

Programación en C: Enumerados, registros y arrays

1ºDAM IoT

Contenido

- Introducción
- Enumeraciones
- Registros
- Arrays
 - Arrays unidimensionales
 - Arrays multidimensionales
- Composición de estructuras de datos
 - Registro con campo basado en registro
 - Registro con campo array
 - Array de registros

Introducción

Las **estructuras de datos** son una composición de múltiples datos que guardan relación entre sí.

Los tipos primitivos (int, char, double, float) sirven de base para construir estas estructuras.

Introducción

Dependiendo de cómo sea dicha relación, del tipo de datos y de las operaciones que podemos realizar con ellos, nos encontramos con diferentes tipos de estructuras (estáticas y dinámicas).

- **Estructuras de datos estáticas:**
 - No tienen posibilidad de alterar su composición a lo largo de la ejecución del programa.
 - Los valores sí pueden cambiar, pero la cantidad de datos y las relaciones entre ellos no.
- **Estructuras de datos dinámicas:**
 - Sí permiten mayor flexibilidad en términos de crecer, reducirse o, en general, reestructurarse.
 - Son más complejas, las veremos en la siguiente unidad.

Las estructuras de datos pueden componerse no sólo de tipos primitivos, sino también de otras estructuras más simples.

Enumeraciones

Una enumeración es un conjunto de constantes enteras con nombre y especifica todos los valores legales que pueden tener unas variables. Las enumeraciones se declaran de la siguiente forma:

```
enum nombre_enum{lista_de_enumeración} lista_de_variables;
```

Las enumeraciones asignan una constante entera a cada uno de los símbolos de la enumeración, empezando por el valor 0.

Las enumeraciones tienen por objetivo que el programa sea más legible para cuando tengamos que hacer cambios.

Enumeraciones

Declaración de un enumerado.

```
enum dias_semana
{
    LUNES,
    MARTES,
    MIERCOLES,
    JUEVES,
    VIERNES,
    SABADO,
    DOMINGO
};
enum dias_semana op;
```

LUNES toma el valor de 0, MARTES el 1, MIÉRCOLES 2, etc...

Posible uso del enumerado

```
switch (op)
{
    case LUNES:
        printf("Empieza la semana");
        break;
    case MARTES:
        printf("Ya no es lunes!");
        break;
    case MIERCOLES:
        printf("Ya llevas la mitad");
        break;
    case JUEVES:
        printf("Ya llevas la mitad");
        break;
    case VIERNES:
    case SABADO:
    case DOMINGO:
        printf("Estamos en fin de semana");
        break;
    default:
        printf("Error");
}
```

Enumeraciones

Puedo declarar una variable de tipo enumerado que sólo podrá tener el rango de valores definido por enumerado.

```
enum diasSemana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO };  
enum diasSemana laborable;  
laborable = JUEVES;  
if (laborable <= VIERNES) printf("Sigue trabajando, :(");  
else printf("Ya es fin de semana");
```

Enumeraciones

¿Cómo crearías un tipo booleano con una enumeración?

Enumeraciones

Puedo crear un enumerado para simular un booleano.

Recuerda que cuando una expresión a evaluar es 0 es falso y cuando es verdadera es 1. FALSO = 0, VERDADERO = 1.

```
enum miBooleano {FALSO, VERDADERO} encontrado;  
encontrado = FALSO; //Inicializamos a Falso  
//Secuencia de instrucciones....  
//En un momento dado le asignamos el valor de verdadero  
encontrado = VERDADERO;  
if (encontrado) printf("Hemos encontrado lo que queríamos");  
else printf("No encontramos lo que queríamos");
```

Registros o estructuras

Un registro o estructura es una colección de datos individuales que el programador decide agrupar bajo un identificador común.

Los datos agrupados en un registro cumplen dos propiedades:

- No tiene por qué ser del mismo tipo de dato (se mezclan números con textos, etc.).
- Todos ellos se refieren a una misma cosa y ese es el sentido de agruparlos.

Registros

Ejemplo: Si necesitamos manejar información referida a una posición de los ejes de coordenadas “x” e “y”. Seguramente utilizaríamos dos variables de tipo float.

```
float posicion_x, posicion_y;
```

¿Qué ocurre si tengo que guardar la información de 10 posiciones?

```
float posicion_x1, posicion_y1;  
float posicion_x2, posicion_y2;  
float posicion_x3, posicion_y3;  
float posicion_x4, posicion_y4;  
float posicion_x5, posicion_y5;  
float posicion_x6, posicion_y6;  
float posicion_x7, posicion_y7;  
float posicion_x8, posicion_y8;  
float posicion_x9, posicion_y9;  
float posicion_x10, posicion_y10;
```

Tendrás que crear 20 variables para las 10 posiciones.

