

II. SOFTWARE

5. Fundamentos de programación

ULL

Universidad
de La Laguna

Curso de Acceso a la
Universidad para Mayores
de 25 y 45 (CAM 25-45)



Introducción a la Informática

Coromoto León Hernández
Gara Miranda Valladares
Carlos Segura González

Curso 2012/2013

Contenido

| | |
|----------------------------------------------------------|----|
| 1. Algoritmos..... | 2 |
| 1.1. Componentes de un algoritmo..... | 3 |
| 1.1.1. Variable..... | 4 |
| 1.1.2. Constante..... | 4 |
| 1.1.3. Expresión..... | 4 |
| 1.1.4. Sentencias..... | 4 |
| 1.2. Métodos de representación de algoritmos..... | 5 |
| 1.3. Fases del proceso de resolución de un problema..... | 8 |
| 1.4. ¿Qué es un programa? | 8 |
| 2. Lenguajes de programación..... | 9 |
| 3. Tipos de lenguajes de programación..... | 11 |
| 3.1. Nivel de abstracción | 11 |
| 3.1.1. Lenguajes de bajo nivel..... | 11 |
| 3.1.2. Lenguajes de alto nivel..... | 11 |
| 3.2. Forma de ejecución | 12 |
| 3.2.1. Lenguajes compilados..... | 12 |
| 3.2.2. Lenguajes interpretados | 12 |
| 3.3. Paradigmas de programación..... | 13 |
| 3.3.1. Imperativo..... | 13 |
| 3.3.2. Declarativo | 13 |
| 3.3.3. Orientado a Objetos..... | 14 |
| 4. Traductores, compiladores e intérpretes..... | 14 |
| 5. Bibliografía y enlaces de interés..... | 15 |

1. Algoritmos.

En el Diccionario de la Real Academia Española se define [algoritmo](#) como:

1. *Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.*
2. *Método y notación en las distintas formas del cálculo.*

Teniendo en cuenta estas definiciones, podríamos decir que un algoritmo es una sucesión de pasos que se deben realizar para resolver un determinado problema. Partiendo de esta definición podemos concluir que el concepto no es sólo informático, sino que es un concepto mucho más amplio, pues se trata de un método de resolución de problemas general. Quizás no seamos conscientes de ello pero en la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas comunes:

- Manuales de usuario o instrucciones de uso que muestran los algoritmos para usar un determinado aparato, dispositivo o software:
 - Instrucciones de uso de una televisión.
 - Instrucciones de uso de un microondas.
- Instrucciones o pasos para la realización de tareas comunes:
 - Pasos necesarios para hacer una tarta de galletas y chocolate.
 - Pasos necesarios para hacer una cama.
 - Pasos necesarios para hacer el nudo de una corbata.
- Procedimientos matemáticos:
 - Algoritmo de la división para calcular el cociente de dos números.
 - Algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos

El algoritmo nos da la solución genérica a un problema y lo podremos emplear todas las veces que se nos presente ese mismo problema. Por ejemplo, el algoritmo de la división es genérico e independiente de los números que tengamos que dividir. Por otro lado, una vez descubierto un algoritmo para efectuar una tarea, la realización de ésta ya no requiere entender los principios en que se basa el algoritmo, pues el proceso se reduce a seguir las instrucciones o los pasos determinados por el mismo. Esto implica, por ejemplo, que podemos hacer una división siguiendo el algoritmo sin entender por qué funciona el procedimiento.

Gracias a la capacidad para comunicar nuestros pensamientos mediante algoritmos, podemos construir máquinas cuyo comportamiento simula inteligencia. En cualquier caso, el nivel de inteligencia que simula la máquina estará limitado por la inteligencia que podamos comunicarle por medio de algoritmos. Por lo tanto, si queremos que un ordenador efectúe una tarea determinada, lo primero que debemos hacer es definir un algoritmo para llevar a cabo la tarea. Lo siguiente que deberíamos hacer es *programar* el algoritmo en la máquina. Esto es, se debe transformar el algoritmo (pasos o indicaciones conceptuales) en un conjunto de instrucciones entendibles por la máquina. Para ello, se programan las instrucciones en un lenguaje de programación, entendible por la máquina y sin introducir ningún tipo de ambigüedad.

