**NAME:** AGNIV PRAMANICK **SECTION:** A **USN:** 1NT21IS017 **DATE:** 27/02/23

**Q. WRITE A PROGRAM TO IMPLEMENT CIRCULAR QUEUES USING ARRAY.**

**THEORY**

* A circular queue is a queue variation that stores its elements in a circular array. In a circular queue, the front and rear pointers are used to keep track of the indices of the members at the front and rear of the queue, respectively. However, the queue wraps around when the end of the array is reached rather than expanding and contracting as new elements are added or withdrawn.
* We increase the rear pointer and store the element at the corresponding index in the array to add it to the circular queue. The rear pointer wraps around to the beginning of the array if it hits the end of the array.
* We raise the front pointer and return the element at the corresponding index in the array to remove an element from the circular queue. The front pointer wraps around to the beginning of the array if it reaches the array's end.
* One benefit of utilising a circular queue is that since the front pointer can loop around to the beginning of the array, we can reuse the empty space in the array when elements are removed from the front of the queue. Compared to a standard queue, which may leave empty space at the beginning of the array when elements are deleted, this may be more memory-efficient.
* Circular queues can have some drawbacks, though. They have a limited size, which might result in overflow or underflow issues if the queue fills up or empties out, which is one of their main limitations. We must maintain track of the queue's element count and employ a resizing method to adjust the array's size as necessary in order to prevent these mistakes.
* In some cases, especially when memory use is an issue and we want to utilise the unused space in the array, circular queues can be a helpful and effective technique to design queues. They do, however, differ from conventional queues in that they have a defined size and are susceptible to overflow or underflow issues.

**ALGORITHM**

1. ENQUEUE

Step 1: Check if the queue is full by comparing the value of rear with MAX\_SIZE - 1.

Step 2: If the queue is full, print an overflow message and exit.

Step 3: If the queue is not full, increment the value of rear by 1, and add the new element at the position pointed to by rear in the array.

Step 4: If front is equal to -1, set front to 0, as the queue was previously empty.

2. DEQUEUE

Step 1: Check if the queue is empty by comparing the value of front with -1.

Step 2: If the queue is empty, print an underflow message and exit.

Step 3: If the queue is not empty, retrieve the element at the position pointed to by front in the array, and increment the value of front by 1.

Step 4: If front becomes equal to rear, set both front and rear to -1 to mark the queue as empty.

3. DISPLAY

Step 1: Check if the queue is empty by comparing the value of front with -1.

Step 2: If the queue is empty, print a message stating that the queue is empty, and exit.

Step 3: If the queue is not empty, start a loop from front to rear and print each element in the array.

**CODE**

#include <stdio.h>

#define MAX\_SIZE 5 // maximum size of the queue

// function to insert an element in the circular queue

void enqueue(int \*queue, int \*front, int \*rear, int data)

{

// check if the queue is full

if ((\*rear + 1) % MAX\_SIZE == \*front) {

printf("Queue is full.\n");

return;

}

// insert the element

queue[\*rear] = data;

\*rear = (\*rear + 1) % MAX\_SIZE;

}

// function to delete an element from the circular queue

void dequeue(int \*queue, int \*front, int \*rear)

{

// check if the queue is empty

if (\*front == \*rear) {

printf("Queue is empty.\n");

return;

}

// delete the element

printf("Deleted element: %d\n", queue[\*front]);

\*front = (\*front + 1) % MAX\_SIZE;

}

// function to display the elements of the circular queue

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

{

// check if the queue is empty

if (front == rear) {

printf("Queue is empty.\n");

return;

}

// display the elements

printf("Queue elements:\n");

for (int i = front; i != rear; i = (i + 1) % MAX\_SIZE) {

printf("%d\n", queue[i]);

}

}

// main function

int main()

{

int queue[MAX\_SIZE];

int front = 0, rear = 0, choice, data;

while (1) {

// display menu

printf("\nQueue Operations:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

// insert element in the queue

printf("Enter the data to be enqueued: ");

scanf("%d", &data);

enqueue(queue, &front, &rear, data);

break;

case 2:

// delete element from the queue

dequeue(queue, &front, &rear);

break;

case 3:

// display elements of the queue

display(queue, front, rear);

break;

case 4:

// exit the program

printf("Program exited.\n");

return 0;

default:

printf("Invalid choice.\n");

}

}

return 0;

}

**OUTPUT**





