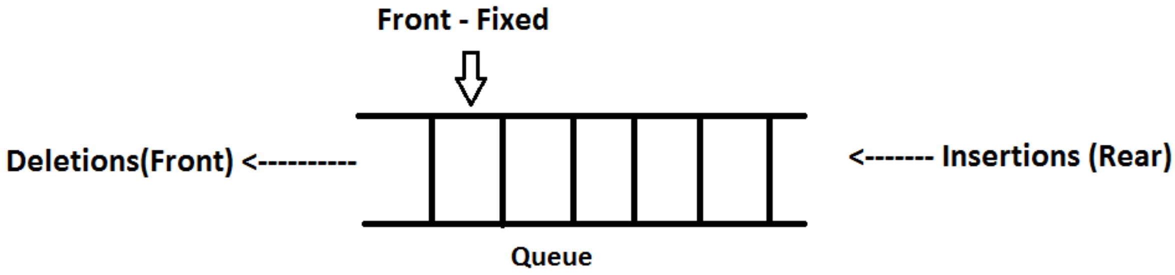
Queue

- Queue is a linear data structure.
- It follows a rule called FIFO(First In First Out).
- Insertions from one end called Read and Deletions from another end called Front.
- ‘Front’ is fixed in Queue.



Operations:

- Insert
- Delete
- isEmpty
- isFull
- traverse

Note:

- We can implement queue in 2 ways
    - Static – fixed size
    - Dynamic – size varies

Static Queue:

- Using arrays, we implement static queue.
- We declare variables globally to process elements of queue.

Global variables:

```
#define  SIZE 5
int queue[SIZE];
int front=0 , rear=0;
```

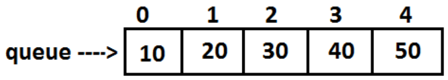
Insert element:

```
void insert(int ele)
{
      if(isFull( ))
      {
            printf("Queue is
                        full \n");
      }
      else
      {
           queue[rear] = ele ;
           ++rear ;
           printf("Inserted....\n");

      }
}
```

```
int  isFull( )
{
      if(rear == SIZE)
            return 1 ;
      else
            return 0;
}
```

SIZE = 5



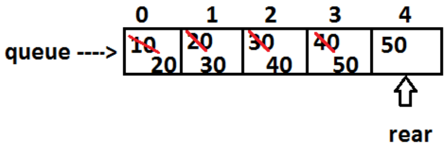
front = 0 , rear = ~~0~~
~~1~~
~~2~~
~~3~~
~~4~~
5

Delete element:

```
int  delete( )
{
      if(isEmpty( )){
            printf("Queue is  empty \n");
      }
      else
      {
            int ele , i ;
            ele = queue[front];
            for(i=0 ; i<rear-1 ; i++)
            {
                  queue[i] = queue[i+1];
            }
            --rear ;
            return ele ;
      }
}
```

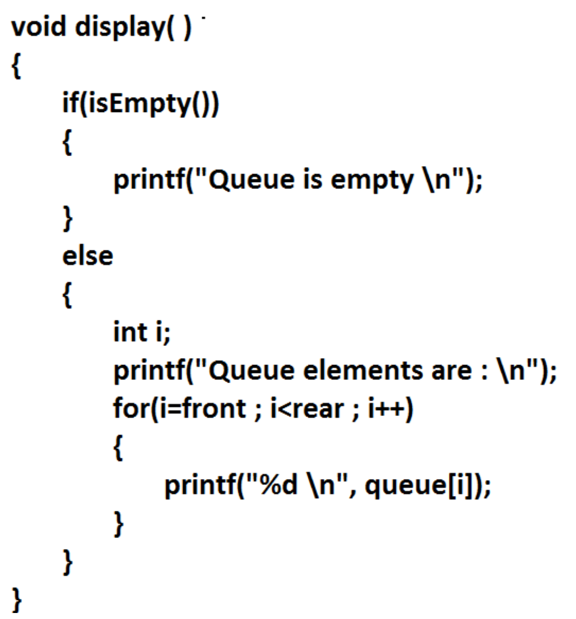
```
int isEmpty()
{
      if(front==rear)
            return 1;
      else
            return 0;
}
```

SIZE = 5

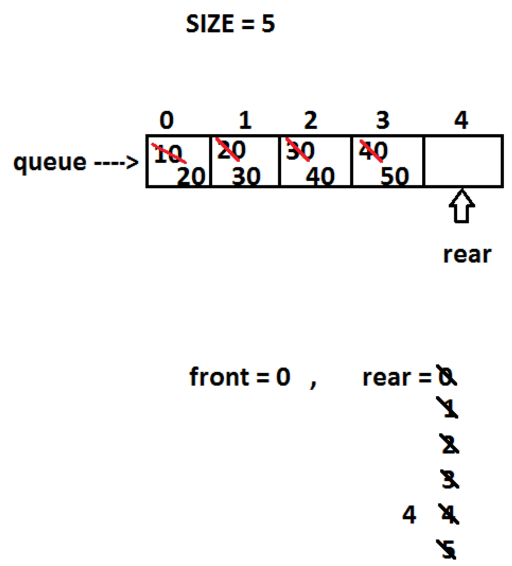


front = 0 , rear = ~~0~~
~~1~~
~~2~~
~~3~~
4 ~~4~~
~~5~~

Display:



```
void display( )
{
    if(isEmpty())
    {
        printf("Queue is empty \n");
    }
    else
    {
        int i;
        printf("Queue elements are : \n");
        for(i=front ; i<rear ; i++)
        {
            printf("%d \n", queue[i]);
        }
    }
}
```



Write the complete program with all queue operations:

Queue implementation using single linked list:

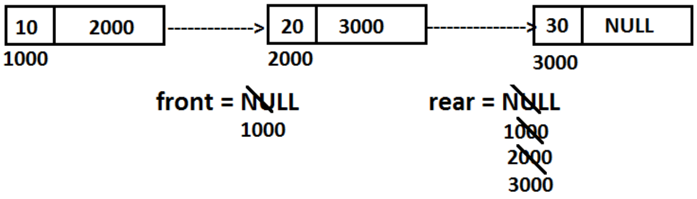
- We can implement Queue rules using single linked list.
- We store elements using Node structure.
- Insertions from rear(using Node type rear pointer).
- Deletions from front(using Node type front pointer);
- It is dynamic queue.

Node structure:

```
struct Node
{
        int data;
        struct Node *link;
};

struct Node *front=NULL;
struct Node *rear=NULL;
```

Insertion:

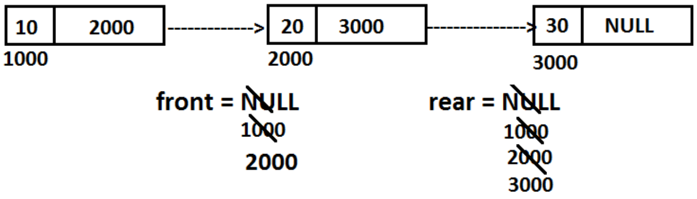


```
void insert(int ele)
{
      struct Node *temp ;
      temp = malloc( );

      temp->data = ele;
      temp->link = NULL;
```

```
      if(front==NULL)
      {
            front = rear = temp;
      }
      else
      {
            rear->link = temp ;
            rear = temp ;
      }
}
```

Deletion:



```
void delete()
{
      if(front == NULL)
      {
            printf("Queue is
                        empty \n");
      }
```

```
      else
      {
            struct Node *temp = front;
            printf("Deleted : %d \n", front->data);
            front = front->link;
            temp->link = NULL;
            free(temp);
      }
}
```