Los principales atributos de un algoritmo son que ha de ser finito, definido, preciso e independiente de lenguaje de programación. Que sea finito significa que debe tener un número finito de pasos así su tiempo de realización esta limitado y también el número de operaciones que realiza. Ser definido implica que ante los mismos datos de entrada se obtienen los mismos datos de salida. La precisión de un algoritmo se refiere a que es necesario que se indique de una forma inequívoca el orden de realización de cada paso. Finalmente, la independencia del lenguaje de programación significa que debe ser de propósito general, y no contener ningún paso que dependa de un lenguaje particular de manera que permita su implementación en cualquier lenguaje de programación, y por tanto, en cualquier tipo de máquina algorítmica. El atributo de independencia es el que marca la diferencia entre lo que es un algoritmo y lo que es programa de ordenador. Un algoritmo siempre ha de existir antes que el programa de ordenador. Es decir, un algoritmo se puede codificar mediante diversos lenguajes de programación, dando lugar a diferentes programas de ordenador.

1.1. Componentes de un algoritmo.

Un algoritmo manipula la información que se le suministra para generar resultados. Los elementos que componen un algoritmo son: **datos** y **sentencias** para operar sobre los datos. Los datos se almacenan como *variables* o *constantes* y se involucran en *expresiones*. Las sentencias describen las acciones que pueden ser ejecutadas y en general realizan *asignaciones*, *entradas/salidas* de datos y *control del flujo* del algoritmo. A continuación se definen cada uno de estos términos.

1.1.1. Variable.

Elemento del algoritmo que posee un valor y que es conocido unívocamente por un determinado nombre o identificador. Las variables llevan asociadas un determinado tipo de datos, generalmente definido al comienzo del algoritmo, y que especifica el tipo de valores que pueden ser almacenados en la variable. Algunos ejemplos de tipos de datos son:

- Números enteros. Por ejemplo, la edad de una persona.
- Números reales. Por ejemplo, el saldo de una cuenta corriente.
- Cadenas de caracteres. Por ejemplo, el nombre o apellidos de una persona.
- Booleano. Sólo puede tener dos posibles valores, *verdadero* o *falso*. Por ejemplo, para cada uno de los alumnos de una asignatura podríamos querer almacenar la información de si son repetidores o no, con lo cual podríamos tener una variable denominada "repetidor", cuyos posibles valores fueran *verdadero* o *falso*.

1.1.2. Constante.

Elemento del algoritmo similar a una variable, pero cuyo valor no cambia a lo largo del algoritmo. Por ejemplo, supongamos que queremos hacer un algoritmo para calcular el área A de un círculo. Sabiendo que el área A viene dado por la expresión $A = \pi \cdot r^2$, donde r es el radio del círculo, vemos que el radio es una variable, cuyo valor dependerá del círculo para el cual estemos calculando el área. Sin embargo, para cualquiera de los círculos que definamos, π tendrá el mismo valor (3.14159...) y, por tanto, se dice que es una constante.

1.1.3. Expresión.

Combinación de variables, constantes, valores constantes, operadores y funciones que al evaluarla en el orden correcto tiene un valor concreto. Por ejemplo, la fórmula anterior para el cálculo del área de un círculo es una expresión.

1.1.4. Sentencias.

Son los elementos que describen lo que debe hacer el algoritmo. Podemos decir que existen tres tipos diferentes de sentencias:

- *Sentencias de asignación*: las sentencias de asignación almacenan un valor en una variable o constante, la operación de asignación se muestra en los algoritmos con el símbolo ' \leftarrow ' o con el símbolo '=', denotando que el valor situado a su derecha se almacena en la variable o constante situada a su izquierda. Esta operación es destructiva, es decir, implica un movimiento de datos hacia el elemento situado a la izquierda, perdiéndose el valor que pudiera tener asignado anteriormente ese elemento.

