TEMA 2: LENGUAJES DE PROGRAMACIÓN.

Componentes del grupo:

- Soraya María Alvaredo Coco
- Alberto Delgado Lastras
- Santiago Aira Zunzunegui
- Alberto Rubio Pascual
- Jesús Herrero Herrero
- Luis María Mozo Vaquero

Objetivos del tema

- Resumir la evolución de los lenguajes de programación, ilustrando cómo ha dado lugar a los paradigmas disponibles en la actualidad.
- Evaluar los diferentes paradigmas, considerando aspectos tales como la eficiencia en cuanto al espacio, la eficiencia en cuanto al tiempo (tanto del ordenador como del programador), la seguridad y la potencia de las expresiones.
- Describir las fases de la traducción de programas desde código fuente a código ejecutable y los ficheros generados por estas fases.
- Comparar modelos de ejecución compilados e interpretados, esbozando las características de cada uno.
- Explicar las diferencias entre la traducción dependiente de la máquina y la independiente, indicando dónde son evidentes estas diferencias en el proceso de traducción.

DESCRIPCIÓN DE CONTENIDOS

En el primer tema se presentó ya el concepto de lenguaje de programación. Este tema comienza recordando dicho concepto.
A continuación, se hace una breve revisión histórica de los lenguajes de programación, comenzando por el lenguaje máquina utilizado en los primeros tiempos y llegando a los lenguajes utilizados en la actualidad.
El tercer apartado presenta los paradigmas de programación fundamentales, indicando las características generales de cada uno de ellos. Se considera interesante que el alumno tenga conocimiento de la existencia de diversos paradigmas de programación, aparte del que se va a utilizar en la asignatura. Se describirán fundamentalmente el paradigma imperativo, el lógico, el funcional y el basado en objetos, pero también se comentarán brevemente otros paradigmas, como la programación concurrente o la manejada por eventos.
El siguiente apartado presenta algunas clasificaciones de los lenguajes actuales, atendiendo a criterios como su proximidad al lenguaje natural, el que sean de propósito general o particular o el paradigma de programación asociado.
El último apartado describe el proceso de traducción de lenguajes. Se comenzará justificando la necesidad de los traductores para ejecutar programas escritos en lenguajes diferentes del lenguaje máquina. Será necesario traducir los programas del lenguaje de programación en el que están escritos a un lenguaje que pueda entender la computadora, como paso previo a su ejecución. Se describirán los tipos de traductores disponibles, así como sus características. Por último, se describirá el proceso de traducción utilizando cada tipo de traductor, analizando las etapas y los ficheros que se generan en cada una.

Puntos a desarrollar

1. Concepto de lenguaje de programación.

2. Historia y evolución de los lenguajes de programación.

3. Paradigmas de programación.

4. Algunas clasificaciones de los lenguajes de programación.

5. Traductores.

INTRODUCCIÓN

Los ordenadores no hablan nuestro idioma, son maquinas y como tales, necesitan un lenguaje especifico pensado por el hombre para ellas. Además, necesitan constantemente interpretar todas las instrucciones que reciben. Dada la dificultad de comunicación insalvable entre el computador y el programador, pronto aparecieron lenguajes de programación que hacen posible la comunicación con el microprocesador, utilizando términos y símbolos relacionados con el tipo de problema que se debe resolver, mediante el empleo de herramientas que brinda la informática.

En la actualidad hay muchos tipos de lenguajes de programación, cada uno de ellos con su propia gramática, su terminología especial y una sintaxis particular. Por ejemplo, existen algunos creados especialmente para aplicaciones científicas o matemáticas generales (BASIC, FORTRAN, PASCAL, etc.); otros, en cambio, se orientan al campo empresarial y al manejo de textos y ficheros, es decir, son en realidad fundamentalmente gestores de información (COBOL, PL/1, etc.), o muy relacionados con el lenguaje máquina del ordenador (como el C y el ASSEMBLER）.

Los ordenadores se programaban en lenguaje máquina pero las dificultades que esto conllevaba, junto con la enorme facilidad de cometer errores, cuya localización era larga y compleja, hicieron concebir, en la década de los 40, la posibilidad de usar lenguajes simbólicos. Los primeros en aparecer fueron los ensambladores, fundamentalmente consistía en dar un nombre (mnemónico) a cada tipo de instrucción y cada dirección (etiqueta). Al principio sé hacia el programa sobre papel y, después se traducía a mano con la ayuda de unas tablas, y se introducían en la máquina en forma numérica, pero pronto aparecieron programas que se ensamblaban automáticamente.

