Queue without a Front Variable (An Easy Approach to Abolish the Use of Circular Queue Using Practical Queue)

Rohan Gupta[1], Puneet Kumar Goyal[2], Mradul Jain[3]

B. Tech, Cse III Year, Abes Engineering College Ghaziabad (Up), India
Sr. Asst. Professor, Abes Engineering College Ghaziabad(Up), India
Asso. Professor, Abes Engineering College Ghaziabad(Up), India

ABSTRACT

This research paper focuses on enhancing the working and efficiency of statically implemented queues,(FIFO, First In First Out data structure) by some additional additive properties to it. If these properties are implemented, can abolish the concept of circular queues in "Statically implemented queues" and will handle the data much more efficiently. This paper mainly focuses on abolishing the traditionally said drawback of the queues i.e. inefficient utilisation which leads wastage of memory by making the successful use of the left over memory left after deletion taking inspiration from real life queue.

Keywords: rear, front, memory.

I. INTRODUCTION

Queue is a practical life based example. In our daily life we came across certain queues like- A queue at the railway reservation counter & queue at temple for Prasad distribution. These examples have one thing in common. The person who is at the front of the queue, will first get its reservation done or would receive Prasad and quits after completion of his/her task. i.e. the queue will start decreasing from the frontal end. If someone has to enter the queue the he/she would go to back. i.e. insertion occurs from rear end.

A queue is a collection of data where the addition of new data occurs from one end, called the rear end, and the removal of existing data items occurs at the other end, commonly called the front end. As an element enters the queue it starts at the rear and makes its way toward the front, waiting until that time when it is the next element to be removed.**[1]**

The most recently added item in the queue must wait at the end of the collection. The item that has been in the collection the longest is at the front. This ordering principle is sometimes called **FIFO**, **first-in first-out**. It is also known as "first-come first-served."

The simplest example of a queue is the typical line that we all participate in from time to time. We wait in a line for a movie, we wait in the check-out line at a grocery store, and we wait in the cafeteria line (so that we can pop the tray stack). Well-behaved lines, or queues, are very restrictive in that they have only one way in and only one way out. There is no jumping in the middle and no leaving before you have waited the necessary amount of time to get to the front. Figure shows a simple queue of data objects.

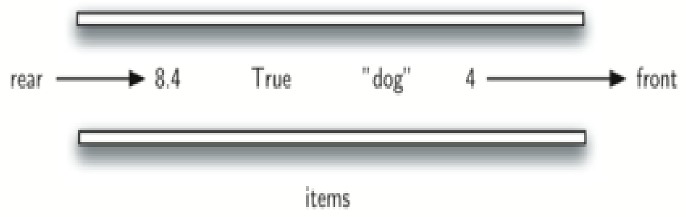
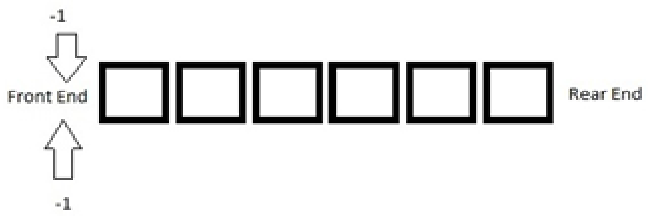


Fig.1 Depicting concept of a queue

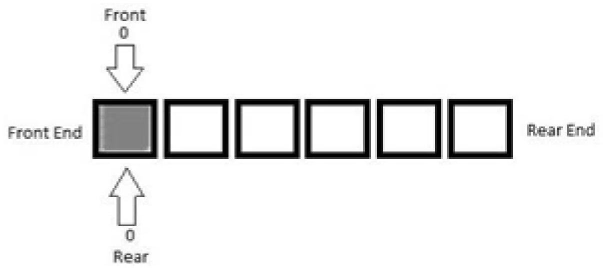
The moment we study the statically implemented queues we are taught that, it has a drawback that once it is full we can't insert elements until it becomes full empty due to repeatedly change in the position of front and rear. This drawback enforced the developers to implement a more better data structure which can remove this drawback and such that the full utilisation of the memory could take place. This lead to formation of a circular queue. In this the last end i.e. REAR is connected with the first end i.e. FRONT virtually by means of some conditional logics.

