Tema 6: Arrays, Matrices y Cadenas de Caracteres

Introducción a la programación I

Ana Isabel Sierra de las Heras
Marcos Novalbos
Francisco Javier Garcia Algarra
Rodrigo Alonso Solaguren-Beascoa
Alfonso Castro Escudero

Índice

1. Arrays

Variable de tipo char

char Letra ‘A’;

¿En qué estructura de datos (tipo) se guardaría una letra?

Utilizaríamos para un carácter una variable tipo char

¿Y para un nombre?

Conjunto de variables de tipo char

char Letra1 ‘A’;
char Letra2 ‘n’;
char Letra3 ‘a’;

En un principio podríamos utilizar tantas variables tipo char como letras tenga el nombre

Pero tendríamos dificultad para declararlos, ya que en nombres muy largos se declararían muchas variables

Array de tipo char

char Nombre [3];
Nombre [0] =‘A’;
Nombre [1] =‘n’;
Nombre [2] =‘a’;

ARRAYS

1. Arrays

Variable de tipo char

char Letra ‘A’;

Conjunto de variables de tipo char

char Letra1 ‘A’;
char Letra2 ‘n’;
char Letra3 ‘a’;

Array de tipo char

char Nombre [3];
Nombre [0] =‘A’;
Nombre [1] =‘n’;
Nombre [2] =‘a’;

Array de tipo char

char Nombre [6];
Nombre [0] =‘A’;
Nombre [1] =‘n’;
Nombre [2] =‘a’;
Nombre [3] =‘b’;
Nombre [4] =‘e’;
Nombre [5] =‘l’;

Un array es una estructura de datos de tamaño determinado que permite almacenar elementos del mismo tipo en una secuencia contigua de memoria. Cada elemento del array se accede mediante un índice (posición) que comienza desde cero

1. Arrays

- Un array es una variable estructurada formada por una secuencia de datos de tamaño determinado del mismo tipo y almacenados de manera contigua.
- 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, inicializado 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, inicializado 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) / número 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)/sizeof(array1[0]));
```

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

1. Ejercicio 1: Arrays

- 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.

1. Ejercicio 1: Arrays

- 1.4 Modificar el anterior, para que no se repitan los números en el array.
- 1.5 Modificar el programa para que el mensaje que se muestra si el número pedido al usuario está incluido en los números del array, incluya la posición del array donde está.

1. Ejercicio 1: Arrays

- 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.

```
#include <stdio.h>
#include <stdlib.h> /* rand */
#include <unistd.h> /* getpid */
#define NUM_MAX_NUM 10

//funcion que devuelve un entero generado de manera aleatoria
//entre 0 y 30. No se le pasa nada
int generaNumero0a30();

void main() {
    int array[NUM_MAX_NUM];
    // Inicializa la semilla para la generación de números aleatorios
    srand((unsigned int)getpid);
    // Rellena el array con números aleatorios entre 0 y 30
    for (int i = 0; i < NUM_MAX_NUM; i++) {
        array[i] = generaNumero0a30;  // Genera números entre 0 y 30
    }
    // Muestra el contenido del array por pantalla
    printf("Contenido del array:\n");
    for (int i = 0; i < NUM_MAX_NUM; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int generaNumero0a30(){
            return((int)rand() % 31);
}
```

1. Ejercicio 1: Arrays

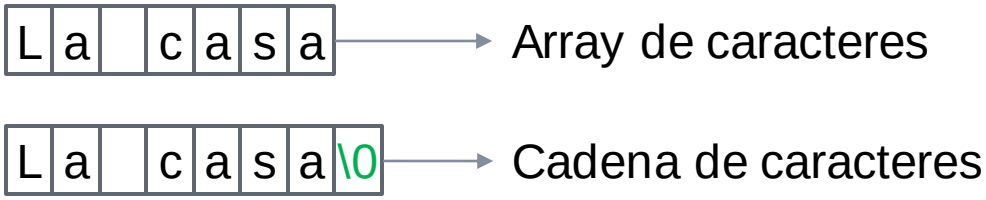
- 1.2 Al programa anterior, añadir lo siguiente: Después de mostrar el resultado, pedirá un número entre el 0 y el 30 al usuario. Buscar si ese número está en el array, y mostrar los siguientes mensajes: "Número encontrado" en caso de encontrarlo. Mostrar "No encontrado" en caso contrario.