Registros

Un registro es una estructura de datos que permite almacenar un conjunto de elementos no necesariamente del mismo tipo.

```
struct nombre_estructura {  
    tipo nombre_variable;  
    tipo nombre_variable;  
    ...  
    tipo nombre_variable;  
};
```

Indicamos el nombre de la estructura después de la palabra reservada struct.

Entre llaves declaramos los tipos de datos que tiene la estructura de la misma forma que declaramos las variables en el código.

; al final

Registros

Volviendo al ejemplo en el que queremos guardar una coordenada.

Vamos a crear una estructura para guardar la posición en x y en y.

Palabra clave “struct”

Los tipos primitivos
aparecen dentro de las
llaves.

```
struct posicion  
{  
    float posicion_x;  
    float posicion_y;  
};
```

; al final

Identificador de la estructura.

Cada tipo primitivo en una
línea.

Registros

Al estructurar los campos dentro de un tipo de dato estructurado, todos ellos quedarán vinculados a una misma variable.

Podremos declarar variables del tipo la estructura que hayamos creado.

En nuestro caso, declararemos variables de tipo “posicion”.

Cada variable de tipo “posicion” constará de 2 campos de tipo float llamados `posicion_x` y `posicion_y`.

Utilizaremos el punto “.” para acceder a cada campo de una variable declarada como un registro/estructura.


Registros

Para declarar una variable del tipo la estructura creada.

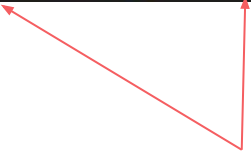
struct *nombre_estructura nombre_variable;*

Ya tenemos nuestra estructura “posicion” definida previamente, ahora vamos a declarar un par de variables de tipo posicion.

```
struct posicion posicion1, posicion2;
```



Indico que se trata de una estructura llamada posicion.



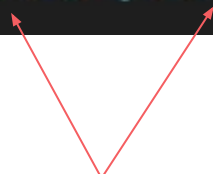
Variables del tipo la estructura creada.

Registros

También podemos declararlas la variables a la vez que declaramos la estructura:

```
struct nombre_estructura {  
    tipo nombre_variable;  
    tipo nombre_variable;  
    ...  
    tipo nombre_variable;  
}variables_estructura;
```

```
struct posicion  
{  
    float posicion_x;  
    float posicion_y;  
} posicion1, posicion2;
```



A la vez que declaramos la estructura, declaramos las variables posicion1 y posicion2.

Registros

También puedo declarar una variable e iniciarle sus valores.

En el siguiente ejemplo fíjate que únicamente inicializo `posicion2`, `1.0` corresponde al valor de `posicion_x` y `2.0` al valor de `posicion_y`.

```
struct posicion
{
    float posicion_x;
    float posicion_y;
} posicion1, posicion2 = {1.0, 2.0};
```

Registros

Para referenciar un elemento de una estructura utilizamos el nombre de la variable de tipo estructura seguido del caracter punto y el nombre de la variable:

variable_estructura.nombre_variable;

```
posicion1.posicion_x = 3.2;  
posicion1.posicion_y = -2.2;  
  
posicion2.posicion_x = 0;  
posicion2.posicion_y = -10;
```

Registros

El lenguaje C permite mediante el uso de la palabra reservada **typedef** definir nuevos nombres para los tipos de datos existentes, esto no debe confundirse con la creación de un nuevo tipo de datos.

La palabra clave typedef permite solo asignarle un nuevo nombre a un tipo de datos ya existente.

La sintaxis general de uso de typedef es:

***typedef** tipo nombre;*

Donde tipo es cualquier tipo de datos permitido, y nombre es el nuevo nombre que se desea tenga ese tipo.

Registros


Aplicado a las estructuras podríamos definir estructuras y luego declararlas de la siguiente forma:

Defino un tipo de dato estructura llamado "posicion".

Ya no es necesario escribir struct en la declaración de una variable de tipo estructura.

```
typedef struct
{
    float posicion_x;
    float posicion_y;
} posicion;

posicion posicion1, posicion2;
```



Registros - Comparación y asignación

Es muy importante recalcar que, dos estructuras , aunque sean del mismo tipo , no pueden ser asignadas ó comparadas la una con la otra , en forma directa

No puedo comparar dos estructuras así:

`posicion1 == posicion2` ← Mal!!!!!!

Ni el contenido de una estructura en otra:

`posicion1 = posicion2;` ← Mal!!!!!!

