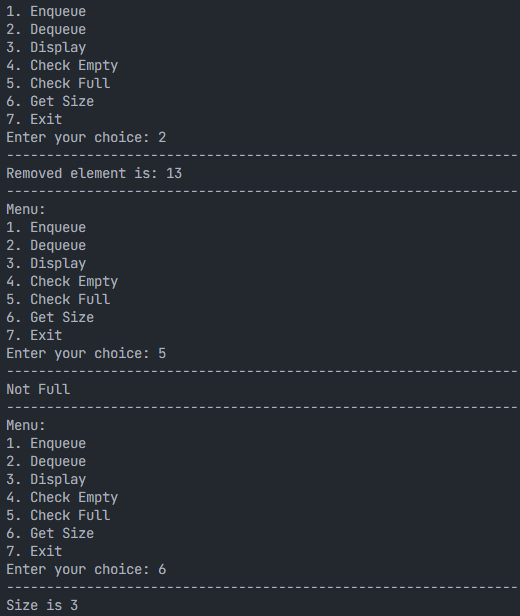
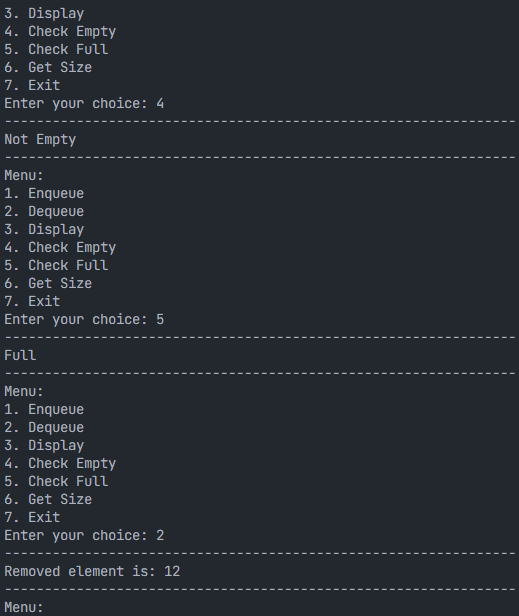
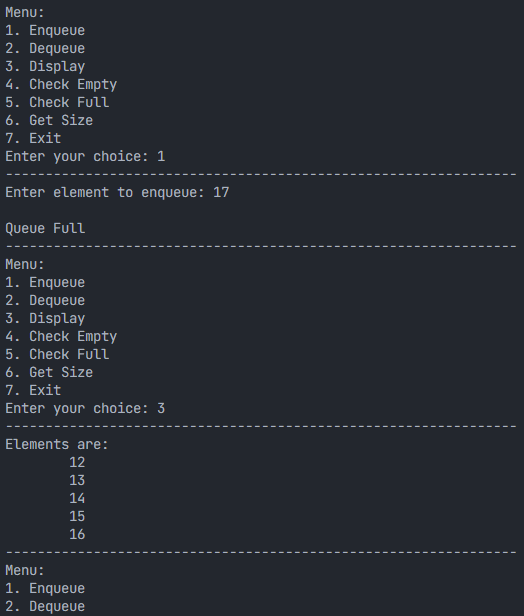
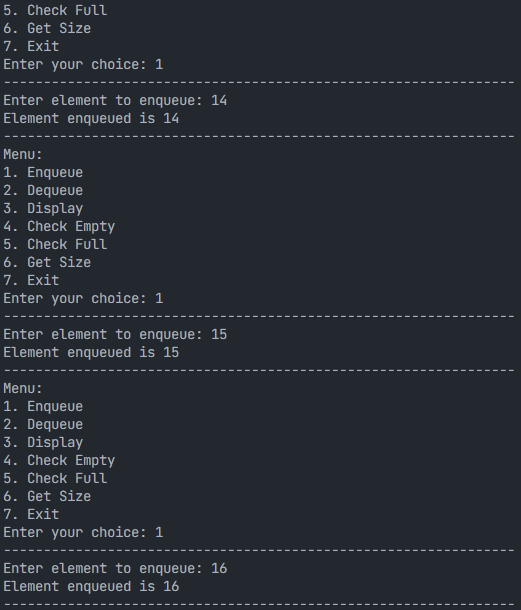
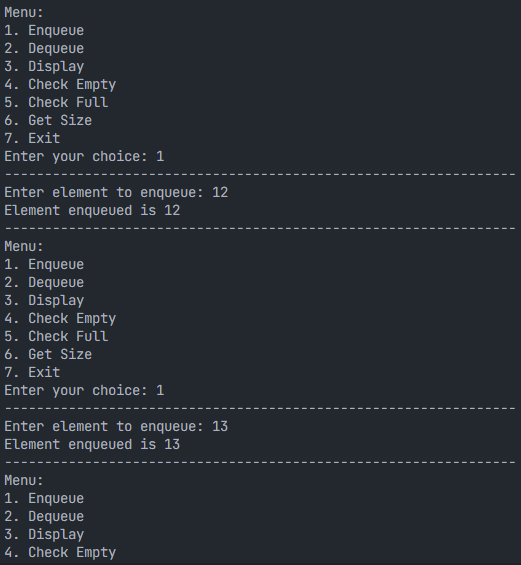
}