```
#include <stdio.h>
#include <stdlib.h> /* rand */
#include <unistd.h> /* getpid */
#define NUM_MAX_NUM 10

//funcion que devuelve un entero generado de manera aleatoria
//entre 0 y 30. No se le pasa nada
int generaNumero0a30();

void main() {
    int array[NUM_MAX_NUM];
    // Inicializa la semilla para la generación de números aleatorios
    srand((unsigned int)getpid);
    // Rellena el array con números aleatorios entre 0 y 30
    for (int i = 0; i < NUM_MAX_NUM; i++) {
        array[i] = generaNumero0a30;  // Genera números entre 0 y 30
    }
    // Muestra el contenido del array por pantalla
    printf("Contenido del array:\n");
    for (int i = 0; i < NUM_MAX_NUM; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int generaNumero0a30(){
            return((int)rand() % 31);
}
```

```
printf("Ingresa un número entre 0 y 30: ");
scanf("%d", &numeroUsuario);
for (int i = 0; i < NUM_MAX_NUM; i++) {
     if (miarray[i] == numero)
          NumeroEncontrado = 1;
}
if (NumeroEncontrado)) {
     printf("El número %d está en el array.\n", numeroUsuario);
} else {
     printf("El número %d no está en el array.\n", numeroUsuario);
}
```

2. Cadenas de caracteres 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 de caracteres 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 de caracteres 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 de caracteres en C. Lectura de cadenas

Ejemplo scanf()

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

2. Cadenas de caracteres en C. Lectura de cadenas

Ejemplo scanf()

- 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 de caracteres. Lectura de cadenas. `scanf()`

- Cuando se utiliza la consola, el programa no lee directamente el texto tal cual lo introduce el usuario sino 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 o 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 de caracteres. 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 de caracteres. 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.

2. Cadenas de caracteres. 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 caracter)
```

2. Cadenas de caracteres. Lectura de cadenas

Ejemplo getchar() y putchar():

- Solicita al usuario su nombre y apellido utilizando getchar(). Almacénalo en una cadena de caracteres de 40 elementos. Imprime ambas la cadena por pantalla con putchar().

2. Cadenas de caracteres. Lectura de cadenas

Solución Ejemplo getchar() y putchar():

- Solicita al usuario su nombre y apellido utilizando getchar(). Almacénalo en una cadena de caracteres de 40 elementos. Imprime ambas la cadena por pantalla con putchar().

```
#include <stdio.h>
#define TAM 40

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 de caracteres. Lectura de cadenas

Solución Ejemplo getchar() y putchar():

- Solicita al usuario su nombre y apellido utilizando getchar(). Almacénalo en una cadena de caracteres de 40 elementos. Imprime ambas la cadena por pantalla con putchar().

```
#include <stdio.h>
#define TAM 40

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');
}
```

¿Es un nombre compuesto? ¿Calcula el número de palabras que tiene el nombre y apellidos?

Escribe cada una de las palabras en una línea distinta

Cifrad el nombre y apellidos escribiéndolos al revés

2. Cadenas de caracteres. 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 de caracteres en C. Lectura de cadena. `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. **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 de caracteres 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 el paráetro "n" (es decir n-1) 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.

`fgets` es más segura que `gets`, pero tiene un par de inconvenientes;

- Si se han introducido más caracteres de los que se han podido leer, dichos caracteres se quedan en el buffer de entrada (en stdin), por lo que deben ser eliminados para que no sean leídos en el próximo `fgets` (siempre que no se desean leer).
- `Si fgets` ha leido el carácter \n, éste se incluye al final, así que será necesario eliminarlo.

2. Ejemplo scanf() y gets()

- Supongamos que el usuario introduce un número y retorno de línea, otro número y retorno de línea y por último un nombre y retorno de línea.
  - "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(s1);`	
Juan\n	Juan\n	`gets(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 sino 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 (consumiendo 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. Matrices.

- Los arrays multidimensionales son aquellos que tiene más de una dimensión y, en consecuencia, más de un índice.
- Los arrays 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. Matrices.

- 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. Matrices.

- Acceso a elementos de arrays bidimensionales mediante bucles

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

int indiceFila, indiceCol;

for (indiceFila=0 ; indiceFila < NUMFILAS ; ++indiceFila)
  for (indiceCol=0 ; indiceCol < NUMCOL ; ++indiceCol)
    {
        printf("%d\n",elementos[indiceFila][indiceCol]);
    }
```

