Pilas y colas

E. Rivas

Abril de 2014

Resumen

Pilas

- Definición y propiedades
- Implementaciones

Colas

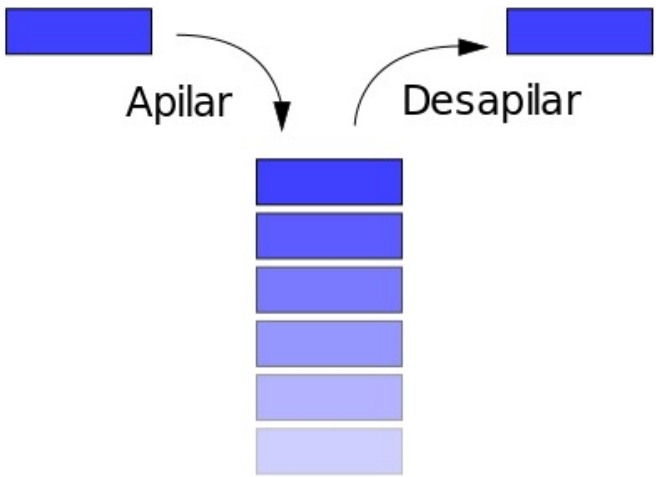
- Definición y propiedades
- Implementaciones

Generalidad

Pilas - definición

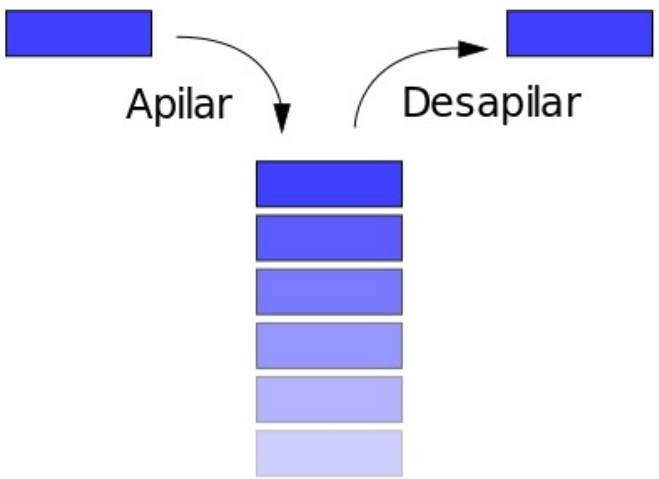
Una **pila** es un tipo abstracto de datos que lleva una colección ordenada de valores de manera LIFO (Last-In-First-Out).

A diferencia de una lista, sus principales operaciones son la inserción de elementos, y la extracción de los mismos.



Pilas - definición

La sigla **LIFO** se refiere a que los elementos se quitan en el orden inverso al que fueron ingresados: *último en entrar, primero en salir*.



Pilas - propiedades y operaciones

Al igual que las listas, llevan un orden lineal. Sin embargo, no podemos acceder de manera directa a cualquier elemento, ni agregar en cualquier posición, o conocer su "longitud".

Sus principales operaciones son:

- Insertar un elemento: push.
- Observar el elemento superior: top.
- Quitar el elemento superior: pop.
- Verificar si está vacía: is_empty.

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones...

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

¿Qué estructuras podemos utilizar para implementar una pila?

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

¿Qué estructuras podemos utilizar para implementar una pila?

Arreglos

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

¿Qué estructuras podemos utilizar para implementar una pila?

Arreglos, listas enlazadas

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

¿Qué estructuras podemos utilizar para implementar una pila?

Arreglos, listas enlazadas, listas enlazadas dobles

Pilas - implementaciones

¿Como podemos implementar este tipo abstracto?

Necesitamos hacer concreta la manera en que almacenamos los datos y programamos las operaciones... ¡estructuras de datos!

¿Qué estructuras podemos utilizar para implementar una pila?

Arreglos, listas enlazadas, listas enlazadas dobles.

Pilas - implementación con arreglos

La siguiente estructura en C modela pilas con arreglos:

```
typedef struct _AStack {
    int     data[MAX];
    int     last;
} AStack;
```

El entero `last` hace referencia a cuál es el último elemento ingresado, lo iniciamos en `-1`.

Pilas - implementación con arreglos

La siguiente estructura en C modela pilas con arreglos:

```
typedef struct _AStack {
    int     data[MAX];
    int     last;
} AStack;
```

El entero `last` hace referencia a cuál es el último elemento ingresado, lo iniciamos en `-1`.

Ejercicio: implementar `void astack_push(AStack *s, int val)` que inserta un elemento al tope de una pila.

