Resulta que …

En una empresa, un empleado del departamento de sistemas, tiene como función principal entregar a su jefe un reporte de la facturación total del día; y además registrar ese importe y la fecha en un libro comercial.

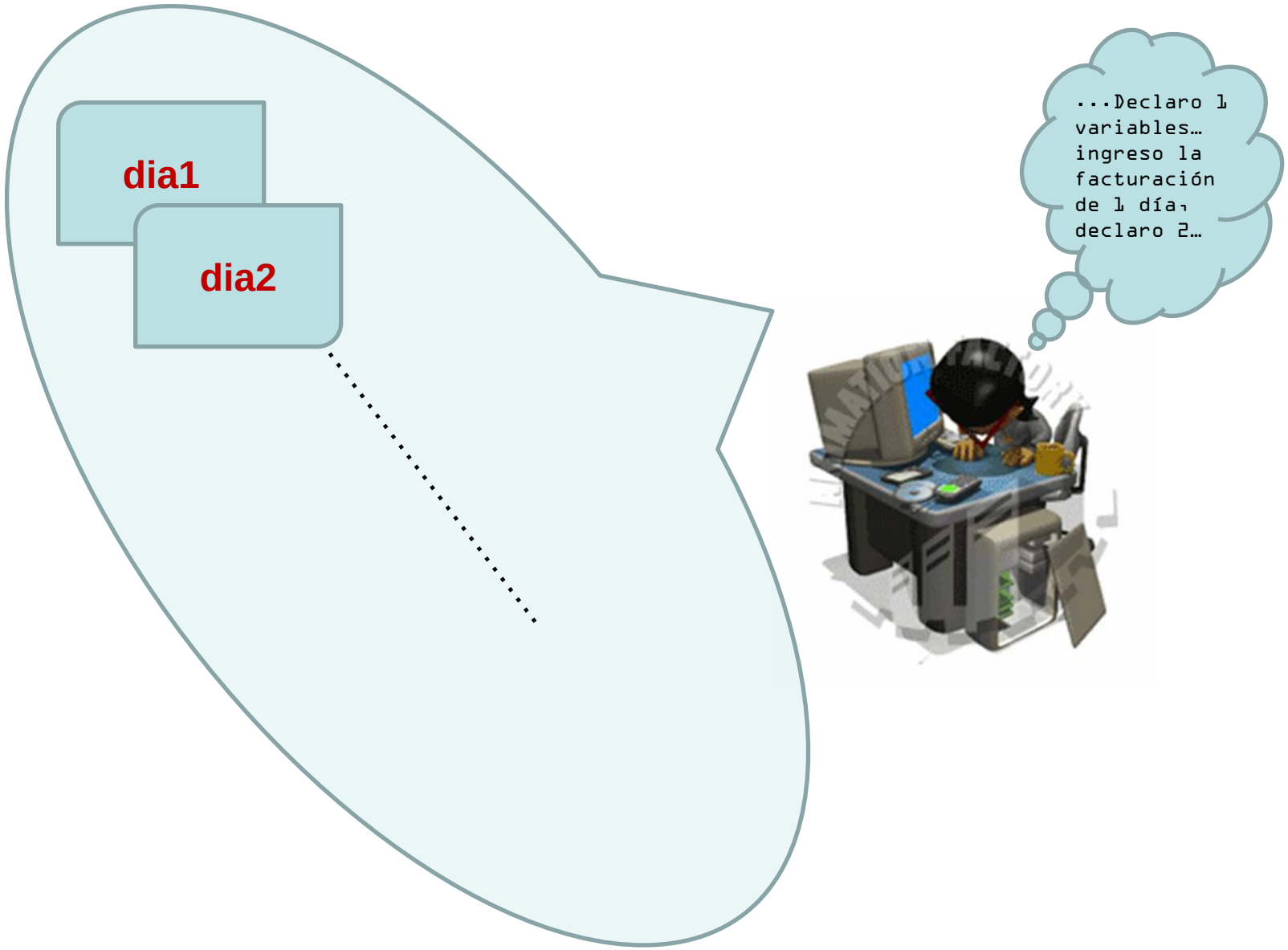
Para resolver el día a día, generó un programa que utiliza a diario, dónde va ingresando el importe de las facturas emitidas y al finalizar el día laboral genera el resultado requerido.

No le resultó difícil el desarrollo, ya que con una variable que reciba el dato y un acumulador lograba el objetivo deseado.

