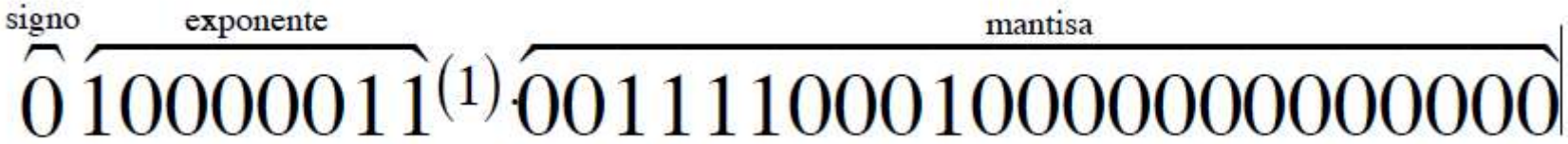
1.2.3 Números reales

Notación de punto flotante

- Esta notación se encuentra definida en la convención IEEE 754.
- Cada número de precisión simple ocupa 32 bits.
- El número se construye de la siguiente forma:
  - ✓1 bit para el signo
  - ✓8 bits para el número correspondiente al exponente
  - ✓23 bits para la parte fraccionaria de la mantisa.



1.2.3 Números reales

Notación de punto flotante

- Para encontrar la representación decimal de un número binario en punto flotante de precisión simple se debe realizar lo siguiente:

  1. Verificar el signo del número (a).
  2. Convertir los 8 dígitos del exponente a decimal.
  3. A ese número restar la constante E=127. El número obtenido será el exponente para el número 2 (b).

1.2.3 Números reales

Notación de punto flotante

4. Convertir los dígitos de la mantisa de la misma forma que se hace en notación de punto fijo.
5. Al número obtenido anteponer un 1 antes del punto para obtener un número de la forma 1.xxxx (c).
6. Multiplicar los 3 números obtenidos (a x b x c).

1.2.3 Números reales

Notación de punto flotante

- De esta manera un número está representado por 3 cantidades:

$$num. = (-1)^{signo} \times 2^{exponente-E} \times 1.mantisa$$

Donde:

signo = primer dígito

exponente = siguientes 8 dígitos

E= 127

1.2.3 Números reales

Notación de punto flotante

- Para encontrar la representación binaria a partir de un número decimal se realiza el siguiente procedimiento:
  1. El primer dígito del número binario corresponderá al signo del número.
  2. Convertir la parte entera del número decimal de la forma convencional.
  3. Recorrer el punto decimal **n** posiciones hasta obtener una notación de la forma 1.xxxx
  4. A la constante E = 127, sumarle **n**

1.2.3 Números reales

Notación de punto flotante

5. Convertir el número obtenido a binario.
6. Concatenar los bits obtenidos en el paso 1, el paso 5, los bits restantes del paso 3.
7. El número correspondiente a la parte real se debe multiplicar por 2 de manera sucesiva, los bits que se deben considerar son la parte entera del resultado de dicha multiplicación.
8. El proceso termina cuando ya se han llenado los 32 bits o cuando al realizar la multiplicación se obtiene un cero.

1.2.3 Números reales

Notación de punto flotante

- En doble precisión se utilizan 64 bits de la siguiente forma:
  - ✓1 bit para el signo
  - ✓11 bits para el exponente
  - ✓52 bits para la parte fraccionaria
  - ✓La constante E = 1023

1.2.4 Tipos primitivos

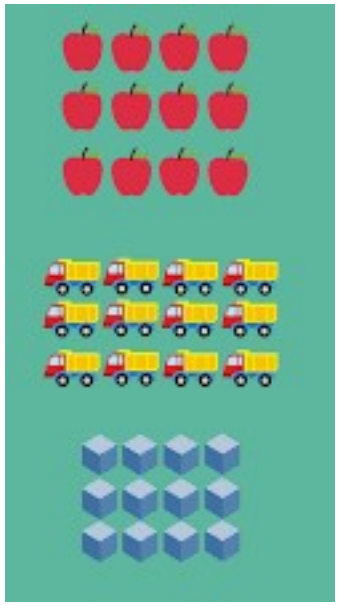
- Los tipos primitivos son los tipos de datos básicos que maneja cada lenguaje de programación para representar datos y almacenar valores.
- El manejo de los datos es una *capa de abstracción* que proporciona el lenguaje y que hace transparente para el usuario el manejo de este tipo de datos en los programas.

