

# Introducción al Lenguaje C

Arreglos, Cadenas de Caracteres y Acceso a Archivos

Claudio Omar Biale

Facultad de Ciencias Exactas Químicas y Naturales

Universidad Nacional de Misiones

22/08/2012 y 24/08/2012

## Arreglos

---

- Es una pieza contigua de memoria que puede contener varios valores.
- Se puede acceder de manera individual a los valores contenidos en un arreglo.
- Se utiliza un índice para acceder de manera individual a los elementos de un arreglo.

# Arreglos

---

□ Ejemplo:

|        |   |   |   |   |   |    |
|--------|---|---|---|---|---|----|
| notas  | 7 | 4 | 8 | 3 | 2 | 10 |
| índice | 0 | 1 | 2 | 3 | 4 | 5  |

- ▣ `notas[0]` es 7
- ▣ `notas[1]` es 4
- ▣ `notas[2]` es 8
- ▣ `notas[3]` es 3
- ▣ `notas[4]` es 2
- ▣ `notas[5]` es 10

## Declaración de un Arreglo

---

- La declaración de un arreglo tiene la siguiente sintaxis:

```
tipo_de_datos nombre_arreglo [capacidad];
```

- ▣ capacidad es el número de valores que se pueden almacenar.
- ▣ El índice comienza en cero, por ende va desde cero hasta capacidad-1.

## Operaciones sobre Arreglos

---

□ Declaración, asignación, acceso, operaciones, entrada y salida:

```
#include <stdio.h>

int main(void) {
    int enteros[10]; // arreglo de 10 enteros
    int valor;

    enteros[0] = 5;
    valor = enteros[0];
    enteros[2] = enteros[0] * 2;
    enteros[3] = enteros[0] + enteros[2];
    scanf("%d", &enteros[1]);
    printf("El primer elemento del arreglo es %d\n", enteros[0]);
    return 0;
}
```

## Operaciones sobre Arreglos

---

□ Declaración, asignación, acceso, operaciones, entrada y salida:

```
#include <stdio.h>

int main(void) {
    int enteros[10]; // arreglo de 10 enteros
    int valor;

    enteros[0] = 5;
    valor = enteros[0];
    enteros[2] = enteros[0] * 2;
    enteros[3] = enteros[0] + enteros[2];
    scanf("%d", &enteros[1]);
    printf("El primer elemento del arreglo es %d\n", enteros[0]);
    return 0;
}
```

■ Si ingresamos tres cuando llamamos a `scanf()` obtenemos:

|   |   |    |    |    |    |    |    |    |    |
|---|---|----|----|----|----|----|----|----|----|
| 5 | 3 | 10 | 15 | ¿? | ¿? | ¿? | ¿? | ¿? | ¿? |
|---|---|----|----|----|----|----|----|----|----|

## Operaciones sobre Arreglos

□ Declaración, asignación, acceso, operaciones, entrada y salida:

```
#include <stdio.h>

int main(void) {
    int enteros[10]; // arreglo de 10 enteros
    int valor;

    enteros[0] = 5;
    valor = enteros[0];
    enteros[2] = enteros[0] * 2;
    enteros[3] = enteros[0] + enteros[2];
    scanf("%d", &enteros[1]);
    printf("El primer elemento del arreglo es %d\n", enteros[0]);
    return 0;
}
```

▣ Si ingresamos tres cuando llamamos a `scanf()` obtenemos:

|   |   |    |    |    |    |    |    |    |    |
|---|---|----|----|----|----|----|----|----|----|
| 5 | 3 | 10 | 15 | ¿? | ¿? | ¿? | ¿? | ¿? | ¿? |
|---|---|----|----|----|----|----|----|----|----|

El valor de cualquier variable o elemento de un arreglo sin inicializar no está definido

## Recorrer un Arreglo

---

- La forma más simple de recorrer un arreglo es utilizando una sentencia `for`:

```
#include <stdio.h>

int main(void) {
    int valores[10], i;

    for (i = 0; i < 10; i++)
        valores[i] = 0;

    for (i = 0; i < 10; i++)
        printf("%d\n",valores[i]);

    return 0;
}
```