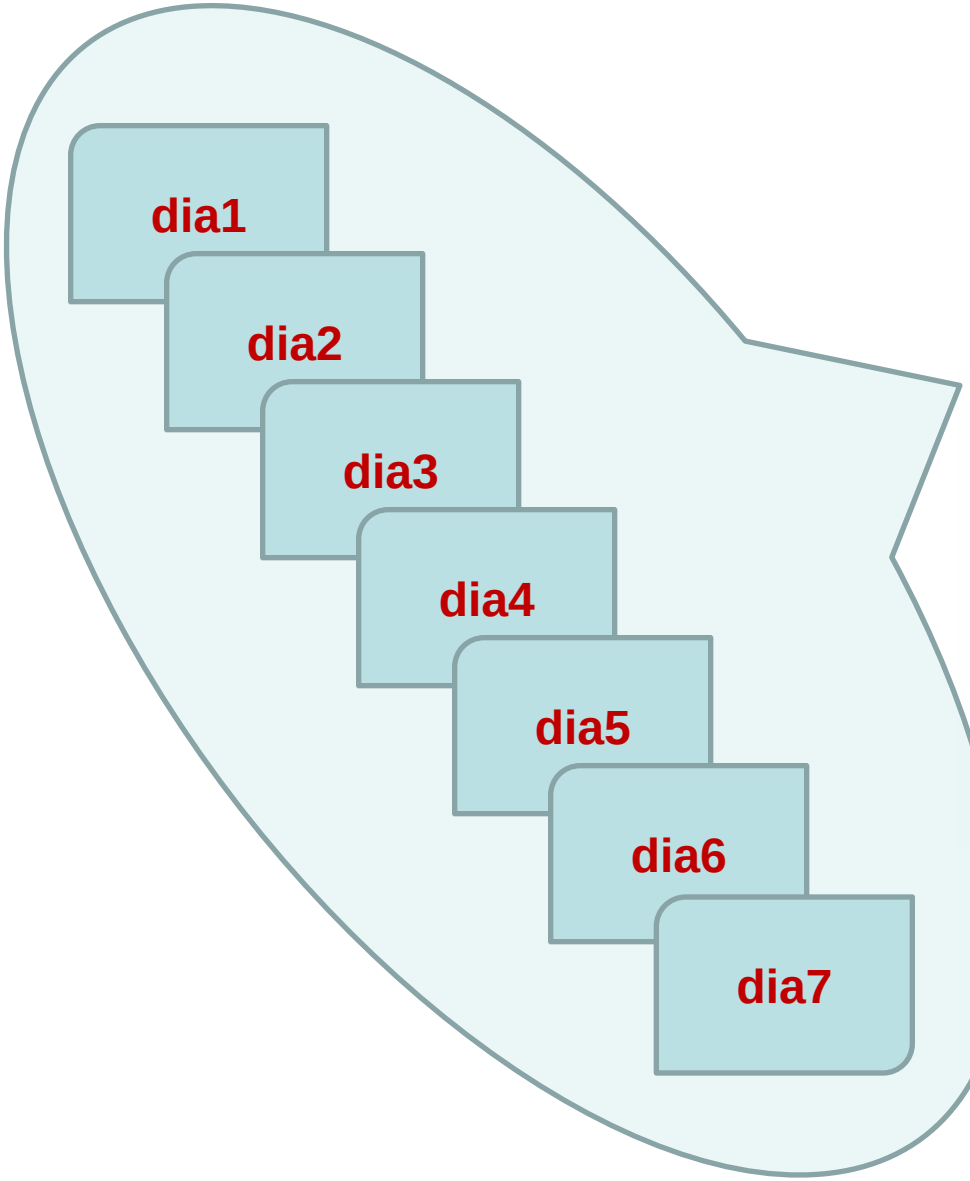
Días después, el departamento de marketing, elabora un proyecto para promover las ventas. Para presentarle la idea al presidente de la empresa, necesitan un reporte visual de la facturación de los últimos 7 días y consideran que el más apropiado es un histograma (en estadística, un histograma es una representación gráfica de una variable en forma de barras).

En consecuencia, le solicitan al hábil empleado programador, que construya un programa que realice esa representación….

El empleado tiene registro en el libro de esa facturación y comienza a pensar en cuántas variables necesitará en el programa para guardar ese dato de facturación diaria ...



Concluye en que necesitará 7 variables para guardar la facturación de cada día…..




```
….
int main(){
int dia1=0,
dia2=0,……
…….
return 0;
}
```

Entonces se dedica trabajar en el desarrollo…..

```c
#include <stdio.h>

void imprimirHistograma(int dia){
 int i;
 for (i=0; i<dia; i++)
    printf("*");
 printf("\n");
 }

void emitirFacturacionSemanal( int dia1, int dia2,
int dia3, int dia4, int dia5, int dia6, int dia7){
  printf("\n\n\n******HISTOGRAMA*******\n");
   printf("dia 1->");
    imprimirHistograma(dia1);
   printf("dia 2->");
    imprimirHistograma(dia2);
   printf("dia 3->");
    imprimirHistograma(dia3);
   printf("dia 4->");
    imprimirHistograma(dia4);
   printf("dia 5->");
    imprimirHistograma(dia5);
   printf("dia 6->");
    imprimirHistograma(dia6);
   printf("dia 7->");
   imprimirHistograma(dia7);
   }
```

Como decidió usar variables, este desarrollo sería una posible solución. En el código se observa la declaración de 7 variables y las llamadas a cada función con las mismas.

```c
void ingresarDiaria( int *dia1, int *dia2, int *dia3, int *dia4, int *dia5,
int *dia6, int *dia7){

   printf("Ingrese el total facturado en el dia 1: ");
    scanf("%d", dia1);
   printf("Ingrese el total facturado en el dia 2: ");
    scanf("%d", dia2);
   printf("Ingrese el total facturado en el dia 3: ");
    scanf("%d", dia3);
   printf("Ingrese el total facturado en el dia 4: ");
    scanf("%d", dia4);
   printf("Ingrese el total facturado en el dia 5: ");
    scanf("%d", dia5);
   printf("Ingrese el total facturado en el dia 6: ");
    scanf("%d", dia6);
   printf("Ingrese el total facturado en el dia 7: ");
   scanf("%d", dia7);
   }


int main(){
int dia1=0, dia2=0, dia3=0, dia4=0, dia5=0, dia6=0,
dia7=0;

ingresarDiaria(&dia1, &dia2, &dia3, &dia4, &dia5, &dia6,
&dia7);

emitirFacturacionSemanal(dia1, dia2, dia3, dia4, dia5, dia6,
dia7);

printf("\n\n\n ");
getchar();
return 0;
}
```

Testea su programa y verifica si cumple con lo requerido…

```
Ingrese el total facturado en el dia 1: 12
Ingrese el total facturado en el dia 2: 15
Ingrese el total facturado en el dia 3: 8
Ingrese el total facturado en el dia 4: 19
Ingrese el total facturado en el dia 5: 16
Ingrese el total facturado en el dia 6: 11
Ingrese el total facturado en el dia 7: 9



******HISTOGRAMA*******
dia 1->***********
dia 2->**************
dia 3->********
dia 4->******************
dia 5->***************
dia 6->**********
dia 7->*********
```

Cumple con el objetivo?. Sí.

Si bien el programa cumple con lo requerido, el empleado analiza y cuestiona lo difícil que es trabajar con tantas variables, con nombres distintos, para procesar datos que son parte del mismo conjunto, es decir el conjunto de facturaciones de 7 días. Pero también sabe que necesita esos datos -todos al mismo tiempo- en memoria para el proceso.

Tras cuestionarse, concluye en que resulta evidente que necesitará de otro tipo de estructura en memoria, que almacene y mantenga el conjunto de valores en forma compacta. Él piensa en el problema resuelto con variables y bosqueja una idea de cómo debería ser esa estructura. ..

12	15	8	19	16	11	9

cada uno de los cuadros representa una variable de las utilizadas en el programa…. Existe algo así?...