Registros - Comparación y asignación

Para comparar dos variables de tipo estructura tengo que comparar cada uno de sus campos.

```
posicion1.posicion_x = 0;
posicion1.posicion_y = -10;

posicion2.posicion_x = 0;
posicion2.posicion_y = -10;

if (posicion1.posicion_x == posicion2.posicion_x && posicion1.posicion_y == posicion2.posicion_y)
{
    printf("Son iguales");
}
else
{
    printf("Son diferentes");
}
```

Registro

Ejercicio: Crea un registro que guarde el precio de un producto y el IVA. Solicita al usuario que introduzca el precio e IVA de dos productos. Guardalos en dos registros y muestra por pantalla el precio de cada producto incluido el IVA.

Registro

Solución:

```
struct producto
{
    float precio;
    int iva;
    float precioIva;
};

struct producto producto1;
struct producto producto2;
printf("Introduce los datos del primer producto\n");
printf("Precio:");
scanf("%f", &producto1.precio);
printf("IVA:");
scanf("%d", &producto1.iva);

printf("Introduce los datos del segundo producto\n");
printf("Precio:");
scanf("%f", &producto2.precio);
printf("IVA:");
scanf("%d", &producto2.iva);

producto1.precioIva = producto1.precio + (producto1.precio * producto1.iva / 100);
producto2.precioIva = producto2.precio + (producto2.precio * producto2.iva / 100);

printf("Precio con IVA del producto 1: %g\n", producto1.precioIva);
printf("Precio con IVA del producto 2: %g\n", producto2.precioIva);
```


Registros

Ejercicio: Crea una estructura que guarde de un artículo el valor de su peso y un carácter que indique la unidad. Para la unidad tomaremos “K” como kilo y “g” como gramos.

Crea 3 variables del tipo la estructura creada.

Inicializa las dos primeras con valores introducidos por el usuario.

1. Compara si los datos de los artículos son iguales.
2. Informa cuál pesa más. Ten en cuenta que 1 kilo son 1000 gramos.
3. Asigna el valor de la primera variable estructura a la tercera variable. Imprime los valores del artículo 1 y el artículo 3.

Registros

Solución:

Creación de la estructura,
declaración de variables y
declaración de datos.

```
struct  articulos
{
    float peso;
    char unidad;
};
struct articulos articulo1, articulo2, articulo3;
//DATOS DEL PRIMER ARTÍCULO
printf("Artículo1: Peso\n");
scanf("%d", &articulo1.peso);
printf("Artículo1: Unidad\n");
fflush(stdin);
scanf("%c", &articulo1.unidad); //Voy a suponer que solo introduce K o g
fflush(stdin);

//DATOS DEL SEGUNDO ARTÍCULO
printf("Artículo2: Peso\n");
scanf("%d", &articulo2.peso);
printf("Artículo2: Unidad\n"); //Voy a suponer que solo introduce K o g
fflush(stdin);
scanf("%c", &articulo2.unidad);
```

Registros

Solución:

```
//Comprobamos si los artículos son iguales
if(articulo1.peso == articulo2.peso && articulo1.unidad == articulo2.unidad){
    printf("Son iguales\n");
}else printf("No son iguales\n");

//Comprobamos cuál pesa más
float pesoAux1, pesoAux2;
if (articulo1.unidad == 'K') pesoAux1 = articulo1.peso*1000;
else pesoAux1 = articulo1.peso;
if (articulo2.unidad == 'K') pesoAux2 = articulo2.peso*1000;
else pesoAux2 = articulo2.peso;

if (pesoAux1 > pesoAux2) printf("El artículo 1 pesa más.\n");
else if (pesoAux1 < pesoAux2) printf("El artículo 2 pesa más.\n");
else printf("Los dos pesan igual.\n");

//Asignamos el primer artículo al tercero
articulo3.peso = articulo1.peso;
articulo3.unidad = articulo1.unidad;
printf("Artículo 1: Peso %f y unidad %c\n", articulo1.peso, articulo1.unidad);
printf("Artículo 3: Peso %f y unidad %c\n", articulo3.peso, articulo3.unidad);
```

Arrays

También conocidos como arreglos, vectores o matrices.

Hasta ahora hemos estado trabajando con variables y haciendo uso de ellas para almacenar unos cuantos valores que necesitábamos para que nuestros programas funcionaran.

Imagina que tienes un programa que relaciona los dorsales de una carrera con el nombre del corredor. ¿Sería viable utilizar una variable para cada nombre de corredor? Podemos usar registros, pero habría que declarar muchos (uno por cada corredor).

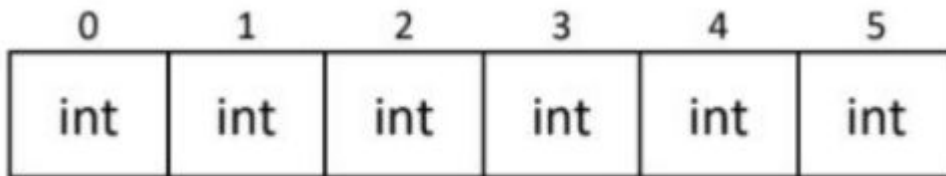
Con los arrays podemos almacenar un número grande de valores bajo un mismo identificador de variable.

