# Hay novedades…

Recuerdan al empleado programador?

Tuvo mucho éxito con sus soluciones computacionales, también lo ascendieron y le dieron más trabajo de desarrollo de programas para resolver cuestiones de la empresa.

Pero en los últimos días, en el departamento de ventas, surge un problema y necesitan saber cuánto vendió cada vendedor en el último trimestre y además necesitan las ventas totales de cada uno de esos 3 meses… Entonces le solicitan al hábil empleado, que construya un programa que realice esos cálculos….

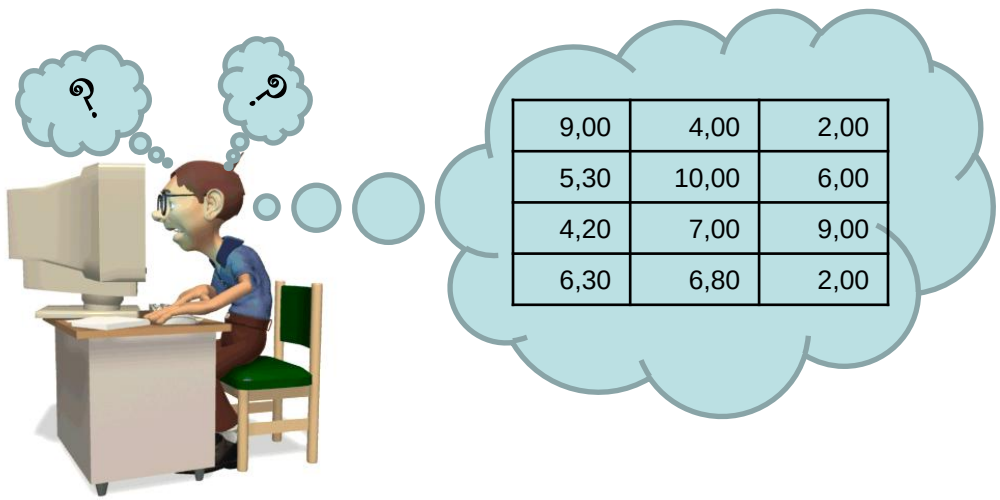
Le dan los datos de 4 vendedores, sus ventas por mes y luego de analizar todo llega a la conclusión…..

… que para 4 vendedores tenemos las ventas mensuales durante 3 meses, en un formato tipo tabla de doble entrada:

| | *Mes_1* | *Mes_2* | *Mes_3* | **Totales por vendedor** |
|---|---|---|---|---|
| ***Vend_1*** | 9,00 | 4,00 | 2,00 | **15,00** |
| ***Vend_2*** | 5,30 | 10,00 | 6,00 | **21,30** |
| ***Vend_3*** | 4,20 | 7,00 | 9,00 | **20,20** |
| ***Vend_4*** | 6,30 | 6,80 | 2,00 | **15,10** |
| **Totales por mes** | **24,80** | **27,80** | **19,00** | |

Ahora bien,...es evidente que para hacer esos cálculos necesita todos los datos en memoria. Lo que conoce y sabe si bien puede resolver este problema en particular, presenta complicaciones en la implementación. También sabe que hay más vendedores y que pueden pedirle la misma información sobre más tiempo…

Existe alguna estructura en memoria que permita manejar datos por fila y por columna?...



# Arrays

## Arrays Bidimensionales en C

**Sabemos que:**

Un array es una estructura de datos que contiene datos de un mismo tipo, organizados de un modo particular.

Es posible tratar un conjunto de datos del mismo tipo como si estuviesen dispuestos en una tabla (2 dimensiones) o apilados por capas en 3 dimensiones, o en más, aunque en la memoria se encuentren en forma consecutiva.

## Indexación en un array bidimensional:

| | *columna_0* | | *columna_1* | | *columna_2* | |
|---|---|---|---|---|---|---|
| ***fila_0*** | 9,00 | | 4,00 | | 2,00 | |
| | ***fila_0*** | ***columna_0*** | ***fila_0*** | ***columna_1*** | ***fila_0*** | ***columna_2*** |
| ***fila_1*** | 5,30 | | 10,00 | | 6,00 | |
| | ***fila_1*** | ***columna_0*** | ***fila_1*** | ***columna_1*** | ***fila_1*** | ***columna_2*** |
| ***fila_2*** | 4,20 | | 7,00 | | 9,00 | |
| | ***fila_2*** | ***columna_0*** | ***fila_2*** | ***columna_1*** | ***fila_2*** | ***columna_2*** |
| ***fila_3*** | 6,30 | | 6,80 | | 2,00 | |
| | ***fila_3*** | ***columna_0*** | ***fila_3*** | ***columna_1*** | ***fila_3*** | ***columna_2*** |