- *Sentencias de entrada/salida*: la operación de lectura o de entrada permite introducir los datos desde dispositivos externos (teclado) o desde ficheros externos (unidad de disco). Este método es el más adecuado si se pretende realizar un programa que permita, cuando se codifique el algoritmo, manipular diferentes datos cada vez que se ejecute el programa. Por otro lado, la sentencia de salida o de escritura, permite que los resultados que se obtienen de un algoritmo puedan visualizarse mediante algún dispositivo externo (impresora, pantalla) o puedan almacenarse en algún fichero externo. o en un puerto de E/S (tarjeta de adquisición de datos).
- *Sentencias de control del flujo*: todo algoritmo puede ser escrito usando tres estructuras de control básicas: secuenciales, selectivas y repetitivas o cíclicas. Las estructuras de control secuenciales son aquellas en las que todas las instrucciones o sentencias se ejecutan una después de la otra. Empezando por el principio o inicio del algoritmo y terminando por su final. Las estructuras de control selectivas, también denominadas de decisión o alternativas, se utilizan para tomar decisiones lógicas. En este tipo de estructuras se evalúa una expresión lógica o relacional, y en función de su resultado se selecciona cuál de las posibles opciones se toma. Esta estructura permite codificar bifurcaciones en el cuerpo del algoritmo así como salidas múltiples, según los datos que tengamos en cada caso. Por último, las estructuras de control repetitivas, también llamadas cíclicas, bucles o lazos, se utilizan para realizar varias veces el mismo conjunto de operaciones. Entre ellas se encuentran aquellas en las que el número de repeticiones depende de una condición lógica o relacional (bucles controlados por bandera) y las que se manejan mediante una variable contador. Es importante notar que en un mismo algoritmo pueden aparecer las tres estructuras (secuenciales, selectivas y/o repetitivas) combinadas de distintas formas y en distintas partes del mismo.

1.2. Métodos de representación de algoritmos.

Antes de codificar un algoritmo, una práctica recomendable, y en muchas ocasiones imprescindible, es representar de un modo sencillo las operaciones que debe realizar. Existen dos tipos de métodos para representar algoritmos:

- Los **métodos informales**, cuyo ejemplo más claro es el lenguaje natural, tratan de describir un algoritmo como si se estuviese contando una historia. Tiene la ventaja de que es un método muy intuitivo y de fácil comprensión por cualquiera. Sin embargo, tiene la gran desventaja de que el lenguaje natural es poco

adecuado para la descripción de operaciones matemáticas, siendo una descripción imprecisa que no garantiza la correcta definición del algoritmo, pues puede incurrir en ambigüedades.

- Los **métodos formales** disponen de la sintaxis y elementos gramaticales adecuados para describir la implementación del algoritmo de resolución de un problema en un ordenador. Existen dos tipos de métodos formales, una representación textual (pseudocódigo) o una representación gráfica (diagramas de flujo).

A modo de ejemplo, podríamos analizar la forma en la que expresaríamos un algoritmo para determinar si una persona es mayor de edad, o no:

1. Si utilizáramos un método informal podríamos expresar el algoritmo de la forma siguiente:

En primer lugar, tendríamos que dirigirnos a la persona y preguntarle qué edad tiene. Cuando la persona nos responda, si la edad que nos ha dicho es 18 o un número mayor a 18 entonces sabremos que la persona es mayor de edad, pero si la edad que nos ha dicho es un número menor que 18 entonces sabremos que la persona no es mayor de edad.

2. Si utilizáramos un método formal con representación textual (pseudocódigo) podríamos expresar el algoritmo de la forma siguiente:

Algoritmo MayoríaEdad

Variables

enteras: *edad*

Inicio

Escribir (“¿Cuál es tu edad?”)

Leer (*edad*)

Si *edad* \geq 18 entonces

 Escribir (“Eres mayor de edad”)