II. PRESENTLY USED QUEUES

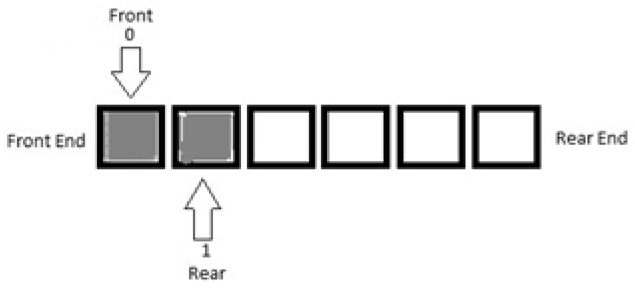
1. Initially the rear and the front are set to -1.Under this case queue is said to be empty or underflow.



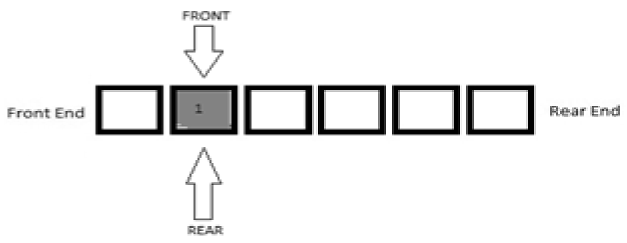
2. As the first element is entered the rear and front both shifts to 0th position.



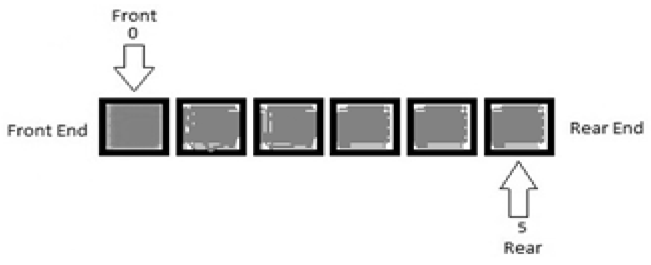
3. Then if we insert elements, the rear increase by 1.



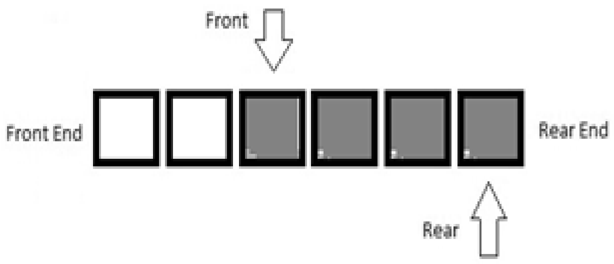
4. However when we delete elements, the front shifts by 1.



5. The queue is said to be overflow as the rear becomes MAXSIZE – 1. i.e.(consisder maxsize to be 6)

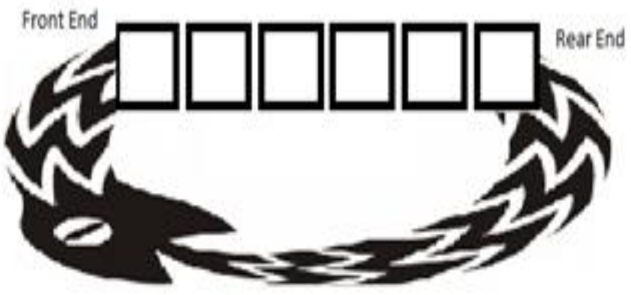


6. However the problem arises when we the following condition occurs.



7. We are having initial positions empty but it will show it as overflow as rear is still pointing at '-1' .

8. To overcome this the concept of circular queue was established, under which the tail of rear end is connected to front end.



