CAPÍTULO V: PROGRAMACIÓN DE COMPUTADORAS

INTRODUCCIÓN

Para que un ordenador funcione es necesario utilizar programas. El programa indica al computador qué tiene que hacer, y éste únicamente realiza aquellas operaciones que aquel incluye. La gran oferta de Software existente en el mercado, es más que suficiente para poder encontrar la aplicación que se adapte a nuestras necesidades. Pero evidentemente estas aplicaciones han sido realizadas por alguien. Es aquí cuando estamos hablando de programación.

Debemos entender por lenguaje de programación, la herramienta mediante la cual podremos comunicarnos con el Hardware proporcionándole las órdenes deseadas para llevar a cabo un determinado proceso.

Un programa y sus sentencias se construyen o redactan con unos símbolos, y de acuerdo con unas reglas, que constituyen la gramática del lenguaje de programación.

LENGUAJES DE PROGRAMACIÓN.

Un lenguaje de programación es un conjunto de símbolos y de reglas para combinarlos, que se usan para expresar algoritmos. Los lenguajes de programación, al igual que los lenguajes que usamos para comunicarnos, poseen un léxico (vocabulario o conjunto de símbolos permitidos), una sintaxis, que indica como realizar las construcciones del lenguaje y una semántica, que determina el significado de cada construcción correcta.

Dependiendo de la función que deseemos llevar a cabo, deberemos optar por el lenguaje más apropiado.

Lenguaje o código máquina.

Se trata del primer lenguaje utilizado en programación de ordenadores, actualmente en desuso por la dificultad que ofrecía a la hora de corregir errores, o realizar modificaciones. Es el único que comprende directamente el ordenador, ya que se utiliza el código binario (0 y 1).

Lenguaje ensamblador.

Se trata del sustituto del lenguaje máquina y se basa en instrucciones equivalentes a código máquina pero con términos nemotécnicos. Es por ello que presenta la misma complejidad, si bien, ocupa muy poca memoria y el tiempo de ejecución es sumamente más rápido respecto a los lenguajes de alto nivel.

Lenguajes de alto nivel.

Bajo este término, se engloban los lenguajes de programación aptos para generar aplicaciones de gestión, científicas, específicas o de propósito general, y poseen unas ventajas respecto a los descritos anteriormente. Los lenguajes de programación, o lenguajes de alto nivel, están específicamente diseñados para programar computadoras. Entre sus características fundamentales se tiene que:

✍ Son independientes de la arquitectura física del computador, es decir, permiten utilizar los mismos programas en diferentes computadoras, con distinto lenguaje máquina (portabilidad).

✍ Normalmente, una sentencia en un lenguaje de alto nivel da lugar, al ser traducida a varias instrucciones en lenguaje máquina.

Lenguaje de alto nivel FORTRAN	Lenguaje ensamblador	Lenguaje máquina

A = B + C	LDA 0,4,3
	LDA 2,3,3
	ADD 2,0
	STA 0,5,3
		021404
		031403
		143000
		041405

✍ Utilizan notaciones cercanas a las habituales en el ámbito en que se usan. Con estos lenguajes las operaciones se expresan con sentencias o frases muy parecidas al lenguaje matemático o al lenguaje natural. Usualmente en los lenguajes de programación se utilizan palabras o términos en inglés.

La utilización de conceptos habituales suele implicar las siguientes cualidades:

a. Las instrucciones se expresan por medio de *texto*, conteniendo caracteres alfanuméricos y caracteres especiales (+, =, /, etc...).

b. Se puede asignar un *nombre simbólico* a determinados componentes del programa, para facilitar su comprensión por las personas. En los lenguajes imperativos, el programador puede definir las *variables* que desee, dándoles los nombres que considere oportuno (DEBE, HABER, SALDO, MEDIA, TOTAL, etc...), siendo las reglas para denominación de las mismas muy poco restrictivas. La asignación de memoria para variables y constantes las hace directamente el traductor.

c. Dispone de instrucciones potentes, conteniendo *operadores y funciones* de una gran diversidad: aritméticas (seno, coseno, módulo, etc...), especiales (cambiar un dato de tipo real a entero, por ejemplo), lógicas (comparar la función lógica Y, etc...), de tratamiento de caracteres (buscar una subcadena en una cadena de caracteres,.....), etc.....

d. Pueden incluirse *comentarios* en las líneas de instrucciones, o en líneas específicas de comentarios. Esto facilita la **legibilidad** de los programas, tanto para el propio programador, como para otras personas.

