Introducción a la Programación

Seminario de Ingreso

Tecnicatura Universitaria en Programación

Contenido

UNIDAD N.º 1: Introducción al Software

Definición de Software

El software de una computadora es un conjunto de instrucciones de programa detalladas que controlan y coordinan los componentes hardware y controlan las operaciones de un sistema informático. El software es como un traductor que hace que nuestras órdenes se conviertan en realidad, manipulando el hardware (la parte física).

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.

```
printf("\nIntrodusca nombre del cliente: ");
scanf("%s",nom);
printf("Introdusca numero de cedula del cliente, sin guiones: ");
scanf("%d",&ced);
printf("Introdusca la edad del cliente: ");
scanf("%d",&edad);
    for(a=0;a<1;a++){
        otro=0;
        printf("\t\n\n Elija su combo: \t\t\n Primer combo= 1 \t\t\n Segundo combo= 2 \t\t\n
        scanf("%d",&combo);

            if(combo==1){c1=c1+3;}
            printf("\nSubtotal del combo 1: %.2f",c1);
            if(combo==2){c2=c2+4;}
            printf("\nSubtotal del combo 2: %.2f",c2);
            if(combo==3){c3=c3+5;}
            printf("\nSubtotal del combo 3: %.2f",c3);
        printf("\n\nDesea otro combo? \t\t\n Si=1 \t\t\n No=2 \t\t\n Opcion:  ");
        scanf("%d",&otro);
        if(otro==1){a=a-1;}
    }
    sub=(c1+c2+c3);
        if(edad>50){des=sub*0.03;}
    printf("\nSu descuento es de: %.2f",des);
```

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 impresora, monitor, mouse, etc.

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.

Cuando un usuario interactúa con una computadora, la interacción está controlada por el sistema operativo. Un usuario se comunica con un sistema operativo a través de una **interfaz de usuario** de ese sistema operativo. Los sistemas operativos modernos utilizan una **interfaz gráfica de usuario**, (*Graphical User Interface*, GUI) que hace uso masivo de iconos, botones, barras y cuadros de diálogo para realizar tareas que se controlan por el teclado o el ratón (mouse) entre otros dispositivos.

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

UNIDAD N.º 2: Lenguajes de programación

Para que un procesador realice un proceso se le debe suministrar, en primer lugar, un **algoritmo** adecuado. El procesador debe ser capaz de interpretar el algoritmo, lo que significa:

- comprender las instrucciones de cada paso
- realizar las operaciones correspondientes.

Cuando el procesador es una computadora, el algoritmo se ha de expresar en un formato que se denomina programa, ya que el pseudocódigo o el diagrama de flujo no son comprensibles por la computadora, aunque pueda entenderlos cualquier programador. Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman programación. Así pues, los lenguajes utilizados para escribir programas de computadoras son los **lenguajes de programación** y **programadores** son los escritores y diseñadores de programas. El proceso de traducir un algoritmo en pseudocódigo a un lenguaje de programación se denomina **codificación**.

Hoy en día, la mayoría de los programadores emplean lenguajes de programación como C++, C#, Java, HTML, PHP, JavaScript, Python, Kotlin, Swift.

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:

Lenguaje de 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 esta. El lenguaje de máquina ordena a la computadora realizar sus operaciones fundamentales una por una. Es el único lenguaje capaz de entender por una computadora.

A+B -> 10100101010010

Lenguaje 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.

Lenguaje 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.

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

Anteriormente hemos nombrado algunos de los lenguajes de alto nivel mas utilizados, ellos son: C++, C, C#, Java, HTML, PHP, JavaScript, .NET, ASP.

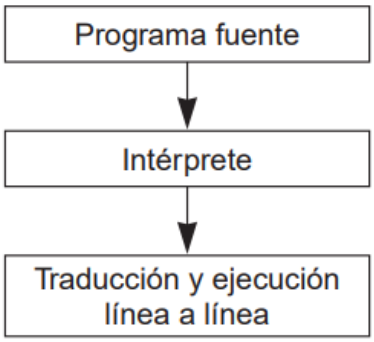
UNIDAD N.º 3: Traductores de lenguajes

El proceso de traducción de un **programa fuente** escrito en un lenguaje de alto nivel a un lenguaje máquina comprensible por la computadora, se realiza mediante programas llamados **“traductores”**. Los traductores de lenguaje son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en **compiladores** e **intérpretes**.