## Inicializar un Arreglo

---

- De igual forma que se puede inicializar con valores una variable cuando es declarada, también, es posible asignar valores iniciales a los elementos de un arreglo.
- Se deben listar los valores iniciales de un arreglo, empezando por el primer elemento.

```
int x[3] = {0, 1, 2}  
  
char v[3] = {'a', 'b', 'c'}
```

x = 

|   |   |   |
|---|---|---|
| 0 | 1 | 2 |
|---|---|---|

v = 

|   |   |   |
|---|---|---|
| a | b | c |
|---|---|---|

## Inicializar un Arreglo

---

- No es necesario que se inicialicen todos los valores de un arreglo.
  - ▣ Si un menor número de valores iniciales se especifica, solo un número igual de elementos se inicializan.
  - ▣ Los elementos restantes se inicializan en `cero`.

```
int y[5] = {4, 3}
```

y = 

|   |   |   |   |   |
|---|---|---|---|---|
| 4 | 3 | 0 | 0 | 0 |
|---|---|---|---|---|

## Inicializar un Arreglo

---

- Al encerrar un número de elemento entre corchetes, se pueden inicializar elementos específicos del arreglo (*No importa su orden*).

```
int z[7] = {[2] = 80, [5] = 10}
```

z =

|   |   |    |   |   |    |   |
|---|---|----|---|---|----|---|
| 0 | 0 | 80 | 0 | 0 | 10 | 0 |
|---|---|----|---|---|----|---|

- C no dispone de ningún mecanismo rápido que permita de inicializar elementos de un arreglo con un valor repetido.
  - ▣ Para ello debemos usar una sentencia repetitiva.

## Buena Práctica

---

- Usar `#define` para definir el tamaño de un arreglo.

```
#include <stdio.h>
#define LIMITE 10

int main(void) {
    int valores[LIMITE], i;

    for (i = 0; i < LIMITE; i++)
        valores[i] = 0;

    for (i = 0; i < LIMITE; i++)
        printf("%d\n",valores[i]);

    return 0;
}
```

- Si necesito cambiar el tamaño del arreglo solo modifico el valor de la constante.

## Arreglos como Parámetros de Funciones

---

- Es posible pasar arreglos como parámetros de funciones.
- Siempre se pasan por referencia.
- Podemos pasar un arreglo en el cual:
  - ▣ indicando su tamaño o
  - ▣ sin indicar su tamaño.

## Ejemplo

---

- Realizar una función `menor_valor` que retorne el menor valor contenido en un arreglo de enteros.
- El programa sin la especificación de la función es:

```
#include <stdio.h>
#define LIMITE 7

int main(void) {
    int valores[LIMITE] = {10, 3, -23, 2}

    printf("%d\n", llamada_a_menor_valor);
    return 0;
}
```

## Ejemplo

---

### □ Programa:

```
#include <stdio.h>
#define LIMITE 7

int main(void) {
    int valores[LIMITE] = {10, 3, -23, 2};
    int menor_valor(int valores[]); // prototipado

    printf("%d\n", menor_valor(valores));
    return 0;
}

int menor_valor(int valores[LIMITE]) {
    int i, menor = valores[0];
    for (i = 1; i < LIMITE; i++)
        if (valores[i] < menor)
            menor = valores[i];
    return menor;
}
```

## Ejemplo

---

### □ Variante 1:

```
#include <stdio.h>
#define LIMITE 7

int main(void) {
    int valores[LIMITE] = {10, 3, -23, 2};
    int menor_valor(int valores[]); // prototipado

    printf("%d\n", menor_valor(valores));
    return 0;
}

int menor_valor(int valores[]) {
    int i, menor = valores[0];
    for (i = 1; i < LIMITE; i++)
        if (valores[i] < menor)
            menor = valores[i];
    return menor;
}
```



