INICIACIÓN A LA PROGRAMACIÓN I

ÍNDICE

1. ¿Qué es programar?

Programar consiste en **crear** una serie de **órdenes dirigidas a un ordenador o dispositivo** para que lleve a cabo las tareas que le encomendemos, con el objetivo de **solucionar un determinado problema.** La **programación** es el instrumento que **permite la ejecución de las tareas automatizadas de un sistema informático**.. Es pues una herramienta muy poderosa.

Para conseguir esto, se utilizan los **lenguajes de programación**, a través de las cuales se crearán los **programas** que contienen las instrucciones que se dan a la máquina para que ésta ejecute determinada acción.

2. Programas

Un programa **es una secuencia de instrucciones que han sido escritas en un lenguaje de programación concreto, entendibles por el ordenador, y que permiten realizar un trabajo o resolver un problema**. Si no sabemos resolver este problema, no podremos escribir el programa. De ahí que **un buen programador siempre comienza su trabajo con un análisis y compresión exhaustivos del problema que pretende resolver.**

Si el **objetivo de un programa** es la **resolución de un problema**, lo primero que tenemos que considerar es que las formas de resolver un mismo problema no son únicas, de forma que ante un mismo problema un programador podrá escribir **diferentes programas** que solucionen de distinta forma el problema planteado. Pero, si bien lo anterior es evidente, lo que sí podemos afirmar es que, en la codificación de cualquier programa, de forma general, se pueden distinguir las siguientes partes:

- **Entrada de datos**: instrucciones que recogen datos de un dispositivo o periférico para ser almacenados en la memoria principal a la espera de su proceso posterior.

- **Proceso**: parte del programa que engloba las instrucciones encargadas de procesar los datos recogidos anteriormente. Los resultados se almacenan de nuevo en la memoria principal.
- **Salida de resultados**: instrucciones del programa que recogen los resultados obtenidos en la fase anterior y los envían a los dispositivos, periféricos, etc de salida de la información.

3. Los lenguajes de programación

De la misma forma que el término genérico "lenguaje" se refiere a cualquier sistema estructurado de comunicación (por ejemplo, gestos, sonidos, símbolos, ... y las reglas que rigen su uso), podemos **definir lenguaje de programación** como un **conjunto de caracteres y reglas que permiten crear un programa , para introducir y tratar la información en un ordenador**. Son lenguajes artificiales creados con el objetivo de conseguir la comunicación entre el humano y la máquina, o entre máquinas.

3.1 Clasificación de los lenguajes de programación

Los lenguajes de programación **se pueden clasificar atendiendo a diversos criterios**.

3.1.1 Clasificación según el nivel de abstracción

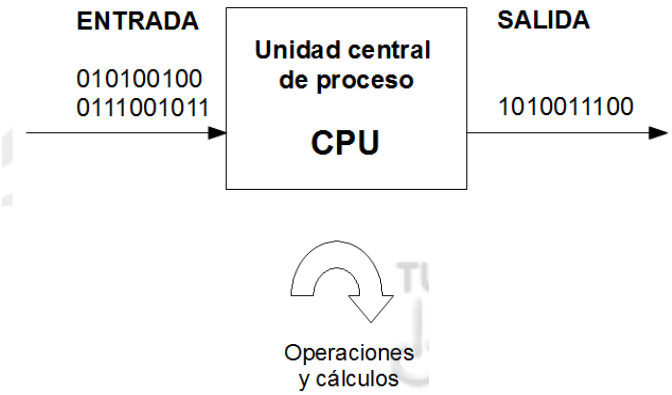
De todos ellos, el criterio que ayuda a una mejor comprensión de lo que en realidad son los lenguajes de programación y para qué se utilizan, es el nivel de abstracción. Según dicho nivel de abstracción, los lenguajes de programación pueden ser:

- **Lenguaje máquina**

```
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
11001010 11001010 11110101 00101011
11001010 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
```

Es el lenguaje de programación que **entiende directamente la computadora o máquina**. Utiliza el **sistema binario**, es decir**, únicamente utiliza dos dígitos, el 0 y el 1.** Con estos dos únicos dígitos, conocidos como **bits,**

forma las cadenas binarias (**combinaciones de ceros y unos**) con las que la CPU de una máquina procesa la información.



