Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT

Algoritmos y Estructuras de Datos

Esaú Eliezer Escobar Juárez

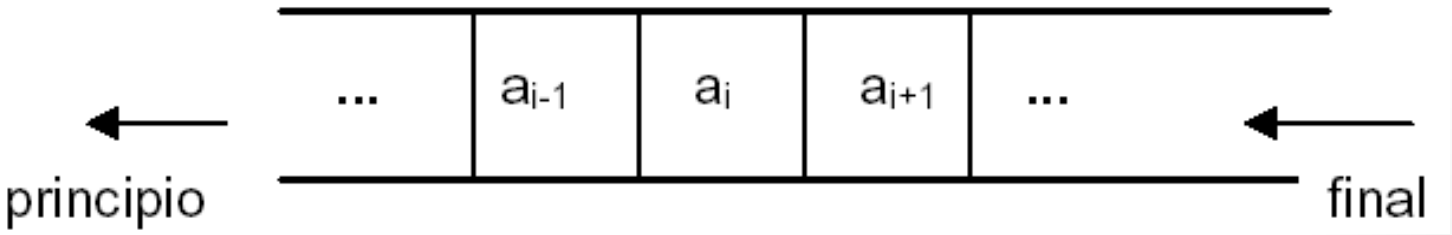
Colas y su implementación en C

TDA COLA

Definición

Def: una cola es una lista ordenada de elementos en la que todas las inserciones se realizan por un extremo (frente o principio) y las supresiones se realizan por el otro (final).

Estructura FIFO (*First In First Out*): "primero en entrar primero en salir"

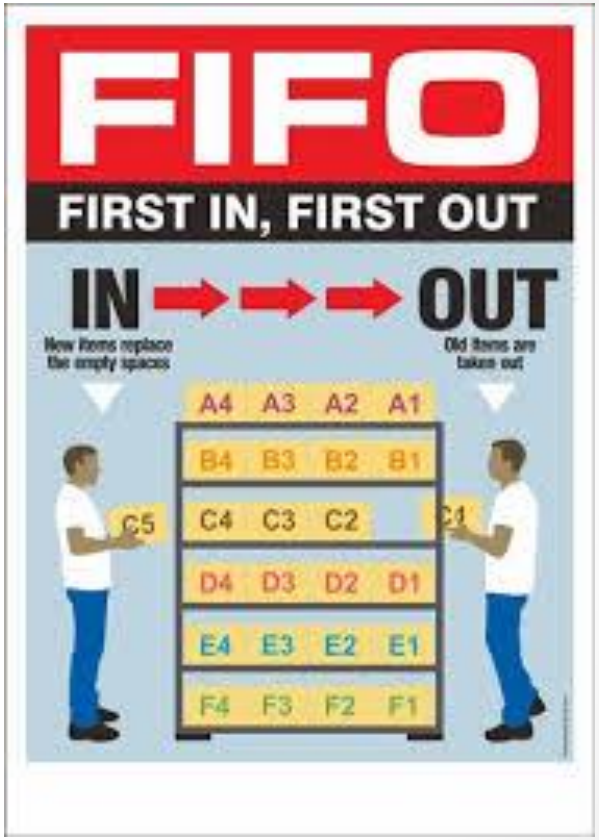



designed by freepik

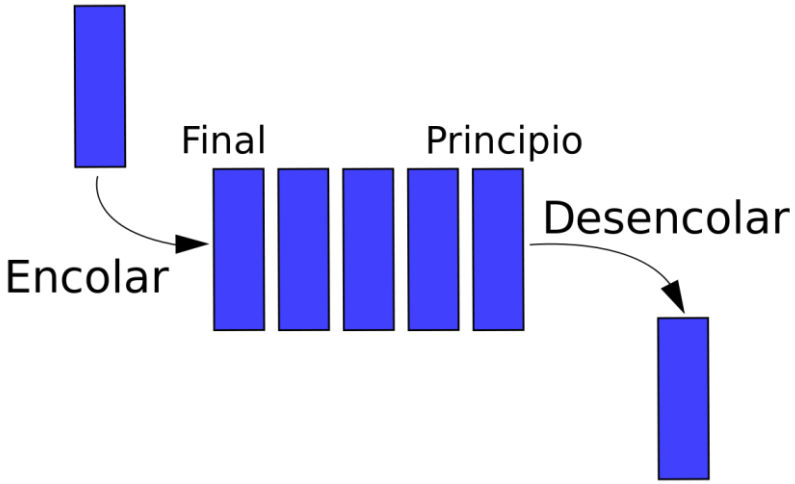

ATM
shutterstock.com · 1251055930

Cola: Funcionamiento

- Para el manejo de los datos cuenta con dos operaciones básicas: adicionar (add, queue), que coloca un objeto al final de la cola, y su operación inversa, retirar o sacar (poll, dequeue), que retira el primer elemento de la fila. En cada momento sólo se tiene acceso al primer elemento de la fila, es decir, al primer objeto.

