International Journal of Engineering Research & Management Technology
ISSN: 2348-4039
Email: editor@ijermt.org September- 2014 Volume 1, Issue-5 www.ijermt.org

A Study on Queue & Its Implementation

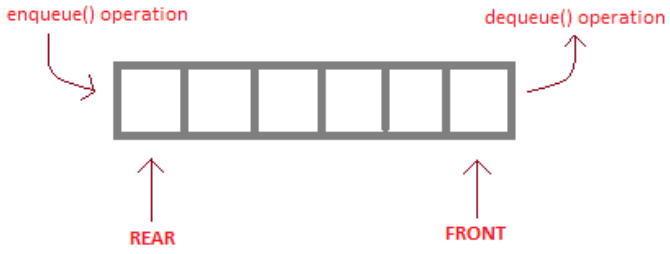
Sagar Rao
Department of Information Technology
Dronacharya College of Engineering,
Gurgaon

Mukul Sohlot
Department of Information Technology
Dronacharya College of Engineering
Gurgaon

ABSTRACT:
This paper is general view about Queue. This paper is divided into four parts. In first part is introduction of queue. In second parts is about Circular buffer, Deque, Priority queue and there description . The queue is a First-In-First-Out (FIFO) data structure .In four parts discuss about implementation of queue.
Keywords: queue, Circular buffer, Deque, Priority queue, FIFO, implementation.

1. INTRODUCTION

Queue: [1] A queue is an ordered list of items, differing from a stack in that the first item added to the queue is the first item to be removed. A queue is also known as a "First in First Out" (FIFO) list. A queue can also be implemented in an array, but managing the pointers to the beginning and end of the array is somewhat more complicated – as the queue runs into the end of the physical array it must wrap around to the beginning of the array to use any space freed by removing items from the queue.



enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

2. Types of queues

2.1 Circular buffer:[2]**It** is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure lends itself easily to buffering data streams. The useful property of a circular buffer is that it does not need to have its elements shuffled around when one is consumed. Circular buffering makes a good implementation strategy for a queue that has fixed maximum size. Should a maximum size be adopted for a queue, then a circular buffer is a completely ideal implementation; all queue operations are constant time. However, expanding a circular buffer requires shifting memory, which is comparatively costly.

2.2 Deque:[3] Double ended Queue: double-ended queue (often abbreviated to deque, pronounced *deck*) is an abstract data structure that implements a queue for which elements can only be added to or removed from the front (head) or back (tail). It is also often called a head-tail linked list. There are at least two common ways to efficiently implement a deque: with a modified dynamic array or with a doubly-linked list. Dynamic array implementation uses a variant of a dynamic array that can grow from both ends, sometimes called array deques. These array deques have all the properties of a dynamic array, such as constant time random access, good locality of reference, and inefficient insertion/removal in the middle, with the addition of amortized constant time insertion/removal at both ends, instead of just one end.

2.3 Priority queue: [4] it is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.Priority queues are a generalization of stacks and queues. Rather than inserting and deleting elements in a fixed order, each element is assigned a priority represented by an integer. We always remove an element with the highest priority, which is given by the minimal integer priority assigned.Priority queues often have a fixed size.

3. Implementation

3.1 Queue implementation

```
#include<stdio.h>
#define MAX 50
int queue_arr[MAX];
int rear = - 1;
int <span id="IL_AD6" class="IL_AD">front</span> = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert\n");
      printf("2.<span id="IL_AD8" class="IL_AD">Delete</span>\n");
      printf("3.Display\n");
      printf("4.Quit\n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         del();
         break;
         case 3:
         display();
         break;
```

```
            case 4:
            exit(1);
            default:
            printf("Wrong choice\n");
        } /*End of switch*/
    } /*End of while*/
} /*End of main()*/
insert()
{
    int added_item;
    if (rear == MAX - 1)
    printf("Queue <span id="IL_AD5" class="IL_AD">Overflow</span>\n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Input the element for adding in queue : ");
        scanf("%d", &added_item);
        rear = rear + 1;
        queue_arr[rear] = added_item;
    }
} /*End of insert()*/
del()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow\n");
        <span id="IL_AD7" class="IL_AD">return</span> ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_arr[front]);
        front = front + 1;
    }
} /*End of del() */
display()
{
    int i;
    if (front == - 1)
    printf("Queue is empty\n");
    else
    {
        printf("Queue is :\n");
        for (i = front; i <= rear; i++)
        printf("%d ", queue_arr[i]);
        printf("\n");
```

```
   }
} /*End of display() */
```

