

UNIDAD 1.

FUNDAMENTOS DE PROGRAMACIÓN

PROGRAMACIÓN
CFGS DAW

Autores: Blanca Calderón

2023

ÍNDICE DE CONTENIDO

1. Introducción	5
2. Algoritmo.	5
3. Ciclo de vida de un programa.....	6
3.1 Fase de definición.....	6
3.2 Fase de desarrollo	7
3.3 Fase de mantenimiento.....	8
4. Documentación	8
5. Objetos de un programa.....	9
5.1 Constantes	9
5.2 Variables	9
5.3 Expresiones.....	9
5.4 Operadores	10
5.4.1 Relacionales	10
5.4.2 Aritméticos.	11
5.4.3 Lógicos o booleanos.	12
5.4.4 Paréntesis ()	13
5.4.5 Operador Alfanumérico (+)	14
5.4.6 Orden de evaluación de los operadores.....	14
6. Representación	14
6.1 Diagramas de flujo (ordinogramas)	14
6.1.1 Símbolos de operación.....	15
6.1.2 Símbolos de decisión.....	16
6.1.3 Símbolos de conexión	17
6.1.4 Ejemplo	17
6.2 Pseudocódigo	18
6.2.1 Ejemplo	19

UD01. FUNDAMENTOS DE PROGRAMACIÓN

1. INTRODUCCIÓN

La razón principal por la que una persona utiliza un **ordenador** es para **resolver problemas** (en el sentido más general de la palabra), o, en otras palabras, procesar una información para obtener un resultado a partir de unos datos de entrada.

Los ordenadores resuelven los problemas **mediante la utilización de programas escritos** por los programadores. Los programas de ordenador no son entonces más que métodos para resolver problemas. Por ello, para escribir un programa, lo primero es que el programador sepa resolver el problema que estamos tratando.

El programador debe **identificar** cuáles son los **datos de entrada** y a partir de ellos obtener los datos de salida, es decir, la solución, a la que se llegará por medio del procesamiento de la información que se realizará mediante la utilización de un método para resolver el problema que denominaremos algoritmo.

¿Cuántas acciones de las que has realizado hoy, crees que están relacionadas con la programación?

Hagamos un repaso de los primeros instantes del día: te ha despertado la alarma de tu teléfono móvil-despertador, has preparado el desayuno utilizando el microondas, mientras desayunabas has visto u oído las últimas noticias a través de tu receptor de televisión digital terrestre, te has vestido y puede que hayas utilizado el ascensor para bajar al portal y salir a la calle, etc. Quizá no es necesario que continuemos más para darnos cuenta de que casi todo lo que nos rodea, en alguna medida, está relacionado con la programación, los programas y el tratamiento de algún tipo de información.

El volumen de datos que actualmente manejamos y sus innumerables posibilidades de tratamiento constituyen un vasto territorio en el que los programadores tienen mucho que decir.

Generalmente, la primera razón que mueve a una persona hacia el aprendizaje de la programación es utilizar el ordenador como herramienta para resolver problemas concretos. Como en la vida real, la búsqueda y obtención de una solución a un problema determinado, utilizando medios informáticos, se lleva a cabo siguiendo unos pasos fundamentales. En la siguiente tabla podemos ver estas analogías.

Resolución de problemas	
En la vida real...	En Programación...
Observación de la situación o problema.	Análisis del problema: requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle.
Pensamos en una o varias posibles soluciones.	Diseño o desarrollo de algoritmos: procedimiento paso a paso para solucionar el problema dado.
Aplicamos la solución que estimamos más adecuada.	Resolución del algoritmo elegido en la computadora: consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona verdaderamente el problema.

¿Qué virtudes debería tener nuestra solución?

- **Corrección y eficacia:** si resuelve el problema adecuadamente.
- **Eficiencia:** si lo hace en un tiempo mínimo y con un uso óptimo de los recursos del sistema.

Para conseguirlo, cuando afrontemos la construcción de la solución tendremos que tener en cuenta los siguientes conceptos:

1. **Abstracción:** se trata de realizar un análisis del problema para descomponerlo en problemas más pequeños y de menor complejidad, describiendo cada uno de ellos de manera precisa. **Divide y vencerás**, es una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.
2. **Encapsulación:** consiste en ocultar la información para poder implementarla de diferentes maneras sin que esto influya en el resto de elementos.
3. **Modularidad:** estructuraremos cada parte en módulos independientes, cada uno de ellos tendrá su función correspondiente.

Citas para pensar

“El comienzo de la sabiduría para un ingeniero de software es reconocer la diferencia entre hacer que un programa funcione y conseguir que lo haga correctamente.” Roger Pressman

2. ALGORITMO

Por algoritmo entendemos un **conjunto ordenado y finito de operaciones que permiten resolver un problema** que además cumplen las siguientes características:

- Tiene un número finito de pasos.
- Acaba en un tiempo finito. Si no acabase nunca, no se resolvería el problema.
- Todas las operaciones deben estar definidas de forma precisa y sin ambigüedad.
- Puede tener varios datos de entrada y como mínimo un dato de salida.

Un claro ejemplo de algoritmo es una receta de cocina, donde tenemos unos pasos que hay que seguir en un orden y deben de estar bien definidos, tiene un tiempo finito y tiene unos datos de entrada (ingredientes) y una salida (el plato).

Por ejemplo, el algoritmo para freír un huevo podría ser el siguiente:

Datos de entrada: Huevo, aceite, sartén, fuego.

Datos de salida: huevo frito.

Procedimiento:

1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, cascar el huevo e introducirlo.
4. Cubrir el huevo de aceite.
5. Cuando el huevo esté hecho, retirarlo.

La codificación de un algoritmo en un ordenador se denomina **programa**.

La diferencia fundamental entre algoritmo y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado **lenguaje de programación** para que puedan ser ejecutados en el ordenador y así obtener la solución.

Los lenguajes de programación son sólo un medio para expresar el algoritmo y el ordenador un procesador para ejecutarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a algoritmos diferentes, igualmente válidos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de diseño descendente o **diseño modular (top-down design)**. Este sistema se basa en el lema **divide y vencerás**.

Para representar gráficamente los algoritmos que vamos a diseñar, tenemos a nuestra disposición diferentes herramientas que ayudarán a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje que nos interese. Entre otras tenemos:

- **Diagramas de flujo:** esta técnica utiliza símbolos gráficos para la representación del algoritmo. Suele utilizarse en las fases de análisis.
- **Pseudocódigo:** esta técnica se basa en el uso de palabras clave en lenguaje natural, constantes, variables, otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.
- **Tablas de decisión:** En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Suele ser una técnica de apoyo al pseudocódigo cuando existen situaciones condicionales complejas.

<https://www.youtube.com/watch?v=IFlxFhfS2LY>

3. CICLO DE VIDA DE UN PROGRAMA

La creación de cualquier programa (software o SW) implica la realización de **tres pasos** genéricos:

- Definición ¿Qué hay que desarrollar?
- Desarrollo.
- Mantenimiento.

3.1 Fase de definición

Se intenta **caracterizar el sistema que se ha de construir**. Se debe determinar la información que ha de usar el sistema, qué funciones debe realizar, qué condiciones existen, cuáles son las interfaces del sistema y qué criterios de validación se utilizarán.

El estudio y definición del problema dan lugar al planteamiento del problema que se escribirá en la documentación del programa. Si no se sabe lo que se busca, no se lo reconoce si se lo encuentra. Es decir que, si no sabemos con claridad qué es lo que tenemos que resolver, no podremos encontrar una solución. Aquí se declara cuál es la situación de partida y el entorno de datos de entrada, los resultados deseados, dónde deben registrarse y cuál será la situación final a la que debe conducir el problema después de ser implementado.

3.2 Fase de desarrollo

En esta fase **se diseñan estructuras** de datos y de los programas, **se escriben y documentan** éstos, y **se prueba** el software.

En esta etapa del ciclo de vida de desarrollo de programas, los analistas trabajan con los requerimientos del software desarrollados en la etapa de análisis. Se determinan todas las tareas que cada programa realiza, como así también, la forma en que se organizarán estas tareas cuando

se codifique el programa. Los **problemas** cuando son **complejos**, se pueden resolver más eficientemente con el ordenador cuando **se descomponen en subproblemas que sean más fáciles de solucionar** que el original. La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples que pueden ser implementados para su solución en el ordenador se denomina diseño descendente (top-down design como hemos comentado un poco más arriba). Las ventajas más importantes del diseño descendente son:

- El problema se comprende más fácilmente al dividirse en partes más simples denominadas módulos.
- Las modificaciones en los módulos son más fáciles.
- La comprobación del problema se puede verificar fácilmente.

En esta etapa, además, se utilizan auxiliares de diseño, que son diagramas y tablas que facilitan la delineación de las tareas o pasos que seguirá el programa, por ejemplo: diagramas de flujo, pseudocódigo, etc.

En esta fase, **se convierte el algoritmo en programa**, escrito en un lenguaje de programación de alto nivel como C, Java, etc. La codificación del programa suele ser una tarea pesada que requiere un conocimiento completo de las características del lenguaje elegido para conseguir un programa eficaz. Sin embargo, si el diseño del algoritmo se ha realizado en detalle con acciones simples y con buena legibilidad, el proceso de codificación puede reducirse a una simple tarea mecánica. Las reglas de sintaxis que regulan la codificación variarán de un lenguaje a otro y el programador deberá conocer en profundidad dichas reglas para poder diseñar buenos programas.

Para aumentar la productividad, es necesario adoptar una serie de normas, como ser:

- Estructuras aceptables (programación estructurada)
- Convenciones de nominación: maneras uniformes de designación de archivos y variables.
- Convenciones de comentarios.

3.3 Fase de mantenimiento

Una vez obtenido el **programa fuente**, es necesaria su **traducción al código máquina**, ya que los programas escritos en un lenguaje de alto nivel no son directamente ejecutables por el ordenador. Según el tipo de traductor que se utilice, los lenguajes de alto nivel se clasifican en **lenguajes interpretados y lenguajes compilados**.

Son **lenguajes interpretados** aquellos en los que el sistema traduce una instrucción y la ejecuta, y así sucesivamente con las restantes.

Son lenguajes compilados aquellos en los que, primero se traduce el programa fuente completo, obteniéndose un código intermedio o módulo objeto (**programa objeto**); después, se fusiona éste con rutinas o librerías necesarias para su ejecución en un proceso llamado **linkado** y que obtiene como resultado un módulo ejecutable (**programa ejecutable**). La **ventaja de los lenguajes compilados**, frente a los interpretados, son su **rápida ejecución** y, en caso de necesitar posteriores ejecuciones del mismo programa, se hará del ejecutable almacenado.

La **puesta a punto** consta de las siguientes etapas:

- Detección de errores.
- Depuración de errores.
- Prueba del programa.

En cada una de estas fases se pueden detectar problemas que nos hacen replantearnos conceptos de la fase anterior y rehacer el software creado con las oportunas correcciones.

4. DOCUMENTACIÓN

La mayor parte de los proyectos exigen la realización de una **planificación** previa. Esta planificación debe determinar el modelo de ciclo de vida a seguir, los plazos para completar cada fase y los recursos necesarios en cada momento. Todo esto se debe plasmar en una documentación completa y detallada de toda la aplicación.

La **documentación** asociada al software puede clasificarse en interna y externa. La **documentación interna** corresponde a la que se incluye **dentro del código fuente** de los programas. Nos aclaran aspectos de las propias instrucciones del programa. La **documentación externa** es la que corresponde a todos los documentos relativos al diseño de la aplicación, a la descripción de la misma y sus módulos correspondientes, a los manuales de usuario y los manuales de mantenimiento.

✍ En el módulo que nos ocupa tendremos que documentar el código fuente que desarrollaremos durante la elaboración de los distintos programas.

Sean cuales sean las fases en las que realicemos el proceso de desarrollo de software, y casi independientemente de él, siempre se debe **aplicar un modelo de ciclo de vida**.

Ciclo de vida del software: es una sucesión de estados o fases por las cuales pasa un software a lo largo de su "vida".

El proceso de desarrollo puede involucrar siempre las siguientes etapas mínimas:

- Especificación y Análisis de requisitos.
- Diseño.
- Codificación.
- Pruebas.
- Instalación y paso a Producción.
- Mantenimiento.

5. OBJETOS DE UN PROGRAMA

Como hemos visto, en todo el proceso de resolución de un problema mediante la creación de software, después del análisis del problema y del diseño del algoritmo que pueda resolverlo, es necesario traducir éste a un lenguaje que exprese claramente cada uno de los pasos a seguir para su correcta ejecución. Este lenguaje recibe el nombre de **lenguaje de programación**.

Lenguaje de programación: conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas.

Gramática del lenguaje: reglas aplicables al conjunto de símbolos y palabras especiales del lenguaje de programación para la construcción de sentencias correctas.

Léxico: es el conjunto finito de símbolos y palabras especiales, es el vocabulario del lenguaje.

Sintaxis: son las posibles combinaciones de los símbolos y palabras especiales. Está relacionada con la forma de los programas.

Semántica: es el significado de cada construcción del lenguaje, la acción que se llevará a cabo.

Hay que tener en cuenta que pueden existir sentencias sintácticamente correctas, pero semánticamente incorrectas. Por ejemplo, *“Un avestruz dio una tremenda dentellada a su cuidador”* está bien construida sintácticamente, pero es evidente que semánticamente no.

Una característica relevante de los lenguajes de programación es, precisamente, que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos. A través de este conjunto se puede lograr la construcción de un programa de forma colaborativa.

Los lenguajes de programación pueden ser clasificados en función de lo cerca que estén del lenguaje humano o del lenguaje de los computadores. El lenguaje de los computadores son códigos binarios, es decir, secuencias de unos y ceros. Detallaremos seguidamente las características principales de los lenguajes de programación.

Lenguaje máquina.



GNU GPL [Sasa Stefanovic](#)

Éste es el lenguaje utilizado directamente por el procesador, consta de un conjunto de instrucciones codificadas en binario. Es el sistema de códigos directamente interpretable por un [circuito microprogramable](#).

Se trata del primer lenguaje utilizado para la programación de computadores. De hecho, cada máquina tenía su propio conjunto de instrucciones codificadas en ceros y unos. Cuando un algoritmo está escrito en este tipo de lenguaje, decimos que está en código máquina.

Programar en este tipo de lenguaje presentaba los siguientes **inconvenientes**:

- Cada programa era válido **sólo para un tipo de procesador** u ordenador.
- **La lectura o interpretación de los programas era extremadamente difícil** y, por tanto, **insertar modificaciones resultaba muy costoso**.
- Los programadores de la época debían **memorizar largas combinaciones de ceros y unos**, que equivalían a las instrucciones disponibles para los diferentes tipos de procesadores.
- Los programadores se encargaban de introducir los códigos binarios en el computador, lo que provocaba **largos tiempos de preparación y posibles errores**.

A continuación, se muestran algunos códigos binarios equivalentes a las operaciones de suma, resta y movimiento de datos en lenguaje máquina.

Algunas operaciones en lenguaje máquina.	
Operación	Lenguaje máquina
SUMAR	00101101
RESTAR	00010011
MOVER	00111010

Dada la complejidad y dificultades que ofrecía este lenguaje, fue sustituido por otros más sencillos y fáciles utilizar. No obstante, hay que tener en cuenta que todos **los programas para poder ser ejecutados, han de traducirse siempre al lenguaje máquina** que es el único que entiende la computadora.

Rellena el hueco con el concepto adecuado:

En el lenguaje máquina de algunos procesadores, la combinación 00101101 equivale a la operación de .

Lenguaje ensamblador.

La evolución del lenguaje máquina fue el lenguaje ensamblador.

Las instrucciones ya no son secuencias binarias, se sustituyen por códigos de operación que describen una operación elemental del procesador. Es un lenguaje de bajo nivel, al igual que el lenguaje máquina, ya que dependen directamente del hardware donde son ejecutados.

Mnemotécnico: son palabras especiales, que sustituyen largas secuencias de ceros y unos, utilizadas para referirse a diferentes operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.

En ensamblador, cada instrucción (mnemotécnico) se corresponde a una instrucción del procesador. En la siguiente tabla se muestran algunos ejemplos.

Algunas operaciones y su mnemotécnico en lenguaje Ensamblador.

Operación	Lenguaje Ensamblador
MULTIPLICAR	MUL
DIVIDIR	DIV
MOVER	MOV

En el siguiente gráfico puedes ver parte de un programa escrito en lenguaje ensamblador. En color rojo se ha resaltado el código máquina en [hexadecimal](#), en magenta el código escrito en ensamblador y en azul, las direcciones de memoria donde se encuentra el código.



[German](#)

Pero, aunque ensamblador fue un intento por aproximar el lenguaje de los procesadores al lenguaje humano, presentaba **múltiples dificultades**:

- Los programas **seguían dependiendo directamente del hardware** que los soportaba.
- Los programadores **tenían que conocer detalladamente la máquina** sobre la que programaban, ya que debían hacer un uso adecuado de los recursos de dichos sistemas.
- **La lectura, interpretación o modificación** de los programas seguía presentando **dificultades**.

Todo programa escrito en lenguaje ensamblador necesita de un intermediario, que realice la traducción de cada una de las instrucciones que componen su código al lenguaje máquina correspondiente. Este intermediario es el **programa ensamblador**. El programa original escrito en lenguaje ensamblador constituye el **código fuente** y el programa traducido al lenguaje máquina se conoce como **programa objeto** que será directamente ejecutado por la computadora.

Lenguajes compilados.

Para paliar los problemas derivados del uso del lenguaje ensamblador y con el objetivo de acercar la programación hacia el uso de un lenguaje más cercano al humano que al del computador, nacieron los **lenguajes compilados**.

Algunos ejemplos de este tipo de lenguajes son: **Pascal, Fortran, Algol, C, C++**, etc.

Al ser lenguajes más cercanos al humano, también se les denomina **lenguajes de alto nivel**. Son más fáciles de utilizar y comprender, las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador.

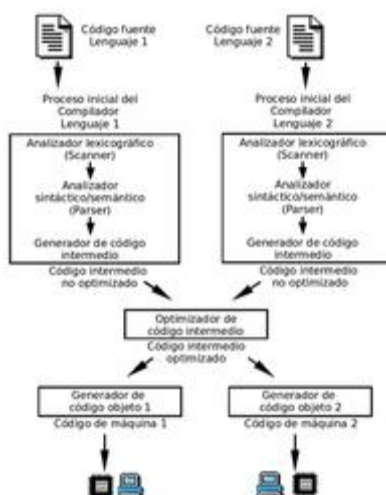
¿Cuáles son sus **ventajas**?

- Son **mucho más fáciles de aprender y de utilizar** que sus predecesores.
- Se **reduce el tiempo para desarrollar programas**, así como **los costes**.
- Son **independientes del hardware**, los programas pueden ejecutarse en diferentes tipos de máquina.
- La **lectura, interpretación y modificación** de los programas es **mucho más sencilla**.

Pero un programa que está escrito en un lenguaje de alto nivel también **tiene que traducirse** a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman **compiladores**.

Compilador: es un programa cuya función consiste en traducir el código fuente de un programa escrito en un lenguaje de alto nivel a lenguaje máquina. Al proceso de traducción se le conoce con el nombre de compilación.

Para ilustrar el proceso de compilación de programas te proponemos el siguiente esquema:



[GNU Free Documentation License.](#)

[Magnus Colossus](#)

El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como **código objeto**. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el **código ejecutable**.

Lenguajes interpretados.

¿Recuerdas que en un apartado anterior ya hablamos de que existen dos formas de traducir los programas escritos en un lenguaje de alto nivel a código máquina y que una de esas formas es mediante un intérprete? Pues bien, ahora vamos a ver cuáles son las características de los lenguajes interpretados.

Se caracterizan por estar diseñados para que su ejecución se realice a través de un **intérprete**. Cada instrucción escrita en un lenguaje interpretado se analiza, traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

Intérprete: es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Los programas en lenguajes interpretados presentan el inconveniente de ser algo más lentos que los compilados, ya que han de ser traducidos durante su ejecución. Por otra parte, necesitan disponer en la máquina del programa intérprete ejecutándose, algo que no es necesario en el caso de un programa compilado, para los que sólo es necesario tener el programa ejecutable para poder utilizarlo.

Ejemplos de lenguajes interpretados son: **Perl, PHP, Python, JavaScript**, etc.

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes que podemos denominar **pseudo-compilados** o **pseudo-interpretados**, es el caso del **Lenguaje Java**. Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de bytecodes, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). La [Máquina Virtual Java](#) se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.

Debes conocer

Puedes entender por qué Java es un lenguaje compilado e interpretado a través del siguiente enlace:

[El lenguaje Java es compilado e interpretado.](#)

Entendemos por **objeto** de un programa **todo aquello que pueda ser manipulado por las instrucciones**. En ellos se almacenarán tanto los datos de entrada como los de salida (resultados).

Sus atributos son:

- **Nombre**: el identificador del objeto.
- **Tipo**: conjunto de valores que puede tomar.
- **Valor**: elemento del tipo que se le asigna.

5.1 Constantes

Son objetos cuyo **valor permanece invariable** a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo.

Por ejemplo: $\pi = 3.14.1592$ $e = 2.718281$

También son utilizadas las constantes para facilitar la modificabilidad de los programas, es decir para hacer más independientes ciertos datos del programa. Por ejemplo, supongamos un programa en el que cada vez que se calcula un importe al que se debe sumar el IVA utilizáramos siempre el valor 0.16, en caso de variar este índice tendríamos que ir buscando a lo largo del programa y modificando dicho valor, mientras que si le damos nombre y le asignamos un valor, podremos modificar dicho valor con mucha más facilidad.

5.2 Variables

Son objetos cuyo **valor puede ser modificado** a lo largo de la ejecución de un programa.

Por ejemplo: una variable para calcular el área de una circunferencia determinada, una variable para calcular una factura, etc.

5.3 Expresiones

Las expresiones **según el resultado que produzcan** se clasifican en:

- **Numéricas**: Son las que producen resultados de tipo numérico. Se construyen mediante los operadores aritméticos.

Por ejemplo:

π	*
$\text{sqr}(x)$	

- Alfanuméricas: Son las que producen resultados de tipo alfanumérico. Se construyen mediante operadores alfanuméricos.

Por ejemplo:

"Don " + "José"

- Booleanas o lógicas: Son las que producen resultados de tipo Verdadero o Falso. Se construyen mediante los operadores relacionales y lógicos.

Por ejemplo:

```
a < 0
(a > 1) and (b < 5)
```

5.4 Operadores

Son símbolos que hacen de enlace entre los argumentos de una expresión.


5.4.1 Relacionales

Se usan para formar expresiones que al ser evaluadas devuelven un valor booleano: verdadero o falso.

Operador	Definición
<	Menor que
>	Mayor que
==	Igual que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto que

Ejemplos:

Expresión	Resultado
A' < 'B'	Verdadero, ya que en código ASCII la A está antes que la B
1 < 6	Verdadero
10 < 2	Falso

 **ASCII** (acrónimo inglés de American Standard Code for Information Interchange — Código Estándar Estadounidense para el Intercambio de Información), es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. Puedes ver la tabla en el siguiente enlace: <http://ascii.cl/es/>

5.4.2 Aritméticos

Se utilizan para realizar operaciones aritméticas.

Operador	Definición
+	Suma
-	Resta
*	Multiplicación
\wedge	Potencia
/	División
%	Resto de la división

Ejemplos:

Expresión	Resultado
$3 + 5 - 2$	6
$24 \% 3$	0
$8 * 3 - 7 / 2$	25

5.4.3 Lógicos o booleanos

La combinación de expresiones con estos operadores produce el resultado verdadero o falso.

Operador	Definición
No	Negación
Y	Conjunción
O	Disyunción

El comportamiento de un operador lógico se define mediante su correspondiente [tabla de verdad](#), en ella se muestra el resultado que produce la aplicación de un determinado operador a uno o dos valores lógicos. Las operaciones lógicas más usuales son:

- NO lógico (NOT) o negación:

A	NOT A
V	F
F	V

El operador NOT invierte el valor: Si es verdadero (V) devuelve falso (F), y viceversa.

- O lógica (OR) o disyunción:

A	B	A OR B
V	V	V
V	F	V
F	V	V
F	F	F

El operador OR devuelve verdadero (V) si alguno de los dos valores es verdadero. De lo contrario, devuelve Falso (F).

- Y lógica (AND) o conjunción:

A	B	A AND B
V	V	V
V	F	F
F	V	F
F	F	F

El operador AND devuelve Verdadero (V) solo si ambos valores son verdaderos. En cualquier otro caso devuelve Falso (F).

Ejemplos (Suponiendo que $a < b$):

Expresión	Resultado
$9 = (3 * 3)$	Verdadero
$3 < 2$	Verdadero
$9 = (3 * 3) \text{ Y } 3 < 2$	Verdadero
$3 > 2 \text{ Y } b < a$	Verdadero Y Falso = Falso
$3 > 2 \text{ O } b < a$	Verdadero O Falso = Verdadero
$\text{no}(a < b)$	No Verdadero = Falso
$5 > 1 \text{ Y NO}(b < a)$	Verdadero Y no Falso = Verdadero

5.4.4 Paréntesis ()

Anidan expresiones.

Operación $(3 * 2) + (6 / 2)$ Resultado 9

Ejemplos:

5.4.5 Operador Alfanumérico (+)

Une datos de tipo alfanumérico. También llamado concatenación. Ejemplos:

Expresión	Resultado
"Ana " + "López"	Ana López
"saca " + "puntas"	sacapuntas

5.4.6 Orden de evaluación de los operadores

A la hora de resolver una expresión, el orden a seguir es el siguiente:

1. Paréntesis.
2. Potencia ^

3. Multiplicación y división * /
4. Sumas y restas + -
5. Concatenación +

- 6. Relacionales < <= > >= etc.
- 7. Negación NOT
- 8. Conjunción AND
- 9. Disyunción OR

La **evaluación de operadores** de igual orden se realiza **de izquierda a derecha**. Este orden de evaluación tiene algunas modificaciones en determinados lenguajes de programación.

6. REPRESENTACIÓN

Existen diversas formas de representación de algoritmos. Las más importantes son los diagramas de flujo (también llamados 'ordinogramas') y el pseudocódigo.

6.1 Diagramas de flujo (ordinogramas)

Durante el diseño de un programa y en sus fases de análisis y programación, surge la necesidad de **representar de una manera gráfica los flujos que van a seguir los datos** manipulados por el mismo, así como la secuencia lógica de las operaciones para la resolución del problema.

Esta representación gráfica debe tener las siguientes cualidades:

1. Sencillez en su construcción.
2. Claridad en su comprensión.
3. Normalización en su diseño.
4. Flexibilidad en sus modificaciones.

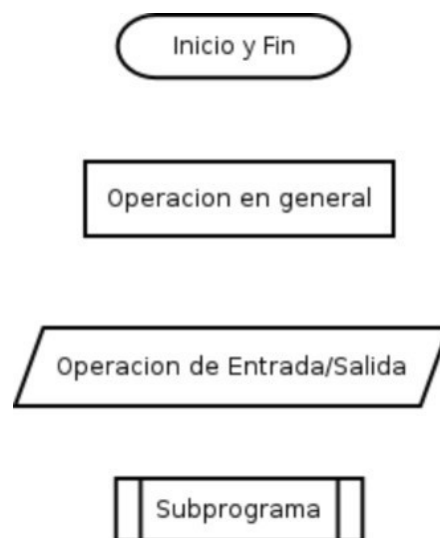
✈ En la práctica se suelen utilizar indistintamente los términos **diagrama de flujo**, **organigrama** y **ordinograma** para referenciar cualquier representación gráfica de los flujos de datos o de las operaciones de un programa.

Es importante diferenciarlos porque no corresponden a las mismas fases de diseño de los programas. Aunque utilicen algunos símbolos comunes, el significado de éstos no es el mismo.

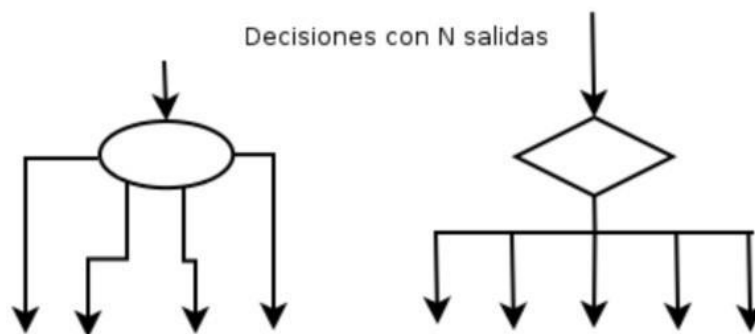
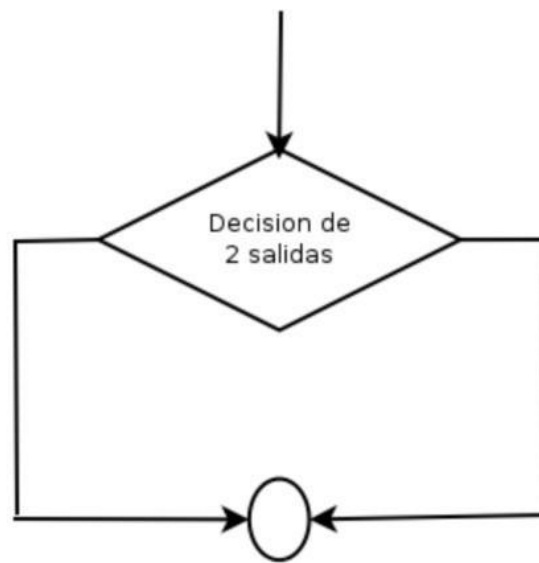
En la representación de ordinogramas es conveniente seguir las siguientes reglas:

- El comienzo del programa figurará en la parte superior del ordinograma.
- El símbolo de comienzo deberá aparecer una sola vez en el ordinograma.
- El flujo de las operaciones será, siempre que sea posible de arriba a abajo y de izquierda a derecha.
- Se evitarán siempre los cruces de líneas utilizando conectores.

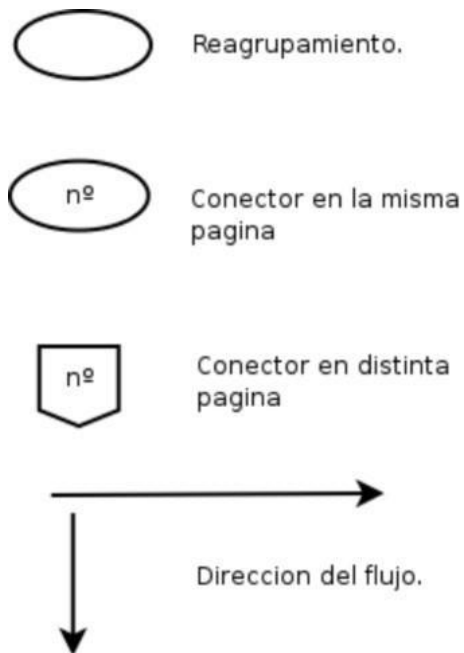
Esta será la representación que utilizaremos durante el curso.



6.1.1 Símbolos de operación



6.1.2 Símbolos de decisión

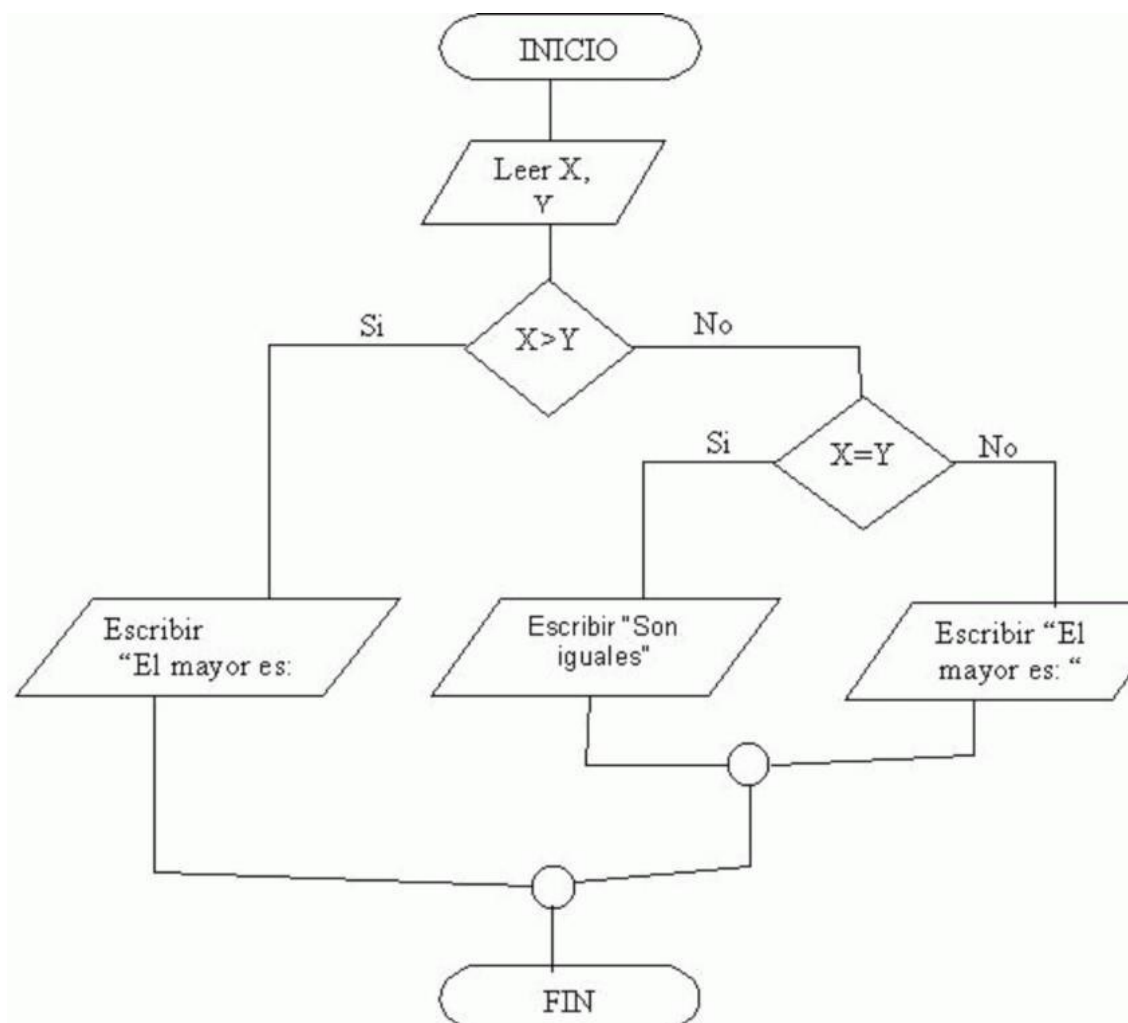


6.1.3 Símbolos de conexión

6.1.4 Ejemplo

Algoritmo que lee dos números "X" e "Y", determina si son iguales, y en caso de no serlo, indica cuál de ellos es el mayor.

Su representación gráfica mediante ordinograma podemos verla en el gráfico:



6.2 Pseudocódigo

Además de las representaciones gráficas, un programa puede describirse mediante un **lenguaje intermedio entre el lenguaje natural y el lenguaje de programación**, de tal manera que permita flexibilidad para expresar las acciones que se van a realizar y, también imponga algunas limitaciones, que tienen importancia cuando se quiere codificar el programa a un lenguaje de programación determinado.

La notación en pseudocódigo se caracteriza por:

- Facilitar la obtención de la solución mediante la utilización del diseño descendente o Top-down.
- Ser una forma de codificar los programas o algoritmos fácil de aprender y utilizar.
- Posibilitar el diseño y desarrollar los algoritmos de una manera independiente del lenguaje de programación que se vaya a utilizar cuando se implemente el programa.
- Facilitar la traducción del algoritmo a un lenguaje de programación específico.
- Permitir una gran flexibilidad en el diseño del algoritmo que soluciona el problema, ya que se

pueden representar las acciones de una manera mas abstracta, no estando sometidas a las reglas tan rígidas que impone un lenguaje de programación.

- f) Posibilitar futuras correcciones y actualizaciones en el diseño del algoritmo por la utilización una serie de normas, que acotan el trabajo del desarrollador.

Cuando se escribe un algoritmo mediante la utilización de pseudocódigo, se debe "sangrar" el texto con respecto al margen izquierdo, con la finalidad de que se comprenda más fácilmente el diseño que se está realizando.

6.2.1 Ejemplo

Algoritmo que lee dos números "X" e "Y", determina si son iguales, y en caso de no serlo, indica cuál de ellos es el mayor.

Su representación gráfica mediante pseudocódigo será:

