LENGUAJES DE PROGRAMACIÓN

MODULO I: INTRODUCCIÓN

ÁNGEL LUIS ÁLVAREZ MARTÍN
DIEGO RANILLA LORENZO
IVAN UNZUETA GALLEGO
MARIO GONZALEZ SILVA

PROGRAMACION I
INGENIERIA TEC. INFORMÁTICA DE GESTIÓN
ESCUELA POLITÉCNICA SUPERIOR DE ZAMORA

INDICE

BIBLIOGRAFIA

Paginas de Internet :

http://es.wikipedia.org

http://www.monografias.com

http://www.alegsaonline.com/art.php?id=13

Bibliografía escrita:

Joyanes Aguilat, L. (2003), Fundamentos de programación. Algoritmos, estructuras de datos y objetos. McGraw Hill

CONCEPTO DE LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.
Aunque muchas veces se usa lenguaje de programación y lenguaje informático como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el HTML.

Un lenguaje de programación permite a un programador especificar de *manera precisa*: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural, tal como sucede con el lenguaje lexico.

Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, es decir, ser traducido al lenguaje de máquina, o ser interpretado para que pueda ser ejecutado por el ordenador. También existen lenguajes de scripting que son ejecutados a través de un intérprete y no necesitan compilación.

HISTORIA Y EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

En esta parte nos vemos obligados a ser breves ya que podriamos realizar un trabajo bastante extenso ya que es muy grande la cantidad de lenguajes de programa ción que existen.

Para hacernos una idea mostramos esta pequeña lista de lenguajes con su año de creación y cometido:

1900s BINARIO Bool primer lenguaje

1946 Plankalkul Konrad Zuse creado para jugar al ajedrez

1949 Short Code lenguaje traducido a mano

1950 ASM (ensamblador) lenguaje ensamblador

1951 A-0 GRACE HOPPER fue el primer compilador

1952 AUTOCODE Alick E. Glennie compilador muy rudimentario

1956 FORTRAN IBM sistema de TRAducción de FORmulas matemáticas

1956 COBOL Compilador

1958 ALGOL 58

1960 LISP Interprete orientado a la Inteligencia Artificial

1961 FORTRAN IV IBM sistema de TRAducción de FORmulas matemáticas

1961 COBOL 61 Extendido

1960 ALGOL 60 Revisado

1964 PASCAL Niklaus Wirth programacion estructurada

1964 BASIC Universidad de Dartmouth (california) Beginners All Purpose Symbolic Instruction Code

1965 SNOBOL

1965 APL solo anotacion

1965 COBOL 65

1966 PL/I

1966 FORTRAN 66 IBM sistema de TRAducción de FORmulas matemáticas

1967 SIMULA 67

1968 ALGOL 68

1968 SNOBOL4

1970s GW-BASIC antiguo y clasico BASIC

1970 APL/360

1972 SMALLTALK Centro de Investigación de Xerox en Palo Alto pequeño y rapido

1972 C Laboratorios Bell lenguaje con tipos

1974 COBOL 74

1975 PL /I Lenguaje sencillo

1977 FORTRAN 77 IBM sistema de TRAducción de FORmulas matemáticas

1980s SMALLTALK/V Digitalk pequeño y rapido

1980 C con clases Laboratorios Bell lenguaje con clases

1981 PROLOG Ministerio Japonés de Comercio Internacional e Industria (MITI) Lenguaje estandar para la Inteligencia Artificial

1982 ADA Ministerio de Defensa de los EE.UU lenguaje muy seguro

1984 C++ AT&T Bell Laboratories (Bjarne Stroustrup) compilador

1985 CLIPPER compilador para bases de datos

En cuanto a la evolución, no existen unas fases muy definidas ya que siempre a habido cierta coexistencia entre los diferentes tipos de lenguajes. Pero vamos a permitirnos hacer una breve reseña a ciertas evoluciones como:

1990-95 Las bases de datos relacionales.

Todos los lenguajes Xbase se basaban en el concepto de bases de datos relacionales, es decir la agrupación de la información en forma de tablas, denominadas campos y registros, cada uno de ellos preformateados para recibir cierto tipo de dato (ej: fechas, caracteres, números, valores lógicos, etc.); pudiendo "unir" diferentes bases por medio de campos comunes.

1995-2000 La orientación a objetos.

A medida que los años van pasando el concepto de Bases relacionales empieza a decaer relativamente, surge entonces una variante que se aplica a todos los lenguajes: La orientación a objetos. Ya no solo se habla de programación

estructurada, sino que los módulos de programación son vistos como objetos, las estructuras representan objetos y/o funciones que se adaptan en forma general a procesos específicos es la maximización de la programación modular.

