

## INTRODUCCIÓN A LA PROGRAMACIÓN

### INTRODUCCIÓN AL SOFTWARE

#### SOFTWARE

Página | 1

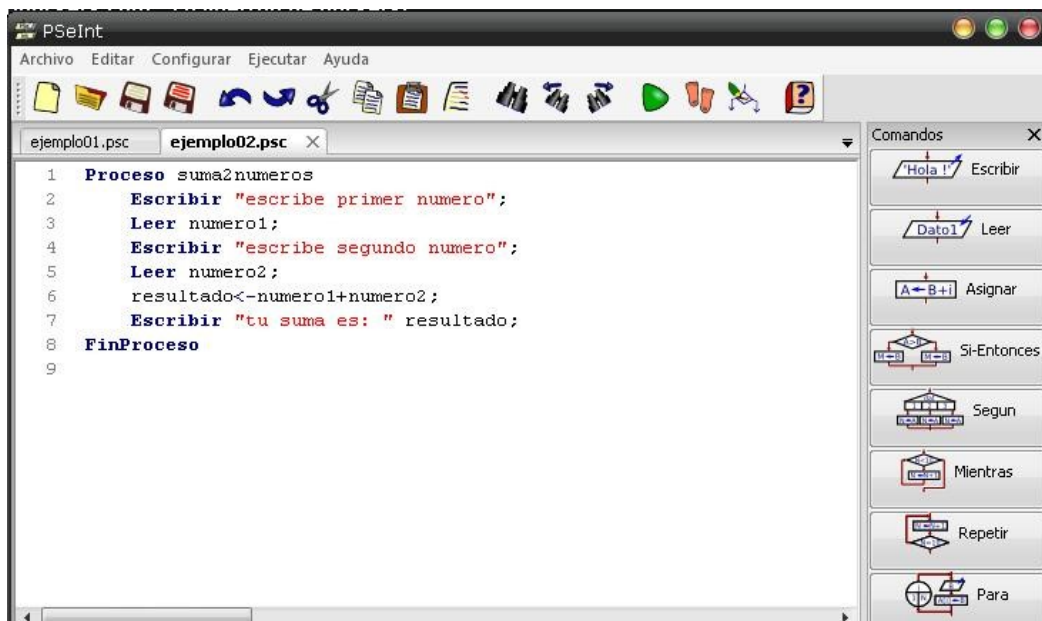
Nos referimos con software a la parte lógica de la computadora, a los procedimientos que el hardware realiza inducidos por el software y este a su vez por nosotros. El software es como un traductor que hace que nuestras órdenes se conviertan en realidad, manipulando el hardware o la parte física.

El software está compuesto por programas y aplicaciones de computadora.

#### CÓDIGO FUENTE

Le daremos el nombre de código fuente a los programas que escribamos en un determinado lenguaje de programación, que simplemente estará compuesto por instrucciones escritas por un programador.

El código fuente no constituye software propiamente dicho, pero es una instancia mediante la cual se llega al Software.



## SISTEMA OPERATIVO

Es el programa más importante que se ejecuta en una computadora. Cualquier computadora de propósito general debe operar con un sistema operativo para lograr ejecutar otros programas.

El sistema operativo ejecuta las tareas básicas, como de reconocer entradas desde el teclado, enviar mensajes a pantalla, manteniendo rastro de los archivos y directorios en el disco, además de controlar los dispositivos periféricos como las impresoras.

Para grandes sistemas, el sistema operativo tiene una gran responsabilidad y cualidades. Es como un policía de tránsito, quien se asegura de que los diferentes programas que se ejecutan al mismo tiempo no interfieran unos con otros. También es responsable de la seguridad, protegiendo para que usuarios no autorizados accedan al sistema.

El sistema operativo provee de una plataforma de software por encima de la cual otros programas, llamados aplicaciones, pueden ejecutarse.

Los programas de aplicación tienen que crearse de acuerdo a la plataforma en donde se van a ejecutar. La elección de sistema operativo, entonces, determina el tipo de uso que se le va a dar a la PC como también el tipo de aplicaciones que se puedan ejecutar.

Ejemplos de sistemas operativos más populares son: Windows, Linux, Android, Macintosh OS, Unix.

Página | 2

## LENGUAJES DE ALTO NIVEL Y LENGUAJES DE BAJO NIVEL

Los programadores escriben instrucciones en diversos lenguajes de programación. La computadora puede entender directamente algunos de ellos, pero otros requieren pasos de traducción intermedios.

Hoy día se utilizan cientos de lenguajes de computadora, los cuales pueden dividirse en tres tipos generales:

### 1. LENGUAJE MÁQUINA

Es un lenguaje escrito en formato binario (formado por 1 y 0), propio de cada computadora y relacionado con el diseño del hardware de la misma. El lenguaje de máquina ordena a la computadora realizar sus operaciones fundamentales una por una. Una computadora sólo puede entender el lenguaje máquina.

A+B -> 10100101010010

### 2. LENGUAJES DE BAJO NIVEL (ENSAMBLADOR)

Consiste en abreviaturas similares al inglés, llamadas instrucciones mnemotécnicas, que permiten representar las operaciones elementales de la computadora (también es dependiente de la computadora).

```
MOV AX, @data          ; carga en AX la dirección del segmento de datos
MOV DS, AX              ; mueve la dirección al registro de segmento por medio de AX
MOV DX, offset Cadena1  ; mueve a DX la dirección del string a imprimir
MOV AH, 9               ; AH = código de la función del MS DOS para imprimir en pantalla
```



INT 21h ; llamada al MS DOS para imprimir un string en la pantalla INT 20h  
; llamada al MS DOS para finalizar el programa

---

### 3. LENGUAJES DE ALTO NIVEL

Los lenguajes de alto nivel permiten a los programadores escribir instrucciones que asemejan el inglés cotidiano y contiene notaciones matemáticas de uso común.

Página | 3

Cualquier programa escrito debe poder ejecutarse en cualquier computadora después de compilarlo. A esto se le conoce como portabilidad de programas.

Existen lenguajes de programación de Alto y Bajo nivel; entre los más conocidos de Alto nivel podemos mencionar a .Net, C Sharp, PHP, ASP, Visual Basic, Java, C, C++, JAVA, Fortran, T. Pascal, etc.



## TRADUCTOR DE LENGUAJES DE PROGRAMACIÓN

Los traductores son programas que traducen los programas en código fuente, escritos en lenguajes de alto nivel, a programas escritos en lenguaje máquina. Los traductores pueden ser de dos tipos: compiladores e intérpretes.

Página | 4

### COMPILADOR

Un compilador es un programa que lee el código escrito en un lenguaje (lenguaje origen), y lo traduce en un programa equivalente escrito en otro lenguaje (lenguaje objetivo).

Como una parte fundamental de este proceso de traducción, el compilador le hace notar al usuario la presencia de errores en el código fuente del programa. Vea la figura de abajo.



Por ejemplo, el C++ es un lenguaje que utiliza un compilador y su trabajo es el de llevar el código fuente escrito en C++ a un programa escrito en lenguaje máquina.

El compilador al realizar su tarea hace una comprobación de errores en el programa, es decir, revisa que todo esté en orden. Por ejemplo, que las variables y las funciones estén bien definidas, respetando todas las reglas sintácticas del lenguaje. Esta fuera del alcance del compilador que, por ejemplo, el algoritmo utilizado en el problema funcione bien.

La siguiente figura muestra los pasos para tener un programa ejecutable desde el código fuente:



### INTERPRETE

Los intérpretes en lugar de producir un Lenguaje objetivo, como lo hacen los compiladores, realizan la operación que debería realizar el lenguaje origen. Un intérprete lee el código como está escrito y luego lo convierte en acciones, es decir, lo ejecuta en ese instante.

Existen lenguajes que utilizan un Intérprete, como por ejemplo: JAVA, HTML, Javascript entre otros. El intérprete traduce en el instante mismo de lectura el código en lenguaje máquina para que pueda ser ejecutado.

La siguiente figura muestra el funcionamiento de un intérprete.



## DIFERENCIA ENTRE COMPILADOR E INTÉRPRETE

Los compiladores difieren de los intérpretes en varios aspectos:

Un programa que ha sido compilado puede correr por sí solo, dado que en el proceso de compilación lo transformó en lenguaje máquina.

Un intérprete traduce el programa cuando lo lee, convirtiendo el código del programa directamente en acciones.

La ventaja del intérprete es que cualquier programa puede ser ejecutado en cualquier plataforma (sistema operativo), en cambio el archivo generado por el compilador solo funciona en la plataforma en donde se lo ha creado.

Pero, por otro lado, un archivo compilado puede ser distribuido fácilmente conociendo la plataforma, mientras que un archivo interpretado no funciona si no se tiene el intérprete instalado.

Hablando de la velocidad de ejecución, un archivo compilado es de 10 a 20 veces más rápido que un archivo interpretado.

## ALGORITMOS

Un algoritmo es un procedimiento a seguir, para resolver un problema en términos de:

### 1. Las acciones por ejecutar

### 2. El orden en que dichas acciones deben ejecutarse

Página | 6

Un algoritmo nace en respuesta a la aparición de un determinado problema. Un algoritmo está compuesto de una serie finita de pasos que convergen en la solución de un problema, pero además estos pasos tienen un orden específico.

Entenderemos como problema a cualquier acción o evento que necesite cierto grado de análisis, desde la simpleza de cepillarse los dientes hasta la complejidad del ensamblado de un automóvil.

En general, cualquier problema puede ser solucionado utilizando un algoritmo. En este sentido podemos utilizar los algoritmos para resolver problemas de cómputo.

Un algoritmo para un programador es una herramienta que le permite resaltar los aspectos más importantes de una situación y descartar los menos relevantes. Todo problema de cómputo se puede resolver ejecutando una serie de acciones en un orden específico.

Por ejemplo, considere el algoritmo que se elaboraría para el problema o situación de levantarse todas las mañanas para ir al trabajo:

1. Salir de la cama
2. quitarse el pijama
3. ducharse
4. vestirse
5. desayunar
6. arrancar el automóvil para ir al trabajo o tomar transporte.

Nótese que en el algoritmo anterior se ha llegado a la solución del problema en 6 pasos, y no se resaltan aspectos como: colocarse los zapatos después de salir de la cama, o abrir la llave de la ducha para bañarse.

Estos aspectos que han sido descartados no tienen mayor trascendencia, en otras palabras, los estamos suponiendo.

En cambio, existen aspectos que no podemos obviarlos o suponerlos, de lo contrario nuestro algoritmo perdería lógica. Un buen programador deberá reconocer esos aspectos importantes y tratar de simplificar al mínimo su problema.

Es importante recalcar que los pasos de un algoritmo no son conmutativos, porque no darían solución al mismo problema a tratar.

### Otro ejemplo: Reemplazar una rueda del auto pinchada

Inicio

    Si tengo el gato hidráulico en el auto  
        Sacar el gato



```
        Poner el gato
        Aflojar y sacar los tornillos de la rueda
        Levantar el coche con el gato
        Quitar la cubierta
        Buscar la cubierta de repuesto
        Poner la cubierta de repuesto
        Poner los tornillos y apretarlos
        Bajar el gato
        Guardar el gato
    SINO
        Llamar al servicio mecánico
    FIN SI
Fin
```

---

## ROBUSTEZ DE UN ALGORITMO

Quiere decir que un algoritmo debe contemplar todas las posibles facetas del problema que queremos resolver, al elaborar un algoritmo no se nos debe escapar ningún detalle que provoque un funcionamiento malo en nuestro algoritmo.

Si logramos construir un algoritmo robusto, cualquier giro inesperado del problema será controlado por el algoritmo, es decir, debe ser flexible a cambios.

---

## CORRECTITUD DE UN ALGORITMO

Es correcto cuando da una solución al problema a tratar y cumple con todos los requerimientos especificados tal que cumplamos con los objetivos planteados.

---

## COMPLETITUD DE UN ALGORITMO

Cuando un algoritmo cuenta con todos los recursos para poder llegar a una solución satisfactoria.

---

## EFICIENCIA Y EFICACIA DE UN ALGORITMO

Un algoritmo es **eficiente** cuando logra llegar a sus objetivos planteados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria, de pasos y de esfuerzo humano.

Un algoritmo es **eficaz** cuando alcanza el objetivo primordial, el análisis de resolución del problema se lo realiza prioritariamente.

Puede darse el caso de que exista un algoritmo eficaz pero no eficiente, en lo posible debemos de manejar estos dos conceptos conjuntamente.

---

## RESOLUCIÓN DE PROBLEMAS

Para lograr resolver cualquier problema se deben seguir básicamente los siguientes pasos:

**Análisis del problema:** en este paso se define el problema, se lo comprende y se lo analiza con todo detalle.

**Diseño del algoritmo:** se debe elaborar un algoritmo que refleje paso a paso la resolución del problema.

**Resolución del algoritmo en la computadora:** se debe codificar el algoritmo.



## PSEUDOCÓDIGO

Pseudocódigo **es un lenguaje artificial e informal** que ayuda a los programadores a desarrollar algoritmos.

El Pseudocódigo es similar al lenguaje cotidiano; es cómodo y amable con el usuario, aunque no es realmente un verdadero lenguaje de computadora. No se ejecutan en las computadoras más bien sirven para ayudar al programador a razonar un programa antes de intentarlo en algún lenguaje.

Por ejemplo, supongamos que la nota para aprobar un examen es de 60. El enunciado en Pseudocódigo sería:

```
Si calificación >= 60 entonces
  Mostrar "Aprobado"
FinSi
```

El mismo enunciado se puede escribir en PHP como:

```
if ( calif >= 60 )
  echo(Aprobado);
```

Nótese que la operación de trasladar el Pseudocódigo a código fuente, se lo realiza con el mínimo esfuerzo, no se necesita de un mayor análisis.



## DIAGRAMAS DE FLUJO

Un diagrama de flujo **es una representación gráfica de un algoritmo o de una parte de este**. Los diagramas de flujo ayudan en la comprensión de la operación de las estructuras de control (Si, Mientras).

La ventaja de utilizar un algoritmo es que se lo puede construir independiente de un lenguaje de programación, pues al momento de llevarlo a código se lo puede hacer en cualquier lenguaje.

Página | 9

Dichos diagramas se construyen utilizando ciertos símbolos de uso especial como son rectángulos, diamantes, óvalos, y pequeños círculos, estos símbolos están conectados entre sí por flechas, conocidas como *líneas de flujo*. A continuación, se detallarán algunos de estos símbolos.



**Terminal.** Representa el inicio y fin de un programa.



**Proceso.** Son acciones que el programa tiene que realizar



**Decisión.** Indica operaciones lógicas o de comparación, así como expresiones



**Entrada / Salida.** Nos permite ingresar datos, de un periférico, así como mostrarlos



**Salida.** Es usado para mostrar datos o resultados



**Conector.** Se coloca al principio y fin de un pedazo de programa, enlaza dos partes cualesquiera de un programa



**Línea de flujo** o indicador de dirección.

## VARIABLES

Un programa necesita un medio de grabar los datos que usa. Las variables y Constantes ofrecen varias maneras para representar y manipular los datos.

Página | 10


### DEFINICIÓN DE VARIABLE

Una variable **es un espacio para guardar información**. Entrando más a detalle una variable **es una ubicación** en la memoria de la computadora en la cual se puede grabar un valor y por la cual se puede recuperar ese valor más tarde.

La memoria RAM de la computadora puede ser vista como una serie de pequeñas casillas, cada una de las casillas esta numerada secuencialmente, este número que se le asigna representa su dirección de memoria y su objetivo es identificarla.

Una variable reserva una o más casillas en las cuales es posible grabar datos.

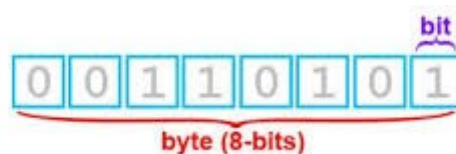
Los nombres de las variables (por ejemplo, Edad) es una etiqueta en una sola casilla, para que se pueda encontrarla fácilmente sin saber su actual dirección de memoria.

 RAM es la memoria de acceso aleatorio. Un programa cuando es ejecutado o esta corriendo, es grabado temporalmente en la memoria RAM. Todas las variables, son también, creadas en la memoria RAM. Cuando los programadores hablan de memoria, generalmente se están refiriendo a la memoria RAM.

### RESERVANDO MEMORIA

Se reserva memoria en el momento de definición de las variables, en este momento es donde se debe de especificar al compilador que clase de variable es: un entero (int), un caracter (char), etc. Esta información le dice al compilador cuanto de espacio debe separar o reservar, y que tipo de valor se va a guardar en la variable.

Cada casilla de memoria tiene un byte de capacidad. Si el tipo de variable que se crea es de dos bytes de tamaño, este necesita de dos bytes de memoria, o de dos casillas. El tipo de variable (por ejemplo, entero) le dice al compilador cuanta memoria (o cuantas casillas) tiene que reservar para la variable.



## TIPOS DE DATOS BÁSICOS

### Entero

Subconjunto finito de los números enteros, cuyo rango dependerá del lenguaje en el que posteriormente codifiquemos el algoritmo y del ordenador. El rango depende de cuantos bits utilice para codificar el número, normalmente **2 bytes**, Para números **positivos**, con 16 bits se pueden almacenar  $2^{16} = 65536$  números enteros diferentes: de 0 al 65535, y de -32768 al 32767 para números con signo.

Por ejemplo 2, 14, -199,....

Operaciones asociadas al tipo entero:

Como norma general las operaciones asociadas a un tipo cualquiera serán aquellas cuyo **resultado** sea un **elemento del mismo tipo**, por tanto:

+, -, \*, / (división), % (modulo)

### Real

Subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión. Suelen ocupar 4, 6 ó 10 bytes. Se representan por medio de la **mantisa**, y un **exponente** ( $1E-3 = 0.001$ ), utilizando **24 bits** para la mantisa (1 para el signo y 23 para el valor) y **8 bits** para el exponente (1 para el signo y 7 para el valor). El orden es de  $10^{-39}$  hasta  $10^{38}$ .

Por ejemplo 6.9, 33.00123.....

Las operaciones asociadas serían +, -, \*, /, %, etc.

### Lógico

Conjunto formado por los valores Cierto y Falso. '1' y '0'. Operaciones: todas las lógicas y relacionales AND, OR, >, <, == (igual), >=, <=, ....

### Carácter

Conjunto finito y ordenado de los caracteres que el ordenador reconoce. Se almacenan en un **byte**. Con 8 bits se podrán almacenar  $2^8 = 256$  valores diferentes (normalmente entre 0 y 255; con ciertos compiladores entre -128 y 127).

Un carácter (*letra*) se guarda en un solo byte como un número entero, el correspondiente en el **código ASCII**, que se muestra en la Tabla para los caracteres estándar, existe un código ASCII extendido que utiliza los 256 valores:

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	eng	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	\$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	del		

Sabiendo la fila y la columna en la que está un determinado carácter puede componerse el número correspondiente. Por ejemplo, la letra A está en la fila 6 y la columna 5. Su número ASCII es por tanto el 65. El carácter % está en la fila 3 y la columna 7, por lo que su representación ASCII será el 37.

Los caracteres se representan entre comillas simples 'a', 'D'... y abarcan desde '0'..'9', 'a'..'z' y 'A'..'Z'. Un espacio en blanco ( ) también es un carácter.

### Cadena

Los datos de este tipo contendrán una serie finita de caracteres.

Podrán representarse de estas dos formas:

'H' 'O' 'L' 'A' == "HOLA",

No es lo mismo "H" que 'H', ya que el primero es una cadena y el segundo un carácter. Las operaciones relacionadas con las cadenas son: ., Longitud(C1), Extraer (C1,N), **ejemplos:**

C1="Hola", C2="Pepe"

C3= C1 . C2 --> C3="HolaPepe"

X= Longitud (C3) --> X=8

C3=Extraer(C1,3) -->C3="Hol"

Entero, Real, Carácter, Cadena y Lógico son tipos predefinidos en la mayoría de los lenguajes de programación.

Además, el usuario podrá definir sus propios tipos de datos, si estos facilitan de alguna manera la resolución del problema.

## DEFINIR UNA VARIABLE

Para crear una variable es preciso definirla. La definición de una variable se realiza manifestando su tipo, seguida de uno o más espacios, luego se escribe el nombre de la variable y para finalizar punto y coma.

El nombre de la variable puede ser cualquier combinación de letras, claro que sin espacios. Nombres de variables aceptadas son: x, jap007, miedad.

Página | 13

**Importante.** Los nombres buenos de variables nos dicen para que la variable es utilizada, usando buenos nombres se nos hace más fácil la comprensión del programa. La siguiente sentencia define una variable entera llamada **mi\_edad**.

```
int mi_edad;
```

Como práctica general de programación, se debe evitar los nombres incorrectos como j23qrs o xxx y restringir los nombres de variables de una sola letra como x ó y, para valores que sean de uso rápido y no perduren en todo el programa.

Se debe tratar de usar nombres extensos como **mi\_edad** o **contador**. Algunos nombres son fáciles de entender tres semanas después en lugar de romperse la cabeza imaginándose que significan nombres cortos.

## INICIALIZAR UNA VARIABLE

Una vez definida una variable se debe proceder a darle un valor, es cierto que este valor puede cambiar a lo largo del programa, pero es bueno acostumbrarse a dar siempre un valor inicial a nuestras variables.

PHP es un lenguaje no tipado. Esto significa que las variables no necesitan ser inicializadas y su tipo de dato no solo no precisa ser indicado, sino que éste puede cambiar. PHP simplemente sabe el tipo de dato que se utiliza en cada momento dependiendo del contexto en que se utilice.

Por ejemplo:

```
<?php
$var1 = 12;      // $var1 es un integer
$var2 = 12.3;    // $var2 es un float
$var2 = 'hola';  // $var2 es ahora un string
$var2 = '13' + 1; // $var2 es ahora un integer
echo $var2;     // Imprime: 14
```

No es necesario iniciar variables en PHP, sin embargo, es una muy buena práctica. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo dependiendo del contexto en el que son usadas.

## ASIGNACIÓN DE UN VALOR

Se le puede asignar valores a una variable cuantas veces se quiera durante el programa, se le asigna un valor utilizando el caracter de igualdad “=”.



## PALABRAS RESERVADAS O CLAVES

Existen en todos los lenguajes nombres o palabras que ya están siendo usadas, y por eso se les da el nombre de palabras reservadas o claves.

Por ejemplo en el caso de PHP palabras reservadas son: if, while, for, ..., etc. No podemos nombrar por ejemplo una variable definida por nosotros con ninguna palabra reservada porque el compilador encontraría un error.

Página | 14

## CONSTANTES

Las constantes son variables que contienen un valor que no cambia durante todo el programa. Una constante simbólica al igual que cualquier variable tiene un tipo y un nombre.

Ejemplo en PHP:

```
define("CONSTANTE", "Hola mundo.");  
echo CONSTANTE; // muestra "Hola mundo."  
  
// Funciona a partir de PHP 5.3.0  
const CONSTANTE = 'Hola Mundo';  
echo CONSTANTE;
```

## OPERADORES

Un **operador** es un carácter o grupo de caracteres que actúa sobre una, dos o más variables para realizar una determinada **operación** con un determinado **resultado**. Ejemplos típicos de operadores son la suma (+), la diferencia (-), el producto (\*), etc. Los operadores pueden ser **unarios**, **binarios** y **ternarios**, según actúen sobre uno, dos o tres operandos, respectivamente.

## OPERADORES ARITMÉTICOS

Los **operadores aritméticos** son los más sencillos de entender y de utilizar. Todos ellos son operadores binarios. Se utilizan los cinco operadores siguientes:

Suma: +

Resta: -

Multiplicación: \*

División: /

Resto: % (**resto de la división entera**. Este operador se aplica solamente a constantes, variables o expresiones de tipo **entero**).\*

(\*) 23%4 es 3, puesto que el resto de dividir 23 por 4 es 3.

Si **a%b** es cero, **a** es múltiplo de **b**.

## OPERADORES DE ASIGNACIÓN



Los **operadores de asignación** atribuyen a una variable –es decir, depositan en su zona de memoria correspondiente– el resultado de una expresión o el valor.

Variable (en realidad, una variable es un caso particular de una expresión).

El operador de asignación más utilizado es el **operador de igualdad (=)**, que no debe ser confundido con la igualdad lógica (==). Su forma general es:

Página | 15

nombre\_variable = expresión;

Primero se evalúa **expresión** y el resultado se pone en **nombre\_variable**, sustituyendo cualquier otro valor que hubiera en esa posición de memoria anteriormente. Una posible utilización de este operador...

variable = variable + 1;

Desde el punto de vista matemático este ejemplo no tiene sentido pues Equivale a  $0 = 1$ , pero sí lo tiene considerando que en realidad **el operador de asignación (=) representa una sustitución**, pues se toma el valor de la **variable** contenido en la memoria(2),

variable = 2

variable = 2 + 1 (variable = variable + 1)

variable = 3

le suma 1 y el valor resultante vuelve a depositarse en memoria en la zona correspondiente al identificador **variable**, sustituyendo al valor que había anteriormente. Esta operación se denomina **acumulación**.

## OPERADORES RELACIONALES

Una característica imprescindible de cualquier lenguaje de programación es la de **considerar alternativas**, esto es, la de proceder de un modo u otro según se cumplan o no ciertas condiciones. Los **operadores relacionales** permiten estudiar si se cumplen o no esas condiciones.

En un programa si una condición se cumple, el resultado es **cierto**; en caso contrario, el resultado es **falso**. Un 0 representa la condición de **falso**, y cualquier número distinto de 0 equivale a la condición **cierto**. Los **operadores relacionales** son los siguientes:

- Igual que: ==
- Menor que: <
- Mayor que: >
- Menor o igual que: <=
- Mayor o igual que: >=
- Distinto que: !=

Todos los **operadores relacionales** son operadores **binarios** (tienen dos operandos), A continuación, se incluyen algunos ejemplos de estos operadores aplicados a constantes:



`(2==1) // resultado=0 porque la condición no se cumple`

`(3<=3) // resultado=1 porque la condición se cumple`

`(3<3) // resultado=0 porque la condición no se cumple`

`(1!=1) // resultado=0 porque la condición no se cumple`

## OPERADORES LÓGICOS

Los operadores lógicos son operadores binarios que permiten combinar los resultados de los operadores relacionales, comprobando que se cumplen las condiciones necesarias. Como **operadores lógicos** tenemos: el operador Y (&&), el operador O (||) y el operador NO (!). En inglés son los operadores AND, OR y NOT. Su forma general es la siguiente:

`expresion1 && expresion2, expresion1 || expresion, !expresión`

Cuando las **operaciones lógicas son bit a bit**, típicas para hacer testeos de uno o varios bits (o máscaras) disponemos de los siguientes:

& And lógica bit a bit

|| Or lógico bit a bit

Los operadores && y || se pueden combinar entre sí mediante paréntesis. Por ejemplo:

`(2==1) || (-1==1) // el resultado es 1`

`(2==2) && (3==1) // el resultado es 0`

`((2==2) && (3==3)) || (4==0) // el resultado es 1`

`((6==6) || (8==0)) && ((5==5) && (3==2)) // el resultado es 0`

## EXPRESIONES

Una expresión es todo aquello que se evalúa y devuelve un valor. Existen varios tipos de expresiones de acuerdo lo que contienen.

Las **expresiones aritméticas** consisten en una secuencia de operadores y operandos que especifican una operación determinada. Los operandos pueden ser variables, constantes y los operadores aritméticos son (+ - \* / %). Es más sencillo pensar que una expresión aritmética es como una ecuación o una fórmula matemática.

Una expresión aritmética sencilla es:

`area = base * altura;`

En la anterior línea de código, el resultado de la expresión **base \* altura** se guarda en **area**.

Las **expresiones lógicas** también conocidas como expresiones booleanas. Están compuestas por operadores y operandos, los operadores en este caso son operadores relacionales y operadores lógicos.



Este tipo de expresiones es evaluado y devuelve un valor, la diferencia está en que este valor sólo puede ser verdadero o falso.

Los operadores relacionales son(< > == <= >= ...).

Los operadores lógicos son (&& || & |).

Página | 17

Un ejemplo de expresiones lógicas es el siguiente:

if ( (nota > 70) && (nota < 90) )

En la anterior instrucción el segmento (nota > 70) && (nota < 90) devolverá 1 (verdadero) ó 0 (falso).

X	Y	And
V	V	V
V	F	F
F	V	F
F	F	F

X	Y	OR
V	V	V
V	F	V
F	V	V
F	F	F

X	NOT (X)
V	F
F	V

Bibliografía recomendada:

Guía Total del Programador - USERS  
Autor: Nicolás Arrioja Landa Cosio

Programación en C. Metodología, Algoritmos y estructura de datos – Segunda Edición  
Autor: Luis Joyanes Aguilar, Ignacio Zahonero Martínez

C++ Cómo Programar – Sexta Edición  
Autor: P. J. Deitel, H. M. Deitel

Correo electrónico de la cátedra: [programacion.snfco@gmail.com](mailto:programacion.snfco@gmail.com)