3.2 circular buffer implementation

```
#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#define MAXSIZE 5
int cq[MAXSIZE];
int front,rear;
void main()
{
        void add(int,int);
        void del(int);
        int will=1,i,num;
        front = -1;
        rear = -1;
        clrscr();
        printf("\nProgram for Circular Queue demonstration through array");
        while(1)
        {
                printf("\n\nMAIN MENU\n1.INSERTION\n2.DELETION\n3.EXIT");
                printf("\n\nENTER YOUR CHOICE : ");
                scanf("%d",&will);
                switch(will)
                {
                case 1:
                        printf("\n\nENTER THE QUEUE ELEMENT : ");
                        scanf("%d",&num);
                        add(num,MAXSIZE);
                        break;
                case 2:
                        del(MAXSIZE);
                        break;
                case 3:
                   exit(0);
                        default: printf("\n\nInvalid Choice . ");
                }

} //end of  outer while
}               //end of main
void add(int item,int MAX)
{
        //rear++;
        //rear= (rear%MAX);
        if(front ==(rear+1)%MAX)
        {
        printf("\n\nCIRCULAR QUEUE IS OVERFLOW");
```

```
        }
        else
        {
          if(front==-1)
            front=rear=0;
            else
            rear=(rear+1)%MAX;
            cq[rear]=item;
            printf("\n\nRear = %d    Front = %d ",rear,front);
        }
}
void del(int MAX)
{
int a;
if(front == -1)
        {
        printf("\n\nCIRCULAR QUEUE IS UNDERFLOW");
        }
        else
        {
                a=cq[front];
                if(front==rear)
                 front=rear=-1;
                else
                  front = (front+1)%MAX;
           printf("\n\nDELETED ELEMENT FROM QUEUE IS : %d ",a);
                printf("\n\nRear = %d    Front = %d ",rear,front);

        }
}
```

3.3 Deque Implementation

```
#include<stdio.h>
#include<conio.h>
int q[5],front=-1,rear=-1,max=5;
void print()
{
int i;
if(front==-1)
printf("QUEUE EMPTY");
else
{
if(front<=rear)
for(i=front;i<=rear;i++)
printf(" %d ",q[i]);
else
{
```

```
for(i=0;i<=rear;i++)
printf(" %d ",q[i]);
for(i=front;i<max;i++)
printf(" %d ",q[i]);

}
}
}

void insertrear()
{

if((front==rear+1)((front==0)&&(rear==max-1)))
printf("QUEUE IS FULL");
else
{
if(rear==max-1)
rear=0;
else
rear++;
if(front==-1)
front++;

printf("ENTER THE ELEMENT TO BE INSERTED :");
scanf("%d",&q[rear]);
printf("\ QUEUE AFTER INSERTION :");
print();
}
}
void delbeg()
{
if(front==-1)
printf("QUEUE EMPTY");
else
{
printf("ELEMENT DELETED : %d",q[front]);
if(front==rear)
{
front=-1;
rear=-1;
}
else
if(front==max-1)
front=0;
else
front++;
```

```
printf("\ QUEUE AFTER DELETION :");
print();
}
}
void insertbeg()
{

if((front==rear+1)((front==0)&&(rear==max-1)))
printf("QUEUE IS FULL");
else
{
if(front==-1)
{
front++;
rear++;
}
else
if(front==0)
front=max-1;
else
front--;
printf("ENTER THE ELEMENT TO BE INSERTED :");
scanf("%d",&q[front]);
printf("\ QUEUE AFTER INSERTION :");
print();
}
}
void delrear()
{
if(front==-1)
printf("QUEUE EMPTY");
else
{
printf("ELEMENT DELETED : %d",q[rear]);
if(front==rear)
{
front=-1;
rear=-1;
}
else
if(rear==0)
rear=max-1;
else
rear--;
printf("\ QUEUE AFTER DELETION :");
print();
}
```

```
}

void main()
{
int ch;
clrscr();
do
{

printf("\n1) INSERTION AT FRONT \n2)INSERTION AT REAR\n3.DELETEION AT
FRONT\n4.DELETION AT REAR\n ");
printf("\n ENTER CHOICE ");
scanf("%d",&ch);
switch(ch)
{
case 1:insertbeg();
break;
case 2:insertrear();
break;
case 3:delbeg();
break;
case 4:delrear();
break;
default :printf("WRONG CHOICE");
break;
}

}while(ch>=1&&ch<=4);
getch();
}
```

