

# Algoritmos y Estructura de Datos I


**Tema 3- Estructura  
y componentes de  
un programa**

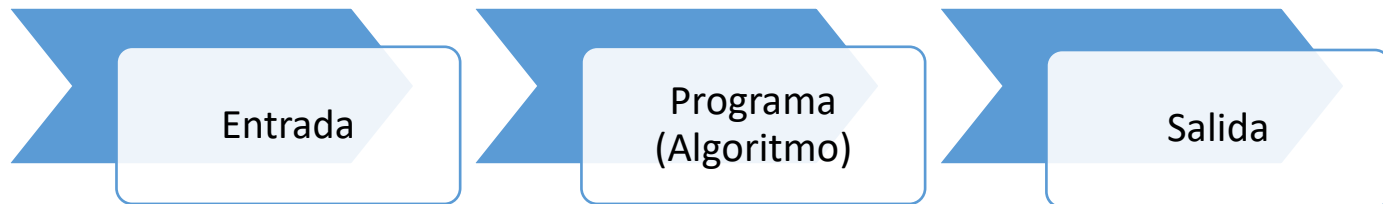


# Resolución de problemas

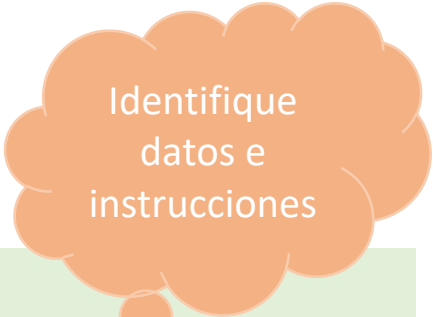
Resolución de problema = **programa**

**Programa = Datos + instrucciones**

- 
- ```
graph LR; A[Entrada] --> B[Programa (Algoritmo)]; B --> C[Salida]
```
- Diagrama de flujo de procesamiento de datos:
- Entrada
  - Programa (Algoritmo)
  - Salida



# EJEMPLO DE PROGRAMA



Identifique  
datos e  
instrucciones

```
include <stdio.h>
/* Declaración de prototipos de
funciones */
void ingresarDatos();
void calcularPromedio();
/* Declaración de variables globales
*/
int lu;
float nota1, nota2, promedio;
/* Programa principal*/
int main(){
    ingresarDatos();
    while(nota1 != 0) {
        calcularPromedio();
        ingresarDatos();
    }
    return 0;
}
```

```
/* Implementación de funciones */
void ingresarDatos() {
    printf("\nIngrese nro de LU: ");
    scanf("%d", &lu);
    printf("Ingrese nota 1: ");
    scanf("%f", &nota1);
    printf("Ingrese nota 2: ");
    scanf("%f", &nota2);
}

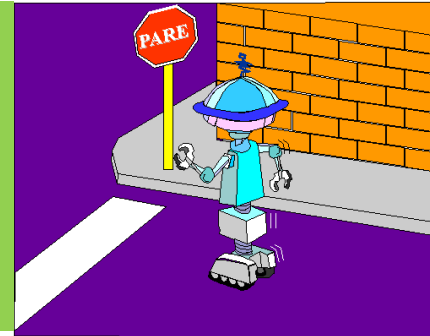
void calcularPromedio() {
    promedio = (nota1 + nota2)/2;
    printf("\nLU: %d\n", lu);
    printf("El promedio de notas es:
%.2f\n", promedio);
}
```

# Componentes de un programa

## 1- Instrucciones y tipos de instrucciones

- El proceso de *diseño del algoritmo* consiste en definir las **ACCIONES o INSTRUCCIONES** que resolverán el problema

Las *acciones o instrucciones* se deben escribir y posteriormente almacenar en memoria en el mismo orden en que han de ejecutarse.



Las instrucciones son equivalentes al concepto de “primitiva” de PilasBloques.

# 1- Instrucciones y tipos de instrucciones

**PilasBloques** Principal Desafíos Acerca de <Program.AR/>

Ejecutar Abrir Guardar

**Primitivas**  
Mis procedimientos

Saludar ejecutar

Avanzar

Volver

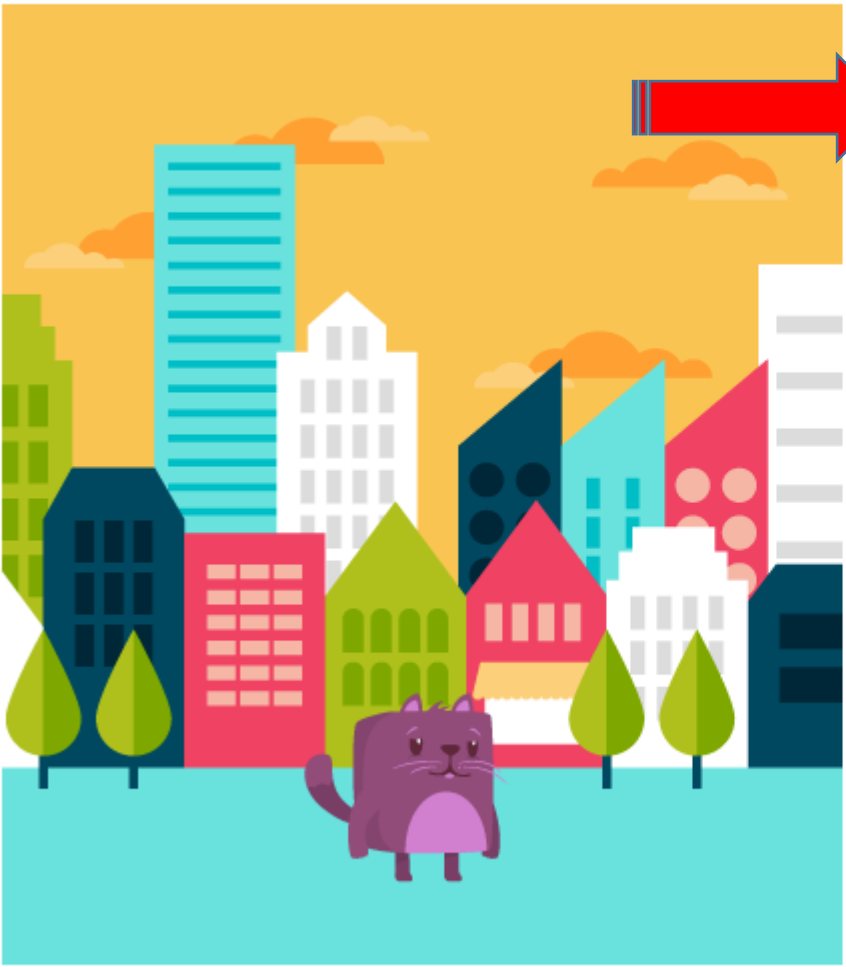
Abrir ojos

Cerrar ojos

Acostarse

Pararse

Soñar



El gato en la calle

# Tipos de instrucciones

| Tipo de instrucción       | Seudocódigo inglés          | Seudocódigo español        |
|---------------------------|-----------------------------|----------------------------|
| Comienzo de proceso       | BEGIN                       | INICIO                     |
| Fin de proceso            | END                         | FIN                        |
| Entrada (lectura)         | READ                        | LEER                       |
| Salida (escritura)        | WRITE                       | ESCRIBIR                   |
| Asignación                | A=5 ( A:=5)<br><i>A ← 5</i> | B=7 (B:=7)<br><i>B ← 7</i> |
| Bifurcación condicional   | IF                          | SI                         |
| Bifurcación incondicional | GOTO                        | IR                         |

