TEMA 2: LENGUAJES DE PROGRAMACIÓN

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.

1.- CONCEPTO DE LENGUAJE DE PROGRAMACIÓN

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.

Estos lenguajes permiten, por un lado, escribir las operaciones que son necesarias realizar para resolver el problema de un modo parecido a como se escribiría convencionalmente (es decir, redactar adecuadamente el algoritmo de resolución del problema) y, por el otro, se encarga de traducir el algoritmo al lenguaje máquina (proceso conocido como compilación) con lo que se le confiere al programa la capacidad de corre (ser ejecutado) en el ordenador. El ordenador es en realidad tan sólo una máquina virtual, capaz de resolver todos los problemas que los usuarios seamos capaces de expresar mediante un algoritmo (programa).

No obstante, es complicado definir qué es y qué no es un lenguaje de programación. Se asume generalmente que la traducción de las instrucciones a un código que comprende la computadora debe ser completamente sistemática. Normalmente es la computadora la que realiza la traducción.

A continuación, hemos redactado unas cuantas definiciones de los lenguajes de programación.

Un lenguaje de programación es una notación para escribir programas, a través de los cuales podemos comunicarnos con el hardware y dar así las órdenes adecuadas para la realización de un determinado proceso. Un lenguaje esta definido por una gramática o conjunto de reglas que se aplican a un alfabeto constituido por el conjunto de símbolos utilizados. Los distintos niveles de programación existentes nos permiten acceder al hardware, de tal forma que según utilicemos un nivel u otro, así tendremos que utilizar un determinado lenguaje ligado a sus correspondientes traductores.

Conjunto de normas “lingüísticas” (palabras y símbolos) que permiten escribir un programa y que éste sea entendido por el ordenador y pueda ser trasladado a ordenadores similares para su funcionamiento en otros sistemas.

Conjunto de instrucciones, ordenes y símbolos reconocibles por autómata, a través de su unidad de programación, que le permite ejecutar la secuencia de control deseada. Al conjunto de total de estas instrucciones, ordenes y símbolos que están disponibles se le llamar lenguajes de programación del autómata. El programa esta formado por un conjunto de instrucciones, sentencias, bloques funcionales y grafismos que indican las operaciones a realizar. Las instrucciones representan la tarea más elemental de un programa: leer una entrada, realizar una operación, activar una salida, etc. La sentencia representa el mínimo conjunto de instrucciones o sentencias que realizan una tarea o función compleja: encontrar el valor de una función lógica en combinación de varias variables, consultar un conjunto de condiciones, etc. El bloque funcional es el conjunto de instrucciones o sentencias que realizan una tarea o función compleja: contadores, registros de desplazamientos, transferencias de información, etc. Todos estos elementos están relacionados entre sí mediante los símbolos o grafismos.

Es un conjunto de palabras y símbolos que permiten al usuario generar comandos e instrucciones para que la computadora los ejecute. Los lenguajes de programación deben tener instrucciones que pertenecen a las categorías ya familiares de entrada/salida, calculo/manipulación, de textos, lógica/comparación, y almacenamiento/recuperación.

2.- 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. Era un profesor matemático de la universidad de Cambridge e inventor ingles, que a principios del Siglo XIX predijo muchas de las teorías en que se basan los actuales ordenadores. Consistía en lo que él denominaba la máquina 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 realizó programas para aquélla supuesta máquina de Babagge, en tarjetas perforadas. Como la máquina no llegó nunca a construirse, los programas de Ada, lógicamente, tampoco llegaron a ejecutarse, pero si suponen un punto de partida de la programación, sobre todo si observamos que 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. A pesar de ello, Ada ha permanecido como la primera programadora de la historia. Se dice por tanto que estos dos genios de antaño, se adelantaron un siglo a su época, lo cual describe la inteligencia de la que se hallaban dotados.

En 1823 el gobierno Británico lo apoyó para crear el proyecto de una máquina de diferencias, un dispositivo mecánico para efectuar sumas repetidas. Pero Babagge se dedico al proyecto de la máquina analítica, abandonando la maquina de diferencias, que se pudiera programar con tarjetas perforadas, gracias a la creación de Charles Jacquard (francés). Este hombre era un fabricante de tejidos y había creado un telar que podía reproducir automáticamente patrones de tejidos, leyendo la información codificada en patrones de agujeros perforados en tarjetas de papel rígido. Entonces Babagge