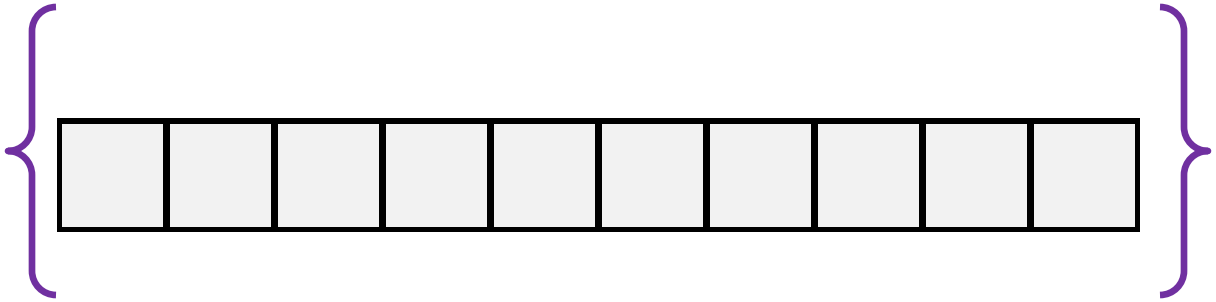
Arrays

Primera parte:

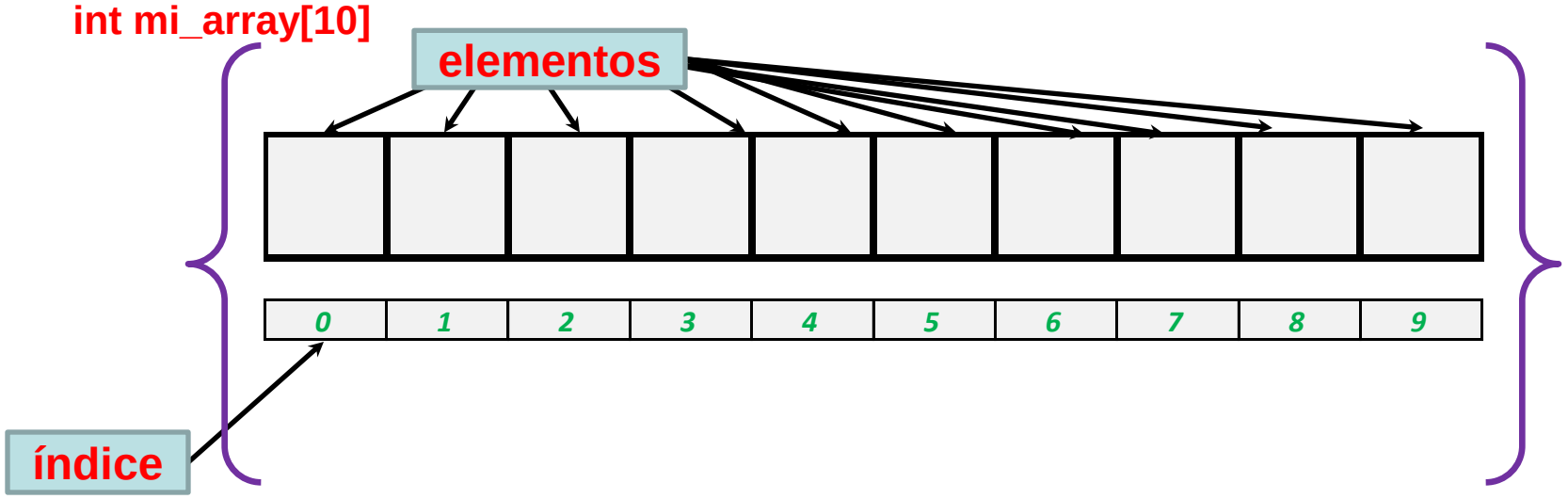
Arrays Unidimensionales en C

Un array o vector unidimensional es un conjunto de valores, finito, es decir que tiene **dimensión o tamaño**), y ese conjunto es del mismo **tipo de datos.**

Idea de la estructura:

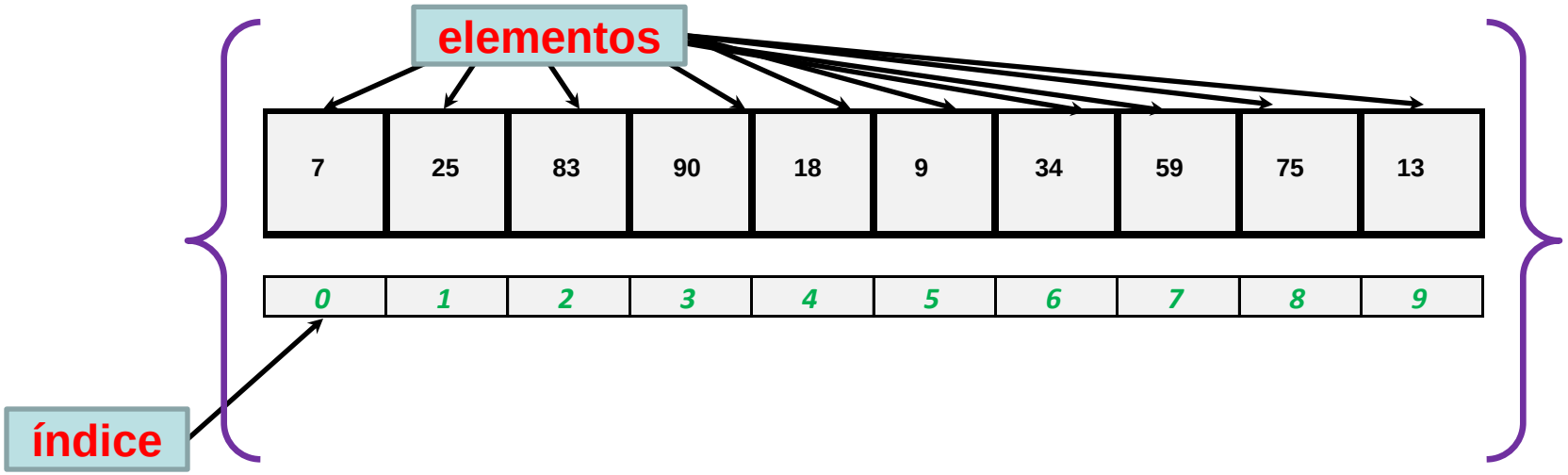


Al array debemos darle un nombre (identificador) y cada elemento del array se identifica por el nombre del array y su **posición** señalada por el **índice**.