# Ejemplo de los distintos tipos de instrucciones

**ALGORITMO** promedioNotas

**VAR**

**ENTERO:** LU, nota1, nota2

**REAL:** promedio

**INICIO**

**LEER** LU, nota1, nota2

**MIENTRAS** nota1 <> 0

promedio = (nota1 + nota2)/2

**ESCRIBIR** promedio

**LEER** LU, nota1, nota2

**FIN-MIENTRAS**

**FIN**



# Instrucciones de Fin y de Inicio

- Estas instrucciones indican a la computadora el inicio del programa y la finalización del mismo.
- La indicación de fin de programa **puede estar en cualquier lugar del programa** y puede haber más de una instrucción de fin, sujeta a las condiciones que evalúa el programa.

Por ejemplo, se finaliza el programa cuando se procesaron todos los datos de un conjunto de datos o cuando se encuentra un error severo de datos en una operación aritmética, etc

# Instrucciones de lectura (entrada)

- Las instrucciones de lectura dan ingreso a los datos que se procesan, esto es, indican las variables o posiciones de memoria en las que se almacenarán los datos en el momento de la ejecución del programa.

- Ejemplo: **LEER A, B, C**

¿Cuál es el significado de esta instrucción?

- Si se ingresan por teclado los valores 100, 50 y 30, entonces las variables de lectura tomarán los valores:

**A=100, B=50, C=30**

Una instrucción de lectura deposita valores en las **posiciones de memoria** indicadas por los nombres de **variables**.



# Instrucciones de escritura (salida)

- Esta instrucción escribe datos en un dispositivo de salida.
- Cuál es el significado de la instrucción siguiente?
- **ESCRIBIR A, B, C**
- Se mostrarán en pantalla los datos 100, 50 y 30

Una instrucción de escritura toma los valores de las posiciones de memoria indicadas por los nombres de variables y visualiza o registra ese valor en algún dispositivo de salida: pantalla de monitor, salida o dispositivos de almacenamiento externo: disco, pendrive, etc.

Si no se especifica el dispositivo, asume pantalla.



# Ejemplo de lectura y escritura

*/\* Ingresa 2 dos notas de exámenes y calcula el promedio \*/*

```
#include <stdio.h>
int main() {
    float nota1, nota2, promedio;
    printf("Ingresa nota correspondiente al primer parcial ");
    scanf("%f",&nota1);
    printf("Ingresa nota correspondiente al segundo parcial ");
    scanf("%f",&nota2);
    promedio=(nota1 + nota2)/2;
    printf("El Promedio de Notas es: %.2f",promedio);
    return 0;
}
```

# Ejemplo de lectura y escritura

- Lectura

```
Ingrese nota correspondiente al primer parcial
```

- Escritura

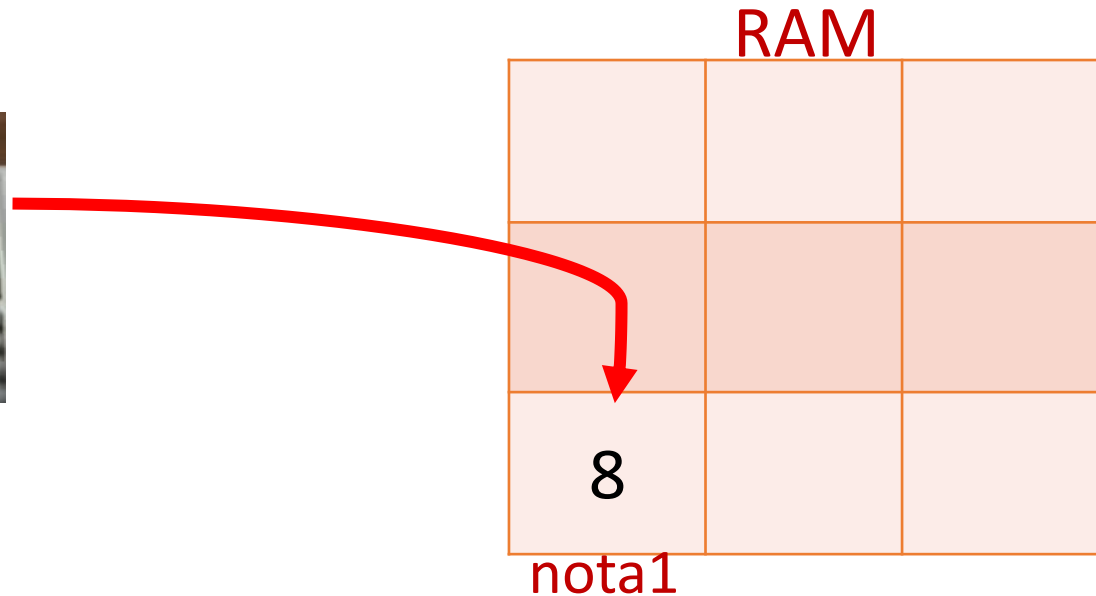
```
Ingrese nota correspondiente al primer parcial 8
Ingrese nota correspondiente al segundo parcial 9
El Promedio de Notas es: 8.50
-----
Process exited with return value 0
Press any key to continue . . .
```

# Ejemplo de lectura y escritura

- Lectura

```
scanf("%f",&nota1);
```

```
Ingresa nota correspondiente al primer parcial
```



