¿Cómo se construye el Software?

El software, como cualquier otro producto, se construye aplicando un proceso que conduzca a un resultado de calidad, que satisfaga las necesidades de quienes lo utilizan. Un proceso de desarrollo de software es una secuencia estructurada de actividades que conduce a la obtención de un producto de software. En definitiva, un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. En este caso el objetivo es construir un producto de software nuevo o mejorar uno existente.

Figura 1: Proceso de Construcción del Software

Pueden identificarse cuatro actividades fundamentales que son comunes a todos los procesos de software:

1. **Especificación del software**: donde clientes y profesionales definen el software que se construirá, sus características y las restricciones para su uso.
2. **Desarrollo del software**, donde se diseña y programa el software.
3. **Validación del software**, donde se controla que el software satisfaga lo que el cliente quiere.
4. **Evolución del software**, donde se incorporan mejoras y nuevas características que permitirán a ese producto adaptarse a las necesidades cambiantes del cliente y el mercado.

Si consideramos las características del software que se explicaron anteriormente, determinamos como conclusión que el software no se obtiene por medio de un proceso de manifactura en serie o como líneas de producción, sino que para obtenerlo usamos un **proyecto**, que se lo puede definir como un esfuerzo planificado, temporal y único, realizado para crear productos o servicios únicos que agreguen valor. Estos proyectos utilizan **procesos** que definen que tareas deben realizar las personas que trabajan en el proyecto, para obtener los resultados deseados, utilizando como apoyo herramientas que facilitarán su trabajo. En este caso el resultado deseado en el **Producto de Software.**

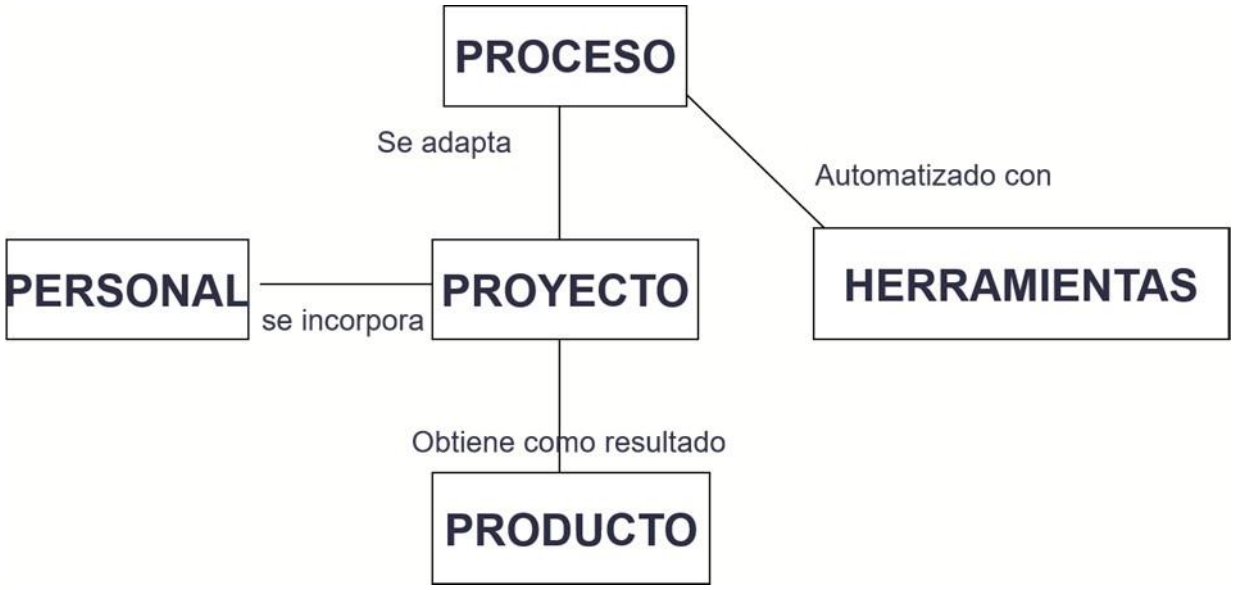

Figura 2: Relación entre Proceso, Proyecto y Producto en el desarrollo de Software

Diseño de Algoritmos

El hombre, en el día a día, se enfrenta constantemente a diferentes problemas que debe solucionar y para lograr solucionarlos hace uso de herramientas que le facilitan la tarea. Así, podemos pensar el uso de una calculadora para poder sumar el precio de los productos en un local y así cobrarle al cliente.

Al igual que la calculadora, la computadora también sirve para resolver problemas, pero la diferencia está en la capacidad de procesamiento de las computadoras, que hace que se puedan resolver problemas de gran complejidad, que, si los quisiéramos resolver manualmente, nos llevaría mucho tiempo o ni siquiera podríamos llegar a resolverlos.

Un programador es antes que nada una persona que resuelve problemas; el programador procede a resolver un problema, a partir de la definición de un algoritmo y de la traducción de dicho algoritmo a un programa que ejecutará la computadora.

En la oración anterior se nombran algunos conceptos que debemos profundizar:

Algoritmo: un algoritmo es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados tal que, partiendo de ciertos datos de entrada, permite obtener ciertos resultados que conforman la solución del problema. Así, como en la vida real, cuando tenemos que resolver un problema, o lograr un objetivo, por ejemplo: “Tengo que atarme los cordones”, para alcanzar la solución de ese problema, realizamos un conjunto de pasos, de manera ordenada y secuencial. Es decir, podríamos definir un algoritmo para atarnos los cordones de la siguiente forma:

1. Ponerme las zapatillas.
2. Agarrar los cordones con ambas manos.
3. Hacer el primer nudo.
4. Hacer un bucle con cada uno de los cordones.
5. Cruzar los dos bucles y ajustar.
6. Corroborar que al caminar los cordones no se sueltan y la zapatilla se encuenta correctamente atada.

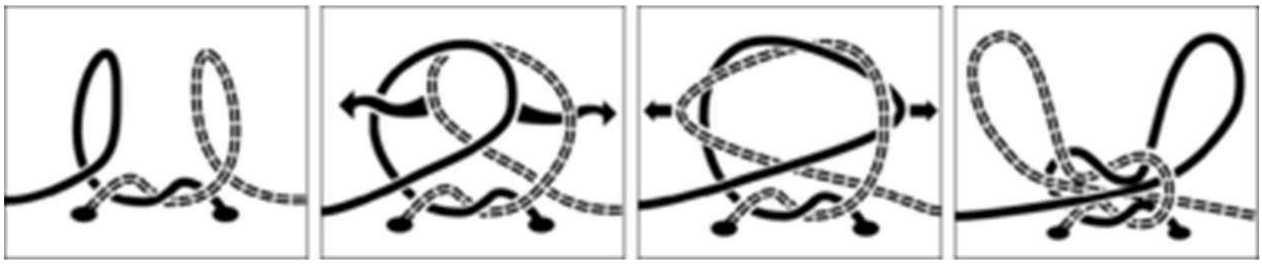

Figura 3: Algoritmo gráfico para atarse los cordones

El concepto de algoritmo es fundamental en el proceso de programación de una computadora, pero si nos detenemos a observar a nuestro alrededor, así como el ejemplo anterior podemos descubrir muchos otros: nos están dando un algoritmo cuando nos indican la forma de llegar a una dirección dada, seguimos algoritmos cuando conducimos un automóvil o cualquier tipo de vehículo. Todos los procesos de cálculo matemático que normalmente realiza una persona en sus tareas cotidianas, como sumar, restar, multiplicar o dividir, están basados en algoritmos que fueron aprendidos en la escuela primaria. Como se ve, la ejecución de algoritmos forma parte de la vida moderna.

Por otro lado, la complejidad de los distintos problemas que podamos abordar puede variar desde muy sencilla a muy compleja, dependiendo de la situación y la cantidad de elementos que intervienen. En casos de mayor complejidad suele ser una buena solución dividir al problema en diferentes subproblemas que puedan ser resueltos de manera independiente. De esta forma la solución final al problema inicial será determinada por las distintas soluciones de los problemas más pequeños cuya resolución es más sencilla.

Programa: luego de haber definido el algoritmo necesario, se debe traducir dicho algoritmo en un conjunto de instrucciones, entendibles por la computadora, que le indican a la misma lo que debe hacer; este conjunto de instrucciones conforma lo que se denomina, un programa.

Para escribir un programa se utilizan lenguajes de programación, que son lenguajes que pueden ser entendidos y procesados por la computadora. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Algoritmos

Concepto

Es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados, tal que, partiendo de ciertos datos de entrada, permite obtener ciertos resultados que conforman la solución del 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 específicamente definido. Es decir, si se ejecuta un mismo algoritmo dos veces, con los mismos datos de entada, 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. Debe tener un inicio y un final.

• Un algoritmo debe ser correcto: el resultado del algoritmo debe ser el resultado esperado.

• Un algoritmo es independiente tanto del lenguaje de programación en el que se expresa como de la computadora que lo ejecuta.

Como vimos anteriormente, el programador debe constantemente resolver problemas de manera algorítmica, lo que significa plantear el problema de forma tal que queden indicados los pasos necesarios para obtener los resultados pedidos, a partir de los datos conocidos. Lo anterior implica que un algoritmo básicamente consta de tres elementos: Datos de Entrada, Procesos y la Información de Salida.

Figura 4 : Estructura de un programa, datos de entrada y salida

Cuando dicho algoritmo se transforma en un programa de computadora:

- Las entradas se darán por medio de un dispositivo de entrada (como los vistos en el bloque anterior), como pueden ser el teclado, disco duro, teléfono, etc. Este proceso se lo conoce como entrada de datos, operación de lectura o acción de leer.
- Las salidas de datos se presentan en dispositivos periféricos de salida, que pueden ser pantalla, impresora, discos, etc. Este proceso se lo conoce como salida de datos, operación de escritura o acción de escribir.

Dado un problema, para plantear un algoritmo que permita resolverlo, es conveniente entender correctamente la situación problemática y su contexto, tratando de deducir del mismo los elementos ya indicados (entradas, procesos y salida). En este sentido entonces, para crear un algoritmo:

1. Comenzar identificando los resultados esperados, porque así quedan claros los objetivos a cumplir.

2. Luego, individualizar los datos con que se cuenta y determinar si con estos datos es suficiente para llegar a los resultados esperados. Es decir, definir los datos de entrada con los que se va a trabajar para lograr el resultado.

3. Finalmente si los datos son completos y los objetivos claros, se intentan plantear los procesos necesarios para pasar de los datos de entrada a los datos de salida.

Para comprender esto, veamos un ejemplo:

Problema:

Obtención del área de un rectángulo:

Altura: 5 cm

Base: 10 cm

1. **Resultado que espero obtener**: área del rectángulo.
Esta es la **Salida**: área

2. Los datos con los que se dispone, es decir las **entradas** de datos
son:Dato de Entrada 1: altura: 5 cm
Dato de Entrada 2: base: 10 cm

3. El **proceso** para obtener el área del rectángulo
es:área = base * altura área = 50

Como se especificó anteriormente, un algoritmo es independiente del lenguaje de programación que se utilice. Una de las formas de representarlo es el

1. Lenguaje de especificación de algoritmos: pseudocódigo.

Pseudocódigo

Conocido como lenguaje de especificación de algoritmos, el pseudocódigo tiene una estructura muy similar al lenguaje natural y sirve para poder expresar algoritmos y programas de forma independiente del lenguaje de programación. Además, es muy utilizado para comunicar y representar ideas que puedan ser entendidas por programadores que conozcan distintos lenguajes. El pseudocódigo luego se traduce a un lenguaje de programación específico ya que la computadora no puede ejecutar el pseudocódigo. Su uso tiene ventajas porque permite al programador una mejor concentración de la lógica y estructuras de control y no preocuparse de las reglas de un lenguaje de programación específico.

Un ejemplo básico de pseudocódigo, considerando el ejemplo utilizado anteriormente, es el siguiente:

```
INICIO FUNCION CALCULAR_AREA
  DEFINIR BASE: 5
  DEFINIR ALTURA: 10
  DEFINIR AREA: BASE * ALTURA
  DEVOLVER AREA
FIN
```

Lenguajes de Programación

Concepto

Los lenguajes de programación son lenguajes que pueden ser entendidos y procesados por la computadora.

Tipos de Lenguajes de Programación

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- Lenguajes máquina
- Lenguaje de bajo nivel (ensamblador)
- Lenguajes de alto nivel

La elección del lenguaje de programación a utilizar depende mucho del objetivo del software. Por ejemplo, para desarrollar aplicaciones que deben responder en tiempo real como por ejemplo, el control de la velocidad crucero en un sistema de navegación de un auto[1], debemos tener mayor control del hardware disponible, por lo que privilegiaremos lenguajes de más bajo nivel que nos permitan hacer un uso más eficiente de los recursos. En cambio, para aplicaciones de escritorio como sistemas de gestión de productos, calendarios, correo electrónico, entre otras privilegiaremos la elección de lenguajes de más alto nivel que nos permitan ser más eficientes en cuanto a la codificación ya que, en términos generales, es necesario escribir menos líneas de código en los lenguajes de alto nivel, que para su equivalente en bajo nivel.

[1] **Control de velocidad crucero**: El control de velocidad, también conocido como regulador de velocidad o autocrucero, es un sistema que controla de forma automática el factor de movimiento de un vehículo de motor. El conductor configura la velocidad y el sistema controlará la válvula de aceleración del vehículo para mantener la velocidad de forma continua.

El lenguaje máquina

Los lenguajes máquina son aquellos que están escritos en lenguajes cuyas instrucciones son cadenas binarias (cadenas o series de caracteres -dígitos- O y 1) que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

En los primeros tiempos del desarrollo de los ordenadores era necesario programarlos directamente de esta forma, sin embargo, eran máquinas extraordinariamente limitadas, con muy pocas instrucciones por lo que aún era posible; en la actualidad esto es completamente irrealizable por lo que es necesario utilizar lenguajes más fácilmente comprensibles para los humanos que deben ser traducidos a código máquina para su ejecución.

Ejemplo de una instrucción:
1110 0010 0010 0001 0000 0000 0010 0000

El lenguaje de bajo nivel

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual, que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador o assembler. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como mnemotécnicas (mnemonics). Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, ADD, SUB, DIV, etc.; en español, SUM, para sumar, RES, para restar, DIV, para dividir etc.

Lenguajes de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no dependen del diseño del hardware o de una computadora en particular. En consecuencia, los programas escritos en lenguaje de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora. Esto es posible porque los lenguajes de alto nivel son traducidos a lenguaje máquina por un tipo de programa especial denominado "compilador". Un compilador toma como entrada un algoritmo escrito en un lenguaje de alto nivel y lo convierte a instrucciones inteligibles por el ordenador; los compiladores deben estar adaptados a cada tipo de ordenador pues deben generar código máquina específico para el mismo.

Ejemplos de Lenguajes de Alto Nivel:

C, C++, Java, Python, VisualBasic, C#, JavaScript

¿Qué es un Programa?

Es un algoritmo escrito en algún lenguaje de programación de computadoras.

Pasos para la construcción de un programa

DEFINICIÓN DEL PROBLEMA

En este paso se determina la información inicial para la elaboración del programa. Es donde se determina qué es lo que debe resolverse con el computador, el cual requiere una definición clara y precisa.

Es importante que se conozca lo que se desea que realice la computadora; mientras la definición del problema no se conozca del todo, no tiene mucho caso continuar con la siguiente etapa.

ANÁLISIS DEL PROBLEMA

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

- Los datos de entrada.
- Los datos de salida
- Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es la de colocarse en el lugar de la computadora y analizar qué es lo que se necesita que se ordene y en qué secuencia para producir los resultados esperados.

DISEÑO DEL ALGORITMO

Se puede utilizar algunas de las herramientas de representación de algoritmos mencionadas anteriormente. Este proceso consiste en definir la secuencia de pasos que se deben llevar a cabo para conseguir la salida identificada en el paso anterior.

CODIFICACIÓN

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora. La serie de instrucciones detalladas se conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

PRUEBA Y DEPURACIÓN

Se denomina prueba de escritorio a la comprobación que se hace de un algoritmo para saber si está bien realizado. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes.

Elementos de un Programa

Variables y Constantes

A la hora de elaborar un programa es necesario usar datos; en el caso del ejemplo del cálculo del área del rectángulo, para poder obtener el área del mismo, necesitamos almacenar en la memoria de la computadora el valor de la base y de la altura, para luego poder multiplicar sus valores.

Recordemos que no es lo mismo grabar los datos en memoria que grabarlos en el disco duro. Cuando decimos grabar en memoria nos estaremos refiriendo a grabar esos datos en la RAM. Ahora bien, para grabar esos datos en la RAM podemos hacerlo utilizando dos elementos, llamados: variables y constantes. Los dos elementos funcionan como fuentes de almacenamiento de datos, la gran diferencia entre los dos es que en el caso de las constantes su valor dado no varía en el transcurso de todo el programa.

Podría decirse que tanto las variables como las constantes, son direcciones de memoria con un valor, ya sea un número, una letra, o valor nulo (cuando no tiene valor alguno, se denomina valor nulo). Estos elementos permiten almacenar temporalmente datos en la computadora para luego poder realizar cálculos y operaciones con los mismos. Al almacenarlos en memoria, podemos nombrarlos en cualquier parte de nuestro programa y obtener el valor del dato almacenado, se dice que la variable nos devuelve el valor almacenado.

A continuación, se muestra un esquema, que representa la memoria RAM, como un conjunto de filas, donde, en este caso, cada fila representa un byte y tiene una dirección asociada.

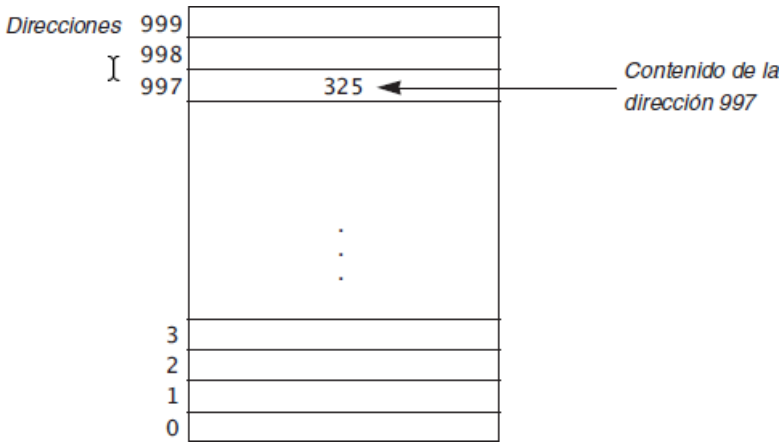

Figura 5: Representación de la memoria RAM

Variables

Son elementos de almacenamiento de datos. Representan una dirección de memoria en donde se almacena un dato, que puede variar en el desarrollo del programa. Una variable es un grupo de bytes asociado a un nombre o identificador, y a través de dicho nombre se puede usar o modificar el contenido de los bytes asociados a esa variable.

En una variable se puede almacenar distintos tipos de datos. De acuerdo al tipo de dato, definido para cada lenguaje de programación, será la cantidad de bytes que ocupa dicha variable en la memoria.

Type	Size	Range
byte	1 byte	−128 to 127
short	2 bytes	−32,768 to 32,767
int	4 bytes	−2,147,483,648 to 2,147,483,647
long	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	$-3.40282347 \times 10^{38}$ to 3.4028347×10^{38}
double	8 bytes	$-1.79769313486231570 \times 10^{308}$ to $1.79769313486231570 \times 10^{308}$
char	2 bytes	one character
String	2 or more bytes	one or more characters

Tabla 1: Tabla de equivalencias de las unidades de memoria

Lo más importante de la definición de las variables y la elección del tipo de datos asociados es el significado de la variable, o su semántica: ya que en base al tipo de datos seleccionado serán las operaciones que podamos realizar con esa variable, por ejemplo: si tenemos la variable edad deberíamos seleccionar un tipo de datos como integer (número entero) ya que las operaciones relacionadas serán de comparación, sumas o restas y no es necesario tener una profundidad de decimales.

A continuación, se listan algunos de los tipos de datos más comunes y sus posibles usos:

Tipo de Datos	Significado	Ejemplos de uso
Byte	Número entero de 8 bits. Con signo	Temperatura de una habitación en grados Celsius

Short	Número entero de 16 bits. Con signo	Edad de una Persona
int	Número entero de 32 bits. Con signo	Distancia entre localidades medida en metros
Long	Número entero de 64 bits. Con signo	Producto entre dos distancias almacenadas en variables tipo int como la anterior
Float	Número Real de 32 bits.	Altura de algún objeto
Double	Número Real de 64 bits.	Proporción entre dos magnitudes
Boolean	Valor lógico: true (verdadero) o false (falso)	Almacenar si el usuario ha aprobado un examen o no

Tabla 2: Tipos de dato

Para algunos lenguajes de programación es posible especificar si las variables numéricas tendrán o no signo (es decir que puedan tomar valores negativos y positivos, o sólo positivos). La elección de tener o no signo (definidas con la palabra reservada unsigned) depende del significado de la variable ya que al no tenerlo podremos almacenar el doble de valores, por ejemplo, una variable short sin signo posee valores desde el 0 al 65535.

Constantes

Elementos de almacenamiento de datos. Representan una dirección de memoria en donde se almacena un dato pero que no varía durante la ejecución del programa. Se podría pensar en un ejemplo de necesitar utilizar en el programa el número pi, como el mismo no varía, se puede definir una constante pi y asignarle el valor 3.14.

Operadores

Los programas de computadoras se apoyan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad. Los operadores son símbolos especiales que sirven para ejecutar una determinada operación, devolviendo el resultado de la misma.

Para comprender lo que es un operador, debemos primero introducir el concepto de Expresión. Una expresión es, normalmente, una ecuación matemática, tal como 3 + 5. En esta expresión, el símbolo más (+) es el operador de suma, y los números 3 y 5 se llaman operandos. En síntesis, una expresión es una secuencia de operaciones y operandos que especifica un cálculo.

Existen diferentes tipos de operadores:

Operador de asignación

Es el operador más simple que existe, se utiliza para asignar un valor a una variable o a una constante. El signo que representa la asignación es el = y este operador indica que el valor a la derecha del = será asignado a lo que está a la izquierda del mismo.

Ejemplo en pseudocódigo:

Entero edad = 20

Decimal precio = 25.45

Operadores aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división entera (%), por ejemplo 50%8=2 porque necesitamos obtener un número entero que queda luego de determinar la cantidad de veces que 8 entra en 50.

Ejemplo:

Expresión	Operador	Operandos	Resultado arrojado
5 * 7	*	5 , 7	35
6 + 3	+	6 , 3	9
20 – 4	-	20 , 4	16
50 % 8	%	50 , 8	2
45/5	/	45,5	9

Operador	Significado
+	Suma
-	Resta
*	Producto
/	División
%	Resto de la división entera

Tabla 3 Resumen Operadores Aritméticos

Los operadores unitarios

Los operadores unitarios requieren sólo un operando; que llevan a cabo diversas operaciones, tales como incrementar/decrementar un valor de a uno, negar una expresión, o invertir el valor de un booleano.

Operador	Descripción	Ejemplo	Resultado
++	operador de incremento; incrementa un valor de a 1	int suma=20; suma++;	suma=21
--	operador de decremento; Reduce un valor de a 1	int resta=20; resta--;	resta=19
!	operador de complemento lógico; invierte el valor de un valor booleano	boolean a=true; boolean b= !a;	b=false

Operadores Condicionales

Son aquellos operadores que sirven para comparar valores. Siempre devuelven valores booleanos: TRUE O FALSE. Pueden ser Relacionales o Lógicos.

Operadores relacionales

Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor.

Los operadores relacionales determinan si un operando es mayor que, menor que, igual a, o no igual a otro operando. La mayoría de estos operadores probablemente le resultará familiar. Tenga en cuenta que debe utilizar " == ", no " = ", al probar si dos valores primitivos son iguales.

Operador	Significado
==	Igual a
!=	No igual a
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Ejemplos

Expresión	Operador	Resultado
a > b	>	true: si a es mayor que b false: si a es menor que b
a >= b	>=	true:si a es mayor o igual que b false: si a es menor que b
a < b	<	true: si a es menor que b false: si a es mayor que b
a <= b	<=	true: si a es menor o igual que b false: si a es mayor que b.
a == b	==	true: si a y b son iguales. false: si a y b son diferentes.
a != b	!=	true: si a y b son diferentes false: si a y b son iguales.

Operadores lógicos

Los operadores lógicos (AND, OR y NOT), sirven para evaluar condiciones complejas. Se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales.

Expresión	Nombre Operador	Operador	Resultado
a && b	AND	&&	true: si a y b son verdaderos. false: si a es falso, o si b es falso, o si a y b son falsos
a || b	OR	||	true: si a es verdadero, o si b es verdadero, o si a y b son verdaderos. false: si a y b son falsos.

Debe notarse que en ciertos casos el segundo operando no se evalúa porque no es necesario (si ambos tienen que ser true y el primero es false ya se sabe que la condición de que ambos sean true no se va a cumplir).