Concepto de lenguaje de programación.

Los programas indican al ordenador qué tienes que hacer, y éste únicamente realiza las operaciones que el programa incluye. Un programa y las sentencias que lo constituyen se construyen con unos símbolos, y de acuerdo con unas reglas, que constituyen la gramática del lenguaje de programación.

Se puede definir un **lenguaje de programación** como el conjunto de símbolos y de reglas para combinarlos, que se usan para expresar algoritmos.

Los lenguajes de programación a semejanza de los lenguajes que usan los humanos para comunicarse, poseen un léxico (vocabulario o conjunto de símbolos permitidos), una sintaxis que indica cómo realizar construcciones del lenguaje y una semántica que determina el significado de cada construcción correcta.

Historia y evolución de los lenguajes de programación.

Los primeros lenguajes de programación surgieron de la idea de Charles Babagge, la cual se le ocurrió a este hombre a mediados del siglo XIX. Consistía en lo que él denominaba la maquina analítica, pero que por motivos técnicos no pudo construirse hasta mediados del siglo XX. Con él colaboro Ada Lovedby, la cual es considerada como la primera programadora de la historia, pues realizo programas para aquélla supuesta maquina de Babagge, en tarjetas perforadas. En cuanto se empezó a programar, los programadores utilizaron las técnicas diseñadas por Charles Babagge, y Ada, que consistían entre otras, en la programación mediante tarjetas perforadas.

Paradigmas de programación.

• Definición Teórica

Un paradigma está constituido por los supuestos teóricos generales, las leyes y las técnicas para su aplicación que adoptan los miembros de una determinada comunidad científica.

Paradigmas de Programación: Representan un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

• Tipos de Paradigmas de Programación

1. **Paradigmas Imperativo:** Son aquellos que facilitan los cálculos por medio de cambios de estado, entendiendo como estado la condición de una memoria de almacenamiento. Los lenguajes estructurados en bloques, se refieren a los ámbitos anidados, es decir los bloques pueden estar anidados dentro de otros bloques y contener sus propias variables. La RAM representa una pila con una referencia al bloque que está actualmente activo en la parte superior.

> *Paradigma heurístico:* Define un modelo de resolución de problemas en el que se incorpora algún componente heurístico, sobre la base de una representación más apropiada de la estructura del problema, para su resolución con técnicas heurísticas.
>
> *Paradigma concurrente*: La programación distribuida ha sido dividida en dos amplias categorías, sistemas acoplados en forma débil o fuerte. El término distribuido se refiere por lo general a lenguajes para sistemas acoplados débilmente que soportan un grupo de programadores trabajando en un programa particular de manera simultánea y comunicándose a través de paso de mensajes mediante un canal de comunicación. Un sistema acoplado fuertemente permite que más de un proceso en ejecución tenga acceso a la misma ubicación de memoria. Un lenguaje acoplado con el sistema debe sincronizar el uso compartido de la memoria, de modo que solo un proceso escriba una variable compartida a la vez, y de modo que un proceso pueda esperar hasta que ciertas condiciones se satisfagan por completo antes de continuar la ejecución. La memoria compartida tiene la ventaja de la velocidad, por que no se necesita pasar mensajes.

2. Paradigmas Procedimentales: Modelos de Desarrollo: Orientado a Objetos, a Eventos, y a Agentes. Secuencia computacional realizada etapa a etapa para resolver el

problema. Su mayor dificultad reside en determinar si el valor computado es una solución correcta del problema.

3. Paradigmas Declarativos: Modelos de Desarrollo: Funcional, Lógico y de Flujo de Datos. Se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución.

4. Paradigmas Demostrativos: Modelos de Desarrollo: Genético. Cuando se programa bajo un paradigma demostrativo (también llamada programación por ejemplos), el programador no especifica procedimentalmente cómo construir una solución sino que presentan soluciones de problemas similares.

5. Paradigmas Funcional: Modelo matemático de composición funcional donde el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado.

6. Paradigma Orientado a Objeto: disciplina de ingeniería de desarrollo y modelado de software que permite construir más fácilmente sistemas complejos a partir de componentes individuales. Objetos + Mensajes = Programa.

En teoría, no se puede decir que este mal relacionar un paradigma con un modelo aunque puede dar lugar a alguna confusión, es natural que los lenguajes de programación puros nos establecen paradigmas claros (C, Haskell, Smalltalk, Prolog), pero también nos confunden los lenguajes híbridos al incorporar a sus nuevas versiones paradigmas o seudoparadigmas nuevos (C++, Visual Fox, Builder, Scheme, Lisp).