Cualquier cosa que queramos que realice el ordenador debe expresarse con este código de ceros y unos, que **es el lenguaje que realmente entiende y utiliza el microprocesador** de una ordenado o dispositivos. **El lenguaje máquina fue el primer lenguaje de programación.**

Los primeros programadores programaban en lenguaje máquina, con números binarios, pero la complejidad de las operaciones realizadas y la facilidad de cometer errores derivó en la necesidad de crear lenguajes de programación más parecidos al lenguaje humano, más sencillos y rápidos de manejar y de entender.

De esta forma, para escribir un programa de forma más sencilla y fácil de recordar, se utilizan los **lenguajes de bajo y alto nivel**. En ellos la sucesión de unos y ceros es sustituida por letras o palabras, mucho más fáciles de recordar y utilizar, por su mayor semejanza con el lenguaje humano, generalmente el inglés.

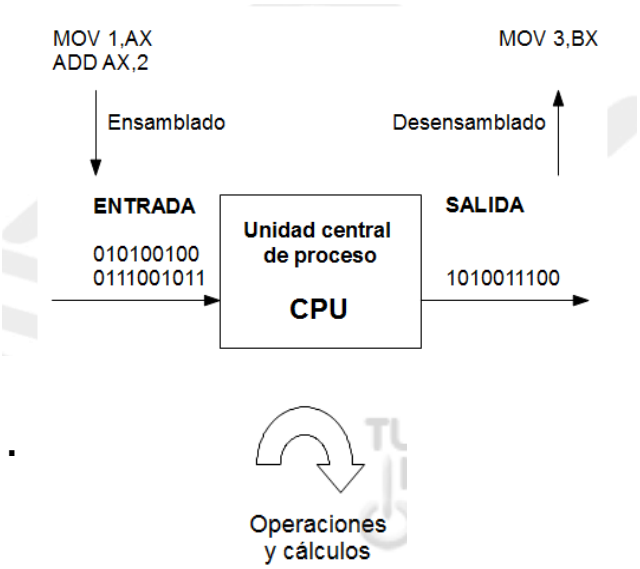
- **<u>Lenguajes de Programación de Bajo Nivel (Lenguaje ensamblador)</u>**

```
INICIO  BSF     STATUS,RPO      ; ENTRA EN EL BANCO 1
        MOVLW   0X1F            ; MUEVE 1Fh A W
        MOVWF   TRISA           ; CONFIGURA EL PUERTO A COMO
ENTRADA
        MOVLW   0X00            ; MUEVE 00 A W
        MOVWF   TRISB           ; CONFIGURA EL PUERTO B COMO SALIDA
        BCF     STATUS,RPO      ; SALE DEL BANCO 1
        CLRF    PORTB           ; LIMPIA EL PUERTO B

OTRA    BTFSC   PORTA,4         ; PREGUNTA SI EL PIN 4 DEL PUERTO A I
CERO
        GOTO    OTRA            ; SALTA A LA ETIQUETA OTRA
        MOVLW   0X18            ; MUEVE 18h A W
        MOVWF   PORTB           ; CARGA W EN EL PUERTO B
        CALL    RETARDO         ; LLAMA A LA ETIQUETA RETARDO
        BTFSC   PORTA,4         ; PREGUNTA SI EL PIN 4 DEL PUERTO A I
CERO
        GOTO    OTRA1           ; SALTA A LA ETIQUETA OTRA1

        MOVLW   0X24            ; MUEVE 24h A W
        MOVWF   PORTB           ; CARGA W EN EL PUERTO B
        CALL    RETARDO         ; LLAMA A LA ETIQUETA RETARDO
```

A este grupo pertenecen **los primeros lenguajes creados intentando sustituir el lenguaje máquina por uno más similar utilizado por el hombre.** Se denominan frecuentemente **lenguajes ensambladores.**

