A R R E G L O S:

V E C T O R E S , M A T R I C E S y S T R I N G S

¿Qué es un array?

Un array (arreglo) es un conjunto de variables del mismo tipo, las cuales se diferencian entre sí por su posición (ubicación), que se encuentran ordenadas a partir de un subíndice. Para ilustrarlo, supongamos una tabla dividida en celdas y cada celda es una variable en particular de esa colección, por supuesto que todas las celdas son del mismo tipo y pertenecen a la misma tabla.

Pero ¿qué quiere decir esto y para qué lo queremos?. Pues bien, supongamos que somos un metereólogo y queremos guardar en nuestra PC la temperatura que ha hecho cada hora del dia. Para darle cierta utilidad al final calcularemos el promedio de las temperaturas. Con lo que sabemos hasta ahora sería algo así:

```
#include <stdio.h>

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

                                                /* Ahora tenemos que dar el valor de cada una */
     printf("Temperatura de la hora  0:  ");
     scanf ("%i", &temp0 );
     printf("Temperatura de la hora  1:  ");
     scanf ("%i", &temp1 );
     printf("Temperatura de la hora  2:  ");
     scanf ("%i", &temp2 );
     ...
     printf("Temperatura de la hora 23:  ");
     scanf ("%i", &temp23);

     pro_media = (temp0 + temp1 + temp2 + temp3 + temp4 + ... + temp23) / 24.0;
     printf("\nLa temperatura promedio es %.1f\n", pro_media);
}
```

NOTA: Los puntos suspensivos los he puesto para no tener que escribir todo y que no ocupe tanto, no se pueden usar en un programa.

Para acortar un poco el programa podríamos hacer algo así:

```
#include <stdio.h>

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

                                                /* Ahora tenemos que dar el valor de cada una */
     printf("Introduzca las temperaturas desde las 0 hasta las 23:  ");
     scanf ("%i %i %i ... %i", &temp0, &temp1, &temp2, ... &temp23 );

     media = (temp0 + temp1 + temp2 + temp3 + temp4 + ... + temp23) / 24.0;
     printf("\nLa temperatura promedio es %.1f\n", pro_media);
}
```

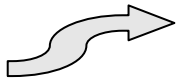
Lo que no deja de ser muy penoso. Y esto con un ejemplo que tiene tan sólo 24 variables, ¡¡imagínate si son más!!

Y precisamente aquí es donde nos vienen de maravilla los arrays. Vamos a hacer el programa con un array. Usaremos nuestros conocimientos de bucles y de scanf ( )...

Pero antes veamos algunos pormenores de lo que son los arreglos... recordemos la definición:

"Un array es un conjunto ó colección de variables del mismo tipo que tienen el mismo nombre y se diferencian entre sí por un subíndice".

int numeros [7] ;



int_1	int_2	int_3	int_4	int_5	int_6	int_7
[0]	[1]	[2]	[3]	[4]	[5]	[6]

Declaración de un Array

La forma general de declarar un array es la siguiente:

tipo_de_dato **nombre_del_array [dimensión] ;**

- ***tipo_de_dato*** es uno de los tipos de datos conocidos (int, char, float...), en el ejemplo era **int**.
- ***nombre_del_array*** es el nombre que damos al array, en el ejemplo era **temp**.
- ***[dimensión]*** es el número de elementos que tiene el array, en el ejemplo era **24**.

Como he indicado antes, al declarar un array reservamos en memoria bajo el (***nombre_del_array)*** tantas variables del (***tipo_de_dato)*** como las indicadas en ***[dimensión]***.

El nombre del arreglo es cualquier identificador válido y el tamaño es el tamaño que deseamos que tenga el arreglo, este siempre debe ir entre corchetes. Los datos del arreglo empiezan a introducirse por el dato[0]. Así, los siguientes arreglos los podemos representar:

```
char nums [10] = {20, 12, 13, 5, 50, 32, 27, 7, 18, 99};
char  nom [10] = {'H', 'O', 'L', 'A', ' ', 'M', 'U' ,'N', 'D', 'O'};
int     num  [5] = {22723, -405, 0, 75, -23000};
```

1 byte (char)