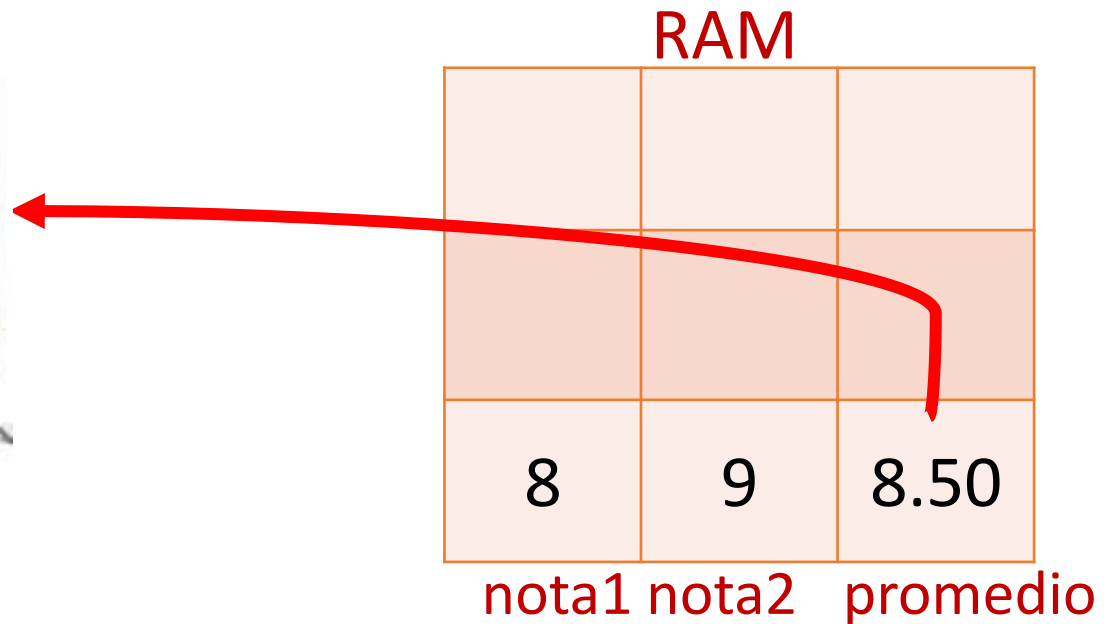
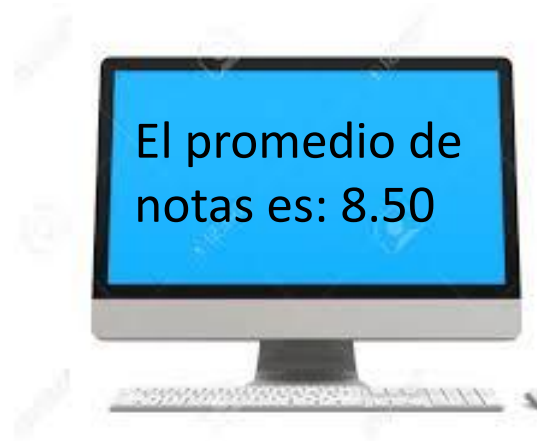
Una instrucción de lectura deposita valores en las **posiciones de memoria** indicadas por los nombres de **variables**.

# Ejemplo de lectura y escritura

## • Escritura

```
printf ("El Promedio de Notas es: %.2f",promedio);
```

```
El Promedio de Notas es:  8.50  
-----
```



Una instrucción de escritura toma los valores de las posiciones de memoria indicadas por los **nombres de variables** y visualiza o registra ese valor en algún dispositivo de salida (pantalla, impresora).

# Instrucciones de asignación

Una **instrucción de asignación** coloca un valor determinado en una **variable** o **posición de memoria**.

## Ejemplo 1:

**A = 80** (la variable denominada **A** toma el valor **80**)

Otras notaciones: **A ← 80** Pascal: **A:=80** C: **A = 80**

## Ejemplo 2:

Indicar el valor de A, B y AUX al ejecutarse la instrucción 5.

1. **A=10**
2. **B=20**
3. **AUX=A**
4. **A=B**
5. **B=AUX**

El resultado de la evaluación después de ejecutar la instrucción 5 será:

**A=20, B=10, Aux=10.**



# Instrucciones de asignación

Indicar el valor de A, B y AUX al ejecutarse la instrucción 5.

1. **A=10**

2. **B=20**

|    |    |     |
|----|----|-----|
|    |    |     |
| 10 | 20 |     |
| A  | B  | AUX |

3. **AUX=A**

|    |    |     |
|----|----|-----|
|    |    |     |
| 10 | 20 | 10  |
| A  | B  | AUX |

4. **A=B**

5. **B=AUX**

|    |    |     |
|----|----|-----|
|    |    |     |
| 20 | 10 | 10  |
| A  | B  | AUX |

Posiciones  
de  
memoria

# Instrucciones de asignación

## *Ejemplo 3:*

Cuál será el valor de **N** después de la ejecución de esta asignación?:

**N = N + 5**

Consideramos que **N** tiene un valor previo igual a **2**.

El resultado de N es **7**.

Al evaluar  $N + 5$ , N toma el valor de su contenido (que es 2), luego realiza la suma  $(2 + 5)$ , y el resultado de la expresión (7) se guarda en la variable N.

# Instrucciones de bifurcación

La alteración de la secuencia lineal de las instrucciones de un programa se realiza a través de las **instrucciones de bifurcación**.

Las bifurcaciones pueden ser:

a) *Condicionales*, si dependen del cumplimiento de una condición.

Ejemplo: **IF N > 0 THEN fin**

b) *Incondicionales*, si la bifurcación no depende de ninguna condición.

Ejemplo: **GOTO a la instrucción 5.**

Las bifurcaciones incondicionales (GOTO) no son una práctica recomendada en programación, no respetan los principios de la programación estructurada, y complican la comprensión de los programas.

# Verificación y depuración

(Simular las acciones de la computadora para comprobar los resultados)

## ALGORITMO Promedio\_Notas

### VAR

**ENTERO:** LU, N1, N2

**REAL:** promedio

### INICIO

**LEER** LU, N1, N2

**MIENTRAS** N1 <> 0

$\text{promedio} = (N1 + N2) / 2$

**ESCRIBIR** promedio

**LEER** LU, N1, N2

**FIN-MIENTRAS**

**FIN**

| ENTRADA |    |    |
|---------|----|----|
| LU      | N1 | N2 |
| 345     | 8  | 5  |
| 156     | 3  | 7  |
| 678     | 9  | 10 |
| 000     | 0  | 0  |

| SALIDA |          |
|--------|----------|
| LU     | Promedio |
| 345    | 6,50     |
| 156    | 5        |
| 678    | 9,50     |

| RAM – Lectura alumno 1 |    |    |
|------------------------|----|----|
| 345                    | 8  | 5  |
| LU                     | N1 | N2 |
| 6,50                   |    |    |
| Promedio               |    |    |

| RAM – Lectura alumno 2 |    |    |
|------------------------|----|----|
| 156                    | 3  | 7  |
| LU                     | N1 | N2 |
| 5                      |    |    |
| Promedio               |    |    |

| RAM – Lectura alumno 3 |    |    |
|------------------------|----|----|
| 678                    | 9  | 10 |
| LU                     | N1 | N2 |
| 9,50                   |    |    |
| Promedio               |    |    |

| RAM – Lectura alumno 4 |    |    |
|------------------------|----|----|
| 000                    | 0  | 0  |
| LU                     | N1 | N2 |
| 9,50                   |    |    |
| Promedio               |    |    |

