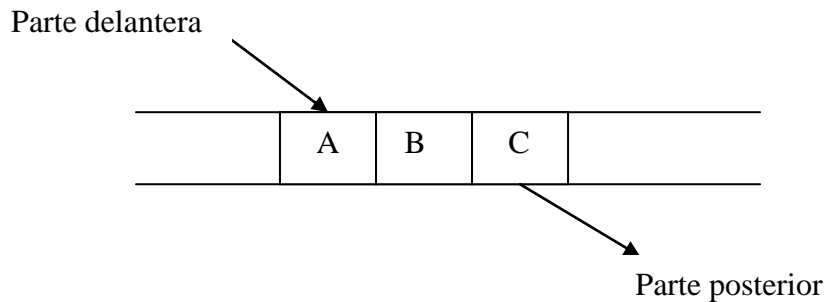


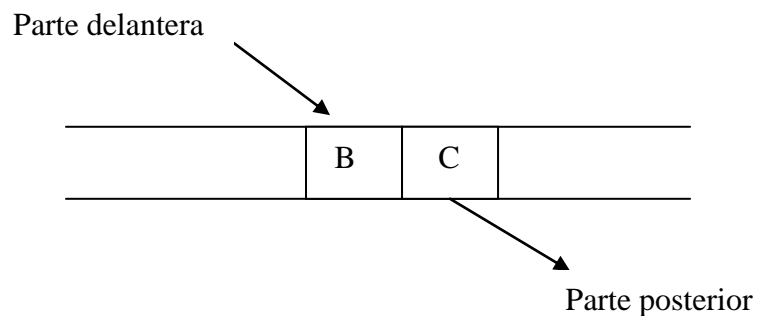
UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION II (SISTEMAS)

La estructura de tipo cola y su representación secuencial.

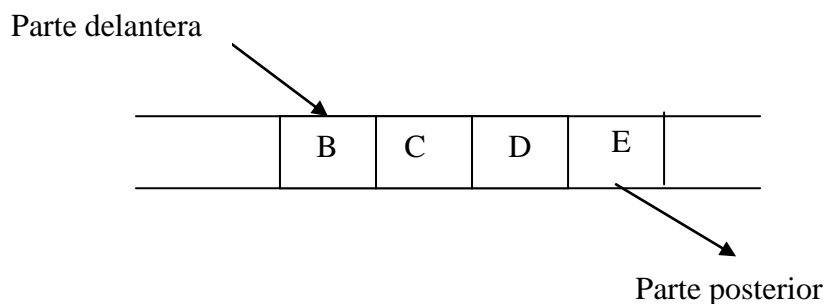
Una cola es un conjunto ordenado de elementos del que pueden suprimirse elementos de un extremo (llamado la parte delantera de la cola) y en el que pueden insertarse elementos del otro extremo (llamado la parte posterior de la cola).



En la figura anterior se ilustra una cola que contiene tres elementos A, B, y C. A esta en la parte delantera de la cola y C esta en la posterior.



En la figura anterior, se ha eliminado un elemento de la cola. Dado que solo pueden suprimirse elementos de la parte delantera de la cola, se remueve A y B está ahora al frente.



En la figura anterior, cuando se insertan los elementos D y E, debe hacerse en la parte posterior de la cola. Puesto que D se insertó antes de E, esta se retirará primero. El primer elemento insertado en la cola es el primer elemento que se suprime. Por esta razón, en ocasiones, una cola se denomina una lista fifo (first in first out, el primero que entra, el primero en salir).

Los ejemplos de cola abundan en el mundo real: una fila en un banco, un grupo de automóviles esperando en una caseta de peaje, etc.

OPERACIONES PRIMITIVAS:

Se aplican tres operaciones primitivas a una cola.

- La operación **insert(q, x)** inserta el elemento x en la parte posterior de la cola q.
- La operación **x = remove(q)**, suprime el elemento delantero de la cola q y establece su contenido en x.
- La operación **empty(q)**, retorna false o true, dependiendo si la cola contiene elementos o no.

La operación **insert(q, x)** puede ejecutarse siempre, pues no hay límite en la cantidad de elementos que puede contener una cola. Sin embargo, la operación **remove (q)** sólo puede aplicarse si la cola no está vacía; no hay forma de remover un elemento de una cola que no contiene elementos. El resultado de un intento no válido de remover un elemento de una cola vacía se denomina **subdesbordamiento**.

