

# LENGUAJE C

## Tema 2 – Elementos de un programa

# ELEMENTOS DE UN PROGRAMA

- ❖ **Comentarios.**
- ❖ **Identificadores.**
- ❖ **Constantes.**
- ❖ **Variables.**
- ❖ **Operadores.**
- ❖ **Sentencias o instrucciones.**

# COMENTARIOS

Los comentarios en C pueden ocupar varias líneas y se encuentran delimitados entre `/*` y `*/`. Estos son ignorados por el compilador.

```
Int main()  
{  
    /* Esto es un comentario de varias  
        líneas.*/  
    return 0;  
}
```

# IDENTIFICADORES

- ❖ Se utilizan para nombrar variables, funciones, etiquetas y elementos definidos por el usuario.
- ❖ Los primeros seis caracteres deben ser significativos (distinguirse de otro similar) y máximo puede tener hasta 31 caracteres.
- ❖ El primer carácter debe de ser una letra o subguión. Posteriormente pueden ser letras, números, signos de subrayado.
- ❖ Existe diferencia entre mayúsculas y minúsculas.
- ❖ No pueden emplearse palabras reservadas como identificadores.
- ❖ No pueden emplearse nombres de funciones que ya existan en el programa o en la librería de funciones de C.
- ❖ No puede llamarse main.

## Convenciones:

- Empezar los nombres de funciones y de variables con una letra minúscula.
- Las constantes escritas con #define van con mayúsculas como #define PI 3.1416
- Las palabras intermedias comienzan con mayúsculas. sumaMatrices
- Utilizar el subguión para separar palabras intermedias. suma\_Matrices
- Emplear nombres cortos para optimizar. (i, j, k, cont)

# DELIMITADORES

Son símbolos que permiten al compilador separar y reconocer las diferentes unidades sintácticas del lenguaje.

**Los principales delimitadores son:**

- ❖ `;` es necesario para finalizar sentencias o declaraciones.
- ❖ `,` separa dos elementos consecutivos de una lista.
- ❖ `()` enmarca una lista de parámetros.
- ❖ `[]` enmarca la dimensión o el subíndice de una tabla.
- ❖ `{}` enmarca un bloque de instrucciones o una lista de valores iniciales.

# CONSTANTES

- ❖ Se utiliza para asignar un identificador a una constante, cuyo valor no se modificará durante toda la ejecución del programa.

```
#define PI 3.1416
```

```
#define NCOLS 20
```

- ❖ El pre-procesador de C, sustituye la ocurrencia de PI por el valor 3.1416 en todo el programa antes de efectuar la compilación, del mismo modo se sustituye NCOLS por 20, o sea, no se guarda memoria para la misma.

# VARIABLES

Una variable es una **posición de memoria** cuyo valor puede ser **cambiado** durante la ejecución del programa.

Inicialmente el valor de una variable es indeterminado.

Todas las variables deben de ser **declaradas** para ser utilizadas.

<tipo de dato> <identificador>;  
<tipo de dato> <identificador> = <valor>;

```
int nro;  
float area, radio,  
      volumen;  
char letra = 'a';  
double iva = 0,21;
```

## Las variables pueden ser declaradas:

- ❖ Fuera de las funciones: Variables globales
- ❖ Dentro de una función: Variable local
- ❖ En la cabecera de una función: parámetros formales de la función (es una variable local)

# MODIFICADORES DE ACCESO

- ❖ **Const** permite asignar a una variable un **valor constante**, es decir que una vez asignado a dicha variable su valor no podrá ser modificado durante el programa.

```
const <tipo dato> <identificador> = valor;
```

```
const int a=10;  
const char pais[]="ARGENTINA";
```

- ❖ **Volatile** permite cambiar el valor de una variable por **medios no explícitamente** especificados por el programa. Por ejemplo la dirección de una variable global que apunta a un puerto externo.

```
volatile <tipo dato> <identificador> = valor;
```

```
volatile unsigned char *puerto = 0x30;
```



# TIPOS DE DATOS BÁSICOS

## Existen 5 tipos de datos básicos

- ❖ **Entero**
  - ❖ Int
- ❖ **Real**
  - ❖ Float (Simple precisión)
  - ❖ Double (Doble precisión)
- ❖ **Carácter**
  - ❖ Char (representación interna en ASCII)
- ❖ **Tipo vacío que se utiliza cuando una función no devuelve ningún valor**
  - ❖ Void
- ❖ **El tipo de datos lógico se implementa como entero**
  - ❖ Cualquier valor distinto de 0 es verdad
  - ❖ Cualquier valor igual a 0 es falso

**El tamaño y el rango de cada uno depende del compilador**

**La función sizeof() permite conocer el tamaño de cada tipo**

Ciertos tipos básicos admiten diversos modificadores:

- **unsigned** : Sólo valores positivos (sin signo).
- **signed** : Valores positivos y negativos (por omisión).
- **long** : Formato largo (solo enteros).
- **Short**: Formato reducido (solo enteros).