Arrays

Por ejemplo, si esta es la representación para una variable de tipo entero (int).



Esta sería la representación de un array de 10 enteros.



Los arrays pueden tener una o varias dimensiones. Unidimensionales o multidimensionales.

Arrays unidimensionales

Son los más simples.

Los elementos se disponen numerados y uno a continuación de otro.

Al valor de numeración que reciben los elementos se le conoce como “índice” del array.

Debemos indicar el tamaño del array en la declaración.

El primer elemento de un array es el que posee el índice 0.

Un array de 5 elementos posee sus elementos numerados de 0 a 4.

Cuando declaro un array de 5 elementos, internamente se reserva la memoria de 5 posiciones de memoria consecutivas correspondientes con el tipo de dato del array.

sueldos				
1200	750	820	550	490
sueldos[0]	sueldos[1]	sueldos[2]	sueldos[3]	sueldos[4]

Arrays unidimensionales

Los declaramos de la siguiente forma: *tipoDato identificador [dimensión];*

Declaración del array unidimensional de 5 posiciones

```
int sueldos[5];  
sueldos[0] = 1200;  
sueldos[1] = 750;  
sueldos[2] = 820;  
sueldos[3] = 550;  
sueldos[4] = 490;
```

Entre corchetes se indica la longitud del array

Inicialización de todas las posiciones del array. Como es de tamaño 5, empieza en 0 y termina en 4.

```
sueldos[0] = sueldos[0] + 100;  
printf("Primer sueldo %i", sueldos[0]);
```

Ejemplos de uso de un elemento del array.

Arrays unidimensionales

Para recorrer el contenido del array podemos hacer uso de la estructura repetitiva for. Es la más usada para recorrer un array.

```
#include <stdio.h>

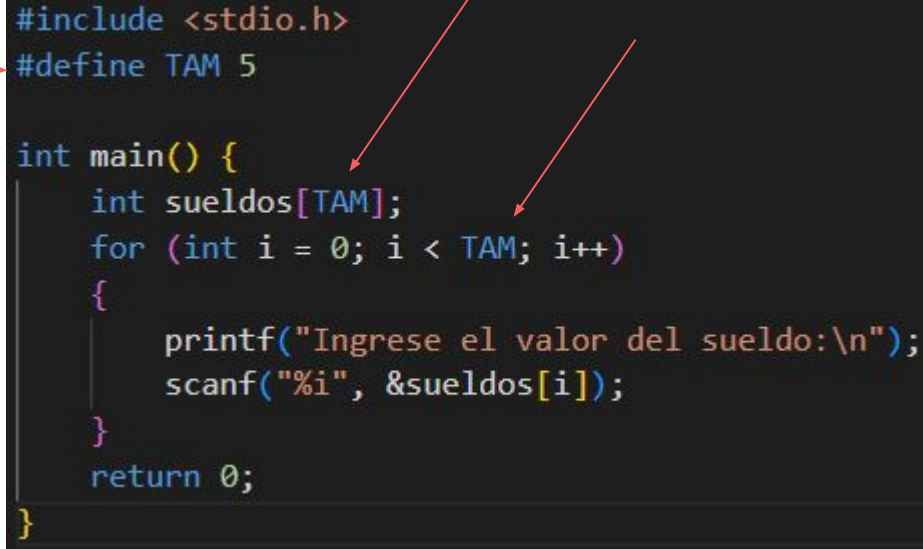
int main() {
    int sueldos[5];
    for (int i = 0; i < 5; i++)
    {
        printf("Ingrese el valor del sueldo:\n");
        scanf("%i", &sueldos[i]);
    }
    return 0;
}
```

Utilizo la variable `i` para recorrer cada posición del array. Empezando en 0 y llegando al 4. La posición 5 no existe, 5 es el tamaño del array.

Observa cómo leemos un entero que guardamos en el array en la posición actual.
Seguimos utilizando `&`.

Arrays unidimensionales

Es interesante definir una constante para el tamaño del array. Y utilizarla para indicar el tamaño del array y el número de repeticiones en bucles FOR.