## Ejemplo

---

### □ Variante 2:

```
#include <stdio.h>
#define LIMITE 7

int main(void) {
    int valores[LIMITE] = {10, 3, -23, 2};
    int menor_valor(int valores[], int t); // prototipado

    printf("%d\n", menor_valor(valores, LIMITE));
    return 0;
}

int menor_valor(int valores[], int t) {
    int i, menor = valores[0];
    for (i = 1; i < t; i++)
        if (valores[i] < menor)
            menor = valores[i];
    return menor;
}
```

## Arreglos Multidimensionales

---

- La declaración de un arreglo de dos dimensiones tiene la siguiente sintaxis:

```
tipo_de_datos nombre_arreglo [filas][columnas];
```

- Físicamente es un único bloque de memoria.
- Los elementos son almacenados de acuerdo a la fila a la que pertenecen, en primer lugar la fila 1 hasta llegar a la fila N.

## Inicializar Arreglos Multidimensionales

---

- La inicialización de arreglos de dos dimensiones es análoga a los arreglos de una dimensión.
  - ▣ Cuando se listan los elementos a inicializar, los valores son listados por fila
  - ▣ Se utilizan llaves para separar los elementos inicializados en distintas filas.

```
int M[2][3] = {  
    {10, 5, 0},  
    {3, 6, 9}  
};
```

M =

|    |   |   |
|----|---|---|
| 10 | 5 | 0 |
| 3  | 6 | 9 |

## Inicializar Arreglos Multidimensionales

---

- Una variante a la inicialización completa de un arreglo es indicar todos los valores en una única lista.

```
int M1[2][3] = {10, 10, 10, 3, 3, 3};
```

M1 =

|    |    |    |
|----|----|----|
| 10 | 10 | 10 |
| 3  | 3  | 3  |

## Inicializar Arreglos Multidimensionales

---

- También es posible inicializar solo algunos elementos de un arreglo multidimensional.

```
int M2[2][3] = {  
    {10, 10},  
    {3}  
};  
  
int M3[2][2] = { [0][0] = 1, [1][1] = 1};
```

M2 =

|    |    |   |
|----|----|---|
| 10 | 10 | 0 |
| 3  | 0  | 0 |

M3 =

|   |   |
|---|---|
| 1 | 0 |
| 0 | 1 |

## Ejemplo

---

□ Inicialización de un arreglo multidimensional e impresión de sus elementos:

```
#include <stdio.h>

int main(void) {
    int M[2][3] = {
        {10, 5, 0},
        {3, 6, 9}
    };
    int i, j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            printf("M[%d][%d] = %d\n", i, j, M[i][j]);

    return 0;
}
```

## Arreglos Multidimensionales y Funciones

---

- El tamaño de la primer dimensión de un arreglo multidimensional se puede omitir, pero la declaración de la segunda (y *subsiguientes, si existieran*) son obligatorias.

```
#include <stdio.h>
#define FILAS 2
#define COLUMNAS 3

void imprimir(int v[][COLUMNAS]) {
    int i, j;
    for (i = 0; i < FILAS; i++)
        for (j = 0; j < COLUMNAS; j++)
            printf("[%d][%d] = %d\n", i, j, v[i][j]);
}

int main(void) {
    int M[FILAS][COLUMNAS] = {{10, 5, 0},{3, 6, 9}};
    imprimir(M);
    return 0;
}
```

## Cadena de Caracteres

---

- Una cadena de caracteres (*string*) es un arreglo de caracteres.

```
char cadena[50];
```

- Las constantes de cadenas de caracteres están delimitadas por `"`.

- No es posible utilizar el operador de asignación para asignar una cadena de caracteres a un arreglo de caracteres.

```
cadena = "HOLA MUNDO"; // ERROR
```

- Si es posible asignar valores a los elementos del arreglo.

```
cadena[0] = ' ';
```

- Una cadena de caracteres nula se representa por `"`.



