

Heap y Stack

En el nombre de Stuart Little! ¿Qué es un heap y el stack?

Son espacios de la memoria de ejecución del programa, cada uno con su uso particular e implicancias respecto a la forma de programar, siendo consciente de esta relación

Stack (Memoria)

- Se utiliza para almacenar variables locales y temporales.

Stack (Memoria)

- Se utiliza para almacenar variables locales y temporales.
- Es una estructura de datos de tipo LIFO (Last In, First Out).

Stack (Memoria)

- Se utiliza para almacenar variables locales y temporales.
- Es una estructura de datos de tipo LIFO (Last In, First Out).
- Es administrado automáticamente por el compilador y el sistema operativo.

Stack (Memoria)

- Se utiliza para almacenar variables locales y temporales.
- Es una estructura de datos de tipo LIFO (Last In, First Out).
- Es administrado automáticamente por el compilador y el sistema operativo.
- Es más rápido que el heap debido a que su acceso es más directo.

Stack (Memoria)

- Se utiliza para almacenar variables locales y temporales.
- Es una estructura de datos de tipo LIFO (Last In, First Out).
- Es administrado automáticamente por el compilador y el sistema operativo.
- Es más rápido que el heap debido a que su acceso es más directo.
- Es limitado en tamaño y puede causar errores de desbordamiento si se excede su capacidad.

Heap (Memoria)

- Se utiliza para almacenar datos dinámicos que se crean y eliminan en tiempo de ejecución.

Heap (Memoria)

- Se utiliza para almacenar datos dinámicos que se crean y eliminan en tiempo de ejecución.
- Es una estructura de datos de tipo FIFO (First In, First Out).

Heap (Memoria)


- Se utiliza para almacenar datos dinámicos que se crean y eliminan en tiempo de ejecución.
- Es una estructura de datos de tipo FIFO (First In, First Out).
- No es administrado automáticamente por el compilador y el sistema operativo.

Heap (Memoria)


- Se utiliza para almacenar datos dinámicos que se crean y eliminan en tiempo de ejecución.
- Es una estructura de datos de tipo FIFO (First In, First Out).
- No es administrado automáticamente por el compilador y el sistema operativo.
- Es más lento que el stack debido a que su acceso es menos directo.

Heap (Memoria)

- Se utiliza para almacenar datos dinámicos que se crean y eliminan en tiempo de ejecución.
- Es una estructura de datos de tipo FIFO (First In, First Out).
- No es administrado automáticamente por el compilador y el sistema operativo.
- Es más lento que el stack debido a que su acceso es menos directo.
- Es más flexible que el stack y puede crecer o disminuir según sea necesario.



```
int main(){  
    int N;  
  
    N = 10;  
  
    MyStruct stack_struct;  
    stack_struct.value = 10;  
  
    MyStruct* heap_struct = calloc(1, sizeof(MyStruct));  
    heap_struct->value = 10;  
  
    free(heap_struct);  
    return 0;  
}
```



```
int main(){  
    int N;
```

```
    N = 10;
```

← STACK

```
    MyStruct stack_struct;  
    stack_struct.value = 10;
```

← STACK

```
    MyStruct* heap_struct = calloc(1, sizeof(MyStruct));  
    heap_struct->value = 10;
```

← Heap

```
    free(heap_struct);  
    return 0;  
}
```

Nota Importante!

- Las variables están **SIEMPRE** en stack. Una variable es sobre lo que se opera en C.

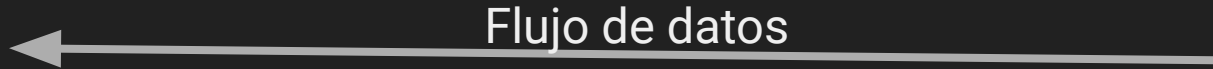
Por ejemplo. `int number`, `int* number`. Ambas son variables. Pero en la primera la variable almacena el número como tal. Y en la otra almacena una dirección de memoria (Puntero)

Las variables pueden almacenar cualquier cosa. Pero siempre hay que recordar que están en stack.