Implementación de colas en C:

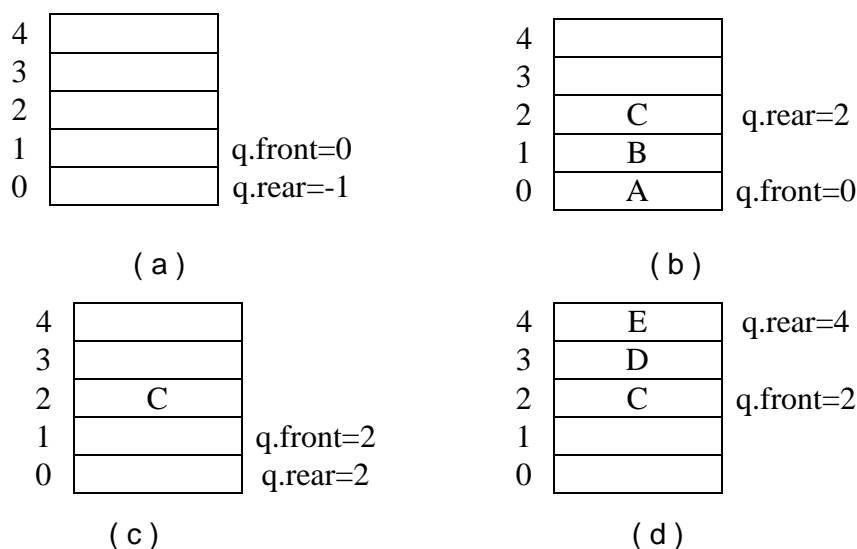
Opción: arreglo lineal

Se usa un arreglo para contener los elementos de la cola y dos variables, front (parte delantera) y rear (parte posterior) para contener las posiciones dentro del arreglo de los elementos primero y ultimo de la cola.

Al principio se establece q.rear en -1 y q.front en 0. La cola esta vacía cada vez que $q.rear < q.front$.

La cantidad de elementos en la cola en cualquier momento es igual al valor de $q.rear - q.front + 1$.

Problema:



Figura

En (a) la cola esta vacía, no tiene elementos, $q.front = 0$ y $q.rear = -1$.
En (b) se insertan los elementos A, B, y C.

En (c) se han suprimido dos elementos A y B.

En (d) se han insertado dos nuevos elementos, D y E. El valor de q.front es 2 y el valor de q.rear es 4, porque solo hay $4 - 2 + 1 = 3$ elementos de la cola. Debido a que el arreglo contiene cinco elementos, habría espacio para que la cola creciera sin la preocupación de un desbordamiento.

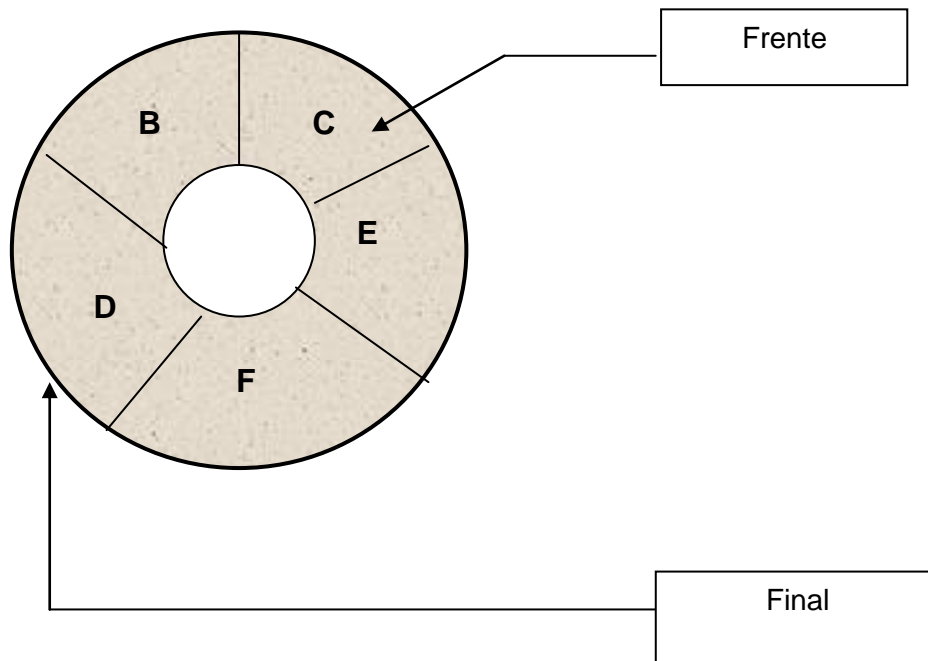
Si ahora se quiere insertar F en la cola, q.rear debe incrementar su valor en 1 hasta llegar a 5 y q.ítems[5] debe establecerse en el valor F. Pero, el arreglo solo tiene 5 elementos, así que no puede hacerse esta inserción.

