PANORAMA DE LOS LENGUAJES DE PROGRAMACION

INTRODUCCION

Toda computadora digital trabaja internamente con señales eléctricas que representan unos y ceros. Ellos forman un CÓDIGO BINARIO que puede representar cualquier número, letra o símbolo. Así se expresa el único lenguaje que es capaz de interpretar la computadora y que se denomina LENGUAJE MAQUINA. Como variante del lenguaje máquina, se utiliza la escritura hexadecimal, que resulta más práctica porque los números así representados tienen menos cantidad de dígitos, con lo cual se escriben más rápidamente y se cometen menos errores. También podemos ver, por ejemplo cuando mostramos un programa ejecutable por pantalla, que el mismo está representado por varios símbolos en un código llamado ASCCI. Allí se ha utilizado este código con el fin de abreviar la representación del programa, pero internamente, es decir en la memoria, el programa se halla escrito en el lenguaje de unos y ceros.

NIVELES DE LENGUAJE

¿Qué entendemos por lenguaje en una computadora? Simplemente un conjunto de símbolos y reglas utilizadas para intercambiar informaciones y comunicarnos con la computadora.
Se ha tratado de elaborar lenguajes de computación lo más parecidos posible al lenguaje humano. Como los desarrollos de este tipo se realizan en los países de habla inglesa, los lenguajes de computación tienen términos en inglés (por ejemplo el Pascal, Basic).
Por lo tanto, al construirse un nuevo lenguaje resulta imprescindible que, mediante algún procedimiento automático, el mismo sea traducido al único lenguaje que conoce la computadora, el lenguaje máquina.
Los lenguajes se clasifican en tres grandes grupos:

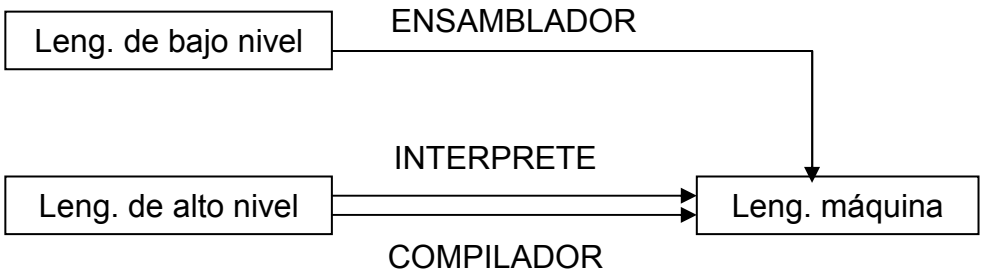
- LENGUAJE MAQUINA: ya lo mencionamos. La ventaja fundamental de los programas escritos en este lenguaje es que resultan mucho más rápidos y requieren menos memoria que los programas escritos en alto nivel. Dependen fuertemente del hardware.

- LENGUAJE DE BAJO NIVEL: el más conocido es el ASSEMBLER, que permite escribir un programa utilizando instrucciones nemotécnicas, en lugar de binarias. Es de hacer notar que, aprender a programar en bajo nivel requiere un mayor conocimiento de la estructura interna del microprocesador y de su modo de operar. El Assembler es, por eso, muy dependiente del hardware y por lo tanto no es portable.

- LENGUAJE DE ALTO NIVEL: Más cercanos al lenguaje humano (inglés) como el Basic, Pascal, etc. Hacen, por eso, más fácil la programación. Además son portables.

TRADUCCION DEL LENGUAJE

Una vez redactado un programa en alto o bajo nivel (llamado programa fuente), el mismo debe ser traducido al lenguaje máquina, originando otro archivo llamado programa objeto.
Para realizar la traducción se utilizan "diccionarios", que son programas grabados en el disco, de tres tipos:



- ENSAMBLADOR: diccionario utilizado para pasar un programa fuente de bajo nivel (Assembler) a un programa objeto (lenguaje máquina). Cada instrucción assembler origina una, dos o tres instrucciones en lenguaje máquina.