# El mismo algoritmo codificado en C

```
#include <stdio.h>
/* Declaración de prototipos de funciones */
void ingresarDatos();
void calcularPromedio();
/* Declaración de variables globales */
int lu;
float nota1, nota2, promedio;

/* Programa principal*/
int main(){
    ingresarDatos();
    while(nota1 != 0) {
        calcularPromedio();
        ingresarDatos();
    }
    return 0;
}

/* Implementación de funciones */
void ingresarDatos() {
    printf("\nIngrese nro de LU: ");
    scanf("%d", &lu);
    printf("Ingrese nota 1: ");
    scanf("%f", &nota1);
    printf("Ingrese nota 2: ");
    scanf("%f", &nota2);
}

void calcularPromedio() {
    promedio = (nota1 + nota2)/2;
    printf("\nLU: %d\n", lu);
    printf("El promedio de notas es: %.2f\n", promedio);
}
```

Pueden copiar y  
pegar en el DevC++  
y realizar la  
verificación con los  
mismos casos de  
prueba

# Contadores y acumuladores

En el ejemplo anterior, se pueden visualizar dos **instrucciones de asignación** que se usarán con mucha frecuencia en la resolución de problemas. Estas son:

- **Contador:**

Es una variable que se incrementa **en un valor constante** y se utiliza para registrar el número de veces que se presenta un evento. Ejemplo: para contar los alumnos procesados, se incrementa en 1 por cada lectura de datos de alumnos.

**cantAlumnos** = 0

**cantAlumnos** = **cantAlumnos** + 1

- **Acumulador:**

Un acumulador es una variable, definida por el programador, que hace referencia a una dirección de memoria que almacenará un **"total móvil"** de valores individuales a medida que vayan apareciendo en el proceso.

Ejemplo, el importe total de las líneas de compra de un ticket de supermercado. Esta dirección de memoria debe ser inicializada en cero.

**totalCompra** = 0

importeProducto = cantidad \* precio

**totalCompra** = **totalCompra** + importeProducto

# Elementos básicos de un programa

- Lenguajes de programación= *reglas y sintaxis*.
- Los elementos básicos constitutivos de un programa o algoritmo son:
  - Identificadores (nombres de variables, funciones,..)
  - Palabras reservadas
  - Caracteres especiales
  - Constantes
  - Variables
  - Expresiones
  - Instrucciones



# Elementos básicos de un programa

**A-** Un **identificador** es una secuencia de caracteres, letras dígitos y subrayados (`_`), usados para identificar a las *variables, funciones, procedimientos*, que se utilizan en el programa.

En general, el primer carácter debe ser una letra.

Ejemplos: `edad`, `nota_examen`, `notaExamen`, `índice`, ...

El lenguaje C es sensible a las mayúsculas (***case sensitive***)

`Alfa` y `alfa` son distintos.

**Se recomienda escribir identificadores de variables en minúsculas, las constantes en mayúsculas y las funciones con minúscula/mayúscula.**



# Elementos básicos de un programa

**B-** Las **palabras reservadas** son identificadores predefinidos (tienen un significado especial) que no es posible utilizar para otros fines distintos para los que han sido definidas.

En todos los lenguajes de programación existe un conjunto de palabras reservadas. Por ejemplo, en C, algunas de ellas:

**void, int, char, case, const, float**

## **C- Comentarios**

En C los comentarios tienen el formato: `/* ... */` y pueden extender a varias líneas. Constituyen una valiosa documentación interna del programa.

Ejemplos:

**`/* Calcula el promedio de notas de los alumnos */`**

# Elementos básicos de un programa

## D-Signos de puntuación y separadores

Ejemplos:

%   &   \*   ( )   { }   /   :   [ ]   <   > ,

Los separadores son espacios en blanco, tabulaciones, retornos de carro y avance de línea. Estos 2 últimos son **códigos de control** dependientes de los sistemas operativos y las aplicaciones.

## E-Archivos de cabecera

Archivo especial que contiene las declaraciones de elementos y funciones de biblioteca. Para utilizar macros, constantes, tipos y funciones almacenadas en una biblioteca, se usa la directiva **#include** para insertar el archivo de cabecera correspondiente.

Por ejemplo, si un programa utiliza la función **pow** (potencia) que se encuentra en la biblioteca **math.h**, debe contener la directiva:

**#include <math.h>**

# Tipos de datos

- Un dato es la expresión general que describe los objetos con los que opera la computadora.
- *Ejemplos de datos:* el nombre de una persona, el valor de una temperatura, una cifra de venta de supermercado, la fecha de un cheque, etc.

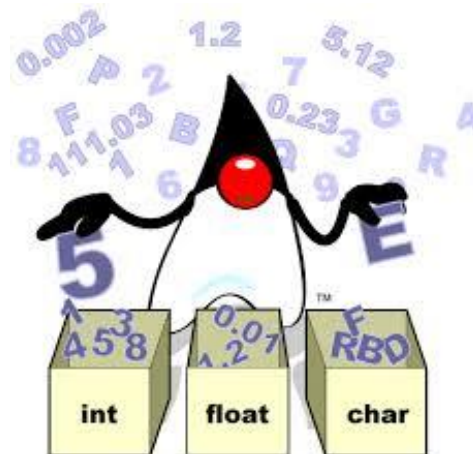


# Tipos de datos Representación

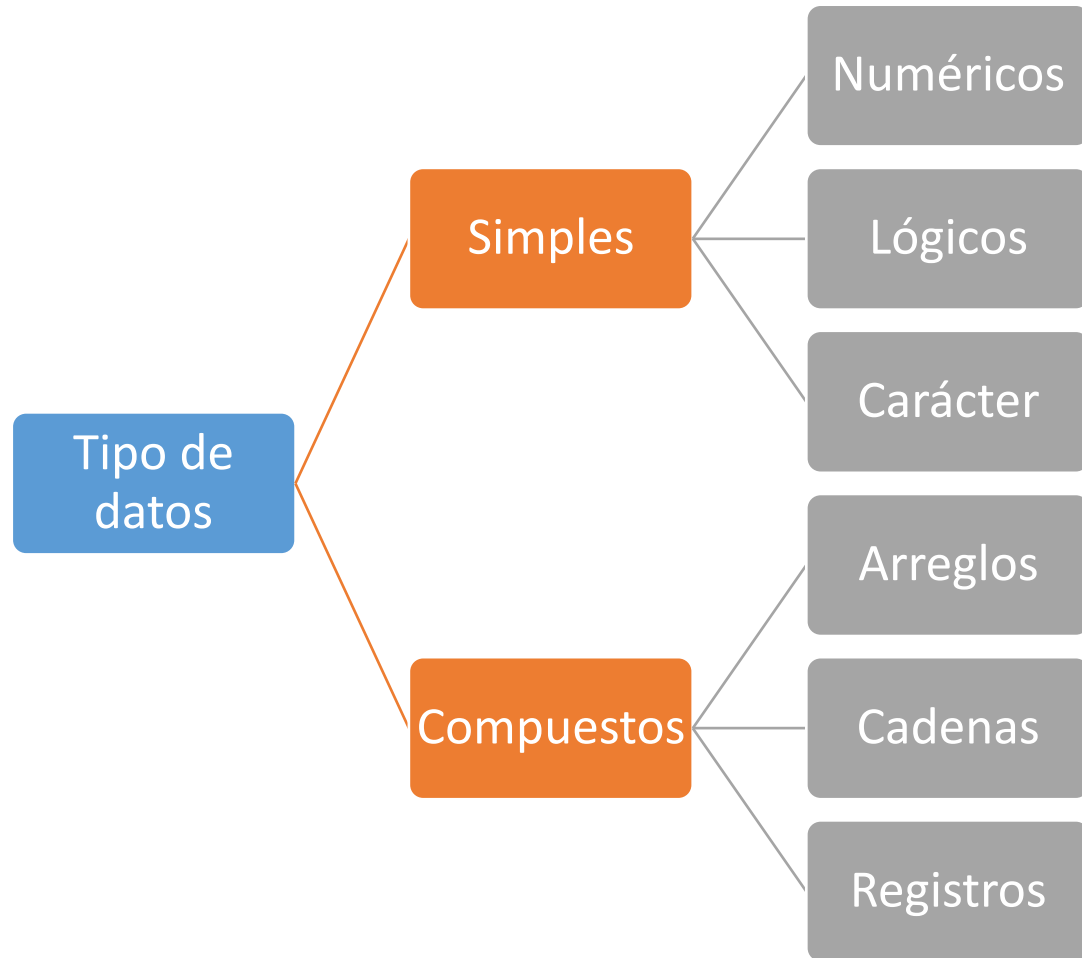
- Los distintos tipos de datos se representan en diferentes formas en la computadora.
- A nivel de máquina, un dato es un conjunto de bits (dígitos 0 o 1).
- El tipo de un dato asocia a un dato un determinado rango de valores. Si se intenta asignar un valor fuera del rango se producirá un error en la representación.
- La asignación de tipos a los datos tiene dos objetivos principales:
  - detectar errores en las operaciones
  - determinar cómo ejecutar estas operaciones

# Tipos de datos. Representación

- *Los lenguajes de programación fuertemente tipados exigen que todos los datos deben tener un tipo declarado explícitamente*
- *Pascal* es un lenguaje fuertemente tipado, algunas versiones de C, tales como C++ y C#, también. Esto significa que:
  - existen ciertas restricciones en las expresiones en cuanto a los tipos de datos que en ellas intervienen.
  - ventaja: se requiere menos esfuerzo en depurar (corregir) los programas gracias a los errores que detecta el compilador.



# Tipos de datos




# Tipos de datos Simple

- Es aquel cuyo contenido se trata como una unidad que no puede separarse en partes más elementales.
- Los más usuales son:
  - Numéricos (*integer, real*)
  - Lógicos (*boolean*)
  - Carácter (*char*)



Ejemplo de dato simple

Que puede ser de cualquiera de los siguientes tipos de datos



Entero

Real

Caracter

Bolaeno

# Tipos de datos Compuesto

- Permite almacenar un conjunto de elementos bajo una estructura particular, darle un único nombre, pero con la posibilidad de acceder en forma individual a cada componente.
- Ejemplos: [Arreglos](#), [cadenas](#), [registros](#)

|         | col(0) | col(1) | col(2) | col(3) | col(4) |
|---------|--------|--------|--------|--------|--------|
| fila(0) | 4.66   | 56.7   | 3.24   | 33.2   | 55.4   |
| fila(1) | 12.3   | 5.4    | 33.1   | 4.0    | 99.0   |
| fila(2) | 5.2    | 4.3    | 5.88   | 33.2   | 26.1   |
| fila(3) | 12.3   | 66.5   | 2.2    | 874.5  | 5.32   |
| fila(4) | 4.55   | 76.3   | 4.3    | 33.2   | 8.7    |

Matriz(2,3) = 33.2



# Datos Numéricos

- Pueden representarse en dos formas distintas:

- ***Entero***

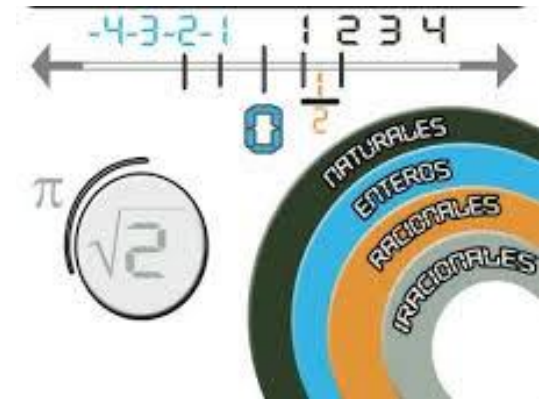
- ***Real***

# Enteros

- Pueden ser positivos o negativos y no tienen decimales.
- Ejemplos: 5, 15, -30, 12567.
- La mayoría de los lenguajes de programación utilizan enteros que se almacenan en **2 bytes (entero corto)**, de ahí que los valores mínimo y máximo son, respectivamente, **-32768 y 32767**.



# Reales



- Pueden ser positivos o negativos y tienen punto decimal.
- Un número real consta de una parte entera y de una parte decimal. Ejemplos: -45,78, 3,0, 0,008, -13456,89.
- Una computadora sólo puede representar un número fijo de dígitos que depende del tamaño de la **palabra**.

Puede haber problemas para representar y almacenar números muy grandes o muy pequeños, de aquí surge la **coma flotante** para disminuir estas dificultades.

# Reales

- En este tipo de datos se puede utilizar la **notación científica**, para números grandes o pequeños:

$$3.0E5 = 3.0 * 10^5 = 3.0 * 100000 = 300000$$

$$1.5E-4 = 1.5 * 10^{-4} = 1.5 * 0.0001 = 0.00015$$

- Los tipos de datos reales, empiezan a perder precisión cuando se sobrepasa el valor de sus bits de mantisa.
- Ejemplo, en 24 bits solo se puede almacenar un valor aproximado de 16.7 millones.
- Cuando se pasa de esos 16.7 millones la variable que contiene el número empieza a perder precisión y almacena **un dato aproximado**.

# Tipos de datos básicos en C

**int** : Enteros (números enteros positivos y negativos)

**char** : Caracteres (letras)

**float** : Números en coma flotante (números reales)

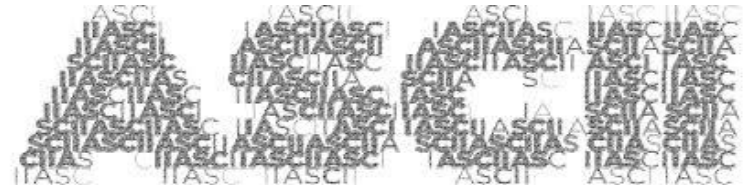
**double** : Números en coma flotante de doble precisión

# Datos Lógicos (booleanos)

- El tipo *lógico*, también denominado *booleano*, sólo puede tomar 2 valores: **verdadero** (*true*) o **falso** (*false*).
- La asignación **prueba = true**, asigna el valor lógico verdadero a una variable de tipo lógico.

Ejemplos: **FinDatos=Verdadero; Error=Falso**

# Datos Char



- El tipo carácter (*char*) contiene un solo símbolo.
- En general, estos caracteres se almacenan en ASCII y el orden de los caracteres es el valor numérico que le asigna este código.
- Un valor constante de tipo carácter se escribe entre apóstrofos.
- Ejemplo:

`letra = 'X'`

asigna a la variable de tipo carácter *letra* la constante carácter X.

# Datos String

- Una cadena (*string*) de caracteres representa un conjunto de caracteres.
- **No es un dato de tipo simple** dado que está integrado por elementos a los cuales se puede acceder en forma individual. Se trata de un tipo de datos **estructurado**.
- Una cadena de caracteres tiene dos características importantes: la **longitud física** y la **longitud dinámica** o **lógica**.
- La longitud física se define en la declaración del tipo de dato y permite al procesador reservar el **espacio máximo de memoria** necesario para almacenar el valor de una variable de ese tipo.

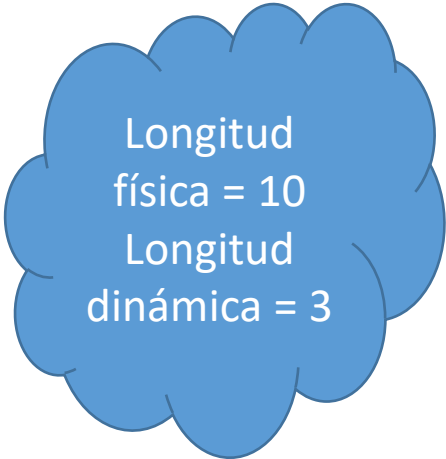


# Datos String

- Ejemplo:

- `STRING nombre[10]`
- `nombre = 'Ema'`

- Las constantes de tipo string se escriben también entre apóstrofes.



Longitud  
física = 10  
Longitud  
dinámica = 3

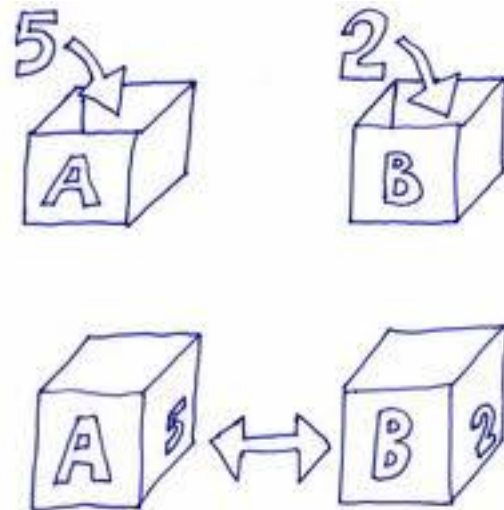
# Constantes

- Son valores que no cambian durante la ejecución del programa.
- Ejemplos de constantes:
  - Constante de tipo real: 3.141592; -0.234
  - Constante de tipo carácter: 'B', '4'
  - Constante de tipo cadena: '9 de julio 1449', 'Ciencias Exactas', 'Juan Pérez'
  - Constantes lógicas: true, false

Tener en cuenta que C adopta la notación decimal inglesa, o sea, un punto decimal en lugar de una coma decimal.

# Variables

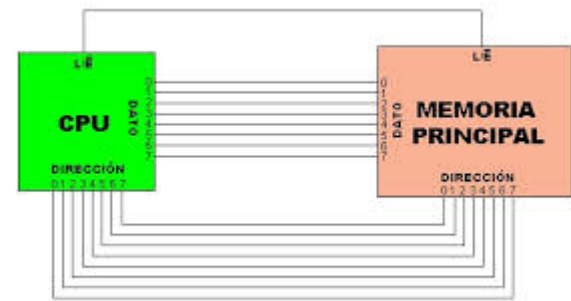
- Las variables de un programa son **espacios de la memoria principal** que se **identifican con un nombre** y son de un **tipo de dato**.
- Importante: el “**tipo**” de variables condiciona las operaciones que pueden realizarse con ella!



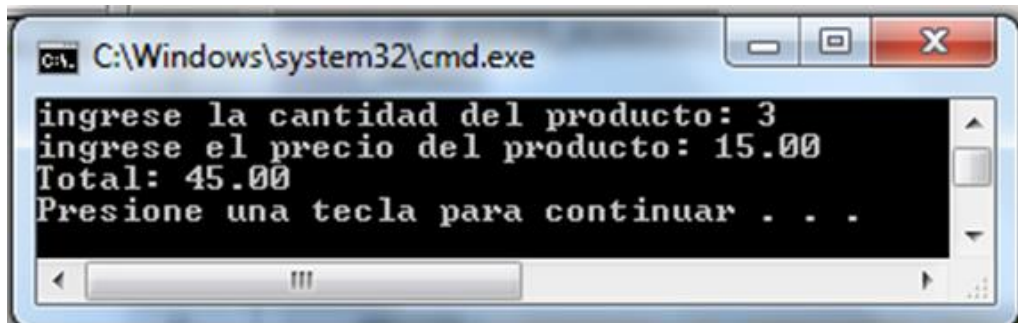
# Variables como posiciones de memoria

## Programa fuente escrito en C

```
#include <stdio.h>
int main() {
    int cantidad;
    float precio, total;
    char respuesta;
    printf("ingrese la cantidad del producto: \n");
    scanf("%d", &cantidad);
    printf("Ingrese el precio del producto: \n");
    scanf("%f", &precio);
    total = cantidad * precio;
    printf("Total: %2.2f \n", total);
    return 0;
}
```



## Ejecución del programa:

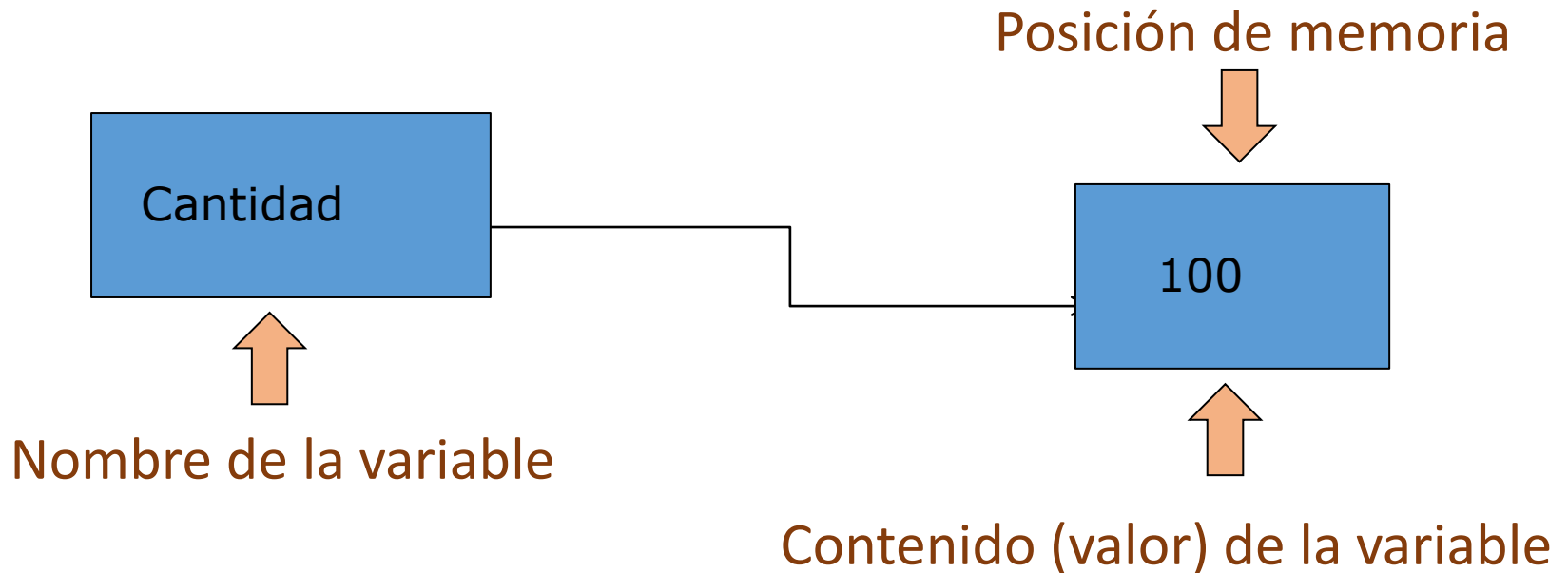


|          |        |  |  |  |
|----------|--------|--|--|--|
| cantidad |        |  |  |  |
|          | precio |  |  |  |
|          |        |  |  |  |
|          | total  |  |  |  |

Tener en cuenta que los valores de las variables del programa se almacenan en la memoria en una determinada cantidad de bits, de las cuales depende el valor y precisión a representar.

# Nombre de Variable

- El **nombre** de una variable **identifica** el espacio de memoria reservado para la misma y el valor que allí se almacena.



# Nombre de Variable

- Las **restricciones** del nombre dependen del lenguaje de programación. En general, las buenas prácticas indican:
- Debe ser *mnemotécnico* (descriptivo del contenido). Ejemplo: `nombreAlumno, edad, domicilio, ...`
- No utilizar palabras reservadas ni caracteres especiales (`espacio, letras acentuadas, ñ, coma, punto`).
- Limitarse a letras, dígitos y guión bajo (`_`) para separar palabras. Ej. `Importe_netto`.
- Algunos lenguajes distinguen mayúsculas de minúsculas (por ejemplo, **NOTA** no es lo mismo que **Nota** o **nota**).

# Tipo de variables

- El **tipo de dato** se especifica cuando se declaran las variables.

Ejemplo:

**VAR**

**REAL:** precio\_unitario, precio\_final

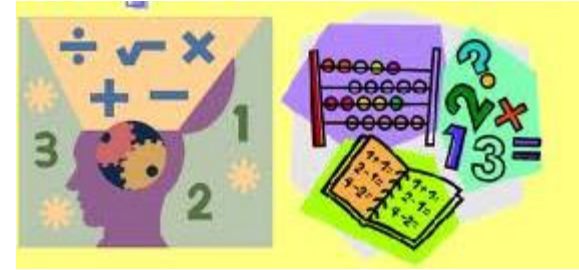
**ENTERO:** cantidad

**STRING:** domicilio[30]

**CHAR:** codigo

- En el ejemplo se declaran tres variables numéricas: dos de tipo real y una de tipo entero, una variable de tipo carácter y otra de tipo cadena.
- No todos los lenguajes de programación requieren que se declaren las variables a utilizar, pero es una buena práctica hacerlo.

# EXPRESIONES



- Las expresiones son **combinaciones de constantes, variables, operadores, paréntesis y nombres de funciones**. Maneja las mismas ideas que la notación matemática convencional.
- Por ejemplo:  **$a + (b+3) + \sqrt{C}$**
- Una expresión consta de **operandos** y **operadores**. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en:
  - **Aritméticas** (*Resultado de tipo Numérico*)
  - **Lógicas** (*Resultado de tipo Lógico*)
  - **Carácter** (*Resultado de tipo Carácter*)
  - **Cadena de caracteres** (*resultado de tipo Cadena*)



# Expresiones aritméticas

- Son análogas a las formulas matemáticas. Las variables y constantes son numéricas (entera o real) y las operaciones son las aritméticas.
- **Operadores aritméticos** (los símbolos pueden diferir en los distintos lenguajes de programación)

| Símbolo          | Operación                       |
|------------------|---------------------------------|
| +                | Suma                            |
| -                | Resta                           |
| *                | Multiplicación                  |
| /                | División                        |
| <b>**</b> , ^, ↑ | Exponenciación                  |
| div              | división de enteros             |
| mod              | resto de la división de enteros |



# Operadores div y mod

- **div**: división de enteros.
- **Mod**: resto de la división entera.

$$\begin{array}{r|l} 79 & 8 \\ -72 & \\ \hline 7 & 9 \text{ div} \\ \text{mod} & \end{array}$$

si  $A=79$  y  $B=8$

$A \text{ div } B$  ?


$A \text{ mod } B$  ?


