TEMA 2: Lenguajes de programación

2.1.- Introducción a los lenguajes de programación

¿Qué es un lenguaje?

Conjunto de símbolos y palabras (vocabulario) y conjunto de reglas (sintaxis y semántica) que permiten agrupar los símbolos para formar las frases del lenguaje.

¿De programación?

Que sirve para especificar algoritmos sobre un ordenador.

Un programa se escribe como una secuencia de frases del lenguaje.

2.1.1.- Sintaxis

Consta de unas definiciones, denominadas reglas sintácticas o producciones que especifican la secuencia de símbolos que forman una frase del lenguaje. Estas reglas dicen si una frase está bien escrita o no.

Las reglas sintácticas pueden contener dos tipos de elementos:

- Elementos Terminales ($\in$ Vocabulario)
- Elementos no Terminales, que son construcciones intermedias de la gramática.

Existen diversas formas de especificar las reglas, pero únicamente vamos a ver dos de ellas:

- Notación BNF (Backus-Naur Form). Es de las primeras notaciones que se empezó a utilizar para especificar lenguajes de programación.

  Notación BNF: <elemento no terminal>::= Definición1 | Definición2 | ...

  Los elementos terminales, o sea, que pertenecen al vocabulario, se escriben tal cual. Los elementos no terminales se escriben entre los símbolos <>.

Ejemplo: *Descripción sintáctica de una expresión matemática en notación BNF:*
---> 4(3+1)*

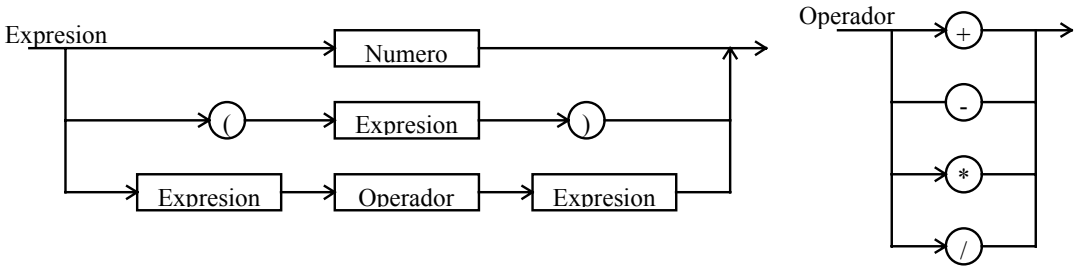
<expresión>::=<numero>|(<expresión>)|<expresión><operador><expresión>

<operador>::=+|-|*|/

- Diagramas sintácticos. Es una representación gráfica de la sintaxis. Tiene la ventaja de ser más intuitivo.

  Los elementos terminales se inscriben en una elipse. Los elementos no terminales se inscriben en un rectángulo.

Ejemplo: *Descripción sintáctica de una expresión matemática en diagrama sintáctico:*
---> 4(3+1)*



2.1.2.- Semántica

Define el significado de las construcciones sintácticas del lenguaje.

Ejemplo:

```
if (a>b) max := a else max := b;
```

el significado corresponde a la construcción algorítmica

Si … entonces … sino …

Además la expresión después de If debe tener un resultado lógico (verdad o falso.)

2.2.- Lenguajes de bajo nivel y lenguajes de alto nivel.

Los lenguajes de programación se pueden clasificar en lenguajes de bajo y alto nivel dependiendo de lo cercanos o lejanos que estén de la arquitectura de la máquina en la que van a funcionar.

2.2.1.- Lenguajes de bajo nivel:

- Están basados directamente en la máquina de Von Neumann, por lo que están a un nivel muy cercano a la máquina.
- Las instrucciones del lenguaje son las instrucciones del microprocesador del ordenador, que normalmente son demasiado simples.
- Es exclusivo de cada ordenador.
- Es difícil y costoso de programar.

En lenguajes de bajo nivel distinguimos entre lenguaje máquina y lenguaje ensamblador.

