TEMA 1: INTRODUCCIÓN

1.- INTRODUCCIÓN

Aunque los problemas o tareas que se plantean diariamente pueden ser resueltos mediante el uso de la capacidad intelectual y la habilidad manual de la persona, la utilización del ordenador para la realización automática de una tarea aporta grandes ventajas, como la ***rapidez de ejecución y la fiabilidad de los resultados obtenidos***. Un buen número de problemas conllevan complicados cálculos, así como el manejo de grandes cantidades de datos. En el primer caso, el riesgo de equivocarse es grande y en el segundo, el trabajo se convierte en pesado y rutinario. Mediante el uso del ordenador se eliminan estos inconvenientes debido a las capacidades de la máquina, basadas en las siguientes características:

- ✓ Rapidez.
- ✓ Precisión.
- ✓ Memoria

Hoy en día, los ordenadores se encuentran plenamente integrados en la sociedad actual, y cada vez en mayor medida. Tal es así, que algunos autores apuestan por llamar a esta época, la "**Era de la Información y la Comunicación**".

No obstante, pese a que en el cine y la literatura de ciencia-ficción se utiliza mucho el argumento del ordenador que piensa por sí mismo, o el que se rebela contra los humanos, el ordenador por sí solo no sabría resolver ni el más sencillo problema que se nos pueda ocurrir. Es preciso, para que pueda hacerlo, describirle con detalle y en su lenguaje todos los pasos que ha de llevar a cabo para la resolución del problema. Una descripción de este tipo es lo que se llama ***programa*** y su objetivo es dirigir el funcionamiento de la máquina.

Un ***lenguaje de programación*** es un conjunto de símbolos y de reglas para combinarlos, que se usan para expresar *algoritmos* o *programas* (más adelante veremos las diferencias entre estos dos conceptos).

El objetivo de este módulo es, para un problema dado, diseñar una solución que pueda ser realizada por un ordenador.

2. ¿QUE ES UN ORDENADOR?

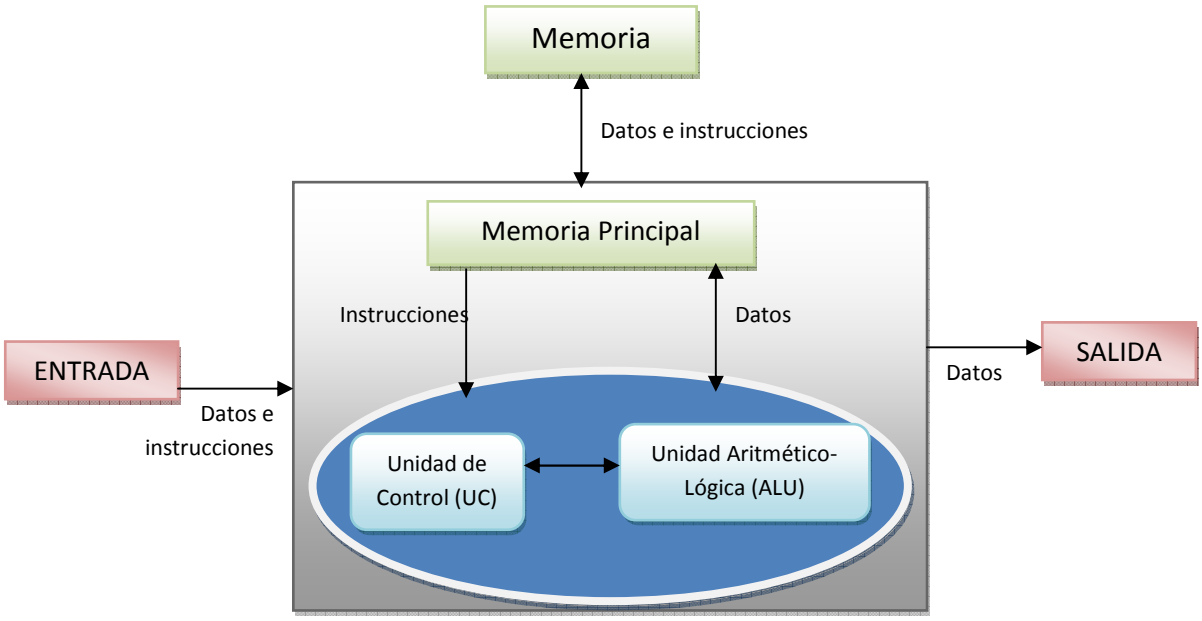
Un **ordenador** es un **dispositivo electrónico programable capaz de almacenar y procesar información**. Pero por sí solo, no es capaz de hacerlo, necesita todo un **sistema** a su alrededor para realizar estas tareas.