Pilas - implementación con listas enlazadas

La siguiente estructura en C modela pilas usando listas enlazadas:

```
typedef struct _SLStackNode {
    int                  data;
    struct _SLStackNode  *next;
} SLStackNode;

typedef SLStackNode *SLStack;
```

Una pila es representada con un puntero al nodo superior de la pila. El puntero `NULL` representa el stack vacio.

Pilas - implementación con listas enlazadas

La siguiente estructura en C modela pilas usando listas enlazadas:

```
typedef struct _SLStackNode {
    int                  data;
    struct _SLStackNode  *next;
} SLStackNode;

typedef SLStackNode *SLStack;
```

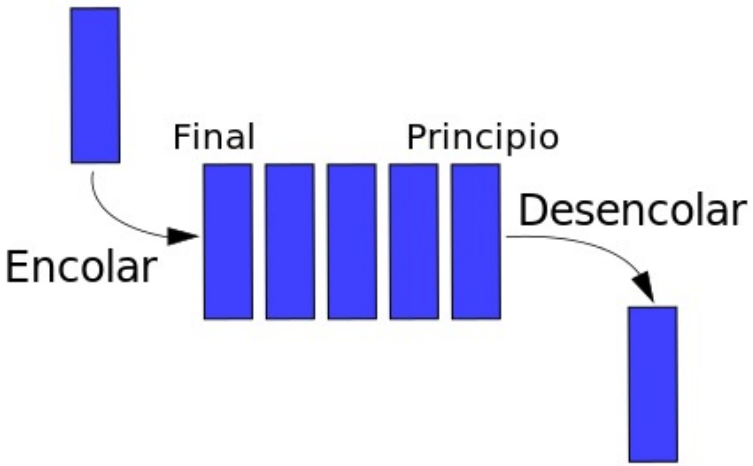
Una pila es representada con un puntero al nodo superior de la pila. El puntero `NULL` representa el stack vacio.

Ejercicio: implementar `void slstack_push(SLStack *s, int val)` que inserta un elemento al tope de una pila.

Colas - definición

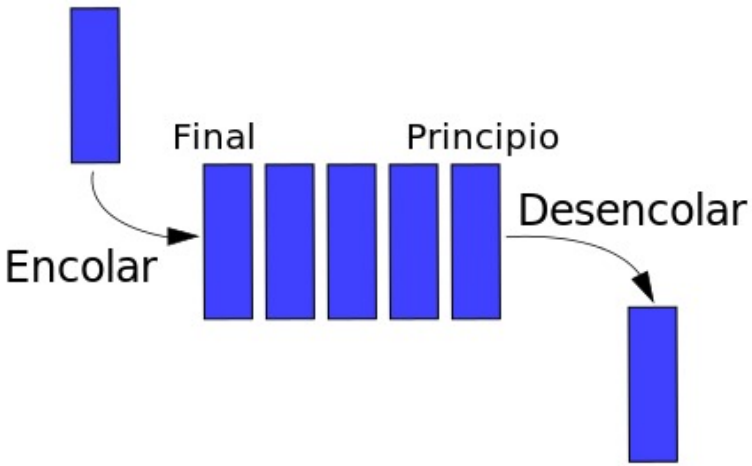
Una **cola** es un tipo abstracto de datos que lleva una colección ordenada de valores de manera FIFO (First-In-First-Out).

Al igual que una pila, sus principales operaciones son la inserción de elementos, y la extracción de los mismos.



Colas - definición

La sigla **FIFO** se refiere a que los elementos se quitan en el mismo orden al que fueron ingresados: *primero en entrar, primero en salir*.



Colas - propiedades y operaciones

Sus propiedades son similares a las de las pilas.

Sus principales operaciones son:

- Insertar un elemento al final: enqueue.
- Observar el elemento al principio: front.
- Quitar el elemento al principo: dequeue.
- Verificar si está vacía: is_empty.

Colas - implementaciones

Nuevamente, las implementaciones disponibles se pueden basar en las estructuras de datos lineales vistas: arreglos o listas enlazadas.

Colas - implementaciones

Nuevamente, las implementaciones disponibles se pueden basar en las estructuras de datos lineales vistas: arreglos o listas enlazadas.

Esta vez debemos ser más cuidadosos en las operaciones: ya que sacamos y ponemos elementos en diferentes posiciones, debemos llevar registro de ambas. ¿Qué invariante deben cumplir estas posiciones?