3.4 Priority queue Implementation

```
#define SIZE 5          /* Size of Queue */
int PQ[SIZE],f=0,r=-1;      /* Global declarations */

PQinsert(int elem)
{
   int i;      /* Function for Insert operation */
   if( Qfull()) printf("\n\n Overflow!!!!\n\n");
   else
   {
      i=r;
      ++r;
      while(PQ[i] <= elem && i >= 0) /* Find location for new elem */
      {
         PQ[i+1]=PQ[i];
         i--;
```

```
        }
        PQ[i+1]=elem;
    }
}

int PQdelete()
{                   /* Function for Delete operation */
    int elem;
    if(Qempty()){ printf("\n\nUnderflow!!!!\n\n");
    return(-1); }
    else
    {
        elem=PQ[f];
        f=f+1;
        return(elem);
    }
}

int Qfull()
{                   /* Function to Check Queue Full */
    if(r==SIZE-1) return 1;
    return 0;
}

int Qempty()
{                   /* Function to Check Queue Empty */
    if(f > r) return 1;
    return 0;
}

display()
{                   /* Function to display status of Queue */
    int i;
    if(Qempty()) printf(" \n Empty Queue\n");
    else
    {
        printf("Front->");
        for(i=f;i<=r;i++)
            printf("%d ",PQ[i]);
        printf("<-rear font="font">
    }
}

main()
{                       /* Main Program */
    int opn,elem;
    do
```

```
{
    clrscr();
    printf("\n ### Priority Queue Operations(DSC order) ### \n\n");
    printf("\n Press 1-Insert, 2-Delete,3-Display,4-Exit\n");
    printf("\n Your option ? ");
    scanf("%d",&opn);
    switch(opn)
    {
    case 1: printf("\n\nRead the element to be Inserted ?");
       scanf("%d",&elem);
       PQinsert(elem); break;
    case 2: elem=PQdelete();
       if( elem != -1)
          printf("\n\nDeleted Element is %d \n",elem);
       break;
    case 3: printf("\n\nStatus of Queue\n\n");
       display(); break;
    case 4: printf("\n\n Terminating \n\n"); break;
    default: printf("\n\nInvalid Option !!! Try Again !! \n\n");
       break;
    }
    printf("\n\n\n\n  Press a Key to Continue . . . ");
    getch();
  }while(opn != 4);
}
```

4. APPLICATION

Typical uses of queues are in simulations and operating systems.

- Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
- Computer systems must often provide a “holding area” for messages between two processes, two programs, or even two systems. This holding area is usually called a “buffer” and is often implemented as a queue.

Our software queues have counterparts in real world queues. We wait in queues to buy pizza, to enter movie theaters, to drive on a turnpike, and to ride on a roller coaster. Another important application of the queue data structure is to help us simulate and analyze such real world queues.

5. REFERENCES

1. Donald Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 2.2.1: Stacks, Queues, and Deques, pp. 238–243.
2. Simon Cooke. "The Bip Buffer - The Circular Buffer with a Twist". 2003.
3. http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf C. Okasaki, "Purely Functional Data Structures", September 1996
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 20: Fibonacci Heaps, pp.476–497. Third edition p518.