Queue (COLA / FILA)

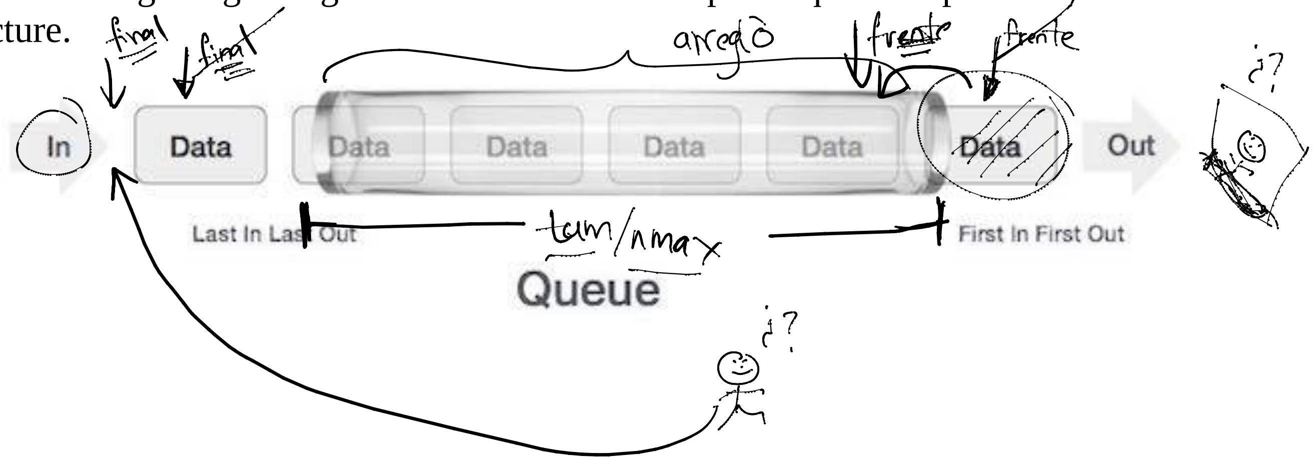
Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enQueue) and the other is used to remove data (deQueue). Queue follows First-In-First-Out methodology (FIFO), i.e., the data item stored first will be accessed first.

LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

QUEUE REPRESENTATION

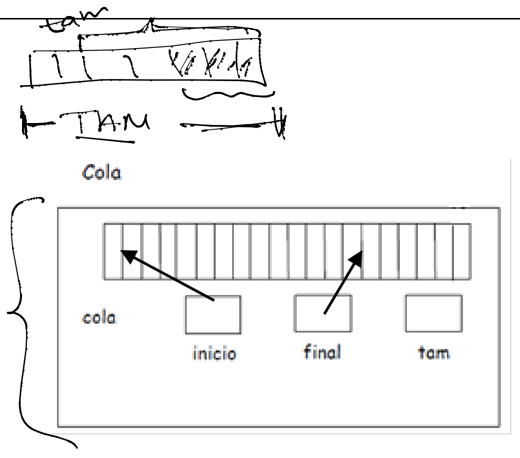
The following diagram given below tries to explain queue representation as data structure.



As in stacks, a queue can also be implemented using Arrays one-dimensional.

Structures:

```
#define TAM 100 ✓
typedef struct ✓
{ ✓
  "TipoDato" cola[TAM]; //Arreglo
   int frente, final; //Indices 70 ✓
  int tam; ✓  //50  95 ✓
                    15 ✓
}  COLA;        ~~105~~
```



BASIC OPERATIONS

Here we shall try to understand the basic operations associated with queues:

- **iniQueue**() – set the empty state to the queue (inicializar fila/cola)
- **enQueue()** − add (store) an item to the queue (encolar/poner/formar)
- **deQueue()** − remove (access) an item from the queue (desencolar/quitar/atender)

Aditional functions:

- **isFull()** − Checks if the queue is full.
- **isEmpty()** − Checks if the queue is empty.
- **peek()** − Gets the element at the ***front*** of the queue without removing it.

Implementation

iniQueue Operation => empty state

```
int initQueue(COLA *c, int nm)      50 0
{                                   10
        int res = 0;                75 ✓
                                    120
        if(nm <= TAM)   ¿?
        {
                c->tam = nm;
                c->frente = c->final = -1;
                res = 1;
        }
        return(res);
}
```