intentó crear la máquina que se pudiera programar con tarjetas perforadas para efectuar cualquier cálculo con una precisión de 20 dígitos. Pero la tecnología de la época no bastaba para hacer realidad sus ideas. Si bien las ideas de Babagge no llegaron a materializarse de forma definitiva, su contribución es decisiva, ya que los ordenadores actuales responden a un esquema análogo al de la máquina analítica. En su diseño, la máquina constaba de cinco unidades básicas:
1) **Unidad de entrada**, para introducir datos e instrucciones
2) **Memoria**, donde se almacenaban datos y resultados intermedios
3) **Unidad de control**, para regular la secuencia de ejecución de las operaciones
4) **Unidad Aritmético-Lógica**, que efectúa las operaciones
5) **Unidad de salida**, encargada de comunicar al exterior los resultados.

Charles Babbage, conocido como el "padre de la informática" no pudo completar en aquella época la construcción del computador que había soñado, dado que faltaba algo fundamental: la electrónica. El camino señalado de Babbage, no fue nunca abandonado y siguiéndolo, se construyeron los primeros computadores.

Cuando surgió el primer ordenador, el famoso ENIAC (Electronic Numerical Integrator And Calculator), su programación se basaba en componentes físicos, o sea, que se programaba, cambiando directamente el Hardware de la máquina, exactamente lo que se hacía era cambiar cables de sitio para conseguir así la programación de la maquina. La entrada y salida de datos se realizaba mediante tarjetas perforadas. La dificultad de programar máquinas como la ENIAC de esta manera limitaba drásticamente su utilidad, y proporcionaba un fuerte incentivo para que se desarrollaran lenguajes de programación más orientados hacia la expresión de soluciones con la notación de los problemas mismos.

Los primeros lenguajes de programación se conocieron como **Lenguajes Ensambladores**, un ejemplo es: TRANSCODE, desarrollado para la computadora FERUT. En los lenguajes ensambladores se define un código especial llamado **mnemónico** para cada una de las operaciones de la máquina y se introduce una notación especial para especificar el dato con el cual debe realizarse la operación.

A mediados de los años 60 aparecieron los primeros lenguajes de propósito general como FORTRAN, FORTRAN IV, ALGOL, COBOL, BASIC, PL/I, ADA, C, C++, PASCAL, etc. pero el desarrollo de nuevas tecnologías, tanto en arquitectura de computadoras como en lenguajes de programación, continúa a paso acelerado, cada vez con mayor velocidad, el panorama está cambiando de una etapa de sistemas y lenguajes especialmente desarrollados para aplicaciones individuales. Los lenguajes de programación actuales son los conocidos como ***Lenguajes visuales***, como por ejemplo Visual Fox, Visual Basic, Visual C.

3.-PARADIGMAS DE PROGRAMACION

Representan un enfoque particular o filosófico 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 mas apropiado que otro.

Tipos de paradigmas de programación:

- **Paradigmas Imperativo:** Modelo abstracto que consiste en un gran almacenamiento de memoria donde la computadora almacena una representación codificada de un calculo y ejecuta una secuencia de comandos que modifican el contenido de ese almacenamiento. Algoritmos + Estructura de Datos = Programa.

- **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.

- **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

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

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

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

4.- ALGUNAS CLASIFICACIONES DE LOS LENGUAJES DE PROGRAMACIÓN

Clasificación según se asemejen más al lenguaje del programador o menos:

Lenguaje máquina:

- Es directamente inteligible por la máquina.
- Las instrucciones son cadenas binarias en las cuales se especifica:
- Esta ligado al Hardware; difiriendo así de una computadora a otra.

Ventajas e inconvenientes del lenguaje maquina:

Ventajas:

a Mayor velocidad de ejecución, por ser directamente ejecutable.

Inconvenientes:

a) Dificultad y lentitud de la codificación
b) Poca fiabilidad.
c) Gran dificultad a la hora de verificar y poner a punto los programas.
d) Los programas solo son ejecutables para el procesador sobre el que se crean.

Lenguajes de bajo nivel (Ensamblador)

- Son más fáciles de interpretar que los lenguajes máquina.
- Dependen de la maquina.
- Sus instrucciones son nemotécnicos (más parecidos al lenguaje del programador que el anterior, que es el lenguaje de la maquina como su propio nombre dice).

Ventajas e inconvenientes:

1 Ventajas:

1.1.aSe codifican mejor que en el lenguaje máquina.
1.1.bMayor velocidad de calculo

1 Inconvenientes:

a) Dependencia total de la máquina.
b) Los programadores necesitan tener una visión global del sistema; es decir, necesitan conocer el lenguaje y el interior de la maquina (conocer su software).