# Reglas de prioridad


- El **orden** de las operaciones, se denominan **reglas de prioridad o precedencia** y son:
- Las operaciones encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
- Prioridad **operaciones aritméticas**:
  - 1° - Operador de exponenciación ( $**$ ,  $^$ ,  $\uparrow$ )
  - 2° - Operadores  $*$ ,  $/$ ,  $\text{div}$  y  $\text{mod}$
  - 3° - Operadores  $+$ ,  $-$ ,

# Reglas de prioridad

- Cuando hay varios operadores con igual prioridad, el orden de prioridad es de **izquierda a derecha**.
- Ejemplo:

$$3 + 6 * 14 / 2$$


$$3 + 84 / 2$$


$$3 + 42$$


$$45$$

# Expresiones booleanas

- La expresión *lógica* o *booleana* devuelve siempre verdadero o falso.
- Opera las constantes y variables lógicas que toman solamente los valores *verdadero (true)* o *falso (false)*.
- Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas utilizando los *operadores lógicos not, and y or* y los *operadores relacionales* (de relación o comparación).

# Operadores de relación

- Permiten realizar comparaciones de valores de tipo numérico o carácter.

| Operador | Significado          | Operadores en C |
|----------|----------------------|-----------------|
| =        | igual                | ==              |
| >        | mayor que            | >               |
| <        | menor que            | <               |
| ≥        | mayor o igual<br>que | >=              |
| ≤        | menor o igual<br>que | <=              |
| <>       | distinto de          | !=              |

# Operadores de relación

Ejemplo:

Dadas las variables **A** y **B**, con los valores **A=4** y **B=3**:

**A** > **B** es *verdadera*

$$4 > 3$$

**(A-2)** < **(B-4)** es *falsa*

$$4-2 < 3-4 \rightarrow 2 < -1$$

# Operadores lógicos

- Los operadores lógicos o booleanos básicos son **not** (no), **and** (y) y **or** (o).

| Operador lógico | Expresión lógica       | Significado         |
|-----------------|------------------------|---------------------|
| <b>NO</b> (not) | <b>NO</b> p (not p)    | Negación de p       |
| <b>Y</b> (and)  | p <b>Y</b> q (p and q) | Conjunción de p y q |
| <b>O</b> (or)   | p <b>O</b> q (p or q)  | Disyunción de p y q |



# Tablas de verdad

- Las definiciones de las operaciones **no**, **y**, **o** se resumen en las llamadas *tablas de verdad*.

| a         | no a      |
|-----------|-----------|
| verdadero | falso     |
| falso     | verdadero |

| a         | B         | a y b     |
|-----------|-----------|-----------|
| verdadero | verdadero | Verdadero |
| verdadero | Falso     | Falso     |
| falso     | verdadero | falso     |
| falso     | Falso     | falso     |


| a         | b         | a o b     |
|-----------|-----------|-----------|
| verdadero | verdadero | verdadero |
| verdadero | falso     | verdadero |
| Falso     | verdadero | verdadero |
| Falso     | falso     | falso     |

## Expresiones lógicas:

$(11 < 20) \text{ y } (3 < 9)$  la evaluación de la expresión es **verdadera**

$(15 > 30) \text{ o } ('X' < 'Z')$  la evaluación de la expresión es **verdadera**

# Prioridad de los operadores lógicos

| Operador                                                   | Prioridad                                                                                                                |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>no</b> (not)<br><br><b>y</b> (and)<br><br><b>o</b> (or) | Más alta (primera ejecutada)<br><br> |

# Características de C vinculadas con el tratamiento de los datos

**Operadores especiales:** C admite algunos operadores especiales que sirven para propósitos diferentes:

- El operador ( ): Operador de llamada a funciones, encierra los argumentos de una función.
- El operador [ ]: Sirve para dimensionar arreglos y designar un elemento de arreglo.

`int notas [20]` /\* define un arreglo de 20 elementos.

- El operador **sizeof**: Permite conocer el tamaño en byte de un tipo de dato o variable.

`sizeof (unsigned int)`

devuelve la cantidad de bytes del tipo de dato ingresado (**depende de la arquitectura de la computadora**).

# Conversión de tipos

Con frecuencia se necesita convertir un tipo de dato a otro, sin cambiar el valor que representa.

## C1) Conversión implícita:

Los operandos de tipo más bajo se convierten en los de tipo más alto. Ejemplos:

Ejemplos:

```
int k = 12;
```

```
float m = 4;
```

```
m = m + k;
```

/\* el valor de **k** se convierte temporalmente en *float* antes de la suma, **m = 16** \*/

```
m = k / 5;
```

/\* realiza la división entera **k/5**, el resultado es **m = 2.0** \*/

```
m = 4;
```

```
m = m / 5; /* Convierte 5 a float, realiza la división y el resultado es m = 0,8 */
```

# Conversión de tipos

## C2) Conversión explícita:

C fuerza la conversión de tipo mediante el operador *cast*, cuyo formato es:

**(tipo dato) variable**

Ejemplos:

**(int) m**            /\* Cambia el tipo de dato de **m** a entero \*/

**(float) k**           /\* Cambia el tipo de dato de **k** en float \*/

**m = (float)k/5;**

/\* realiza la división **k / 5**, convierte el resultado a float y asigna a **m** \*/

Si **k** es 12 el valor que se almacena en **m** es 2.4.

# Ejemplo de expresiones lógicas:

**Datos:** nombre, edad, sexo (V/M), Trabaja (True, False)

**Salida requerida:** Alumnos varones menores de 22 que trabajan o que sean mayores de 25

**Expresión:**

(Sexo = "V" Y ((Edad < 22 Y trabaja = True) o (Edad > 25 ))

**Datos reales:**

José, 27, V, True

María, 22, M, False

Martín, 18, V, True

Juan, 22, V, True

Ana, 23, M, True

Carlos, 25, V, True

¿Cuántos varones de esta lista cumplen la condición requerida?

# Ejemplo de expresiones lógicas:

## Expresión:

(Sexo = "V" Y ((Edad < 22 Y trabaja = True) o (Edad > 25 ))

## Resultados

|                   |                  |
|-------------------|------------------|
| José, 27, V, True | <i>verdadero</i> |
| María, 22, M, NO  | <i>falso</i>     |
| Martín, 18,V,True | <i>verdadero</i> |
| Juan, 22,V, True  | <i>falso</i>     |
| Ana, 23, M, True  | <i>falso</i>     |
| Carlos, 25,V,True | <i>falso</i>     |

Dos varones de esta lista cumplen la condición requerida.

Modifique la expresión para que seleccione los varones menores de 22 años que trabajan o las personas menores de 25 años. ¿Cuántas personas de la lista serían seleccionadas?



**Fin de la clase!!!**