20	12	13	5	50	32	27	7	18	99
'H'	'O'	'L'	'A'	' '	'M'	'U'	'N'	'D'	'O'
22723		-405		0		75		-23000	

2 byte (int)

float res = num[0] * num[3];

1704225		

4 byte (float)

La cantidad máxima de elementos que puede albergar un arreglo está solo limitada por la cantidad de memoria disponible. Nótese que la cantidad de elementos de un array es **"n",** pero sus subíndices van de **[0]** (cero) a **[n-1].** Cualquier otro número usado como subíndice traerá datos de otras zonas de memoria cuyo contenido es impredecible. Se puede referenciar a cada elemento en forma individual, tal como se hace con cualquier variable. *Ejemplos:*

char letra [3] = { 'o', 'p', 'q'} ; ***donde:*** letra [0] = {'o'} ; letra [1] = {'p'} ; letra [2] = {'q'} ;

letra [0]	letra [1]	letra [2]
'o'	'p'	'q'

int var1 [10] ;

```
var1 [5] = 40 ;                          /* asigno un valor al quinto elemento del arreglo de 10 elementos */
contador = var1 [3] + 7 ;                /* uso el contenido del tercer elemento en una operación aritmética */
if (var1 [0] >= 37) printf(" ..... ") ;  /* uso el valor contenido en el primer elemento del arreglo en una condición */
```

También es posible utilizar como subíndice expresiones aritméticas, valores enteros retornados por funciones, etc. Así podríamos escribir:

```
printf (" %d ", var1 [ ++i] ) ;
var1 [8] = var1 [ i + j ] ;
```

La inicialización de los arrays sigue las mismas reglas que vimos para los otros tipos de variables, es decir: Si se declaran como globales (fuera del cuerpo de todas las funciones) cada uno de sus elementos será automaticamente inicializado a cero. Si en cambio, su declaración es local a una función, no se realiza ninguna inicialización, quedando a cargo del programa cargar los valores de inicio. La inicialización de un array local, puede realizarse en su declaración, dando una lista de valores iniciales:

```
int numero[8] = { 4 , 7 , 0 , 0 , 0 , 9 , 8 , 7 } ;
```

Obsérvese que la lista está delimitada por llaves. Otra posibilidad, sólo válida cuando se inicializan todos los elementos del array, es escribir:

```
int  numero[ ] = { 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 } ;
```

dónde se obvia la declaración de la cantidad de elementos, ya que está implícita en la lista de valores constantes. También se puede inicializar parcialmente un array, por ejemplo:

```
int numero[10] = { 1 , 1 , 1 };
```

en éste caso los tres primeros elementos del mismo valdrán 1, y los restantes cero en el caso que la declaración sea global, ó cualquier valor impredecible en el caso de que sea local .

Para poder introducir los datos en un arreglo debemos recorrerlo, o sea, ir a cada celda del arreglo y ahí escribir el dato. Para esto se usa frecuentemente un ciclo *for* ... podemos ahora retomar el ejemplo del meteorólogo:

```
#include <stdio.h>

int main()
{
     int temp[24];                          /* Con esto ya tenemos declaradas las 24 variables */
     float pro_media = 0.0;
     int hora;

                                                 /* Ahora tenemos que dar el valor de cada una */
     for(hora=0; hora<24; hora++)
     {
          printf("Temperatura de la hora %i:  ", hora);
          scanf ("%i", &temp[hora]);
          pro_media += temp[hora];
     }
     pro_media = pro_media / 24;

     printf("\nLa temperatura promedio del dia es %.1f\n", pro_media);
}
```

Como vemos es un programa más rápido de escribir (y es menos aburrido hacerlo), y más cómodo para el usuario que el anterior.

El primer elemento de un vector en ***C*** tiene el índice [0], el segundo tiene el [1] y así sucesivamente. De modo que si queremos dar un valor al cuarto elemento (índice [3]) haremos:

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

NOTA: No hay que confundirse. En la declaración de un array el número entre corchetes es el número de elementos, en cambio cuando ya usamos el arreglo el número entre corchetes es el subíndice.

Continuamos trabajando con el ejemplo de las temperaturas de las 24 horas de un día:

Los arreglos al igual que las otras variables se pueden inicializar, para hacerlo se prosigue de la siguiente manera:

```
int temperaturas[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 };
```

De esta manera hemos inicializado el arreglo con los valores correspondientes.

