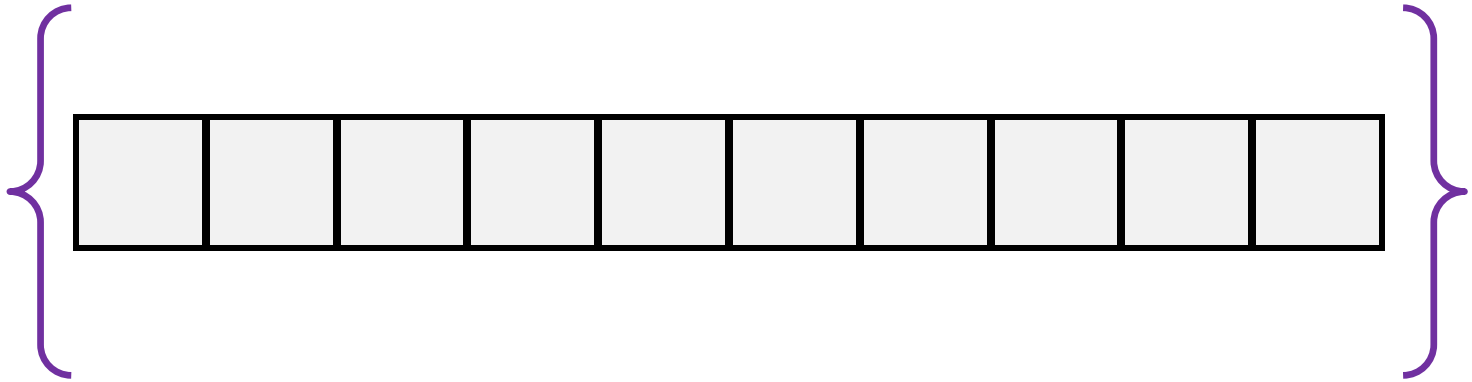
Arrays Unidimensionales

[en Lenguaje C]

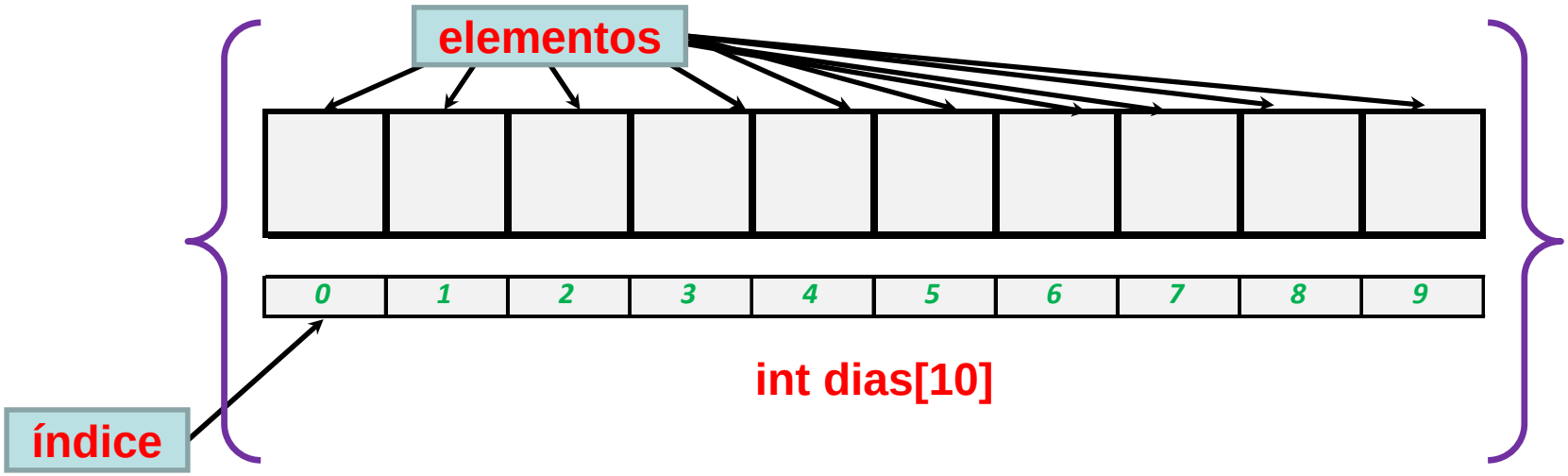
Características

- Un array o vector unidimensional, es un conjunto de valores, finito (**dimensión**), de un mismo **tipo de dato**.



Características

• Cada elemento del array se identifica por el **nombre del array (identificador)** y su posición señalada por el **índice**.



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

Ventajas y desventajas de utilizar arrays

Ventajas	Desventajas
• 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.	• 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 arreglo desordenado es muy lento.

Declaración (*primera forma*):

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

Ejemplos:

```
int dias[7];

float array1[10];

double array2[100];

char array3[25];
```

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

```
[7]
const int dim=10;
#define DIM 10
```

Inicialización:

Puede realizarse en el momento de la declaración:

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

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

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

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

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

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

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

Otra forma de asignar valores es:

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

Pero es un método poco práctico.

Declaración (*segunda forma*)

<tipo de dato> <nombre_array> [];

Ejemplo: int dias[];

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,…};

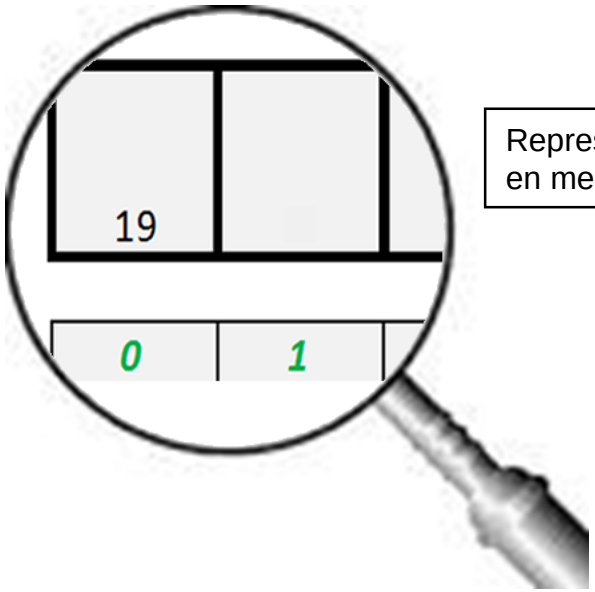
Ejemplo: int dias[] = {7,9,15,13,11,8,5};

El array toma la dimensión de la cantidad de elementos.