Por lo tanto, esta representación de arreglo lineal no es aceptable.

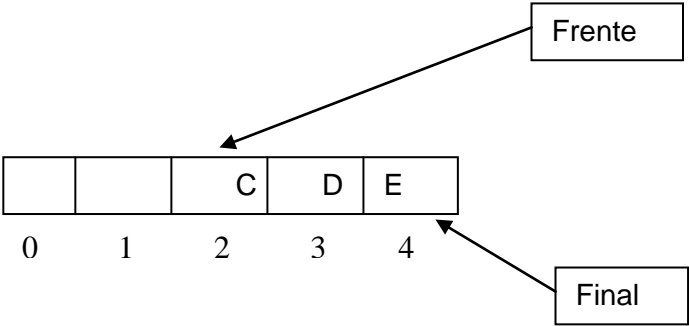
Una solución: modificar la operación remove para que cuando se suprima un elemento, toda la cola se cambie al principio del arreglo. La cola ya no necesita la variable front ya que el elemento en la posición 0 del arreglo siempre será la parte delantera de la cola.

Este método no es eficiente. Cada supresión de un elemento significa mover cada uno de los elementos restantes en la cola.

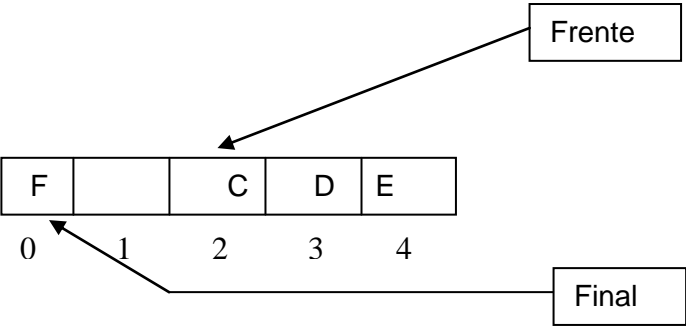
Otra solución: Considerar que el arreglo que contiene la cola es un círculo. Imaginamos que el primer elemento del arreglo está inmediatamente después de su último elemento. Esto implica que incluso si se ocupa el último elemento, puede insertarse un valor nuevo detrás de él en el primer elemento del arreglo, mientras este primer elemento esté vacío.



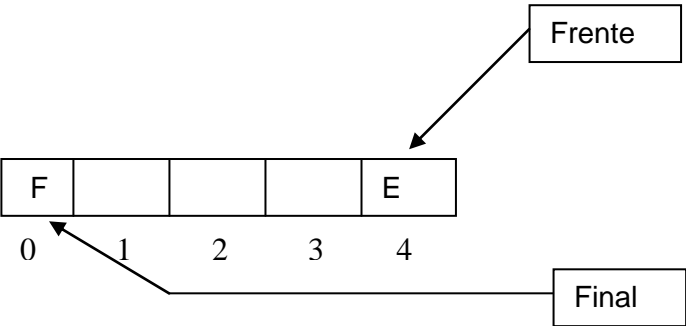
A continuación se verá un ejemplo. Supóngase que una cola contiene tres elementos en las posiciones 2, 3, y 4 de un arreglo de 5 elementos.