Así: temperatura[0] tiene el valor 15, temperatura[1] tiene el valor 18 y así sucesivamente.

Sobre la dimensión de un Array

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 con nombre, por ejemplo ELEMENTOS.

```
#include <stdio.h>

#define  ELEMENTOS  24

int main()
{
     int temp[ELEMENTOS];                  /* Con esto ya tenemos declaradas las 24 variables */
     float pro_media = 0.0;
     int hora;

                                                /* Ahora tenemos que dar el valor de cada una */
     for( hora=0; hora<ELEMENTOS; hora++ )
     {
          printf( "Temperatura de las %i: ", hora );
          scanf( "%i", &temp[hora] );
          pro_media += temp[hora];
     }
     pro_media = pro_media / ELEMENTOS;

     printf( "\nLa temperatura promedio es %.1f\n", pro_media );
     return 0;
}
```

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

Por otra parte si no especificamos la cantidad de elementos con que declaramos el vector de enteros, nos veremos en la obligación de efectuar la inicialización del mismo con valores que serán leidos por el compilador y que considerará a estos como la capacidad del arreglo, reservando espacio en memoria solo para esa cantidad de elementos.

```
#include <stdio.h>          /* ejemplo del uso de sizeof para determinar tamaño de un erreglo */

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

     elementos = sizeof temperaturas / sizeof(int);

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

STRINGS (CADENAS DE CARACTERES)

Vamos a ver por fin como manejar texto con C, hasta ahora solo sabiamos como mostrarlo por pantalla. Para empezar diremos que en C no existe un tipo string como en otros lenguajes. No existe un tipo de datos para almacenar texto, se utilizan arrays de chars. Funcionan igual que los demás arrays con la diferencia que ahora jugamos con letras en vez de con numeros. Se les llama "cadenas de texto", "strings" o "cadenas de caracteres". Los strings son simplemente arreglos de caracteres, tal como los vimos hasta ahora, con el agregado de un ultimo elemento constante: el caracter NULL (ASCII==0, simbolizado por la secuencia de escape '\0'); este agregado permite a las funciones que los procesan determinar facilmente la finalizacion de los datos. A partir de ahora les llamaremos cadenas. Para declarar una cadena se hace como un array:

```
/* declaracion de un string de 20 caracteres sin inicializar */

        char texto[20];

/* ERROR: declaracion de un string sin definir cantidad de elementos ni inicializar */

        char texto[ ];

/* declaraciones e inicializaciones*/

        char abc[ ] = { 'A', 'B', 'C', 'D', 0};
        char ABC[ ] = {'a', 'b', 'c', 'd', '\0'};
        char abc[ ] = "ABCD";
        char ABC[5] = "abcd";
```

Nótese la equivalencia entre el 0 del tercer caso y la secuencia de escape '\0' del cuarto. Existe, sin embargo, una forma más compacta de declararlos e inicializarlos: quinto y sexto casos; simplemente en la declaración de los mismos se encierran los caracteres que los componen entre comillas dobles, estas inicializaciones agregan en forma automática el NULL como último elemento del string. Al igual que en los arrays no podemos meter más de 20 elementos en la cadena que declaramos para 20 elementos. *Por ejemplo:*

```
void main ( )
{
    char nombre[20] ;
    printf ( "Introduzca su nombre (19 letras máximo):   ") ;                    /* son 19 + '\0' (NULL) */
    scanf ( "%s", nombre) ;

    printf ( "\nEl nombre que ha escrito es: %s\n", nombre) ;
}
```

Vemos cosas curiosas como por ejemplo que en el scanf ( ) no se usa el símbolo '&'. No hace falta porque es un array, y escribir el nombre del array es equivalente a poner: &nombre[0].
También puede llamar la atención la forma de imprimir el array. Con sólo usar **"%s"** ya se imprime todo el array. Ya veremos esto más adelante.

Inicialización de una Cadena