4. Paso de Arrays y Matrices como parámetros a funciones

En C no se copia el array o la matriz cuando se pasan como argumento a una función, sino que se le dice en qué parte de la memoria están para que la función trabaje sobre el array o la matriz original. Esta solución, se conoce como paso de parámetros por referencia a diferencia del mecanismo de paso por copia también denominado paso de parámetros por valor.

4.1 Paso de arrays como parámetros

- El prototipo típico de una función que acepta un array como argumento es:

```
tipo_devuelto NombreFunción(tipo nombre_array[], int longitud_array);
```

- Es decir, se ponen dos corchetes [] a continuación del nombre del array. No es necesario especificar el tamaño del array entre los corchetes para conseguir que la función es genérica y valga para cualquier tamaño. Por ello es casi obligatorio enviar la longitud del array como un argumento adicional.
- Por ejemplo, una función que calcule el producto escalar de dos vectores de igual dimensión tendrá como prototipo:

```
double ProdEscalar(double array1[],  double array2[], int dim);
```

4. Paso de Arrays y Matrices como parámetros a funciones

y en este caso el código del programa para invocar la función y la definición de la función serían:

```
#include <stdio.h>
#define  DIMENSION 3
// Prototipos de las funciones
double ProdEscalar(double arr11[], double arr22[], int dim);

int main(void)
{
      double array1[3]={1, 2, 3};
      double array2[3]={3, 2, 1};
      double d_escalar;

      d_escalar = ProdEscalar(array1, array2, DIMENSION);
      printf("El producto escalar vale %g\n", d_escalar);

      return 0;
}

double ProdEscalar(double arr1[], double arr2[], int dim)
{
      int i;
      double producto;
      producto = 0;
      for(i=0; i<dim; i++)
      {
                  producto += arr1[i]*arr2[i];
      }
      return producto;
}
```

4. Paso de Arrays y Matrices como parámetros a funciones

4.2 Paso de cadenas de caracteres como parámetros

Todo lo que se acaba de comentar sobre el paso de arrays a funciones es aplicable al paso de cadenas a funciones, pues las cadenas de caracteres no son más que arrays de tipo char. La única diferencia es que las cadenas utilizan el carácter '\0' para indicar el final de la cadena en el último elemento del array y por tanto no es necesario indicar su longitud al llamar a la función. Sin embargo, en arrays de tipo numérico, como ya hemos visto sí se indica la longitud.

Veamos dos ejemplos:

- Prototipo de una función que cuenta el número de valores negativos de un array:

```
int Negativos(double array[], int dim);
```

- Prototipo de una función que cuenta el número de consonantes de una cadena de caracteres:

```
int Consonantes(char cadena[]);
```

4. Paso de Arrays y Matrices como parámetros a funciones

4.2 Paso de cadenas de caracteres como parámetros. Ejemplo

Crea una función que imprima una cadena de caracteres.

```
#include <stdio.h>

// Función que imprime una cadena de caracteres
//void imprimirCadena(char *cadena);
void imprimirCadena(char cadena[]);

void main() {
    // Definir una cadena de caracteres
    char miCadena[] = "Hola, mundo!";

    // Llamar a la función e imprimir la cadena
    imprimirCadena(miCadena);
}

//void imprimirCadena(char *cadena) {
void imprimirCadena(char cadena[]) {
    //while (*cadena != '\0') {
    //    printf("%c", *cadena);
    //    cadena++;
    //}
       int i=0;
       while (cadena[i] != '\0') {
        printf("%c", cadena[i]);
        i++;
    }
    printf("\n");
}
```