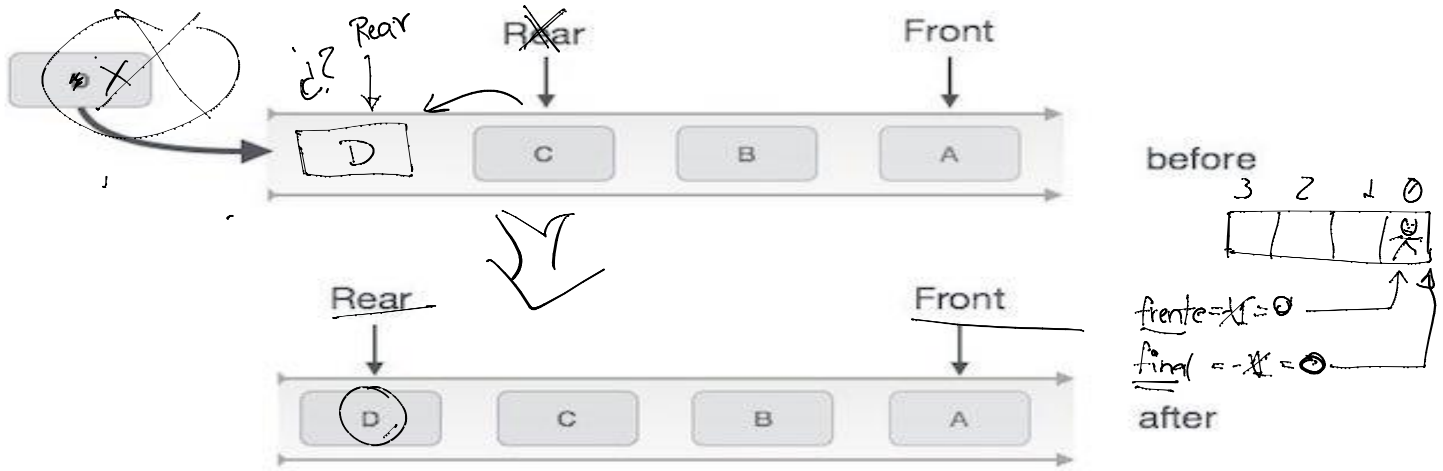
Sometimes, we also check to see if a queue is initialized or not, to handle any unforeseen situations.

ENQUEUE OPERATION

Queues maintain two data indexs, **front** and **ends**. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue:

- **1** − Check if the queue is not full.
- **2** - Increment **ends** index to point the next empty space
- **3-** Add data element to the queue location, where the **ends** is pointing.
- **4-** Produces: success code(1)
- **5** − Otherwise
- **6** − produces: error code(0)
- **7** − return code



Queue Enqueue

```
int enQueue(COLA *c, int dato)   ≈ push
{
 int res=0;
 if(c->final + 1 < c->tam)   // NO  Está llena
     {
            c->final++; ✓ (-1+1 = 0)
            c->cola[c->final] = dato; ✓// Colocar el dato (final)
            if(c->frente < 0)
               c->frente = 0;   } ✓
            res = 1; ✓
     }
 return(res); ←
}
```