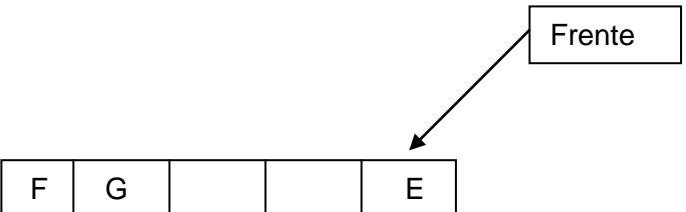
(a)



(b)



(c)



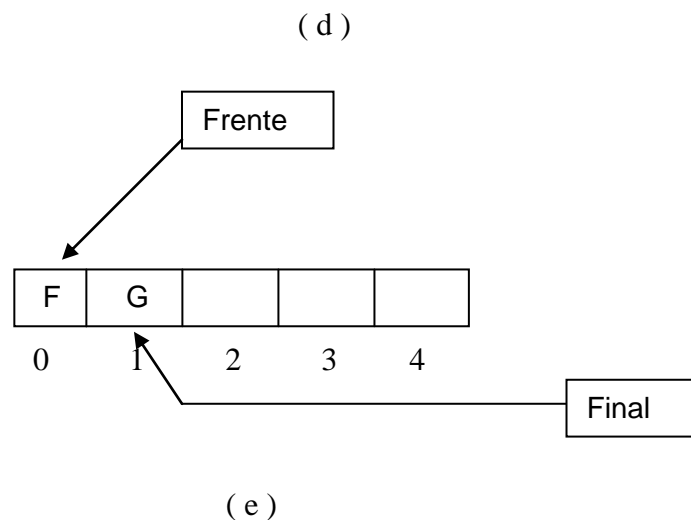


FIGURA 6.3

Ahora se insertará el elemento F en la cola, este puede colocarse en la posición 0 del arreglo como se muestra en la figura 6.3b. El primer elemento de la cola está en `q.items [2]`, el que es seguido por `q.items [3]`, `q.items[4]`, y `q.items[0]`. Las figuras 6.3c, 6.3d y 6.3e muestran el estado de la cola cuando eliminan en primer lugar los elementos D y C, luego se inserta G y por último se borra E.

Desafortunadamente, no es fácil determinar con esa representación si la cola está vacía. La condición `q.rear < q.front` deja de ser válida para la verificación de si la cola esta vacía. Las figuras 6.3b, c y d, por ejemplo, ilustran situaciones en las que la condición es verdadera aunque la cola no esté vacía.

Una manera de resolver este problema es establecer la regla convencional de que el valor de `q.front` es el índice del arreglo que precede de inmediato al primer elemento de la cola en lugar del índice del primer elemento mismo. Por lo tanto, como `q.rear` es el índice del último elemento de la cola, la condición `q.front == q.rear` implica que la cola está vacía.

En consecuencia, una cola de enteros puede declararse e inicializarse por medio de:

```
# define maxq 100
struct queue {
    int items[maxq];
    int front,rear;
};
```

```
struct queue q;  
q.front=q.rear ==maxq-1;
```

Adviértase que q.front y q.rear son inicializadas como el último índice del arreglo y no como -1 ó 0, porque el último elemento del arreglo precede de inmediato al primero dentro de la cola de esta representación. Debido a que q.rear es igual a q.front; al inicio la cola está vacía.

La Función Empty.

La función empty puede codificarse como:

```
Int empty(struct queue *pq)  
{  
    return((pq->front==pq->rear)? TRUE : FALSE );  
}/* fin del empty*/
```

Si ya esta definida la función empty entonces ahora podemos decir en el programa principal:

```
If ( empty(&q) )  
/*la cola está vacía*/  
else  
/* la cola no está vacía*/
```

La Operación remove

```
int remove (struct queue *pq)  
{  
    if ( empty(pq) ) {  
        printf ("la cola está vacía");  
        exit(1);  
    } /*fin de if*/  
    if (pq->front ==maxq-1)  
        pq->front=0;  
    else  
        (pq->front)++;  
    return (pq->items[pq->front]);  
}/* fin de remove*/
```

La Operación insert

La operación insert comprende una verificación de desborde que ocurre cuando todo el arreglo está ocupado por elementos de la cola y se intenta inserta otro. Por ejemplo considere la cola de la figura 6.1b hay tres elementos en la cola: c, d y e en q.items[2],q.items[3], q.items[4],respectivamente. Como el último elemento de la cola ocupa el lugar q.items[4],q.rear igual a 4, y q.front es igual a 1 debido a que el primer elemento de la cola está en q.items[2]. Si luego insertamos en la cola los elementos F y G. En este momento el arreglo está lleno y cualquier intento por insertar otro elemento causaría desborde. Pero esto está indicado por el hecho que q.front es igual a q.rear, lo que constituye precisamente la indicación de subdesbordamiento.