4. Paso de Arrays y Matrices como parámetros a funciones

4.3 Paso de matrices como parámetros

El paso de matrices a una función es un poco distinto al de los arrays. En el caso de las matrices o arrays multidimensionales es necesario indicar todas las dimensiones de la matriz salvo la primera. Adicionalmente es necesario indicar las dimensiones reales en el caso de no aprovechar completamente la matriz.

Por ejemplo, si queremos realizar una función para inicializar parcialmente matrices de 30x50, el prototipo de la función será:

```
void InicializaMatriz(float matriz[][50], int n, int m);
```

Y el cuerpo de esta función es:

4. Paso de Arrays y Matrices como parámetros a funciones

```
void InicializaMatriz(float matriz[][50], int n, int m)
{
   int i;
   int j;
   for(i=0; i<n; i++){
     for(j=0; j<m; j++){
        mat[i][j] = i*m+j;
     }
   }
}
```

y para llamarla desde otra función se escribe el nombre de la matriz sin corchetes de la misma forma que se hace con los arrays. Por ejemplo para invocar la función anterior:

```
...
float mat[30][50];
int filas = 0, columnas = 0;
...
printf("Introduzca el número de filas: ");
scanf(" %d", &filas);
printf("Introduzca el número de columnas: ");
scanf(" %d", &columnas);
InicializaMatriz(mat, filas , columnas);
...
```

5. Ejercicios

Ejercicio 2: Arrays Multidimensionales:

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

5. Ejercicios

Ejercicio 2: Arrays Multidimensionales (Solución):

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

```
#include <stdio.h>

#define MAXFILAS 3
#define MAXCOL 4

void main()
{

          int i,j;
          int Matriz[MAXFILAS][MAXCOL];

          /* Lecdtura por files */
          for (i=0 ; i<MAXFILAS ; i++)
              for (j=0 ; j<MAXCOL ; j++)
              {
                   printf ("Introduce el elemento [%d] [%d] de la matriz\n",i,j);
                    scanf ("%d",&Matriz[i][j]);
              }
          /*Escritura por columnas */
          for (j=0 ; j<MAXCOL ; j++)
          {
                        for (i=0 ; i<MAXFILAS ; i++)
                              printf ("%5d",Matriz[i][j]);
                        printf ("\n");

          }
}
```

5. Ejercicios

Ejercicio 3: 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.

5. Ejercicios

Ejercicio 3: Arrays Multidimensionales. Solución:

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

```
#include <stdio.h>
#define MAXFILCOL 3

void main(){
          int i,j,suma = 0;
          int Matriz[MAXFILCOL][MAXFILCOL];

          /* Lectura por files */
          for (i=0 ; i< MAXFILCOL ; i++)
               for (j=0 ; j< MAXFILCOL ; j++){
                    printf ("Introduce el elemento [%d] [%d] de la matriz\n",i,j);
                    scanf ("%d",&Matriz[i][j]);
                         }
          /*Escritura por filas */
          for (i=0 ; i<MAXCOL ; i++){
               for (j=0 ; j<MAXFILAS ; j++)
                    printf ("%5d",Matriz[i][j]);
          printf ("\n");

          }
          /* Suma diagonal */
          for (i=0 ; i<MAXCOL ; i++){
               for (j=0 ; j<MAXFILAS ; j++)
                    if (i == j)
                         suma += Matriz[i][j];
          printf ("La suma de los elementos de la diagonal principal es: %d\n",suma);
}
```

5. Ejercicios

Ejercicio 4: 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, en orden inverso usado para introducirlos.