Ejemplos:

```
unsigned int
signed char
long int (usualmente
representado como long)
unsigned long int
```

# CARACTERES ESPECIALES

|                 |                            |
|-----------------|----------------------------|
| <code>\0</code> | Fin de cadena              |
| <code>\n</code> | Nueva línea                |
| <code>\t</code> | Tabulado                   |
| <code>\b</code> | Retroceso                  |
| <code>\r</code> | Retorno de carro           |
| <code>\f</code> | Salto de página            |
| <code>\\</code> | Contra barra               |
| <code>\'</code> | Apóstrofe (comilla simple) |
| <code>\"</code> | Comillas                   |
| <code>\a</code> | Audio de alerta            |

# OPERADORES

Son palabras o símbolos que implican una acción sobre ciertas variables. Pueden ser unarios (1 variable), binarios(2 variables) o ternarios (3 variables).

- ❖ Operadores Aritméticos
- ❖ Operadores Relacionales
- ❖ Operadores Lógicos
- ❖ Operadores de Asignación
- ❖ Operadores de Bits
- ❖ Operadores de Dirección

# OPERADORES ARITMÉTICOS

| Operador       | Nombre         | Descripción                        |
|----------------|----------------|------------------------------------|
| +              | Suma           | $5+2 \rightarrow 7$                |
| -              | Resta          | $5-2 \rightarrow 3$                |
| *              | Multiplicación | $5*2 \rightarrow 10$               |
| /              | División       | $5/2 \rightarrow 2$                |
| %              | Módulo         | $5\%2 \rightarrow 1$               |
| (tipo de dato) | “Cast” forzado | $(\text{double})5 \rightarrow 5.0$ |

# OPERADORES RELACIONALES

| Operador           | Nombre            | Descripción                |
|--------------------|-------------------|----------------------------|
| <code>==</code>    | Igual a           | <code>if (a=='s')</code>   |
| <code>!=</code>    | Diferente de      | <code>if (a!=null)</code>  |
| <code>&gt;</code>  | Mayor que         | <code>if (a&gt;0.5)</code> |
| <code>&lt;</code>  | Menor que         | <code>if (a&lt;2l)</code>  |
| <code>&gt;=</code> | Mayor o igual que | <code>if (a&gt;=2f)</code> |
| <code>&lt;=</code> | Menor o igual que | <code>if (a&lt;=3)</code>  |

# OPERADORES LÓGICOS

| Operador | Nombre       | Descripción                          |
|----------|--------------|--------------------------------------|
| &&       | Y (AND)      | if ((a>3) && (a<9))                  |
|          | O (OR)       | if ((a==2)    (a==3))                |
| !        | NEGADO (NOT) | if (!(a==3)) es igual a<br>if (a!=3) |

Importante:

**FALSO** es igual a cero.

**VERDADERO** es diferente de cero.

# OPERADORES DE ASIGNACIÓN

| Operador | Abreviado   | No Abreviado |
|----------|-------------|--------------|
| =        | a=2;        | a=2;         |
| ++       | n++; o ++n; | n=n+1;       |
| -        | n-; o --n;  | n=n-1;       |
| +=       | n+=2;       | n=n+2;       |
| -=       | n-=2;       | n=n-2;       |
| *=       | n*=2;       | n=n*2;       |
| /=       | n/=2;       | n=n/2;       |
| %=       | n%=2;       | n=n%2;       |

# OPERADORES DE BITS

| Operador | Nombre                     | Descripción                |
|----------|----------------------------|----------------------------|
| <<       | Corrimiento a la izquierda | <code>b=a&gt;&gt;2;</code> |
| >>       | Corrimiento a la derecha   | <code>b=a&lt;&lt;3;</code> |
| &        | Y (AND) entre bits         | <code>c=a&amp;128;</code>  |
|          | O (OR) entre bits          | <code>c=a 0x0a;</code>     |
| ~        | Complemento A1             | <code>c=~a;</code>         |
| ^        | O exclusivo (XOR)          | <code>c=^a;</code>         |



# OPERADORES DE DIRECCIÓN

| Operador | Nombre               | Descripción   |
|----------|----------------------|---|
| *        | Operador indirección | Me da el valor que está almacenado en una dirección de memoria. También sirve para declarar un puntero. |
| &        | Operador dirección   | Me da la dirección de memoria de una variable.  |

