Tema 6: Arrays y Cadenas

Introducción a la programación I

Marcos Novalbos
Elena García Gamella

Índice

1. Arrays

- Un array es una variable estructurada formada por una secuencia de datos del mismo tipo.
- Los datos se llaman elementos del array o componentes y se enumeran consecutivamente 0, 1, 2, 3 … (valores índice o subíndice del array)
- En general, el valor i-esimo ocupa la posición i-1.
- Si el array “a” tiene n elementos, sus nombres son a[0], a[1], a[2], …, a[n-1].
- El tipo de elementos almacenados en el array puede ser cualquier tipo de dato de C, incluyendo estructuras definidas por el usuario.

1. Arrays. Definición

- Un array se declara de modo similar a otros tipos de datos, excepto que se debe indicar al compilador el tamaño o longitud del array.
- Para indicar al compilador el tamaño o longitud del array se debe hacer seguir al nombre, el tamaño encerrado entre corchetes `[ ]`.
- La sintaxis para declarar un array de una dimensión es:

```
tipo nombreArray [numero de elementos]
int alturas[10];
```

- En el array `int alturas[10]` los índices validos son `alturas[0]`, `alturas[1]`, ..., `alturas[9]`. Pero si se pone `alturas[15]` el compilador no genera error y el resultado puede ser impredecible.

Es decir, C no comprueba que los índices del array están dentro del rango definido

1. Arrays. Inicialización.

- Inicialización de Arrays
  - Se deben asignar valores a los elementos del array antes de utilizarlos, tal y como se asignan valores a variables. Cuando declaramos un array no podemos asegurar el valor que tendrán sus elementos.
  - Para asignar valores a cada elemento de un array de enteros p, se puede escribir:

```
p[0] = 10; /* Asigna el valor 10 al elemento 0 del array */
p[1] = 20; /* Asigna el valor 20 al elemento 1 del array */
```

  - Este método no es práctico para arrays de muchos elementos.
  - El método utilizado es inicializar el array completo en una instrucción y hacerlo en el momento de su declaración.
  - Cuando se inicializa un array, el tamaño del array se puede determinar automáticamente por las constantes de inicialización.
  - Estas constantes de separan por comas (`,`) y se encierran entre llaves (`{}`)

```
int num[6] = {10,20,30,40,50,60};
int num2[ ] = {1,2,3};
char ch[ ] = {'H','o','l','a'};
```

1. Arrays

- Ejemplos:
  - Crear un array de 5 posiciones de tipo entero, relleno con los números del 1 al 5:
  - Asignar en la segunda posición, el valor 10
  - Mostrar el contenido de la quinta posición:
  - Mostrar el contenido de todo el array:
  - Pedirle al usuario un dato para rellenar en la tercera posición del array:

1. Arrays

- Ejemplos:
  - Crear un array de 5 posiciones de tipo entero, relleno con los números del 1 al 5:
    ```
    int arr1[5]={1,2,3,4,5}
    ```
  - Asignar en la segunda posición, el valor 10
    ```
    arr1[1]=10; // se empieza a contar posiciones en el índice 0
    ```
  - Mostrar el contenido de la quinta posición:
    ```
    printf("%d\n",arr1[4]);
    ```
  - Mostrar el contenido de todo el array:
    ```
    int contador=0;

    for(contador=0;contador<(sizeof(arr)/sizeof(int));contador++)
      printf("posición %d contiene %d\n", arr1[contador]);
    ```
  - Pedirle al usuario un dato para rellenar en la tercera posición del array:
    ```
    scanf("%d\n", &arr1[2]);
    ```

1. Arrays

- Los elementos de los arrays se almacenan en bloques contiguos de memoria.
- En los programas se pueden referencia elementos del array utilizando formulas o expresiones enteras para los subíndices.
- Si a es un array de tipo float y `a[0]` ocupa la dirección `x`, el elemento `a[i]` ocupa la dirección de memoria `x+(i)*4` (un float ocupa 4 bytes)
- El operador `sizeof` devuelve el número de bytes necesarios para contener la variable que se pasa como argumento.
- Si se usa `sizeof` para solicitar el tamaño de un array, esta función devuelve el número de bytes reservados para el array completo
- Conociendo el tipo de dato almacenado en el array y su tamaño, se obtiene la longitud (dimensión) del array, mediante

```
sizeof (array) / numero bytes tipo dato.
```