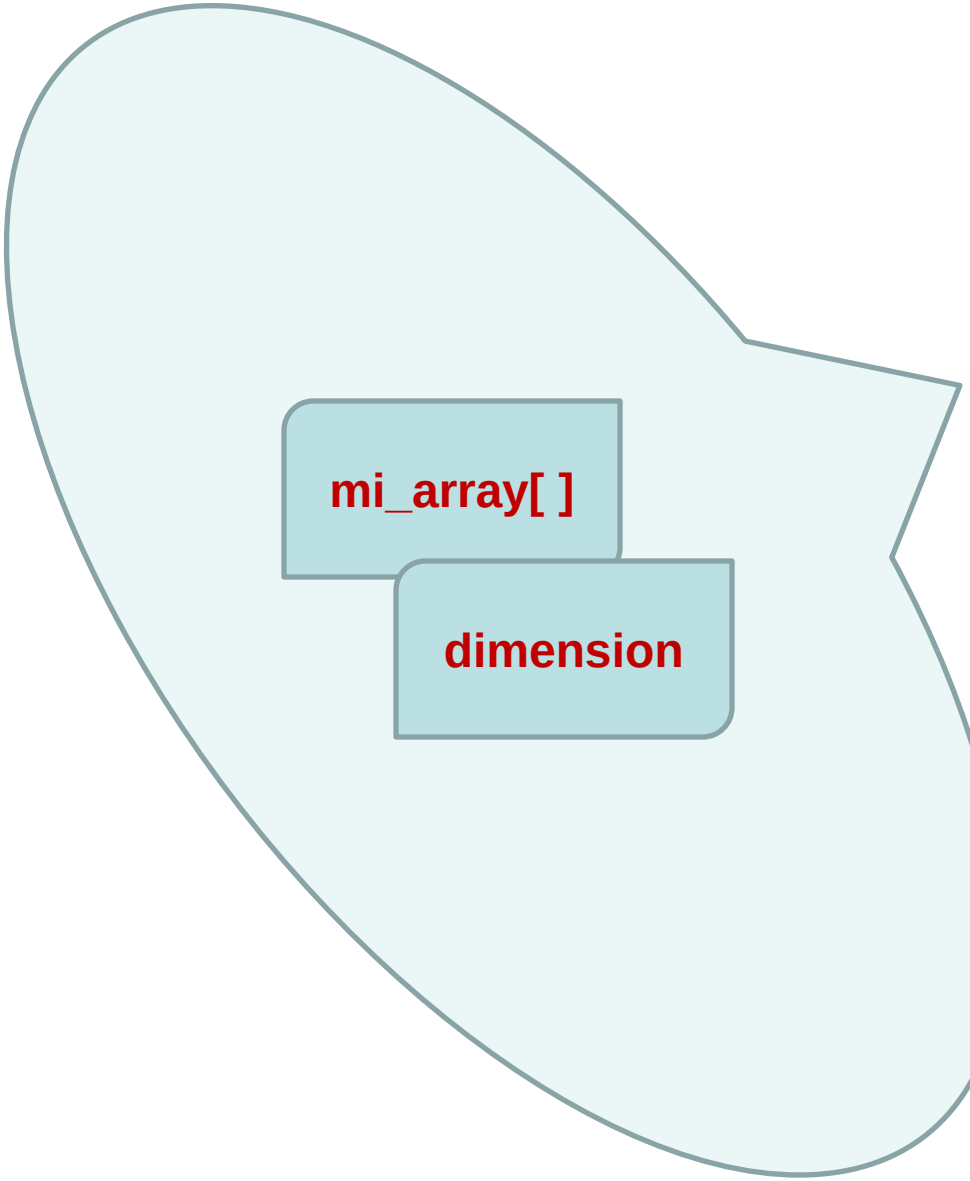
El índice del primer elemento es el **cero**. El método se denomina **"indexación basada en cero".**

int mi_array[10]



Descripción del array o vector del ejemplo	
Nombre del array	mi_array
Tipo de datos que guarda	int
Dimensión , tamaño, cantidad de elementos que puede guardar	10
Contenido o valor o elemento de la posición [0]	7
Contenido o valor o elemento de la posición [1]	25
Contenido o valor o elemento de la posición [2]	83
Contenido o valor o elemento de la posición [7]	59
Contenido o valor o elemento de la posición [10]	**Error**
Elemento 90 está en la posición	[3]
Elemento 9 está en la posición	[5]
Elemento 13 está en la posición	[9]

Entonces, ahora que sabe como resolverlo, el empleado programador construye la nueva solución…..




```
….
int main(){
int mi_array[
]……
…….
return 0;
}
```

```c
#include <stdio.h>

void imprimirHistograma(int d[], int n){
int i;
 for (i=0; i<d[n]; i++)
    printf("*");
 }

void emitirFacturacionSemanal(int d[], int t) {
 int i;
   printf("\n\n\n******HISTOGRAMA*******\n");
   for (i=0; i<t; i++){
        printf("\ndia %d  ->", (i+1));
        imprimirHistograma(d, i);
   }
}

void ingresarDiaria( int d[], int t) {
 int i;
   for (i=0; i<t; i++)
   {
   printf("Ingrese el total facturado en el dia %d: ", i+1);
   scanf("%d", &d[i]);
   }
}

int main(){
int const tam=7;
int mi_array[tam];
ingresarDiaria(mi_array, tam);
emitirFacturacionSemanal(mi_array, tam);
printf("\n\n\n ");
getchar();
return 0; }
```

Testea y verifica habiendo utilizado un array:

```
Ingrese el total facturado en el dia 1: 12
Ingrese el total facturado en el dia 2: 15
Ingrese el total facturado en el dia 3: 8
Ingrese el total facturado en el dia 4: 19
Ingrese el total facturado en el dia 5: 16
Ingrese el total facturado en el dia 6: 11
Ingrese el total facturado en el dia 7: 9



******HISTOGRAMA*******
dia 1  ->***********
dia 2  ->***************
dia 3  ->********
dia 4  ->******************
dia 5  ->***************
dia 6  ->***********
dia 7  ->*********
```

Cumple con el objetivo?. Sí.

