Arrays en el lenguaje C

Introducción

Una posible definición de *array* sería:

> Un conjunto de datos del mismo tipo, identificados por el mismo nombre, y que se pueden distinguir mediante un número de índice.

Pero ¿qué quiere decir esto y para qué lo queremos?. Pues bien, supongamos que queremos almacenar la temperatura media de cada hora del día y la temperatura promedio del día. Con lo que sabemos hasta ahora podríamos hacer algo así:

```
#include <stdio.h>

main()
{
        /* Declaramos 24 variables, una para cada hora del dia */
        int temp1,  temp2,  temp3,  temp4,  temp5,  temp6,  temp7,  temp8;
        int temp9,  temp10, temp11, temp12, temp13, temp14, temp15, temp16;
        int temp17, temp18, temp19, temp20, temp21, temp22, temp23, temp0;
        float media;

        /* Asignamos el valor de cada una */
        printf( "Introduzca las temperaturas desde las 0 hasta las 23
                separadas por un espacio: " );
        scanf( "%i %i %i ... %i", &temp0, &temp1, &temp2, ... &temp23 );

        media = ( temp0 + temp1 + temp2 + temp3 + temp4 + ... + temp23 ) / 24;
        printf( "\nLa temperatura media es %f\n", media );
}
```

Los puntos ... se utilizan por brevedad en el ejemplo; no constituyen una expresión válida en C.

Observamos que hay que realizar un notable trabajo repetitivo de escritura de código. Precisamente es aquí donde son de utilidad los *arrays*. Vamos a repetir el programa anterior con un *array*:

```
#include <stdio.h>

main()
{
        int temp[24]; /* Con esto declaramos las 24 variables */
        float media;
        int hora;

        /* Ahora damos valor a cada una */
        for( hora=0; hora<24; hora++ ) {
                printf( "Temperatura de las %i: ", hora );
                scanf( "%i", &temp[hora] );
                media += temp[hora];
        }
        media = media / 24;

        printf( "\nLa temperatura media es %f\n", media );
}
```

El programa resulta más rápido de escribir y más cómodo para el usuario que el anterior.

Como ya hemos comentado, cuando declaramos una variable lo que estamos haciendo es reservar una zona de la memoria para ella. Cuando declaramos el *array* de este ejemplo es reservar espacio en memoria para 24 variables de tipo `int`. El tamaño del *array* (24) lo indicamos entre corchetes al definirlo. Esta es la parte de la definición que dice: *Un array es un conjunto de datos del mismo tipo identificados por el mismo nombre*.

La parte final de la definición dice: *y se distinguen mediante el índice*. En el ejemplo recorremos la matriz mediante un bucle `for` y vamos dando valores a los distintos elementos de la matriz. Para indicar a qué elemento nos referimos usamos un número entre corchetes (en este caso la variable hora), este número es lo que se llama **índice** del *array*.

En C, el primer elemento de una matriz tiene el índice 0, el segundo tiene el 1, y así sucesivamente. De modo que si queremos dar un valor al elemento 4 (índice 3) haremos:

```
temp[ 3 ] = 20;
```

No hay que confundirse. En la declaración del *array* el número entre corchetes es el número total de elementos; en cambio, cuando usamos la matriz, el número entre corchetes es el índice.

Declaración de un *array*

La forma general de declarar un array es la siguiente:

```
tipo_de_dato identificador_del_array[ dimensión ];
```

donde:

- el *`tipo_de_dato`* es uno de los tipos de datos conocidos (`int`, `char`, `float`, etc.). En el ejemplo era `int`.
- El *`identificador_del_array`* es el nombre que le damos (en el ejemplo era `temp`).
- La *`dimensión`* es el número de elementos que tiene el *array*.

Como se ha indicado antes, al declarar un *array* reservamos en memoria tantas variables del *`tipo_de_dato`* como las indicada en *`dimensión`*.