Lenguajes de alto nivel:

- Es un lenguaje más parecido al del programador.
- Es independiente de la máquina.

Ventajas e inconvenientes:

1. Ventajas:
   a) Tiempo de formación de los programas es relativamente corto.
   b) Las modificaciones y puestas a punto de los proyectos son más fáciles.
   c) Reducción del coste de los programas.
   d) Transportabilidad.

1. Inconvenientes:
   a) Incremento del tiempo de ejecución.
   b) No se aprovechan al 100% los recursos de la máquina.
   c) Aumento del tamaño en memoria.

Tipo de instrucciones (el lenguaje lo determina):

LENGUAJES DE PROGRAMACI?N DECLARATIVOS

Se les conoce como lenguajes declarativos en ciencias computacionales a aquellos lenguajes de programaci?n en los cuales se le indica a la computadora qu? es lo que se desea obtener o qu? es lo que se esta buscando, por ejemplo: Obtener los nombres de todos los empleados que tengan m?s de 32 a?os. Eso se puede lograr con un lenguaje declarativo como SQL.

La programaci?n declarativa es una forma de programaci?n que implica la descripci?n de un problema dado en lugar de proveer una soluci?n para dicho problema, dejando la interpretaci?n de los pasos espec?ficos para llegar a dicha soluci?n a un int?rprete no especificado. La programaci?n declarativa adopta, por lo tanto, un enfoque diferente al de la programaci?n imperativa tradicional.

En otras palabras, la programaci?n declarativa provee el □gqu?□h, pero deja el □gc?mo□h liberado a la implementaci?n particular del int?rprete. Por lo tanto se puede ver que la programaci?n declarativa tiene dos fases bien diferenciadas, la declaraci?n y la interpretaci?n.

Es importante se?alar que a pesar de hacer referencia a int?rprete, no hay que limitarse a □glenguajes interpretados□h en el sentido habitual del t?rmino, sino que tambi?n se puede estar trabajando con □glenguajes compilados□h.

1. CARACTERÍSTICAS DE LOS LENGUAJES DE PROGRAMACIÓN DECLARATIVOS

· Los lenguajes declarativos están orientados a buscar la solución del problema, sin preocuparse por la forma de llegar a ello; es decir, el programador debe concentrarse en la lógica del algoritmo, más que en el control de la secuencia.

· Los programas están formados por un conjunto de definiciones o ecuaciones, las cuales describen lo que debe ser calculado, no en sí la forma de hacerlo.

· Las variables sólo pueden tener asignado un solo valor a lo largo de la ejecución del programa, lo cual implica que no puede existir asignación destructiva. Debido a esto, cobra especial importancia el uso del anidamiento y la recursividad.

· Las listas representan la estructura fundamental de datos.

· El orden de la ejecución no resulta importante debido a que no existen efectos colaterales; es decir, que al calcular un valor, resulta imposible afectar el cálculo de otros y con esto se puede afirmar que cualquier secuencia de ejecución deberá conducir al mismo resultado.

· Las expresiones o definiciones pueden ser usadas como valores y por lo tanto se pueden tratar como argumentos de otras definiciones.

· El control de la ejecución no es responsabilidad del programador.

2. DESVENTAJAS DE LA PROGRAMACIÓN DECLARATIVA

La principal desventaja de la programación declarativa es que no puede resolver cualquier problema dado, sino que está restringida al subconjunto de problemas para los que el intérprete o compilador fue diseñado.

Otra desventaja de la programación declarativa está relacionada con la eficiencia. Dado que es necesaria una fase de interpretación extra, en la cual se deben evaluar todas las consecuencias de todas las declaraciones realizadas, el proceso es relativamente más lento que en la programación imperativa, en que los cambios de estado del sistema están dados por instrucciones particulares y no por un conjunto de condiciones arbitrariamente grande.

3. VENTAJAS DE LA PROGRAMACIÓN DECLARATIVA

A pesar de lo anterior existen algunas ventajas en el uso de la programación declarativa. Entre las ventajas se destaca que la solución de un problema se puede realizar con un nivel de abstracción considerablemente alto, sin entrar en detalles de implementación irrelevantes, lo que hace a las soluciones más fácil de entender por las personas. La resolución de problemas complejos es resuelta por el intérprete a partir de la declaración de las condiciones dadas.

La programación declarativa es muy usada en la resolución de problemas relacionados con inteligencia artificial, bases de datos, configuración, y comunicación entre procesos; sin embargo, ningún leguaje declarativo se aproxima en popularidad a los lenguajes imperativos.

