Queues.

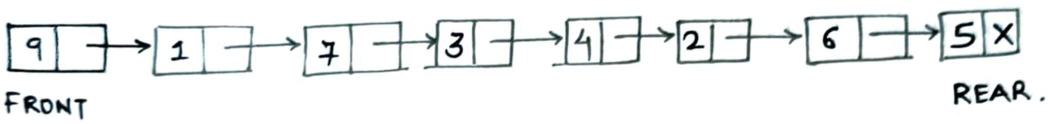
In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element.

FRONT = REAR = NULL → Indicate Queue is empty.

Operation on linked Queues.

- All deletion is done at the front end.
- All Insertion is done at the Rear end.
- A Queue has 2 basic Operation
  - → Insertion → add element @ end or REAR
  - → Deletion → remove all elements from the front or start of the queue.

Insert Operation



ALGORITHM TO INSERT AN ELEMENT IN A LINKED QUEUE

Step 1: Allocate Memory for the New node and name it as PTR.

Step 2: SET PTR→ DATA = VAL

Step 3: IF FRONT = NULL

SET FRONT = REAR = PTR

SET FRONT → NEXT = REAR→ NEXT = NULL

ELSE

SET REAR → NEXT = PTR

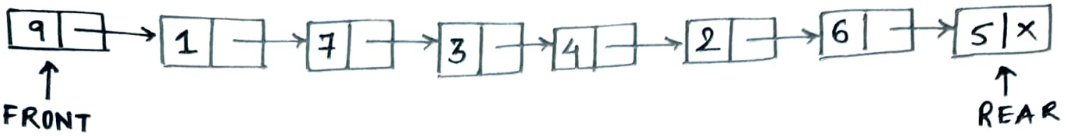
SET REAR = PTR

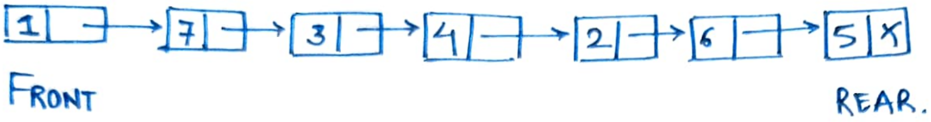
SET REAR → NEXT = NULL

[END OF IF]

Step 4: END

ALGORITHM FOR DELETING FROM A QUEUE.





ALGORITHM.

Step 1 IF FRONT = NULL
Write " UNDERFLOW"
GOTO STEP 5
[END OF IF]

Step 2 SET PTR = FRONT

Step 3 SET FRONT = FRONT -> NEXT

Step 4 FREE PTR

Step 5 END.

Queues Implementation with the help of array.

A queue is a FIFO (First In, First Out) data structure in which the element that is inserted first one to be taken out.
The element in a queue are added at one end called the REAR and removed from the other end called the FRONT.
Queues can be implemented by using either arrays or linked lists.

Array Representation of Queues.

→ Operation on Queues.

FRONT = 0 and REAR = 5.

12	9	7	18	14	36	
0	1	2	3	4	5	

Add → 45 @ REAR

12	9	7	18	14	36	45			
0	1	2	3	4	5	6			

Queue after insertion of New element

	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Queue after deletion of an element.
here, FRONT = 1 and REAR = 6.

ALGO TO INSERT AN ELEMENT IN A QUEUE.

Step 1 : IF REAR = MAX-1
WRITE "OVERFLOW"
GOTO STEP 4

Step 2: IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0

ELSE
SET REAR = REAR + 1

[END OF IF]