Algunas clasificaciones de los lenguajes de programación.

- **LENGUAJE MÁQUINA:**

El lenguaje máquina es el único que entiende directamente la computadora, ya que esta escrito en lenguajes directamente inteligibles por la máquina), utiliza el alfabeto binario, que consta de los dos únicos símbolos 0 y 1, denominados bits. Sus instrucciones son cadenas binarias que especifican una operación y, las posiciones de memoria implicadas en la operación se denominan instrucciones de máquina o código maquina. Fue el primer lenguaje utilizado en la programación de computadoras, pero dejo de utilizarse por su dificultad y complicación, siendo sustituido por otros lenguajes más fáciles de aprender y utilizar, que además reducen la posibilidad de cometer errores. Generalmente, en la codificación de los programas se empleaba el sistema hexadecimal para simplificar el trabajo de escritura. Todas las instrucciones preparadas en cualquier lenguaje máquina tienen por lo menos dos partes. La primera es el comando u operación, que dice a las computadoras cual es la función que va a realizar.

- **LENGUAJES DE BAJO NIVEL (ensamblador):**

Son más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador. El lenguaje ensamblador es el primer intento de sustituir el lenguaje maquina por otro más similar a los utilizados por las personas. A principios de la década de los 50 y con el fin de facilitar la labor de los programadores, se desarrollaron códigos mnemotécnicos para las operaciones y direcciones simbólicas. Los códigos mnemotécnicas son los símbolos alfabéticos del lenguaje maquina. La computadora sigue utilizando el lenguaje maquina para procesar los datos, pero los programas ensambladores traducen antes los símbolos de código a lenguaje maquina. Estos programas de ensamble o ensambladores permiten a la computadora convertir las instrucciones en lenguaje ensamblador del programador en su propio código maquina. Un programa de instrucciones escrito en lenguaje ensamblador por un programador se llama programa fuente. Después de que el ensamblador convierte el programa fuente en código maquina a este se le denomina programa objeto. Para los programadores es más fácil escribir instrucciones en un lenguaje ensamblador que en código de lenguaje maquina.

Un modo más fácil de comprender el código máquina es dando a cada instrucción un mnemónico, como por ejemplo STORE, ADD o JUMP.

Los lenguajes de bajo nivel permiten crear programas muy rápidos, pero que son, a menudo, difíciles de aprender. Más importante es el hecho de que los programas escritos en un bajo nivel sean altamente específicos de cada procesador..

Los lenguajes ensamblador tienen sus aplicaciones muy reducidas, se centran básicamente en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos.

- **LENGUAJES DE ALTO NIVEL:**

Estos lenguajes son los mas utilizado por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho mas fácil que los lenguajes máquina y ensamblador. Un programa escrito en lenguaje de alto nivel es independiente de la máquina. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras. Son lenguajes de programación en los que las instrucciones enviadas para que el ordenador ejecute ciertas órdenes son similares al lenguaje humano. Dado que el ordenador no es capaz de reconocer estas ordenes, es necesario el uso de un intérprete que traduzca el lenguaje de alto nivel a un lenguaje de bajo nivel que el sistema pueda entender.

Por lo general se piensa que los ordenadores son máquinas que realizan tareas de cálculos o procesamiento de texto. La descripción anterior es sólo una forma muy esquemática de ver una computadora. Hay un alto nivel de abstracción entre lo que se pide a la computadora y lo que realmente comprende. Existe también una relación compleja entre los lenguajes de alto nivel y el código máquina.

Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales, como el inglés. En BASIC, el lenguaje de alto nivel más conocido, los comandos como “IF CONTADOR=10 THEN STOP” pueden utilizarse para pedir a la computadora que pare si CONTADOR es igual a diez. Por desgracia para muchas personas esta forma de trabajar es un poco frustrante, dado que a pesar de que las computadoras parecen comprender un lenguaje natural, lo hacen en realidad de una forma rígida y sistemática.