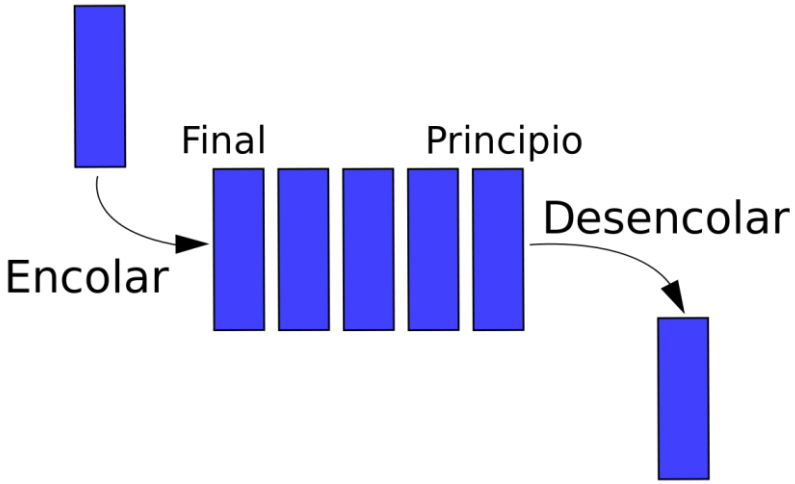


Operaciones básicas



Cola Vacía

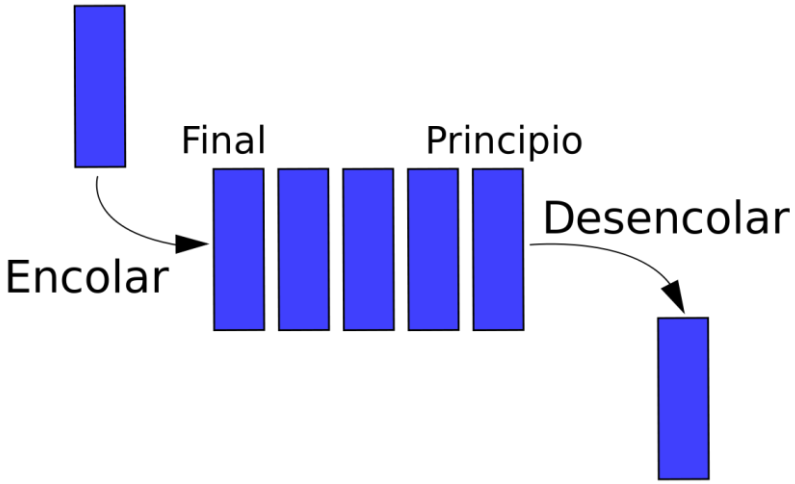
Operaciones básicas



Cola Vacía

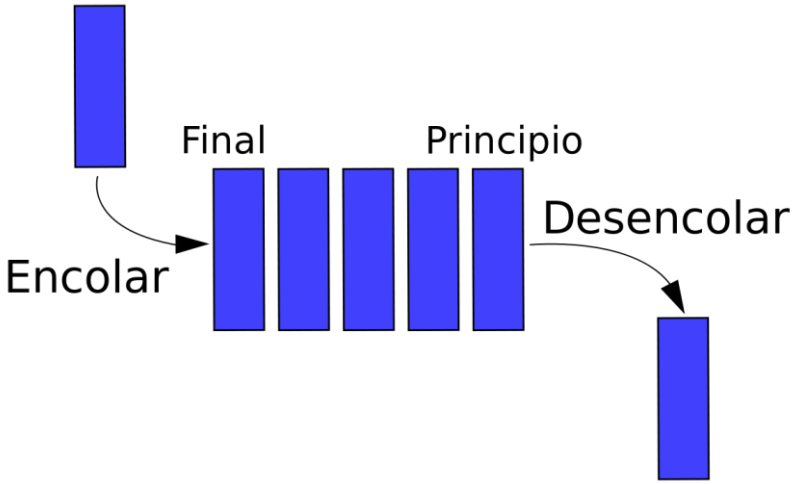
Adiciona

Operaciones básicas



Cola con 1 elemento

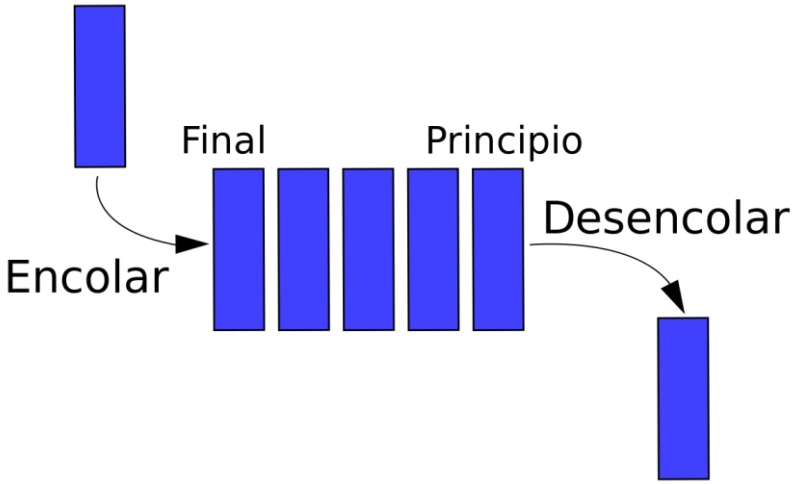
Operaciones básicas



Cola con 1 elemento

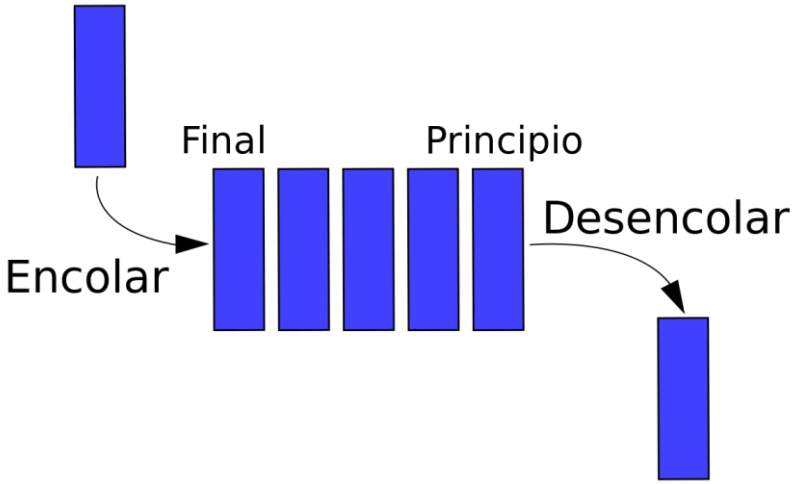
Adiciona

Operaciones básicas



Cola con 2 elementos

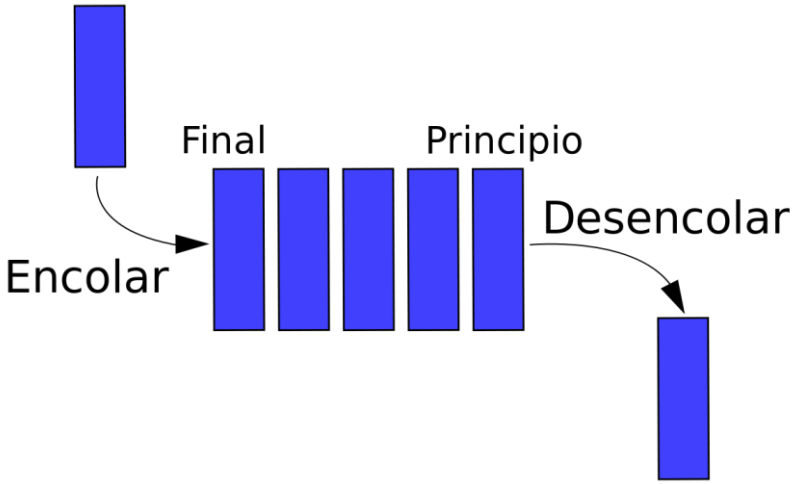
Operaciones básicas



Cola con 2 elementos

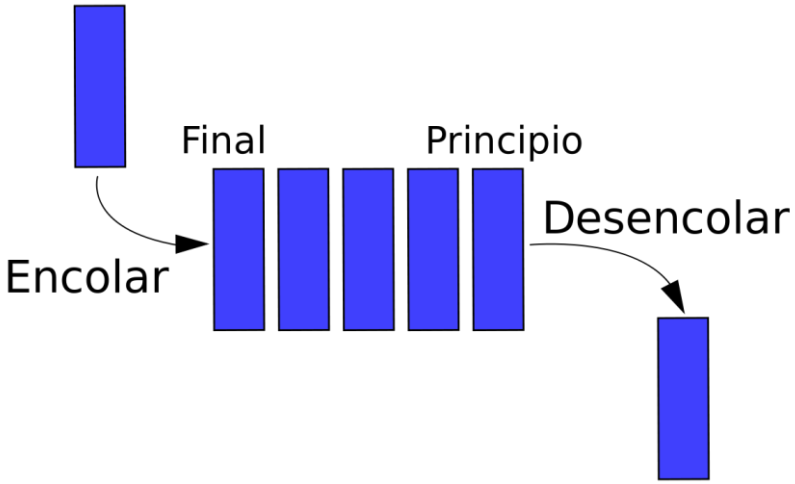
Adiciona

Operaciones básicas



Cola con 3 elementos

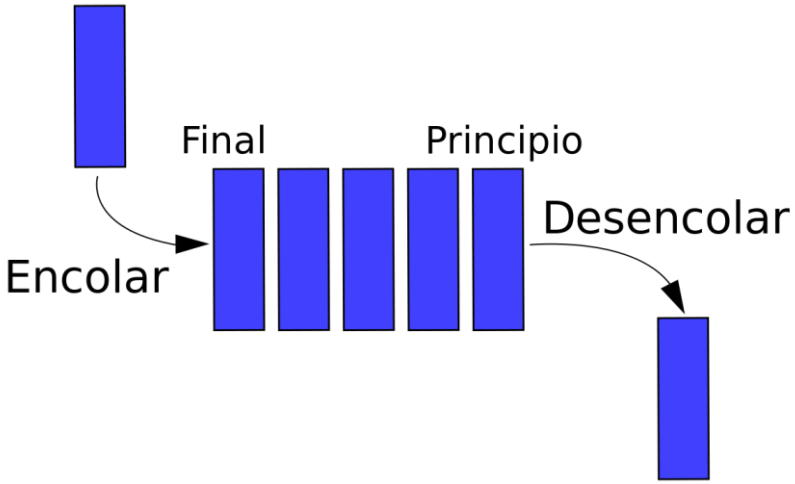
Operaciones básicas



Cola con 3 elementos

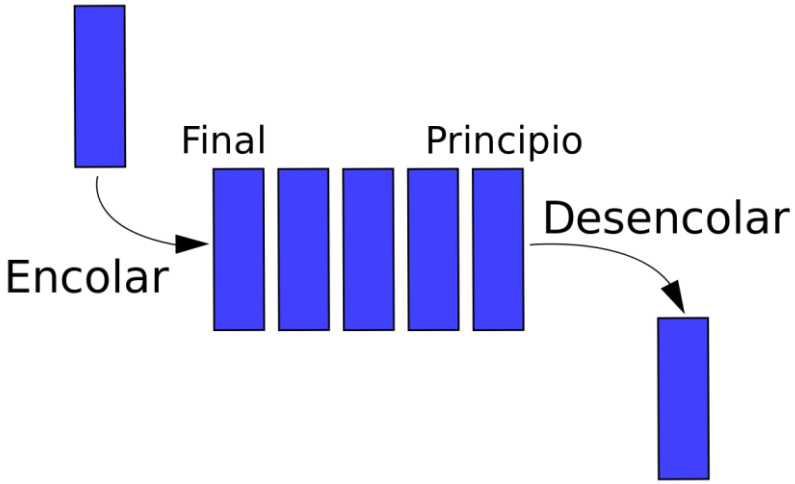
Adiciona

Operaciones básicas



Cola con 4 elementos

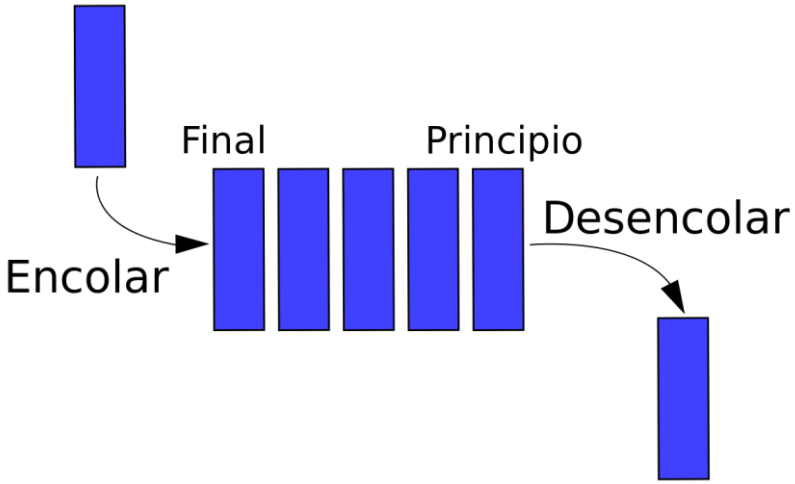
Operaciones básicas



Cola con 4 elementos

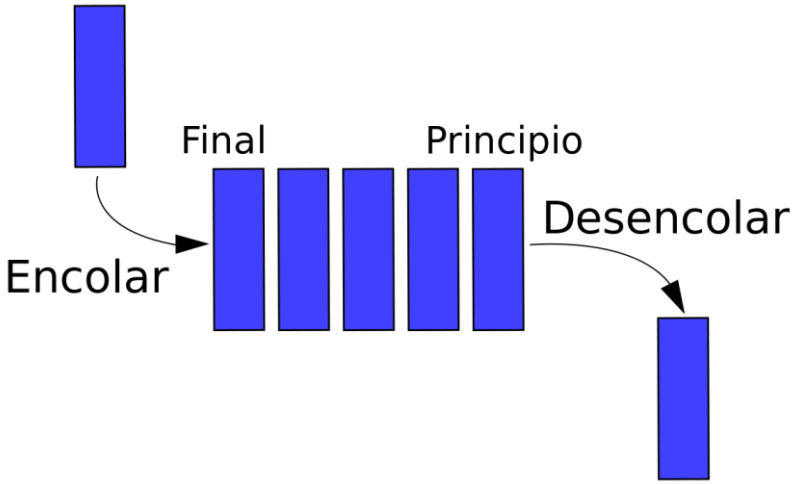
Adiciona

Operaciones básicas



Cola con 5 elementos

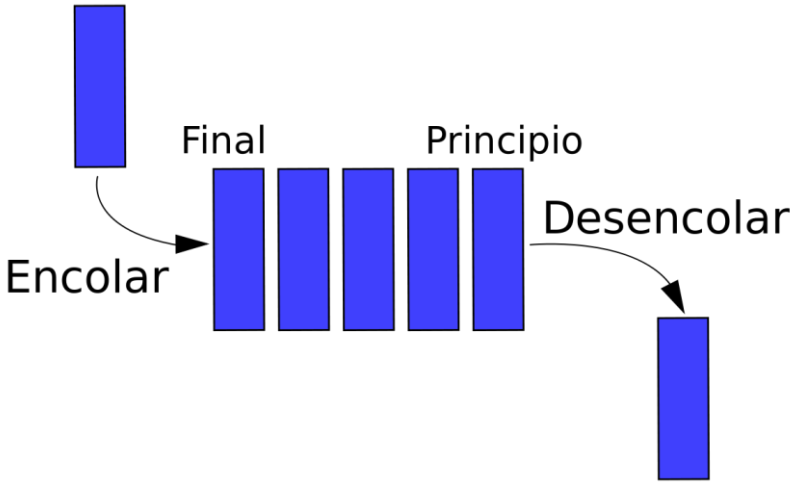
Operaciones básicas



Cola con 5 elementos

Retirar

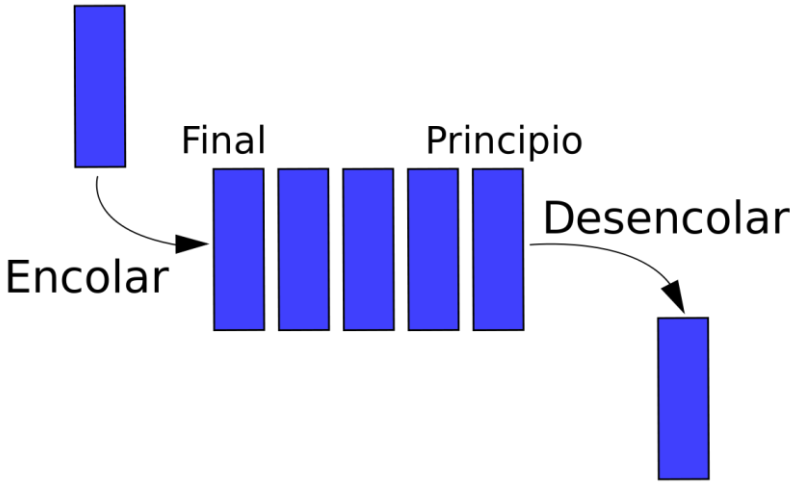