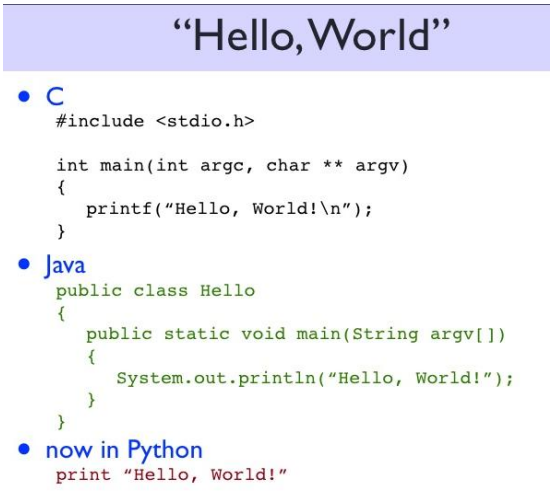
Son mucho más fáciles de utilizar que el lenguaje máquina, pero **son específicos de cada procesador,** lo que obliga a reescribir los programas hechos con este tipo de lenguaje si nos llevamos el programa a otro computador **(poca portabilidad),** siendo éste su principal inconveniente**.**

Estos lenguajes están constituidos básicamente por las instrucciones del lenguaje máquina escritas en forma simbólica, mediante códigos nemotécnicos (los mnemónicos son grupos de 3-4 caracteres alfanuméricos que simbolizan las órdenes o tareas a realizar)., fáciles de leer y de recordar(ejemplo: ADD(sumar), MOV (mover), ...). Un programa escrito en este lenguaje no es directamente ejecutable en la máquina. De esta forma, debido a su simbolismo, debe de ser traducido a un programa equivalente, escrito en lenguaje de máquina, capaz de ser ejecutado. Esta traducción se realiza mediante programas denominados programas ensambladores.



En la actualidad se suele usar en ambientes reducidos, académicos y de investigación, cuando se va a manipular hardware y se pretende conseguir un uso de recursos muy controlado, o cuando se pretende conseguir unos altos rendimientos. En los ensambladores se necesitan muchas instrucciones para tareas simples y su utilización exige grandes conocimientos sobre el hardware de la máquina, por lo que **programar en ensamblador es realmente complejo.** Muchos dispositivos programables (como los microcontroladores) aún cuentan con el ensamblador como la única manera de ser manipulados.

- **Lenguajes de Programación de Alto Nivel**



Los lenguajes de alto nivel son **más fáciles de aprender y utilizar porque son los que más se parecen al lenguaje natural, ya que se usan palabras o comandos del lenguaje humano** (generalmente inglés). Por ello son los más utilizados por los programadores.

Además los lenguajes de alto nivel son **independientes de la máquina,** por lo que se pueden usar en cualquier computador con muy pocas modificaciones o sin ellas.

Sirven fundamentalmente para crear programas informáticos que solucionan diferentes problemas.

Existen muchos lenguajes de programación de alto nivel con sus diferentes versiones y lenguajes derivados. Entre ellos tenemos el **HTML, , C, C++, SQL, JAVA, FORTRAN, Python, Pascal, LOGO, Processing, ...**

El primer lenguaje de programación de alto nivel que fue creado es **FORTRAN** que significa Traductor de Formulas y proviene de las palabras en inglés **FOR**mula **TRAN**slating.

Fortran fue desarrollado por un equipo de investigadores de IBM en la década de 1950, que buscaban un lenguaje de programación que tuviera apariencia al lenguaje humano.

Fortran se sigue utilizando hoy en día para la programación de aplicaciones científicas y matemáticas

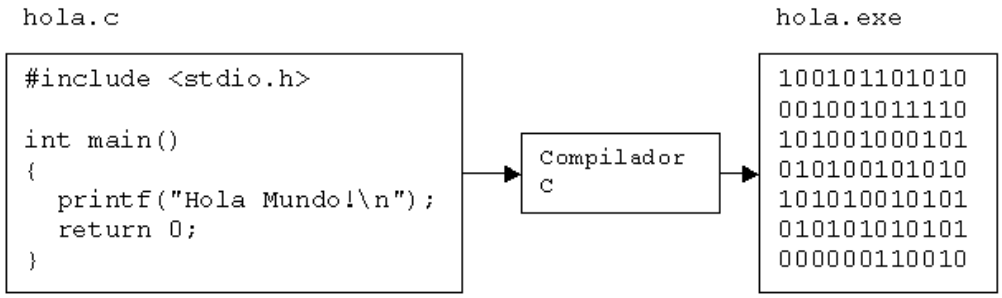
Los **lenguajes de alto nivel necesitan** un **programa compilador o un programa intérprete que lo traduzca a lenguaje máquina** para que la computadora pueda entenderlo. Estos programas de traducción pueden ser: Compiladores o Intérpretes.

