K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A2 Roll No.: 16010122041
Experiment / assignment / tutorial No.
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

Title: Implementation of Basic operations on queue for the assigned application using Array and Linked List- Create, Insert, Delete, Destroy

Objective: To implement Basic Operations on Queue i.e. Create, Push, Pop, Destroy for the given application

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4.

Abstract:

- A queue in the data structure can be considered similar to the queue in the real world.
- Queues have a FIFO (first-in-first-out) structure, where deletion and insertion happen at opposite ends.
- Queues have 2 pointers looking to the front and back of the queue.
- Queues use enqueue () and dequeue () functions: adding to the end of the queue and removing from the front of the queue.

There are two fundamental operations performed on a Queue - enqueue and dequeue:

- **Enqueue:** The enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue:** The dequeue operation performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front- end. It returns an integer value. The dequeue operation can also be designed to void.
- **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
- **Queue overflow (isfull):** When the Queue is completely full, then it shows the overflow condition.
- **Queue underflow (isempty):** When the Queue is empty, i.e., no elements are in the Queue then it throws the underflow condition.

List 5 Real Life applications of Queue:

- A queue of people at ticket-window: The person who comes first gets the ticket first. The person who is coming last is getting the tickets in last. Therefore, it follows first-in-first-out (FIFO) strategy of queue.
- Vehicles on toll-tax bridge: The vehicle that comes first to the toll tax booth leaves the booth first. The vehicle that comes last leaves last. Therefore, it follows first-in-first-out (FIFO) strategy of queue.
- Phone answering system: The person who calls first gets a response first from the phone answering system. The person who calls last gets the response last. Therefore, it follows first-in-first-out (FIFO) strategy of queue.
- Luggage checking machine: Luggage checking machine checks the luggage first that comes first. Therefore, it follows FIFO principle of queue.
- Patients waiting outside the doctor's clinic: The patient who comes first visits the doctor first, and the patient who comes last visits the doctor last. Therefore, it follows the first-in-first-out (FIFO) strategy of queue.

Define and explain various types of queue with suitable diagram and their application(s):

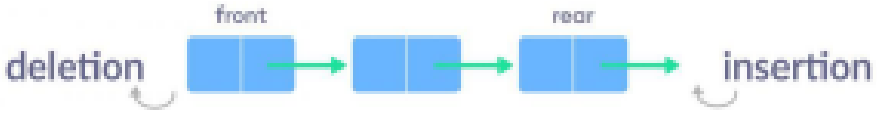
Types of Queues

There are four different types of queues:

- Simple Queue
- Circular Queue
- Priority Queue
- Double Ended Queue

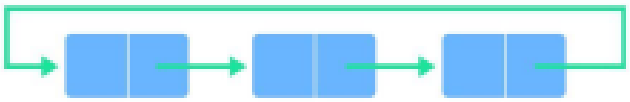
1. **Simple Queue**

   In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



2. **Circular Queue**

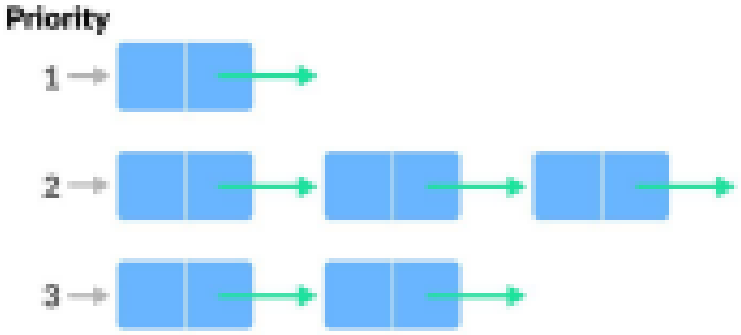
   In a circular queue, the last element points to the first element making a circular link.



   The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the first position. This action is not possible in a simple queue.

3. **Priority Queue**

   A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.



   Insertion occurs based on the arrival of the values and removal occurs based on priority.

4. **Deque (Double Ended Queue)**

   In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.

Queue ADT:

/* value definition */
Abstract typedef <front, size> Queue
Condition: size!= 0 && front = 0 && rear = -1;

/* operator definition */
Abstract Queue isempty()
Post Condition: true if rear < front;
else return false;

Abstract Queue peek()
Pre-Condition: isempty() == false;
Post Condition: return front element;

Abstract Queue enqueue (int x)
Pre-Condition: size!= N
Post Condition: rear = rear + 1;
Queue[rear] = x;

Abstract Queue dequeue(int x)
Pre-Condition: isempty() == false;
Post Condition: y = Queue[front];
rear = rear – 1;
front = front + 1;
return y;

Algorithm for Queue operations using array/Linked list: (Write only the algorithm for assigned type)

enqueue()

a. Start
b. Create new node newNode
c. Set newNode->data = added_item
d. Set newNode->next = NULL
e. Check if head has lower priority. If true follow go to step 6. Else go to Step 8.
f. Set newNode->next = front
g. Set front = newNode. Go to step 10.
h. Set temp = front

i. While temp->next != NULL and temp->next->priority -> priority Set temp = temp->next.
j. Set newNode->next = temp->next
k. Set temp->next = newNode
l. End

dequeue()

a. Start
b. If front == null, then print "underflow" and go to step 6
c. Set Temp = front
d. Set front = front->next
e. Free temp
f. End

peek()

a. Start
b. Return front->data
c. End

Algorithm for Queue operations using array/Linked list : (Write only the algorithm for assigned type)

Arrays:

```
#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insert();
void delete();
void display();
int front = -1, rear = -1;
int queue[maxsize];
void main ()
{
    int choice;
    while(choice != 4)
    {
        printf("\n1.Insert an element\n2.Delete an
element\n3.Display the queue\n4.Exit\n");
        printf("\nEnter your choice ?");
        scanf("%d",&choice);
        switch(choice)
        {
```

```
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nEnter valid choice(1-4)!\n");
        }
    }
}
void insert()
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nInserted!!\n");
```

```
}
void delete()
{
    int item;
    if (front == -1 || front > rear)
    {
        printf("\nUNDERFLOW\n");
        return;

    }
    else
    {
        item = queue[front];
        if(front == rear)
        {
            front = -1;
            rear = -1 ;
        }
        else
        {
            front = front + 1;
        }
        printf("\nDeleted element is  %d\n", item);
    }


}

void display()
{
    if(rear == -1)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nQueue is:\n");
        for(int i=front;i<=rear;i++)
        {
            printf("\n%d\n",queue[i]);
```

```
        }
    }
}
```

Output Screenshots:

```
1.Insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter the element
23
Inserted!!
Inserted!!
Inserted!!

1.Insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter the element
2

Inserted!!

1.Insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

Queue is:

23

2

1.Insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?2

Deleted element is  23

1.Insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?4
```

Applications of Queue in computer science:

1) When a resource is shared among multiple consumers. Nowadays computer handles multiuser, multiprogramming environment and time-sharing environment. In this environment a system(computer) handles several jobs at a time, to handle these jobs the concept of a queue is used.
2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
3) In Operating systems:
   a) FCFS (first come first serve) scheduling, example: FIFO queue
   b) Spooling in printers
4) In Networks:
   a) Queues in routers/ switches
   b) Mail Queues

Conclusion: -

Through this experiment we implemented priority queues and applied the concept for given application. We understood the algorithm and concept and learnt about different queues.