- INTERPRETE: diccionario utilizado para pasar un programa fuente de alto nivel a un programa objeto. Cada instrucción del programa fuente origina generalmente más de una instrucción binaria. Con cada instrucción el interprete hace lo siguiente:
a) examina la instrucción verificando si existen errores de sintaxis.
b) traduce la instrucción al lenguaje máquina.
c) realiza la operación indicada por la instrucción, es decir, la ejecuta. Luego sigue con la siguiente instrucción.
d) No almacena el programa objeto por lo que, cada vez que se desee correr el programa fuente, se realizará la traducción del mismo.

- COMPILADOR: diccionario utilizado para pasar un programa fuente de alto nivel a un programa objeto como lo hace el Intérprete, pero opera de un modo diferente:
a) examina la instrucción verificando si existen errores de sintaxis.
b) traduce la instrucción al lenguaje máquina.
c) hace lo mismo con cada instrucción hasta traducir todo el programa, sólo entonces se podrá ejecutar y verificar si hay errores de programación.
d) Almacena el programa máquina generado, en el disco. Este programa resultará más rápido al ejecutarlo que el equivalente obtenido con un intérprete.

La ventaja del Intérprete es que, en caso de existir errores de programación, estos son rápidamente detectados porque cada instrucción es traducida y luego ejecutada, una a una. En cambio en el Compilador, la ejecución sólo puede realizarse después que se ha traducido todo el programa, haciendo más lenta la tarea de corrección (debbugging) del mismo.
La ventaja del Compilador reside en que, una vez puesto a punto el programa (es decir, después de haber corregido todos los errores), es más rápido en su ejecución puesto que todo el programa ya traducido se encuentra grabado en el disco, en cambio el Intérprete deberá traducir el programa cada vez que se desee utilizarlo.

LENGUAJES DE ALTO NIVEL

PARADIGMAS DE PROGRAMACION

Esta sección tiene por objeto configurar un panorama conceptual de los distintos lenguajes de programación de alto nivel.
Un paradigma de programación es un modelo, forma o manera de programar. Los tipos básicos de programación son:

- **Paradigma algorítmico:** es el que utilizan los lenguajes más difundidos (Basic, Pascal, C). Un algoritmo es un método para resolver un problema, en el cual los pasos de la solución están secuenciados con precisión. La programación algorítmica utiliza cuatro tipos de estructuras de control, que encaminan el flujo del programa:
  - secuenciales: asignaciones, cálculo, etc.
  - selección: condicionales simples y múltiples: si...entonces, etc.
  - repetición: hacer mientras, etc.
  - salto: goto.

  Los lenguajes que utiliza este paradigma son: Basic, Fortran, Cobol.
  La programación algorítmica evolucionó hacia la PROGRAMACIÓN ESTRUCTURADA (ver más abajo). Esta técnica se creó con el objeto de facilitar la depuración de los programas. Utiliza tres ideas fundamentales:
  - elimina el salto, utilizando sólo las tres primeras estructuras señaladas.
  - programación modular: el problema a resolver se divide en módulos.
  - cada módulo tiene un diseño descendente (top-down): cada etapa de diseño resuelve el problema en un nivel mayor de detalle.

  Entre los lenguajes estructurados tenemos: QuickBasic, Pascal, C.

- **Programación orientada a objetos:** La POO se funda en la programación estructurada, pero incorpora ciertas características que facilitan la reutilización del software. La programación estructurada define estructuras que almacenan los datos y procedimientos que los manipulan. En la POO, en cambio, ambos (datos y procedimientos) se combinan dentro de un objeto. Los objetos son entidades que podemos distinguir de manera clara y definida (una persona, un botón, una computadora, etc.).
  Los objetos tienen tres aspectos a considerar:
  - propiedades: características observables del objeto (ubicación, forma, color, etc.)
  - métodos: acciones que el objeto puede realizar, habitualmente modifican las propiedades del objeto.
  - eventos: estímulo que un objeto ejerce sobre otro objeto (por ejemplo: un click del objeto mouse sobre un objeto botón).