Ingreso de datos, índice y representación de almacenamiento en memoria

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

Posición del elemento en el array

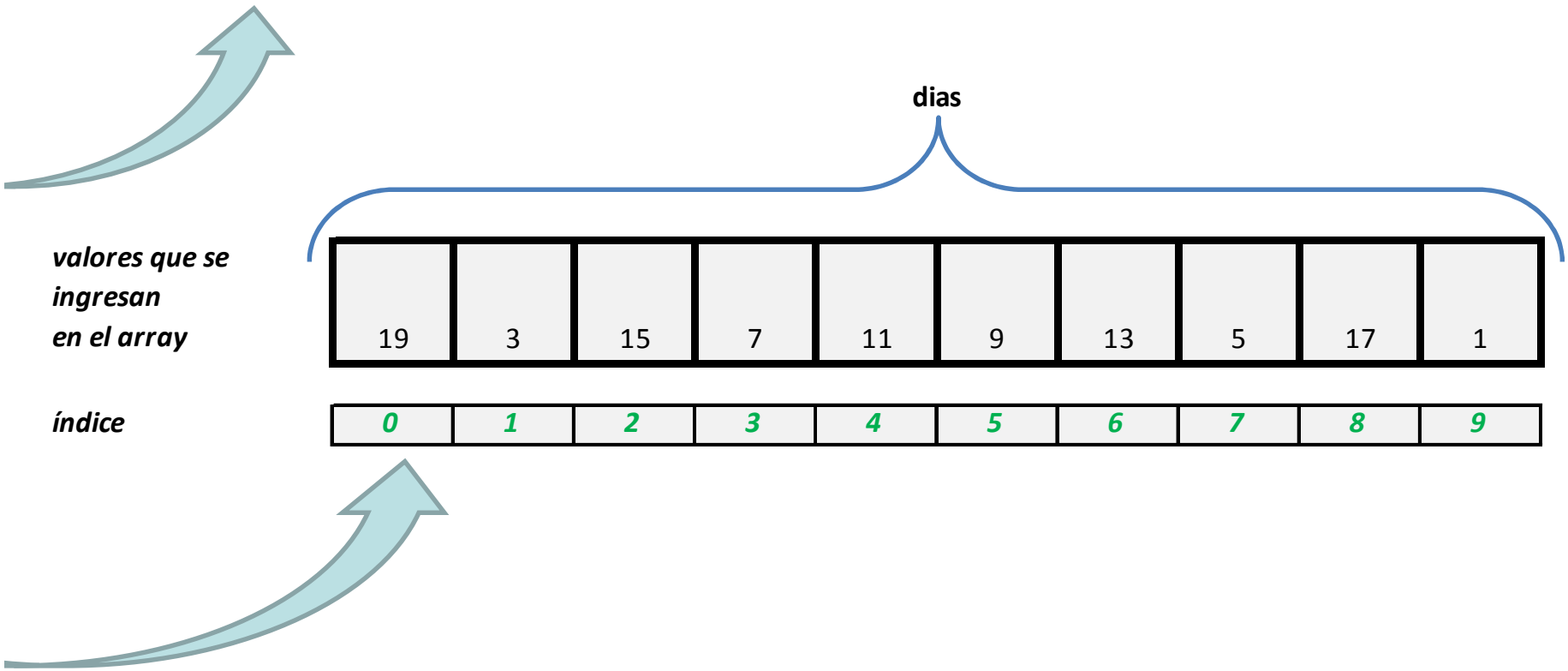


Representación del almacenamiento en memoria

```
...
scanf("%d", &dias[1]);
scanf("%d", &dias[2]);
scanf("%d", &dias[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: ", i+1);
      scanf("%d", &dias[i]);    }
} …
```

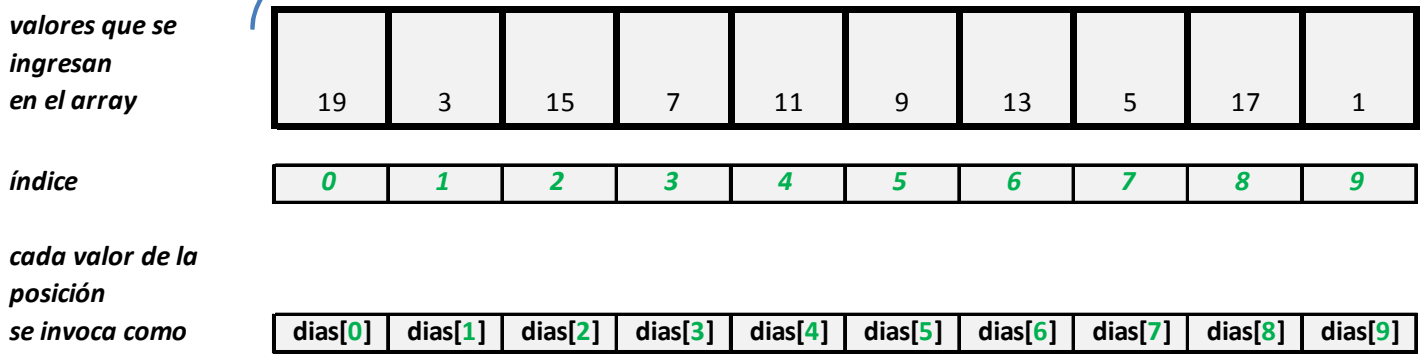


Acceso a los elementos

…¿Cómo accedemos a los valores ingresados?