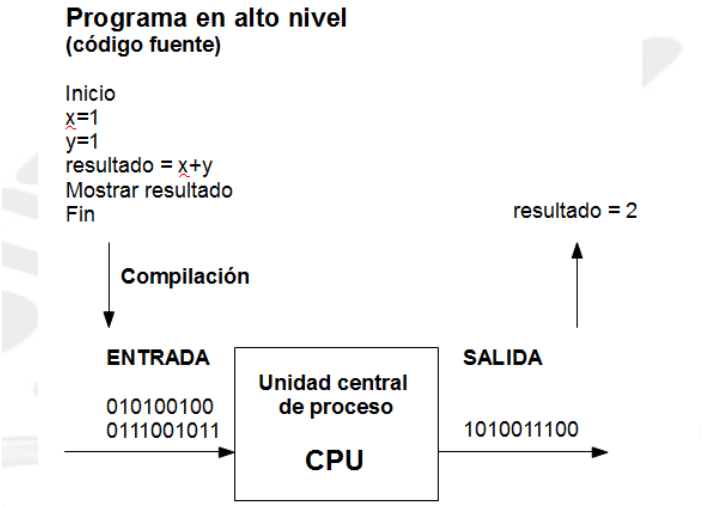
Compiladores e intérpretes

Tanto el **compilador** como el **intérprete** son **programas** cuya **finalidad es "traducir" las instrucciones** enviadas a una máquina **en un lenguaje de alto nivel a un lenguaje que el equipo pueda entender.**

– *Compiladores*.

Son programas especiales que procesan las instrucciones escritas en un lenguaje de programación de alto nivel y las convierte a lenguaje de máquina. El compilador **traduce el programa completo a código máquina antes de empezar a ejecutarlo**, a lo que se llama código objeto, y si no hay errores, genera un archivo ejecutable. Gráficamente:





Los compiladores son programas muy grandes que permiten la comprobación de errores. En realidad algunos compiladores pueden traducir lenguaje de alto nivel a un lenguaje ensamblador intermedio, que luego se traduce a código de máquina por un programa ensamblador. Pro la mayoría de los compiladores generan código de máquina directamente.

Por todo lo anterior**, todo lenguaje de alto nivel tiene un compilador incorporado**. Básicamente, el compilador es la parte central de dicho lenguaje, porque define y traduce las instrucciones recibidas a código máquina.

Una vez compilado el programa, podemos ejecutarlo, es decir, hacer que se inicie la carga del programa y comiencen a realizarse las acciones programadas.. Ejemplos de lenguajes compilados son Pascal o C++.

Las ventajas del utilizar un compilador son:

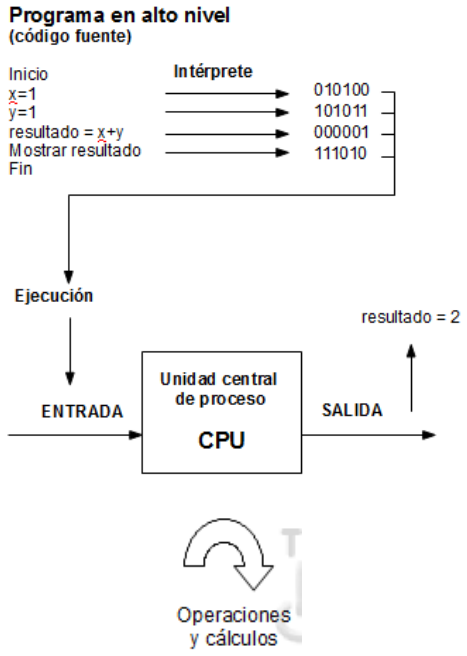
- El programa ejecutable no incluye el código fuente, por lo tanto, el código compilado es más seguro.
- Un programa compilado se ejecutan más rápidamente que un programa que está siendo interpretado ya que la traducción se realiza directamente a código máquina.

Las desventajas de utilizar un compilador son:

- El código objeto necesita crearse para obtener el archivo ejecutable final, y si el programa es muy largo, el proceso de compilación puede ser muy lento.
- El código fuente debe estar libre de errores para que el archivo ejecutable se genere.
- El archivo ejecutable generado por el compilador solo funcionará en el sistema operativo en el que fue creado.

– ***Intérpretes***.

Un **intérprete es un programa que**, en tiempo de ejecución, **transcribe a lenguaje máquina y ejecuta una por una**, en la secuencia descrita en el programa fuente, **el conjunto de instrucciones** del mismo (sin generar un programa objeto). De esta forma, cada vez que se ejecuta el programa fuente, éste deberá ser interpretado de nuevo; cada vez que se ejecuta una sentencia ésta debe ser interpretada de nuevo.



Es decir, **el programa intérprete traduce y ejecuta las instrucciones del programa línea a línea,** siguiendo la secuencia real del programa. Cuando detecta un error, la traducción y ejecución del programa se detienen, e indica un código de error. La ejecución del programa en un intérprete es por lo general, más lenta que en un compilador. El intérprete no genera un archivo ejecutable. La ventaja de ello es que cualquier programa puede ser interpretado en cualquier sistema operativo.

Ejemplos de lenguajes interpretados son las primeras versiones de BASIC o Scratch.

Las ventajas del utilizar un intérprete son:

- Al ejecutarse línea a línea, resulta ser más fácil de depurar y comprobar errores.
- Si el programa es muy largo, un intérprete resulta es más rápido que un compilador porque no tiene que traducir todo el programa para ejecutarlo.

Las desventajas de utilizar un intérprete son:

- El código fuente se requiere para poder ejecutar un programa
- Los intérpretes son más lentos que los programas compilados o ejecutables.

De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a como piensa un ser humano (lenguaje de alto nivel), para luego *compilarlo o interpretarlo* hasta un programa más manejable por una computadora. (lenguaje máquina).

3.1.2 Clasificación según el propósito de creación

Atendiendo al propósito para el que se crean, los lenguajes de programación pueden ser:

- **Lenguajes de Propósito General:** permiten la implementación de prácticamente cualquier algoritmo, el nivel de abstracción es más uniforme, proporciona razonable rendimiento. Ej Pascal, C, C++, Java, Delphi, Lisp, Scheme.

- **Lenguajes de Propósito Específico:** tienen por lo general un conjunto muy restringido de características y un alto nivel de abstracción para cumplir tareas especificas como el procesamiento de textos, gráficos, audio, video e ingeniería. Ejemplos: Snobol, SQL, Matlab.

3.1.3 Clasificación según su paradigma de programación

Otro posible **criterio de clasificación** de los lenguajes de programación es el hacerlo según su **paradigma* de programación.**

* El término “paradigma” ha sido objeto de muchas interpretaciones. En su origen griego, significaba “modelo”, “ejemplo” o “patrón”. Sobre este punto de partida, podemos hablar de un paradigma como un **conjunto de creencias, prácticas y conocimientos que guían el desarrollo de una disciplina durante un período de tiempo**. En diversas ramas de la ciencia, un conjunto de ideas en vigencia puede ser reemplazado drásticamente por otro que entre en conflicto con él y se demuestre más acertado. La programación tiene sus propios paradigmas, pero el término “paradigma de programación” no necesariamente representa un modelo único que deba ser respetado hasta que aparezca otro mejor.

Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis. Según su paradigma, los lenguajes de programación pueden ser:

- **Lenguajes Imperativos.**

Se llama lenguajes imperativos a aquellos en los cuales se le ordena a la computadora cómo realizar una tarea siguiendo una serie de pasos o instrucciones, por ejemplo:

Paso 1, solicitar número.

Paso 2, multiplicar número por dos.

Paso 3, imprimir resultado de la operación. ... etc,

El proceso anterior se puede realizar con un lenguaje imperativo como por ejemplo BASIC, C, C++, Java, Clipper, Dbase, C#, PHP, Perl, etc.