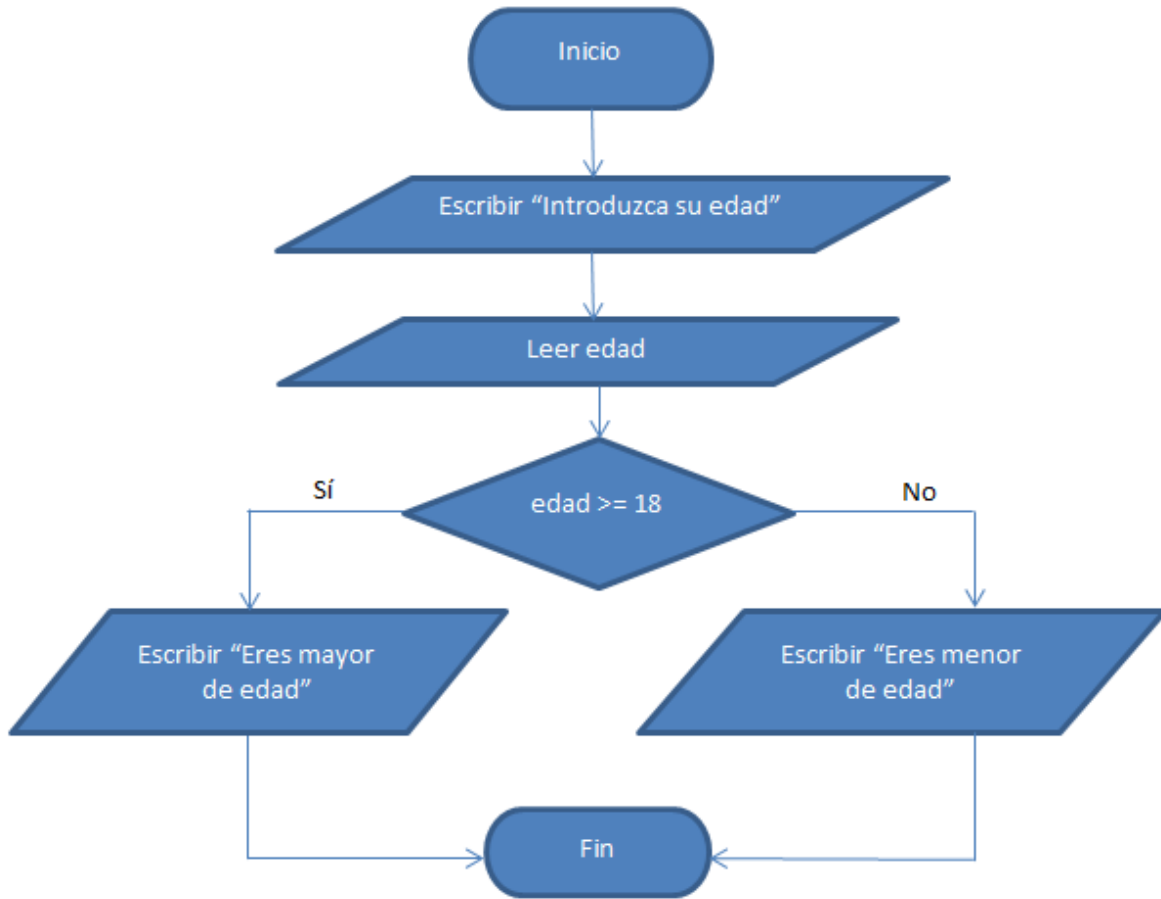
En otro caso

 Escribir (“No eres mayor de edad”)

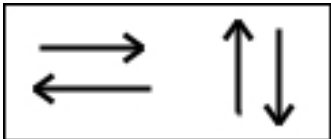

Fin-Si


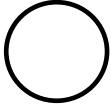

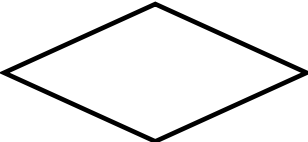
Fin

3. Si utilizáramos un método formal con representación gráfica (diagramas de flujo) podríamos expresar el algoritmo de la forma siguiente:



Hay que tener en cuenta que para la definición de algoritmos mediante diagramas de flujo, se utilizan símbolos estandarizados que permiten reflejar cada uno de los pasos (generales o específicos) del algoritmo. Algunos de los símbolos más utilizados en este tipo de diagramas son los que se muestran en la tabla siguiente:

| Símbolo | Nombre | Descripción |
|-------------------------------------------------------------------------------------|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
|  | Líneas de flujo | Muestran la dirección y sentido del flujo del programa. Interconectan los distintos símbolos. |
|  | Terminador: Comienzo o final de procesos | Dentro situamos información o acciones para comenzar el proceso o para mostrar el resultado final del mismo. |

| Símbolo | Nombre | Descripción |
|-----------------------------------------------------------------------------------|--------------------------------|----------------------------------------------------------------------------------------------------------|
|  | Proceso o actividad | Tarea o actividad llevada a cabo durante el algoritmo. Puede tener muchas entradas pero sólo una salida. |
|  | Conector | Nombramos un proceso independiente que en algún momento está relacionado con el proceso principal. |
|  | Datos Entrada/Salida | En su interior situamos la información necesaria para alimentar una actividad. |
|  | Decisión o bifurcación | Indican puntos donde se debe tomar una decisión. |

1.3. Fases del proceso de resolución de un problema.

El desarrollo de un proyecto software para la resolución de un problema requiere de los siguientes pasos:

1. Definición y análisis del problema
2. Diseño del algoritmo
3. Codificación del programa
4. Compilación
5. Depuración de errores y verificación del programa
6. Explotación, documentación y mantenimiento.