2000 y más allá Los lenguajes visuales

Con al llegada de Windows todo es Visual, todo es iconos, todo es botones, todo es Ventanas. Para programar en lenguajes visuales, primero hay que comprender lo que es Windows. La forma de programar los sistemas evolucionó radicalmente. Con Windows es preciso programar conservando las convenciones del mismo, guardando sus características y funcionalidades. Los sistemas hechos para Windows, deben ser tan Windows como el propio sistema operativo

PARADIGMAS DE PROGRAMACIÓN

Definición:

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc.

Los paradigmas pueden ser considerados como patrones de pensamiento para la resolución de problemas. Desde luego siempre teniendo en cuenta los lenguajes de programación, según nuestro interés de estudio.

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.

Hay multitud de ellos atendiendo a alguna particularidad metodológica o funcional

Cuando un lenguaje refleja bien un paradigma particular, se dice que soporta el paradigma, y en la práctica un lenguaje que soporta correctamente un paradigma, es difícil distinguirlo del propio paradigma, por lo que se identifica con él.

Tipos de paradigmas de programación:

Podemos clasificar los paradigmas de programación en:

Paradigma imperativo, heuristico, concurrente, funcional, lógico, paradigma basado en objetos.

Paradigma 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.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la "transparencia referencial" (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de "efectos laterales".

El objetivo es conseguir lenguajes expresivos y "matemáticamente elegantes", en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de "estado" del cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la "reescritura" de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan "definiciones dirigidas".

A este tipo de paradigma de programación se le suele llamar algorítmico

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Algunos de los lenguajes imperativos son:

- C
- C++
- C#
- Basic
- Java
- Perl

Paradigma heuristico:

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.

Se puede definir como "aquel tipo de programación computacional que aplica para la resolución de problemas reglas de buena lógica (reglas del pulgar). denominadas heurísticas, las cuales proporcionan entre varios cursos de acción uno que presenta visos de ser el más prometedor, pero no garantiza necesariamente el curso de acción más efectivo."

La Programación Heurística implica una forma de modelizar el problema en lo que respecta a la representación de su estructura, estrategias de búsqueda y métodos de resolución, que configuran el Para

Este tipo de programación se aplica con mayor intensidad en el campo de la Inteligencia Artificial (IA), y en especial, en el de la Ingeniería del Conocimientodigma Heurístico.

La Programación Heurística se presenta y utiliza desde diferentes puntos de vista:

- Como técnica de búsqueda para la obtención de metas en problemas no algorítmicos, o con algoritmos que generan explosión combinatoria
- Como un método aproximado de resolución de problemas utilizando funciones de evaluación de tipo heurístico
- Como método de poda para estrategias de programas que juegan, aunque estos métodos no son realmente heurísticos

Paradigma funcional:

Sus orígenes provienen del Cálculo Lambda (o λ-cálculo), una teoría matemática elaborada por Alonzo Church como apoyo a sus estudios sobre computabilidad. Un lenguaje funcional es, a grandes rasgos, un *azúcar sintáctico* del Cálculo Lambda.

El paradigma funcional está basado en el modelo matemático de composición funcional. En este modelo, el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado.

No existe el concepto de celda de memoria que es asignada o modificada. Más bien, existen valores intermedios que son el resultado de cálculos anteriores y las entradas a cálculos subsiguientes. Tampoco existen sentencias imperativas y todas las funciones tienen transparencia referencial.

La programación funcional incorpora el concepto de función como objeto de primera clase, lo que significa que las funciones se pueden tratar como datos (pueden pasar como parámetros, calculadas y devueltas como valores normales, y mezcladas en el cálculo con otras formas de datos).

En este paradigma el informático concibe la solución como una composición de funciones

La forma en que se especifican las funciones puede variar. Se pueden especificar procedimentalmente o matemáticamente mediante su definición, sin secuencia de control.

Un lenguaje funcional es el Lisp

Paradigma lógico:

La Programación Lógica es un Paradigma de Programación basado en la Lógica.

Los programas construidos un lenguaje lógico están construidos únicamente por expresiones lógicas, es decir, que son ciertas o falsas, en oposición a un expresión interrogativa (una pregunta) o expresiones imperativas (una orden). Un ejemplo de lenguaje lógico es Prolog (**Pro**gramación **lóg**ica).

Prolog, proveniente del inglés *Programming in Logic*, es un lenguaje lógico bastante popular en el medio de investigación en Inteligencia Artificial. Prolog es un lenguaje muy diferente, tanto de los imperativos como Fortran, Pascal, C etc., como de los funcionales como Lisp. En todos los mencionados, las instrucciones se ejecutan normalmente en orden secuencial, es decir, una a continuación de otra, en el mismo orden en que están escritas, que sólo varía cuando se alcanza una instrucción de control (un bucle, una instrucción condicional o una transferencia).