A screenshot of a C program snippet on a dark background. The code defines a constant TAM as 5 and uses it in an array declaration and a for loop. Three red arrows point to specific parts: one to the #define line, one to the TAM in the array declaration, and one to the TAM in the loop condition.

```
#include <stdio.h>
#define TAM 5

int main() {
    int sueldos[TAM];
    for (int i = 0; i < TAM; i++)
    {
        printf("Ingrese el valor del sueldo:\n");
        scanf("%i", &sueldos[i]);
    }
    return 0;
}
```

Arrays unidimensionales

Ejercicio: Crea un array que guarde la temperatura de los últimos 7 días. Para ello solicita al usuario los datos. Una vez guardados imprime la temperatura media de los últimos 7 días por pantalla. Utiliza una constante para la dimensión del array.

Arrays unidimensionales

Solución:

```
#include <stdio.h>

#define TAM 7

int main()
{
    float temperatura[TAM];
    int i;
    float media;
    for (i = 0; i < TAM; i++)
    {
        printf("Introduce la temperatura ");
        scanf("%f", &temperatura[i]);
        media += temperatura[i];
    }
    media = media / TAM;
    printf("La temperatura media de los últimos %i días es %g\n", TAM, media);
    return 0;
}
```

Arrays unidimensionales


Podemos también inicializar el array en la propia declaración. Ese caso podríamos omitir si quisiéramos el tamaño, ya que entenderá que el tamaño es el número de elementos que le asignamos.

```
float temperatura[TAM] = {25, 26.2, 25.9, 26.1, 27, 27.2, 27.5};
```

Si no ponemos el tamaño del array y lo inicializamos, se guarda el espacio en memoria de los datos introducidos.

En los lenguajes tradicionales, cualquier acceso a un índice no permitido de un array suele desembocar en un error que fuerza la finalización del programa (lo que conocemos como excepción). En el ejemplo el array ha sido creado para 4 valores y TAM es igual 7. Los 3 últimos printf están accediendo a posiciones de memoria erróneas. Están imprimiendo basura aunque también podríamos haber accedido a una posición de memoria que no tenemos permisos y haber finalizado el programa bruscamente.

```
float temperatura[] = {25, 26.2, 25.9, 26.1};  
int i;  
for (i = 0; i < TAM; i++)  
{  
    printf("%g\n", temperatura[i]);  
}
```



```
25  
26.2  
25.9  
26.1  
5.60519e-045  
8.99954e-039  
3.59306e-039
```

Arrays unidimensionales

Tamaño del array: Si inicializamos el array sin indicar la dimensión podemos averiguar el número de elementos que tiene mediante el operador “**sizeof**”. **sizeof** nos devuelve el tamaño en bytes.

En el ejemplo `sizeof temperatura` me devuelve el espacio total en bytes del array y `sizeof(float)` el espacio que ocupa un float. Al dividirlo obtengo de posiciones del array, es decir, su longitud.

```
float temperatura[] = {25, 26.2, 25.9, 26.1};
int elementos = sizeof temperatura / sizeof(float);

int i;
for (i = 0; i < elementos; i++)
{
    printf("%g\n", temperatura[i]);
}
```

Arrays unidimensionales

Ejercicio: Inicializa un array con los valores enteros 2, 4, 10, 9, 1. Solicita al usuario un número del 1 al 10 y comprueba si existe en el array. Utiliza sizeof para calcular el número de elementos en el array.

Arrays unidimensionales

Solución: Inicializa un array con los valores enteros 2, 4, 10, 9, 1. Solicita al usuario un número del 1 al 10 y comprueba si existe en el array. Utiliza sizeof para calcular el número de elementos en el array.

```
int numeros[] = {2, 4, 10, 9, 1};
int tam = sizeof(numeros)/sizeof(int);
int num;
enum miBooleano {FALSO, VERDADERO} encontrado;
encontrado = FALSO; //Inicializamos a Falso
do{
    printf("Introduzca un número del 1 al 10");
    scanf("%d",&num);
}while(num<0 || num>10);
for (int i = 0; i < tam; i++)
{
    if(numeros[i]==num){
        encontrado = VERDADERO;
    }
}
if(encontrado) printf("El usuario acertó el número");
else printf("No lo encontró");
```

Arrays unidimensionales

Ejercicio: Escribe que rellene un array con los 100 primeros números enteros. Después muéstralos por pantalla en orden ascendente.

Debe mostrarlos en una misma línea separados por guiones.

¿Y si también los queremos imprimir de forma descendente?

Arrays

Solución:

```
#include <stdio.h>

#define TAM 100

int main()
{
    /* Declaramos el array */
    int listaEnteros[TAM];

    /* Lo recorremos para rellenarlo con los 100 primeros números enteros */
    for (int i = 0; i < TAM; i++)
    {
        listaEnteros[i] = i + 1;
    }

    /* Orden ascendente */
    printf("Lista con los 100 primeros números enteros ordenados ascendentemente: \n");
    for (int i = 0; i < TAM; i++)
    {
        printf("%d - ", listaEnteros[i]);
    }

    /* Para que elimine el último guión*/
    printf("\b\b \n");

    /* Orden descendente */
    printf("Lista con los 100 primeros números enteros ordenados descendentemente: \n");
    for (int i = TAM - 1; i >= 0; i--)
    {
        printf("%d - ", listaEnteros[i]);
    }
    printf("\b\b ");

    return 0;
}
```

Arrays de char

Ya hemos comentado que en C no tenemos el tipo de dato String para almacenar una cadena de caracteres o texto.