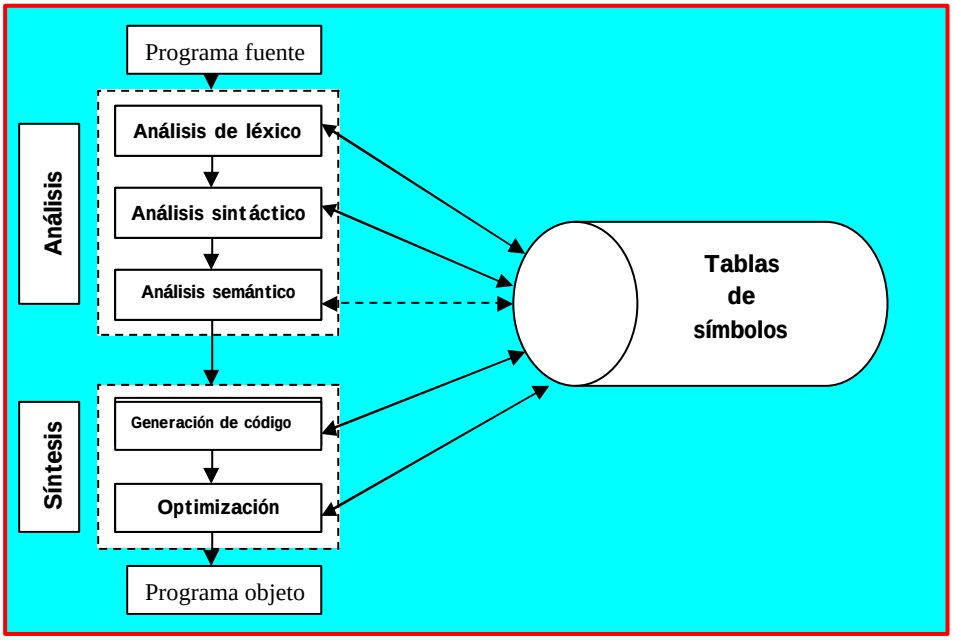
Como consecuencia de este alejamiento de la máquina y acercamiento a las personas, los programas escritos en lenguajes de programación no pueden ser directamente interpretados por el computador, siendo necesario realizar previamente su traducción a lenguaje máquina. Hay dos tipos de traductores de lenguajes de programación: **los compiladores** y **los intérpretes.**

EL PROCESO DE TRADUCCIÓN.

Para facilitar el uso de los ordenadores se han desarrollado lenguajes de programación que permiten utilizar una simbología y una terminología próximas a las utilizadas tradicionalmente en la descripción de problemas. Como el computador puede interpretar y ejecutar únicamente código máquina, existen programas traductores, que traducen programas escritos en lenguajes de programación a lenguaje máquina. Un traductor es un programa que recibe como entrada un texto en un lenguaje de programación concreto y produce, como salida, un texto en lenguaje máquina equivalente. **El programa inicial se denomina programa fuente y el programa obtenido programa objeto**. Este programa fuente lo utiliza el traductor como conjunto de datos que debe procesar, y a partir de él genera como resultado el programa objeto.

La traducción por un compilador (la compilación) consta de dos etapas fundamentales, que a veces no están claramente diferenciadas a lo largo del proceso: la etapa de análisis del programa fuente y la etapa de síntesis del programa objeto. Cada una de estas etapas conlleva la realización de varias fases. El análisis del texto fuente implica la realización de un análisis del léxico, de la sintaxis y de la semántica. La síntesis del programa objeto conduce a la generación de código y su optimización.

La compilación es un proceso complejo y que consume a veces un tiempo muy superior a la propia ejecución del programa. En cualquiera de las fases de análisis, el compilador puede dar mensajes sobre los errores que detecta en el programa fuente, cancelando en ocasiones la compilación, para que el usuario realice las correcciones oportunas en el archivo fuente.



Análisis lexicográfico.