1. Arrays

- Aclaraciones sobre "sizeof"
  - OJO: **Solo funciona con arrays estáticos (los que estamos usando ahora)**, no con arrays dinámicos/punteros. Se desaconseja usar "sizeof" sobre nombres de variables que sean de tipo "puntero", a menos que se quiera averiguar el tamaño del puntero (8 bytes en arquitecturas de 64 bit).
  - Esta es una función que nos sirve para averiguar el tamaño en bytes de una variable o zona de memoria.
  - Es una función propia del compilador, averigua los tamaños en tiempo de compilación, consultando **los tipos de datos** de las variables pasadas por parámetros.

1. Arrays

- Aclaraciones sobre "sizeof"
  - Ejemplo:

```
int var1=0;
printf("El tamaño de var1 es: %d\n", sizeof(var1));
```

    - --> Se intercambia por "sizeof(int)", devuelve 4 bytes

```
int array1[10];
printf("El tamaño de array1 es: %d\n", sizeof(array1));
```

    - -->Se intercambia por sizeof "(int[10])", devuelve 40

1. Arrays

Ejercicio 1

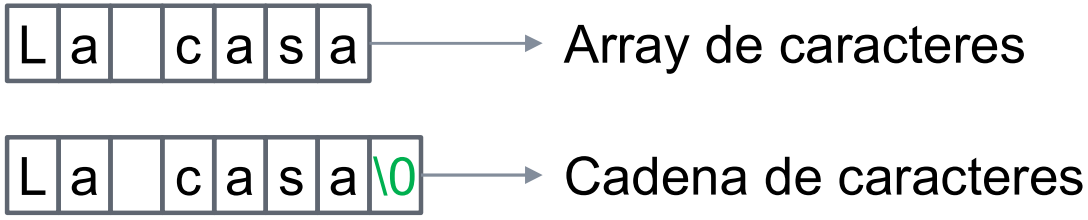
- 1.1 Crear un programa que rellene un array de 10 posiciones con números aleatorios entre el 0 y el 30 (pueden estar repetidos). Mostrar el resultado por pantalla.

- 1.2 Al programa anterior, añadir lo siguiente: Después de mostrar el resultado, pedirá un número entre el 0 y el 30 al usuario. Buscar si ese número está en el array, y mostrar los siguientes mensajes: "Número encontrado" en caso de encontrarlo. Mostrar "No encontrado" en caso contrario.

- 1.3 Modificar el anterior, para que vuelva a pedir números al usuario si no se ha encontrado. Sólo saldrá cuando haya encontrado el número introducido por el usuario.

2 Cadenas en C. Concepto

- El lenguaje C no tiene datos predefinidos tipo cadena.
- En su lugar, C manipula cadenas mediante arrays de caracteres que terminan con el carácter ASCII nulo (‘\0’).
- Una cadena se considera como un array unidimensional de tipo char o unsigned char.



El número total de caracteres de una cadena en C es siempre igual a la longitud del array más 1.

2 Cadenas en C. Inicialización

- Una cadena no se puede inicializar fuera de la declaración.
- La razón es que un identificador de cadena, como cualquier identificador de array, se trata como un valor de dirección, como un puntero constante.
- Puede inicializar un array de caracteres con un literal de cadena

```
char letras[] = "abc";
```

- Se inicializa letras como un array de caracteres de cuatro elementos. El cuarto elemento es el carácter null, ('\0') que finaliza todos los literales de cadena
- Esta inicialización es equivalente a las siguientes:

```
char letras[] = {'a','b','c','\0'};

char letras[4] = "abc";
```

2 Cadenas en C. Lectura de cadenas

¿Qué opciones tenemos para realizar una lectura de cadenas?

Realizad un programa, pidiendo al usuario que introduzca tres cadenas con nombre y apellido con la función que estiméis mejor y luego que escriba cada una de ellas en una línea distinta.

2 Cadenas en C. Lectura de cadenas

- Solicita al usuario su nombre y apellidos. Almacénalo en dos cadenas de caracteres de 20 elementos. Imprime ambas cadenas por pantalla.