Una forma de almacenar una cadena de caracteres es haciendo uso de arrays. Un char guarda un carácter. Un array de char guarda más de un carácter.

Ejemplo → `char cadena[10] = "Mi cadena";`

Problema: Tenemos que definir la longitud de la cadena teniendo en cuenta que el último carácter de la cadena es `"\0"`. Ese `\0` lo introduce C.

En el caso del ejemplo se guardaría esta información:

0	1	2	3	4	5	6	7	8	9
'M'	'i'	' '	'c'	'a'	'd'	'e'	'n'	'a'	'\0'

Arrays de char

Si en el ejemplo anterior el número de caracteres es inferior al tamaño reservado.

char cadena [10]= "Hola";

El array sigue ocupando el mismo espacio reservado.

Lo que hay a partir de '\0' se considera basura. No significa que estén vacías esas posiciones.

0	1	2	3	4	5	6	7	8	9
'H'	'o'	'l'	'a'	'\0'	' '	' '	' '	' '	' '

Arrays de char

```
#include <stdio.h>
#define TAM 5

int main() {
    char cadena [TAM]= "Hola";

    for (int i = 0; i < TAM; i++)
    {
        printf("Posicion %d --> %c\n",i, cadena[i]);
    }

    return 0;
}
```

Internamente guarda Hola\0

```
Posicion 0 --> H
Posicion 1 --> o
Posicion 2 --> l
Posicion 3 --> a
Posicion 4 -->
```

Recorremos el array completo, \0 no se imprime por pantalla pero está.

Arrays de char

Para imprimir por pantalla utilizamos el formato “%s”:

`printf(“%s”, cadena);`

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

int main() {
    char cadena [TAM]= "Hola";
    printf("---%s---\n", cadena);
    for (int i = 0; i < TAM; i++)
    {
        printf("Posicion %d --> %c\n",i, cadena[i]);
    }

    return 0;
}
```

Imprime justo hasta que se encuentra el \0.

Nuestro for sigue imprimiendo todos los elementos.

```
---Hola---
Posicion 0 --> H
Posicion 1 --> o
Posicion 2 --> l
Posicion 3 --> a
Posicion 4 --> 
Posicion 5 --> 
Posicion 6 --> 
Posicion 7 --> 
Posicion 8 --> 
Posicion 9 --> 
```

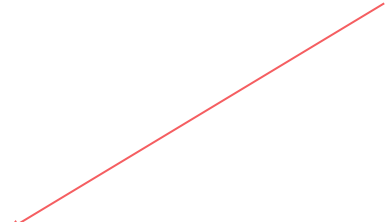
Arrays de char

Si cuento el tamaño sin acordarme del espacio del \0. El printf("%s") escribirá caracteres hasta que se encuentre el \0 que es el final de una cadena.

```
#include <stdio.h>
#define TAM 4

int main() {
    char cadena [TAM]= "Hola";
    printf("---%s---\n", cadena);
    for (int i = 0; i < TAM; i++)
    {
        printf("Posicion %d --> %c\n",i, cadena[i]);
    }

    return 0;
}
```



```
---HolaÇ---
Posicion 0 --> H
Posicion 1 --> o
Posicion 2 --> l
Posicion 3 --> a
```

Arrays de char

Observa que si TAM tiene un valor inferior a la cadena salta un warning y no se muestra bien la información.

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

int main() {
    char cadena [TAM]= "Hola";
    printf("---%s---\n", cadena);
    for (int i = 0; i < TAM; i++)
    {
        printf("Posicion %d --> %c\n",i, cadena[i]);
    }

    return 0;
}
```

```
arrayChar.c: In function 'main':
arrayChar.c:5:24: warning: initializer-string for array of chars is too long
    char cadena [TAM]= "Hola";
                        ^~~~~~
---HolÇ---
Posicion 0 --> H
Posicion 1 --> o
Posicion 2 --> l
```

Arrays de char

Para leer una cadena utilizamos la función ***gets(nombre_array)***. Esta cadena elimina el intro de la cadena (\n) y añade automáticamente \0 al final.

gets(cadena);

El problema de esta función es que si el operador carga más caracteres que los reservados en la variable produce errores inesperados.