Intérprete

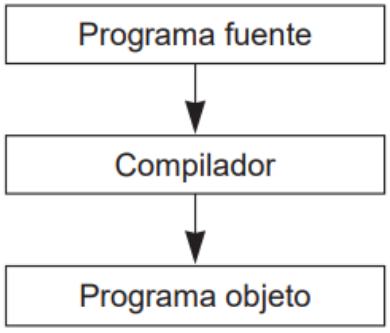
Un intérprete es un traductor que toma un programa fuente, lo traduce y, a continuación, lo ejecuta. El sistema de traducción consiste en: traducir la primera sentencia del programa a lenguaje máquina, se detiene la traducción, se ejecuta la sentencia; a continuación, se traduce la siguiente sentencia, se detiene la traducción,

se ejecuta la sentencia y así sucesivamente hasta terminar el programa.



Compiladores

Un compilador es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina. La traducción del programa completo se realiza en una sola operación denominada compilación del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque. El programa compilado y depurado (eliminados los errores del código fuente) se denomina programa ejecutable porque ya se puede ejecutar directamente y cuantas veces se desee; sólo deberá volver a compilarse de nuevo en el caso de que se modifique alguna instrucción del programa. De este modo el programa ejecutable no necesita del compilador para su ejecución. Los lenguajes compiladores típicos más utilizados son: C, C++, Java, C#, entre otros.



Diferencias entre intérpretes y compiladores

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.

UNIDAD N.º 4: Algoritmos

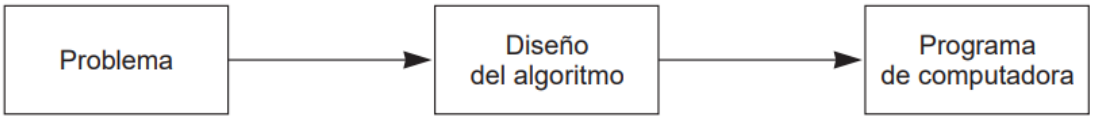
Un algoritmo es un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, algoritmo proviene de *Mohammed al-Khowârizmi*, matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra *algorismus* derivó posteriormente en algoritmo.

El profesor Niklaus Wirth —inventor de Pascal, Modula-2 y Oberon— tituló uno de sus más famosos libros, *Algoritmos + Estructuras de datos = Programas*, significándonos que **sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos**.

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

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



Los pasos para la resolución de un problema son:

1. Diseño del algoritmo, que describe la secuencia ordenada de pasos —sin ambigüedades— que conducen a la solución de un problema dado. (Análisis del problema y desarrollo del algoritmo.)
2. Expresar el algoritmo como un programa en un lenguaje de programación adecuado. (Fase de codificación.)
3. Ejecución y validación del programa por la computadora.

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.

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.

Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser *preciso* e indicar el orden de realización de cada paso.
- Un algoritmo debe estar *definido*. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser *finito*. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: *Entrada, Proceso* y *Salida*. En el algoritmo de receta de cocina citado anteriormente se tendrá:

Entrada: ingredientes y utensilios empleados.
Proceso: elaboración de la receta en la cocina.
Salida: terminación del plato (por ejemplo, cordero).

Veamos otro ejemplo de algoritmo, ahora en versión pseudocódigo (ya veremos más delante de que trata esto):

Reemplazar rueda del auto pinchada

Inicio

------ SI tengo el gato hidráulico en el auto

-------- Sacar el gato hidráulico

-------- Colocar el gato hidráulico

-------- Aflojar y quitar la rueda

-------- Levantar el auto con el gato hidráulico

-------- Quitar la rueda

-------- Colocar la rueda auxiliar

-------- Ajustar la rueda auxiliar colocada

-------- Bajar el auto con el gato hidráulico

-------- Guardar el gato hidráulico

------ SINO

-------- Llamar al servicio de auxilio 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 él. Si logramos construir un algoritmo robusto, cualquier giro inesperado del problema será controlado por el mismo, es decir, debe ser flexible a cambios.