```
void main ( ) {
   char  nombre[ ] = "Esteban Quito";
   printf( "Texto: %s\n", nombre);
   printf( "Tamaño de la cadena: %i bytes\n", sizeof (nombre));
}                                          /* sizeof (nom_array);  funcion que nos devuelve la cantidad de caracteres que posee un string */
```

Como vemos la inicialización es parecida a la de un arreglo, solo que aquí se escribe toda la frase para inicializarla. Cuando inicializamos una cadena, *C* automáticamente introduce como ultimo elemento el '\0' que indica fin de la cadena. Esto sirve para que a la hora de imprimir, *C* sepa cuando debe parar de imprimir.
Es importante no olvidar que la longitud de una cadena es la longitud del texto más el símbolo de fin de cadena. Por eso cuando definamos una cadena tenemos que reservarle un espacio adicional.

ENTRADA DE CADENAS DE CARACTERES POR EL TECLADO

La función *scanf( )*

Hemos visto en capítulos anteriores el uso de scanf() para números, ahora es el momento de ver su uso con cadenas. La función scanf() almacena en memoria (en un buffer) lo que vamos escribiendo. Cuando pulsamos ENTER lo analiza, comprueba si el formato es correcto y por último lo mete en la variable que le indicamos. scanf() toma una palabra como cadena. Usa los espacios para separar variables. Así, si introducimos una palabra, damos espacio y escribimos otra, scanf() solo almacenará la primea palabra. Es importante siempre asegurarse de que no vamos a almacenar en cadena más letras de las que caben. Para ello debemos limitar el número de letras que le va a introducir scanf().

La función *gets( )*

Esta función nos permite introducir frases enteras, incluyendo espacios. Almacena lo que vamos tecleando en la variable buffer hasta que pulsamos ENTER. Si se ha almacenado algún caracter en buffer le añade un **'\0'** al final y devuelve un puntero a su dirección. Si no se ha almacenado ninguno devuelve un puntero **NULL**. Por ejemplo:

```
void main ( ) {
   char cadena[30] ;
   printf ( "Escriba una frase:  " ) ;
   gets ( cadena ) ;
   if (cadena[0] != NULL)  printf ( "He guardado: \"%s\" \n", cadena ) ;
   else   printf ( "No he guardado nada!\n" ) ;
}
```

Qué son los buffer? un **buffer de datos** es una porción de memoria en una computadora o en un instrumento digital reservada para el almacenamiento temporal de información digital, mientras que está esperando ser procesada.

```
#include <stdio.h>
void main ( ) {
    char ch ;
    char nombre[20] , apellido[20] , telefono[10] ;
    printf ( "Escribe tu nombre: " ) ;
    scanf ( "%[A-Z]s", nombre ) ;
    printf ( "Lo que leemos con scanf ( ) es: %s\n", nombre ) ;
    printf ( "Lo que quedo en el buffer: " ) ;
    while ( (ch = getchar( )) != '\n' )    printf ( "%c", ch ) ;
}
```

// En este ejemplo el [A-Z] lo que hace es que solamente leer las letras mayúsculas. Y la salida sería:

Escribe tu nombre: **PABLito**
Lo que leemos con scanf () es: **PABL**
Lo que quedo en el buffer: **ito**

ALGUNAS FUNCIONES PARA EL MANEJO DE STRING

Si bien más adelante vamos a dedicar un capítulo especialmente dedicado a este tema, podemos aquí anticipar algunas de las funciones básicas para el manejo de cadenas de caracteres.
Es imposible efectuar asignaciones directas entre cadenas, pero esto puede solucionarse utilizando una función de la librería standard <string.h>: strcpy (nombre_string_destino , nombre_string_origen);

Ejemplo:

```
char  nombre[9] ;
char persona[ ] = "Jorge";
nombre = persona ;                                  /* INCORRECTO */
strcpy (nombre , persona) ;                         /* CORRECTO */
```

Simplemente, lo que hace la función es copiar un string sobre el otro, elemento a elemento, hasta que detecta que ha copiado el carácter NULL, en ese caso finaliza el proceso de copiado.