Sin embargo hay manera de distinguir entre la cola vacía y la que está llena. Es evidente que una situación como ésta no resulta satisfactoria. Una solución es

sacrificar un elemento del arreglo y permitir que la cola pueda crecer únicamente hasta alcanzar el tamaño del arreglo menos uno. Así, cuando un arreglo de 100 elementos se declara como una cola, ésta puede tener hasta 99 elementos. Si se intentará insertar el centésimo elemento, ocurriría un desbordamiento. Por lo tanto la rutina insert puede escribirse como sigue:

```
void insert(struct queue *pq, int, x)
{
/* hace espacio para un nuevo elemento */
if ( pq->rear==maxq-1)
    pq->rear=0;
else
    ( pq->rear )++;
/* comprueba que no hay desbordamiento */
if ( pq->rear==pq->front ){
    printf ("desbordamiento en la cola");
    exit(1);
}/* fin de if */
pq->items[pq->rear]=x;
return;
} /* fin de insert*/
```

Colas de prioridad

Definición:

Es una estructura de datos en la que el ordenamiento intrínseco de los elementos determina los resultados de sus operaciones básicas. Hay dos tipos de prioridad ascendente y descendente.

- (ascendente) es una colección de elementos en las que se pueden insertar elementos de manera arbitraria y de las que se pueden eliminarse sólo el elemento menor.

Si apq es una cola de prioridad ascendente, la operación `pqinsert(apq, x)` inserta el elemento `x` dentro de `apq` y la operación `pqmindelete(apq)` elimina el elemento mínimo de `apq` y regresa su valor.

- (descendente) es similar, pero sólo permite la eliminación del elemento mayor. Las operaciones aplicables a una cola de este tipo, `dpq`, son: `pqinsert(dpq, x)` y `pqmaxdelete(dpq)`.

`pqinsert(dpq,x)` inserta el elemento `x` en `dpq` y es idéntica desde el punto de vista lógico a `pqinsert` en el caso de la cola de prioridad ascendente.

`Pqmaxdelete(dpq)` elimina el elemento máximo de `dpq` y regresa el valor.

La operación `empty(pq)` se aplica a ambos tipos de cola de prioridad y termina si la cola de prioridad está vacía. `pqinsert(dpq, x)` y `pqmaxdelete(dpq)` solo pueden aplicarse a colas de prioridad ocupadas (es decir, cuando `empty(pq)` es FALSE).

Una vez que se aplica `pqmindelete` para recuperar el siguiente elemento más pequeño de una cola de prioridad ascendente, se puede ocupar de nuevo para recuperar el siguiente elemento más pequeño, y así sucesivamente. Por esto, la operación recupera de manera sucesiva, elementos de una cola de prioridad de manera ascendente (sin embargo, cuando se inserta un elemento pequeño después

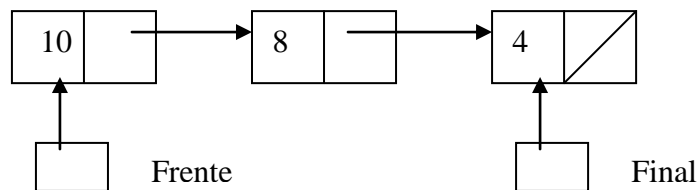
de varias eliminaciones, la siguiente recuperación traerá ese pequeño elemento, que puede ser más pequeño que alguno previamente recuperado). De manera similar, pqmaxdelete recupera los elementos de una cola de prioridad ascendente en el mismo orden. Esto explica la designación de una cola de prioridad como ascendente o descendente.

Los elementos de una cola de prioridad no necesitan ser números o caracteres que se comparen de manera directa. Pueden ser estructuras complejas que están ordenadas en uno o más campos. Por ejemplo, las listas del directorio telefónico, que consisten en apellidos, nombres, direcciones, y números telefónicos, están ordenadas por apellidos.

A veces el campo con el que se ordenan los elementos de una cola de prioridad ni siquiera es parte de los propios elementos; puede ser un valor externo especial, usado con el objetivo específico de ordenar la cola de prioridad. Por ejemplo una pila puede verse como una cola de prioridad descendente cuyos elementos están ordenados por el tiempo de inserción. El último elemento insertado tiene mayor valor de tiempo de inserción y es el único que puede recuperarse. De manera análoga, una cola puede verse como una cola de prioridad ascendente, cuyos elementos están ordenados de acuerdo con el tiempo de inserción.