Los lenguajes de alto nivel, también denominados lenguajes evolucionados, surgen con posterioridad a los anteriores (lenguaje máquina, lenguajes de bajo nivel o ensamblador) con los siguientes objetivos, entre otros:
Lograr independencia de la maquina, pudiendo utilizar un mismo programa en diferentes equipos con la única condición de disponer de un programa traductor o compilador, que es suministrado por el fabricante, para obtener el programa ejecutable en lenguaje binario de la maquina que se trate. Además, no se necesita conocer el hardware especifico de dicha maquina. Aproximarse al lenguaje natural, para que el programa se pueda escribir y leer de una forma más sencilla, eliminando muchas de las posibilidades de cometer errores que se daban en el lenguaje maquina, ya que se utilizan palabras (en ingles) en lugar de cadenas de símbolos sin ningún significado aparente.

Incluir rutinas de uso frecuente, como las de entrada / salida, funciones matemáticas, manejo de tablas, etc., que figuran en una especie de librería del lenguaje, de manera que se puedan utilizar siempre que se quiera sin necesidad de programarlas cada vez.

Ventajas de los lenguajes de alto nivel: el tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes. La escritura de

programas se basa en reglas sintácticas similares a los lenguajes humanos, nombres de las instrucciones tales como READ, WRITE, PRINT, OPEN, etc. Las modificaciones y puestas a punto de los programas son más fáciles. Reducción del costo de los programas. Transportabilidad. Permiten tener una mejor documentación. Son más fáciles de mantener.

Desventajas de los lenguajes de alto nivel: incremento del tiempo de puesta a punto al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo. No se aprovechan los recursos internos de la maquina que se explotan mucho mejor en lenguajes máquina y ensambladores. Aumento de la ocupación de memoria. El tiempo de ejecución de los programas es mucho mayor.

Se puede decir que el principal problema que presentan los lenguajes de alto nivel es la gran cantidad de ellos que existen actualmente en uso, además de las diferentes versiones o dialectos que se han desarrollado de algunos de ellos. Es difícil establecer una clasificación general de los mismos, ya que en cualquiera que se realice habrá lenguajes que pertenezcan a mas de uno de los grupos establecidos. Una clasificación muy extendida, atendiendo a la forma de trabajar de los programas y a la filosofía con que fueron concebidos, es la siguiente:

- **Lenguajes imperativos**. Utilizan instrucciones como unidad de trabajo de los programas (Cobol, Pascal, C, Ada).
- **Lenguajes declarativos**. Los programas se construyen mediante descripciones de funciones o expresiones lógicas (Lisp, Prolog).
- **Lenguajes orientados a objetos**. El diseño de los programas se basa mas en los datos y su estructura. La unidad de proceso es el objeto y en el se incluyen los datos (variables) y las operaciones que actúan sobre ellos (Smalltalk, C++).
- **Lenguajes orientados al problema**. Diseñados para problemas específicos, principalmente de gestión, suelen ser generadores de aplicaciones.
- **Lenguajes naturales**. Están desarrollándose nuevos lenguajes con el principal objetivo de aproximar el diseño y construcción de programas al lenguaje de las personas.

Otra clasificación que se puede hacer es la de atendiendo al desarrollo de los lenguajes desde la aparición de las computadoras, que sigue un cierto paralelismo con las generaciones establecidas en la evolución de las mismas:

- **Primera generación.** Lenguajes maquina y ensambladores.
- **Segunda generación.** Primeros lenguajes de alto nivel imperativo (FROTRAN, COBOL).
- **Tercera generación.** Lenguajes de alto nivel imperativo. Son los mas utilizados y siguen vigentes en la actualidad (ALGOL 8, PL/I, PASCAL, MODULA).

- **Cuarta generación.** Orientados básicamente a las aplicaciones de gestión y al manejo de bases de datos (NATURAL, SQL).
- **Quinta generación.** Orientados a la inteligencia artificial y al procesamiento de los lenguajes naturales (LISP, PROLOG).

Para la mejor compresión se harán unas definiciones:

Programa: es un conjunto de instrucciones escritas en un lenguaje de programación que indican a la computadora la secuencia de pasos, para resolver un problema.

Código fuente: esta creado en algún lenguaje de alto nivel, por lo que es entendido 100% por el ser humano. Este debe estar complementado por su documentación o manuales donde se indica el desarrollo lógico del mismo.

Código objeto: es creado por los compiladores y nos sirve como enlace entre el programa fuente y el ejecutable.

Traductores.

Un intérprete es un programa que analiza y ejecuta simultáneamente un programa escrito en un lenguaje fuente.

Cualquier intérprete tiene dos entradas: un programa escrito en un lenguaje fuente LF (en lo sucesivo, se denotará P/LF) junto con los datos de entrada; a partir de dichas entradas, mediante un proceso de interpretación va produciendo unos resultados.