Lenguaje máquina:

- Instrucciones reconocidas por los circuitos del procesador.
- Se codifican en binario.
- Los datos se referencian por su posición de memoria.

Lenguaje ensamblador:

- Codificación mnemotécnica del lenguaje máquina.
- Necesita un traductor.
- Se pueden utilizar etiquetas en vez de posiciones de memoria.

Ejemplo: *Suma de 3 + 5 en un procesador 8086 (también Pentium, Pentium II, ...)*

Ensamblador	Código máquina (Hexadecimal)
mov ax, 0003	B8 03 00
add ax, 0005	05 05 00

2.2.2.- Lenguajes de alto nivel:

- Están basados en máquinas abstractas, que facilitan la comprensión por personas.
- Instrucciones más flexibles y potentes.
- Necesita un traductor para convertir el programa a lenguaje máquina, que es el que entiende el ordenador.
- No depende del procesador, por lo que el mismo programa sirve para diferentes ordenadores.
- Al tener que traducirlo, es más lento e ineficiente que el lenguaje de bajo nivel.

Ejemplo: *Suma de 3 + 5 en C++*

```
x = 3 + 5
```

2.3.- Clasificación de los lenguajes de alto nivel

2.3.1.- Paradigmas de programación.

Por paradigma de programación se entiende una "filosofía" de programar, es decir, los lenguajes que utilizan un mismo paradigma de programación utilizarán los mismos conceptos básicos para programar. Se pueden definir cuatro tipos fundamentales:

- Imperativos - Orientados a objetos	**Procedurales**, describen los pasos a seguir para encontrar la solución
- Funcionales - Logicos	**Declarativos**, describen el problema a solucionar

Imperativos: La unidad de trabajo es la sentencia (acción). *Pascal, C.*

Orientados a Objetos: La unidad de trabajo es el objeto. Agrupa las estructuras de datos con sus algoritmos. *Smalltalk*.

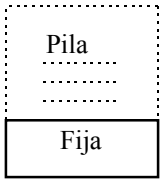
Funcionales: La unidad es la función. Consiste en combinar funciones para conseguir funciones más complejas hasta llegar a la función que es el programa. *Lisp. Basado en el Cálculo Lambda*.

Lógicos: Se especifican los hechos y las propiedades que especifican el problema. El sistema utiliza esa información para encontrar la solución. *Prolog. Basado en la Lógica de Predicados*.

2.3.2.- Clasificación de lenguajes según la administración de memoria

Estáticos: Los requisitos de memoria del programa se pueden calcular antes de ejecutar el programa. No permiten recursividad. *Fortran, Cobol*.

Basados en pila: Se calculan los requisitos de memoria generales del programa antes de ejecutarlo. El resto de la memoria necesaria durante la ejecución del programa se utiliza en forma de pila. *Algol 60*.



Dinámicos: No se puede saber a priori la cantidad de memoria que utilizará el programa. El programa puede crear y destruir estructuras de datos en cualquier lugar del programa. Prolog, Lisp.

C++ es un lenguaje principalmente basado en pila, pero que también tiene características dinámicas.

2.3.3.- Otras clasificaciones

Por la forma en que se pasa a lenguaje máquina.

- Lenguajes compilados → C, Pascal, C++,...
- Lenguajes interpretados → BASIC

Por el objetivo principal de los programas escritos en el lenguaje.

- Lenguajes de propósito general → C, Delphi...
- Lenguajes para la enseñanza → Logo, Pascal, Modula, BASIC...
- Lenguajes para cálculo científico → Fortran, Matlab, Mathematica, Maple...
- Lenguajes para gestión → Cobol...
- Lenguajes para la gestión de bases de datos → *System Query Languages (SQL)* Informix...

- Lenguajes de inteligencia artificial → Prolog, Lisp...
- Programación multiplataforma e internet → Java

2.4.-Historia de los lenguajes de alto nivel

1945- En 1945, el matemático y químico Jonh (Janos) von Neumann presenta los principios generales que debe seguir una máquina de propósito general.

