Decaimientos de arrays

Guillermo Hernández

2023-12-09

Versión en línea: https://www.dih5.es/address-str-literal.html.

¿Cuál será la salida de los siguientes `printf`?

```c
#include <stdio.h>

int main() {
    char s[100]= "hola";

    printf("%lu\n", sizeof(s));
    printf("%lu\n", sizeof(s+1));
    printf("%lu\n", sizeof(s+0));
    printf("%lu\n", sizeof(+s));

    return 0;
}
```

Veamos la respuesta uno a uno:

- `s` es un array de 100 `char`, así que será 100.
- En `s + 1`, `s` se comporta como un puntero constante a `char` que apunta al principio del array. Al evaluar `s + 1` obtenemos un nuevo puntero a `char` que apunta a la posición del array; como es un puntero, lo que resultará de `sizeof` es el tamaño de un puntero (8 bytes en un sistema de 64 bits).
- En `s + 0` ocurre exactamente lo mismo… aunque no cambie la posición, si ocurre el **decaimiento** de `s` en un puntero, por lo que veremos el mismo número que en la anterior.
- La cuarta tiene trampa, es una instrucción ilegal en C (C11 6.5.3.3/1: «The operand of the unary + or - operator shall have arithmetic type»). Sin embargo, en C++ sí sería legal, daría lugar al mismo tipo de resultado que las dos anteriores.

En resumen… ¿no habría salidas, porque el cuarto printf es ilegal? Pue la respuesta correcta en realidad es que **depende**. Yo no he dicho que eso fuera un código C, podría ser perfectamente C++…

Recapitulando ahora lo que teníamos que saber para enfrentarnos a la pregunta: el **decaimiento** consiste en que el nombre de un array se comporta como un puntero constante a su tipo base que apunta a su primer elemento en la mayoría de las circunstancias, pero existen las siguientes excepciones:

- Con el operador dirección de (`&`).
- Con el operador `sizeof` y otros similares como `typeof` (C23).
- Cuando el array corresponde a un literal cadena usado en una inicialización. Por ejemplo, en `char s[]="foo";` ocurre que `"foo"` es un legítimamente un array y no un puntero.

Pues *sí*, el acceso a un elemento de un array con el operador subíndice (corchete) no está entre las excepciones. Recordemos que `m[i]` es exactamente equivalente a `*(m+i)`, siempre hemos estado haciendo operaciones con punteros. *Wait, it's all pointers? Always has been.*

Pero aún quedan más cosas sutiles por revisar, así que continuemos con otra pregunta. Si tenemos la siguiente función:

```
void f(char s[75]){
    printf("%lu\n", sizeof(s));
}
```

y la llamamos desde el *main* anterior, pasándole `s`, ¿qué veremos?

La respuesta correcta vuelve a ser la del tamaño de un puntero (ni 75 ni 100). A pesar de que en la definición se muestre como un array, al ser el array un argumento en la función, también decae automáticamente a puntero. El compilador nos avisará con algo como `warning: ‘sizeof’ on array function parameter ‘s’ will return size of ‘char *’ [-Wsizeof-array-argument]`.

Todavía nos queda otro nivel. Hagamos lo mismo en dos dimensiones:

```
#include <stdio.h>

void f2(char m[25][50]){
    printf("%lu\n", sizeof(m));
    printf("%lu\n", sizeof(m[0]));
}

int main() {
    char n[75][100];
    f2(n);

    return 0;
}
```

¿Cuáles son los números que veremos ahora?

- En primer lugar, `m` decae a un puntero a su tipo base; como es un puntero, la primera respuesta es la del tamaño de un puntero (8 bytes en un sistema de 64 bits). Ahora bien, ¿cuál es el tipo exacto? `m` apunta a filas de 50 elementos, esto es, es un `char (*)[50]`.
- En segundo lugar, como hemos indireccionado el puntero anterior, tenemos un `char [50]`, luego el tamaño es 50.

Si tenemos un array multidimensional como parámetro, solo la primera dimensión decae. En realidad es bastante cabal: un array de n dimensiones se está representando como una sucesión en memoria de arrays de n-1 dimensiones de igual tamaño, lo cual es equivalente a apuntar a la memoria en bloques del tamaño de dichos arrays. Por eso son necesarias las medidas de todas las dimensiones excepto de la primera; y, aunque proporcionemos esta, será irrelevante a nivel de aritmética.

Cuestión aparte sería la correspondencia entre las operaciones con el parámetro `m` y con la matriz `n`. Si el lector tiene dudas, se le deja como ejercicio rellenarla en la función y mostrarla en el main.