## Declaración (primera forma) :

**<tipo de dato> <nombre_array> [1°dimensión-filas][2°dimensión-columnas];**

**Ejemplos:**

int arrayB [15 ][ 7 ];

float arrayB [ 5 ][ 10 ];

double arrayB [ 100 ][ 100 ];

La matriz del ejemplo puede declararse como:

**float ventas [ VENDEDOR ][ MES ]**

lo que equivale a decir:

**float ventas [ 4 ][ 3 ];**

**Es decir que tenemos 2 índices para operar sobre este tipo de arrays**

## Inicialización:

Puede realizarse en el momento de la declaración:

```
float ventas [ 4 ][ 3 ] = {7,9,15,13,11,8,5,9,10,2,8,4};
```

o bien

```
float ventas [ 4 ][ 3 ] = {{7,9,15},{13,11,8},{5,9,10},{2,8,4}};
```

Si se omitieran valores en la inicialización, el compilador setea con valor '0' el resto :

```
float ventas[ 4 ][ 3 ] = {7,9,15};
```

**NO** se puede inicializar todos los elementos de un array en una línea diferente a la de la declaración:

```
float ventas [ 4 ][ 3 ];
ventas  = {7,9,15,13,11,8,5,9,10,2,8,4}; //error
```

**NO** se puede inicializar un array con más elementos de los declarados en la dimensión :

```
float ventas [ 4 ][ 3 ] = {7,9,15,13,11,8,5,25,2,9,10,4,3,6,7}; //error
```

**Inicialización:**

Otra forma de asignar valores es:

```
ventas [ 0 ][ 0 ] = 9.00;
ventas [ 0 ][ 1 ] = 4.00;
ventas [ 0 ][ 2 ] = 2.00;
ventas [ 1 ][ 0 ] = 5.30;
ventas [ 1 ][ 1 ] = 10.00;
ventas [ 1 ][ 2 ] = 6.00;
                  …..
```

Pero es un método poco práctico.

**Declaración (segunda forma):**

**<tipo de dato>  <nombre_array>  [  ] [2°dimensión-columnas];**

**Ejemplo**: ventas [ ][ 3 ];

**Inicialización:**

Se puede omitir la primera dimensión si se inicializa en la declaración, es decir que en este caso la inicialización es forzosa:

**<tipo de dato>< nombre_array>  [ ][ 3 ] = {valor1, valor2,…};**

Ejemplo:  float ventas [ ][ 3 ] = {7,9,15,13,11,8,5,6,7,1,3,8};

**El array toma la dimensión de la cantidad de elementos.**

# Cómo accedemos a los elementos de un array bidimensional con notación de subíndices?

## Ingreso de datos:

```
scanf("%f", &ventas[ 0 ] [ 0 ]);
```

Posición del elemento en el array

9

Ingreso del valor por teclado en la **posición: fila 0, columna 0**

```
…
scanf("%f", &ventas[ 0 ][ 1 ]);
scanf("%f", &ventas[ 0 ][ 2 ]);
…?
```

## Ingreso de datos:

```
…
for (i = 0; i < filas; i++) {
  for(j = 0; j < columnas; j++)
     {  printf("Ingrese la venta  del vendedor :%d, mes: %d\n: ", i, j);
        scanf("%f", &ventas[ i ][ j ]);     }
} …
```

Primera dimensión (filas)

Segunda dimensión (columnas)

| | *columna_0* | *columna_1* | *columna_2* |
|---|---|---|---|
| | 9,00 | 4,00 | 2,00 |
| *fila_0* | **ventas[ 0 ][ 0 ]** | **ventas[ 0 ][ 1 ]** | **ventas[ 0 ][ 2 ]** |
| | 5,30 | 10,00 | 6,00 |
| *fila_1* | **ventas[ 1 ][ 0 ]** | **ventas[ 1 ][ 1 ]** | **ventas[ 1 ][ 2 ]** |
| | 4,20 | 7,00 | 9,00 |
| *fila_2* | **ventas[ 2 ][ 0 ]** | **ventas 2 ][ 1 ]** | **ventas[ 2 ][ 2 ]** |
| | 6,30 | 6,80 | 2,00 |
| *fila_3* | **ventas [ 3 ][ 0 ]** | **ventas[ 3 ][ 1 ]** | **ventas[ 3 ][ 2 ]** |

