**ASSIGNMENT – 18**

**1.PROBLEM STATEMENT**

Write a program in C to implement linear queue using array.

**2.ALGORITHMS**

Algorithm **Insert\_In\_Queue**

**Input:** The pointer to the array ‘arr’ representing the queue, the ‘front’ and ‘rear’ pointers pointing to the first and last element of the queue respectively, and the element ‘item’ to be inserted.

**Output:** The element ‘item’ inserted into the queue at ‘rear’.

**Remarks:** The ‘front’ and ‘rear’ variables must be passed as pointers. When queue is empty, ‘front’ and ‘rear’ are both equal to 0.

**Steps:**

1. rear=rear+1 //increment rear by 1
2. arr[rear]=item //insert item at the position of rear in queue
3. **If**(front=0) **then** //if front is 0,that is during first insertion
4. front=front+1 //increment front by 1
5. **EndIf**
6. **Stop**

Algorithm **Remove\_From\_Queue**

**Input:** The pointer to the array ‘arr’ representing the queue, the ‘front’ and ‘rear’ pointers pointing to the first and last element of the queue respectively.

**Output:** The element at ‘front’ is removed from the queue and is returned.

**Remarks:** The ‘front’ and ‘rear’ variables must be passed as pointers. When queue is empty, ‘front’ and ‘rear’ are both equal to 0.

**Steps:**

1. value=arr[front] //store the element at front in value
2. front=front+1 //increment front by 1
3. **If**(front>rear) **then** //if front exceeds rear,that is queue is empty
4. front=rear=0 //set front and rear to empty state
5. **EndIf**
6. **Return** value
7. **Stop**

**3.SOURCE CODE**

#include <stdio.h>

#include <stdlib.h>

//function to display queue

void display(int \*arr, int \*front, int \*rear)

{

int i;

if (\*front > \*rear) //if front exceeds rear

{

printf("Queue is empty");

return;

}

printf("\nQueue Elements: ");

for (i = \*front; i <= \*rear; i++)//traverse from front to rear

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

}

//function to insert an element in queue

void qinsert(int \*arr, int \*front, int \*rear, int item)

{

(\*rear)++; //increment rear by 1

arr[\*rear] = item; //insert item at rear

printf("INSERTED: %d", item);

if (\*front == -1) //for first insertion

(\*front)++; //increment front by 1

display(arr, front, rear);

}

//function to remove an element from queue

int qremove(int \*arr, int \*front, int \*rear)

{

int val;

val = arr[\*front]; //store element at front in value

(\*front)++; //increment front by 1

display(arr, front, rear);

if (\*front > \*rear) //if front exceeds rear,queue becomes empty

\*front = \*rear = -1; //set front,rear to empty state

return val;

}

int main(void)

{

int \*arr, max, ch, front, rear, item;

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

scanf("%d", &max);

arr = (int \*)calloc(max, sizeof(int));

front = rear = -1;

while (1)

{

printf("\nMENU:");

printf("\n1.INSERT\n2.REMOVE\n3.DISPLAY\n4.EXIT");

printf("\nEnter Your Choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

if (rear == max - 1) //if rear is at the end of the queue

{

printf("Queue Is Full");

break;

}

else

{

printf("Enter the element to be inserted: ");

scanf("%d", &item);

qinsert(arr, &front, &rear, item);

}

break;

case 2:

if (rear == -1) //in empty state, front=rear=-1

{

printf("Queue is already empty...cannot remove");

break;

}

else

{

item = qremove(arr, &front, &rear);

printf("\nREMOVED: %d", item);

}

break;

case 3:

display(arr, &front, &rear);

break;

case 4:

return 0;