Operaciones básicas



Cola con 4 elementos

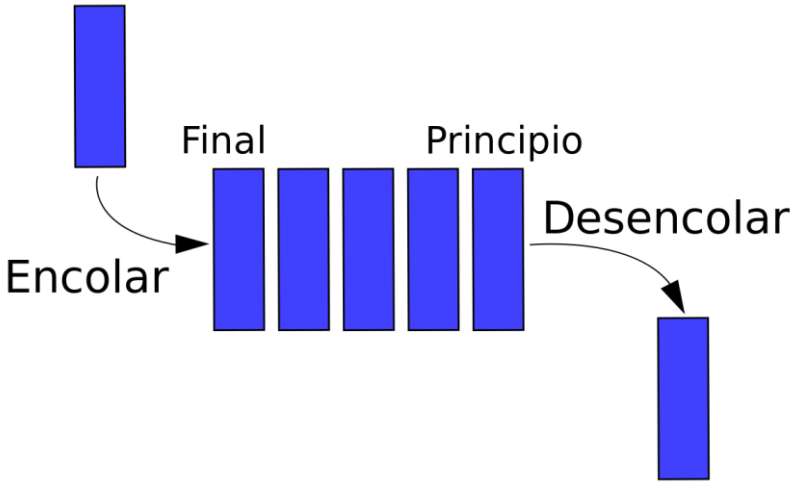
Retirar

Operaciones básicas



Cola con 4 elementos

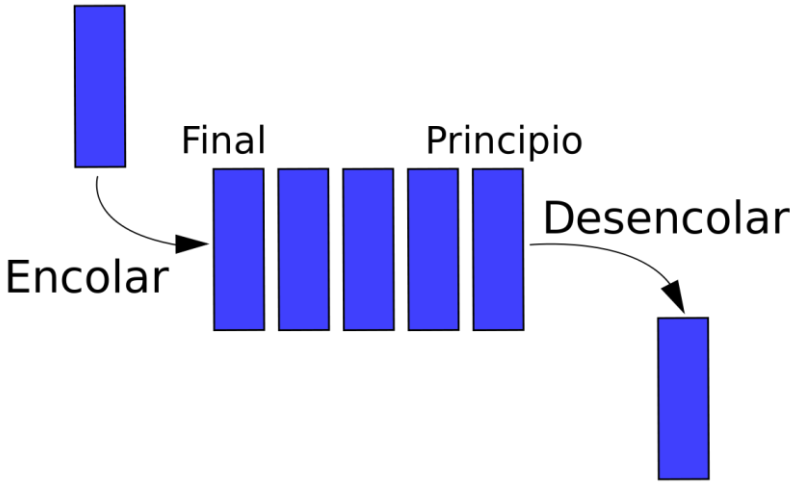
Operaciones básicas



Cola con 4 elementos

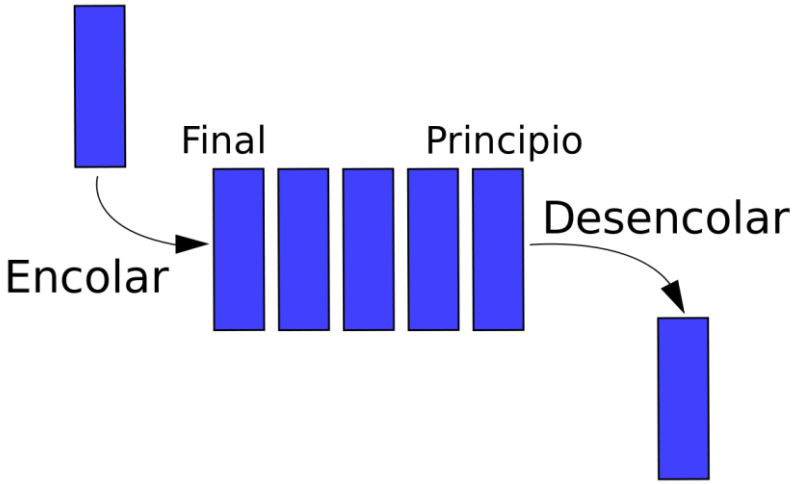
Adiciona

Operaciones básicas



Cola con 5 elementos

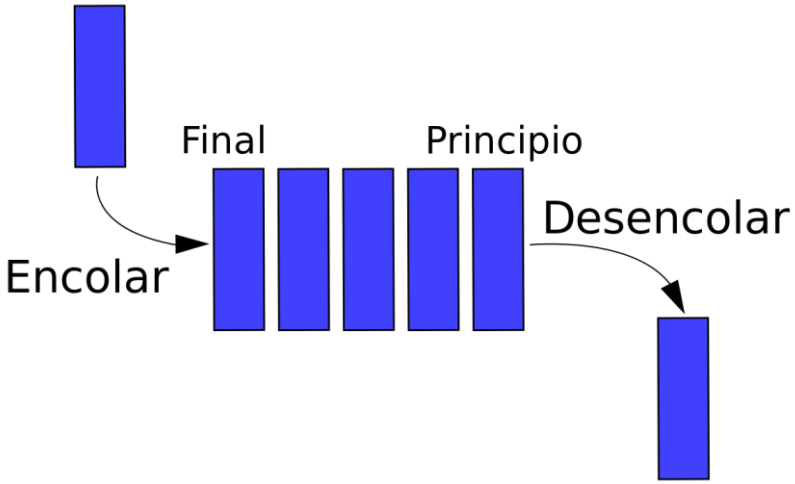
Operaciones básicas



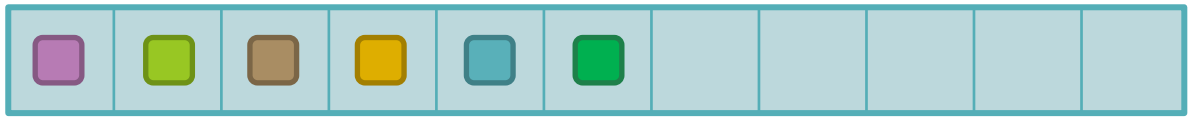
Cola con 5 elementos

Adiciona

Operaciones básicas

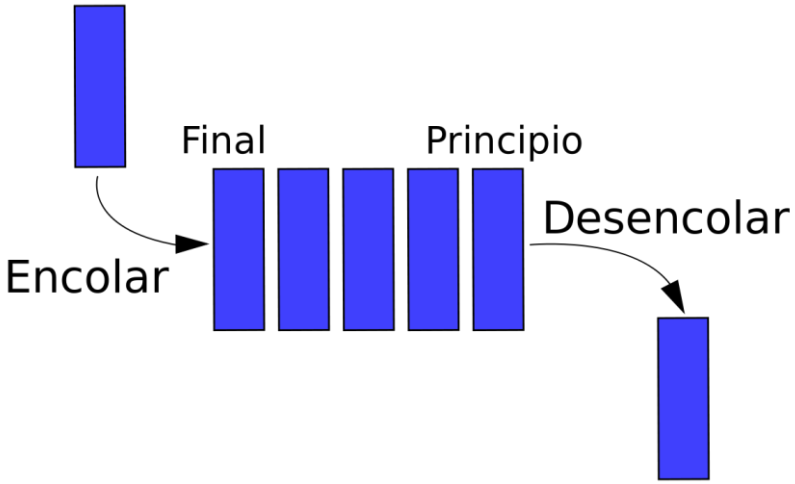


Cola con 6 elementos

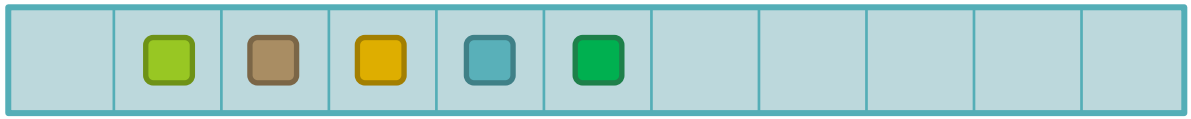


Retirar

Operaciones básicas

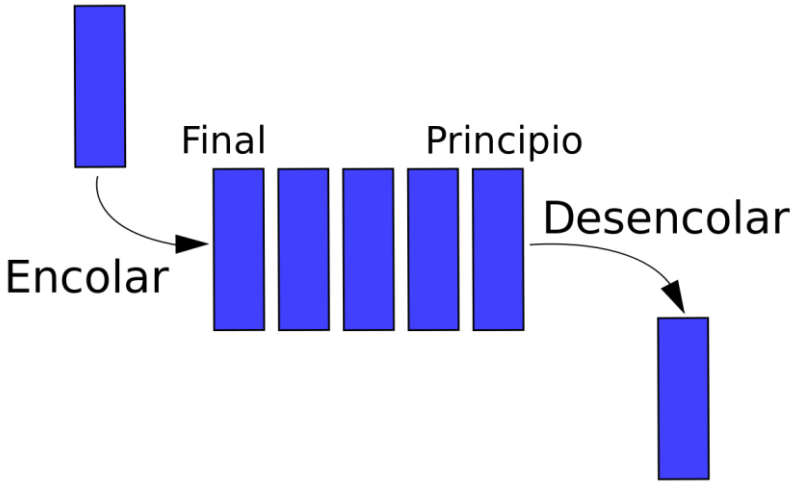


Cola con 5 elementos



Retirar

Operaciones básicas



Cola con 5 elementos

Operaciones

- **Crear:** crea la cola vacía. (C: cola, ok: lógico)
- **Vacía:** devuelve cierto si la cola está sin elementos o falso en caso de que contenga alguno. (C: cola, resp: lógico)
- **Llena:** devuelve cierto si la cola está llena, en implementación estática, falso en caso contrario. (C: cola, resp: lógico)