Dentro de la programación imperativa, se tiene un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

- **Lenguajes Declarativos.**

Son aquellos lenguajes de programación en los cuales se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando, sin especificar los pasos a seguir para conseguirlo.

Por ejemplo: "Obtener los nombres de todos los empleados que tengan más de 32 años" . Eso se puede lograr con un lenguaje declarativo como SQL.

Se expresa lo que el programa debe lograr sin prescribir cómo hacerlo. Ejemplos: SQL, HTML, RPG

- **Lenguajes Funcionales.** Estos programas se basan en la utilización de funciones matemáticas predefinidas, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida. Ejemplos: Lisp, Scheme, Common Lisp, ML, CAML.

- **Lenguajes de Lógicos.**

Expresan tareas utilizando la lógica matemática como lenguaje de programación.

Ejemplo: Prolog.

- **Lenguajes Orientados a Objetos.** Crean un sistema de clases y objetos siguiendo el esquema del mundo real para definir los objetos, acciones y forma de comunican entre objetos.

Un **objeto es una entidad provista de un conjunto de propiedades o atributos** ("datos") y de comportamiento o funcionalidad ("métodos"). Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa). **Una clase** es una **colección de objetos similares, con unas propiedades y comportamiento comunes** que les hacen pertenecer a esa clase y no a otra.

Ejemplos: C++, Java, Python, Visual Basic.

4. Código fuente

El **código fuente** (source code) consiste en **uno o más archivos que contienen las instrucciones de programación con las cuales un desarrollador de software ha creado determinado programa o aplicación.**

.

Cuando un desarrollador de software escribe un programa utilizando algún tipo de lenguaje de programación (C, C++, JAVA, Python, etc.). El archivo o archivos que son escritos forman lo que se denomina programa origen o código fuente. Para ejecutar el programa, el código fuente debe

ser traducido a lenguaje de máquina, para que el computador lo pueda entender y así ejecutar la aplicación de acuerdo a las instrucciones recibidas.

Acceder al código fuente concreto de un programa significa acceder a todo lo creado, o, como se suele decir, "**abrir el programa**".

Dentro del argot, se suele decir también que se va a "**liberar**" el código cuando se va a compartir el programa en forma de texto, para que cualquier persona lo analice, modifique, lo comparta o lo copie. Cuando se libera un programa este queda expuesto, por lo que queda en estado de inestabilidad.

Por regla general, el **código fuente** no se libera en los programas comerciales. Al comprar un sistema operativo o alguna aplicación de uso general como Microsoft Office, por lo general recibes una aplicación en forma de código objeto compilado, que solo permite ser ejecutado, no puede ser modificado porque el código fuente no está incluido.

5. Proceso de solución de problemas. Algoritmos

5.1 Concepto de algoritmo

Cuando queremos resolver un problema a través de la creación de un programa informático habremos de diseñar previamente lo que se denomina un algoritmo.

Se llama **algoritmo** a la **secuencia de pasos organizados a seguir para resolver un problema**. Cualquier algoritmo debe tener las siguientes características:

- La descripción de cada paso no debe llevar a ambigüedades, los pasos son absolutamente explícitos y no inducen a error.
- El número de pasos debe ser finito, de forma que el algoritmo se pueda ejecutar en un tiempo finito.

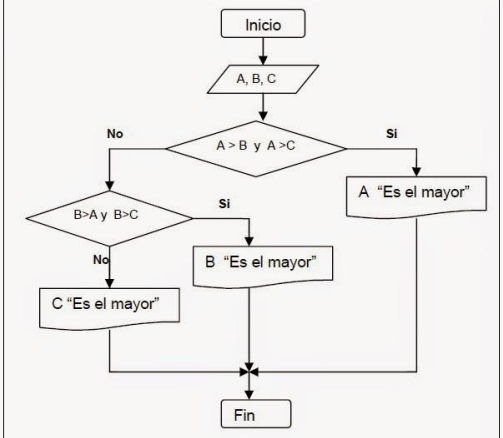
Los algoritmos son independientes de la sintaxis de cada lenguaje de programación en particular, siendo evidente que el algoritmo que lleve a la solución de un determinado problema puede ser expresado utilizando distintos lenguajes de programación.

5.2 Formas de representación de un algoritmo: pseudocódigos y diagramas de flujo

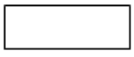
Hay **distintas formas de escribir un algoritmo**, bien usando un lenguaje específico de descripción de algoritmos (**pseudocódigo)**, bien mediante representaciones gráficas **(organigramas o diagramas de flujo**).

Veamos un ejemplo de diseño de un algoritmo, expresado de ambas formas:

Desarrolle un algoritmo que permita leer tres valores y almacenarlos en las variables A, B y C respectivamente. El algoritmo debe imprimir cual es el mayor y cual es el menor. Recuerde constatar que los tres valores introducidos por el teclado sean valores distintos. Presente un mensaje de alerta en caso de que se detecte la introducción de valores iguales.

Pseudocódigo	Diagrama de Flujo
1. **Inicio** 2. Inicializar las variables A, B y C 3. **Leer** los tres valores 4. Almacenar en las variables A, B y C 5. **Si** A > B y A > C **Entonces** 6. **Escribir** A "Es el mayor" 7. **Sino** 8. **Si** B > A y B > C **Entonces** 9. **Escribir** B "Es el mayor" 10. **Sino** 11. **Escribir** C "Es el mayor" 12. **Fin_Si** 13. **Fin_Si** 14. **Fin**	

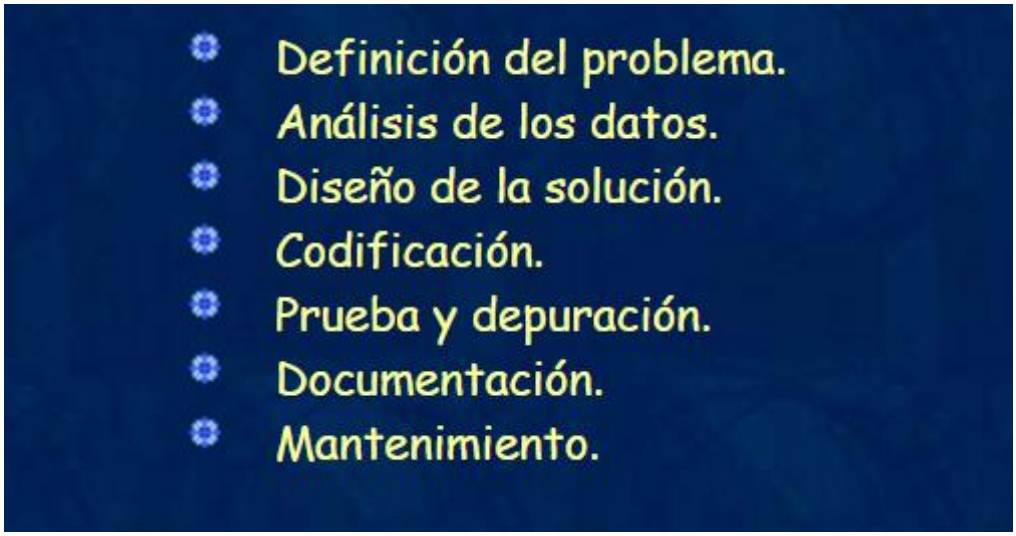
En los diagramas de flujo existen símbolos estandarizados que permiten reflejar distintos tipos de pasos en un algoritmo.

SIMBOLOGIA USADA EN LSO DIAGRAMAS DE FLUJO	
Símbolo	Significado
	**Inicio/Fin.** Determina el inicio y el fin de los algoritmos.
	**Entrada por teclado.** Representa el ingreso de los datos al programa.
	**Proceso.** Representa las operaciones que se efectúan para obtener el resultado.
	**Decisión.** Representa las operaciones de tipo lógico que contenga el algoritmo.
	**Salida a impresora.** Se utiliza cuando solamente se va a mostrar el resultado en pantalla.
	**Salida por pantalla.** Se utiliza cuando solamente se va a mostrar el resultado en pantalla.
	**Conector.** Se utiliza para conectar bloques del diagrama cuando el diagrama es grande y es necesario dividirlo.
	**Líneas de flujo.** Indican la secuencia del flujo de las operaciones del diagrama.