5. Ejercicios

Ejercicio 4: Arrays Unidimensionales. Solución:

- Escribe un programa que le pida al usuario 10 valores enteros y realice la suma, la media y nos devuelva el máximo y el mínimo. Debe mostrar los números después de mostrar los cálculos.

```
#include <stdio.h>
#define NUM_ELEMEBTOS 10

void main(){
          float i,suma = 0, max = 0, min = 0;
          float numeros[NUM_ELEMEBTOS];

          /* Lectura array*/
          for (i=0 ; i< NUM_ELEMEBTOS ; i++)
               printf ("Introduce el Numero [%d] \n",i);
               scanf ("%d",&numeros[i]);
          }
          min = numeros[0];
          /*Calculo suma, media, máximo y minimo*/
          for (i=0 ; i<NUM_ELEM ; i++){
               suma += numeros[i];
               if (numeros[i]>max)
                    max = numeros[i];
               if (numeros[i]<min)
                    min = numeros[i];
                         printf ("\n");
          }
          media = suma / NUM_ELEMEBTOS ;
          printf ("La suma, media, máximo y minimo de los números es: %f, %f, %f, %f\n",suma,media,max,min);
}
```

5. Ejercicios

Ejercicio 5: 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

5. Ejercicios

Ejercicio 5: Arrays Multidimensionales. Solución:

Introducir de 2 a 10 palabras separadas por coma (,) y presentarlas en orden inverso

```c
#include <stdio.h>

#define MAX_NUM_PALABRAS 10
#define MAX_TAM_PALABRAS 15

void main(){

    char palabras [MAX_NUM_PALABRAS][MAX_TAM_PALABRAS];
    char letra;
    int i=0,j=0, k=0, palabralarga=0;
    printf ("Introduce de 2 a 10 palabras separadas por comas\n");
    do{
        letra = getchar();
        /* Si no es la ultima */
        if (letra != '\n'){
            if (letra == ','){
                /* Aniadri el nulo al final de cada palabra (menos la ultima) */
                palabras [j][i] = '\0';
                j++;
                i = 0;
            }
            else{
                if (i<MAX_TAM_PALABRAS-1){
                    palabras [j][i] = letra;
                    printf ("%c/",letra);
                    i++;
                }else{
                    palabralarga =1;
                    palabras [j][i] = '\0';
                }
            }
            printf ("\n");
        }else{
            /*anidair el nulo al final de la ultima palabra */
            palabras [j][i] = '\0';
        }
    }while ((letra != '\n') && (j<MAX_NUM_PALABRAS) && (!palabralarga));

    k=0;
    /*Escritura palabras tal como se introducen */
    if (!palabralarga){
        for (k=0 ; k<=j ; k++)
            printf ("%s, ",palabras[k]);
        printf ("\n");

        for (k=j ; k>=0 ;k--)
            printf ("%s, ",palabras[k]);
        printf ("\n");
    }else
        printf ("Has introducido una palabra con mas letras de las permitidas\n");

}
/*
```

5. Ejercicios

Ejercicio 6: Cadenas:

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

5. Ejercicios

Ejercicio 6: Cadenas. Solución:

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

```
#include <stdio.h>
#include <math.h>
#define NUM_DIGITOS  10

int main()
{
        char cadena_digitos[NUM_DIGITOS];
        int i=0; int numero=0;
        int valor_dig;
        char car;
        int exp=0;

        printf("Introduce una cadena de digitos: \n");
        while((car=getchar()) != '\n')
        {
             cadena_digitos[i]=car;
             i++;
        }
        cadena_digitos[i]='\0';
        printf("cadena de digitos: %s\n", cadena_digitos);

        for(int j=i-1; j>=0; j--)
        {
           valor_dig=cadena_digitos[j]-'0';
           numero=numero+valor_dig*pow(10,exp);
           exp++;
        }
        printf("Valor numerico de la cadena: %d\n", numero);
}
```

5. Ejercicios

Ejercicio 7: Arrays Multidimensionales: Multiplicación matrices

- Escribir un programa que calcule la multiplicación de dos matrices. Comprobad los resultados con la multiplicación de una matriz de 3x3 con una de 3x2.