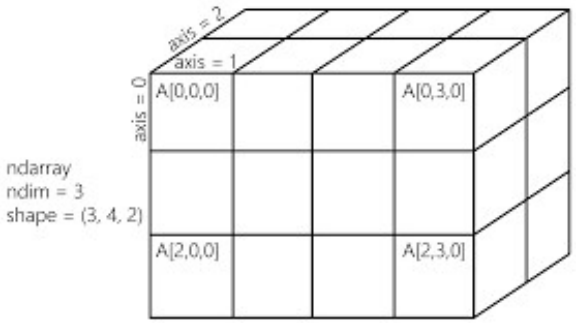
1.2.4 Tipos primitivos

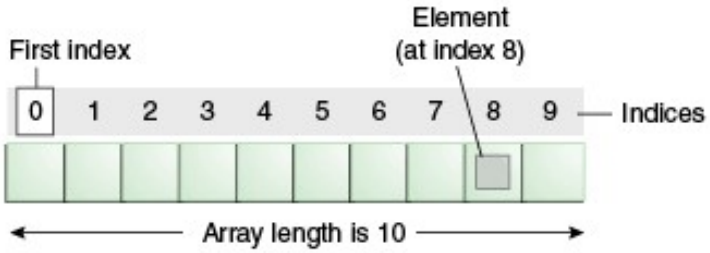
Tipo	Representación	Tamaño (bits)
char	Carácter	8
int	Entero con signo	32
short	Entero con signo	16
long	Entero con signo	32
float	Punto flotante simple	32
double	Punto flotante doble	64
unsigned	Entero positivo	32

1.2.4 Tipos primitivos

Tipo	Representación	Tamaño (bits)
bool	True or false	1
char	Carácter	16
byte	Entero con signo	8
short	Entero con signo	16
int	Entero con signo	32
long	Entero con signo	64
float	Punto flotante	32
double	Punto flotante	64




axis = 2
axis = 1
axis = 0
A[0,0,0]
A[0,3,0]
A[2,0,0]
A[2,3,0]
ndarray
ndim = 3
shape = (3, 4, 2)


First index
Element
(at index 8)
0 1 2 3 4 5 6 7 8 9
Indices
Array length is 10

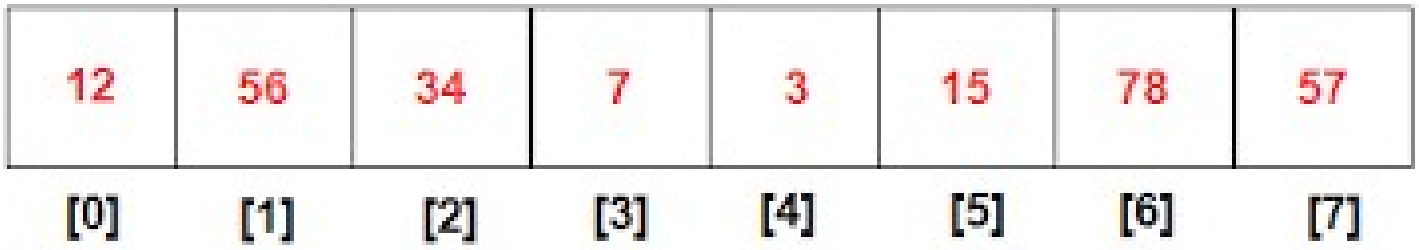
1.3 Arreglos

1.3.1 Definición y operaciones

- Un arreglo es una colección o conjunto de variables del mismo tipo que se asocian a un nombre común.
- Los arreglos constan de posiciones de memoria contiguas.
- Pueden tener una o varias dimensiones.
- Son entidades estáticas debido a que se conservan del mismo tamaño a lo largo de la ejecución del programa.

1.3.1 Definición y operaciones

- El manejo de los elementos de un arreglo se realiza con índices.
- Los índices indican la posición del elemento dentro del arreglo.
- En el índice es posible utilizar variables, constantes o expresiones aritméticas .



1.3.1 Definición y operaciones

```
tipoDeDato identificador[numeroElementos];
```

```
int c[20];
int arreglo[5];
char letra[10];

int[20] c;    //ERROR

int arreglo1[];      //ERROR
```

1.3.1 Definición y operaciones

Acceso a elementos en Arreglos

```
int arreglo[20];
arreglo[20] = 8;      //ERROR
int a = 2;
arreglo[a] = 2;
double b = 5.0;
arreglo[b]= 1;     //ERROR
arreglo[a*2] = 7;
```

1.3.1 Definición y operaciones

Acceso a elementos en Arreglos

```
printf("El valor es: %d",a[2]);
int suma = a[0] + a[3];
printf("Resultado = %d",suma);
a[1] = 25;
int x = 5;
a[x] = 30;
```

1.3.1 Definición y operaciones

Acceso a elementos en Arreglos

```
char letras[10];
letras[0] = 'a';
letras[1] = '}';
letras[2] = 64;
```

1.3.1 Definición y operaciones

Acceso a elementos en Arreglos

El índice de un arreglo incluso se puede referenciar con el índice de otro arreglo.

```
int arreglo1[5];
arreglo1[0] = 1;
arreglo1[1] = 2;
arreglo1[2] = 3;
int arreglo2[3];
arreglo2[arreglo1[1]] = 1;
```