Representación de colas empleando memoria dinámica.

- Usamos dos atributos de tipo puntero, frente y final, que apunten a los nodos que contienen los elementos frente y final



- ¿Que sucedería si intercambiáramos las posiciones de frente y final en la lista enlazada?

Estructura Nodo:

Para la implementación dinámica de la cola se necesita definir un nodo. Este nodo va a estar formado de una parte donde se guardará el dato (entero, flotante, carácter, etc) y un apuntador hacia el siguiente nodo.

```
struct queueNode {
    char dato;
    struct queueNode * siguiente;
};
```

Apuntadores:

Para completar la implementación de cola se necesita también tener dos apuntadores: uno hacia el frente de la cola y uno hacia el final de la misma. Ver figura anterior. Estos apuntadores deben inicializarse a NULL para que al principio no apunten a nada.

```
typedef struct queueNode * QUEUEPTR;
QUEUEPTR frontPtr = NULL, rearPtr = NULL;
```

Cola vacía:

Para verificar si la cola esta vacía es necesario verificar hacia donde apunta el puntero que indica el frente de la cola; si el puntero es NULL entonces la cola esta vacía. También se puede definir la función isEmpty para que haga esta labor.

```
int isEmpty(QUEUEPTR headPtr)
{
    return headPtr == NULL;
}
```

Inserción de elementos en la cola:

La inserción de elementos en la cola se hace por la parte trasera de la misma.

Para la inserción de elementos en la cola:

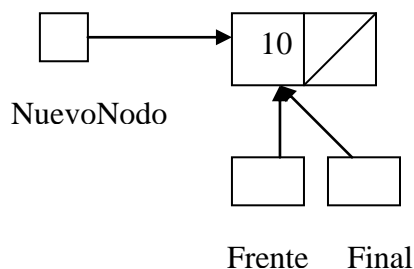
Crear un nuevo nodo.

Verificar si realmente se cuenta con un nuevo nodo. Si no es así imprimir un error.

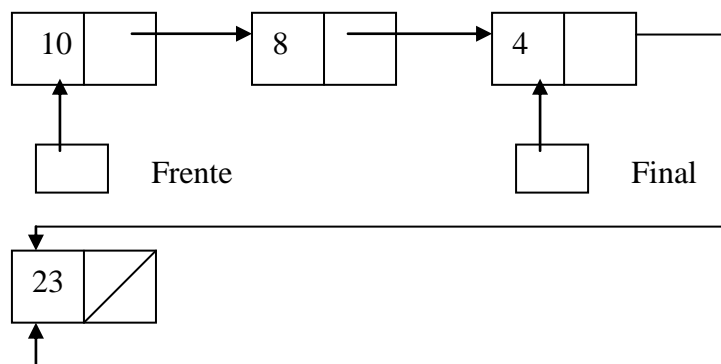
Si la cola esta vacía, hacer que el apuntador del frente de la cola y el del final de la cola apunten al nuevo nodo.


Si la cola no esta vacía, colocar el nodo al final de la cola. El elemento apuntador siguiente del nodo hacia el cual apunta el puntero final debe apuntar hacia el nuevo nodo. Luego cambiar el puntero final para que apunte hacia el nuevo nodo.

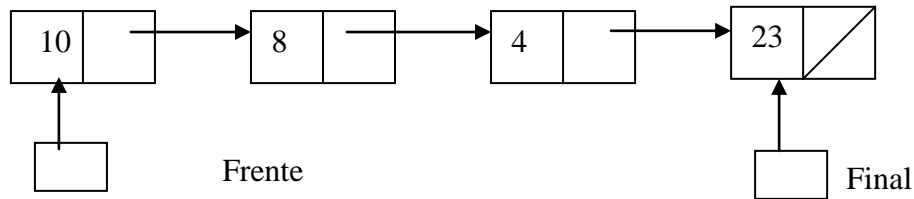
Inserción con cola vacía



Inserción con cola con elementos



 NuevoNodo



```

void insertar ( QUEUEPTR * headPtr, QUEUEPTR *tailPtr, char valor )
{
    QUEUEPTR newPtr;
    newPtr = (QUEUEPTR) malloc (sizeof (QUEUENODE));

    if ( newPtr != NULL ) {
        newPtr->dato = valor;
        newPtr->siguiente = NULL;

        if (isEmpty(*headPtr))
            *headPtr = newPtr;
        else
            (*tailPtr)->siguiente = newPtr;

        *tailPtr = newPtr;
    }
    else
        printf("%c not inserted. No memory available.\n", valor);
}
    
```

Remover un elemento de la cola:

Para quitar un elemento de la cola se hace por el frente. Para remover un elemento de la cola:

Guardar el elemento dato del nodo al frente de la cola en una variable temporal.

