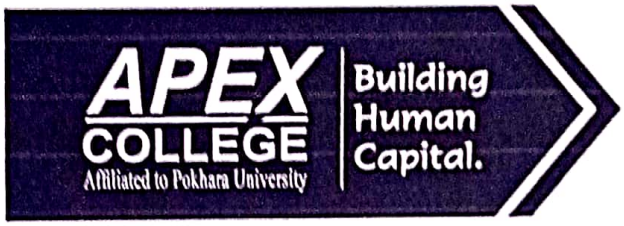
Apex College

BCIS Program

Affiliated to Pokhara University



Data Structure & Algorithms
Lab Report

3. Implementation of Circular Queue

Date: __-__-____

Submitted by:
Ishwor Shrestha
Roll no.: 2018-BCIS-414

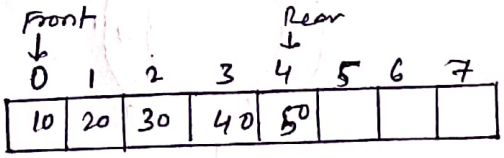
Submitted to:
Pravakar Ghimire, &
Anmol Shrestha
Apex College

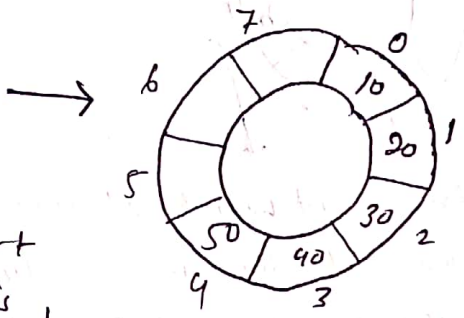
Lab 3 Objectives

- To understand circular queue and implementation of it's operations.
- To create and execute algorithms of enqueue and dequeue operations of circular queue.

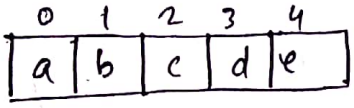
Introduction : Circular Queue

Circular queue is a linear data structure in which the operations are performed based on FIFO sequence and last position is connected back to the first position of queue to make circle. It is also called 'Ring Buffer'.





In above normal queue, we can insert element until queue becomes full, but we cannot insert the next element even if there is a space in front of queue



F = 0, 1, 2, 3, 4

R = -1, 0, 1, 2, 3

Now, F > R

then, if we try to insert, it shows full state

and

if we try to ~~insert~~ delete, it shows empty

that is called. Absurd Condition,

So, to solve this problem, we use Circular queue where we have to sacrifice either one element or one space in queue

Applications of Circular Queue

- CPU scheduling
- Memory Management
- Traffic Management

In. Circular queue, we use Modulo Operation to find the value of Rear and Front pointer,

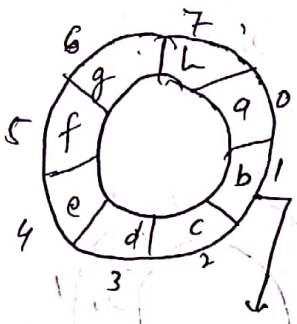
Front = (Front+1) % MAX

Rear = (Rear+1) % MAX

We can define Full and Empty condition after sacrification of either a element or a space in a circular queue.

* One Element Sacrifies Method

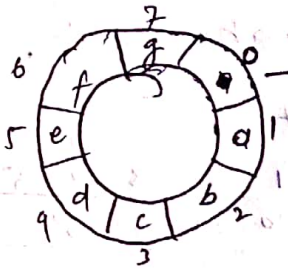
R = F



This element won't get service, because it shows 'queue is full' & 'queue is empty' at some time

* One Space sacrifies Method

R = 0 ; F = 1



but

Now, we can define

(R+1) % MAX == F (i.e. R=1, F=1) as a empty queue, and.

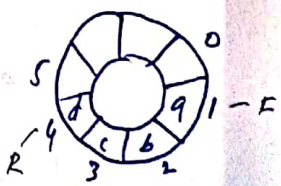
(R+2) % MAX == F (i.e. R=0, F=1) as a full queue.