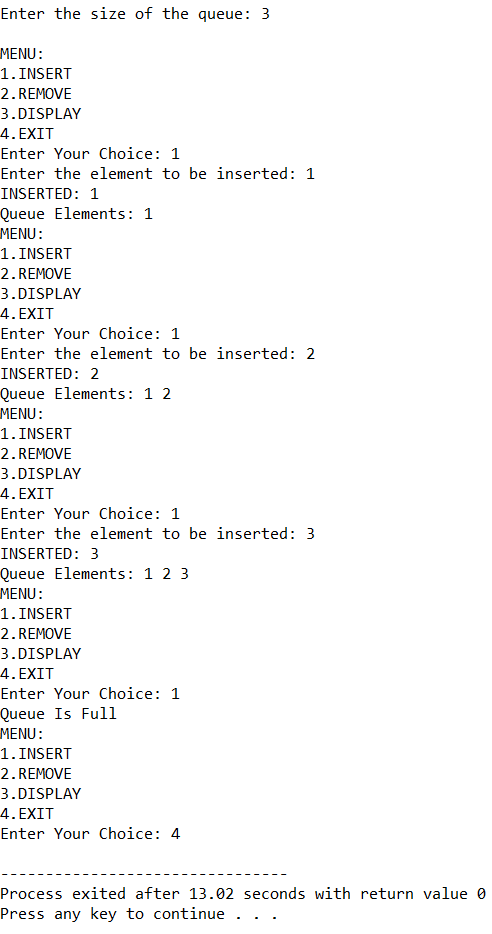
}

}

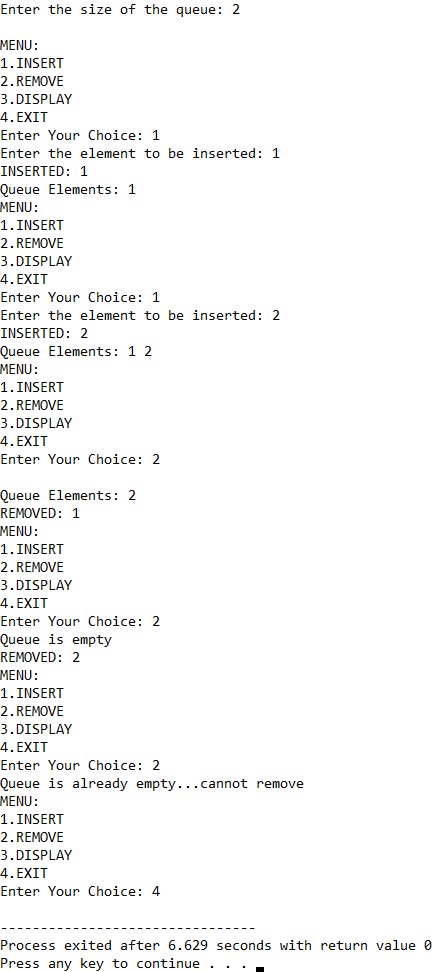
}

**4.OUTPUT**

**SET 1:** Insertion tillqueue full



**SET 2:** Insertion and deletion till queue empty



**5.DISCUSSIONS**

**Variable Description**

* **\*arr:** pointer to the array that represents a queue.
* **Max:** size of the queue.
* **Ch:** variable to take user input in switch-case-default.
* **Front:** pointer to the first element in queue.
* **Rear:** pointer to the las element in queue.
* **Item:** Element to be inserted using Insert\_In\_Queue method.
* **i:** loop counter.

**Uses:**

* In a multiprogramming environment, a CPU has to serve multiple programs, operating systems maintain a ‘priority queue’ in which tasks are pushed with certain priorities, helping the CPU to perform the most important tasks first.
* They can be used to create simulation models of real-life scenarios that involve queueing of data.

**Limitations:**

* The program displayed above generates a linear queue, thus if at one point of time, the front pointer has traversed some distance in the array and there are free memory locations behind it still the queue full state will be called if the rear pointer reaches the end of the queue, thus linear queue implementation is inefficient in terms of memory efficiency.

**Future Scope:**

* The linear queue can be modified into a much more memory effiecient circular queue.
* The array representation can be replace with a linkedlist representation for better memory efficiency.