El primer lenguaje en el que se programaron los ordenadores fue el propio del procesador, es decir, instrucciones análogas a las presentes en la máquina de Von Neumann. Sin embargo era necesario tener en cuenta los detalles propios de la máquina para poder realizar cualquier cálculo y además era muy tedioso introducir el programa en el ordenador.

En 1951, apenas siete años después de que Von Neumann introdujera el concepto de programa almacenado en memoria, Wilkes, Wheeler y Gill describen un cargador de programas que realiza la conversión de valores decimales a binarios para permitir una mayor comodidad en la codificación de instrucciones y direcciones. Con objeto de simplificar la programación, los ensambladores fueron enriqueciéndose paulatinamente, hasta convertirse en traductores de representaciones simbólicas (mnemotécnicas) del lenguaje máquina (lenguajes ensambladores) al propio lenguaje máquina.

Los lenguajes ensambladores siguen siendo próximos a los lenguajes de máquina y, si bien simplifican considerablemente el proceso de la programación, mantienen dos de sus principales inconvenientes: requieren un excesivo nivel de detalle en la escritura de programas y son dependientes del sistema computador concreto, de cuyas instrucciones elementales hace representación simbólica.

Es por ello que se intentó crear un nuevo lenguaje que no estuviese basado directamente en las instrucciones propias de la máquina (que no dependiese de la máquina concreta), sino en una abstracción de éstas y que fuese más cómoda para el programador. De la misma forma, no se utilizarían directamente los dispositivos físicos (registros, celdas de memoria, etc.) sino abstracciones de éstos (variables.) De esta forma surge un nuevo concepto de lenguaje de programación, donde cada lenguaje lleva asociado una máquina abstracta sobre el que se puede ejecutar su código.

Si deseamos ejecutar sobre un ordenador concreto los programas escritos en un lenguaje de alto nivel, debemos traducirlos a otros equivalentes en un código máquina concreto (manualmente o mediante un proceso denominado compilación) o disponer de una herramienta que lea el programa e interprete paso a paso el significado de cada sentencia del programa (proceso de interpretación).

1957- Trabajando en esta dirección, entre 1954 y 1958, John Backus lideró un grupo de trabajo que tenía por objeto la realización de un traductor a código máquina de fórmulas matemáticas que expresaran cálculos. El resultado fue tanto la especificación de un lenguaje de alto nivel, el **Fortran,** como la realización de un compilador que traducía

dicho lenguaje al código máquina de un ordenador concreto (IBM 704). Alrededor del año 1960 se crearon tres lenguajes decisivos: el Algol 60, el Cobol y el Lisp.

1958- El **Lisp**, diseñado por John McCarthy, fue también un lenguaje muy innovador en el sentido de que se aleja mucho del concepto de la máquina de Von Neumann. Se basa casi exclusivamente en el uso de funciones y en el cálculo lambda y su objetivo principal era el cálculo simbólico. Fue el precursor de los denominados lenguajes funcionales. Este lenguaje ha sido ampliamente utilizado en el ámbito de la inteligencia artificial.

1959- El **Cobol** fue encargado por el Departamento de Defensa de EUA. Su objetivo era claramente práctico y aunque se realizó sin tener en cuenta algunos de los avances realizados en el momento en el diseño de lenguajes de programación, fue innovador en el tratamiento de los datos. Fue además el primer lenguaje estandarizado, lo que favoreció su utilización posterior.

1960- El **Algol 60** fue principalmente un lenguaje académico, en donde se introdujeron numerosos conceptos innovadores que fueron adoptados por gran parte de los lenguajes posteriores. El lenguaje fue completamente especificado con la notación BNF, un formalismo equivalente a las gramáticas incontextuales.

1964- En 1964 nace el **Basic**. Este lenguaje se diseñó desde el punto de vista del usuario, es decir, se procuró que el lenguaje fuera fácil de aprender y manejar. Este lenguaje tuvo bastante éxito en la enseñanza y sobre todo, en la programación de los primeros micro-computadores, pero apenas se llegó a utilizar en el ambiente profesional.