Hemos visto en el ejemplo que tenemos que indicar en varios sitios el tamaño del *array*: en la declaración, en el bucle `for` y al calcular la media. Este es un programa pequeño; en un programa mayor probablemente habrá que escribirlo muchas más veces. Si en un momento dado queremos cambiar la dimensión del *array* tendremos que cambiar todos. Si nos equivocamos al escribir el tamaño (ponemos 25 en vez de 24) cometeremos un error y puede que no nos demos cuenta. Por eso es mejor usar una constante simbólica, por ejemplo `NUM_HORAS`:

```
#include <stdio.h>

#define NUM_HORAS       24

main()
{
        int temp[NUM_HORAS]; /* Con esto declaramos las 24 variables */
        float media;
        int hora;

        /* Ahora damos valor a cada una */
        for( hora=0; hora<NUM_HORAS; hora++ ) {
                printf( "Temperatura de las %i: ", hora );
                scanf( "%i", &temp[hora] );
                media += temp[hora];
        }
        media = media / NUM_HORAS;

        printf( "\nLa temperatura media es %f\n", media );
}
```

Ahora con sólo cambiar el valor de NUM_HORAS una vez lo estaremos haciendo en todo el programa.

Inicialización de un *array*

En C se pueden inicializar los *arrays* al declararlos igual que se hace con las variables. Es decir, por ejemplo:

```
int hojas = 34;
```

Así, con *arrays* se puede hacer:

```
int temp[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22,
                 21, 20, 18, 17, 16, 17, 15, 14, 14, 14, 13, 12 };
```

Ahora el elemento 0, el primero, es decir, `temperaturas[0]`, valdrá 15. El elemento 1, el segundo, valdrá 18, y así con todos. Vamos a ver un ejemplo:

```
#include <stdio.h>

main()
{
        int hora;
        int temp[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22,
                         21, 20, 18, 17, 16, 17, 15, 14, 14, 14, 13, 12 };

        for( hora=0; hora<24; hora++ ) {
                printf( "La temperatura a las %i era de %i grados.\n",
                        hora, temp[hora] );
        }
}
```

Pero al introducir los datos no es difícil olvidarse alguno. Hemos indicado al compilador que nos reserve memoria para un *array* de 24 elementos de tipo `int`. ¿Qué ocurre si introducimos menos de los reservados? No sucede nada especial, sólo que los elementos que falten se inicializarán a 0.

```
#include <stdio.h>

main()
{
        int hora;
        int temp[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22,
                         21, 20, 18, 17, 16, 17, 15, 14, 14 };
                         /* Faltan los tres últimos elementos */

        for( hora=0; hora<24; hora++ ) {
                printf( "La temperatura a las %i era de %i grados.\n",
                        hora, temp[hora] );
        }
}
```

El resultado será:

```
La temperatura a las 0 era de 15 grados.
La temperatura a las 1 era de 18 grados.
La temperatura a las 2 era de 20 grados.
La temperatura a las 3 era de 23 grados.
...
La temperatura a las 17 era de 17 grados.
La temperatura a las 18 era de 15 grados.
La temperatura a las 19 era de 14 grados.
La temperatura a las 20 era de 14 grados.
La temperatura a las 21 era de 0 grados.
La temperatura a las 22 era de 0 grados.
La temperatura a las 23 era de 0 grados.
```

Vemos que los últimos 3 elementos son nulos, precisamente aquellos a los que no hemos dado valores iniciales. El compilador no nos avisa de que hemos introducido menos datos de los reservados.

NOTA: Para recorrer del elemento 0 al 23 (24 elementos)hacemos: `for(hora=0; hora<24; hora++)`. La condición es que hora sea menos de 24. También podíamos haber hecho que `hora!=24`.

Ahora vamos a ver el caso contrario, introducimos más datos de los reservados. Supongamos 25 en vez de 24. Si hacemos esto, dependiendo del compilador obtendremos un error o, al menos, un *warning* (aviso). En unos compiladores el programa se creará y en otros no, pero aún así nos avisa del fallo. Debe indicarse que estamos intentando guardar un dato de más, no hemos reservado memoria para él.