Independiente del lenguaje de programación que vaya a usarse, un algoritmo que esté escrito en **pseudocódigo** o **con un diagrama de flujo es fácilmente traducible a muchos lenguajes de programación**. El pseudocódigo es una forma de escribir los pasos, pero de la forma más cercana al lenguaje de programación que vamos a utilizar, es como un falso lenguaje muy cercano al lenguaje humano.

5.3 Fases del proceso de resolución de problemas mediante la creación de un programa .

La creación de un programa informático es un proceso ordenado que ha de realizarse de modo secuencial. Desde el programa más sencillo, con pocas instrucciones, hasta la programación de grandes aplicaciones que contienen miles de líneas de instrucciones, se han de seguir una serie de pasos, que constituyen las ETAPAS DE PROGRAMACION.



- ✓ **Definición del problema**

El enunciado del problema, el cual debe ser claro y complejo. Es importante que conozcamos exactamente "**que se desea obtener al final del proceso**" ; mientras esto no se comprenda no puede pasarse a la siguiente etapa.

- ✓ **Análisis de los datos**

Para poder definir con precisión el problema se requiere que las **especificaciones de entrada y salida** sean descritas con detalle ya que esto es un requisito para lograr una solución eficaz.

Una vez que el problema ha sido definido y comprendido, deben analizarse los siguientes aspectos :

- Los resultados esperados.

- Los datos de entrada disponibles.

- Herramientas a nuestro alcance para manipular los datos y alcanzar un resultado (fórmulas, tablas, accesorios diversos).

Ejemplo: *Diseñar un programa informático que permita, conocido el radio de una circunferencia, calcular e imprimir su superficie y su longitud de circunferencia*.

ANALISIS: Como entrada tendremos el radio del círculo, que será un número real.

Las salidas serán dos variables : superficie y circunferencia que también serán de tipo real. Es decir:

Entradas : Radio del círculo (variable RADIO).

Salidas : Superficie del círculo (variable AREA).

Circunferencia del círculo (variable CIRCUNFERENCIA).

Variables : RADIO, AREA, CIRCUNFERENCIA tipo real.

- ✓ **Diseño del algoritmo (solución)**

Esta fase se refiere a dar una solución al problema planteado, para lo que **se diseña el algoritmo correspondiente.** En caso de obtenerse varios algoritmos que den solución al mismo problema se selecciona uno de ellos, utilizando criterios ya conocidos.

Esta etapa incluye la descripción del algoritmo resultante en **pseudocódigo o en forma de diagrama de flujo.**

- ✓ **Codificación del programa (obtención del código fuente y del código objeto)**

Se refiere a la **obtención de un programa definitivo que pueda ser comprensible para la máquina**. Incluye una primera etapa **de escritura del programa en lenguaje de alto nivel** (código fuente), **y** una segunda que se reconoce como **compilación,** donde el lenguaje de alto nivel se traduce a **lenguaje máquina**, obteniéndose así el programa en código objeto.

- ✓ **Prueba y depuración**

Una vez que se ha obtenido el programa ejecutable, este es sometido a prueba a fin de determinar si resuelve o no el problema planteado en forma satisfactoria.

Las pruebas que se le aplican son de diversa índole y generalmente dependen del tipo de problema que se está resolviendo. Comúnmente se inicia la prueba de un programa introduciendo datos válidos, inválidos e incongruentes y observando cómo reacciona en cada ocasión.

El proceso de depuración consiste en localizar los errores y corregirlos en caso de que estos existan. Si no existen errores, puede entenderse la depuración como una etapa de refinamiento en la que se ajustan detalles para optimizar el desempeño del programa

- ✓ **Documentación**

Es la guía o manual escrito de que indica los pasos a seguir en la utilización del programa.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones.

- ✓ **Mantenimiento**

Se refiere a las actualizaciones que deban aplicarse al programa cuando las circunstancias así lo requieran. Este programa deberá ser susceptible de ser modificado para adecuarlo a nuevas condiciones de operación.

Cualquier actualización o cambio en el programa deberá reflejarse en su documentación.