Consiste en descomponer el programa fuente en sus elementos constituyentes o símbolos (tokens). El analizador lexicográfico aísla los símbolos, identifica su tipo y almacena en las tablas de símbolos la información del símbolo que pueda ser necesaria durante el proceso de traducción (figura precedente). Así, por ejemplo, convierte los números o constantes, que en el programa figuran en código E/S (ASCII), a su representación interna (entero, coma flotante, simple precisión), ya que esta información será necesaria a la hora de generar código. El analizador de léxico realiza, además, otras funciones secundarias: Identifica y salta comentarios, identifica, y pasa, espacios en blanco y tabulaciones e informa de posibles errores de léxico.

Análisis sintáctico.

La sintaxis de un lenguaje de programación especifica cómo deben escribirse los programas, mediante un conjunto de reglas de sintaxis o gramática del lenguaje. Un programa es sintácticamente correcto cuando sus estructuras (expresiones, sentencias declarativas, asignaciones, etc...) aparecen en un orden correcto. La función de un analizador sintáctico consiste en analizar las instrucciones de tal forma que sólo admita las construcciones correctas en el lenguaje en que está escrito el programa.

Análisis semántico.

La semántica de un lenguaje de programación es el significado dado a las distintas construcciones sintácticas. El proceso de traducción es, en esencia, la generación de un lenguaje en código máquina con el mismo significado que el código fuente y en este proceso, el significado de las sentencias se obtiene de la identificación sintáctica de las construcciones sintácticas y de la información almacenada en las tablas de símbolos. En los lenguajes de programación, el significado está ligado a la estructura sintáctica de las sentencias y durante la fase de análisis semántico se pueden producir errores, cuando se detectan construcciones “sin un significado correcto”. Por ejemplo, asignar a una variable definida como dato numérico en simple precisión el valor de una variable cadena de caracteres, es semántico incorrecto en algunos compiladores.

Generación y optimización de código.

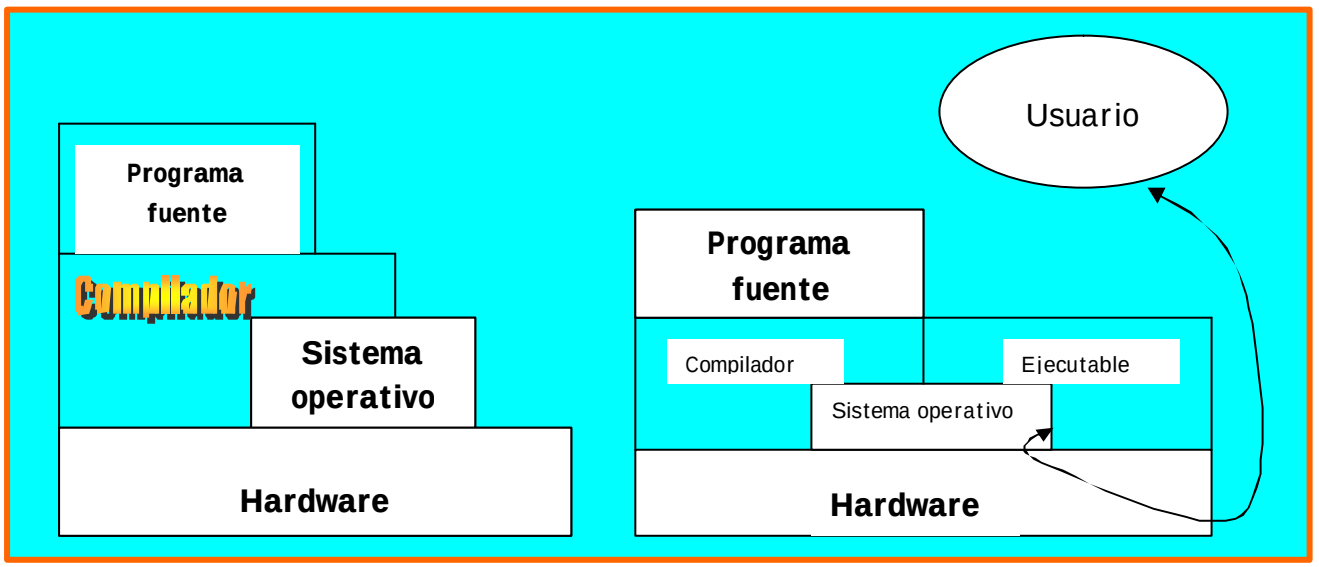
En esta fase se crea un archivo con un código en lenguaje objeto (normalmente lenguaje máquina) con el mismo significado que el texto fuente. El archivo-objeto generado puede ser (dependiendo del compilador) directamente ejecutable, o necesitar otros pasos previos a la ejecución, tales como ensamblado, encadenado y carga. En alguna ocasiones se utiliza un lenguaje intermedio (distinto del código objeto final), con el propósito de facilitar la optimización del código.