## Ingreso de datos:

| | *columna_0* | *columna_1* | *columna_2* |
|---|---|---|---|
| *fila_0* | 9,00 | 4,00 | 2,00 |
| | **ventas[ 0 ][ 0 ]** | **ventas[ 0 ][ 1 ]** | **ventas[ 0 ][ 2 ]** |
| *fila_1* | 5,30 | 10,00 | 6,00 |
| | **ventas[ 1 ][ 0 ]** | **ventas[ 1 ][ 1 ]** | **ventas[ 1 ][ 2 ]** |
| *fila_2* | 4,20 | 7,00 | 9,00 |
| | **ventas[ 2 ][ 0 ]** | **ventas 2 ][ 1 ]** | **ventas[ 2 ][ 2 ]** |
| *fila_3* | 6,30 | 6,80 | 2,00 |
| | **ventas [ 3 ][ 0 ]** | **ventas[ 3 ][ 1 ]** | **ventas[ 3 ][ 2 ]** |

| **Descripción del array o vector del ejemplo** | |
|---|---|
| Nombre del array bidimensional | ventas |
| Tipo de datos que guarda | float |
| Dimensión , tamaño, cantidad de elementos que puede guardar | 12 (4 x 3) |
| Contenido o valor o elemento de la posición [0][ 0] | 9.00 |
| Contenido o valor o elemento de la posición [1][ 2] | 6.00 |
| Contenido o valor o elemento de la posición [2] [1] | 7.00 |
| Contenido o valor o elemento de la posición [3] [2] | 2.00 |
| Contenido o valor o elemento de la posición [3] [3] | **Error** |
| Elemento 10.00 está en la posición | [1][1] |
| Elemento 6.30 está en la posición | [3][0] |
| Elemento 2.00 está en la posición | [0][2] y [3][2] |

## Acceso a los elementos:

```
…
for (i = 0; i < filas; i++) {
   for(j = 0; j < columnas; j++)
      {  printf("La venta del vendedor %d, mes %d , es: \n: ", i, j );
         printf("%.2f", ventas[ i ][ j ]);   }
} …
```

Primera dimensión (filas)

Segunda dimensión (columnas)

| | *columna_0* | *columna_1* | *columna_2* |
|---|---|---|---|
| *fila_0* | 9,00 | 4,00 | 2,00 |
| | **ventas[ 0 ][ 0 ]** | **ventas[ 0 ][ 1 ]** | **ventas[ 0 ][ 2 ]** |
| *fila_1* | 5,30 | 10,00 | 6,00 |
| | **ventas[ 1 ][ 0 ]** | **ventas[ 1 ][ 1 ]** | **ventas[ 1 ][ 2 ]** |
| *fila_2* | 4,20 | 7,00 | 9,00 |
| | **ventas[ 2 ][ 0 ]** | **ventas 2 ][ 1 ]** | **ventas[ 2 ][ 2 ]** |
| *fila_3* | 6,30 | 6,80 | 2,00 |
| | **ventas [ 3 ][ 0 ]** | **ventas[ 3 ][ 1 ]** | **ventas[ 3 ][ 2 ]** |

# Arrays bidimensionales y su relación con punteros

## Arrays y direcciones de memoria:

Tal como sucede con los arrays unidimensionales, según nuestro ejemplo , la declaración '**ventas' corresponde a la dirección de comienzo de la matriz** ventas [ 4][ 3 ].

**En C una matriz se maneja como un array de arrays.**

Eso hace que sea válido usar el **nombre de la matriz con un solo subíndice para indicar la dirección de comienzo de cada fila**. Así,

**ventas identifica el array de direcciones de filas.**

**y**

**ventas = ventas [ 0 ] = & ventas[ 0 ][ 0 ]**

**ventas + 1 = ventas [ 0 ] + 1= &ventas[ 0 ][ 1]**

**La dirección de cada fila i es ventas[ i ] con un solo subíndice**

# Arrays y direcciones de memoria:

Un array bidimensional ocupa posiciones contiguas de memoria, de acuerdo a ello su estructura em memoria sería:

| 9,00 | 4,00 | 2,00 | 5,30 | 10,00 | 6,00 | 4,20 | 7,00 | 9,00 | 6,30 | 6,80 | 2,00 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 60FED4 | 60FED8 | 60FEDC | 60FEE0 | 60FEE4 | 60FEE8 | 60FEEC | 60FEF0 | 60FEF4 | 60FEF8 | 60FEFC | 60FF00 |

Fila 0 comienza en **60FED4** y con nuestro ejemplo corresponde a **ventas[ 0 ]**

Fila 1 comienza en **60FEE0** y con nuestro ejemplo corresponde a **ventas[ 1 ]**

Fila 2 comienza en **60FEEC** y con nuestro ejemplo corresponde a **ventas[ 2 ]**

Fila 3 comienza en **60FEF8** y con nuestro ejemplo corresponde a **ventas[ 3 ]**

Considerando el formato de tabla puede verse de esta manera:

| 9,00 | 4,00 | 2,00 |
|---|---|---|
| 60FED4 | 60FED8 | 60FEDC |
| 5,30 | 10,00 | 6,00 |
| 60FEE0 | 60FEE4 | 60FEE8 |
| 4,20 | 7,00 | 9,00 |
| 60FEEC | 60FEF0 | 60FEF4 |
| 6,30 | 6,80 | 2,00 |
| 60FEF8 | 60FEFC | 60FF00 |