2 Cadenas en C. Lectura de cadenas

- Solicita al usuario su nombre y apellido. Almacénalo en dos cadenas de caracteres de 20 elementos. Imprime ambas cadenas por pantalla.

```
#include <stdio.h>
#define TAM 20

void main(){

     char nombre [TAM];
     char apellido [TAM];

     printf ("Introduce el nombre: \n ");
     scanf ("%s",nombre);
     printf ("El nombre es: %s\n",nombre);

     printf ("Introduce el apellido: \n ");
     scanf ("%s",apellido);
     printf ("El apellido es: %s\n",apellido);
}
```

2 Cadenas en C. Lectura de cadenas. `scanf()`

- Cuando se utiliza la consola, el programa no lee directamente el texto tal cual lo introduce el usuario si no que éste se almacena en una tabla intermedia que llamaremos buffer. Cada vez que el usuario pulsa el retorno de carro, se llena el buffer con la línea introducida (incluyendo el carácter '\n')
- La lectura usual de datos se realiza con la función **scanf()**
- Cuando se aplica a datos cadena el código de formato es %s (string).
- **La función da por terminada la cadena cuando encuentra un espacio en blanco y el fin de línea.**
- Sin embargo "%s" plantea dos problemas:
  - Leerá caracteres hasta encontrarse con el primer espacio en blanco y ahí se detendrá (dejando el resto de los caracteres que hubiera tecleado el usuario sin leer, a la espera del siguiente scanf()). Esto no es normalmente lo que se desea, ya que sólo permitiría leer una palabra si el usuario escribe varias separadas por espacios.
  - Es posible que el usuario escriba más caracteres de los que podemos guardar en la variable texto. scanf() no comprueba los límites de esa variable (en realidad no puede hacerlo, aunque quisiera, porque todo lo que recibe es la dirección donde comienza el array texto, pero no su tamaño). Si el usuario escribiera más caracteres (sin espacios) de los que caben, el resto sobre escribirían otras partes de la memoria del programa, con el riesgo de seguridad que ello conlleva (problemas por buffer overrun).

2 Cadenas en C. Lectura de cadenas `scanf()`

- La solución pasa por :

```
char texto[10001];
scanf("%10000[^\n]s", texto);
getchar();
```

- Esta cadena de formato sigue esperando un string, por el especificador de formato “s” del final, pero:
  - El número (10000) sería el máximo de caracteres a leer. Eso evita el posible buffer overrun.
  - El [^\n] indica la categoría de caracteres a admitir, y es una expresión regular que significa "todo lo que no sea el carácter \n".

2 Cadenas en C. Lectura de cadenas `scanf()`

- Por tanto, esa cadena de formato leería una línea completa, con espacios y todo, deteniéndose en cuanto encuentre un \n, o cuando haya leído 10000 caracteres (lo que ocurra antes).

- El \n queda sin leer, a la espera de la próxima instrucción que lea algo de la entrada estándar.

- Es por ese \n que se debe hacer luego un getchar(), para "consumirlo", pues de lo contrario sería encontrado por el próximo scanf() que se ejecutara, lo que le confundiría y consideraría que la entrada es una línea en blanco.

int putchar(int char)

2. Cadenas en C. Lectura de cadenas `getchar()`

- La función **`getchar()`** se utiliza para leer carácter a carácter.
- La llamada a `getchar()` devuelve el carácter siguiente del flujo de entrada de `stdin`.

```
int getchar(void)
```

- En caso de error, o de encontrar el fin de archivo devuelve EOF (macro definida en stdio.h).
- La función **`putchar()`** se utiliza para escribir en la salida stdout carácter a carácter.
- El carácter que se escribe es el trasmitido como argumento.

```
int putchar(int char)
```

2 Cadenas en C. Lectura de cadenas

- Solicita al usuario su nombre y apellido utilizando getchar(). Almacénalo en dos cadenas de caracteres de 20 elementos. Imprime ambas cadenas por pantalla con putchar().