9. Its algorithm lead to many complications in enhancing understandablity of this type of queue.

10. However both these type of queues doesn't hold true in practical life.

III. APPROACH USED IN THIS PAPER

In this paper we are trying to relate it with the real life queue examples quoted in above context.

In real life as a person goes out of queue, the rest of the elements shifts forward. If we use this concept, we can abolish the use of a front variable, hereby performing deletion from the fixed front most end only, i.e. 0th position in an array.

Fig2. Depicting a view at reservation counter.[3]

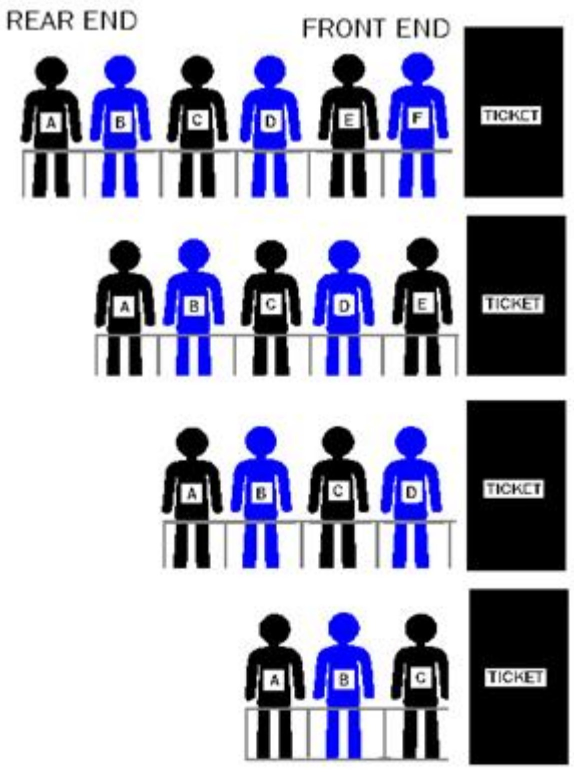


Fig.3 Depticting concept of queue deletion used in this paper

IV. ALGORITHM

```
1.  Initialisation of REAR:

    Set REAR=0

2.  Input the values to be inserted.
     i.e. ITEM

3.  Make insertion from the rear end.
    i.e. QUEUE [REAR]=ITEM

4.  While making insertion, make increment in rear i.e. REAR=REAR+1.

5.  While deletion, delete element at 0th position & then swap all elements one step forward. i.e.

    a.  Set  I = 1
    b.  Check I<=REAR
        If false , goto → g
    c.  QUEUE[I-1] = QUEUE[ I ]
    d.  I=I+1
    e.  Goto → b
    f.  REAR=REAR-1
    g.  END of  loop

6.  END
```