1.3.1 Definición y operaciones

Inicialización:

```
int c[] = {11,2,8};
int c[3] = {11,2,8};
int c[8] = {10,20,30};
char cadena[] = "Hola";
char cadena[5] = "hola";
char cadena[15] ="adiós";
```

1.3.1 Definición y operaciones

Declaración de arreglos (Java)

```
byte[] edad = new byte[4];
short[] edad = new short[4];
int[] edad = new int[4];
float[] estatura = new float[3];
boolean[] estado = new boolean[5];
char[] sexo = new char[2];
String[] nombre = new String[2];
```

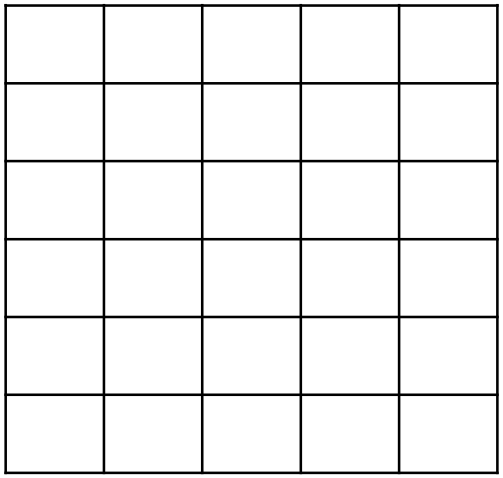
1.3.2 Arreglos unidimensionales

- Son los arreglos más simples y constan de un sólo índice, también suelen llamarse vectores.
- Como los elementos se almacenan de forma adyacente, su representación en la memoria es simple.

Dirección	Elemento
...	...
z	$elem\,(0)$
$z+1$	$elem\,(1)$
...	...
$z+n-1$	$elem\,(n-1)$
...	...

1.3.3 Matrices

- Se trata de arreglos que tienen 2 índices.
- Es el caso más sencillo de arreglos multidimensionales y tiene diversas aplicaciones computacionales.



1.3.3 Matrices

Declaración:

```
int c[3][3];
int arr[10,3];          //ERROR
int matriz[2][3];
int arreglo[5][3][1];
char letra[10][20][20][5];
```

1.3.3 Matrices

```
int a[5][5];

a[3][2]= 80;
a[2][3]= 80;
a[1][2]= 80;
a[1][4]= ?
a[4][4]= 200;
a[3][1]= 33;
```

✘
✘
✓
100

10	20	30	40	50
60	70	80	90	100
110	120	130	140	150

1.3.3 Matrices

Manejo práctico de arreglos

```
void main(){
     int i,arreglo[100];
     for (i = 0;i <= 99; i++){
          arreglo[i] = i;
     }
}
```

1.3.3 Matrices

Manejo práctico de arreglos

```
void main(){
    int i,j,num=2;
    int numeros[3][3];
    for (i=0;i<3;i++){
            for (j=0;j<3;j++){
                    numeros[i][j] = num;
                    num = num*2;
            }
    }
```

1.3.3 Matrices

- Dado un arreglo bidimensional, en la memoria se almacena ordenando primero los renglones y posteriormente las columnas

```
int a[3][2]
```

50	60
70	80
90	100

En memoria:

…
50
60
70
80
90
100
…

1.3.4 Arreglos Multidimensionales

Inicialización arreglos multidimensionales:

Existen 2 alternativas para inicializar arreglos multidimensionales

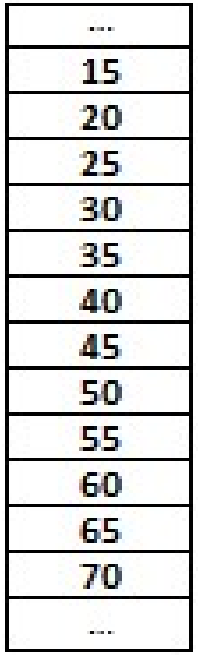
```
int c[3][3]={1,2,3,4,5,6,7,8,9};
int otro[3][3]= {{1,2,3},{4,5,6},{7,8,9}};

int unomas[2][4]={10,20,30,40,50,60,70,80};
int nuevo[2][4]={{10,20,30,40},{50,60,70,80}};
```

1.3.4 Arreglos Multidimensionales

- No importa cuántas dimensiones tenga un arreglo, su almacenamiento en memoria es contiguo

```
int a[2][2][3] = { 15,20,25,30,35,40,45,50,55,60,65,70}
```



Lo esencial en arreglos…

- Todos los elementos son del mismo tipo.
- Son estáticos.
- Se almacenan en localidades de memoria consecutivas.
- El primer índice de un arreglo siempre es 0.