1.4. ¿Qué es un programa?

Los **programas** son una serie o secuencia de instrucciones entendibles por los ordenadores que permiten la realización de las acciones o tareas para las que han sido creadas. Para escribir un programa se utilizan distintos lenguajes, denominados *lenguajes de programación*.

Los programas se escriben en lo que se denomina **lenguaje fuente**, en el que se especifican las instrucciones que el programador desea que se ejecuten con el fin de realizar las acciones para las que se ha diseñado el algoritmo. Sin embargo, este código no lo puede entender directamente el ordenador y debe ser traducido al único lenguaje que el ordenador puede interpretar: el **lenguaje máquina** o código binario. Para realizar esta transformación del código se utilizan unos traductores, denominado **compiladores** e intérpretes, que convierten las instrucciones dadas por el programador en instrucciones comprensibles por un ordenador y generan el llamado código objeto, que es el archivo fuente traducido a lenguaje máquina.

2. Lenguajes de programación.

La programación, en términos informáticos, es la parte de la informática que se dedica a la creación de programas. Un **lenguaje de programación** es un conjunto de símbolos y palabras (instrucciones y sentencias) que el usuario tiene a su disposición para elaborar un programa. Formalmente, un **lenguaje de programación** se define como un conjunto de símbolos, reglas sintácticas y reglas semánticas que se utiliza para controlar el comportamiento físico y lógico de un ordenador. Las reglas sintácticas definen la estructura del lenguaje y las reglas semánticas definen el significado de sus elementos y expresiones. Aunque muchas veces se usan los términos 'lenguaje de programación' y 'lenguaje informático' como si fuesen sinónimos, no es del todo correcto, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otro tipo de lenguajes, como por ejemplo HTML que es un lenguaje de marcas.

Un lenguaje de programación permite especificar de manera precisa sobre qué datos debe trabajar un ordenador, cómo se deben almacenar o transmitir dichos datos y qué acciones deben tomar bajo un amplio conjunto de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural.

Para definir un lenguaje de programación es necesario especificar:

- Conjunto de símbolos y palabras clave utilizables.
- Reglas gramaticales para construir sentencias (instrucciones, órdenes) sintáctica y semánticamente correctas. Se entiende por sintaxis el conjunto de normas que determinan cómo escribir las sentencias del lenguaje. Se denomina semántica al significado de las sentencias.

Esto implica que, para implementar un mismo algoritmo en distintos lenguajes de programación, tendremos que utilizar en cada caso el conjunto de símbolos y palabras reservadas, así como el conjunto de reglas gramaticales correspondientes al lenguaje de programación en cuestión. Por ejemplo, a continuación incluimos el código de tres programas que hacen lo mismo (mostrar por pantalla el texto “Hola mundo”) pero que se han implementado utilizando distintos lenguajes de programación:

Pascal

```
program HolaMundo;  
  begin  
    writeln ('Hola mundo')  
  end.
```

C

```
#include <stdio.h>  
int main()  
{  
    printf ("Hola mundo");  
    return 0;  
}
```

Java

```
public class HolaMundo {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
}
```

Nótese que si se siguen las reglas de indentación se puede observar claramente los niveles en la estructura del programa utilizados en cada caso.

3. Tipos de lenguajes de programación.

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios: según el nivel de abstracción, la forma de ejecución o el paradigma de programación que utiliza cada uno de ellos.

3.1. Nivel de abstracción

Según el nivel de abstracción los lenguajes de programación se clasifican en:

3.1.1. Lenguajes de bajo nivel

En este tipo se encuentran los lenguajes máquina directamente entendible por la máquina (el ordenador), siendo sus instrucciones cadenas binarias (de 0 y 1). El **lenguaje máquina** es el lenguaje en el que hay que expresar cualquier cosa que se desee que realice un ordenador. Es obvio que para los humanos expresarse en estos términos es muy complicado, y por eso este tipo de lenguaje dejó de utilizarse, a causa de su complejidad y de la facilidad para cometer errores al utilizarlo.