En la generación de código intermedio se completan y consultan las tablas generadas en fases anteriores (tablas de símbolos, de constantes, etc...), realizándose también la asignación de memoria a los datos definidos en el programa.

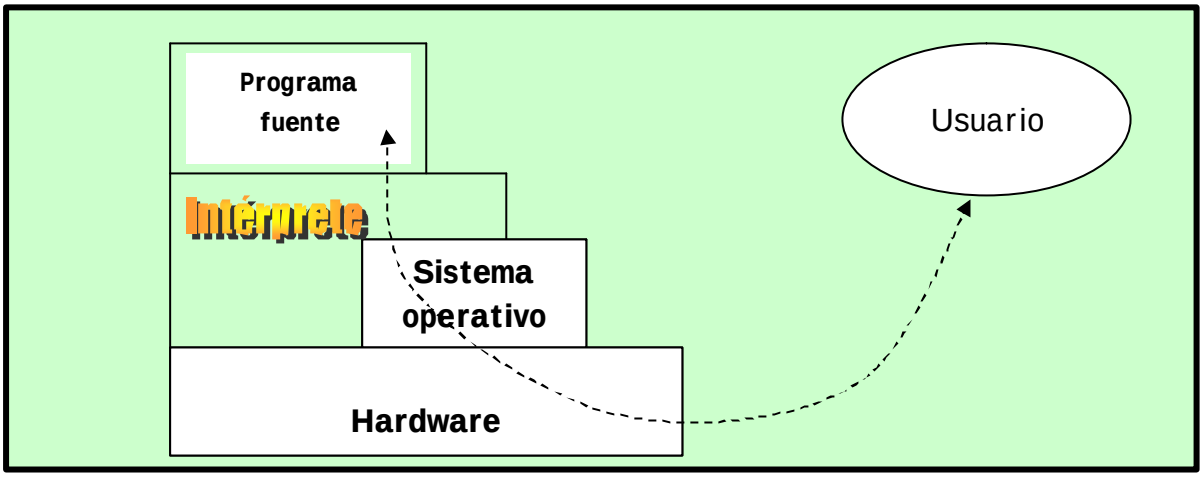
En la fase de optimización se mejora el código intermedio, analizándose el programa objeto globalmente, con lo que la compilación reduce el tiempo de ejecución. Usualmente, las optimizaciones se realizan en varias fases y sobre el código intermedio.

COMPILADORES E INTÉRPRETES.

Un compilador traduce un programa fuente, escrito en un lenguaje de alto nivel, a un programa objeto, escrito en lenguaje ensamblador o máquina. El programa fuente suele estar contenido en un archivo, y el programa objeto puede almacenarse como archivo en memoria masiva para ser procesado posteriormente, sin necesidad de volver a realizar la traducción. Una ver traducido el programa, su ejecución es independiente del compilador, así, por ejemplo, cualquier interacción con el usuario sólo será controlada por el sistema operativo.



Un intérprete hace que un programa fuente escrito en un lenguaje vaya, sentencia a sentencia, traduciéndose y ejecutándose directamente por el computador. El intérprete capta una sentencia fuente, la analiza e interpreta dando lugar a su ejecución inmediata, no creándose, por tanto, un archivo o programa objeto almacenable en memoria masiva para ulteriores ejecuciones. Por tanto, la ejecución del programa está supervisada por el intérprete.