Es decir:

| | columna_0 *mes_1* | columna_1 *mes_2* | columna_3 *mes_3* |
|---|---|---|---|
| fila_0 *vend_1* | 9,00 | 4,00 | 2,00 |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1 *vend_2* | 5,30 | 10,00 | 6,00 |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2 *vend_3* | 4,20 | 7,00 | 9,00 |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3 *vend_4* | 6,30 | 6,80 | 2,00 |
| | 60FEF8 | 60FEFC | 60FF00 |

# Arrays y direcciones de memoria:

De lo visto anteriormente se deduce que invocando **sólo la primera dimensión** obtenemos la dirección memoria del comienzo de cada fila.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9,00 | 4,00 | 2,00 | 5,30 | 10,00 | 6,00 | 4,20 | 7,00 | 9,00 | 6,30 | 6,80 | 2,00 |
| 60FED4 | 60FED8 | 60FEDC | 60FEE0 | 60FEE4 | 60FEE8 | 60FEEC | 60FEF0 | 60FEF4 | 60FEF8 | 60FEFC | 60FF00 |

| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* |
|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 |
| | ventas[ 0 ] | | |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 |
| | ventas[ 1 ] | | |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 |
| | ventas[ 2 ] | | |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 |
| | ventas[ 3 ] | | |
| | 60FEF8 | 60FEFC | 60FF00 |

# Arrays y direcciones de memoria:

Ahora bien, sabemos que cada dirección de memoria podemos obtenerla escribiendo **&<nombre_array>[1°d][2°d],** en nuestro ejemplo :

| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* |
|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 |
| | &ventas[ 0 ][ 0 ] | &ventas[ 0 ][ 1 ] | &ventas[ 0 ][ 2 ] |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 |
| | &ventas[ 1 ][ 0 ] | &ventas[ 1 ][ 1 ] | &ventas[ 1 ][ 2 ] |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 |
| | &ventas[ 2 ][ 0 ] | &ventas[ 2 ][ 1 ] | &ventas[ 2 ][ 2 ] |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 |
| | &ventas[ 3 ][ 0 ] | &ventas[ 3 ][ 1 ] | &ventas[ 3 ][ 2 ] |
| | 60FEF8 | 60FEFC | 60FF00 |

# Arrays y direcciones de memoria:

Sintetizando, de acuerdo a nuestro ejemplo podemos decir que:

**ventas = ventas[ 0 ] = &ventas[ 0 ] [ 0 ]**

| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* | |
|---|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 | |
| | &ventas[ 0 ][ 0 ] | &ventas[ 0 ][ 1 ] | &ventas[ 0 ][ 2 ] | } = |
| | ventas[ 0 ] | ventas[ 0 ] + 1 | ventas[ 0 ] + 2 | |
| | 60FED4 | 60FED8 | 60FEDC | |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 | |
| | &ventas[ 1 ][ 0 ] | &ventas[ 1 ][ 1 ] | &ventas[ 1 ][ 2 ] | } = |
| | ventas[ 1 ] | ventas[ 1 ] + 1 | ventas[ 1 ] + 2 | |
| | 60FEE0 | 60FEE4 | 60FEE8 | |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 | |
| | &ventas[ 2 ][ 0 ] | &ventas[ 2 ][ 1 ] | &ventas[ 2 ][ 2 ] | } = |
| | ventas[ 2 ] | ventas[ 2 ] + 1 | ventas[ 2 ] + 2 | |
| | 60FEEC | 60FEF0 | 60FEF4 | |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 | |
| | &ventas[ 3 ][ 0 ] | &ventas[ 3 ][ 1 ] | &ventas[ 3 ][ 2 ] | } = |
| | ventas[ 3 ] | ventas[ 3 ] + 1 | ventas[ 3 ] + 2 | |
| | 60FEF8 | 60FEFC | 60FF00 | |

# Arrays y direcciones de memoria:

Si sabemos que **ventas = ventas[ 0 ] = &ventas[ 0 ] [0 ],** observando el ejemplo, de fila en fila, podemos deducir que **ventas + 0 = ventas [ 0 ], ventas + 1 = ventas [1], ventas + 2 = ventas [2]** … y los valores los estaría representando el índice de fila, si usamos la variable i como índice, tendríamos: **ventas + i**

| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* |
|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 |
| = | &ventas[ 0 ][ 0 ] | &ventas[ 0 ][ 1 ] | &ventas[ 0 ][ 2 ] |
| = | ventas[ 0 ] | ventas[ 0 ] + 1 | ventas[ 0 ] + 2 |
| = | ventas + 0 | | |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 |
| = | &ventas[ 1 ][ 0 ] | &ventas[ 1 ][ 1 ] | &ventas[ 1 ][ 2 ] |
| = | ventas[ 1 ] | ventas[ 1 ] + 1 | ventas[ 1 ] + 2 |
| = | ventas + 1 | | |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 |
| = | &ventas[ 2 ][ 0 ] | &ventas[ 2 ][ 1 ] | &ventas[ 2 ][ 2 ] |
| = | ventas[ 2 ] | ventas[ 2 ] + 1 | ventas[ 2 ] + 2 |
| = | ventas + 2 | | |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 |
| = | &ventas[ 3 ][ 0 ] | &ventas[ 3 ][ 1 ] | &ventas[ 3 ][ 2 ] |
| = | ventas[ 3 ] | ventas[ 3 ] + 1 | ventas[ 3 ] + 2 |
| = | ventas + 3 | | |
| | 60FEF8 | 60FEFC | 60FF00 |

# Arrays y direcciones de memoria:

Vimos que con **ventas + i** podemos recorrer las filas. Cómo accedemos a las columnas?, si sabemos que la primera dimensión nos devuelve una dirección y con la segunda dimensión tenemos que ubicarnos en las columnas, entonces, observando el ejemplo y teniendo en cuenta una variable como índice de columna que podría ser j, podemos decir que : *** (ventas + i) + j**, nos posicionaría en la dirección del elemento.

| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* |
|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 |
| = | &ventas[ 0 ][ 0 ] | &ventas[ 0 ][ 1 ] | &ventas[ 0 ][ 2 ] |
| | ventas[ 0 ] | ventas[ 0 ] + 1 | ventas[ 0 ] + 2 |
| | *(ventas + 0) + 0 | *(ventas + 0) + 1 | *(ventas + 0) + 2 |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 |
| = | &ventas[ 1 ][ 0 ] | &ventas[ 1 ][ 1 ] | &ventas[ 1 ][ 2 ] |
| | ventas[ 1 ] | ventas[ 1 ] + 1 | ventas[ 1 ] + 2 |
| | *(ventas + 1) + 0 | *(ventas + 1) + 1 | *(ventas + 1) + 2 |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 |
| = | &ventas[ 2 ][ 0 ] | &ventas[ 2 ][ 1 ] | &ventas[ 2 ][ 2 ] |
| | ventas[ 2 ] | ventas[ 2 ] + 1 | ventas[ 2 ] + 2 |
| | *(ventas + 2) + 0 | *(ventas + 2) + 1 | *(ventas + 2) + 2 |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 |
| = | &ventas[ 3 ][ 0 ] | &ventas[ 3 ][ 1 ] | &ventas[ 3 ][ 2 ] |
| | ventas[ 3 ] | ventas[ 3 ] + 1 | ventas[ 3 ] + 2 |
| | *(ventas + 3) + 0 | *(ventas + 3) + 1 | *(ventas + 3) + 2 |
| | 60FEF8 | 60FEFC | 60FF00 |

**Podemos acceder a los elementos de un array bidimensional con notación de punteros?**

# Arrays y direcciones de memoria:

Si *** (ventas + i) + j**, nos posiciona en la dirección del elemento, cómo accedemos al elemento?. Como hemos visto anteriormente, accedemos con el operador *, entonces para acceder al elemento con notación de punteros :

