

# Pila

El trabajo a realizar es el de una implementación de pila dinámica (es decir que pueda crecer o reducirse según la cantidad de elementos) que contenga punteros genéricos (`void*`).

En el adjunto en [el sitio de descargas](#) encontrarán el archivo `pila.h` que tienen que utilizar. En este archivo están definidas las primitivas que tendrán que implementar, con su correspondiente documentación. Todas las primitivas tienen que funcionar en *tiempo constante*.

Hay que escribir el archivo `pila.c`, con la implementación de la estructura de la pila y de cada una de las primitivas incluidas en el encabezado. Además de las primitivas, pueden tener funciones auxiliares, de uso interno, que no hace falta que estén declaradas dentro de `pila.h`. En `pila.h` se encuentran únicamente las primitivas que el usuario de la pila tiene que conocer.

En el archivo `pila.c` ya se les sugiere la siguiente estructura para la pila:

```
struct pila {
    void** datos;
    size_t cantidad; // Cantidad de elementos almacenados.
    size_t capacidad; // Capacidad del arreglo 'datos'.
};
```

Además de `pila.c`, tienen que entregar otro archivo `pruebas_pila.c`, que contenga las pruebas unitarias para verificar que la pila funciona correctamente, y que al ejecutarlo puede verificarse que todo funciona bien y no se pierde memoria. El archivo `pruebas_pila.c` debe contener una función llamada `pruebas_pila_estudiante()` que ejecute todas las pruebas. Se permite (y recomienda) usar funciones auxiliares.

Las pruebas deberán verificar que:

1. Se pueda crear y destruir correctamente la estructura.
2. Se puedan apilar elementos, que al desapilarlos se mantenga el invariante de pila.
3. *Prueba de volumen*: Se pueden apilar muchos elementos (1000, 10000 elementos, o el volumen que corresponda): hacer crecer la pila hasta un valor sabido mucho mayor que el tamaño inicial, y desapilar elementos hasta que esté vacía, comprobando que siempre cumpla el invariante. Recordar *no apilar siempre el mismo puntero*, validar que se cumpla siempre que el tope de la pila sea el correcto paso a paso, y que el nuevo tope después de cada `desapilar` también sea el correcto.
4. El apilamiento del elemento NULL es válido.
5. Condición de borde: comprobar que al desapilar hasta que está vacía hace que la pila se comporte como recién creada.
6. Condición de borde: las acciones de desapilar y `ver_tope` en una pila recién creada son inválidas.
7. Condición de borde: la acción de `esta_vacia` en una pila recién creada es verdadero.
8. Condición de borde: las acciones de desapilar y `ver_tope` en una pila a la que se le apiló y desapiló hasta estar vacía son inválidas.

Además de todos los casos no descriptos que ustedes crean necesarios.

Para compilar y verificar las pruebas:

1. Compilar todo el código:

```
gcc -g -std=c99 -Wall -Wconversion -Wno-sign-conversion -Werror -o pruebas *.c
```
2. Verificar que no pierden memoria:

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes ./pruebas
```

Al igual que en los casos anteriores, deberán entregar el código en papel, con el nombre y padrón, imprimiendo el archivo `pila.c` y el archivo `pruebas_pila.c`. Además, deben subir el código a la [página de entregas de la materia](#), con el código completo.

## Bibliografía recomendada

- Weiss, Mark Allen, “Data Structures and Algorithm Analysis”: 3.3. *The Stack ADT*.
- Cormen, Thomas H. “Introduction to Algorithms”: 10.1. *Stacks and queues*.