2 Cadenas en C. Lectura de cadenas

- Solicita al usuario su nombre y apellido utilizando getchar(). Almacénalo en dos cadenas de caracteres de 20 elementos. Imprime ambas cadenas por pantalla con putchar().

```
#include <stdio.h>
#define TAM 20

void main(){

      char nombre [TAM];
      char apellido [TAM];
      char newChar;
      int strSize=0;
      int i;
      while((newChar = getchar()) != '\n') {
            nombre[strSize] = newChar;
            strSize++;
      }
      nombre[strSize] = '\0';

      printf ("El nombre es: %s\n",nombre);

      for (i=0 ; i < strSize ; i++){
            putchar (nombre[i]);
            putchar ('\n');
      }
}
```

2. Cadenas en C. Lectura de cadenas `gets()` y `fgets()`

- Mientras que scanf() puede leer distintos tipos de datos, `gets ()` y `fgets()` solo leen cadenas de caracteres introducidas por el usuario
- Se puede utilizar la función **`gets(),`** que permite leer la cadena completa incluyendo cualquier espacio en blanco, hasta el carácter fin de línea.
- La función asigna la cadena al argumento transmitido a la función, que será un array de caracteres o un puntero (char *) a memoria libre, con un número de elementos suficiente para guardar la cadena leída.
- Si ha habido un error en la lectura de la cadena, devuelve NULL.
- `gets()` presenta el problema que no puede controlar el número de caracteres que introduce el usuario pudiendo ocurrir que se copien en la cadena más caracteres que los permitidos por su tamaño máximo.

2 Cadenas en C. Lectura de cadenas `gets()` y `fgets()`

- El prototipo de la función es:

```
char *gets(char *cadena);
```

- Esta función lee caracteres desde la entrada estándar (**stream stdin)**, en el array apuntado por **cadena**, hasta que se encuentre un final de fichero (EOF) o un carácter de línea nueva es leído. **Cualquier carácter de línea nueva es descartado, y un carácter nulo es escrito inmediatamente después del último carácter leído en el array**.
- El puntero “cadena” debe tener espacio suficiente para almacenar la cadena leída.
- La función gets retorna el puntero “cadena” si la lectura es realizada con éxito o NULL en caso contrario.

2 Cadenas en C. Lectura de cadenas `gets()` y `fgets()`

- La alternativa segura de gets() es fgets() que si permite establecer el máximo de caracteres que pueden leerse.
- El prototipo de la función es:

```
char *fgets(char *cadena, int n, FILE *stream);
```

- Esta función lee como máximo uno menos que el número de caracteres indicado por n desde el stream apuntado por stream al array apuntado por cadena. Ningún carácter adicional es leído después del carácter de nueva línea (el cual es retenido) o después de un final de fichero (EOF). Un carácter nulo es escrito inmediatamente después del último carácter leído en el array.
- La función fgets retorna el puntero “cadena” si la lectura es realizada con éxito o NULL en caso contrario.

2 Ejemplo scanf() y gets()

- Supongamos que el usuario introduce un número y retorno de carro, otro número y retorno de carro y por último un nombre y retorno de carro.
  - "33\n55\nJuan\n"
- La secuencia de lectura sería la siguiente:

Entrada	Buffer antes	Instrucción	Buffer después
33\n	33\n	`scanf("%d", &i1);`	\n
55\n	\n55\n	`scanf("%d", &i2);`	\n
	\n	`gets("%s", s1);`	
Juan\n	Juan\n	`gets("%s", s2);`	

```
int i1, i2;
char s1[30], s2[30];

scanf("%d", &i1);
scanf("%d", &i2);

gets(s1);
gets(s2);
```

Recordad: Cuando se utiliza la consola, el programa no lee directamente el texto tal cual lo introduce el usuario si no que éste se almacena en una tabla intermedia que llamaremos buffer. Cada vez que el usuario pulsa el retorno de carro, se llena el buffer con la línea introducida (incluyendo el carácter '\n').

