

Introducción a punteros



Algoritmos y Estructuras de Datos II
2014

Qué es un puntero?

- Un puntero es una variable que almacena la dirección de una posición en memoria.



Declaración de punteros


- Tengo una llave para un cuarto de un tipo dado, pero no apunta a ninguno en particular.
- Una variable puntero se declara usando un tipo de datos seguido por un asterisco:

```
int *ptr = NULL;
```

- El valor NULL corresponde al neutro de las direcciones, no es una dirección válida.
- Eventualmente, este puntero debe contener una dirección correspondiente a un valor de tipo entero.

Cómo leer una declaración

- Una forma fácil es hacerlo de atrás hacia adelante. Si tenemos:

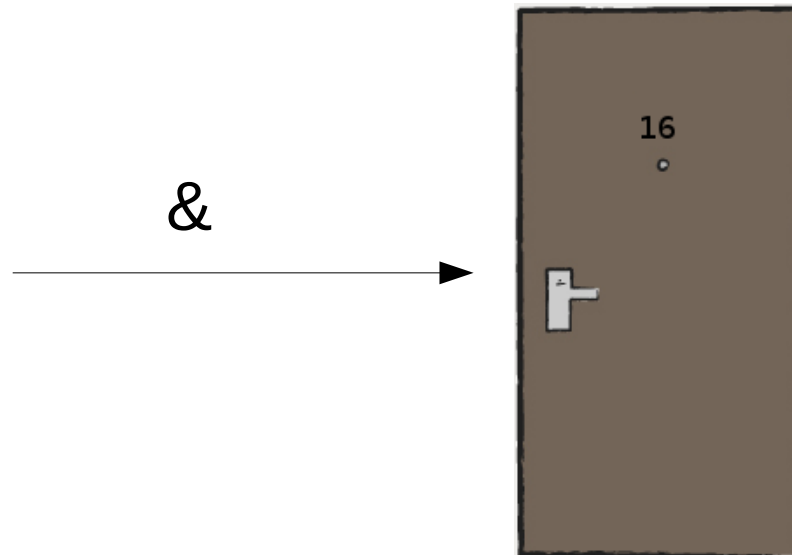
`int *ptr;`


1. Nombre de variable (ptr)
2. Es un puntero (*)
3. Apuntando a un entero (int)

Obtener una dirección

- El operador & devuelve la dirección de memoria de una variable:

```
int num = 5;  
int *ptr = &num;
```



Qué tenemos hasta acá

```
int num = 5;
int *null_ptr = NULL;
int *ptr = &num;

printf("num: %d, address: %p\n", num, (void *)&num);
printf("ptr: %p, address: %p\n", ptr, (void *)&ptr);
printf("null_ptr: %p, address: %p\n",
      (void *)null_ptr, (void *)&null_ptr);
```

num: 5, address: 0x7fffb4174e0c

ptr: 0x7fffb4174e0c, address: 0x7fffb4174e18

null_ptr: (nil), address: 0x7fffb4174e10

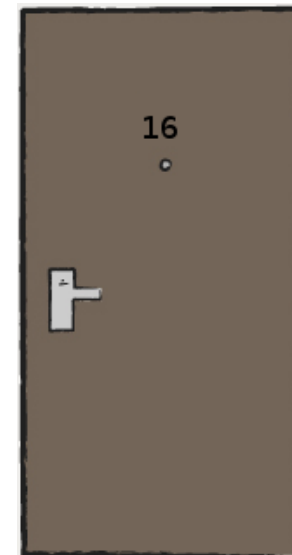
Seguir una dirección

- Es lo que se conoce como desreferenciar un puntero.
- Se hace mediante el operador *:

```
printf("%i\n", *ptr); // 5  
*ptr = 8;  
printf("%i\n", *ptr); // 8
```



*



Memoria dinámica

- Tengo mi llave, quiero que esté asociada a un cuarto del tipo que corresponde.

```
int *ptr = NULL;
```

- Cómo pido un cuarto?

Workflow usual

- Trabajando con memoria dinámica, los pasos habituales son:
 1. Usar una función para pedir memoria
(el check-in)
 2. Usar la memoria
(uso de la habitación)
 3. Liberarla cuando se deja de usar
(el check-out)

malloc

- No me limpiaron la habitación!

```
int *ptr = malloc(sizeof(int));  
printf("%i\n", *ptr);
```

- `malloc` recibe como argumento el tamaño del bloque de memoria que se solicita.
- Devuelve la dirección del bloque asignado en caso de éxito, o `NULL` si no hubiera memoria disponible para el bloque solicitado.

calloc

- La habitación está limpia.

```
int *ptr = calloc(1, sizeof(int));  
printf("%i\n", *ptr);
```

- `calloc` recibe como argumentos la cantidad de bloques de memoria que se solicitan, y el tamaño de cada bloque.
- Devuelve la dirección del bloque asignado en caso de éxito, o `NULL` si no hubiera memoria disponible para el bloque solicitado.

free

- Haciendo el checkout.

```
free(ptr);  
ptr = NULL;
```

- `free` recibe como argumento una variable de tipo puntero, y marca el bloque de memoria como liberado (no pone el bloque a cero ni cambia la dirección en la variable dada).
- Es buena práctica después del `free` de un puntero, setear el valor del mismo a `NULL`.

Violación de segmento

- Entrando al cuarto equivocado!

```
int *ptr = (int *) 127;  
printf("ptr value: %i, ptr: %p\n",  
      *ptr, (void *)ptr);
```

- Al intentar acceder a una dirección arbitraria, que no me fue otorgada por el sistema operativo, se produce una violación de segmento.
- Casos similares: acceder a un índice no válido de un arreglo, o intentar seguir un puntero cuyo bloque de memoria fue liberado.