```
void main ( ) {
    char a [8] ;
    char b [ ] = "palabra";
    int  i = 0;

    while (( a [ i ] = b [ i ] ) != '\0')    i++;
    printf (" a [ ] = \" %s \" y b [ ] = \" %s \" ", a , b) ;
}
```

Podemos mencionar otras funciones como:
strcmp (str1 , str2) ; que compara dos strings, devolviendo 0 (cero) si son iguales y distinto de 0 (cero) si son distintos.
strcat (str1 , str2); que agrega (concatena) el string 2 al final del string 1.

ARREGLOS DE CADENAS DE CARACTERES (introducción a las matrices)

Un array de cadenas puede servirnos para agrupar una serie de mensajes. Por ejemplo todos los nombres de los días de la semana. Luego para acceder a cada mensaje basta con usar su número. Así:

```
#include <stdio.h>
#include <conio.h>
```

```
void main ( ) {
   int num_dia , i ;
   char  dia_semana [7][9] = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"} ;
   do {
      do {
         printf ("Si ingresa cero(0) sale del programa…\nIngrese un numero de uno(1) a siete(7) para ver el día:   " ) ;
         scanf ("%d", &num_dia) ;
      }
      while ((num_dia < 0) || (num-dia > 7)) ;
      if (num_dia == 0)    printf ("Se termina la ejecución.") ;
      else  {
         printf ("El día es:  ") ;
         for (i = 0; i < 9; i++)    printf (" %c", dia_semana [num_dia - 1] [i]) ;
      }
      getch ( ) ;
      clrscr ( ) ;
   }
   while (num_dia  !=  0) ;
}
```

Un array de cadenas es en realidad una matriz de dos dimensiones. El primer elemento de la cadena ("Lunes") tiene un espacio reservado en memoria y dia_semana[0] apunta a ese espacio. Es decir: en la matriz dia_semana[7][9]; el primer elemento es dia_semana[0][0]; y su contenido es 'L'.

ARRAYS MULTIDIMENCIONALES: Los arrays de dos dimensiones (matriz) toman la forma:

tipo_dato nombre_arreglo [filas] [columnas]; Ej.: int matriz[3][4]; donde:

Fila – Columna ↓ →	0	1	2	3	
0	[0][0]	[0][1]	[0][2]	[0][3]	matriz[0][ ]
1	[1][0]	[1][1]	[1][2]	[1][3]	matriz[1][ ]
2	[2][0]	[2][1]	[2][2]	[2][3]	matriz[2][ ]
	matriz[ ][0]	matriz[ ][1]	matriz[ ][2]	matriz[ ][3]	

Ejemplos:

```
char palabras[3][4] = {{'p','a','p','a'}, {'n','e','n','a'}, {'m','a','m','i'}} ;
char numeros[3][3] = {{127, 29, 35}, {48, 51, 67}, {17, 98, 109}} ;
int  identidad[4][4] = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}} ;
float decimal[2][2] = {{5.2, 6.9}, {1.5, 7.4}} ;                    /* declaraciones e inicializaciones */

#include<stdio.h>
#include <conio.h>

void main( ) {

    char mat[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}} ;
    char fil ;

    for ( fil = 0; fil < 3;  fil++) {
        printf ("\n\n\t") ;                                          /* mostrar en formato matricial */
        for (char col = 0; col < 3; col++) {
            printf(" %d ", mat[fil][col]) ;
        }
    }
    printf("\n\n\t\t") ;
    getch () ;
```

```
   for ( fil = 0; fil < 3; fil++)    printf(" %d", mat[fil][fil]) ;                    /* mostrar la diagonal principal */

   col = 2 ;
   printf ("\n\n\t\t\t") ;


   for ( fil = 0; fil < 3; fil++)  {
       printf (" %d", mat[fil][col]) ;
       col-- ;                                                                      /* mostrar la diagonal opuesta */
   }
   getch () ;


   for ( fil = 0; fil < 3; fil++) {
       printf ("\n\n\t") ;                                                  /* para mostrar en formato matricial */
       for ( col = 0;  col < 3; col++) {
            mat[fil][col] = 0 ;                                             /* inicializar la matriz en cero*/
           printf (" %d ", mat[fil][col]) ;                                 /* mostrar la matriz */
       }
   }
}
```

Solo resta agregar que *C* nos permite trabajar con matrices de más de tres dimensiones (matrices de matrices), pero dichas matrices presentan una obvia dificultad a la hora de su visualización en nuestro entendimiento. Por ahora trataremos el tema solo con esta mención, ya que no es el objetivo en este momento introducirnos en un tema tan profundo del Lenguaje de Programación **ANSI C**.