Comparando Resultados

Con variables

```
Ingrese el total facturado en el dia 1: 12
Ingrese el total facturado en el dia 2: 15
Ingrese el total facturado en el dia 3: 8
Ingrese el total facturado en el dia 4: 19
Ingrese el total facturado en el dia 5: 16
Ingrese el total facturado en el dia 6: 11
Ingrese el total facturado en el dia 7: 9



******HISTOGRAMA*******
dia 1->***********
dia 2->**************
dia 3->********
dia 4->******************
dia 5->***************
dia 6->**********
dia 7->*********
```

Con un array

```
Ingrese el total facturado en el dia 1: 12
Ingrese el total facturado en el dia 2: 15
Ingrese el total facturado en el dia 3: 8
Ingrese el total facturado en el dia 4: 19
Ingrese el total facturado en el dia 5: 16
Ingrese el total facturado en el dia 6: 11
Ingrese el total facturado en el dia 7: 9



******HISTOGRAMA*******
dia 1  ->***********
dia 2  ->**************
dia 3  ->********
dia 4  ->******************
dia 5  ->***************
dia 6  ->**********
dia 7  ->*********
```

Comparando el código

Con variables

```
#include <stdio.h>
void imprimirHistograma(int dia){
 int i;
 for (i=0; i<dia; i++)
    printf("*");
 printf("\n");  }
void emitirFacturacionSemanal( int dia1, int dia2, int dia3, int dia4, int dia5, int dia6, int
dia7)
{  printf("\n\n\n******HISTOGRAMA*******\n");
   printf("dia 1->");
    imprimirHistograma(dia1);
   printf("dia 2->");
    imprimirHistograma(dia2);
   printf("dia 3->");
    imprimirHistograma(dia3);
   printf("dia 4->");
    imprimirHistograma(dia4);
   printf("dia 5->");
    imprimirHistograma(dia5);
   printf("dia 6->");
    imprimirHistograma(dia6);
   printf("dia 7->");
   imprimirHistograma(dia7);   }
void ingresarDiaria( int *dia1, int *dia2, int *dia3, int *dia4, int *dia5, int *dia6, int *dia7){
   printf("Ingrese el total facturado en el dia 1: ");
    scanf("%d", dia1);
   printf("Ingrese el total facturado en el dia 2: ");
    scanf("%d", dia2);
   printf("Ingrese el total facturado en el dia 3: ");
    scanf("%d", dia3);
   printf("Ingrese el total facturado en el dia 4: ");
    scanf("%d", dia4);
   printf("Ingrese el total facturado en el dia 5: ");
    scanf("%d", dia5);
   printf("Ingrese el total facturado en el dia 6: ");
    scanf("%d", dia6);
   printf("Ingrese el total facturado en el dia 7: ");
   scanf("%d", dia7);    }
int main(){
int dia1=0, dia2=0, dia3=0, dia4=0, dia5=0, dia6=0, dia7=0;
ingresarDiaria(&dia1, &dia2, &dia3, &dia4, &dia5, &dia6, &dia7);
emitirFacturacionSemanal(dia1, dia2, dia3, dia4, dia5, dia6, dia7);
printf("\n\n\n ");
getchar();
return 0; }
```

Con un array

```
#include <stdio.h>

void imprimirHistograma(int d[], int n){
int i;
 for (i=0; i<d[n]; i++)
    printf("*");
 }

void emitirFacturacionSemanal(int d[], int t) {
 int i;
   printf("\n\n\n******HISTOGRAMA*******");
   for (i=0; i<t; i++){
        printf("\ndia %d  ->", (i+1));
        imprimirHistograma(d, i);
   }
}

void ingresarDiaria( int d[], int t) {
 int i;
   for (i=0; i<t; i++)
   {
   printf("Ingrese el total facturado en el dia %d: ", i+1);
   scanf("%d", &d[i]);
   }
}

int main(){
int const tam=7;
int dias[tam];
ingresarDiaria(dias, tam);
emitirFacturacionSemanal(dias, tam);
printf("\n\n\n ");
getchar();
return 0; }
```

Declaración (*primera forma*):

<tipo de dato> <nombre_array> [dimension];

Ejemplos:

```
int mi_array[ 7 ];
float mi_array1[ 10 ];
double mi_array2[ 100 ];
char mi_array3[ 25 ];
```

La dimensión se puede escribir en forma literal, o declarar como constante o como constante simbólica con la directiva #define, ejemplo:

- literal -> **[7]**
- constante -> **const int dim=10;**
- constante simbólica -> **#define DIM 10**

Inicialización:

Puede realizarse en el momento de la declaración:

```
int mi_array[ 7 ] = {7,9,15,13,11,8,5};
```

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

```
int mi_array[ 7 ] = {7,9,15};
```

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

```
int mi_array[ ];
mi_array = {7,9,15,13,11,8,5}; //error
```

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

```
int mi_array[7] = {7,9,15,13,11,8,5,25,2}; //error
```

Otra forma de asignar valores es:

```
mi_array[0] = 7;
mi_array[1] = 9;
mi_array[2] = 15;
mi_array[3] = 13;
mi_array[4] = 11;
mi_array[5] = 8;
mi_array[6] = 5;
```

Pero es un método poco práctico.

Declaración *(segunda forma)*

<tipo de dato> <nombre_array> [ ];

Ejemplo: int mi_array[ ];

Inicialización

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

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

Ejemplo: int mi_array[ ] = {7,9,15,13,11,8,5};

El array toma la dimensión de la cantidad de elementos.