## Cadena de Caracteres

---

- Un carácter nulo indica el final de una cadena de caracteres.

|   |   |   |   |   |   |    |
|---|---|---|---|---|---|----|
| C | A | D | E | N | A | \0 |
|---|---|---|---|---|---|----|

- Las funciones que manejan cadena de caracteres usan el caracter nulo para ubicar el final de una cadena de caracteres.

## Inicializar una Cadena de Caracteres

---

- Especificando el tamaño del `string`, cada elemento del arreglo y agregando el caracter nulo.

```
char s[10] = {'p', 'e', 'p', 'e', '\0'};
```

s =

|   |   |   |   |    |  |  |  |  |  |
|---|---|---|---|----|--|--|--|--|--|
| P | E | P | E | \0 |  |  |  |  |  |
|---|---|---|---|----|--|--|--|--|--|

- ▣ Cinco elementos del arreglo son ocupados

## Inicializar una Cadena de Caracteres

---

- Especificando el tamaño del `string` e inicializando mediante una constante de cadena de caracteres.

```
char s1[10] = "RAUL";
```

s1 = 

|   |   |   |   |    |  |  |  |  |  |
|---|---|---|---|----|--|--|--|--|--|
| R | A | U | L | \0 |  |  |  |  |  |
|---|---|---|---|----|--|--|--|--|--|

- ▣ El caracter nulo es agregado al final de manera automática.
- ▣ Cinco elementos del arreglo son ocupados

## Inicializar una Cadena de Caracteres

---

- Sin especificar el tamaño del `string` e inicializando mediante una constante de cadena de caracteres.

```
char s2[] = "RAUL";
```

s2 = 

|   |   |   |   |    |
|---|---|---|---|----|
| R | A | U | L | \0 |
|---|---|---|---|----|

- ▣ El caracter nulo es agregado al final de manera automática.
- ▣ El espacio necesario para almacenar el `string` es asignado de manera automática
- ▣ El tamaño del arreglo es de cinco elementos.

## Impresión de Cadena de Caracteres

---

- Para imprimir una cadena de caracteres se utiliza la función `printf()` con el descriptor `%s`.
- Si se quiere imprimir un elemento de la cadena de caracteres se utiliza `%c`.

```
#include <stdio.h>

int main(void) {
    char cadena[] = "abc123";

    printf("%s\n", cadena);
    printf("%c\n", cadena[2]);
    return 0;
}
```

## Ingreso de Cadena de Caracteres

---

- Existen distintas funciones disponibles para el ingreso de cadena de caracteres:
  - ▣ `scanf()`
  - ▣ `fgets()`

## Función scanf()

---

- `scanf()` lee hasta el primer caracter en blanco encontrado, ignorando lo escrito luego de dicho caracter.
- Es conveniente utilizar `scanf()` con el descriptor `%[tamaño]s` para obtener solamente un numero máximo de caracteres.
  - ▣ Ejemplo: `%9s`
  - ▣ Se agrega de manera automática el caracter nulo al final.
- Si se quiere leer una cadena de caracteres hasta el final de línea se utiliza el descriptor `%[^\n]`.
  - ▣ Se puede complementar con el descriptor de tamaño máximo de caracteres a obtener.

## Función scanf()

---

### □ Ejemplo de uso de scanf():

```
#include <stdio.h>

int main(void) {
    char cadena[10];

    scanf("%9s", cadena);
    printf("%s\n", cadena);

    return 0;
}
```



## Función fgets()

---

□ Su formato resumido es:

```
char * fgets(arreglo_caracteres, límite, entrada);
```

- ▣ Definida en el archivo `stdio.h`
- ▣ Obtiene toda una línea, hasta llegar al `límite` o hasta encontrar una nueva línea.
- ▣ Almacena en la cadena de caracteres el caracter de nueva línea: `\n`.
- ▣ Agrega el caracter nulo al final de la cadena de caracteres.
- ▣ Retorna un puntero al arreglo si se ejecuta correctamente o `NULL` si se produce un error.