Un **sistema informático** es el conjunto de recursos que son necesarios para la elaboración y el uso de aplicaciones informáticas y está formado por:

- ✓ El elemento **físico** o **hardware**.
- ✓ El elemento **lógico** o **software**.
- ✓ El elemento **humano** o **personal informático**.

El elemento físico o hardware

El hardware o soporte físico es la máquina en sí, es decir, está constituido por los elementos físicos del ordenador. La estructura típica del hardware de un ordenador es la que se muestra en la siguiente figura:



Un ordenador se compone de las siguientes unidades funcionales:

- ***Unidad Central de Proceso (CPU).***
  - Unidad Aritmético-Lógica (ALU): Esta parte de la máquina se encarga de realizar todas las operaciones aritméticas (sumas, restas, etc.) y lógicas (comparaciones, etc) que sean necesarias durante el proceso. Los datos que manipula proceden de la memoria.
  - Unidad de Control (UC): Controla la ejecución de instrucciones, es decir, dirige y coordina todas las operaciones que tienen lugar en las restantes partes del ordenador. Es una especie de director de orquesta del resto de las componentes.

- ***Memoria*:** Almacena información e instrucciones
  - Principal, Central o Interna: gran velocidad, poca capacidad. Su función es la de contener en todo instante la información inicial (datos), la elaborada (resultados) y las instrucciones necesarias para llegar de los datos de entrada a los resultados.
  - Secundaria, Masiva auxiliar o Externa: poca velocidad, gran capacidad. Está formada por los dispositivos que permiten almacenar información de forma permanente.

- ***Periféricos*:** Cumplen la misión de comunicar al ordenador con el mundo exterior. A grandes rasgos podemos distinguir dos tipos de dispositivos de comunicación con el exterior.
  - Entrada: recogida de información e instrucciones. Transforman la información externa en señales eléctricas codificadas, permitiendo su tratamiento por parte del ordenador.
  - Salida: obtención de resultados. Realizan la operación contraria, transforman códigos binarios en información inteligible para el usuario o bien almacenan la información en algún soporte de memoria externa (discos, cintas, etc).

- ***Buses*:** son los encargados de comunicar los distintos módulos en los que se divide un ordenador. En otras palabras son las autopistas por donde fluye la información dentro del ordenador.

El elemento lógico o software

El software o soporte lógico de una computadora es el conjunto de *programas ejecutables* por la computadora, así como los datos manejados por los programas. Por extensión también se considera software a los documentos y materias relativos al funcionamiento, el diseño y la construcción de programas.

Dentro del software podemos hacer una clasificación:

- **Datos:** son los datos que usan los programas para trabajar y operar con ellos. Por ejemplo: los nombre de alumnos del Instituto, sus notas, cursos, asignaturas,...

- **Programas:** instrucciones para operar con los Datos. Dentro de los programas podemos a su vez hacer otra división:
  - Sistemas Operativos: programas básicos para manejar el hardware del ordenador. Dicho en otras palabras es un conjunto de instrucciones encargado de comunicar al usuario con el ordenador y de permitir el control de los periféricos existentes.
  - Lenguajes de programación**:** sirven para crear programas ejecutables de cualquier tipo.
  - Aplicaciones**:** programas que pueden servir para multitud de utilidades distintas. De hecho, podríamos considerar a los dos apartados anteriores dentro de este mismo, ya que al fin y al cabo TODO son programas para distintas aplicaciones.

El elemento humano o personal informático

Es el elemento más importante del sistema informático. En la sociedad actual, todos en algún momento del día nos vemos relacionados con algún sistema de proceso de información, por ejemplo, usar el cajero automático o mandar un correo electrónico.