El **lenguaje ensamblador** fue el primer lenguaje de programación que trató de sustituir al lenguaje máquina por uno mucho más parecido al de los seres humanos. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como nemónicos: por ejemplo, para indicar que se quiere realizar una suma se usa una instrucción llamada ADD. Para que el programa escrito en lenguaje ensamblador pueda ejecutarse es necesario traducirlo a lenguaje máquina mediante un programa intérprete denominado también ensamblador (*assembler*). El principal inconveniente del lenguaje ensamblador es que está muy ligado al tipo de máquina en que se ejecuta, lo que impide su portabilidad. Además, aunque constituye un importante paso para liberarse del código binario, se necesitan muchas instrucciones para realizar tareas simples y exige grandes conocimientos sobre los componentes hardware de la máquina, por lo que programar en ensamblador resulta aún muy complejo.

3.1.2. Lenguajes de alto nivel

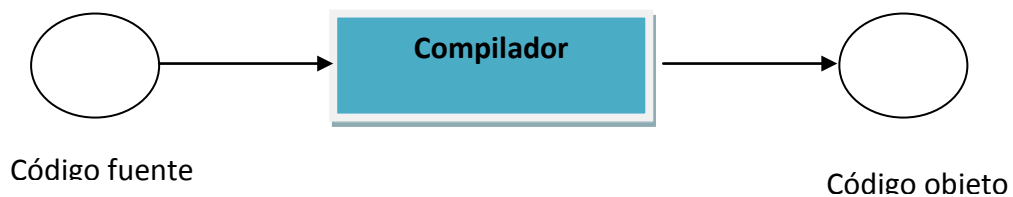
Estos lenguajes están diseñados para que los programadores escriban y entiendan instrucciones lo más parecidas al lenguaje humano (normalmente el inglés), lo cual hace que se necesite menos tiempo para aprender a programar; por ello son los más utilizados por los programadores. Además, los lenguajes de programación de alto nivel son independientes de la máquina y se pueden ejecutar, prácticamente sin modificaciones en diferentes tipos de ordenadores. Como es lógico, el código fuente escrito en un lenguaje de alto nivel debe ser traducido a código máquina mediante un compilador o un intérprete.

3.2. Forma de ejecución

Según la forma de ejecución los lenguajes de programación se clasifican en:

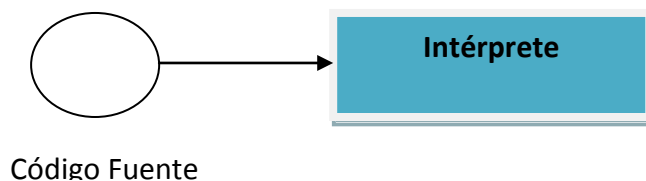
3.2.1. Lenguajes compilados

Antes de poder ejecutar un programa escrito en un lenguaje de programación compilado es necesario invocar a un traductor denominado **compilador** que se encarga de traducir (*compilar*) el programa original (código fuente) al programa equivalente escrito en lenguaje máquina o ensamblador (código objeto). Los binarios son los programas ejecutables y los únicos necesarios para el funcionamiento del programa.



3.2.2. Lenguajes interpretados

Cada vez que se usa el programa debe invocarse a un traductor llamado **intérprete** que se encarga de traducir (interpretar) las instrucciones del programa original (código fuente) a código máquina según van siendo utilizadas. Para el funcionamiento del programa siempre es necesario disponer del código original y del intérprete.



Las principales diferencias entre los lenguajes compilados e interpretados son:

- Los programas escritos en lenguajes compilados se compilan una vez y se utilizan cuantas veces se desee sin necesidad de volver a utilizar el compilador. Los lenguajes interpretados son interpretados, valga la redundancia, cada vez que se ejecutan y necesitan siempre del intérprete.
- Los compiladores analizan todo el programa y no generan resultados si no es correcto todo el código. Los intérpretes analizan las instrucciones según las necesitan y pueden iniciar la ejecución de un programa con errores e incluso terminar

correctamente una ejecución de un programa con errores siempre que no haya sido necesario el uso de las instrucciones que contienen dichos errores.

- Un compilador traduce cada instrucción una sola vez. Un intérprete debe traducir una instrucción cada vez que la encuentra.
- Los códigos objeto son compilados para una arquitectura específica y no se pueden utilizar en otras arquitecturas no compatibles (aunque pueden existir distintos compiladores para generar códigos objeto para diferentes arquitecturas). Un lenguaje interpretado puede ser utilizado en cualquier arquitectura que disponga de un intérprete sin necesidad de cambios.
- Los lenguajes compilados son más eficientes que los interpretados y además permiten distribuir el programa en forma confidencial mediante código binario.