En la práctica el usuario crea un archivo con el lenguaje fuente. Este suele realizarse con un editor específico del propio intérprete del lenguaje. Según se van almacenando las instrucciones simbólicas, se analizan y se producen los mensajes de error correspondientes; así, el usuario puede proceder inmediatamente a su corrección. Una vez creado el archivo fuente el usuario puede dar la orden de ejecución y el intérprete lo ejecuta línea a línea. Siempre el análisis antecede inmediatamente a la ejecución, de forma que:

a. Si una sentencia forma parte de un bucle, se analiza tantas veces como tenga que ejecutarse el bucle (lo cual no sucede en los compiladores, que solamente analizan la sentencia del bucle una sola vez).

b. Las optimizaciones sólo se realizan dentro del contexto de cada sentencia, y no contemplándose el programa o sus estructuras en conjunto.

c. Cada vez que utilicemos un programa tenemos que volver a analizarlo, ya que en la traducción no se genera un archivo objeto que poder guardar en memoria masiva (y utilizarlo en cada ejecución). Con un compilador, aunque la traducción sea más lenta, ésta solo debe realizarse una vez (ya depurado el programa) y cuando deseemos ejecutar un programa, ejecutamos el archivo objeto, que se tradujo previamente.

Los intérpretes, a pesar de los inconvenientes anteriores, son preferibles a los compiladores cuando el número de veces que se va a ejecutar el programa es muy bajo y no hay problemas de velocidad; además, con ellos puede ser más fácil desarrollar programas. Esto es así porque, normalmente, la ejecución de un programa bajo un intérprete puede interrumpirse en cualquier momento para conocer los valores de las distintas variables y la instrucción fuente que acaba de ejecutarse. Cuando esto se hace, se tiene una gran ayuda para la localización de errores en el programa. Con un programa compilador esto no se puede realizar, salvo que el programa se ejecute bajo el control de un programa especial de ayuda denominado depurador (debugger). Además, con un intérprete, cuando se localiza un error sólo debe modificarse este error y volver a ejecutar a partir de la instrucción en cuestión. Con un compilador, si se localiza un error durante la ejecución, el programador debe corregir las instrucciones erróneas sobre el archivo fuente (no sobre el objeto), y volver a compilarlo en su totalidad.

Obviamente, los intérpretes resultan más pedagógicos para aprender a programar, ya que el alumno puede detectar y corregir más fácilmente sus errores.

Otra ventaja de los traductores-intérpretes es que ocupan, por lo general, menos memoria que los compiladores, pudiendo en ciertos casos estar grabados en memoria ROM (caso del BASIC en los PC's).

Existen en la actualidad traductores (Turbo Pascal) que en cierta medida disponen de las ventajas tanto de los intérpretes, como de los compiladores. Estos traductores se denominan compiladores interactivos o incrementales. Un compilador interactivo es un sistema que permite la edición, compilación, ejecución y depuración de programas escritos en un determinado lenguaje de alto nivel.

Existen lenguajes cuyos traductores se idearon como intérpretes (BASIC, APL, LISP y PROLOG), y otros como compiladores (FORTRAN, COBOL, ALGOL, SNOBOL, PL/I, C, MODULA y ADA). No obstante, para un lenguaje dado, pueden existir tanto compiladores como intérpretes (BASIC, FORTRAN, LISP y SNOBOL).

TRADUCTORES CRUZADOS.

Son aquellos traductores que efectúan la traducción de programas fuente a programas objeto en un computador distinto (computador B o computador anfitrión -host-) a aquel en el que se ejecutará el programa objeto (computador A o computador huésped -guest-). Por lo general el computador anfitrión es más potente que el huésped.

EMULADORES.

En un computador puede simularse el comportamiento de otro. Estos programas de simulación se suelen denominar emuladores.

CLASIFICACIÓN DE LOS LENGUAJES.

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios. Un criterio simple es el nivel. El nivel hace referencia a lo próxima que la forma de expresar las sentencias esté al hombre (al lenguaje natural), o a la máquina (al lenguaje de ceros y unos de los circuitos electrónicos).

A un nivel más alto que el de los lenguajes de programación se encuentran los lenguajes declarativos. En los lenguajes declarativos o de órdenes, los programas están formados por sentencias que ordenan «qué es lo que se quiere hacer», no teniendo el programador que indicar al computador el proceso detallado (el algoritmo) de «cómo hacerlo». En este grupo se incluyen ciertos lenguajes especializados, para propósitos muy específicos, tales como recuperación de la información en bases de datos (NATURAL e IMS), análisis de circuitos electrónicos y realización de cálculos estadísticos (SAS, etc...).