## Función fgets()

---

### □ Ejemplo de uso de fgets():

```
#include <stdio.h>

int main(void) {
    char cadena[10];

    fgets(cadena, 10, stdin);
    printf("%s\n", cadena);
    return 0;
}
```

## Apertura de Archivos

---

□ `fopen()` permite la apertura de un archivo, su formato resumido es:

```
FILE * fopen(nombre_archivo, modo_apertura);
```

- ▣ Pertenece a `stdio.h`
- ▣ Recibe como parámetros el nombre de archivo del archivo a abrir y el modo de apertura.
- ▣ Si se ejecuta de manera correcta retorna un puntero que será utilizado para las siguientes operaciones de entrada y salida sobre el archivo.
- ▣ En caso de error retorna un puntero nulo (*NULL*).

## Apertura de Archivos

---

- `fopen()` permite varios modos de apertura entre los que encontramos:
  - ▣ `"r"` : lectura, puntero al comienzo del archivo.
  - ▣ `"w"` : escritura, puntero al comienzo del archivo.<sup>1</sup>
  - ▣ `"a"` : agregado de datos, puntero al final del archivo.
  - ▣ `"r+"` : lectura y escritura, puntero al comienzo del archivo.<sup>2</sup>
  - ▣ `"w+"` : lectura y escritura, puntero al comienzo del archivo.<sup>1</sup>
  - ▣ `"a+"` : lectura y agregado de datos, puntero al comienzo del archivo, escritura de datos al final del archivo.<sup>2</sup>
  
- La apertura en modo exclusivo `"x"` y sus variantes se definió en C11.<sup>3</sup>

---

<sup>1</sup> El archivo es creado si no existe, en caso de existir se trunca.

<sup>2</sup> El archivo es creado si no existe.

<sup>3</sup> Mirar borrador de C11.

## Cierre de Archivos

---

□ `fclose()` cierra un archivo, su formato resumido es:

```
int fclose(puntero_archivo);
```

- ▣ Pertenece a `stdio.h`
- ▣ Recibe como parámetro el puntero al archivo que se quiere cerrar.
- ▣ Si se ejecuta de manera correcta retorna `cero`.
- ▣ En caso de error retorna `EOF`.
- ▣ Independientemente del valor retornado el comportamiento si se quiere usar posteriormente un puntero que se intento cerrar con `fclose()` es indefinido.

## Lectura y Escritura de Archivos

---

- `fscanf()` permite obtener datos de un archivo:

```
int fscanf(puntero_archivo, formato, argumentos);
```

- `fprintf()` permite escribir datos en un archivo:

```
int fprintf(puntero_archivo, formato, argumentos);
```

- ▣ Definidas en `stdio.h`
- ▣ Son análogas a `scanf()` y `printf()`.

## Verificar Final de Archivo

---

- `feof()` verifica si se trata de leer luego del final de un archivo:

```
int feof(puntero_archivo);
```

- ▣ Pertenece a `stdio.h`
- ▣ Retorna un valor distinto de cero cuando la verificación es verdadera y cero si es falsa.

## Indicar Posición al Puntero de Lectura/Escritura

---

□ `fseek()` determina la posición del apuntador :

```
int fseek(puntero_archivo, desplazamiento, lugar);
```

- ▣ Pertenece a `stdio.h`
- ▣ Desplaza la cantidad de posiciones indicada por `desplazamiento` desde `lugar`.
- ▣ Lugar puede tomar los valores `SEEK_SET` (*comienzo*), `SEEK_CUR` (*actual*) o `SEEK_END` (*final*).
- ▣ Retorna un valor `cero` cuando se puede realizar la acción y `menos uno` en caso de error.



# Introducción al Lenguaje C

Arreglos, Cadenas de Caracteres y Acceso a Archivos

Claudio Omar Biale

Facultad de Ciencias Exactas Químicas y Naturales

Universidad Nacional de Misiones

22/08/2012 y 24/08/2012