3.3. Paradigmas de programación

Los lenguajes de programación también pueden clasificarse teniendo en cuenta el paradigma que siguen. Un paradigma de programación representa un enfoque particular o filosofía para la construcción de software. Si bien puede seleccionarse la forma pura de estos paradigmas a la hora de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multi-paradigma. Los diferentes paradigmas de programación son:

3.3.1. Imperativo

Este modelo de programación también se conoce con los nombres de algorítmico, o por procedimientos. Es el más común. Describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea. La implementación de hardware de la mayoría de computadores es imperativa ya que el hardware está diseñado para ejecutar código de máquina que es imperativo.

3.3.2. Declarativo

También se denomina a este paradigma predicativo. Se basa en el uso de predicados lógicos (lógico) o funciones matemáticas (funcional), su objetivo es conseguir lenguajes expresivos en los que no sea necesario especificar cómo resolver el problema (programación convencional imperativa), sino qué problema se desea resolver. Los intérpretes de los lenguajes declarativos tienen incorporado un motor de inferencia genérico que resuelve los problemas a partir de su especificación.

- **Lógico.** El mecanismo de inferencia genérico se basa en los procedimientos de deducción de formulas válidas en un sistema axiomático.
- **Funcional.** El mecanismo de inferencia genérico se basa en la reducción de una expresión funcional a otra equivalente simplificada.

3.3.3. Orientado a objetos.

Se utiliza cada vez más, sobre todo en combinación con el modelo imperativo. De hecho los lenguajes orientados a objetos permiten la programación imperativa. Usa objetos y sus interacciones para diseñar aplicaciones y programas. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

4. Traductores, compiladores e intérpretes.

En los apartados anteriores se ha visto que el lenguaje que entiende la máquina directamente se denomina lenguaje máquina. Cualquier programa escrito en un lenguaje diferente a ese se ha de traducir antes de que el ordenador pueda ejecutarlo. Un **traductor** es un programa que toma como entrada un programa escrito en un lenguaje fuente y lo transforma en un programa escrito en un lenguaje objeto. A proceso de conversión se le denomina traducción y el mismo puede realizarse de dos maneras diferentes, por interpretación o por compilación.

Un **intérprete** es un programa que toma como entrada un programa escrito en lenguaje fuente y lo va traduciendo y ejecutando instrucción por instrucción, una detrás de la otra, de una en una.

Un **compilador** es un programa que toma como entrada un programa escrito en un lenguaje fuente de alto nivel y genera un programa equivalente, denominado programa objeto, escrito en un lenguaje de bajo nivel.

El proceso de traducción se divide generalmente en dos fases: una de análisis y otra de síntesis. La **fase de análisis** es más cercana al lenguaje fuente y consiste en comprobar la sintaxis y la semántica de los programas fuente. Durante la fase de análisis se realizan principalmente, tres tipos de comprobaciones:

1. **Análisis léxico:** elimina del programa fuente toda la información innecesaria (espacios y líneas en blanco, comentarios, etc.) y comprueba que los símbolos del lenguaje (palabras clave, operadores, etc.) se han escrito correctamente.

2. **Análisis sintáctico:** comprueba si lo obtenido en el análisis léxico es sintácticamente correcto, es decir, si está escrito conforme a la gramática del lenguaje.
3. **Análisis semántico:** comprueba que el significado de las sentencias del programa es correcto.

La **fase de síntesis** es más cercana al lenguaje de bajo nivel y se encarga de generar el código más eficiente para la máquina destino de la traducción. Sólo se genera código objeto cuando el programa fuente está libre de errores de análisis, lo cual no quiere decir que el programa se ejecute correctamente, ya que un programa puede tener errores de concepto o expresiones mal calculadas.

5. Bibliografía y enlaces de interés.

- *Tecnología de la Información y la Comunicación*, A. Gómez Gilaberte, E. Parramón Ponz, T. Antúnez Izquierdo, Editorial Donostiarra, 2009.
- *Fundamentos de Informática y Programación Científica, Resolución en C y Matlab*. J.M. Zamarreño. M.T. Álvarez, L. Felipe, M. García, F. Taedo, Junta de Castilla y León, 2000.
- Definición de Programación - Wikipedia:
<http://es.wikipedia.org/wiki/Programaci%C3%B3n>
- Fundamentos de Informática y Programación Científica:
http://www.isa.cie.uva.es/~jesusm/libro/libro_pc.html