Los lenguajes de alto nivel se pueden clasificar, atendiendo al estilo de programación, en dos grandes grupos:

1. Lenguajes basados en la asignación de valores (lenguajes imperativos o procedurales). Se fundamentan en la utilización de variables para almacenar valores, y en la realización de operaciones con los datos almacenados. La mayoría de los lenguajes son de este tipo: FORTRAN, BASIC, COBOL, Pascal, Modula, ADA, C, etc....

2. Lenguajes basados en la definición de funciones o relaciones. No utilizan instrucciones de asignación (sus variables no almacenan valores). Los programas están formados por una serie de definición de funciones (lenguajes funcionales, como LISP) o de predicados (lenguajes de programación lógica, como PROLOG).

APLICACIONES.

Aplicaciones científicas.

Son aquellas en las que predominan el tratamiento de (o cálculos con) números y matrices. Los algoritmos son de tipo matemático (cálculo numérico, estadística, etc...). Lenguajes adecuados para este tipo de problemas son el FORTRAN, APL y Pascal.

Aplicaciones en procesamiento de datos.

En estas aplicaciones predominan problemas de creación, mantenimiento, consulta y listado de datos, organizados en registros, archivos y bases de datos. Se incluyen en este apartado aplicaciones de gestión empresarial tales como programas de nómina, contabilidad, facturación, inventario, control de producción y control de ventas. Lenguajes aptos para este tipo de problemas son el COBOL y el PL/I.

Aplicaciones de tratamiento de textos.

Principalmente relacionados con la manipulación de textos en lenguaje natural. Lenguajes adecuados para ello son el SNOBOL y el C.

Aplicaciones en inteligencia artificial.

Consisten en la realización de programas que emulan el comportamiento inteligente: algoritmos de juegos (ajedrez, etc...), programas para comprender el lenguaje natural, visión artificial, robótica y sistemas expertos (diagnóstico médico, etc...). Los lenguajes que se suelen utilizar en estos casos son el LISP y PROLOG.

Aplicaciones de programación de sistemas.

Se realizan para la programación de módulos de sistemas operativos, compiladores, ensambladores, intérpretes y, en general, aquellos de interfaz entre el hardware y los usuarios. Tradicionalmente se utilizaban para estos cometidos lenguajes ensambladores, pero en la actualidad se están mostrando muy adecuados los lenguajes Ada, C y Modula-2.

LENGUAJES DE PROGRAMACIÓN.

✍ ADA: Lenguaje desarrollado por encargo del Departamento de Defensa e USA. Intentaba conseguir un lenguaje estándar que permitiera confeccionar cualquier tipo de aplicación ya fuera de gestión, técnica o científica.

✍ BASIC: Fue considerado el lenguaje de programación más fácil de aprender, por ello debe su nombre (Código de instrucción simbólico de propósito general para principiantes). Sin lugar a dudas, se trata del más popular, debido a que el MS-DOS, incorpora en la mayoría de los casos los programas "GW-BASIC" o "Qbasic", variantes del lenguaje primario. Además, hay que destacar que puede emplearse indiferentemente para generar aplicaciones técnicas o de gestión. Uno de los lenguajes de programación vinculados al "BASIC" y que ha alcanzado gran nivel de popularidad es "QuickBASIC". La gran ventaja de éste respecto a "GW-BASIC" es que el programa desarrollado se puede "compilar" convirtiéndole en un archivo ejecutable (*.EXE), pudiendo de esta forma ser ejecutado de forma directa.

✍ C: Diseñado por los creadores del sistema operativo UNIX, siendo su función principal la de permitir la programación de sistemas, si bien en la actualidad se emplea en todo tipo de aplicaciones (gestión, técnicas, científicas, etc…). Gracias a la programación estructurada para resolver cierto tipo de tareas y la librería de rutinas incorporada, se ha convertido en uno de los lenguajes de mayor difusión en la actualidad, con variantes como "Quick C", "Turbo C", "C ++" y "Visual C ++".

✍ CLIPPER: Inicialmente se consideraba un compilador de "dBase". Hoy en día podemos hablar de un auténtico sistema para el desarrollo de programas totalmente independientes, combinando las ventajas de las Bases de datos con las posibilidades de un lenguaje de alto nivel.

✍ COBOL: Al contrario que el anterior, su campo de acción son las aplicaciones de gestión, considerándose en el momento de su creación como el lenguaje universal para aplicaciones comerciales.