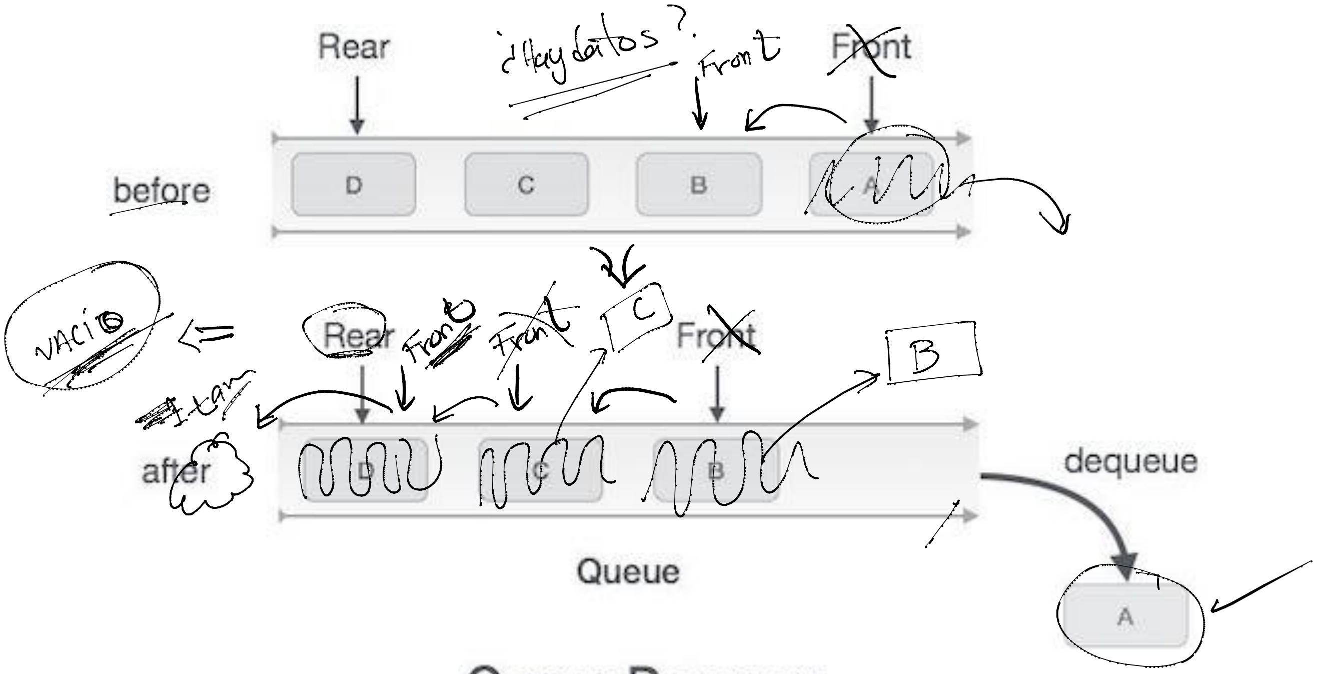
φ

DEQUEUE OPERATION

Accessing data from the queue is a process of two tasks − access the data where **front** is pointing and "remove" the data after access.

The following steps are taken to perform **dequeue** operation:

- **1** − Check if the queue is not empty.
- **2-** Access the data where **front** is pointing
- **3** - Increment **front** index to point to the next available pos
- 4- Produce success code(1)
- 5- Otherwise
- **6** − Produce error code(0)
- **7** −return code



Queue Dequeue

```
int deQueue(COLA *c, int *dato)
{
  int res = 0;
  if(c->frente > -1)   // No Vacía ?
     {
        *dato = c->cola[c->frente];
        if(c->frente == c->final)     // ¿Un solo dato?
           c->frente = c->final = -1;  //Retornar al edo. de vacío
        else
           c->frente++;
        res = 1;
     }
  return(res);
}
```

①

* Función para capturar los datos de una fila

```
void capturaQueue(COLA *c)
{
   int res, dato;
   char resp;

   do {
            printf("Dame el dato: ");
            scanf("%d", &dato);
            res = enQueue(c, dato);
            if(res == 1)
            {
              printf("Dato formado, otro (s/n)?");
              scanf("%c", &resp);
            }
            else
                  printf("Dato no ingresado\n");
      } while(res == 1 && tolower(resp)=='s');
}
```

* Función para mostrar los datos de la fila

```
void muestraFila(FILA *f)
{
  int valor;

  while( deQueue(f, &valor) == 1)
      printf("%d   ", valor);
}
```

* Función para sumar los datos de una Fila/Cola/Queue

```
int sumaFila(FILA *f)
{
  int valor, suma=0;

  while( deQueue(f, &valor) == 1)
      suma += valor;

  return(suma);
}
```

* Función para contar No. pares e impares que hay en una fila/Queue.

```
void  cuentaParImp(FILA *f, int *cPar, int *cImp)
{
  int valor;

  *cPar = *cImp = 0;
  while( deQueue(f, &valor) == 1)
    if(valor%2 == 0)     // ¿Es par?
       (*cPar)++;           // *cPar +=1;
    else
       (*cImp)++;
}
```

* Simular una fila/cola/queue de tal manera que con ciertos valores al ***azar*** se agregue o quite elementos a la misma. (0→poner, 1→ quitar, 2→ salir).

```
void simulaQueue(FILA *f)
{
  int valor, res, op;

  srand(time(NULL));
  do {
        op=rand()%3;  // 0, 1 o 2
        switch(op)
        {
          case 0: printf("Dame el valor: ");
                       scanf("%d", &valor);
                       res = enQueue(f, valor);
                       if(res == 1)
                          printf("Cliente formado\n");
                       else
                          printf("No podemos atenderlo en este momento, favor de regresar más tarde\n");
                       break;
           case 1: res = deQueue(f, &valor);
                        if(res == 1)
                           printf("Atendiendo a: %d\n", valor);
                         else
                           printf("No hay clientes, descansar\n");
                        break;
            case 2: printf("La tienda está cerrando\n");
          }
         } while(op != 2);
}
```

* Función para <u>*vaciar*</u> los datos de una pila y almacenarlos en un(a) queue/cola/fila.

```
int pilaToQueue(PILA *p, FILA *f)
{
  int res, valor;

  res = iniQueue(f, p->tope+1);
  if(res == 1)
     while( pop(p, &valor) == 1)
        enQueue(f, valor);

  return(res);
}
```

* Función isQueueFull : Checks if the queue is full

```
int isQueueFull(FILA f)
{
  int res = 0;

  if( f.final + 1 ==  f.tam )
     res = 1;

  return(res);
}
```

* Función isQueueEmpty : Checks if the queue is empty

```
int isQueueEmpty(FILA f)
{
  int res = 0;

  if( f.final ==  -1)
     res = 1;
  return(res);
}
```

* Función peekQueue: Gets the element at the front of the queue without removing it.

```
int peekQueue(FILA f, int *valor)
{
   int res=0;

   if(f.frente > -1)
   {
     *valor = f.fila[f.frente];
     res=1;
   }

   return(res);
}
```