Podemos hacer uso de otra función llamada fgets en la que le indicamos la longitud máxima a leer, incluido el \0:

fgets(cadena, longitud, stdin);

fflush(stdin); → Utiliza esta instrucción antes de leer una cadena, para borrar “\n” previos que se hayan pulsado por teclado previamente.

Arrays de char

Ejercicio: Solicita al usuario su nombre y apellidos. Nombre y apellidos serán guardados en un array de caracteres distinto cada uno. Imprime por pantalla al usuario sus nombres y apellidos en una sola línea. Tendrás que dar un tamaño adecuado a cada uno de los arrays.

Arrays de char

Solución:

```
#include <stdio.h>

int main()
{
    /*Reservamos espacio para dos cadenas*/
    char nombre[50];
    char apellidos[100];
    printf("Dime tu nombre\n");
    fflush(stdin); /*Limpiamos el buffer*/
    gets(nombre);
    printf("Dime tus apellidos\n");
    fflush(stdin); /*Limpiamos el buffer*/
    gets(apellidos);
    /*Imprimimos las dos cadenas juntas*/
    printf("Te llamas %s %s", nombre, apellidos);
    return 0;
}
```

Arrays de char

Ejercicio: Ingresar una palabra por teclado. Mostrar por pantalla la palabra y la cantidad de caracteres que tiene dicha palabra. Ojo: La cantidad de caracteres no tiene por qué coincidir con el tamaño que le hayas dado al array.

Arrays de char

Ejercicio: Ingresar una palabra por teclado. Mostrar por pantalla la palabra y la cantidad de caracteres que tiene dicha palabra.

```
#include <stdio.h>

int main()
{
    char palabra[40]; /* Suponemos que no va a introducir más de 40 caracteres*/
    printf("Ingrese una palabra:");
    gets(palabra);
    int x = 0;
    while (palabra[x] != '\0')
    {
        x++;
    }
    printf("La palabra %s tiene %i caracteres", palabra, x);
    return 0;
}
```

Arrays multidimensionales

Son aquellos que tienen más de una dimensión. Los más frecuentes son:

- **Bidimensionales:** De dos dimensiones. También llamados matrices.
- **Tridimensionales:** De tres dimensiones.

Cada dimensión tendrá un índice, que recordemos que es un número entero con el que se indica la posición de un elemento.

A la hora de dimensionar el array, se indicará cuántos elementos poseerá en cada dimensión.

El número de elementos totales será el resultante de multiplicar el número de elementos que admite cada índice.

Arrays multidimensionales

Esta podría ser la representación gráfica de un array bidimensional de 6 x 5 elementos de tipo entero:

<i>0,0</i>	<i>1,0</i>	<i>2,0</i>	<i>3,0</i>	<i>4,0</i>	<i>5,0</i>
int	int	int	int	int	int
<i>0,1</i>	<i>1,1</i>	<i>2,1</i>	<i>3,1</i>	<i>4,1</i>	<i>5,1</i>
int	int	int	int	int	int
<i>0,2</i>	<i>1,2</i>	<i>2,2</i>	<i>3,2</i>	<i>4,2</i>	<i>5,2</i>
int	int	int	int	int	int
<i>0,3</i>	<i>1,3</i>	<i>2,3</i>	<i>3,3</i>	<i>4,3</i>	<i>5,3</i>
int	int	int	int	int	int
<i>0,4</i>	<i>1,4</i>	<i>2,4</i>	<i>3,4</i>	<i>4,4</i>	<i>5,4</i>
int	int	int	int	int	int

Arrays multidimensionales

Los declaramos de la siguiente forma:

- Bidimensional: *tipoDato identificador [dimensión1][dimension2];*
- Tridimensional: *tipoDato identificador [dimensión1][dimension2][dimension3];*

```
//Array de float bidimensional 5x2
float ejemploArray2D [5][2];
//Array de enteros tridimensional 10x5x20
int ejemploArray3D [10][5][20];
//Declaración e inicialización
int ejemploArray2D [3][3] = {{2,4,1},{6,4,5},{5,2,1}};
```