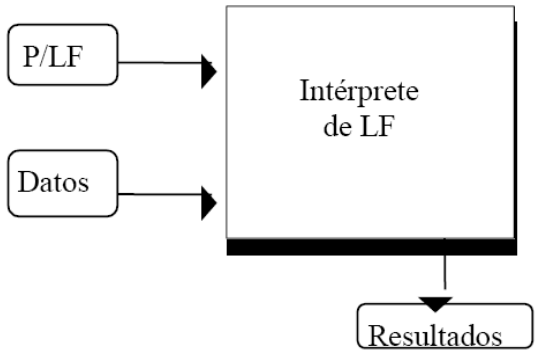

Figura 1: Esquema general de un intérprete

Los compiladores, a diferencia de los intérpretes, transforman el programa a un programa equivalente en un código objeto (fase de compilación), y en un segundo paso generan los resultados a partir de los datos de entrada (fase de ejecución).

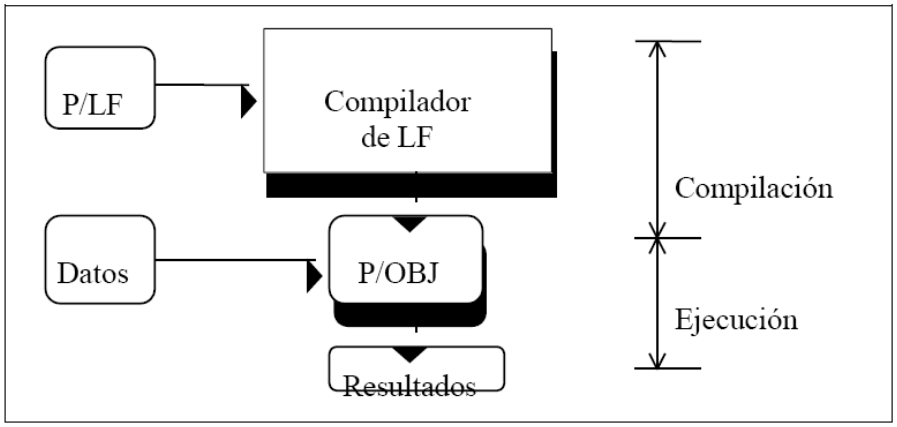

Figura 2: Esquema general de un compilador

Ventajas de la utilización de intérpretes

En general, la utilización de compiladores permite construir programas más eficientes que los correspondientes interpretados. Esto es debido a que durante la ejecución de código compilado no es necesario realizar complejos análisis (ya se hicieron en tiempo de compilación), además, un buen compilador es capaz de detectar errores y optimizar el código generado.

Los intérpretes, por definición, realizan la fase de análisis y ejecución a la vez, lo cual imposibilita tales optimizaciones. Por esta razón, los sistemas interpretados suelen ser menos eficientes que los compilados. No obstante, los nuevos avances informáticos aumentan la velocidad de procesamiento y capacidad de memoria de los ordenadores. Actualmente, la eficiencia es un problema menos grave y muchas veces se prefieren sistemas que permitan un desarrollo rápido de aplicaciones que cumplan fielmente la tarea encomendada.

A continuación se enumeran una serie de ventajas de los sistemas interpretados:

- Los intérpretes, en general, son más **sencillos de implementar**. Lo cual facilita el estudio de la corrección del intérprete y proporciona nuevas líneas de investigación como la generación automática de intérpretes a partir de las especificaciones semánticas del lenguaje.
- Proporcionan una **mayor flexibilidad** que permite modificar y ampliar características del lenguaje fuente. Muchos lenguajes como Lisp, APL, Prolog, etc. surgieron en primer lugar como sistemas interpretados y posteriormente surgieron compiladores.
- No es necesario contener en memoria todo el código fuente. Esto permite su utilización en
- sistemas de poca memoria o en **entornos de red**, en los que se puede obtener el código fuente a medida que se necesita [Plezbert 97].
- Facilitan la **meta-programación**. Un programa puede manipular su propio código fuente a
- medida que se ejecuta. Esto facilita la implementación de sistemas de aprendizaje automatizado y reflectividad [Aït Kaci 91].
- Aumentan la **portabilidad** del lenguaje: Para que el lenguaje interpretado funcione en otra máquina sólo es necesario que su intérprete funcione en dicha máquina.
- Puesto que no existen etapas intermedias de compilación, los sistemas interpretados facilitan el desarrollo rápido de **prototipos**, potencian la utilización de sistemas **interactivos** y facilitan las tareas de **depuración**.