✍ FORTRAN: Es el lenguaje de alto nivel más antiguo, y se considera idóneo para aplicaciones científicas y técnicas, siendo su principal característica la potencia para cálculos matemáticos, quedando muy limitado para aplicaciones de gestión, en las que se utiliza constantemente el empleo de cadenas de caracteres y manejo de archivos.

✍ LISP: También perteneciente al grupo destinado a los ordenadores de "quinta generación" posee una estructura a partir de listas o secuencias de instrucciones.

✍ MODULA:Se trata de una variación de "PASCAL", mucho más severo en la planificación de la estructura del programa, permitiendo de esta forma evitar errores a lo largo del mismo.

✍ PASCAL: La misión inicial de este lenguaje era suministrar un lenguaje idóneo en el área de la enseñanza de las técnicas de programación, si bien se ha convertido en uno de los más usuales en el área técnico-científica, destacándose en la realización de diseños gráficos.

✍ PROLOG: Pertenece al grupo de lenguajes destinados a ser utilizados en ordenadores de la denominada "quinta generación" o "inteligencia artificial". Se basa en la "resolución" (forma de lógica matemática) mediante la cual se indican los problemas y normas a seguir para buscar la solución apropiada.

ALGORITMOS.

Un computador es capaz de realizar determinadas acciones sencillas, tales como sumar, restar o transferir datos. Estas acciones son de por si útiles; sin embargo, los problemas que normalmente interesa resolver son más complejos. Para solucionar un problema real, es necesario encontrar un método de resolución del problema y, posteriormente, descomponerlo en acciones sencillas, que el computador sea capaz de realizar.

No todos los métodos de solución de un problema son susceptibles de ser utilizados por un computador. Para que un procedimiento pueda ser implantado en un computador, o en otra máquina capaz de interpretar instrucciones, debe cumplir ciertos requisitos:

✍ El procedimiento debe estar compuesto de acciones bien definidas, esto es, no ambiguas. El significado de cada acción debe ser único, en el contexto en que aparece.

✍ El procedimiento debe esta formado por una secuencia finita de operaciones. Además, debe quedar perfectamente definido el orden en que se van a realizar las instrucciones.

✍ Por último, el procedimiento debe acabar en un tiempo finito. Un procedimiento que puede no acabar nunca no es útil para resolver un problema.

Un procedimiento o método de solución, para resolver un problema, que cumpla estos requisitos se dice que es un algoritmo que resuelve ese problema. Por ello, podemos definir algoritmo como un procedimiento no ambiguo que resuelve un problema. Un procedimiento es una secuencia de operaciones bien definidas, cada una de las cuales requiere una cantidad finita de memoria y se realiza en un tiempo finito.

No hay ningún procedimiento riguroso que permita construir un algoritmo que resuelva un problema dado. El diseño de algoritmos, como toda tarea creativa, es una tarea compleja, en la que se pueden seguir pocas normas, teniendo por tanto gran importancia la imaginación y experiencia de la persona que lo realiza.

Hay diferentes formas para representar los algoritmos:

1.Pseudocódigo.

No hay reglas fijas para la representación narrativa de algoritmos. No obstante, para describir algoritmos está muy extendido el uso de las estructuras de control del lenguaje Pascal o APL. En este caso, como el objetivo no es escribir un programa para ser ejecutado por un computador, no hay reglas sintácticas estrictas, el interés se centra en la secuencia de instrucciones. Este tipo de descripción se denomina pseudocódigo. La utilización de pseudocódigo presenta las ventajas de ser más compacto que un organigrama, ser más fácil de escribir y ser más fácil de transcribir a un lenguaje de programación.

2. Organigramas.

   Los organigramas son herramientas gráficas para representar algoritmos. Un organigrama está compuesto por una serie de símbolos unidos por flechas. Los símbolos representan acciones, y las flechas el orden de realización de las acciones. Cada símbolo, por tanto, tendrá al menos una flecha que conduzca a él y una flecha que parta de él, salvo cuando el símbolo represente el inicio o el fin.

3. Diagramas de Nassi-Schneiderman.

   Estos diagramas o diagramas de Chapin tienen la ventaja de adecuarse a las técnicas de programación estructurada. No utilizan flechas para indicar el flujo de control, teniendo una serie de ventajas con respecto a los organigramas clásicos, representándose en un dibujo contenido en un rectángulo.

****