  Los lenguajes que la utilizan son: Smalltalk, C++, Java. En otros lenguajes, los objetos se incorporan preprogramados y pueden ser usados por los programadores en el desarrollo de aplicaciones.

- **Programación visual:** en la programación estructurada, cada elemento que el programa muestra en pantalla (por ejemplo: un menú) debe ser creado mediante varias líneas de código. En programación visual, en cambio, cada elemento se dibuja en la pantalla como en los programas de diseño gráfico y luego, para establecer sus funciones, se le adjudica un bloque de código (procedimiento) escrito según los principios de programación estructurada.

- **Programación dirigida por eventos:** en la programación estructurada, las secuencias del programa están dadas por las estructuras de control establecidas en el algoritmo. En la programación dirigida por eventos, en cambio, la secuencia que sigue en programa se da como respuesta a los eventos que ejecuta el usuario.

Estos tres últimos principios de programación está presentes en los lenguajes más utilizados en la actualidad: Visual Basic (evolución visual del QuickBasic), Delphi (evolución visual del Pascal), Visual C++, etc. El Paradigma de objetos y el Paradigma Visual son además transversales a los demás paradigmas ya que hay versiones de Prolog y de Lisp Visuales y orientadas a objetos.

- **Paradigma lógico:** La base de la programación lógica es la de codificar el problema y no su solución (como lo hacemos en el caso de la programación algorítmica). Su implementación se basa en la interpretación del cálculo de predicados de la lógica tradicional, restringido a cláusulas de Horn. Por ejemplo, una instrucción en un paradigma lógico tendría este aspecto:
  si ***Premisa1*** y ***Premisa2*** y ... y ***PremisaN*** entonces ***Conclusión***.
  Cuando se habla de cálculo de predicados se habla de las oraciones de tipo "***si... entonces***". Los predicados que conforman el programa tienen la forma:
  Predicado(Parámetro1, ..., ParámetroN).
  Las cláusulas de Horn son predicados que tienen una y sólo una conclusión.
  Ejemplo: Para decir:
  Si A es padre de B y B es madre de C entonces A es abuelo de C.
  En PROLOG se escribe:
  abuelo(A,C):- padre(A,B), madre(B,C).
  El lenguaje Prolog fue creado para desarrollar un tipo especial de aplicaciones llamadas SISTEMAS EXPERTOS. Son estudiados por una rama de la informática conocida como Inteligencia Artificial, en la cual se procura la solución de problemas por métodos heurísticos. Los métodos heurísticos (a diferencia de los métodos algorítmicos) no arriban a soluciones óptimas, sino que la solución tiene un margen probable de acierto.

- **Paradigma funcional:** El lenguaje más conocido es el LISP. La teoría subyacente es la de las funciones matemáticas. En principio un programa escrito bajo este paradigma no tendría ninguna "instrucción" en particular, sino que estaría formado por una o más funciones.
  Se recordará que las funciones son relaciones entre elementos de conjuntos, que toman valores pertenecientes a el o los conjuntos de entrada y devuelven un valor perteneciente al conjunto de salida. Por ejemplo, matemáticamente se define la función suma entre dos números naturales como:
  suma: N x N -> N
  lo que indica que esta función tomará dos números del conjunto de los naturales y devolverá un número perteneciente al mismo conjunto. En LISP. por ejemplo, se escribirá:
  (suma N N)
  aquí hay tres elementos entre paréntesis: el primero es el nombre de la función, los dos siguientes son los valores de entrada; el resultado será el valor de toda la expresión entre paréntesis, que a su vez puede usarse como entrada para otra función. De esta manera las funciones se van "anidando" una con otra. Por ejemplo, la instrucción:
  (resta (suma 2 2) (suma 3 1))
  equivale a la expresión matemática:
  (2 + 2) – (3 + 1)

El lenguaje Lisp está especialmente diseñado para el manejo de un tipo complejo de estructura de datos llamado LISTAS.