1967- En 1967 se desarrolló **Simula**, basado en el Algol 60. El campo de aplicación principal de este lenguaje era la simulación, pero lo más importante es que introdujo el concepto de clase y es, por tanto, el precursor de los tipos abstractos de datos.

1972- El **Pascal** nació en torno a 1970, después de la crisis del software. Diseñado por Niklaus Wirth con la idea de crear un lenguaje sencillo y claro que permitiese afrontar la complejidad creciente de los programas de la época. Es el primer lenguaje basado exclusivamente en la programación estructurada, y gracias a su simplicidad ha sido el lenguaje ideal sobre el que desarrollar la semántica de los lenguajes y la verificación formal, cuyos inicios datan también de estos años. Además el Pascal ha sido buen progenitor de posteriores lenguajes. En este sentido, Wirth diseñó posteriormente el **Modula-2** lenguaje levantado sobre muchos conceptos introducidos en Pascal, si bien hace hincapié en la construcción del programa entendido como conjunto de módulos independientes.

1972- Después del Pascal se desarrollaron multitud de lenguajes, entre ellos el **C**, que a costa de un menor nivel de abstracción, aporta una gran flexibilidad y control sobre los recursos de la máquina. Aparece también por esta época el concepto de programación concurrente.

1975- El primer lenguaje de programación lógica, el **Prolog**, se desarrolló en 1975 por los grupos de Kowalski y Colmerauer. Sin embargo, a diferencia de lo ocurrido con otros

lenguajes, su gestación fue bastante larga, se puede considerar que los inicios del Prolog son de 1960.

Al igual que el Lisp, el Prolog se aleja totalmente del concepto de la máquina de Von Neumann y se basa casi totalmente en la lógica de primer orden. El Prolog presenta como importante ventaja frente a otros lenguajes que la verificación de programas es casi inmediata debido a su base lógica, sin embargo, tiene como contrapartida que suele ser bastante ineficiente.

1983- En 1983 se crea el **Ada**, un lenguaje desarrollado bajo los auspicios del Departamento de Defensa norteamericano. Se basa en gran medida en el Pascal, aunque es más complejo. Permite abordar adecuadamente la programación concurrente y el manejo de excepciones e introduce el concepto de sobrecarga.

Actualmente existen gran cantidad de nuevos lenguajes (**C++, Java, Modula-3, Oberon, Delphi, Eiffel…**), la mayoría evoluciones de los presentados aquí, a los que se les ha añadido algunos de los conceptos comentados anteriormente como orientación a objetos, manejo de excepciones, sobrecarga, modularidad, etc.

Resumiendo:

(1957) - FORTRAN (FORmula TRANslation)

Para cálculo científico. Se continúa utilizando aunque está ya muy superado.

(1958) - LISP (LISt Processing)

Se utiliza en Inteligencia Artificial. Funcional. Common Lisp en 1984.

(1959) - COBOL

Aplicaciones de gestión, contabilidad y bases de datos.

(1960) - ALGOL 60

Precursor de la programación estructurada.

(1964) - BASIC

Muy simple, para principiantes. Actualmente existen versiones muy mejoradas.

(1967) - Simula

Primer lenguaje que introduce el concepto de clases y objetos.

(1972) - PASCAL

Programación estructurada. Está pensado para la enseñanza.

(1972) - C

Similar al Pascal, pero no tan estricto. Muy adecuado para la programación de sistemas operativos (UNIX) y en general para la programación a bajo nivel.

(1975) - PROLOG (PROgramación LOGica)

(1980) - Smalltalk

Programación orientada a objetos.

(1983) - Ada

Programación concurrente, manejo de excepciones, …

(1983) - C++

Lenguaje C con orientación a objetos

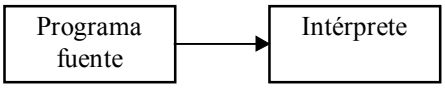
(1994) - JAVA

2.5.- Traductores e intérpretes.