*** ( * ( ventas + i ) + j )**

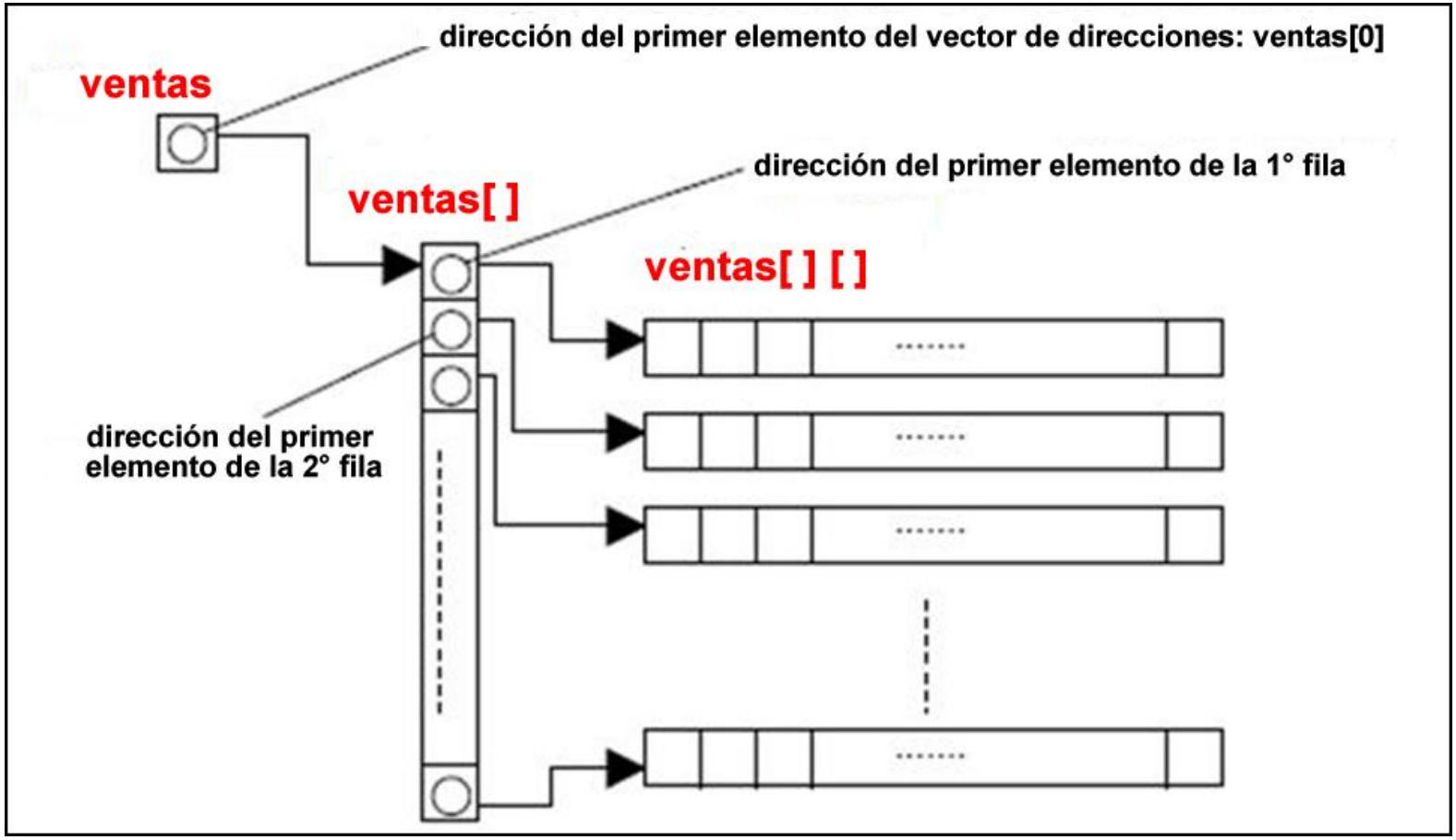
| | columna_0<br>*mes_1* | columna_1<br>*mes_2* | columna_2<br>*mes_3* |
|---|---|---|---|
| fila_0<br>*vend_1* | 9,00 | 4,00 | 2,00 |
| | *(*(ventas + 0) + 0) | *(*(ventas + 0) + 1) | *(*(ventas + 0) + 2) |
| | 60FED4 | 60FED8 | 60FEDC |
| fila_1<br>*vend_2* | 5,30 | 10,00 | 6,00 |
| | *(*(ventas + 1) + 0) | *(*(ventas + 1) + 1) | *(*(ventas + 1) + 2) |
| | 60FEE0 | 60FEE4 | 60FEE8 |
| fila_2<br>*vend_3* | 4,20 | 7,00 | 9,00 |
| | *(*(ventas + 2) + 0) | *(*(ventas + 2) + 1) | *(*(ventas + 2) + 2) |
| | 60FEEC | 60FEF0 | 60FEF4 |
| fila_3<br>*vend_4* | 6,30 | 6,80 | 2,00 |
| | *(*(ventas + 3) + 0) | *(*(ventas + 3) + 1) | *(*(ventas + 3) + 2) |
| | 60FEF8 | 60FEFC | 60FF00 |

Si con ***(* ( ventas + i ) + j )** accedemos al elemento, también vimos en este proceso que podemos acceder con ***( ventas [ i ] + j ).** Por lo tanto:

**ventas [ i ] [ j ] = *(* ( ventas + i ) + j ) = *( ventas [ i ] + j )**

## Arrays y direcciones de memoria:

Resumiendo, la figura siguiente muestra la relación entre matrices y vectores de punteros:



# Arrays bidimensionales, funciones, argumentos actuales y formales

```c
//Ejemplo 1
#include <stdio.h>

#define VENDEDOR
#define  MES

void cargaMatriz(float m[ ] [ MES ]){
int i, j;

for (i=0; i < VENDEDOR; i++) {
    for (j=0; j < MES; j++)
     printf ("Ingrese venta empleado %d mes %d\n", i,j);
    scanf("%f", &m[i][j]);
}
}

void emiteMatriz(float m[ ] [ MES ]){
int i, j;
for (i=0; i < VENDEDOR; i++) {
     printf("\n");
      for (j=0; j < MES; j++)
       printf ("%.2f\t", m[i][j]);
    }
printf("\n\n");
}

int main() {
float ventas[ ] [ MES ]={0.0};
          cargaMatriz(ventas);
          emiteMatriz(ventas);
getchar();
return 0;
}
```