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

Ingreso de datos, índice y representación de almacenamiento en memoria

Supongamos que tenemos esta declaración:

```
const int dim = 10;
int mi_array[dim];
```

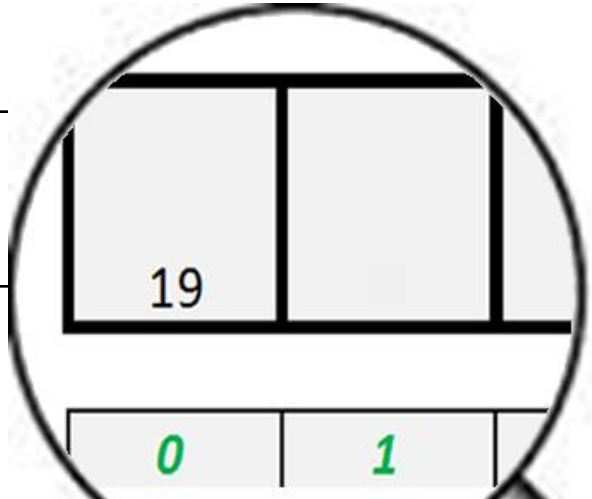
```
scanf("%d", &mi_array[ 0 ]);
```

Posición del elemento en el array

19

Ingreso del valor 19 por teclado en la **posición 0**

Representación del almacenamiento del valor y su posición en el array




```
…
scanf("%d", &mi_array[1]);
scanf("%d", &mi_array [2]);
scanf("%d", &mi_array [3]);
 …?
```

Ingreso de datos, índice y representación de almacenamiento en memoria

```
…
for (i=0; i<dim; i++)
   {  printf("Ingrese el total facturado en el dia %d: ",
      scanf("%d", &mi_array[ i ]);
} …
```

scanf("%d",&mi_array[i]);

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]

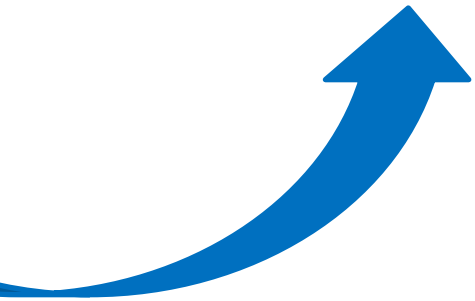
Tengo mi_array cargado con los datos y ahora que puedo hacer?

Acceso a los elementos

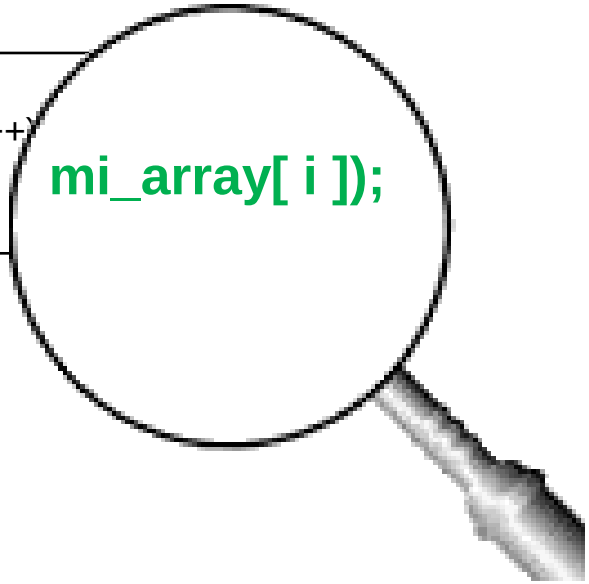
…¿Cómo accedemos a los valores ingresados?

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]



```
…
for (i=0; i<dim; i++)
    printf("%d: ",
…
```



Acceso a los elementos

nombre_array [**índice**]

O cualquier expresión que dé como resultado un número entero.

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]

```
int main() {
int n=5;
int mi_array[10]={19,3,15,7,11,9,13,5,17,1};
….
 mi_array[n - 2]=3 + mi_array[ mi_array[7] – 4 ];

 if (mi_array[4] > n)….

return 0;   }
```

Acceso a los elementos

nombre_array [índice]

O cualquier expresión que dé como resultado un número entero.

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9] ?

```
int main() {

int mi_array[10]={19,3,15,7,11,9,13,5,17,1};

 mi_array [i - 2]=3 + mi_array [mi_array[8] - 4]
 mi_array[10]=6;

return 0;   }
```

Peligro!, se está accediendo fuera de la dimensión del array.

Acceso a los elementos

- **No** se puede operar sobre todos los elementos del array a la vez.
- **Si** podemos acceder a los elementos de uno en uno.

```
int main() {

int mi_array[10]={19,3,15,7,11,9,13,5,17,1};
int otro_array[10];
int i;

otro_array = mi_array;

for(i=0;i<dim;i++)
otro_array[i] = mi_array[i];

return 0;  }
```

NO

SI

Ventajas y desventajas de utilizar arrays

Ventajas

- La escritura de código se reduce.
- Acceso y operaciones más eficiente.
- Permiten almacenar la cantidad dimensionada.
- Menor riesgo de errores en el procesamiento del conjunto de datos.
- Conociendo la posición de un elemento podemos acceder a él directamente.

Desventajas

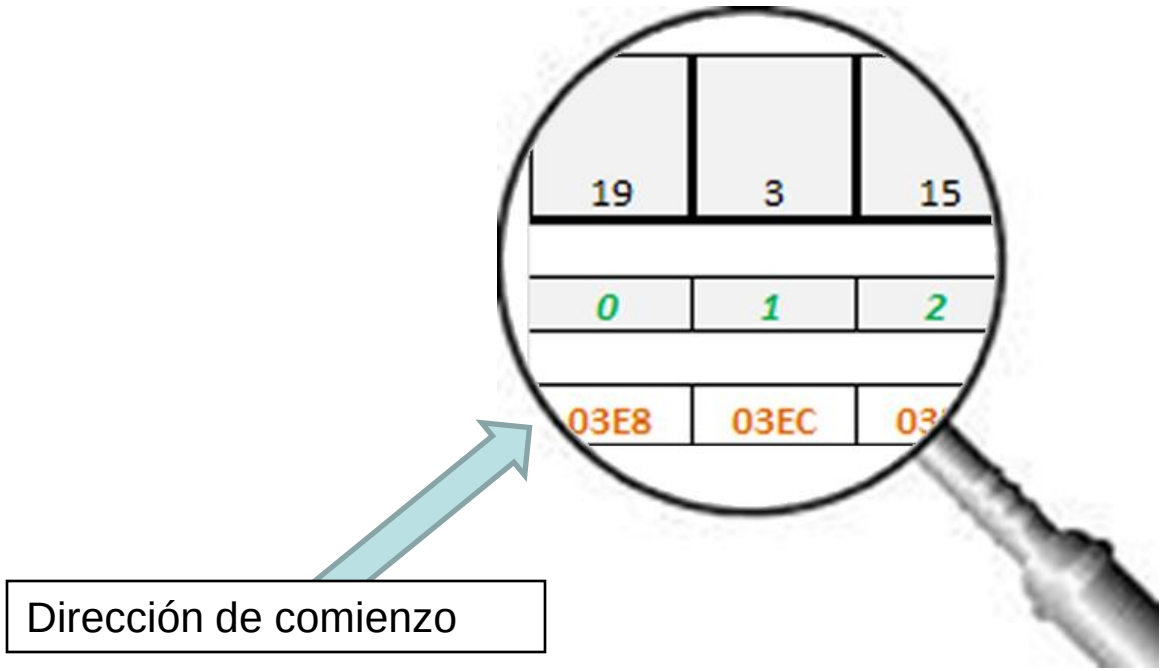
- Los arrays estáticos no pueden redimensionarse.
- Puede generar errores si el espacio reservado es menor del necesario.
- Si el tamaño de reserva es mayor al utilizado, se desperdician las posiciones no utilizadas.
- Eliminar un elemento implica que hay que reubicar el resto.
- Buscar un elemento en un array desordenado es muy lento.