∴ sacrifying one space is better than sacrifying a resource from queue, Hence, we use one space sacrifien method.

A program to demonstrate operations in circular Queue.

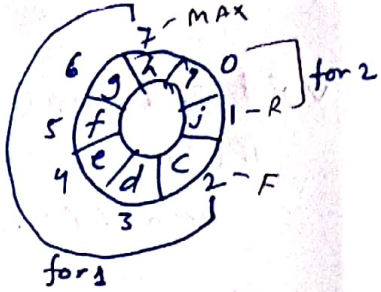
```
#include <stdio.h>
#include <stdlib.h>
# define MAX 10 or 8

struct queue {
   int items[MAX];
   int rear, front;
};
typedef struct queue queue;
```

```
void enqueue(queue *q, int item){
  if((q->rear+1)%MAX == q->front)
    printf("Circular queue is full.\n");
  else{
    q->rear = (q->rear+1)%MAX;
    q->items[q->rear] = item;
  }
}

int dequeue(queue *q){
  int item = -1;
  if((q->rear+1)%MAX == q->front)
    printf("Circular queue is empty.\n");
  else{
    item = q->item[q->front];
    q->front = (q->front+1)%MAX;
  }
  return item;
}

void display(queue *q){
  int i;
  printf("Elements in Circular Queue:");
  if((q->rear+1)%MAX == q->front)
    printf(": is empty.\n");

  else if(q->front <= q->rear){
    for(i=q->front; i<=q->rear; i++)
      printf("%d\n", q->items[i]);
  }
  else{
    for(i=q->front; i<MAX; i++) // 1
      printf("%d\n", q->items[i]);
```



i.e. R ≥ F

```
    for (i=0; i<=q->rear; i++) //2
        printf("%d \n", q->items[i]);
}
```




```
void main (){
    queue q;
    int ch, item;
    q.front = 0;
    q.rear = -1;
    while (1) {
        printf("Enter elements in Circular Queue : \n");
        printf("1. Enqueue. \t 2. Dequeue. \t 3. Display \t 4. Exit \n");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter data : ");
                scanf("%d", &item);
                enqueue(&q, item);
                break;
            case 2:
                item = dequeue(&q);
                printf("%d is dequeued. \n", item);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. \n");
        }
    }
}
```

Activities

① Dequeue

- dequeue operation in circular queue is similar to normal queue's dequeue operation but it used modulo operations to check conditions.

- Check whether the queue is empty or not
  i.e. if ((R+1) % MAX == F)
  printf ("Queue is empty.") & exit

- Otherwise, queue is not ~~full~~ empty
  - set, element = q -> items [q -> F];
  - set, value of front
    i.e. q -> front = (q -> front + 1) % MAX;

  & return item or element,

② Enqueue

- similarly, enqueue also used modulo operations

- check whether queue is full or not
  i.e. if ((q -> rear + 2) % MAX == F)
  printf "Queue is full. \n" & exit

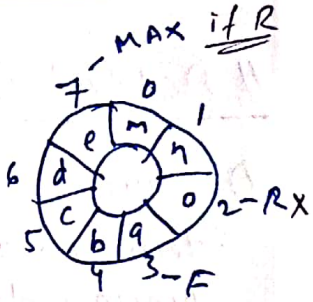
- otherwise, queue is not full
  - set, value of rear
    Rear = (rear + 1) % MAX
  - set value at rear pos
    items [rear] = item

③ Display

- In display function, we have tried a condition which can traverse and ~~check~~ display value till MAX pos of rear but it generates error if rear pointer goes beyond MAX pos in circular queue.

// Function with error generating condition

```
:
else {
  for (i = q->rear ; i >= q->front; i--)
      printf("%d \n", q->items[i]);
}
;
```



→ this condition can traverse only items in between F and MAX if R=MAX.

Now, R is less than MAX, so, that it returns errors.

To overcome this issue, we have used different & multiple method, which is done in program section.

④ Exit

```
- exit(0);
```

#Conclusion

- We have learned about circular queue, difference between normal & circular queue with their own advantages and disadvantages and their implementations.