En Prolog, las cosas son distintas: el orden de ejecución de las instrucciones no tiene nada que ver con el orden en que fueron escritas. Tampoco hay instrucciones de control propiamente dichas. Para trabajar con este lenguaje, un programador debe acostumbrarse a pensar de una manera muy diferente a la que se utiliza en los lenguajes clásicos.

Paradigma basado en objetos:

La programación orientada a objetos (OOP, por las siglas inglesas de Object-Oriented Programming) es una forma de programar que proliferó a partir de los años ochenta.

La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan *estado* (es decir, datos), *comportamiento* (esto es, procedimientos o *métodos*) e identidad (propiedad del objeto que lo diferencia del resto).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos.

Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

Dada esta propiedad de conjunto de una clase de objetos, que al contar con una serie de atributos definitorios, requiere de unos métodos para poder tratarlos (lo que hace que ambos conceptos están íntimamente entrelazados), el programador debe pensar indistintamente en ambos términos, ya que no debe nunca separar o dar mayor importancia a los atributos en favor de los métodos, ni viceversa.

Las principales diferencias entre la programación imperativa y la orientada a objetos son:

- La programación orientada a objetos es más moderna, es una evolución de la programación imperativa que plasma en el diseño de una familia de lenguajes conceptos que existían previamente con algunos nuevos.
- La programación orientada a objetos se basa en lenguajes que soportan sintácticamente la unión entre los tipos abstractos de datos y sus operaciones (a esta unión se la suele llamar clase).
- La programación orientada a objetos incorpora en su entorno de ejecución mecanismos tales como el polimorfismo y el envío de mensajes entre objetos.

Lenguajes orientados a objetos

Entre los lenguajes orientados a objetos destacan los siguientes:

Ada C++ C# VB.NET Clarion Delphi Eiffel Java Lexico (en castellano)

CLASIFICACIONES DE LENGUAJES DE PROGARMACIÓN

En cuanto a su nivel de trabajo:

Lenguaje máquina

El lenguaje máquina es el único que entiende directamente la computadora, utiliza el alfabeto binario que consta de los dos únicos símbolos 0 y 1, denominados bits (abreviatura inglesa de dígitos binarios). Fue el primer lenguaje utilizado en la programación de computadoras, pero dejó 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.

Lenguaje de bajo nivel

Un lenguaje de programación de **bajo nivel** es el que proporciona poca o ninguna abstracción del microprocesador de un ordenador. Consecuentemente es fácilmente trasladado a lenguaje de máquina.En general se utiliza este tipo de lenguaje para programar controladores (drivers).

La programación en un lenguaje de bajo nivel como el lenguaje de la máquina o el lenguaje simbólico tiene ciertas ventajas:

1. Mayor adaptación al equipo.
2. Posibilidad de obtener la máxima velocidad con mínimo uso de memoria.

Pero también tiene importantes inconvenientes:

Imposibilidad de escribir código independiente de la máquina.
Mayor dificultad en la programación y en la comprensión de los programas.

Características

- Se trabaja a nivel de Microinstrucciones, es decir, su programación es al más fino detalle.
- Está orientado a la máquina.

Lenguaje de alto nivel

Los lenguajes de alto nivel logran la independencia del tipo de máquina y se aproximan al lenguaje natural. 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.

Los lenguajes de alto nivel, también denominados lenguajes evolucionados, surgen con posterioridad a los anteriores, con los siguientes objetivos, entre otros:

- Lograr independencia de la máquina, pudiendo utilizar un mismo programa en diferentes equipos con la única condición de disponer de un programa traductor o compilador, que lo suministra el fabricante, para obtener el programa ejecutable en lenguaje binario de la máquina que se trate. Además, no se necesita conocer el hardware específico de dicha máquina.
- 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 máquina, ya que se utilizan palabras (en inglés) en lugar de cadenas de símbolos sin ningún significado aparente.
- Incluir rutinas de uso frecuente como son las de entrada/salida, funciones matemáticas, manejo de tablas, etc, que figuran en una especie de librería del lenguaje, de tal manera que se pueden utilizar siempre que se quieran sin necesidad de programarlas cada vez.

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 (FORTRAN, LISP, ALGOL, COBOL, APL, SNOBOL, PROLOG, MODULA2, ALGOL68, PASCAL, SIMULA67, ADA, C++, LIS, EUCLID, BASIC), además de las diferentes versiones o dialectos que se han desarrollado de algunos de ellos.