5. Ejercicios

Ejercicio 7: Arrays Multidimensionales: Multiplicación matrices

- Escribir un programa que calcule la multiplicación de dos matrices. Comprobad los resultados con la multiplicación de una matriz de 3x3 con una de 3x2.

```
#include <stdio.h>
#define FILAS_MATRIZ_B 3
#define COLUMNAS_MATRIZ_B 2
#define FILAS_MATRIZ_A 3
#define COLUMNAS_MATRIZ_A 3

int main (int argc, char **argv[]){

int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {{3, 2, 1},{1, 1, 3},{0, 2, 1}};
int matrizB[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B] = {{2, 1},{1, 0},{3, 2}};
// Necesitamos hacer esto por cada columna de la segunda matriz (B)
        for (int a = 0; a < COLUMNAS_MATRIZ_B; a++) {
                    // Dentro recorremos las filas de la primera (A)
                    for (int i = 0; i < FILAS_MATRIZ_A; i++) {
                                int suma = 0;
                                // Y cada columna de la primera (A)
                                for (int j = 0; j < COLUMNAS_MATRIZ_A; j++) {
                                            // Multiplicamos y sumamos resultado
                                            suma += matrizA[i][j] * matrizB[j][a];
                                }
                                // Lo acomodamos dentro del producto
                                producto[i][a] = suma;
                    }
        }
}
```

5. Ejercicios

Ejercicio 8: Número capicúa

- Escribir un programa que solicite al usuario un número natural para que lo introduzca por teclado y almacene cada una de las cifras en un array de enteros. Presentar el array de enteros y determinar si el número es capicúa.

5. Ejercicios

Ejercicio 9: Jugando con nombres y apellidos

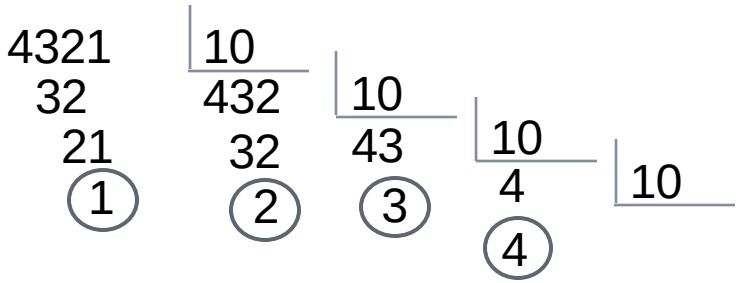
- En un formulario se piden por separado nombre y apellidos y se almacenan en dos cadenas de caracteres, permitiendo introducir para cada uno de ellos 25 caracteres. Generar una cadena que se llame nombre y apellidos y que sea la concatenación de las cadenas nombre y apellidos por separado.

5. Ejercicios

Ejercicio 10: Trabajando con cadenas

- Crear un programa en c que calcule para un texto introducido y que se almacena en una cadena de caracteres de tamaño 100, advirtiendo al usuario que ha superado el número y trabajando con 99 caracteres:
  - El número de las palabras
  - El número de vocales minúsculas introducidas
- Hasta que el usuario introduzca un intro. En ese momento el programa enseña las estadísticas recogidas y acaba.
- El programa deberá tener en cuenta que el usuario puede teclear varios espacios consecutivos sin que aumenten el número de palabras.
- Se deberán implementar las siguientes funciones:

```
//Devuelve un entero con el numero de palabras de un texto. Recibe como parámetros la dirección de comienzo de la cadena de caracteres con el texto
int contarPalabras(char Texto[]);

//Devuelve un entero con el numero de vocales minúsculas distintas de un texto. Recibe como parámetros la dirección de comienzo de la cadena de caracteres con el texto
int contarVocalesMinus(char Texto[]);

//Recibe como parámetros el número de palabras y el número de cada tipo de vocal y lo presenta por pantalla.
void imprimeResultado(int numPalabras, int numVocales);
```


4321
10
32
432
10
21
32
43
10
1
2
3
4
10

CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL