UNIDAD 1. INTRODUCCIÓN A LA PROGRAMACIÓN

PROGRAMACIÓN

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROFESOR: ALBERTO ALEMANY

CONTENIDO

UD1. INTRODUCCIÓN A LA PROGRAMACIÓN

1. INTRODUCCIÓN

¿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 o radio-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.

En esta primera unidad realizaremos un recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo con qué vamos a trabajar, qué técnicas podemos emplear y qué es lo que pretendemos conseguir. Continuando con el análisis de las diferentes formas de programación existentes, identificaremos qué fases conforman el desarrollo de un programa, avanzaremos detallando las características relevantes de cada uno de los lenguajes de programación disponibles. Finalmente, tendremos la oportunidad de conocer con qué herramientas podríamos desarrollar nuestros programas, escogiendo entre una de ellas.

2. PROGRAMAS

2.1. Buscando una solución

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.
2. **Divide y vencerás,** esta suele ser considerada 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.
3. **Encapsulación:** consiste en ocultar la información para poder implementarla de diferentes maneras sin que esto influya en el resto de elementos.
4. **Modularidad:** estructuraremos cada parte en módulos independientes, cada uno de ellos tendrá su función correspondiente.

2.2. Algoritmos y programas

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, pudiendo 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.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

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.

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 (Estructura de datos que se utiliza en los lenguajes de programación que no puede cambiar su contenido en el transcurso del programa.), variables (Estructura de datos que, como su nombre indica, puede cambiar de contenido a lo largo de la ejecución de

un programa.), otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.

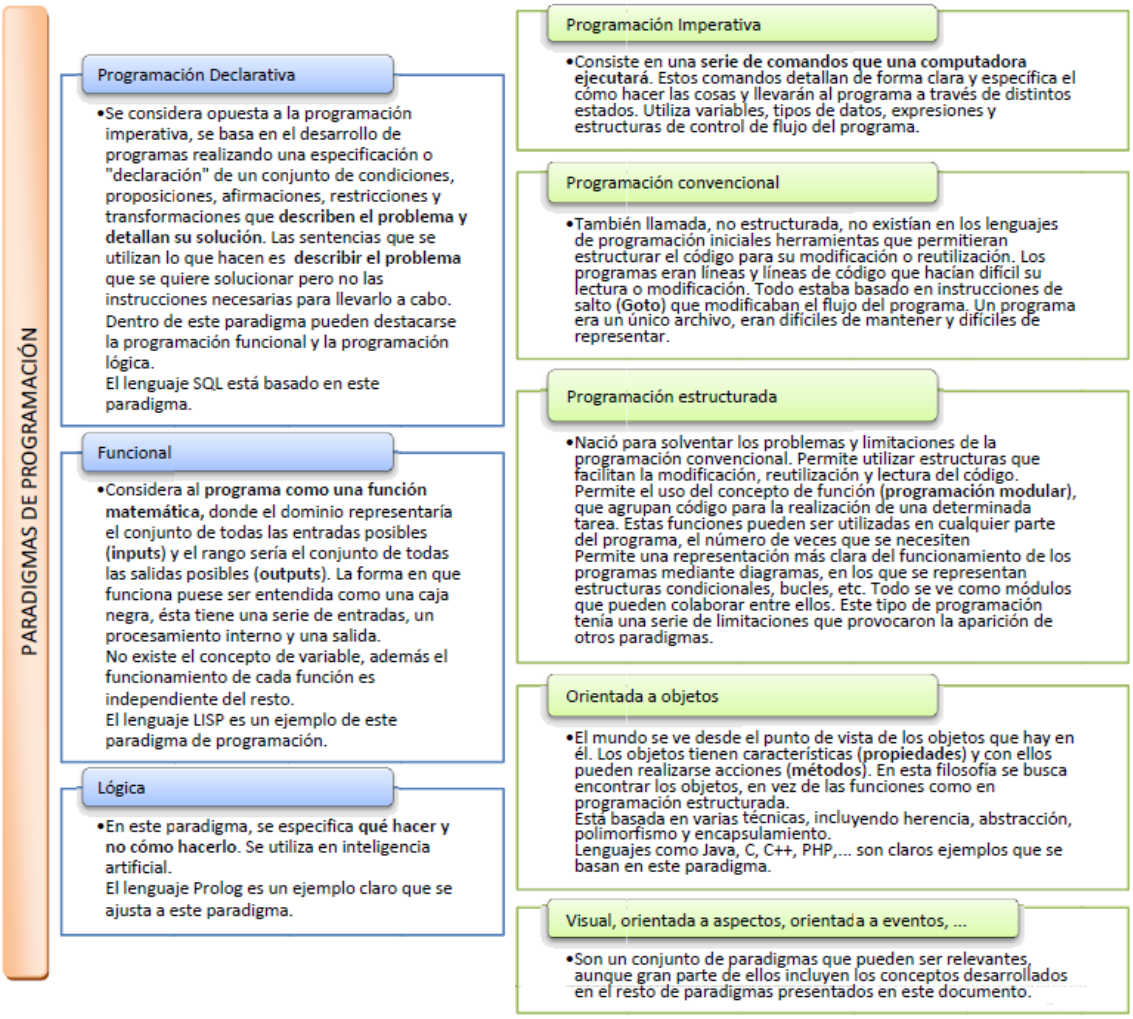
- **Tablas de decisión**: En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Suele ser una técnica de apoyo al pseudocódigo cuando existen situaciones condicionales complejas.

3. PARADIGMAS DE LA PROGRAMACIÓN

¿Cuántas formas existen de hacer las cosas? Supongo que estarás pensando: varias o incluso, muchas. Pero cuando se establece un patrón para la creación de aplicaciones nos estamos acercando al significado de la palabra paradigma.

> **Paradigma de programación**: es un modelo básico para el diseño y la implementación de programas. Este modelo determinará cómo será el proceso de diseño y la estructura final del programa.

El paradigma representa un enfoque particular o filosofía para la construcción de software. Cada uno tendrá sus ventajas e inconvenientes, será más o menos apropiado, pero no es correcto decir que exista uno mejor que los demás.



Como habrás podido apreciar, existen múltiples paradigmas, incluso puede haber lenguajes de programación que no se clasifiquen únicamente dentro de uno de ellos. Un lenguaje como Smalltalk es un lenguaje basado en el paradigma orientado a objetos. El lenguaje de programación Scheme, en cambio, soporta sólo programación funcional. Python, soporta múltiples paradigmas.

¿Cuál es el objetivo que se busca con la aplicación de los diferentes enfoques? Fundamentalmente, reducir la dificultad para el mantenimiento de las aplicaciones, mejorar el rendimiento del programador y, en general, mejorar la productividad y calidad de los programas.

4. FASES DE LA PROGRAMACIÓN

Sea cual sea el estilo que escojamos a la hora de automatizar una determinada tarea, debemos realizar el proceso aplicando un método a nuestro trabajo. Es decir, sabemos que vamos a dar solución a un problema, aplicando una filosofía de desarrollo y lo haremos dando una serie de pasos que deben estar bien definidos.

El proceso de creación de software puede dividirse en diferentes fases:

- Fase de resolución del problema.
- Fase de implementación.
- Fase de explotación y mantenimiento.

A continuación, analizaremos cada una de ellas.

4.1. Resolución del problema

Para el comienzo de esta fase, es necesario que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. A su vez, la fase de resolución del problema puede dividirse en dos etapas:

a) Análisis

Por lo general, el análisis indicará la especificación de requisitos que se deben cubrir. Los contactos entre el analista/programador y el cliente/usuario serán numerosos, de esta forma podrán ser conocidas todas las necesidades que precisa la aplicación. Se especificarán los procesos y estructuras de datos que se van a emplear. La creación de *prototipos* será muy útil para saber con mayor exactitud los puntos a tratar.

El análisis inicial ofrecerá una idea general de lo que se solicita, realizando posteriormente sucesivos refinamientos que servirán para dar respuesta a las siguientes cuestiones:

- ✓ ¿Cuál es la información que ofrecerá la resolución del problema?
- ✓ ¿Qué datos son necesarios para resolver el problema?

La respuesta a la primera pregunta se identifica con los resultados deseados o las salidas del problema. La respuesta a la segunda pregunta indicará qué datos se proporcionan o las entradas del problema.

En esta fase debemos aprender a analizar la documentación de la empresa, investigar, observar todo lo que rodea el problema y recopilar cualquier información útil.

b) Diseño

En esta etapa se convierte la especificación realizada en la fase de análisis en un diseño más detallado, indicando el comportamiento o la secuencia lógica de instrucciones capaz de resolver el problema

planteado. Estos pasos sucesivos, que indican las instrucciones a ejecutar por la máquina, constituyen lo que conocemos como *algoritmo*.

Consiste en plantear la aplicación como una única operación global, e ir descomponiéndola en operaciones más sencillas (diseño top-down), detalladas y específicas. En cada nivel de refinamiento, las operaciones identificadas se asignan a módulos separados.

Hay que tener en cuenta que antes de pasar a la implementación del algoritmo, hemos de asegurarnos que tenemos una solución adecuada. Para ello, todo diseño requerirá de la realización de la **prueba o traza** del programa. Este proceso consistirá en un seguimiento paso a paso de las instrucciones del algoritmo utilizando datos concretos. Si la solución aportada tiene errores, tendremos que volver a la fase de análisis para realizar las modificaciones necesarias o tomar un nuevo camino para la solución. Sólo cuando el algoritmo cumpla los requisitos y objetivos especificados en la fase de análisis se pasará a la fase de implementación.

4.2. Implementación

a) Codificación o construcción

Esta etapa consiste en transformar o traducir los resultados obtenidos a un determinado *lenguaje de programación*. Para comprobar la calidad y estabilidad de la aplicación se han de realizar una serie de pruebas que comprueben las funciones de cada módulo (pruebas unitarias), que los módulos funcionan bien entre ellos (pruebas de interconexión) y que todos funcionan en conjunto correctamente (pruebas de integración).

Cuando realizamos la traducción del algoritmo al lenguaje de programación debemos tener en cuenta las reglas gramaticales y la sintaxis de dicho lenguaje. Obtendremos entonces el código fuente, lo que normalmente conocemos por programa.

Pero para que nuestro programa comience a funcionar, antes debe ser traducido a un lenguaje que la máquina entienda. Este proceso de traducción puede hacerse de dos formas, compilando o interpretando el código del programa.

Compilación: *Es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje que la máquina es capaz de interpretar.*

Compilador: *programa informático que realiza la traducción. Recibe el código fuente, realiza un análisis lexicográfico, semántico y sintáctico, genera un código intermedio no optimizado, optimiza dicho código y finalmente, genera el código objeto para una plataforma específica.*

Intérprete: *programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.*

Una vez traducido, sea a través de un proceso de compilación o de interpretación, el programa podrá ser ejecutado.

b) Prueba de ejecución y validación

Para esta etapa es necesario implantar la aplicación en el sistema donde va a funcionar, debe ponerse en marcha y comprobar si su funcionamiento es correcto. Utilizando diferentes datos de prueba se verá si el programa responde a los requerimientos especificados, si se detectan nuevos errores, si éstos son bien gestionados y si la interfaz es amigable. Se trata de poner a prueba nuestro programa para ver su respuesta en situaciones difíciles.

Mientras se detecten errores y éstos no se subsanen no podremos avanzar a la siguiente fase. Una vez corregido el programa y testeado se documentará mediante:

- Documentación interna: Encabezados, descripciones, declaraciones del problema y comentarios que se incluyen dentro del código fuente.

- Documentación externa: Son los manuales que se crean para una mejor ejecución y utilización del programa.

4.3. Explotación

Cuando el programa ya está instalado en el sistema y está siendo de utilidad para los usuarios, decimos que se encuentra en fase de explotación.

Periódicamente será necesario realizar evaluaciones y, si es necesario, llevar a cabo modificaciones para que el programa se adapte o actualice a nuevas necesidades, pudiendo también corregirse errores no detectados anteriormente. Este proceso recibe el nombre de mantenimiento del software.

Mantenimiento del software*:* *es el proceso de mejora y optimización del software después de su entrega al usuario final. Involucra cambios al software en orden de corregir defectos y dependencias encontradas durante su uso, así como la adición de nuevas funcionalidades para mejorar la usabilidad y aplicabilidad del software.*

Será imprescindible añadir una documentación adecuada que facilite al programador la comprensión, uso y modificación de dichos programas.

5. LENGUAJES DE PROGRAMACIÓN

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.*

Hay que tener en cuenta que pueden existir sentencias sintácticamente correctas, pero semánticamente incorrectas. Por ejemplo, “Un avestruz dio un zarpazo 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.

5.1. Lenguaje máquina

Este 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 (*Dispositivo o conjunto de dispositivos de propósito general, que, según sea necesario, se programan para resolver distintos problemas*).

Este fue el 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.

Operación	Lenguaje máquina	Decimal
SUMAR	00101101	45
RESTAR	00010011	19
MOVER	00111010	58

5.2. 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.

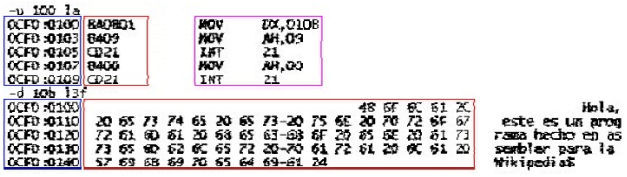
En ensamblador, cada instrucción (mnemotécnico) se corresponde a una instrucción del procesador. En la siguiente tabla se muestran algunos ejemplos.

> **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.*

Algunas operaciones y su mnemotécnico en lenguaje Ensamblador.

Operación	Lenguaje Ensamblador
MULTIPLICAR	MUL
DIVIDIR	DIV
MOVER	MOV

En el siguiente gráfico puede s ver parte de un programa escrito en lenguaje ensamblador. En color rojo se ha resaltado el código máquina en hexadecimal (Sistema numérico en base 16, esto significa que contiene 16 símbolos únicos para representar datos: los números del 0 al 9 y las letras de la A a la F.), en magenta el código escrito en ensamblador y en azul, las direcciones de memoria donde se encuentra el código.



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 debía n 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 pro grama objeto que será directamente ejecutado por la computadora.

5.3. 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++ (Es el lenguaje de programación C ampliado para poder utilizar los mecanismos que permitan la manipulación de objetos. Es un lenguaje multiparadigma.), 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.

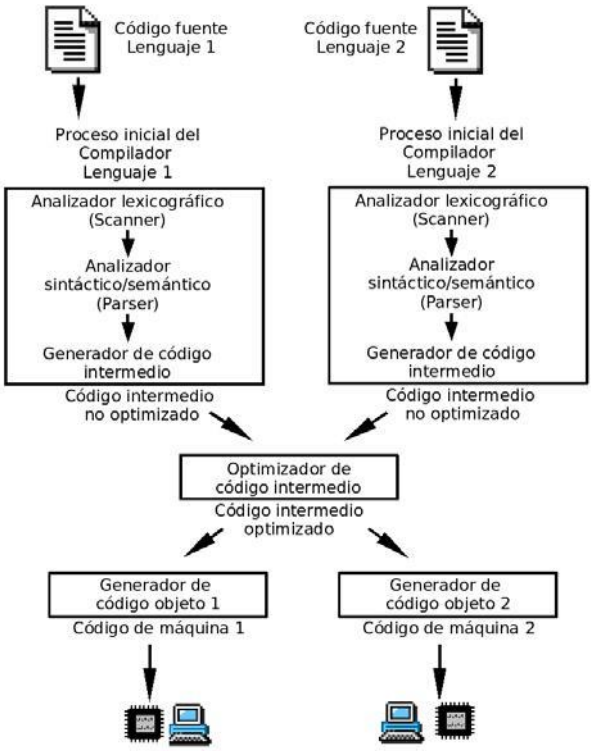
¿Cuáles son sus **ventajas**?

- 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 mostramos la siguiente ilustración:



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**.

5.4. 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 lenguajes interpretados generan programas de menor tamaño que los generados por un compilador, al no guardar el programa traducido a código máquina. Pero presentan el inconveniente de ser algo más lentos, 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.

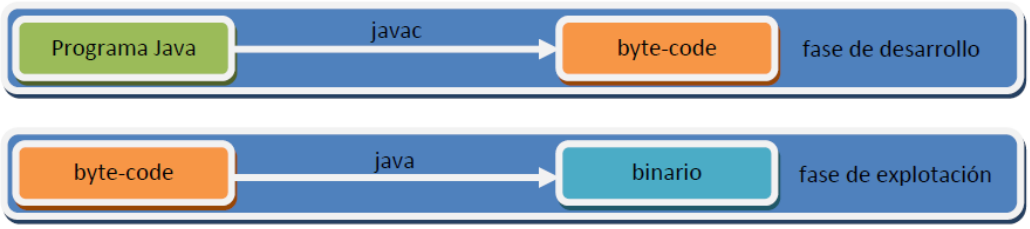
Ejemplos de lenguajes interpretados son: **Perl, PHP, Python, JavaScript**, etc.

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 esquema

Los dos procesos se realizan en fases distintas:



El compilador (javac) sólo tiene que estar en la plataforma de desarrollo, y el intérprete (java) tiene que estar en todos los clientes que quieran ejecutar el applet.

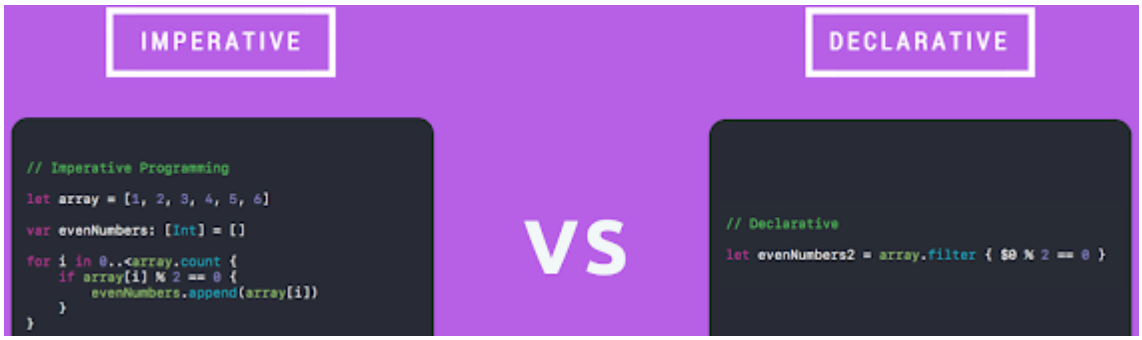
5.5. Clasificación según paradigma

Los programas se pueden clasificar por el paradigma del lenguaje que se use para producirlos. Los dos paradigmas principales son **imperativos** y **declarativos**.

En la programación imperativa, de la cual hacen parte muchos de los principales lenguajes de programación tales como C, C++, Java y PHP, un programa se describe en términos de instrucciones, condiciones y pasos que modifican el estado de un programa al permitir la mutación de variables, todo esto con el objetivo de llegar a un resultado. En contraparte, en la programación declarativa

un programa se describe en términos de proposiciones y afirmaciones que son declaradas para describir el problema, sin especificar los pasos para resolverlo; en este tipo de programas, el estado no puede ser modificado ya que todos los tipos de datos son inmutables. De esta familia hacen parte lenguajes como Scala, Haskell, Erlang y Elixir.

A continuación, se muestra un ejemplo de diferencias entre un código imperativo y otro declarativo:



5.5.1. Lenguajes procedimentales o imperativos

La mayor parte de los lenguajes son de este tipo. Como se ha mencionado, especifican mediante algoritmos los pasos a seguir para la solución de un problema, por lo que son lenguajes controlados por mandatos u orientados (instrucciones).

- **Lenguajes estructurados**

  Este tipo de instrucciones de los lenguajes imperativos todas una tras otra generaban muchos problemas en la comprensión del código en programas grandes, es así, que para solucionar este problema nació la programación estructurada, que es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if, else, switch) e iteración (Bucles for y while). Ejemplos de este tipo de lenguajes serían BASIC, Algol o Pascal, entre otros.

- **Lenguajes procedimentales**

Orientados al uso de procedimientos y funciones que dividen los problemas en subproblemas o subtareas, de manera que resulta más sencillo la reutilización de código y el desarrollo de programas.

- **Lenguajes orientados a objetos**

La programación OO eleva el nivel de abstracción, pero no deja por ello de ser una programación imperativa. Esta se define como: "un método de implementación en el cual los programas son organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases son miembros de una jerarquía de clases unidas vía relaciones de herencia". Así, la programación deja de ser lineal e imperativa, como la programación estructurada. En lugar de intentar ajustar un problema al enfoque procedimental de un lenguaje, POO intenta ajustar el lenguaje al problema. La idea es diseñar formatos de datos que se correspondan con las características esenciales de un problema. Aquí se encuentran los lenguajes más empleados en la actualidad, por lo que está sería la programación dominante a día de hoy. Ejemplos de ellos serían lenguajes como Java, PHP, C++, Python, etc.

- **Lenguajes scripting**

El código fuente de un programa creado usando lenguajes de guiones es almacenado en un archivo de texto plano. Los guiones son casi siempre interpretados, pero no todo programa interpretado es considerado un guion. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso es frecuente que los intérpretes de órdenes sean a la vez intérpretes de este tipo de programas.
Podemos diferenciar varios tipos, los scripts en un sistema operativo, y en diseño web para la parte cliente y servidor.

- En el Sistema Operativo: Conjunto de órdenes guardadas en un archivo de texto, generalmente muy ligero y, que es ejecutado por lotes o línea a línea, en tiempo real por un intérprete. Los scripts son pequeños programas que no son compilados, es decir, por lo general necesitan de un programa lector o interprete que codifique. Algunos ejemplos de lenguajes pueden ser Perl o Python.
- En la WEB: Para la programación web podemos encontrar lenguajes de Scripting. De hecho, los principales frameworks para desarrollo web trabajan con el lenguaje principal de scripting web: JavaScript. Así, los frameworks más importantes (AngularJS, ReactJS, EmberJS o VueJS) emplean este lenguaje de scripts que funciona mediante un único hilo de ejecución. Es también un lenguaje interpretado (no requiere de ser compilado) y orientado a objetos y que se integra con HTML y CSS con el objetivo de crear páginas web. Así, la lógica que pudiera contener la aplicación web residiría en el código implementado mediante JavaScript.

5.5.2. Lenguajes declarativos

Paradigma de programación que está basado en el desarrollo de programas especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan solo se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando).
Dos amplias categorías de lenguajes declarativos son los lenguajes funcionales y los lenguajes lógicos.

- **Lenguajes funcionales**: Se basan en el uso de las funciones matemáticas para producir mejores efectos y que sus resultados sean más eficientes. Tratan de definir el problema que se quiere resolver (el objetivo) y dejar los detalles de la solución al sistema. Lisp sería el lenguaje funcional más antiguo y utilizado.

- **Lenguajes lógicos**: Con estos lenguajes los programas se consideran como una serie de aserciones lógicas. De esta forma, el conocimiento se representa mediante reglas, tratándose de sistemas declarativos. El lenguaje lógico por excelencia es Prolog.

6. OBJETOS DE UN PROGRAMA

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.

6.1. 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.

6.2. 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

6.3. 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:

```
pi * sqr(x)
(2*x)/3
```

- 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
```

6.4. Operadores

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

6.4.1. 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

6.4.2. Aritméticos

Se utilizan para realizar operaciones aritméticas.

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

Ejemplos:

Expresión	Resultado
3 + 5 - 2	6
24 % 3	0
8 * 3 - 7 / 2	21

6.4.3. 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) Y 3 <> 2	Verdadero
3 > 2 Y b < a	Verdadero Y Falso = Falso
3 > 2 O b < a	Verdadero O Falso = Verdadero
no(a < b)	No Verdadero = Falso
5 > 1 Y NO(b < a)	Verdadero Y no Falso = Verdadero

6.4.4. Paréntesis

Anidan expresiones.

Ejemplos:

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

6.4.5. Operador Alfanumérico (+)

Une datos de tipo alfanumérico. También llamado concatenación.

Ejemplos:

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

6.4.6. 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.

7. ENTORNOS INTEGRADOS DE DESARROLLO (IDE)

En los comienzos de cualquier lenguaje, especialmente de los de alto nivel, la utilización de la línea de comandos era algo habitual. El programador escribía el código fuente empleando un editor de texto básico, seguidamente, pasaba a utilizar un compilador y con él obtenía el código compilado. En un paso posterior, necesitaba emplear una tercera herramienta para el ensamblado del programa. Por último, podía probar a través de la línea de comandos el archivo ejecutable.

El problema surgía cuando se producía algún error, lo que provocaba tener que volver a iniciar el proceso completo.
Estas circunstancias hacían que el desarrollo de software no estuviera optimizado. Con el paso del tiempo, se fueron desarrollando aplicaciones que incluían las herramientas necesarias para realizar todo el proceso de programación de forma más sencilla, fiable y rápida. Para cada lenguaje de programación existen múltiples entornos de desarrollo, cada uno con sus ventajas e inconvenientes. Dependiendo de las necesidades de la persona que va a programar, la facilidad de uso o lo agradable que le resulte trabajar con él, se elegirá entre unos u otros entornos.

Para el lenguaje de programación Java existen múltiples alternativas, siendo los principales entornos de desarrollo NetBeans (que cuenta con el apoyo de la empresa Sun), Eclipse y IntelliJ Idea. Los dos primeros son gratuitos, con soporte de idiomas y multiplataforma (Windows, Linux, MacOS).

¿Y cuál será con el que vamos a trabajar? El entorno que hemos seleccionado llevar a cabo nuestros desarrollos de software en este módulo profesional será Eclipse, al ser de código abierto y ofrecer capacidades profesionales. Aunque, no te preocupes, también haremos un recorrido por otros entornos destacables.

7.1. ¿Qué son?

Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa. Podremos realizar las labores de edición, compilación, depuración, detección de errores, corrección y ejecución de programas escritos en Java o en otros lenguajes de programación, bajo un entorno gráfico (no mediante línea de comandos). Junto a las capacidades descritas, cada entorno añade otras que ayudan a realizar el proceso de programación, como, por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.

Hay que tener en cuenta que un entorno de desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. ¿Qué quiere decir eso? Pues que, si tenemos instalado un IDE y no tenemos instalado el compilador, no tenemos nada.

7.2. IDE's actuales

Existen en el mercado multitud de entornos de desarrollo para el lenguaje Java, los hay de libre distribución, de pago, para principiantes, para profesionales, que consumen más recursos, que son más ligeros, más amigables, más complejos que otros, etc.

Entre los que son gratuitos o de libre distribución tenemos:

- NetBeans
- Eclipse
- BlueJ
- Jgrasp
- Jcreator LE

Entre los que son propietarios o de pago tenemos:

- IntelliJ IDEA
- Jbuilder
- Jcreator
- JDeveloper

8. REPRESENTACIÓN

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

8.1. 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 compresión.
3. Normalización en su diseño.

4. Flexibilidad en sus modificaciones.

Debes conocer
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

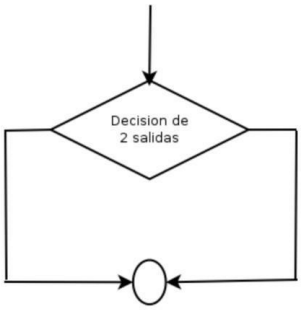
En la representación de ordinogramas es conveniente seguir las siguientes reglas:

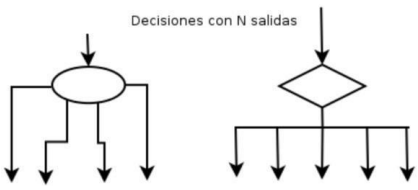
- El comienzo del programa figurará en la parte superior del ordinograma.
- El símbolo de comienzo deberá aparecer una sola vez en el ordinograma.
- El flujo de las operaciones será, siempre que sea posible de arriba a abajo y de izquierda a derecha.
- Se evitarán siempre los cruces de líneas utilizando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

Esta será la representación que utilizaremos durante el curso.

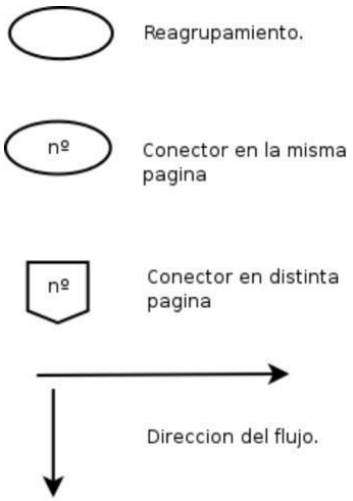
8.1.1. Símbolos de operación

8.1.2. Símbolos de decisión





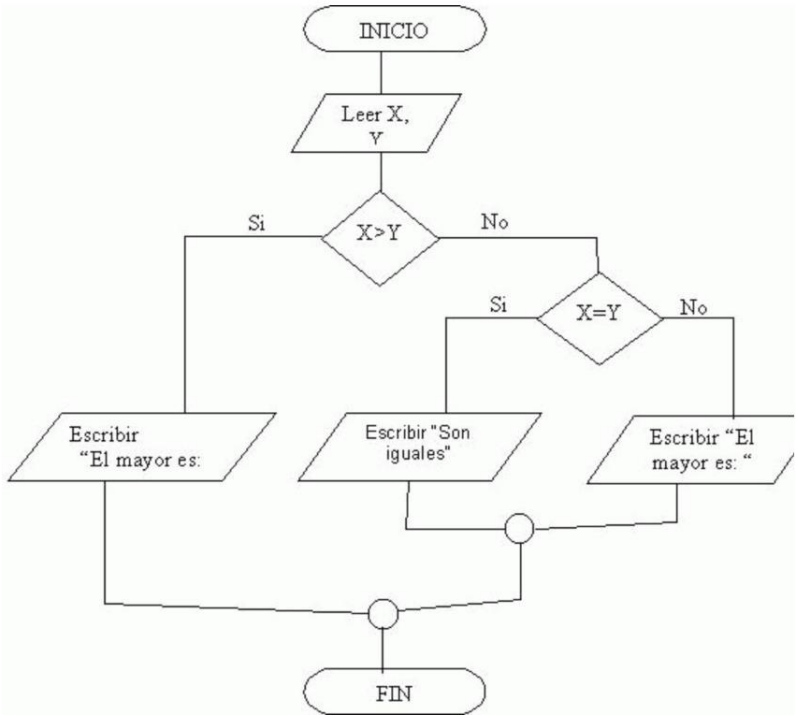
8.1.3. Símbolos de conexión



8.1.4. 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:



8.2. 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:

a) Facilitar la obtención de la solución mediante la utilización del diseño descendente o Top-down.

b) Ser una forma de codificar los programas o algoritmos fácil de aprender y utilizar.

c) 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.

d) Facilitar la traducción del algoritmo a un lenguaje de programación específico.

e) Permitir una gran flexibilidad en el diseño del algoritmo que soluciona el problema, ya que se pueden representar las acciones de una manera más 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

8.2.1. Ejemplo