nombre_vector [**índice**]

O cualquier expresión que dé como resultado un número entero.




```
int main() {

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

 dias[i - 2]=3 + dias[dias[8] - 4];
 dias[10]=6;
 system ("pause");
 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.
- Sólo podemos acceder a los elementos de uno en uno.

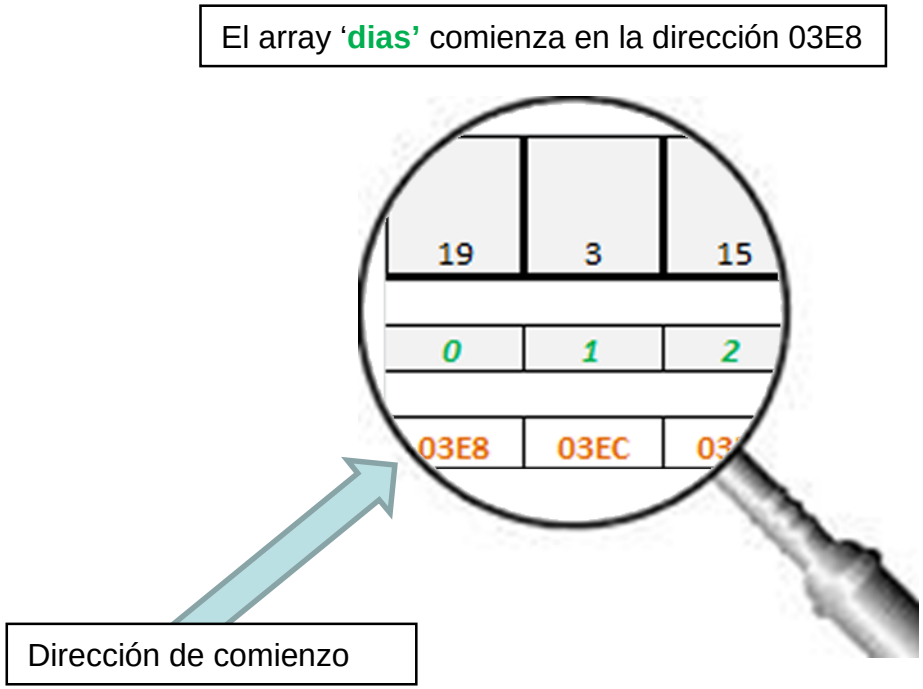
```
int main() {

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

otroVec=dias;

for(i=0;i<dim;i++)
        otroVec[i]=dias[i];

system ("pause");
return 0;  }
```

NO

SI

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.



Arrays y direcciones de memoria

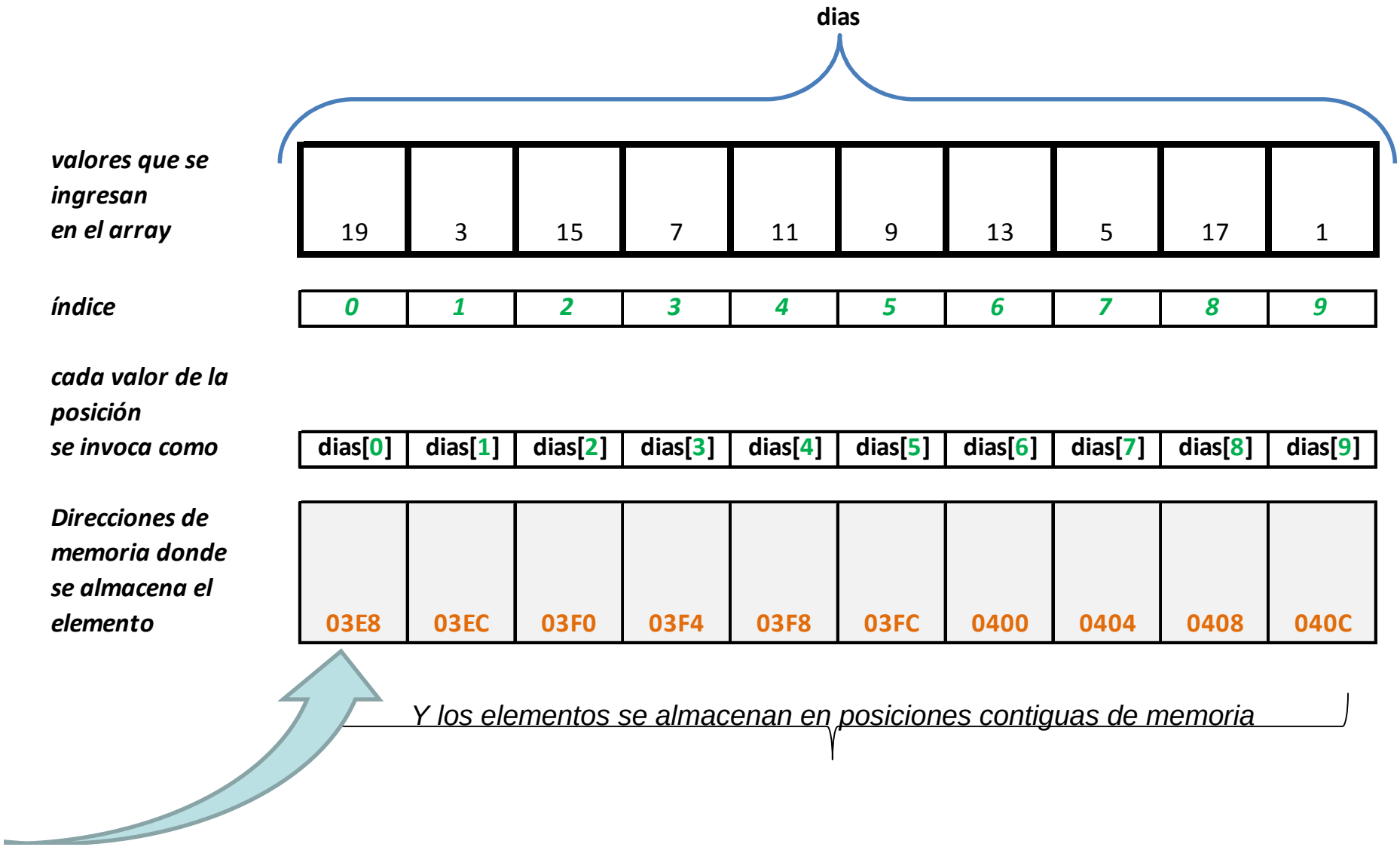
```
int main() {
int const dim = 10;
int dias[]={19,3,15,7,11,9,13,5,17,1};
dias = NULL; //error
system("pause");
return 0;
}
```

```
int main() {
int const dim = 10;
int dias[]={19,3,15,7,11,9,13,5,17,1};
dias++; //error
system("pause");
return 0;
}
```

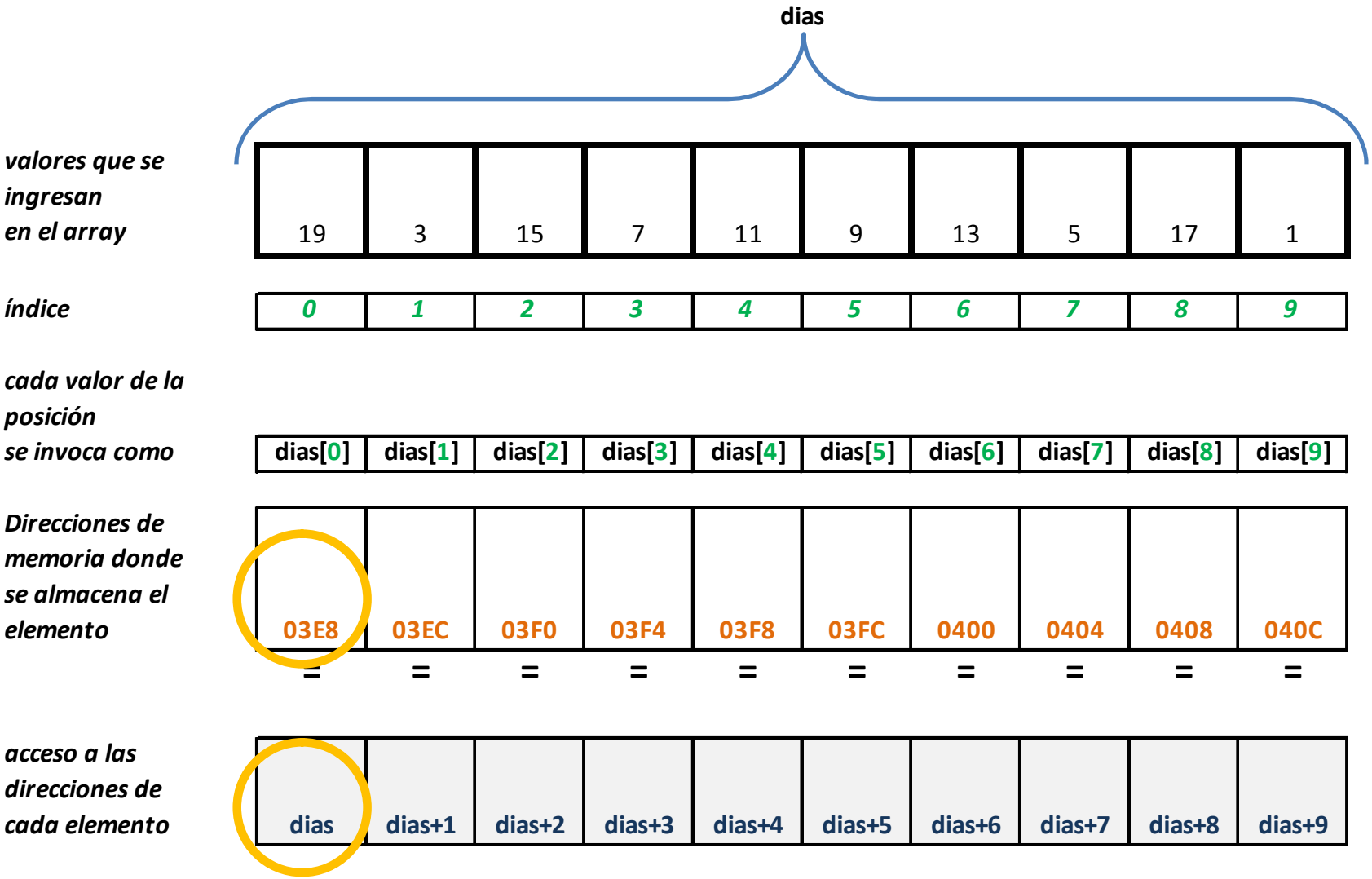
```
int main() {
int const dim = 10;
int dias[]={19,3,15,7,11,9,13,5,17,1}, *p;
dias=p; //error
system("pause");
return 0;
}
```

Arrays y direcciones de memoria

Y los elementos se almacenan en **posiciones contiguas de memoria**:

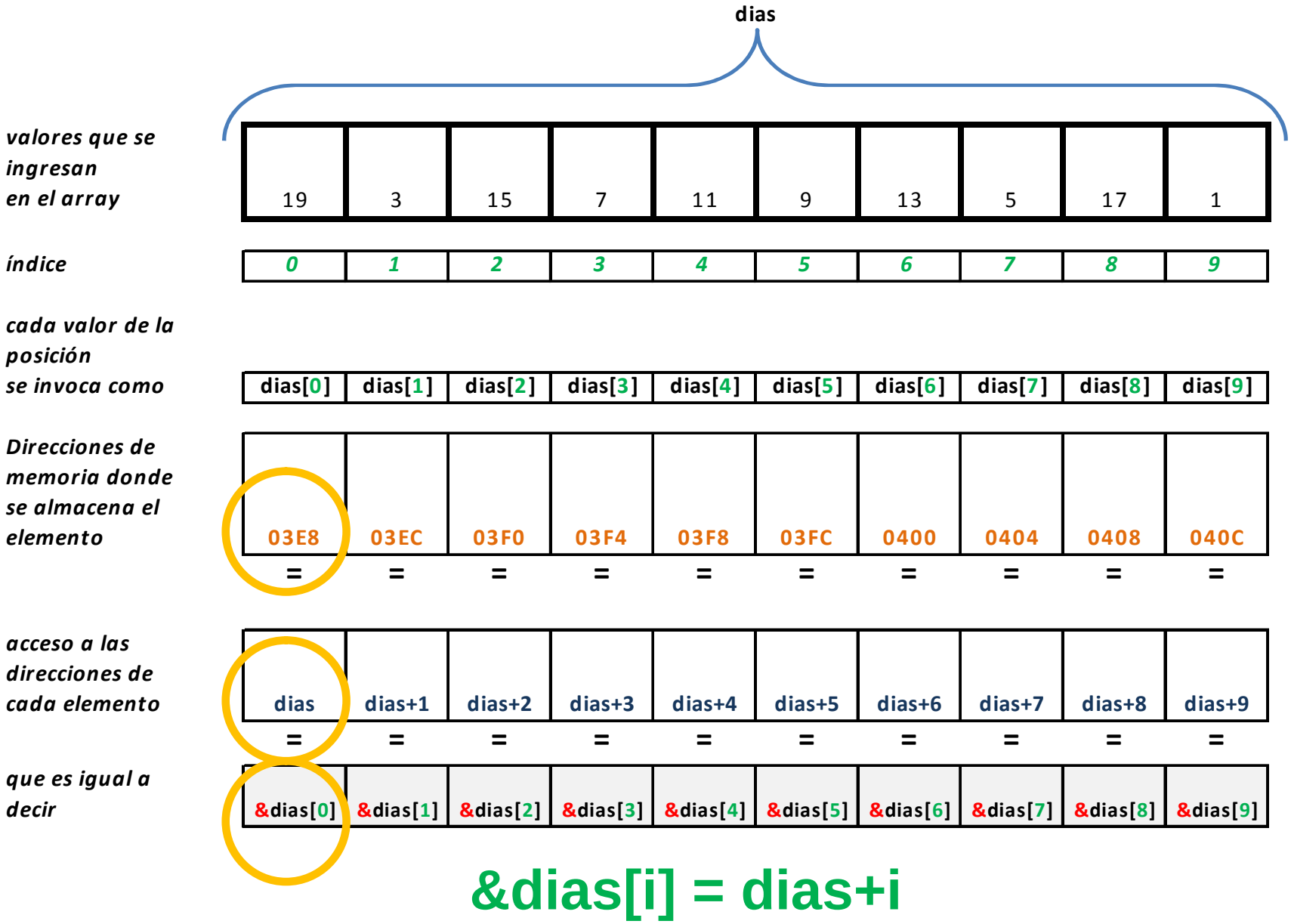


Arrays y direcciones de memoria



//entonces para emitir las direcciones de cada elemento:
printf("El elemento dias[%d] tiene direcci\242n %X \n", i, dias+i\n) ;

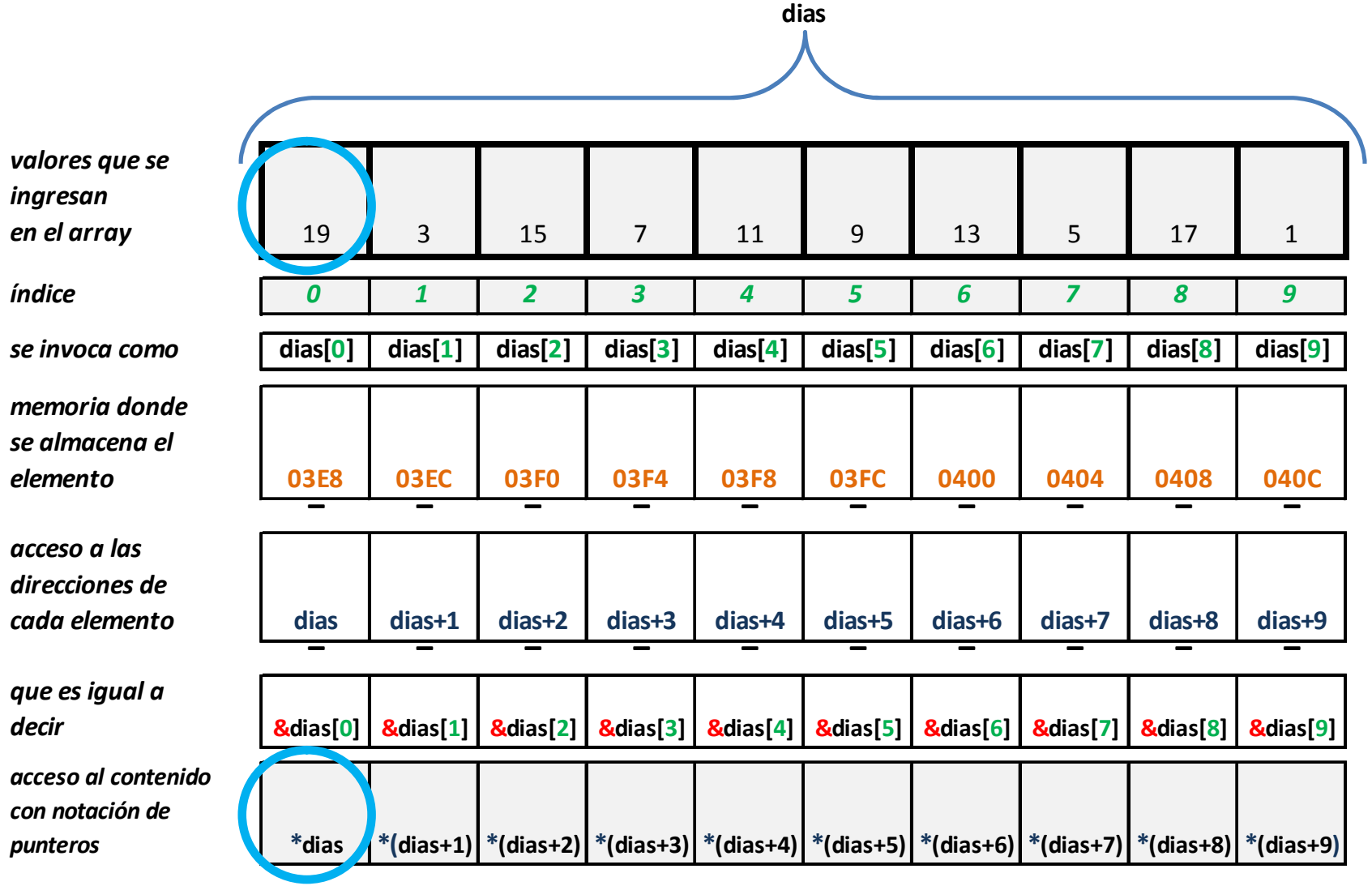
Arrays y direcciones de memoria



&dias[i] = dias+i

Arrays y direcciones de memoria

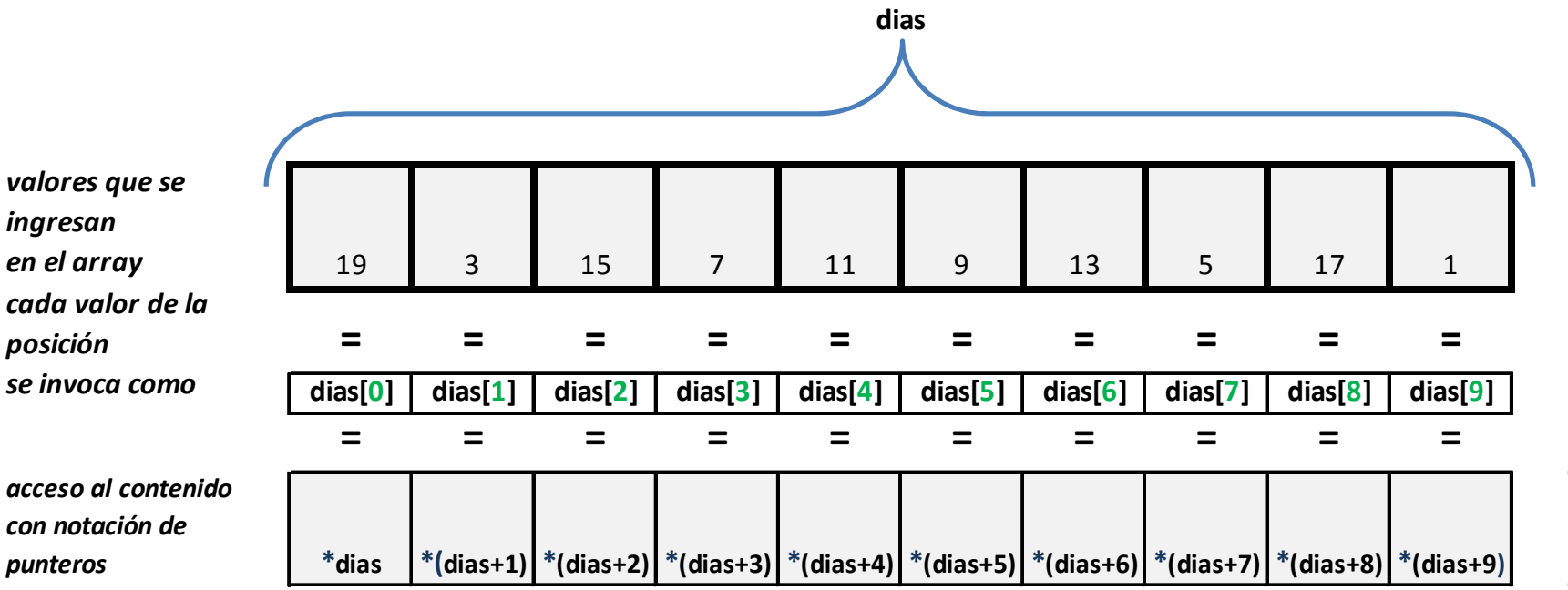
Por lo tanto, se puede acceder al contenido mediante notación de punteros:




```
//emisión mediante notación de punteros:
printf("El contenido de dias[%d] es %d \n", i, *(dias+i)) ;
```

Arrays y direcciones de memoria

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



dias[i] = *(dias+i)

Arrays, funciones, parámetros 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 dias[10]={19,3,15,7,11,9,13,5,17,1};

printf("\nDirección de comienzo del array dias: %X \n", dias);


system ("pause");
return 0;   }
```

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

Arrays, funciones, parámetros actuales y formales

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

```
main(){
const int dim=10;
int dias[dim];

ingresarDiaria(dias, dim);

emitirEstacionalidad(dias, dim);

printf("\n\n\n ");
system("pause");
return 0;
}
```

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

Arrays, funciones, parámetros actuales y formales

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

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

void emitirEstacionalidad(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]);
   }
}
```

Arrays y cadena de caracteres

• En programación denominamos ‘string’ a una secuencia de caracteres. Por ejemplo: ”Hola mundo!”

• C no soporta el tipo string.

Arrays y cadena de caracteres

- Una cadena es un array de caracteres o un puntero a una porción de memoria, que contiene caracteres ASCII.
- Finalizan con un carácter nulo o '\0' que muchas funciones utilizan para saber donde finaliza la cadena de caracteres.
- Todos los elementos son de tipo **char.**

Declaración:

char cadena[11];

O, puede emplearse una constante simbólica:

```
#define TAM 11
. . .
```

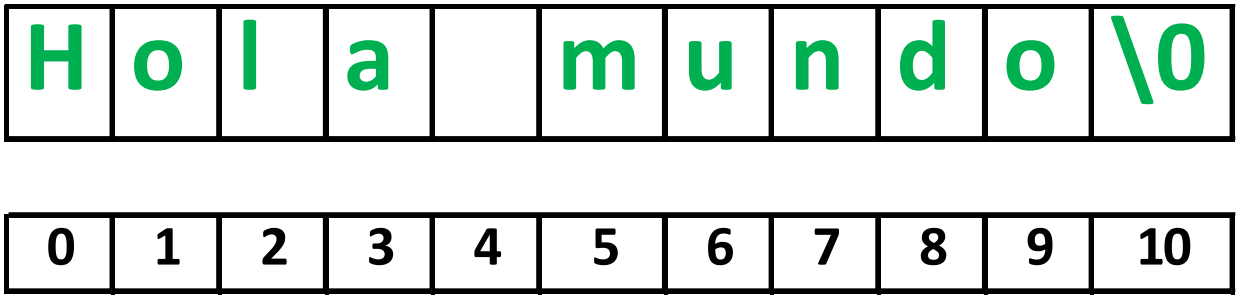
char cadena[TAM];

Arrays y cadena de caracteres

Inicialización:

- Las cadenas de caracteres se declaran e inicializan como cualquier array.

Ejemplo: **char cadena[11]={‘H’, ’o’, ’l’, ’a’, ’ ’, ’m’, ’u’, ’n’, ’d’, ’o’, ’\0’};**

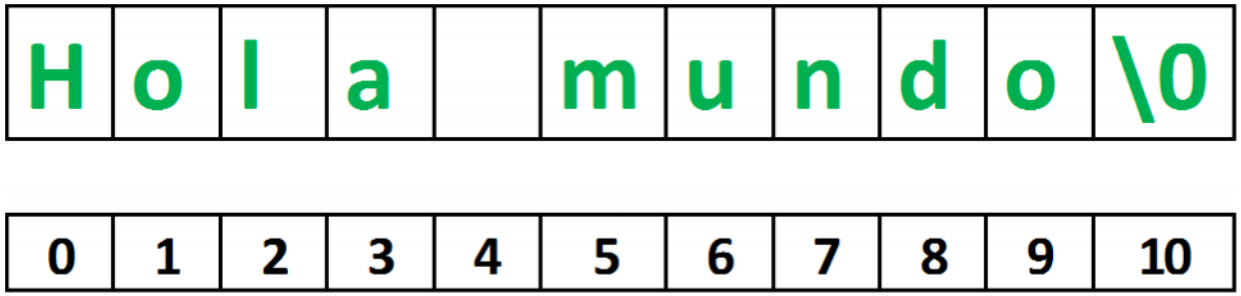


- ‘H’ es un carácter.
- «Hola mundo» es una cadena de caracteres.

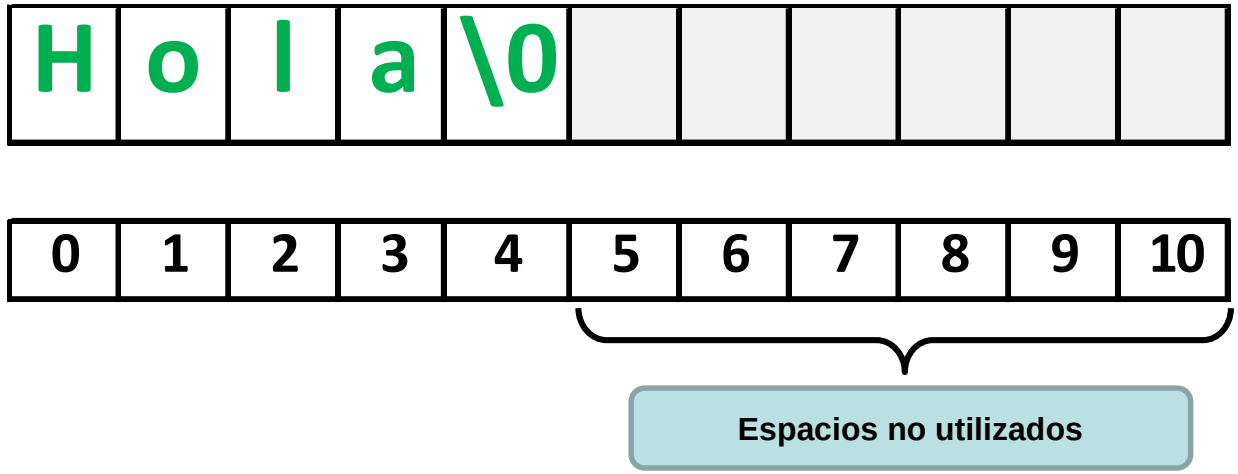
Arrays y cadena de caracteres

Se puede omitir la dimensión de un array si se inicializa en la declaración, en caso contrario es forzoso declarar la dimensión.

Otro ejemplo: **char cadena[ ]=«Hola mundo»;**



Otro ejemplo: **char cadena[11]=«Hola»;**



Arrays y cadena de caracteres

Se puede ingresar una cadena caracter por carácter utilizando la función scanf:

scanf(“%c”, &caracter[i]);

O se puede ingresar el texto completo utilizando la función scanf:

scanf(“%s”, cadena);

En este caso, el nombre del array no lleva el operador de dirección de memoria &, porque lleva implícito la dirección de memoria.

Podemos limitar la entrada de caracteres de esta forma:

scanf(“%10s”, cadena);

La función scanf guarda lo que pulsemos por teclado hasta el primer espacio en blanco o enter, por lo cual, si ingresamos una cadena compuesta, sólo se emitirá la cadena hasta el primer espacio blanco. Podemos evitarlo utilizando:

scanf(“%[^\n]”, cadena);
o
gets(cadena);

Arrays y cadena de caracteres

Y las cadenas pueden emitirse carácter a caracter:

printf ("%c", caracter[i]);

O pueden emitirse completas:

printf ("%s", cadena);

printf recibe un apuntador al inicio de la cadena, es decir, se tiene acceso a una cadena constante por un puntero a su primer elemento.

Que emita la cadena y no una dirección es una característica propia de la función y no un rasgo diferencial de los punteros a char.

%s indica que interprete esa dirección como el comienzo de una cadena y la función interpreta caracter a caracter hasta encontrar el '\0'.

Para las funciones standard de impresión en pantalla en lenguaje C, en el caso de las cadenas, p es la cadena apuntada, y *p, el carácter individual apuntado.

Arrays y cadena de caracteres

Otra forma de declaración es la siguiente:

char * otraCadena;

Entonces:

otraCadena= “Me encanta programar”;

Asigna un puntero al array de caracteres. No es la copia de una cadena, es un puntero.

¿Existe alguna diferencia entre ellas?, sí:

char cadena[]=“Hola mundo”; **Es un array**

char * otraCadena=“Hola mundo”; **Es un puntero al array**

Arrays y cadena de caracteres

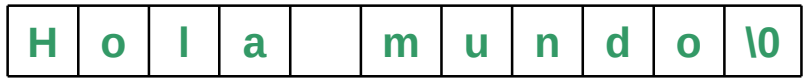
char cadena[]="Hola mundo";

•'cadena' es un **puntero constante**, no se puede cambiar, no es una variable.

• Puede cambiarse el contenido de la cadena.

• Usando la notación de arrays, los bytes de almacenamiento en el bloque de memoria estática son tomados, uno para cada caracter y uno para el caracter de terminación'\0'.

• El array "cadena" es lo mismo que decir &cadena[0], que es la dirección del primer elemento y será fijada durante el tiempo de ejecución del programa.



Arrays y cadena de caracteres

char * otraCadena="Hola mundo"; //literal de cadena

• otraCadena es una variable puntero inicializada para apuntar a una cadena constante, puede cambiarse el apuntador.

• Intentar cambiar el contenido de la cadena puede dar error o un resultado indefinido(*).

• En este tipo de notación, son requeridos los bytes de la cadena más n bytes para alojar la variable puntero otraCadena.

• Los literales de cadena se almacenan, en el segmento de datos, es el mismo sitio que se reserva para los valores constantes y variables globales. Un literal es tratado como una constante.

(*)Un resultado indefinido significa que el programa puede fallar intermitentemente, de formas distintas, y en condiciones poco esperadas.

Arrays y cadena de caracteres

Ejemplo:

```
void mi_funcion_A (char * ptr) {
char a[] = “ABCDE”;
... }
```

En el caso de mi_funcion_A, el contenido, o valor(es), del array ‘a[]’ son considerados como los datos. Se dice que ha sido inicializado a los valores ABCDE.

```
void mi_funcion_B (char * ptr) {
char *cp = “FGHIJ”;
…}
```

En el caso de mi_funcion_B, el valor del puntero ‘cp’ se considera como dato. El puntero ha sido inicializado para apuntar a la cadena FGHIJ.

En ambas funciones las definiciones son variables locales y por tanto la cadena ABCDE se guarda en la pila (stack), así como el valor del apuntador cp. La cadena FGHIJ se guarda en el segmento de datos.

Arrays y cadena de caracteres

Para realizar operaciones con caracteres existen funciones de biblioteca definidas en **ctype.h.**

Ejemplos de funciones para tratamiento de caracteres son:

isalnum(caracter): devuelve cierto (un entero cualquiera distinto de cero) si caracter es una letra o dígito, y falso (el valor entero 0) en caso contrario.
isalpha(caracter): devuelve cierto si caracter es una letra, y falso en caso contrario.
isblank(caracter): devuelve cierto si caracter es un espacio en blanco o un tabulador.
isdigit(caracter): devuelve cierto si caracter es un digito, y falso en caso contrario.
isspace(caracter): devuelve cierto si caracter es un espacio en blanco, un salto de línea, un retorno de carro, un tabulador, etc., y falso en caso contrario.
islower(caracter): devuelve cierto si caracter es una letra minúscula, y falso en caso contrario.
isupper(caracter): devuelve cierto si caracter es una letra mayúscula, y falso en caso contrario.
toupper(caracter): devuelve la mayúscula asociada a caracter, si la tiene; si no, devuelve el mismo caracter.
tolower(caracter): devuelve la minúscula asociada a caracter, si la tiene; si no, devuelve el mismo caracter.

Arrays y cadena de caracteres

Para realizar operaciones con cadenas hay que usar funciones de biblioteca. El lenguaje no dispone de operadores para cadenas pero se utilizarán funciones definidas en **string.h.**

Algunos ejemplos de funciones para tratamiento de cadenas son:

Función	Objetivo	Valor de retorno
char *strcpy(char *s1, const char *s2);	**Copia** la cadena apuntada por s2 (incluyendo el carácter nulo) a la cadena apuntada por s1.	La función retorna el valor de s1.
int strcmp(const char *s1, const char *s2);	**Compara** la cadena apuntada por s1 con la cadena apuntada por s2.	La función retorna un número entero mayor, igual, o menor que cero, según si la cadena apuntada por s1 es mayor, igual, o menor que la cadena apuntada por s2.
char *strcat(char*s1, const char *s2);	**Agrega** una copia de la cadena apuntada por s2 (incluyendo el carácter nulo) al final de la cadena apuntada por s1. El carácter inicial de s2 sobrescribe el carácter nulo al final de s1	La función retorna el valor de s1.
size_t strlen(const char *s);	**Calcula** el número de caracteres de la cadena apuntada por s.	La función retorna el número de caracteres hasta el carácter nulo, que no se incluye.

Fin