4. EJEMPLOS DE LENGUAJES DECLARATIVOS

Algunos lenguajes declarativos que se pueden mencionar son:

· PROLOG
· SQL
· HTML
· WSDL (Web Services Description Language)
· XML Stylesheet Language for Transformation

LENGUAJES DE PROGRAMACI?N IMPERATIVOS

En ciencias de la computaci?n se llama lenguajes imperativos a aquellos en los cuales se le ordena a la computadora c?mo realizar una tarea siguiendo una serie de pasos o instrucciones, por ejemplo:

Paso 1, solicitar n?mero.
Paso 2, multiplicar n?mero por dos.
Paso 3, imprimir resultado de la operaci?n.
Paso 4, 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 programaci?n imperativa, se tiene un conjunto de instrucciones que le indican al computador c?mo realizar una tarea.

Los lenguajes imperativos se basan en comandos u ?rdenes que se le dan a la computadora para que haga algo, con el fin de organizar o cambiar valores en ciertas partes de la memoria.

La ejecuci?n de estos comandos se realiza, en la mayor parte de ellos, secuencialmente, es decir, hasta que un comando no ha sido ejecutado no se lee el siguiente.

Seg?n el dominio, o mejor dicho con el prop?sito que se utiliza el programa, se puede hablar de lenguajes de dominio espec?fico y de dominio general.

1. LENGUAJES IMPERATIVOS PROCEDURALES

En los lenguajes tradicionales o procedurales, es la aplicaci?n quien controla qu? porciones de c?digo se ejecuta, y la secuencia en que este se ejecuta. La ejecuci?n de la aplicaci?n se inicia con la primera l?nea de c?digo, y sigue una ruta predefinida a trav?s de la aplicaci?n, llamando procedimientos seg?n sea necesario.

Los lenguajes procedurales est?n fundamentados en la utilizaci?n de variables para almacenar valores y en la realizaci?n de operaciones con los datos almacenados. Algunos ejemplos son: FORTRAN, PASCAL, C, ADA, ALGOL, c

En este tipo de lenguajes, la arquitectura consta de una secuencia de celdas, llamadas memoria, en las cuales se pueden guardar en forma codificada, lo mismo datos que instrucciones; y de un procesador, el cual es capaz de ejecutar de manera secuencial una serie de operaciones, principalmente aritm?ticas y booleanas, llamadas comandos. En general, un lenguaje procedural ofrece al programador conceptos que se traducen de forma natural al modelo de la m?quina. El programador tiene que traducir la soluci?n abstracta del problema a t?rminos muy primitivos, cercanos a la m?quina.

Con un lenguaje procedural el usuario (normalmente ser? un programador) especifica qu? datos se necesitan y c?mo obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la informaci?n requerida. Estos lenguajes acceden a un registro, lo procesan y bas?ndose en los resultados obtenidos, acceden a otro registro, que tambi?n deben procesar. As? se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados. Las sentencias de un lenguaje procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual.

2. ALGUNOS LENGUAJES IMPERATIVOS

Algunos lenguajes de programaci?n imperativos que se pueden mencionar son:

? BASIC
? C
? C++
? Java
? C#
? PHP
? Perl

5.-TRADUCTORES

De bajo nivel:
Ensamblador: Es el traductor de los programas escritos en ensamblador y genera código máquina; es decir directamente ejecutable por la maquina.

De alto nivel:
Interprete: Este tipo de traductores solo interpretan la información.

Es decir:

- No generan un fichero ejecutable.
- El programa es controlado (simulado) por el interprete.

Compilador: Lo que se realiza es; se toma el fichero con el código fuente y se crea un programa ejecutable que se controla con el S. O.

Es decir:

- Se parte del código fuente y se genera un fichero ejecutable. O Ser enlazable por necesitar un “linkador”.
- Los errores se detectan en el proceso de compilación.
- Se necesita el compilador propio para el código fuente.

Proceso de compilación:

PARTES DE LA ETAPA DE ANÁLISIS:

PARTES DE LA ETAPA DE SINTESIS:

Bibliografía:

Joyanes, L. Zahonero, I. (2002), Programación en C.
http://www.frt.utn.edu.ar/sistemas/paradigmas/page22.html
http://www.ilustrados.com/publicaciones/EEVZuAVVFyqAegcYul.php
Alberto prieto, Antonio Lloris, Juan Carlos Torres. (2002),
Introducción a la informática 2ª edición.