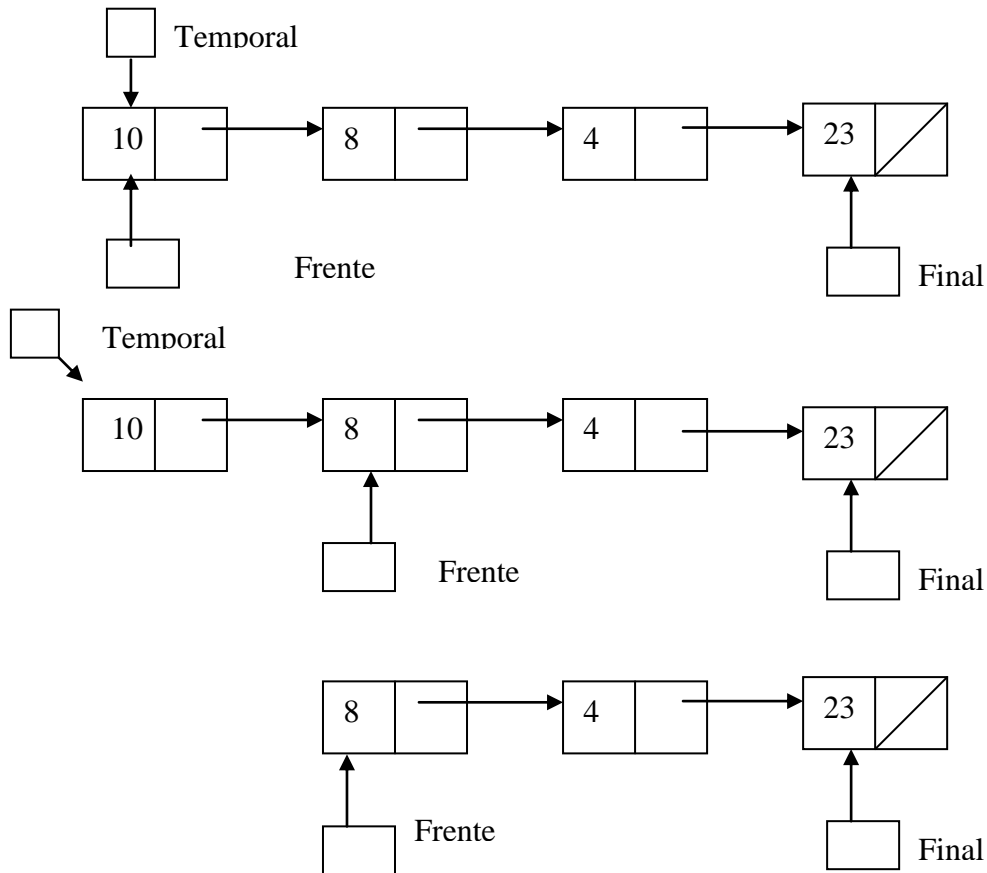
Colocar un puntero temporal hacia el nodo que esta en el frente de la cola.

Luego cambiar el puntero del frente hacia el nodo hacia el cual apunta el nodo que estaba en el frente de la cola.

Si después de eso, el apuntador frente es NULL entonces no hay mas elementos en la cola y el puntero final también debe apuntar hacia NULL.

Liberar la memoria del nodo que se quitó de la cola.

Devolver el valor de la variable temporal con el valor del nodo que se quitó de la cola.



```
char remover (QUEUEPTR *headPtr, QUEUEPTR *tailPtr)
{
    char value;
    QUEUEPTR tempPtr;

    value = (*headPtr)->dato;
    tempPtr = *headPtr;
    *headPtr = (*headPtr)->siguiente;

    if (*headPtr == NULL )
        *tailPtr = NULL;

    free(tempPtr);
    return value;
}
```