Para que un procesador ejecute un programa escrito en un lenguaje de alto nivel es necesario que lo transforme a su equivalente en lenguaje máquina. Para ello existen dos posibilidades, la interpretación y la traducción.

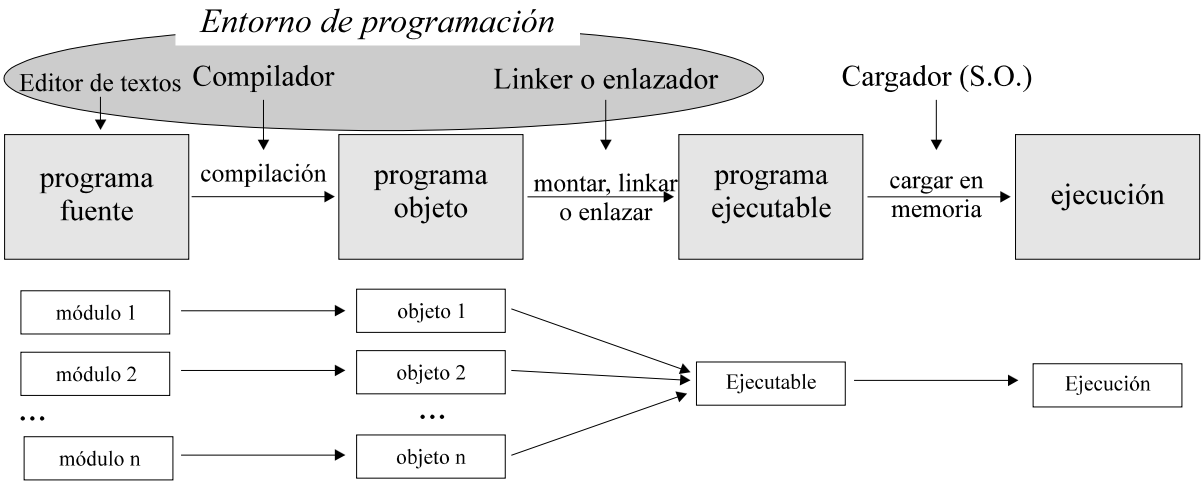
2.5.1.- Interpretación:

Un intérprete traduce y ejecuta sentencia a sentencia el programa original (programa fuente.)



2.5.2.- Traducción:

Traduce el programa original en un programa escrito en lenguaje máquina.



Existen dos tipos de traductores:

- Ensamblador: Cuando el lenguaje fuente es ensamblador.

- Compilador: Cuando el lenguaje fuente es de alto nivel.

2.5.3.- Comparación compiladores-intérpretes:

- Un intérprete puede ejecutar un programa directamente, incluso sin estar completo. Un compilador ha de traducirlo completamente antes de ejecutarlo.
- Un programa interpretado ocupa poca memoria.
- Un programa compilado es más rápido.
- El compilador crea un ejecutable independiente del propio compilador. Un programa interpretado necesita siempre su intérprete.

Actualmente casi todos los lenguajes son compilados o una mezcla entre los dos (Prolog, Java.)

2.5.4.- Fases del proceso de compilación:

- Análisis léxico: Identificación de palabras y símbolos del lenguaje.
- Análisis sintáctico: Comprobación de las reglas sintácticas.
- Análisis semántico: Comprobación de las reglas semánticas (variables no declaradas, comprobación de tipos, …). Interpretación de las órdenes.
- Optimización: Análisis del programa para mejorarlo (en velocidad, en espacio de memoria)

2.5.5.- Tipos de errores:

- Errores de compilación: Los producidos en la fase de compilación o interpretación de un programa, es decir, cuando no se cumplen las reglas sintácticas o semánticas.
- Errores de ejecución: Los producidos durante la ejecución del programa. Estos mensajes de error no son producidos por el compilador, sino por una porción de código que el compilador añade al programa.
- Errores lógicos: Cuando el programa no da ningún error pero los resultados no son los esperados. Puede ser porque el algoritmo ha sido incorrectamente implementado o porque el algoritmo estaba mal hecho.