TRADUCTORES

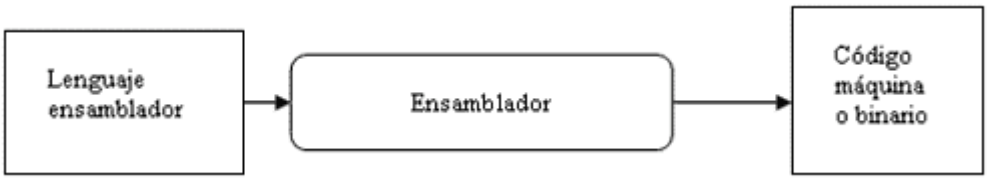
Explicamos el proceso más importante a la hora de hacer funcionar un programa realizado en lenguajes de alto y medio nivel.

Cuando programamos en lenguajes de alto nivel, lo que estamos haciendo en realidad es el código fuente de ese programa.

Este código fuente debe ser traducido a binario para que las instrucciones que contienen puedan ser entendidas y ejecutadas por la máquina.
Para esto existe un programa encargado de realizar la traducción, llamado traductor del lenguaje.

Estos traductores pueden ser de dos tipos:

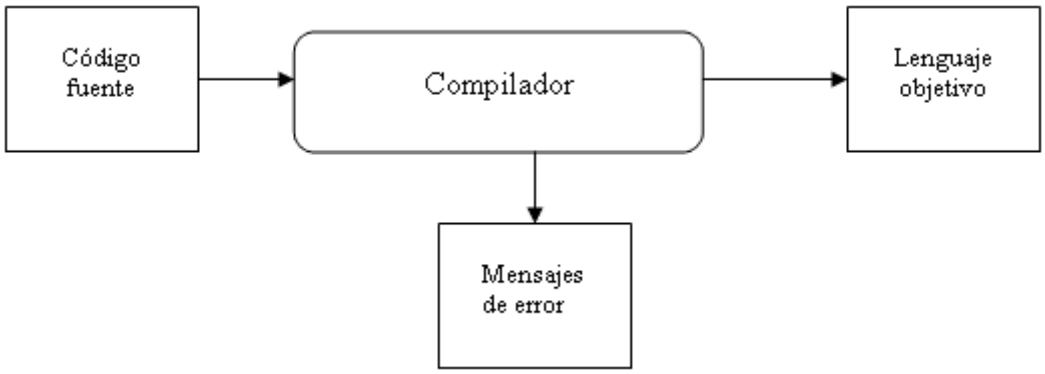
Ensambladores



Son los encargados de traducir los programas escritos en lenguaje ensamblador a lenguaje máquina.

Compiladores

Son programas que leen el código fuente y lo traducen o convierten a otro lenguaje. Estos programas te muestran los errores existentes en el código fuente.



Etapas del proceso de compilación:

Edición. Esta fase consiste en escribir el programa empleando algún lenguaje y un editor. Como resultado nos dará el código fuente de nuestro programa.
Compilación. En esta fase se traduce el código fuente obtenido en la fase anterior a código máquina. Si no se produce ningún error se obtiene el código objeto. En caso de

errores el compilador los mostraría para ayudarnos a corregirlos y se procedería a su compilación de nuevo, una vez corregidos.
Linkado. Esta fase consiste en unir el archivo generado en la fase dos con determinadas rutinas internas del lenguaje, obteniendo el programa ejecutable.
Existen dos tipos de linkados:

linkado estático: Los binarios de las librerías se añaden a nuestros binarios compilados generando el archivo ejecutable.
Linkado dinámico: no se añaden las librerías a nuestro binario sino que hará que se carguen en memoria las librerías que en ese momento se necesiten.

Una vez traducido, compilado y linkado el archivo esta listo para su ejecución donde también podrán surgir problemas y fallos, para los cuales tendríamos que volver a realizar todo el proceso anteriormente citado, de modo que puedan ser corregidos.

Por este motivo es importante realizar numerosas pruebas en tiempo de ejecución antes de presentar el programa al cliente.

Otro sistema para la ejecución de nuestro código fuente es mediante el uso de intérpretes (estos no se encontrarían dentro de los traductores).

Intérpretes

Los intérpretes realizan la traducción y ejecución de forma simultanea, es decir, un intérprete lee el código fuente y lo va ejecutando al mismo tiempo.

Las diferencias entre un compilador y un intérprete básicamente son:

- Un programa compilado puede funcionar por si solo mientras que un código traducido por un intérprete no puede funcionar sin éste.
- Un programa traducido por un intérprete puede ser ejecutado en cualquier máquina ya que, cada vez que se ejecuta el intérprete, tiene que compilarlo.
- Un archivo compilado es mucho más rápido que uno interpretado.