- **Adicionar:** añade un elemento a la cola. (C: cola, X: elemento, resp: lógico)
- **Retirar:** lee y retira el primer elemento de la cola. (C: cola, X: elemento, resp: lógico)
- **Tamaño:** regresa el número de elementos de la cola (C: cola, N: numérico)
- **Borrar:** Limpia todos los elementos. (C: cola, ok: lógico)

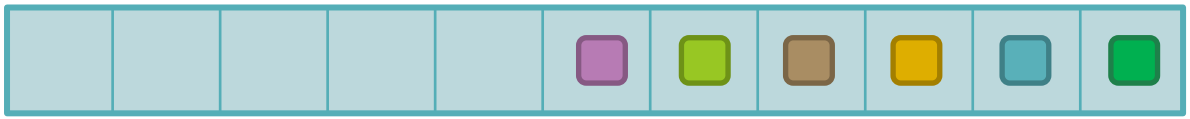
Implementación estática

https://codeshare.io/

Variante de la función retirar

- Al retirar un elemento en lugar de recorrer todos los elementos una posición, esperamos a que final == limite y si hay espacios vacíos al frente entonces recorremos.

Cola con 5 elementos



Retirar

Implementación dinámica

- También llamada representación de lista enlazada, utiliza apuntadores para implementar un estructura de datos de tipo cola.

Implementación dinámica: nodo

De igual manera que la pila, las colas utilizan una estructura *Nodo* que posean 2 atributos*:* un contenido y un apuntador al siguiente Nodo

Añadir un elemento a una cola vacía

- Partiremos de que ya tenemos el nodo a insertar y, por supuesto un puntero que apunte a él, además los punteros que definen la cola, primero y último que valdrán NULL:

- El proceso es muy simple, bastará con que:
  - **nodo->siguiente** apunte a **NULL**.
  - Y que los punteros **primero y último** apunten a **nodo**.

Añadir elemento en una cola no vacía

De nuevo partiremos de un nodo a insertar, con un puntero que apunte a él, y de una cola, en este caso, al no estar vacía, los punteros **primero y último** no serán nulos:

- El proceso sigue siendo muy sencillo:
  - Hacemos que **nodo->siguiente** apunte a **NULL**.
  - Después que **ultimo->siguiente** apunte a **nodo**.
- Y actualizamos **último**, haciendo que apunte a **nodo**.

Añadir elemento en una cola, caso general

- Para generalizar el caso anterior, sólo necesitamos añadir una operación:
- Hacemos que **nodo->siguiente** apunte a **NULL**.
- Si ultimo no es **NULL**, hacemos que **ultimo->siguiente** apunte a **nodo**.
- Y actualizamos **último**, haciendo que apunte a **nodo**.
- Si primero es **NULL**, significa que la cola estaba vacía, así que haremos que **primero** apunte también a **nodo**.

Recuperar un elemento

- Recordemos que leer un elemento de una cola, implica eliminarlo. Ahora también existen dos casos, que la cola tenga un solo elemento o que tenga más de uno.
  - Usaremos un puntero a un nodo auxiliar:
  - Hacemos que nodo apunte al primer elemento de la cola, es decir a primero.
  - Asignamos a primero la dirección del segundo nodo de la pila: primero->siguiente.
  - Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación de lectura en colas implica también borrar.

Leer un elemento en una cola con un solo elemento

También necesitamos un puntero a un nodo auxiliar:

- Hacemos que **nodo** apunte al primer elemento de la cola, es decir a **primero**.
- Asignamos **NULL** a **primero,** que es la dirección del segundo nodo teórico de la cola: **primero=primero->siguiente**.
- Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación de lectura en colas implica también borrar.
- Liberamos la memoria asignada al primer nodo, el que queremos eliminar.
- Hacemos que ultimo apunte a **NULL**, ya que la lectura ha dejado la cola vacía.

Leer un elemento en una cola, caso general

- Hacemos que **nodo** apunte al primer elemento de la pila, es decir a **primero**.
- Asignamos a primero la dirección del segundo nodo de la pila: **primero->siguiente**.
- Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación de lectura en colas implica también borrar.
- Liberamos la memoria asignada al primer nodo, el que queremos eliminar.
- Si primero es **NULL**, hacemos que ultimo también apunte a **NULL**, ya que la lectura ha dejado la cola vacía.

Implementación dinámica

https://codeshare.io/

Aplicaciones de las colas

Principalmente: gestión de recursos

- Sistemas de tiempo compartido: los recursos (CPU, memoria, …) se asignan a los procesos que están en cola de espera en el orden en el que fueron introducidos.
- Colas de impresión: al intentar imprimir varios documentos a la vez o la impresora está ocupada, los trabajos se almacenan en una cola según el orden de llegada.
- Simulación por computadora de situaciones reales: una cola de clientes en un supermercado o el tiempo de espera para ser atendidos por un operador de una línea telefónica.