Arrays multidimensionales

Para recorrer un array bidimensional o matriz necesitamos 2 bucles for.

```
#define TAM 6
int main()
{
    int datos[TAM][TAM] = {
        {1, 2, 3, 4, 5, 6},
        {7, 8, 9, 10, 11, 12},
        {13, 14, 15, 16, 17, 18},
        {19, 20, 21, 22, 23, 24},
        {25, 26, 27, 28, 29, 30},
        {31, 32, 33, 34, 35, 36}};

    int i,
        j;
    for (i = 0; i < TAM; i++)
    {
        for (j = 0; j < TAM; j++)
        {
            printf("Elemento [%d][%d]: %d\n", i, j, datos[i][j]);
        }
    }


    return 0;
}
```

```
Elemento [0][0]: 1
Elemento [0][1]: 2
Elemento [0][2]: 3
Elemento [0][3]: 4
Elemento [0][4]: 5
Elemento [0][5]: 6
Elemento [1][0]: 7
Elemento [1][1]: 8
Elemento [1][2]: 9
Elemento [1][3]: 10
Elemento [1][4]: 11
Elemento [1][5]: 12
Elemento [2][0]: 13
Elemento [2][1]: 14
Elemento [2][2]: 15
Elemento [2][3]: 16
Elemento [2][4]: 17
Elemento [2][5]: 18
Elemento [3][0]: 19
Elemento [3][1]: 20
Elemento [3][2]: 21
Elemento [3][3]: 22
Elemento [3][4]: 23
```


Composición de estructuras de datos

Un campo de un registro contener un array:

```
//Estructura Cliente
struct Cliente
{
    int codigo;
    char nombre[50];
};
```



Composición de estructuras de datos

Un campo de un registro puede ser de un tipo definido previamente como registro:

```
//Estructura Cliente
struct Cliente
{
    int codigo;
    char nombre[50];
};

//Estructura Factura que contiene un cliente
struct Factura
{
    int numeroFactura;
    struct Cliente unCliente;
};

//Inicialización de variables
struct Factura miFactura;
miFactura.numeroFactura = 1;
miFactura.unCliente.codigo = 1;

//Lectura e impresión por pantalla
gets(miFactura.unCliente.nombre);
printf("%s\n", miFactura.unCliente.nombre);
```

Estructura Cliente.

La estructura Factura contiene una variable de tipo estructura Cliente.

Ejemplo de asignación.

Ejemplo de lectura.

Ejemplo de impresión.

Composición de estructuras de datos

Array de registros: Cada uno de sus elementos tiene un tipo registro con campos de distintos tipos.

```
#define MAX_CLIENTES 1000
int main()
{
    // Estructura Factura que contiene un listado de productos
    typedef struct
    {
        int numeroFactura;
        int numProducto[20];
    } Factura;

    // Inicialización de variables
    Factura listaFacturas[MAX_CLIENTES];
    // Primer registro de mi array de facturas
    listaFacturas[0].numeroFactura = 1;
    listaFacturas[0].numProducto[0] = 15;
    listaFacturas[0].numProducto[1] = 21;
    listaFacturas[0].numProducto[1] = 3;

    return 0;
}
```

Array de tipo estructura Factura

Composición de estructuras de datos

Ejercicio:

Crea una estructura para guardar una fecha con día, mes y año.

Crea una estructura que guarde información de un cd de música:

Título, Nombre del artista, Número de canciones, Precio y Fecha de lanzamiento.

Crea un array de cds de música de tamaño 2.

Inicializa 2 cds con datos de ejemplo.

Recorre el array e imprime los datos de los CDs por pantalla.

Composición de estructuras de datos

Solución: Estructuras

```
typedef struct
{
    int dia;
    int mes;
    int anyo;
} Fecha;

typedef struct
{
    char titulo[30];
    char artista[25];
    int num_canciones;
    float precio;
    Fecha fecha;
} CD;
```

Composición de estructuras de

Solución: Creación del array e inicialización.

```
CD listaCD[2];
// Inicializo valores
int i;
for (i = 0; i < TAM; i++)
{
    printf("CD %d: \n", i);
    printf("Título: \n");
    gets(listaCD[i].titulo);

    printf("Artista: \n");
    gets(listaCD[i].artista);
    printf("Número de canciones: \n");
    fflush(stdin);
    scanf("%d", &listaCD[i].num_canciones);
    printf("Precio: \n");
    fflush(stdin);
    scanf("%d", &listaCD[i].precio);
    printf("Día: \n");
    fflush(stdin);
    scanf("%d", &listaCD[i].fecha.dia);
    printf("Mes: \n");
    fflush(stdin);
    scanf("%d", &listaCD[i].fecha.mes);
    printf("Año: \n");
    fflush(stdin);
    scanf("%d", &listaCD[i].fecha.anyo);
}
```

Composición de estructuras de datos

Solución: Impresión de los datos.

```
// Imprimo por pantalla los valores
for (i = 0; i < TAM; i++)
{
    printf("CD %d: \n", i);
    printf("Título: %s\n", listaCD[i].titulo);
    printf("Artista: %s\n", listaCD[i].artista);
    printf("Número de canciones: %d\n", listaCD[i].num_canciones);
    printf("Precio: %f\n", listaCD[i].precio);
    printf("Fecha: %d-%d-%d\n", listaCD[i].fecha.dia, listaCD[i].fecha.mes, listaCD[i].fecha.anyo);
}
```