Circular Queue

Circular queue is like a regular queue except it wraps around the end. So it has the advantage of not wasting space whenever element is dequeued without having to shift all elements. Hence it is economical in both space and time.

Circular queue supports operations same as regular queue; enqueue and dequeue.

Algorithms

Initialize an array of size n to hold items in queue and front and back to -1

Enqueue

1. Start
2. if (rear + 1) % n = front
   error "Queue is full"
   else
   increase rear as rear = (rear + 1) % n
   insert item at rear
3. End

Dequeue

1. Start
2. if (rear = front)
   error "Queue is empty"
   else
   front = (front+1) % size
3. Stop

Implementation

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int *data;
    int front, rear, size;
} CircularQueue;

CircularQueue* init_queue(int size) {
    CircularQueue *q = (CircularQueue*) malloc(sizeof(CircularQueue));
    q->data = (int*) malloc(sizeof(int)*size);
    q->size = size;
    q->front = -1;
    q->rear = -1;
```

```
    return q;
}

void enqueue(CircularQueue *q, int x) {
    if ( (q->rear + 1) % q->size == q->front) {
        printf("Error: Queue is full\n");
        return;
    }
    q->rear = (q->rear+1) % q->size;
    q->data[q->rear] = x;
}

void dequeue(CircularQueue *q) {
    if (q->rear == q->front) {
        printf("Error: Queue is empty.\n");
        return;
    }
    q->front = (q->front+1)%q->size;
    printf("Dequeued value: %d\n", q->data[q->front]);
}

void printqueue(CircularQueue *q) {
    if (q->front == q->rear) {
        printf("Queue is empty");
        return;
    }
    for(int i=q->front+1; i<=q->rear; i++) {
        printf("%d ", q->data[i]);
    }
    printf("\n");
}

int main() {
    int n, ch, x;
    printf("CIRCULAR QUEUE IMPLEMENTATION.\n\n");
    printf("Enter the size of queue: ");
    scanf("%d", &n);
    CircularQueue *q = init_queue(n);
    while(1) {
        printf("1. Enqueue.\n"
            "2. Dequeue.\n"
            "3. Print\n"
            "4. Exit\n"
            ">> ");

        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter the data to enqueue: ");
            scanf("%d", &x);
            enqueue(q, x);
            break;
```

```
        case 2:
            dequeue(q);
            break;
        case 3:
            printqueue(q);
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("Invalid option %d\n", ch);
            break;
        }
    }
}
```

OUTPUT

```
CIRCULAR QUEUE IMPLEMENTATION.

Enter the size of queue: 5
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 12
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 15
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 14
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 3
12 15 14
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 2
```

```
Dequeued value: 12
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 3
15 14
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 12
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 13
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 13
Error: Queue is full
1. Enqueue.
2. Dequeue.
3. Print
4. Exit
>> 1
Enter the data to enqueue: 13
Error: Queue is full
```