Correctitud de un algoritmo

Un algoritmos es correcto cuando da una solución al problema a tratar y cumple con todos los requerimientos especificados, de tal manera que logremos los objetivos planteados.

Completitud de un algoritmo

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

Eficiencia 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.

Eficacia de un algoritmo

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

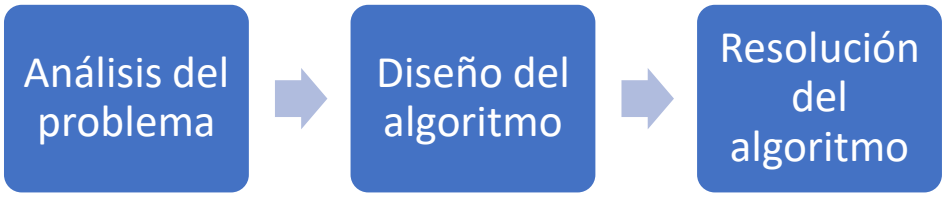
Diferencia entre eficiencia y eficacia

La **eficacia** difiere de la **eficiencia** en el sentido que la **eficiencia** hace referencia a la mejor utilización de los recursos, en tanto, la **eficacia** hace referencia en la capacidad para alcanzar un objetivo sin importar si se le ha dado el mejor uso posible a los recursos.

Resolución de problemas

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto. Para lograr resolver un problema, se pueden seguir los siguientes pasos:

1. **Análisis del problema:** en este paso se define el problema, se lo comprende y se lo analiza con el mayor detalle.
2. **Diseño del algoritmo:** se debe elaborar un algoritmo que refleje paso a paso la resolución del problema.
3. **Resolución del algoritmo:** se debe expresar el algoritmo como un programa en un lenguaje de programación adecuado. (Fase de codificación.)



UNIDAD N.º 5: 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 programación. No se ejecutan en las computadoras, más bien sirven para ayudar al programador a razonar un programa antes de intentar escribirlo en algún lenguaje de programación.

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

Si calificación >= 60 entonces

    Mostrar "Aprobado"

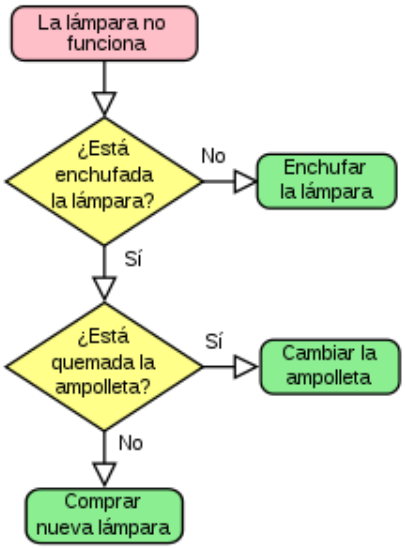
FinSi

En un lenguaje de programación, en este caso PHP, sería:

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

UNIDAD N.º 6: Diagrama de Flujo

El **diagrama de flujo** es la representación gráfica de un algoritmo o proceso (parte de un algoritmo).



Los diagramas de flujo ayudan en la comprensión de los algoritmos. Dichos diagramas se construyen utilizando ciertos símbolos de uso especial como son rectángulos, rombos, óvalos, y pequeños círculos; estos símbolos están conectados entre sí por flechas, conocidas como líneas de flujo.

La ventaja de utilizar un diagrama de flujo 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.

Simbología de los DF

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Linea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

UNIDAD N.º 7: Variables

Definición

Un programa necesita almacenar los diferentes datos que utiliza durante su ejecución. Para ello existen las **variables,** ubicaciones en la memoria de la computadora en las cuales se puede grabar un valor y por la cual se puede recuperar dicho valor más tarde.

En otras palabras, una variable es un espacio para guardar datos/información (valores). Las variables están formadas por un **espacio en el sistema de almacenaje** (memoria principal de un ordenador) y un **nombre simbólico** (identificador) que está asociado a dicho espacio.

¿Cómo funcionan las variables?

