



Pasaje por valor y por
referencia

Ámbito de una función

- Es el espacio en memoria al que puedo acceder desde una función. Esto significa que yo no puedo acceder a variables del `main` si yo estoy adentro de otra función.
- Cada vez que llamamos a una función se crea un nuevo ámbito.
- Cuando salimos de una función (ya sea por un `return` o porque llegamos al final) el ámbito y todo lo que había adentro se destruye.

C Tutor: Nuestro mejor amigo para entender este tema



- Ejecutemos el programa en este [link](#).
- Podemos ver cómo se van creando los ámbitos de cada función (los cuadrados) y como cada ámbito tiene sus variables adentro.
- En azul se marca el ámbito que se está ejecutando en cada momento, veamos que pasa con el ámbito del `main` cuando llamamos a la función.
- También se puede observar como se van destruyendo los ámbitos cuando ya no los necesitamos.

Veamos un ejemplo

- Hagamos una función que reciba dos variables e intercambie sus valores. Por ejemplo si $a=3$ y $b=5$ entonces la función debe dejarlos como $a=5$ y $b=3$.
- Imprimamos los valores antes y después de llamar a la función.
- Para verlo más de cerca pasemos el código a C Tutor y así vemos lo que está pasando.

Pasaje por valor

- Cuando le pasamos una variable a una función se crea una **copia** de esta dentro del nuevo ámbito.
- Como es una copia entonces cualquier cambio que le hagamos no va a modificar la variable original. Tampoco podemos acceder a la variable original porque está afuera del ámbito actual.
- Esto se llama pasaje por valor o por copia.



¿Qué pasa si quiero
modificar una variable que
vive afuera de mi ámbito?

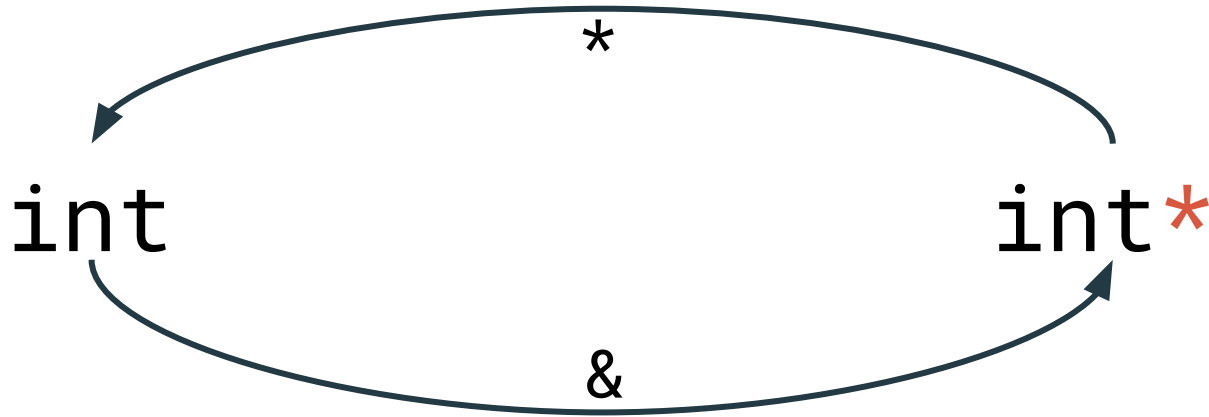
Punteros

- Un puntero es un tipo de dato que guarda direcciones en la memoria.
- Se llaman punteros porque apuntan a una variable y me dicen donde está.
- Si yo conozco la dirección de memoria de una variable entonces la puedo modificar esté donde esté.
- Para crear punteros en C utilizamos la siguiente notación:
 - `tipo_de_dato* nombre_del_puntero;`
 - `int* x; //Puntero a entero`
 - `char* respuesta; //Puntero a caracter`
 - `float* temperatura; //Puntero a float`

Con un nuevo tipo de dato vienen nuevos operadores

Para manejar punteros vamos a necesitar dos nuevos operadores (unarios):

- `&variable`: Devuelve la dirección de memoria de la variable. Osea que nos da un puntero a la variable.
- `*puntero`: Devuelve la variable a la que apunta el puntero. Recordemos que un puntero es una dirección de memoria así que necesitamos una forma de obtener la variable apuntada.
- Podemos verlas en acción en este [link](#).



- ¿Cuál es la diferencia entre `int` e `int*`?
- ¿Cual es la diferencia entre el asterisco negro y el rojo? ¿Qué función cumple cada uno?

Pasaje por referencia

- Si en vez de pasarle la variable a una función le pasamos un puntero (también llamado referencia) vamos a poder modificar la **variable original** por más de que esté afuera de nuestro ámbito.
- Arreglemos nuestra función de intercambio usando pasaje por referencia.
- Hagamos algunos ejemplos más así podemos ver bien como usarlo.