Cola de prioridad

- Una **cola de prioridad** es una generalización de los conceptos de pila y cola en la que, tras entrar en la cola, **la salida de los elementos** no se basa necesariamente en el orden de llegada sino que **se basa en un orden definido entre ellos**.
- La extracción de elementos de una cola de prioridad puede buscar minimizar o maximizar el valor del elemento saliente.
  - La interpretaremos en el sentido de minimizar. Para maximizar tan sólo hay que cambiar el sentido de las comparaciones.
- En el modelo básico de una cola de prioridad la operaciones que consideramos son:
  - insertar en la cola,
  - obtener el elemento mínimo, y
  - eliminar el elemento mínimo.
- La realización de estas operaciones no requiere mantener ordenada (ni total ni parcialmente) la colección de elementos en la cola.

Colas de prioridad

- Algunas aplicaciones importantes de las colas de prioridad:
  - **Gestión de procesos** en un sistema operativo. Los procesos no se ejecutan uno tras otro en base a su orden de llegada. Algunos procesos deben tener prioridad (por su mayor importancia, por su menor duración, etc.) sobre otros.
  - Implementación de **algoritmos voraces**, los cuales proporcionan soluciones globales a problemas basándose en decisiones tomadas sólo con información local. La determinación de la mejor opción local suele basarse en una cola de prioridad.
  - Implementaciones eficientes de **algoritmos de simulación** de sucesos discretos. En éstas, el avance del tiempo no se gestiona incrementando un reloj unidad a unidad, sino incrementado sucesivamente el reloj al instante del siguiente suceso.

TAD Cola de Prioridad

- **Elementos:** En el conjunto de elementos X hay definido un **orden lineal**.
  - CP es el conjunto de multiconjuntos finitos de elementos de X.

  (En un multiconjunto los elementos pueden repetirse. P.e.: {7, 4, 7, 2, 7, 4})

- **Operaciones:** Dados CP y x:
  - **Crear():** Devuelve ∅.
  - **Destruir(CP):** Elimina CP.
  - **EstáVacía(CP):** Devuelve **cierto** si CP = ∅, y **falso** si CP != ∅.
  - **Longitud(CP):** Devuelve |CP| (cardinalidad del multiconjunto).

- Operaciones (cont.):
  - **Mínimo(CP):** Si CP != ∅, entonces devuelve *y* ∈ CP tal que ∀z ∈ CP, y ≤ z; en otro caso, error.
  - **Insertar(CP, x):** Devuelve CP'= CP ∪ {x}.

    (En la unión de multiconjuntos A ∪ B, las ocurrencias de los elementos de A se añaden a las de B.

    P.e.: {7, 4, 7, 2, 7, 4}∪{4, 5, 1, 4} = {7, 4, 7, 2, 7, 4, 4, 5, 1, 4}.)
  - **EliminarMín(CP)** : Si CP !=∅, entonces devuelve CP' =CP − {Mínimo(CP)}; en otro caso, error.

    (En la diferencia de multiconjuntos A−B, para cada elemento de A se descuentan tantas ocurrencias como haya en B de ese elemento. Si en B hay las mismas o más ocurrencias de un elemento que en A, éste no está en A − B.

    P.e.: {7, 4, 7, 2, 7, 4} − {4, 2, 7, 2, 4} = {7, 7}.)

Cola de prioridad

Ejemplos:

$CP =$**Crear()**	**Insertar**(CP,32)	**Insertar**(CP,13)	**Mínimo**(CP)$= 13$
$\{\,\}$	$\{\,32\,\}$	$\{\,32, 13\,\}$	$\{\,32, 13\,\}$
EliminarMín(CP)	**Insertar**(CP,7)	**Insertar**(CP,26)	**Insertar**(CP,7)
$\{\,32\,\}$	$\{\,7, 32\,\}$	$\{\,7, 32, 26\,\}$	$\{\,7, 32, 7, 26\,\}$
Mínimo(CP)$= 7$	**EliminarMín**(CP)	**Insertar**(CP,18)	**Insertar**(CP,18)
$\{\,7, 32, 7, 26\,\}$	$\{\,7, 32, 26\,\}$	$\{\,7, 32, 26, 18\,\}$	$\{\,7, 18, 32, 26, 18\,\}$
Longitud(CP)$= 5$	**EliminarMín**(CP)	**EliminarMín**(CP)	**Mínimo**(CP)$= 18$
$\{\,7, 18, 32, 26, 18\,\}$	$\{\,18, 32, 26, 18\,\}$	$\{\,18, 32, 26\,\}$	$\{\,18, 32, 26\,\}$

Ejercicio

- Realice un programa Agenda que almacene en una cola las actividades del día, el programa deberá permitir:
  - Insertar una actividad.
  - Borrar una actividad.
  - Ver todas las actividades del día.

- A demás de realizar estas funciones deberá crear una función para asignar memoria a la nueva actividad y otra función que borre todos los elementos, esta última función se realiza para liberar toda la memoria asignada dinámicamente.

- La estructura a utilizar es la siguiente:

```
typedef struct actividad agenda;
struct actividad{
      char act[50]; //Actividad a realizar...
      char hora[10]; //Hora de la actividad...
      agenda *siguiente; //Puntero al siguiente elemento
};
```