Flujo de allocs

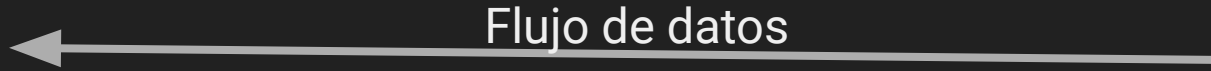
- Las funciones “malloc” y “calloc”. Son las encargadas de *solicitar* memoria en heap.

```
MyStruct* variable = calloc(1, sizeof(MyStruct));
```



Flujo de allocs

```
MyStruct* variable = calloc(1, sizeof(MyStruct));
```

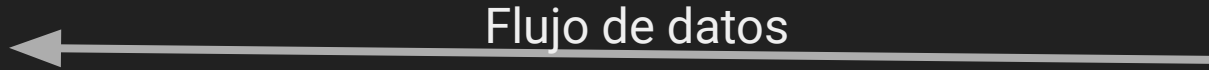


`calloc(1, sizeof(MyStruct)):`

- La función llama al SO y solicita `sizeof(MyStruct)` bytes de memoria
- El SO retorna una dirección de memoria (`Void*`)
- Finalmente `calloc` retorna esta dirección de memoria

Flujo de allocs

```
MyStruct* variable = calloc(1, sizeof(MyStruct));
```



MyStruct* variable

- El programa generará un dato en Stack de tamaño sizeof(MyStruct*). Y le asignará el acceso a través de variable
- Cuando calloc retorna una Dirección de Memoria. Esta es asignada a variable. Con tipo MyStruct*

Por ende. ¿Qué significa puntero?

- Es una dirección de memoria. Esta dirección permite acceder al dato que necesitamos.
- Si tenemos `Data* data`. Y accedemos a `data->value`.

El programa irá a la memoria a obtener el value para dicha dirección de memoria?

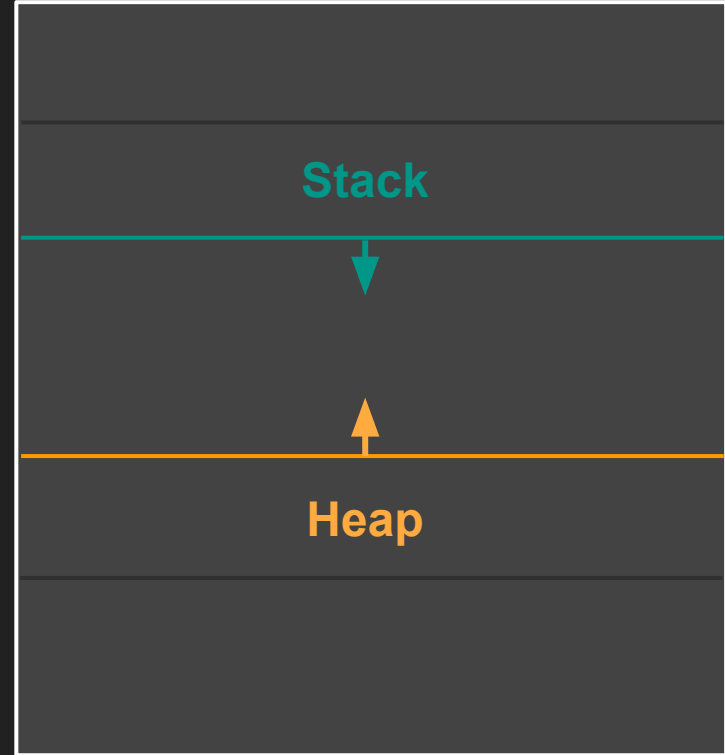
0x11...

Stack



Heap

0x02...



Finalmente. ¿Por qué usamos Heaps para almacenar structs?

Recordando las características del stack:

- Se utiliza para almacenar variables **locales** y **temporales**.
- Es **limitado en tamaño** y puede causar errores de desbordamiento si se excede su capacidad.

DEMO