Arrays y su relación con punteros

Arrays y direcciones de memoria

El **nombre** de un array es la dirección comienzo del mismo y es **constante**, es decir, **no** podemos hacer que comience en otra parte.

El array ‘mi_array’ comienza en la dirección 03E8



Arrays y direcciones de memoria

Ejemplo 1	int main() {
	int const dim = 10;
	int mi_array[ ]={19,3,15,7,11,9,13,5,17,1};
	
	mi_array = NULL; //error
	
	return 0;
	}

Ejemplo 2	int main() {
	int const dim = 10;
	int mi_array[ ]={19,3,15,7,11,9,13,5,17,1};
	mi_array ++; //error
	
	return 0;
	}

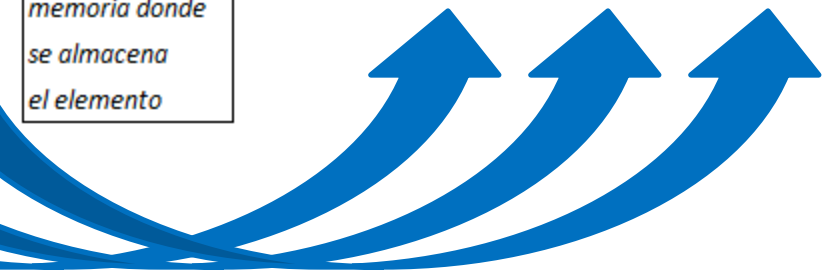
Ejemplo 3	int main() {
	int const dim = 10;
	int mi_array[ ] ={19,3,15,7,11,9,13,5,17,1}, *p;
	
	mi_array = p; //error
	
	return 0;
	}

Arrays y direcciones de memoria

Los elementos se almacenan en **posiciones contiguas de memoria**:

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]
direcciones de memoria donde se almacena el elemento	03E8	03EC	03F0	03F4	03F8	03FC	0400	0404	0408	040C



Y se puede acceder a la memoria?

Arrays y direcciones de memoria – primera forma

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]
direcciones de memoria donde se almacena el elemento	03E8	03EC	03F0	03F4	03F8	03FC	0400	0404	0408	040C
acceso a las direcciones de cada elemento	mi_array	mi_array + 1	mi_array + 2	mi_array + 3	mi_array + 4	mi_array + 5	mi_array + 6	mi_array + 7	mi_array + 8	mi_array + 9

//entonces para emitir las direcciones de cada elemento:

printf("El elemento mi_array[%d] tiene dirección %x \n",

mi_array + i

Arrays y direcciones de memoria – segunda forma

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
índice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]
direcciones de memoria donde se almacena el elemento	03E8	03EC	03F0	03F4	03F8	03FC	0400	0404	0408	040C
acceso a las direcciones de cada elemento	mi_array	mi_array + 1	mi_array + 2	mi_array + 3	mi_array + 4	mi_array + 5	mi_array + 6	mi_array + 7	mi_array + 8	mi_array + 9
que es igual a decir	&mi_array[0]	&mi_array[1]	&mi_array[2]	&mi_array[3]	&mi_array[4]	&mi_array[5]	&mi_array[6]	&mi_array[7]	&mi_array[8]	&mi_array[9]

&mi_array[0]

&mi_array[i] = mi_array + i

Arrays y direcciones de memoria

O sea que:

&mi_array[i] = mi_array + i

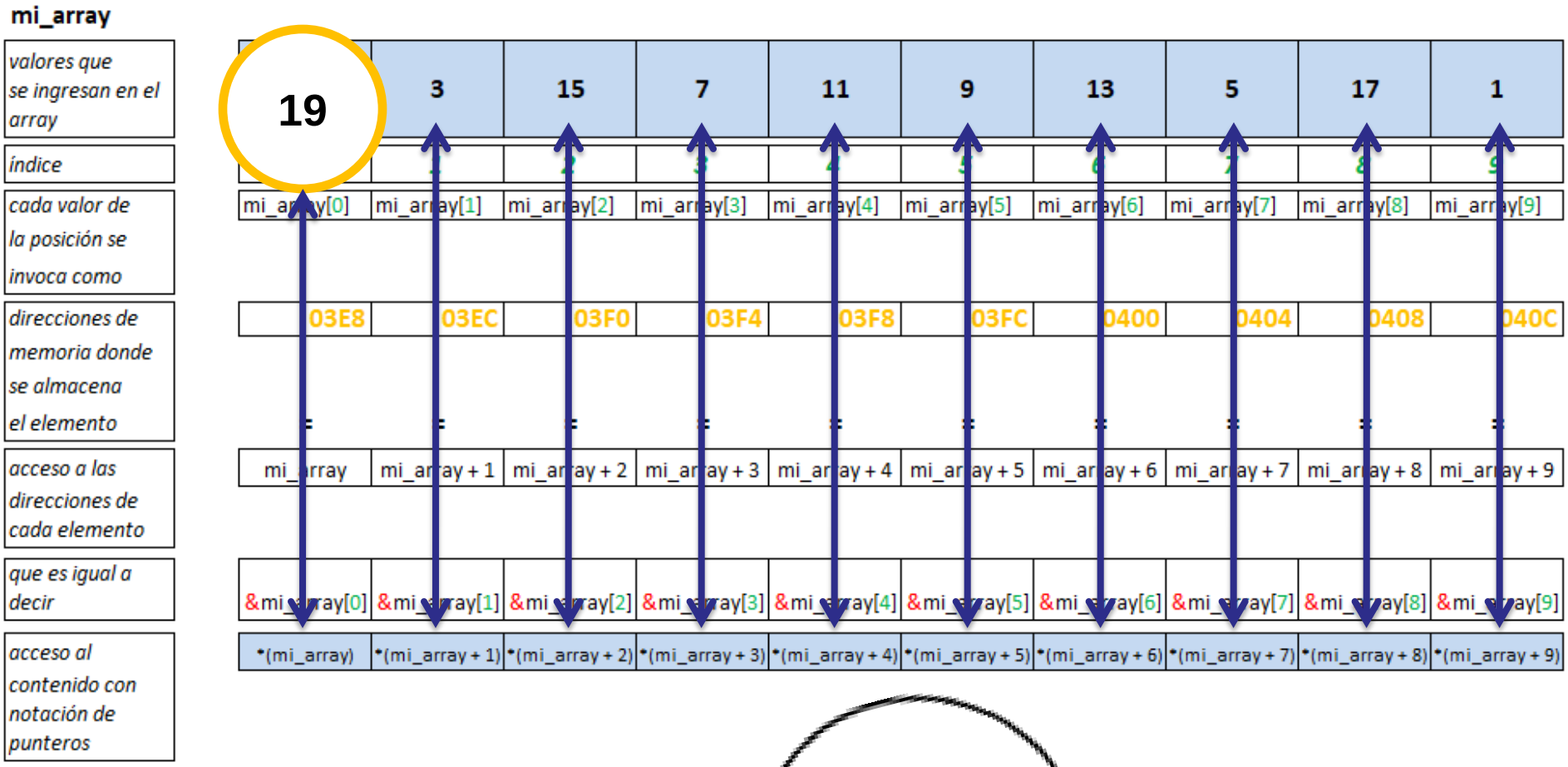
mi_array

direcciones de memoria donde se almacena el elemento	03E8	03EC	03F0	03F4	03F8	03FC	0400	0404	0408	040C
=	=	=	=	=	=	=	=	=	=	=
acceso a las direcciones de cada elemento	mi_array	mi_array + 1	mi_array + 2	mi_array + 3	mi_array + 4	mi_array + 5	mi_array + 6	mi_array + 7	mi_array + 8	mi_array + 9
=	=	=	=	=	=	=	=	=	=	=
que es igual a decir	&mi_array[0]	&mi_array[1]	&mi_array[2]	&mi_array[3]	&mi_array[4]	&mi_array[5]	&mi_array[6]	&mi_array[7]	&mi_array[8]	&mi_array[9]

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

Arrays y direcciones de memoria

Por lo tanto, se puede acceder al contenido mediante notación de punteros:



//emisión mediante notación de punteros:

printf("El contenido de mi_array[%d] es %d \n", i, *(mi_array+i)

Arrays y direcciones de memoria

O sea que:

mi_array [i] = *(mi _array + i)

mi_array

valores que se ingresan en el array	**19**	**3**	**15**	**7**	**11**	**9**	**13**	**5**	**17**	**1**
	=	=	=	=	=	=	=	=	=	=
cada valor de la posición se invoca como	mi_array[0]	mi_array[1]	mi_array[2]	mi_array[3]	mi_array[4]	mi_array[5]	mi_array[6]	mi_array[7]	mi_array[8]	mi_array[9]
	=	=	=	=	=	=	=	=	=	=
acceso al contenido con notación de punteros	*(mi_array)	*(mi_array + 1)	*(mi_array + 2)	*(mi_array + 3)	*(mi_array + 4)	*(mi_array + 5)	*(mi_array + 6)	*(mi_array + 7)	*(mi_array + 8)	*(mi_array + 9)

Arrays, funciones, argumentos actuales y formales

Arrays, funciones, argumentos actuales y formales

Como se mencionó, cuando usamos el **nombre** de un array sin corchetes, **obtenemos la dirección de memoria donde comienza**.

```
int main() {

int mi_array[10]={19,3,15,7,11,9,13,5,17,1};

printf("\nDirección de comienzo del array mi_array: %x \n", mi_array);


return 0;
 }
```

Invocando sólo el nombre, se emite la dirección de memoria donde comienza el array.

Arrays, funciones, argumentos actuales y formales

Los arrays -en C- se pasan como parámetros por referencia o dirección.

```
int main(){
const int dim = 7;
int mi_array[dim];

ingresarDiaria(mi_array, dim);

emitirFacturacionSemanal(mi_array, dim);

printf("\n\n\n ");

return 0;
}
```

Entonces, siendo el nombre del array su dirección de comienzo, se pasa como parámetro sólo el nombre y además, su tamaño.

Arrays, funciones, argumentos actuales y formales

```
void imprimirHistograma ( int d[ ], int n )
{ int i;
   for (i=0; i<d[n]; i++) printf("*");
 }

void emitirFacturacionSemanal ( int d[ ], int t )
{  int i;
   printf("\n\n\n******HISTOGRAMA*******\n");
   for (i=0; i<t; i++)
   { printf("\ndia %d  ->", (i+1));
     imprimirHistograma(d, i);
   }
}

void ingresarDiaria ( int d[ ], int t )
{  int i;
   for (i=0; i<t; i++)
   {
   printf("Ingrese el total facturado en el dia %d: ", i+1);
   scanf("%d", &d[i]);
   }
}
```

Las funciones reciben copias de la dirección del array y la dimensión para operar.

Arrays, funciones, argumentos actuales y formales

```
void imprimirHistograma ( int * d, int n )
{ int i;
   for (i=0; i<*(d+n); i++) printf("*");
 }

void emitirFacturacionSemanal ( int * d, int t )
{  int i;
   printf("\n\n\n******HISTOGRAMA*******\n");
   for (i=0; i<t; i++)
   { printf("\ndia %d  ->", (i+1));
     imprimirHistograma(d, i);
   }
}

void ingresarDiaria ( int * d, int t )
{  int i;
   for (i=0; i<t; i++)
   {
   printf("Ingrese el total facturado en el dia %d: ", i+1);
   scanf("%d", (d+i));
   }
}
```

Podemos utilizar notación de punteros para operar.

Continuará….