## Otros operadores:

- ❖ Evaluación condicional  
`( a > b ) ? a : b ;`
- ❖ Tamaño en bytes  
`sizeof ( variable ); sizeof ( tipo );`

# PRECEDENCIA DE OPERADORES

|  |                              |
|--|------------------------------|
| <code>() [] -&gt;</code>   | Alta prioridad               |
| <code>! ~ + - ++ - &amp; * sizeof</code>                                       | Unarios                      |
| <code>* / % + -</code>   | Aritméticos                  |
| <code>&lt;&lt; &gt;&gt;</code>   | Corrimiento de bits          |
| <code>&lt; &lt;= &gt; &gt;= == !=</code>                                       | Relacionales                 |
| <code>&amp; ^   &amp;&amp;    ?:</code>  | Bits / Lógicos / Condicional |
| <code>= *= /= %= += -= &amp;=</code><br><code>^=  = &lt;&lt;= &gt;&gt;=</code> | Asignación                   |
| <code>,</code>   | Evaluación                   |

# SENTENCIAS (INSTRUCCIONES)

- ❖ Una sentencia es una **instrucción o expresión** en C que tiene una consecuencia. Pueden ser asignaciones, operaciones, llamadas a funciones.
- ❖ Todas las sentencias terminan con el signo de punto y coma ;
- ❖ Pueden ser **simples o compuestas**. Las compuestas van entre llaves:

```
{  
    sentencia1;  
    sentencia2;  
    :  
    sentencian;  
}
```

- Sentencias de Selección:  
    **if – else, switch – case, ?:**
- Sentencias de Repetición:  
    **do – while, while, for**
- Sentencias de Salto:  
    **return, break, continue.**

# SENTENCIAS DE SELECCIÓN IF - ELSE

```
if (expresión)
    sentencia;
else
    sentencia;
```

**Nota:** una expresión en C es todo aquello que regresa un valor. Como por ejemplo una condición lógica, operaciones aritméticas, llamadas a funciones, una variable, una constante (numérica, carácter, etc.).

```
if (expresión)
{
    sentencia1;
    sentencia2;
}
else
    if (expresión2)
        sentencia;
    else
        sentencia;
```

# SENTENCIAS DE SELECCIÓN SWITCH - CASE

```
switch(expresión)
{
    case 1:  sentencias;
            break;
    case 2:  sentencias;
            break;
    :
    case n:  sentencias;
            break;
    default: sentencias_default;
            break;
}
```

# SENTENCIAS DE REPETICIÓN WHILE Y DO - WHILE

## WHILE

```
while (expresión)  
    sentencia;
```

```
while (expresión){  
    sentencian;  
    ...  
    sentencian;  
}
```

## Do - WHILE

```
do sentencia;  
while(expresión);
```

```
do{  
    sentencian;  
    ...  
    sentencian;  
}while(expresión);
```

# SENTENCIAS DE REPETICIÓN FOR

```
for (inicialización; condición; incremento)  
acción;
```

```
for (inicialización; condición; incremento){  
acciones;  
}
```

```
for(expr1;expr2;expr3)  
Sentencia;
```

- expr1 y expr 3 son asignaciones o llamadas a funciones.

- expr2 es una expresión relacional.

- Si expr2 no esta presente se considera siempre verdadero →bucle infinito, unica salida break o return.

# SENTENCIAS DE SALTO BREAK Y CONTINUE

## ❖ **break:**

- Permite controlar las salidas de los bucles.
- Provee una salida temprana para for, while, do, switch.

```
for(i=0;i<1000;i++){  
    //hacer algo  
    if(kbhit()) break;  
}
```

## ❖ **continue:**

- Es utilizada cuando la parte del bucle que sigue es complicada.
- Provoca la próxima iteración del bucle cerrado a ser ejecutado inmediatamente.



# WEBGRAFÍA & LICENCIA:

- ❖ Textos tomados, corregidos y modificados de diferentes páginas de Internet, tutoriales y documentos, entre los que destaco el libro: *C/C++ Curso de programación*, 2da Ed, Javier Ceballos, Alfaomega Ra-Ma.
- ❖ Este documento se encuentra bajo Licencia Creative Commons 2.5 Argentina (BY-NC-SA), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del Prof. Matías E. García y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.

❖ Autor:

*Matías E. García*

Prof. & Tec. en Informática Aplicada

[www.profmatiasgarcia.com.ar](http://www.profmatiasgarcia.com.ar)

[info@profmatiasgarcia.com.ar](mailto:info@profmatiasgarcia.com.ar)

 creative  
commons



[www.profmatiasgarcia.com.ar](http://www.profmatiasgarcia.com.ar)