2 Ejemplo scanf() y gets()

- El primer scanf tiene que leer del buffer hasta que encuentra un número (%d).
- En cuanto encuentra el 33 termina de leer, y deja en el buffer un '\n'.
- El segundo scanf, tras la entrada del usuario, se encuentra con \n55\n, y tiene que realizar la misma tarea que el anterior, leer un número.
- Se salta el primer '\n', y lee 30, dejando de nuevo un '\n' en el buffer.
- La función gets es más simple, y lo único que hace es leer todo lo que haya en el buffer hasta que encuentre un '\n', y lo copia en la variable correspondiente.
  - Así, el primer gets se encuentra un \n, lo consume, pero no copia nada en s1.
  - El segundo gets se encuentra Juan\n, así que lee todo lo que hay en el buffer y lo guarda en s2.
- Para permitir que Juan se copie en s1, una posibilidad es incluir una llamada a getchar() antes de emplear gets para leer s1.
  - Esta función lee un carácter del buffer, consumiendo así el '\n' que impedía rellenar s1 con Juan

3. Arrays Multidimensionales

- Los arrays multidimensionales son aquellos que tiene más de una dimensión y, en consecuencia, más de un índice.
- Los de 2 dimensiones se conocen por el nombre de tablas o matrices.
- La sintaxis de la declaración de un array de dos dimensiones es:

```
<tipo de elemento> <nombre array> [<numero de filas>] [<numero de columnas>]

int a[3][6]
```

- Los elementos de un array multidimensional se almacenan en memoria por filas
- Hay que tener en cuenta que el subíndice más próximo al nombre es la fila y el otro la columna.
- Para el ejemplo anterior el orden de almacenamiento es el siguiente:

```
a[0][0], a[0][1],…, a[0][5], a[1][0], a[1][1], …, a[1][5], a[2][0], a[2][1],
…, a[2][5]
```

3. Arrays Multidimensionales

- Inicialización
  - Los arrays multidimensionales se pueden inicializar al igual que los de una dimensión, cuando se declaran.
  - La inicialización consta de una lista de constantes separadas por comas (,) y encerradas entre llaves ({})

```
int ejemplo [2][3] = {1,2,3,4,5,6}
```

  - Acceso a los elementos de los arrays bidimensionales para la asignación directa de valores es:
    - Inserción de elementos

```
<nombre array> [índice fila] [índice columna] = valor elemento;
```

    - Extracción de elementos

```
<variable> = <nombre array> [índice fila] [índice columna];
```

3. Arrays Multidimensionales

- Acceso a elementos de arrays bidimensionales mediante bucles

```
int indiceFila, indiceCol;
for (indiceFila=0 ; indiceFila < NUMFILAS ; ++indiceFila)
        for (indiceCol=0 ; indiceCol < NUMCOL ; ++indiceCol)
                Procesar elemento [indiceFila][indiceDol];
```

```
ejemplo [2][3] = {1,2,3,4,5,6}
```

3. Ejemplo1: Arrays Multidimensionales

- Escribe un programa que inicialice una matriz de orden 3 por 4, por filas y posteriormente la escriba por columnas (traspuesta)

3. Ejemplo 2: Arrays Multidimensionales

- Escribe un programa que le pida al usuario los valores de una matriz cuadrada 3x3, la presente por pantalla, y luego muestre la suma de todos los números de la diagonal principal.

3. Ejemplo 3: Arrays unidimensionales

- Escribe un programa que le pida al usuario 10 valores enteros , los almacene, y realice la suma, la media y nos devuelva el máximo y el mínimo. Al final del programa, después de haber mostrado los valores anteriores, mostrará la lista de números introducidos.

3. Ejemplo 4: Arrays multidimensionales

- Introducir una única línea que contenga entre 2 y 10 palabras separadas por coma (,), sin espacios, y presentarlas en orden inverso. Las palabras tendrán un máximo de 10 letras. El número de palabras es desconocido, sólo sabemos que serán como mínimo 2 y como máximo 10, el programa debe adaptarse a lo que introduzca el usuario.
  - En caso de introducir palabras de más de 10 letras, el programa mostrará un error y volverá a pedir una línea que contenga las palabras correctas.
  - En caso de introducir más de 10 palabras, o menos de 2, el programa mostrará un error y volverá a pedir la línea.
  - Repetir hasta que se tengan todas las palabras introducidas correctamente.
  - Mostrar las palabras en orden inverso de introducción, separadas por comas.
  - EJ:
    - Usuario introduce la línea : perro,gato,oso
    - El programa muestra la línea: oso,gato,perro

3. Ejemplo 5: Cadenas de carateres

- Escribe un programa lea en una cadena de caracteres un número entero y convierta la cadena a número