Colas - implementación con arreglos

La siguiente estructura en C modela colas con arreglos:

```
typedef struct _AQueue {
    int    data[MAX];
    int    front, back;
} AQueue;
```

El entero `back` hace referencia a cuál es el último elemento ingresado, mientras que el entero `front` hace referencia elemento siguiente en ser quitado.

Colas - implementación con arreglos

La siguiente estructura en C modela colas con arreglos:

```
typedef struct _AQueue {
    int    data[MAX];
    int    front, back;
} AQueue;
```

El entero `back` hace referencia a cuál es el último elemento ingresado, mientras que el entero `front` hace referencia elemento siguiente en ser quitado.

Ejercicio: implementar `void aqueue_enqueue(AQueue *q, int val)` que inserta un elemento al comienzo de la cola.

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo?

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo?
`MAX`.

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo?
`MAX`.

¿Qué deficiencia puede tener la implementación anterior?

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo? `MAX`.

¿Qué deficiencia puede tener la implementación anterior?

Si cargo la cola de `MAX` elementos, luego los extraigo con `aqueue_dequeue`, ¿puedo volver a cargar elementos?

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo? `MAX`.

¿Qué deficiencia puede tener la implementación anterior?

Si cargo la cola de `MAX` elementos, luego los extraigo con `aqueue_dequeue`, ¿puedo volver a cargar elementos? No.

Colas - implementación con arreglos

¿Cuántos elementos puede tener la cola al mismo tiempo? `MAX`.

¿Qué deficiencia puede tener la implementación anterior?

Si cargo la cola de `MAX` elementos, luego los extraigo con `aqueue_dequeue`, ¿puedo volver a cargar elementos? No.

Para evitar este problema se debe pensar al arreglo como una estructura circular.

Colas - implementación con listas enlazadas

La siguiente estructura en C modela colas usando listas enlazadas:

```
typedef struct _SLQueueNode {
    int                     data;
    struct _SLQueueNode  *next;
} SLQueueNode;

typedef SLQueueNode *SLQueue;
```

Una cola es representada con un puntero al nodo final de la pila. El puntero `NULL` representa la cola vacia.

Colas - implementación con listas enlazadas

La siguiente estructura en C modela colas usando listas enlazadas:

```
typedef struct _SLQueueNode {
    int                     data;
    struct _SLQueueNode  *next;
} SLQueueNode;

typedef SLQueueNode *SLQueue;
```

Una cola es representada con un puntero al nodo final de la pila. El puntero `NULL` representa la cola vacia.

Ejercicio: implementar `void slqueue_enqueue(SLQueue *q, int val)` que inserta un elemento al final de la pila.

Generalidad - idea

Hasta ahora definimos listas de enteros, pilas de enteros, colas de enteros...

Generalidad - idea

Hasta ahora definimos listas de enteros, pilas de enteros, colas de enteros... ¿pero qué pasa si queremos listas de flotantes?

Generalidad - idea

Hasta ahora definimos listas de enteros, pilas de enteros, colas de enteros... ¿pero qué pasa si queremos listas de flotantes? ¿y si queremos listas de colas de enteros?

Generalidad - idea

Hasta ahora definimos listas de enteros, pilas de enteros, colas de enteros... ¿pero qué pasa si queremos listas de flotantes? ¿y si queremos listas de colas de enteros?

¡Necesitamos generalidad!

Generalidad - idea

Hasta ahora definimos listas de enteros, pilas de enteros, colas de enteros... ¿pero qué pasa si queremos listas de flotantes? ¿y si queremos listas de colas de enteros?

¡Necesitamos generalidad!

Listas de... cualquier cosa.
Pilas de... cualquier cosa.
Colas de... cualquier cosa.

Generalidad - C

¿Cómo representamos cualquier cosa en C? Nos inspiramos en `malloc`: utilizamos `void *`.

Con `void *` representamos un puntero a cualquier tipo de dato: `int`, `float`, `SList *`, etc.

Para poder derreferenciarlo hay que convertirlo ("castearlo") a un puntero del tipo de dato que corresponda.

Listas generales - C

Podemos re-pensar la estructura de listas enlazadas generales como:

```
typedef struct _SList {
  void *data;
  struct _SList *next;
} SList;
```

Listas generales - C

Podemos re-pensar la estructura de listas enlazadas generales como:

```
typedef struct _SList {
  void *data;
  struct _SList *next;
} SList;
```

Si en `data` ahora almacenamos un puntero a un entero (no un entero), debemos castearlo:

```
printf("%d", *((int *)l->data));
```

Generalidad - ejercicio

- Reimplementar `SList` usando `void *` en lugar de `int`.