} while (choice != 7);

return 0;

}

**OUTPUT:**

****

**Q2. Circular queue implementation with counter**

#include <stdio.h>

#define SIZE 5

*// creating a structure for the queue to store the float.*

typedef struct

{

int a[SIZE];

int front, rear, count;

} Queue;

*// adding elements to the queue*

void enqueue(Queue \**p*, int *ele*)

{

*// if the queue is full, then*

if (*p*->count == SIZE)

{

printf("\nQueue Full\n");

printf("----------------------------------------------------------------");

return;

}

*p*->rear = (*p*->rear + 1) % SIZE;

*p*->a[*p*->rear] = *ele*;

*p*->count++; *// count incremented for every element added*

*// front should only be updated when size is -1 and when elements are dequeued*

if (*p*->front == -1)

{

*p*->front = 0;

}

printf("Element enqueued is %d\n", *ele*);

printf("----------------------------------------------------------------");

}

*/\**

*displays the elements of the queue*

*\*/*

void display(Queue *q*)

{

if (*q*.front == -1)

{

printf("\nQueue Empty");

return;

}

int i;

printf("\nElements are:\n");

for (i = *q*.front; i != *q*.rear; i = (i + 1) % SIZE)

{

printf("\t%d\n", *q*.a[i]);

}

printf("\t%d\n", *q*.a[i]); *// since the loop won't run for q.rear*

printf("----------------------------------------------------------------");

}

*// remove the first element in the queue and returns it*

int dequeue(Queue \**q*)

{

if (*q*->count == 0)

{

printf("\nQueue Empty\n");

printf("----------------------------------------------------------------");

return -1;

}

int ele = *q*->a[*q*->front];

if (*q*->front == *q*->rear)

{

*q*->front = *q*->rear = -1;

*q*->count = 0;

printf("The queue has been reset as there are no more elements in the queue.\n");

}

else

{

*q*->front = (*q*->front + 1) % SIZE;

}

*q*->count--;

return ele;

}

*// returns the size of the queue*

int getSize(Queue *q*)

{

printf("\nSize is %d\n", *q*.count);

printf("----------------------------------------------------------------");

return *q*.count;

}

*// checks whether the size is empty or not*

void isEmpty(Queue *q*)

{

if (*q*.count == 0)

{

printf("\nQueue is empty\n");

}

else

{

printf("\nNot Empty\n");

}

printf("----------------------------------------------------------------");

}

*// check whether the queue is full or not*

void isFull(Queue *q*)

{

if (*q*.count == SIZE)

{

printf("\nFull\n");

}

else

{

printf("\nNot Full\n");

}

printf("----------------------------------------------------------------");

}

*// start of the main function*

int main()