**Se puede dimensionar la matriz con constantes simbólicas.**

***Cuando se pasa una array bidimensional a una función se debe especificar el número de columnas - el número de filas es irrelevante. La razón de lo anterior, son los apuntadores. C requiere conocer cuántas son las columnas para que pueda saltar de fila en fila en la memoria***

**Se pasa como parámetro el nombre de la matriz.**

```c
#include <stdio.h> # ejemplo 2
#include <stdlib.h>
#include <time.h>
#define VENDEDOR 4
#define MES 3

void cargaMatriz(float m[][MES]);
void emiteMatriz(float m[][MES], char vend[][10], char mes[][10]);

int main() {
float ventas[][MES]={0.0};
char vendedores[][10]= {"vend_1","vend_2","vend_3","vend_4"};
char meses[][10]= {"mes_1","mes_2","mes_3"};

cargaMatriz(ventas);
emiteMatriz(ventas, vendedores, meses);
return 0;}

void cargaMatriz(float m[][MES]){
int i, j;
srand(time(NULL));
for (i=0; i < VENDEDOR; i++)
    for (j=0; j < MES; j++)
      m[i][j] = 10 + rand() % 89; }

void emiteMatriz(float m[][MES], char vend[][10], char mes[][10]){
int i, j;
for (i = 0; i < MES; i++ )printf("%15s", mes[i]);
printf("\n\n");
for (i=0; i < VENDEDOR; i++) {
    printf("%s:", vend[i]); {
      for (j=0; j < MES; j++)
       printf ("%10.2f |", m[i][j]);
}
  printf("\n ");
} }
```

Podemos trabajar con arrays de varias dimensiones tomando en cuenta los índices

```
            mes_1          mes_2          mes_3

vend_1:     26.00 |        19.00 |        93.00 |
 vend_2:     92.00 |        36.00 |        33.00 |
 vend_3:     11.00 |        19.00 |        73.00 |
 vend_4:     61.00 |        15.00 |        12.00 |
```

## Arrays Bidimensionales:

De acuerdo a lo que vimos un arrays multidimensional puede ser visto en varias formas en C, y

***un array de dos dimensiones es un arreglo de una dimensión, donde cada uno de los elementos otro array*.**

Por lo tanto, la notación: **a[n][m]** nos indica que los elementos del array están guardados fila por fila.

Considerando que una función deba recibir, por ejemplo int a[5][35], se puede declarar el argumento de la función como:

```
funcion( int a[][35] ) { ..... }
```

o aún

```
funcion( int (*a)[35] ) { ..... }
```

En el último ejemplo se requieren los paréntesis (*a) ya que [ ] tiene una precedencia más alta que *.

# Arrays bidimensionales, y su relación con cadenas de caracteres

## Arrays Bidimensionales:

Ahora veamos la diferencia entre apuntadores y arrays. El manejo de cadenas es una aplicación común de esto. Considera:

**char cadena_1[10][20];**

**char * cadena_2[10];**

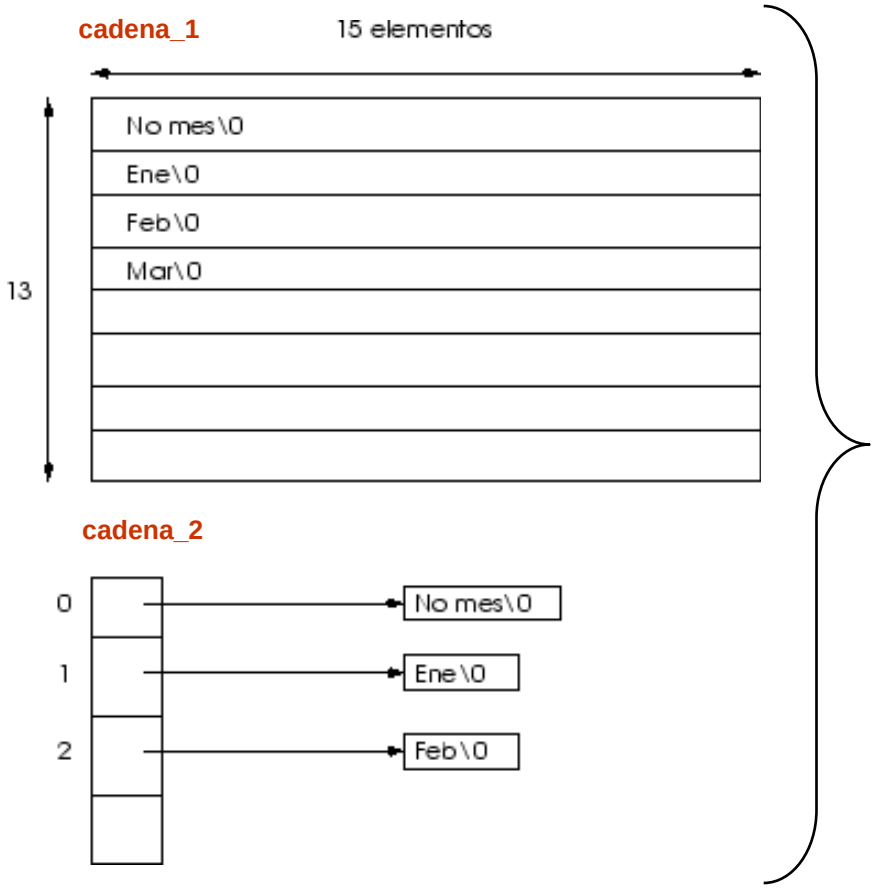
**cadena_1** es un array de 200 elementos (10 x 20) de tipo char. Si cada apuntador en **cadena_1** indica un array de 20 elementos entonces 200 char estarán disponibles en 10 arrays.