PROGRAMACIÓN ESTRUCTURADA

Es una técnica en la cual la estructura de un programa, esto es, la interpelación de sus partes, se realiza tan claramente como es posible mediante el uso de tres estructuras lógicas de control:

a) Secuencia: Sucesión simple de dos o mas operaciones.
b) Selección: bifurcación condicional de una o mas operaciones.
c) Interacción: Repetición de una operación mientras se cumple una condición.

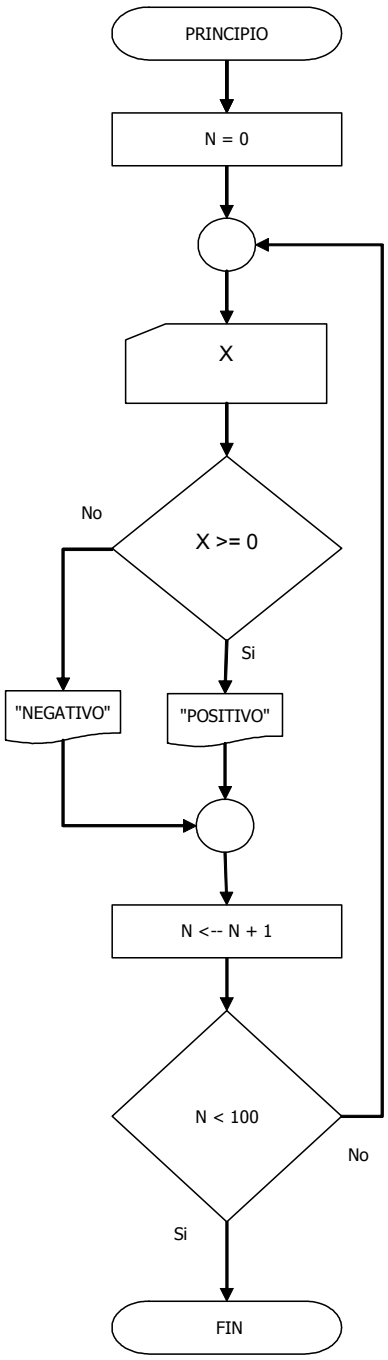
Estos tres tipos de estructuras lógicas de control pueden ser combinadas para producir programas que manejen cualquier tarea de procesamiento de información.

Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación. Esto es importante debido a que, es mucho más fácil comprender completamente el trabajo que realiza una función determinada, si todas las instrucciones que influyen en su acción están físicamente contiguas y encerradas por un bloque.

La facilidad de comprensión del contenido de un programa puede facilitar el chequeo de la codificación y reducir el tiempo de prueba y depuración de programas.
La facilidad de lectura, de comienzo a fin, es una consecuencia de utilizar solamente tres estructuras de control y de eliminar la instrucción de desvío de flujo de control: GOTO.

Se habla de la programación estructurada como una técnica de programación que no utiliza GOTO. El problema del uso de esta instrucción es que los saltos en el flujo de control pueden realizarse en cualquier dirección, hacia arriba o hacia abajo, y pueden entrecruzarse. Por tal motivo se hace muy difícil de seguir el flujo del programa. Si hubiese algún error en la lógica del programa, sus consecuencias afectarían a partes muy alejadas del punto en que se produce el error, debido a las instrucciones de salto. Por lo cual el error es muy difícil de detectar.

A continuación se muestra un ejemplo de programación NO estructurada para remarcar las dificultades que presenta el seguimiento del flujo del algoritmo:



EJEMPLO DE PROGRAMACION NO ESTRUCTURADA

Nº DE LINEA	INSTRUCCIÓN
1	N = 0
2	INGRESAR X
3	SI X >= 0 **SALTAR A 6**
4	ESCRIBIR "NEGATIVO"
5	**SALTAR A 7**
6	ESCRIBIR "POSITIVO"
7	N = N + 1
8	SI N < 100 **SALTAR A 2**
9	FIN

INSTRUCCIONES **SI** NO ESTRUCTURADAS

SALTAR INCONDICIONAL