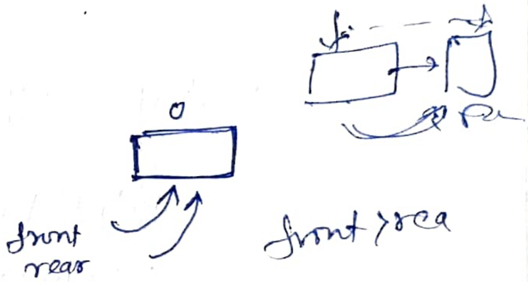
Step 3: SET QUEUE [REAR] = NUM

Step 4: EXIT.

```
Step1: IF FRONT = -1 OR FRONT > REAR
          Write UNDERFLOW
       ELSE
          SET VAL = QUEUE[FRONT]
          SET FRONT = FRONT + 1
       [END OF IF]
Step2: EXIT
```

```c
#include <stdio.h>
#include <conio.h>
#define MAX 10;
int queue[MAX];
int front = -1, rear = -1;
void insert (void);
int delete_element (void);
int peek (void);
void display (void);
int main ()
{
    int option, val;
    do
    {
        printf ("\n\n *** MAIN MENU ***");
        printf ("\n 1. Insert an element");
        printf ("\n 2. Delete an element");
        printf ("\n 3. Peek");
        printf ("\n 4. Display the queue");
        printf ("\n 5. Exit");
        printf (" Enter your option"); scanf ("%d", &option);
        switch (option)
        {
```

```
case 1:
      insert();
      break;

case 2:
    val= delete-element ();
    if(val!=-1)
    printf("\n The number is deleted is : %d", val);

    break;

case 3:
    val= peek();
    if(val!=-1)
    printf("\n The first value in queue is : %d",val);
    break;

case 4:
      display();
      break;

  }

}
while (option!=5)
getch();
return 0;

}

void insert()
{
 int num;
 printf("\n Enter the number to be inserted in the queue :");
 scanf ("%d", &num);
 if (rear== MAX-1)
  printf ("\n OVERFLOW");
  else if (front==-1 && rear==-1)
  front = rear= 0;
  else
   rear++;
   queue[rear] =num;
}
```



a[rear]

a[i]

a[i+1]=num

```
int delete_element()
{
    int val;
    if (front==-1 || front>rear)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    else
    {
        val= queue[front];
        front++;
        if (front> rear)
            front= rear=-1;
        return val;
    }
}

int peek()
{
    if(front==-1 || front>rear)
    {
        printf("\n QUEUE IS EMPTY");
        return -1;
    }
    else
    {
        return queue[front];
    }
}

void display()
{ int i;
  printf("\n");
  if (front==-1 || front>rear)
    printf("\n QUEUE IS EMPTY");
  else
  { for(i=front; i<=rear; i++) → printf("\t %d", queue[i]); }
```

TIME AND SPACE COMPLEXITIES OF QUEUE OPERATION.

1). **enqueue ( ):**

This operation inserts an element at the back of the queue. It takes one parameter, the value that is to be inserted at the back of the queue.

Complexity analysis

- Time Complexity: O(1), In enqueue function a single element is inserted at the last position. This takes a single memory allocation operation which is done in constant time.
- Auxillary Space: O(1) – As no extra space is being used.

2). **dequeue ( ):**

This operation remove an element present at the front of the queue. Also, it results in an error if the queue is empty.

Complexity Analysis

→ Time Complexity :→ O(1). In array implementation, only an arithmetic operation is performed i.e the front pointer is incremented by 1. This is a constant time function.

→ Auxillary Space → O(1) → As No extra space is being used.

3. peek()

This operation prints the element present at the front of the queue.

Time Complexity →

(1) O(1) → In this operation, only a memory addressed is accessed. This is a constant-time operation.

Auxillary Space → O(1) → No extra space is utilized to access the first value.

4. isfull()

Function that returns true if the queue is filled completely else returns false.

Complexity

① Time Complexity → O(1) → it only performs an arithmetic operation to check if the queue is full or not.

② Auxillary Space → O(1) → it requires no extra space.

5. isempty()

Function that returns true if the queue is empty else returns false.

Complexity

① Time complexity → O(1) → It only checks the position stored in the first and last pointers.

② Auxillary Space → O(1) → No extra space is required to check the value of the first and last pointer.

Another Approach

```
#include <stdio.h>
#include <conio.h>

struct queue
 {
   int no;
   struct queue *next;
 };

struct queue *start = NULL;

void add();
int del();
void traverse();

void main()
 {
   int ch;
   char choice;
   do
   {
     clrscr();
     printf("---1. add\n");
     printf("----2. delete\n");
     printf("---3. traverse\n");
     printf("---4. exit\n");
     printf(" Enter your choice \n");
     scanf("%d", &ch);
     switch(ch);
     {
        case 1: add();
                break;
        case 2: printf("the delete element is \n %d", del());
                break;
```

```
        case 3: traverse();
                break;
       case 4: return
       default: printf("wrong choice");
       }
        scanf ("%c", &choice);
    }
    while (choice != 4);
}

void add()
{
  struct queue *p, *temp;
    temp = start;
    p = (struct queue *) malloc(sizeof(struct queue));
    printf ("Enter the data");
    scanf ("%d", &p->no);
    p->next = NULL;
    if (start == NULL)
       {
        start = p;
       }
     else
     {
       while (temp->next != NULL)
          {
           temp = temp->next;
          }
           temp->next = p;
          }
      }
```

```
int del()
{
 struct queue * temp;
 int value;
 if (start == NULL)
   {
    printf("Queue is empty");
     getch();
     return(0);
   }
  else
   {
     temp = start;
     value = temp->no;
     start = start->next;
      free(temp);
   }
   ~~return(val)~~
    return(value);
   }

void traverse()
 {
  struct queue * temp;
   temp = start;
   while (temp->next != NULL)
     { ~~printf(~~
     printf("No = %d", temp->no);
        temp = temp->next;
     ~~getch();~~ }
        printf("no = %d", temp->no);
          getch();
       }
```

75K