En cambio **cadena_2** tiene 10 apuntadores a elementos tipo char. Con este tipo de declaración se tiene la ventaja de que cada apuntador puede apuntar a arrays de diferente longitud.

## Arrays Bidimensionales:

**char  cadena_1[ ][ 15 ] = { "No mes", "Ene", "Feb", "Mar", ... };**

**char * cadena_2[ ] = { "No mes", "Ene", "Feb", "Mar", .... };**

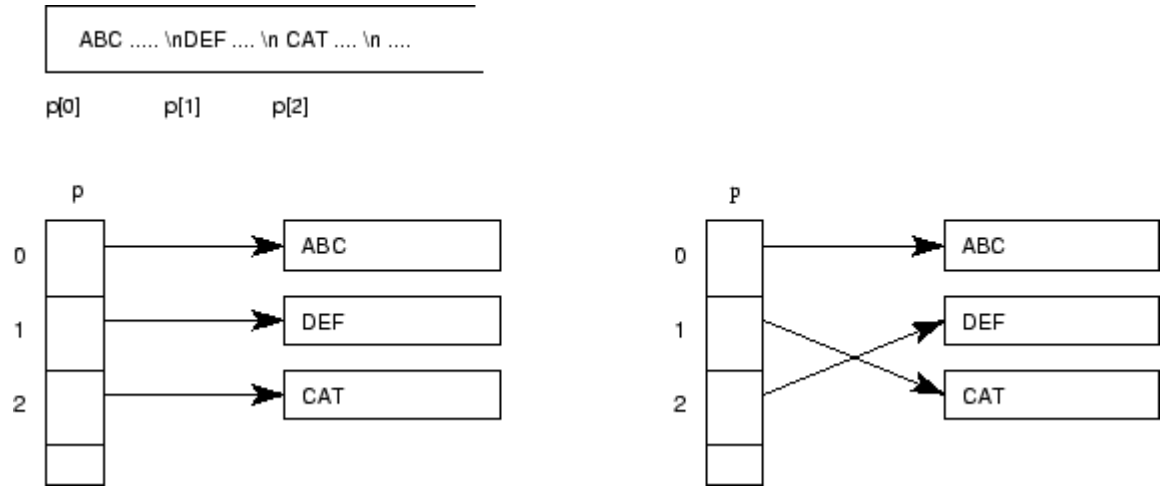



*Según la figura, puede indicarse que **se hace un manejo más eficiente del espacio haciendo uso de un arreglo de apuntadores** que usando un arreglo bidimensional.*

## Arrays Bidimensionales:

*Como mencionamos, en C se pueden tener arreglos de apuntadores ya que los apuntadores son variables. A continuación se muestra un ejemplo de su uso: ordenar las líneas de un texto de diferente longitud. (Los arreglos de apuntadores son una representación de datos que manejan de una forma eficiente y conveniente líneas de texto de longitud variable. ¿Cómo se puede hacer ?)*

- *Guardar todas las líneas en un arreglo de tipo char . Observando que \n marca el fin de cada línea.*
- *Guardar los apuntadores en un arreglo diferente donde cada apuntador apunta al primer caracter de cada línea.*
- *Comparar dos líneas usando la función de la biblioteca estándar strcmp().*
- *Si dos líneas no están en orden -- intercambiar (swap) los apuntadores (no el texto).*




## Arrays de apuntadores

De esta manera se elimina el manejo complicado del almacenamiento y la alta sobrecarga por el movimiento de líneas.

FIN