Si la matriz debe tener una longitud determinada, usamos el método descrito (definiendo el tamaño de la matriz) para definir el número de elementos. En nuestro caso era conveniente, porque los días siempre tienen 24 horas. En otros casos podemos usar un método alternativo: dejar al compilador que cuente los elementos que hemos introducido y reserve espacio para ellos:

```
#include <stdio.h>

main()
{
        int hora;
        int temp[] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24,
                       22, 21, 20, 18, 17, 16, 17, 15, 14, 14 };
                       /* Faltan los tres últimos elementos */

        for( hora=0; hora<24; hora++ ) {
                printf( "La temperatura a las %i era de %i grados.\n",
                        hora, temp[hora] );
        }
}
```

Vemos que no hemos especificado la dimensión del *array* `temp`. Hemos dejado los corchetes en blanco. El compilador contará los elementos que hemos puesto entre llaves y reservará espacio para ellos. De esta forma siempre habrá el espacio necesario, ni más ni menos. La pega es que si ponemos más de los que queríamos no nos daremos cuenta.

Para saber en este caso cuantos elementos tiene la matriz podemos usar el operador `sizeof`. Dividimos el tamaño de la matriz entre el tamaño de sus elementos y tenemos el número de elementos.

```
#include <stdio.h>

main()
{
        int hora;
        int num_elementos;
        int temp[] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24,
                       22, 21, 20, 18, 17, 16, 17, 15, 14, 14 };
                       /* Faltan los tres últimos elementos */

        num_elementos = sizeof(temp) / sizeof(int);
        for( hora=0; hora<24; hora++ ) {
                printf( "La temperatura a las %i era de %i grados.\n",
                        hora, temp[hora] );
        }
        printf( "Han sido %i elementos.\n" , num_elementos );
}
```

Recorrido de un array

En las secciones anteriores veíamos un ejemplo que mostraba todos los datos de un *array*. Veíamos también lo que pasaba si introducíamos más o menos elementos al inicializar la matriz. Ahora vamos a ver qué pasa si intentamos imprimir más elementos de los que hay en la matriz; en este caso vamos a intentar imprimir 28 elementos cuando sólo hay 24:

```
#include <stdio.h>

main()
{
        int hora;
        int temp[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22,
                         21, 20, 18, 17, 16, 17, 15, 14, 14, 14, 13, 12 };

        for( hora=0; hora<28; hora++ ) {
                printf( "La temperatura a las %i era de %i grados.\n",
                        hora, temp[hora] );
        }
}
```

Lo que se obtiene en mi ordenador es:

```
La temperatura a las 0 era de 15 grados.
...
La temperatura a las 23 era de 12 grados.
La temperatura a las 24 era de 24 grados.
La temperatura a las 25 era de 3424248 grados.
La temperatura a las 26 era de 7042 grados.
La temperatura a las 27 era de 1 grados.
```

Vemos que a partir del elemento 24 (incluído) tenemos resultados extraños. Esto es porque nos hemos salido de los límites del *array* e intenta acceder al elemento `temp[24]` y sucesivos que no existen. Así que nos muestra el contenido de la memoria que está justo detrás de `temp[23]` (esto es lo más probable), que puede ser cualquiera. Al contrario que otros lenguajes, C no comprueba los límites de los *array*. Este programa no da error al compilar ni al ejecutar, tan sólo devuelve resultados extraños. Tampoco bloqueará el sistema porque no estamos escribiendo en la memoria sino leyendo de ella.

Otra cosa muy diferente es introducir datos en elementos que no existen. Veamos un ejemplo **(se recomienda no ejecutarlo)**:

```
#include <stdio.h>

main()
{
        int temp[24];
        float media;
        int hora;

        /* Ahora damos valor a cada una */
        for( hora=0; hora<28; hora++ ) {
                printf( "Temperatura de las %i: ", hora );
                scanf( "%i", &temp[hora] );
                media += temp[hora];
        }
        media = media / 24;
        printf( "\nLa temperatura media es %f\n", media );
}
```

Lo más probable es que al ejecutar este código el ordenador se bloqueado y deba reiniciarlo. El problema ahora es que estamos intentando escribir en el elemento `temp[24]`, que no existe y puede ser un lugar cualquiera de la memoria. Como consecuencia de esto podemos estar cambiando algún programa o dato de la memoria que no debemos y el sistema se detiene.