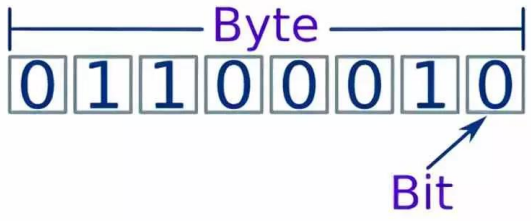
Cuándo creamos una variable lo que hace la pc es reservar un espacio de la memoria RAM. La **memoria RAM** de la computadora puede ser vista como una serie de pequeñas casillas, cada una de las casillas esta numerada secuencialmente, este número que se le asigna representa su dirección de memoria y su objetivo es identificarla.

Una variable reserva una o más casillas en las cuales es posible grabar datos y se accede a ellas mediante el nombre (identificador) asignado a dicha variable. El **nombre 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.

Tipos de variables

Existen variables de diferentes tipos, tales como *int* (almacena nros enteros), *float* (almacena nros con decimales) o *char* (almacena caracteres). 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.



1 Byte = 8 bits

1 Kilobyte = 1024 bits

1 Megabyte = 1024 kilobyte

1 Gigabyte = 1024 megabyte

1 Terabyte = 1024 gigabyte

Enteros

Subconjunto finito de los números enteros, cuyo rango para positivos es de 0 al 65535, y de -32768 al 32767 para números con signo.

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)

Reales

Subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión. Suelen ocupar 4, 6 o 10 bytes.

Por ejemplo 4.8, 53.10149.....

Las operaciones asociadas serian +, -, *,/, %,etc.

Lógicos

Conjunto formado por los valores 'verdadero' (1) y 'falso' (0). Las operaciones asociadas son las lógicas y relacionales: AND, OR, >, <, == (igual), >= (mayor o igual) , <= (menor o igual).

Caracter

Conjunto finito y ordenado de los caracteres que el ordenador reconoce. Se almacenan en un byte. Con 8 bits se podrán almacenar 28= 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 (código que establece un número para cada 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.

Definición de variables

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.

Dato importante: Es aconsejable utilizar buenos nombres para las variables, esto quiere decir que sea representativo y haga saber que guarda dentro suyo. Los nombres buenos de variables nos dicen para que la variable es utilizada, y así se nos hace más fácil la compresión del programa.

Ejemplos:

Int n1 → mal nombre de variable **Int numero1** → buen nombre de variable

Int e → mal nombre de variable **Int edadPersona** → buen nombre de variable

También se aconseja evitar la abreviación en los nombres y utilizar las palabras completas. En vez de 'num' utilizar 'numero'.

Inicialización de variables

Una buena práctica al programar es darle un valor inicial a cada variable creada. 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.

Asignación de valores

Se le pueden asignar valores a una variable cuantas veces se quiera durante la ejecución del programa. Para asignar un valor se utiliza el caracter de igualdad "=".

Ejemplo → Int numero1 = 15;

Palabras reservadas

Son aquellas palabras que tienen asignada una función propia del lenguaje de programación y, por lo tanto, no pueden ser utilizadas con otro fin.

Ejemplos de palabras reservadas: if, for, while… no podremos nombrar ninguna variable con palabras reservadas porque ocurriría un error.

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.

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: suma (+), resta (-), multiplicación (*), etc.

Operadores aritméticos

Los operadores aritméticos son los más sencillos de entender y de utilizar. Todos ellos son operadores binarios.

- 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 de forma entera 23 por 4 es 3. (5 * 4 = 20 + resto = 3 → 23).

Operadores de asignación

Los operadores de asignación atribuyen a una variable el resultado de una expresión o el valor. El operador de asignación más utilizado es el operador de igualdad (=), que no debe ser confundido con la igualdad lógica (==). Su forma general es: **nombre_variable = expresión;**

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). Algunos ejemplos:

- (2 == 1) → falso (0)
- (2 == 2) → verdadero (1)
- (3 < 5) → verdadero (1)
- (3 <= 3) → verdadero (1)
- (5 != 5) → falso (0)

Operadores lógicos

Los operadores lógicos son operadores binarios que permiten combinar los resultados de los operadores relacionales, comprobando que se cumplen las condiciones necesarias.

- AND && (y)
- OR || (o)
- NOT ! (no)

expresion1 && expresion2 (expresión 1 **y** expresión 2)

expresion1 || expresion2 (expresión 1 **o** expresión 2)

!expresión (negación de expresión)

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

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