{

Queue q1;

q1.front = q1.rear = -1; *// initialising an empty queue*

q1.count = 0;

int choice, ele;

do

{

printf("\nMenu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Check Empty\n");

printf("5. Check Full\n");

printf("6. Get Size\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

printf("----------------------------------------------------------------");

switch (choice)

{

case 1:

printf("\nEnter element to enqueue: ");

scanf("%d", &*ele*);

enqueue(&*q1*, ele);

break;

case 2:

ele = dequeue(&*q1*);

if (ele != -1)

printf("\nRemoved element is: %d\n", ele);

printf("----------------------------------------------------------------");

break;

case 3:

display(q1);

break;

case 4:

isEmpty(q1);

break;

case 5:

isFull(q1);

break;

case 6:

getSize(q1);

break;

case 7:

break;

default:

printf("\nInvalid choice.\n");

printf("----------------------------------------------------------------");

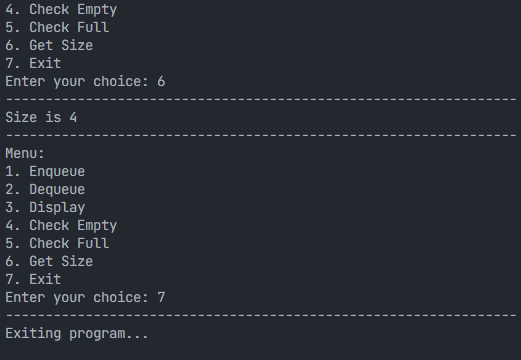
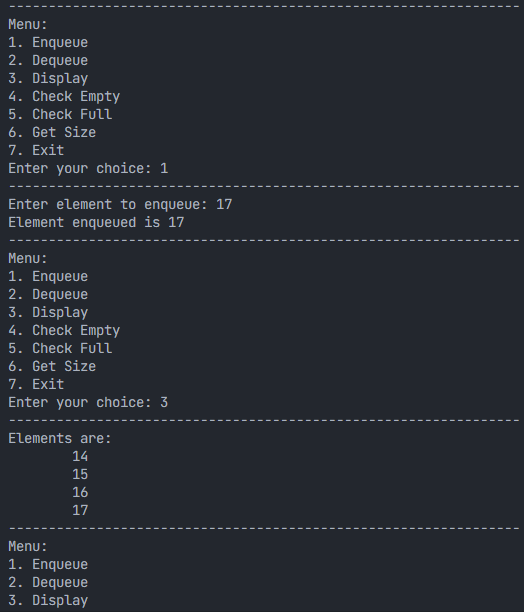
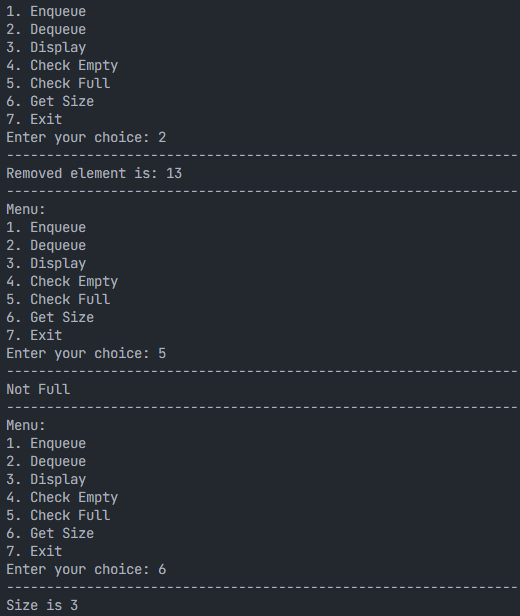
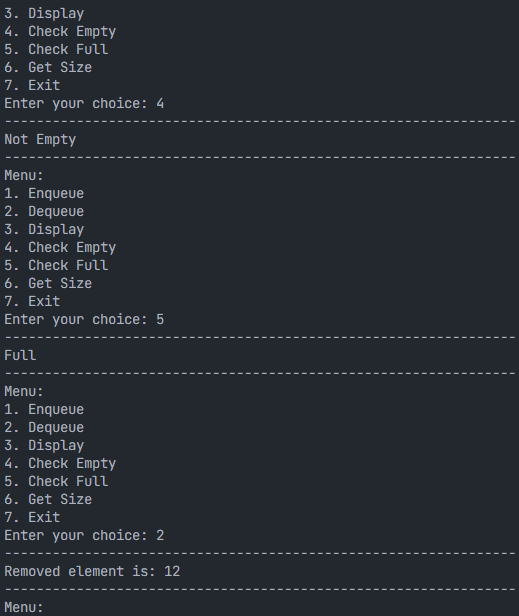
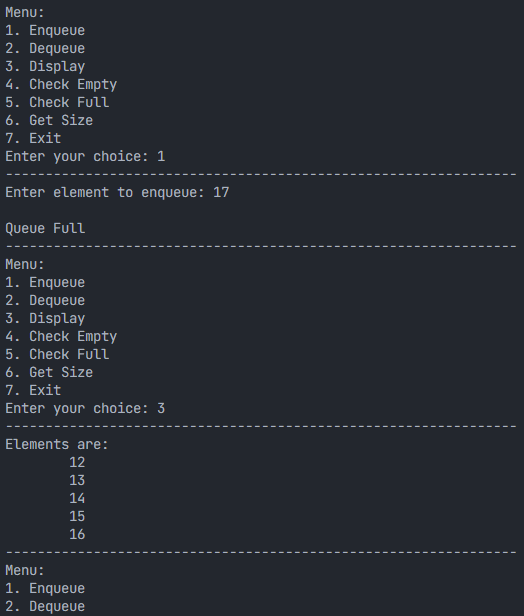
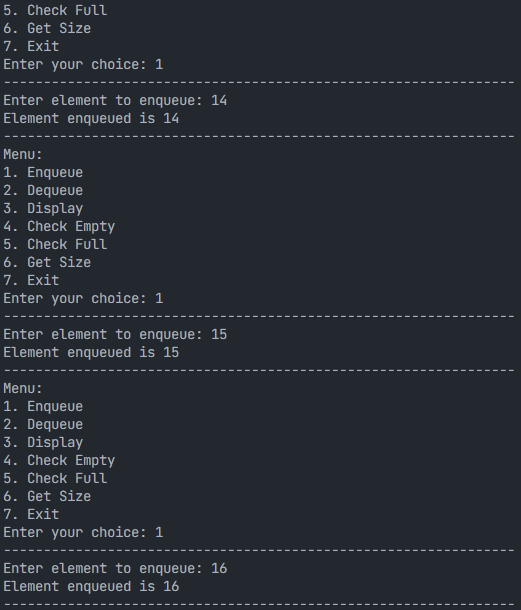
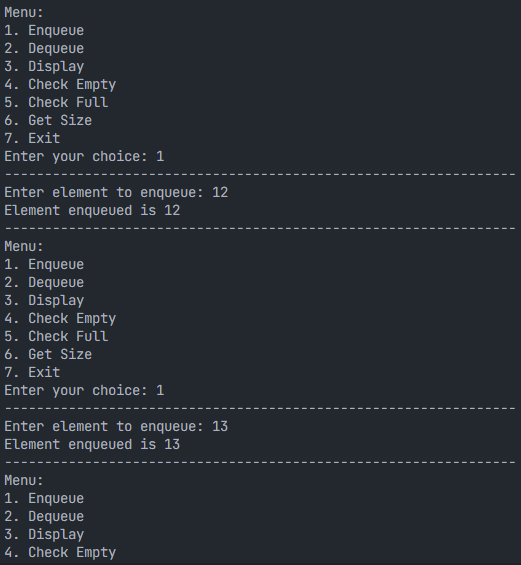
}

} while (choice != 7);

return 0;

}

**OUTPUT:**

****