V. SOURCE CODE IN ‘C’ TO IMPLEMENT THIS QUEUE

```
#include<stdio.h>
#include<conio.h>
void ins_queue(int queue[], int size, int *rear);
void del_queue(int queue[], int *rear);
void trav_queue(int queue[], int rear);
void main()
```

```
{
 int queue[10],choice,size=10,rear=0;
 char ch;
 clrscr();
 do
 {
 printf("MENU\n\n1.INSERTION\n2. DELETION\n3. TRAVERSE\n\nEnter your choice: ");
 scanf("%d",&choice);
 if(choice==1)
 ins_queue(queue,size,&rear);
 else
 if(choice==2)
 del_queue(queue,&rear);
 else
 if(choice==3)
 trav_queue(queue,rear);
 else
 printf("wrong choice");
 printf("\n\nDo you wish to perform more operations ? (Y/N): ");
 scanf(" %c",&ch);
 clrscr();
 }
 while(ch=='y'||ch=='Y');
}

void ins_queue(int queue[], int size, int *rear)
{
 int num,i,c;
 c=*rear;
 if((size-c)==0)
 {
  printf("Overflow");
  goto a;
 }
 printf("\nYou can now enter maximum of %d elements:\n\n",size-c);
 printf("Enter the no. of element you wish to insert: ");
 scanf("%d",&num);
 printf("\n\n");
 if(num>(size-c))
 {
  printf("Invalid");
  goto a;
 }
 else
 {
  for(i=c;i<num+c;i++)
  {
   printf("Enter Element %d: ",*rear+1);
   scanf("%d",&queue[i]);
   *rear=*rear+1;
  }
  c=*rear;
  printf("\nInput Successfull\nNow the queue is:\n\n");
  for(i=0;i<c;i++)
  printf("  %d",queue[i]);
 }
 a:
 return;
}

void del_queue(int queue[], int *rear)
{
 int num,i,c,t,j;
 c=*rear;
 printf("\nCurrently Elements in Queue are: %d\n",c);
 printf("\nEnter the number of Elements you wish to Delete: ");
 scanf("%d",&num);
 if(num>c)
 {
  printf("Cant delete elements ");
  goto a;
 }
 else
 {
  for(i=0;i<num;i++)
  {
   for(j=1;j<c;j++)
   queue[j-1]=queue[j];
   *rear=*rear-1;
   c=*rear;
  }
 }
 printf("Deletion successfull\nnow the queue is:\n");
 for(i=0;i<c;i++)
 printf("  %d",queue[i]);
 a:
 return;
}

void trav_queue(int queue[], int rear)
{
 int i;
 if(rear==0)
 printf("Empty Queue");
 else
 printf("\nCurrent status of queue is:\n");
 for(i=0;i<rear;i++)
 printf("  %d",queue[i]);
```

CONCLUSIONS

If we include this approach in our statically implemented queues, in place of circular queue we can avoid multiple cases using if-else hereby making it less complex and easy to understand. Moreover, this concept can enhance the understanding of queue as it can be related to real life.

LIMITATIONS

Our approach include the following two limitations:

1. This approach can be used in case of statically implemented queue data structures
2. This may increase the time complexity in case of very large arrays.
3. Its time complexity is N while of presently used queue is 1

ACKNOWLEDGEMENT

With due regards, we would like to thank the actual people behind this paper. We would like to thank our head of department, Dr. Prof. R. Radhakrishnan, HOD-CSE and Mrs. Meetu Maheshwari, Asst. professor, AS & Hu Dept. of ABES Engineering College, Ghaziabad for their guidance at every step. We would also like to thank our friends, family members for their extreme support and blessings and to the ALMIGHTY for his blessings.

REFERENCES

[1]. http://en.wikipedia.org/wiki/Queue_(abstract_data_type)
[2]. https://www.google.co.in/search?q=queue+data+structure&biw=1366&bih=643&source=lnms&tbm=isch&sa=X&ei=yXNKVJCNBMnM8gWg5YLwDg&sqi=2&ved=0CAYQ_AUoAQ#imgdii=_

[3]. https://www.google.co.in/search?biw=1366&bih=643&noj=1&tbm=isch&sa=1&q=queue+at+railway+station&oq=queue+at+railway+station&gs_l=img.3...38896.45528.0.45732.29.22.0.0.0.4.464.2614.23j3j2.8.0....0...1c.1.56.img..28.1.464.r6folPcUH1Q

AUTHOR'S PROFILE:

Mr. PUNEET KUMAR GOYAL is a Senior Assistant Professor in Computer Science & Engineering Department at ABES Engineering College, Ghaziabad, Uttar Pradesh. He has completed his B.Tech in Computer Science & Engineering from BIT Durgapur. He has completed his M.Tech in Computer Science & Engineering form NRIIST Bhopal, Madhya Pradesh.

Mr. Rohan Gupta was born in Moradabad, a district in U.P-India on 2nd June 1995. He had received his high school and intermediate education from KCM School, Moradabad. At present, he is a II year student of Computer Science Engineering in ABES Engineering College, Ghaziabad (U.P. - India), presently affiliated to Mahamaya Technical University. His area of interests includes several languages such as C, C++, Java, SQL and fond of Data structures and Mathematics