En general, se llama **usuario** a este grupo de personas que utilizan los ordenadores en última instancia, usado programas de utilidades más o menos complejos creados por otras personas, con el objetivo de ayudarle en alguna actividad. Y se conoce como **personal informático** al conjunto de personas que trabajan para garantizar el correcto funcionamiento de los ordenadores, es decir, además de utilizarlos como herramienta, son el objeto de su trabajo, en este segundo grupo podemos incluir a personal de dirección, analistas, programadores y operadores.

3. REPRESENTACIÓN DE LA INFORMACION EN EL ORDENADOR

Un ordenador maneja información de todo tipo. Nuestra perspectiva humana nos permite rápidamente diferenciar lo que son números, de lo que es texto, imagen,… Sin embargo al tratarse de una máquina digital, el ordenador sólo es capaz de representar números en forma binaria. Por ello todos los ordenadores necesitan codificar la información del mundo real a el equivalente binario entendible por el ordenador.

Es importante conocer como almacena un ordenador la información, esta se puede codificar como patrones de bits que sean fáciles de almacenar y procesar por los elementos internos del ordenador. Las formas de información más significativas son: textos, sonidos, imágenes y valores numéricos.

La información en formato texto se representa mediante un código en el que cada uno de los distintos símbolos de texto se asigna a un único patrón de bits. El texto se representa como una cadena larga en la cual los sucesivos patrones representan los sucesivos símbolos del texto original. Los caracteres que se utilizan en informática suelen agruparse en cinco categorías:

- Caracteres **alfabéticos**: letras mayúsculas y minúsculas (no se incluye la ñ).
- Caracteres **numéricos**: dígitos del 0 al 9.
- Caracteres **especiales**: símbolos ortográficos y matemáticos no incluidos en los grupos anteriores. Por ejemplo: { } Ñ ñ) | @ # Ç
- Caracteres **geométricos y gráficos**: símbolos o módulos con los cuales se pueden representar cuadros, figuras geométricas, etc. Por ejemplo: ♥☺ ╚╣ ┐
- Caracteres de **control:** representan órdenes de control, como el carácter ara pasar a la siguiente línea (NL), o para ir al comienzo de una línea (RC, retorno de carro), emitir un pitido, etc.

Al introducir un texto en un ordenador, a través de un periférico, los caracteres se codifican según un código de entrada/salida de modo que a cada carácter se le asocia una determinada combinación de n bits. Los códigos más utilizados son EBCDIC, ASCII y UNICODE.

- **EBCDIC**, utiliza 8 bits, fue el primer código utilizado y aceptado en principio por IBM.
- **ASCII**, tiene una versión más antigua de 7 bits y una versión ampliada de 8 bits que permite más símbolos entre ellos símbolos y caracteres del idioma español.
- **UNICODE**, aunque ASCII ha sido y es dominante en la representación de caracteres, hoy en día se requiere de la necesidad de representación de la información en otras muchas lenguas como el portugués, el chino, el español, el árabe, etc. Por ese motivo surgió este código ya que utiliza 16 bits lo cual permite tener muchos más símbolos diferentes.

Aquellos alumnos que tengan interés en ver la forma de codificar la información de valores numéricos, sonidos e imágenes lo puede consultar en el libro "*Fundamentos de programación*" de *Luis Joyanes.*

En el Anexo I se pueden consultar las tablas de representación de caracteres.

4. LENGUAJES DE PROGRAMACIÓN

4.1. Historia de los lenguajes de programación

Inicio de la programación

Charles Babbage definió a mediados del siglo XIX lo que él llamó la máquina analítica. Se considera a esta máquina el diseño del primer ordenador. La realidad es que no se pudo construir hasta el siglo siguiente. El caso es que su colaboradora Ada Lovelace escribió en tarjetas perforadas una serie de instrucciones que la máquina iba a ser capaz de ejecutar. Se dice que eso significó el inicio de la ciencia de la programación de ordenadores.

En la segunda guerra mundial debido a las necesidades militares, la ciencia de la computación prospera y con ella aparece el famoso ENIAC (Electronic Numerical Integrator And Calculator), que se programaba cambiando su circuitería. Esa es la primera forma de programar (que aún se usa en numerosas máquinas) que sólo vale para máquinas de único propósito. Si se cambia el propósito, hay que modificar la máquina.

Primera Generación (1GL): Código máquina.

No mucho más tarde apareció la idea de que las máquinas fueran capaces de realizar más de una aplicación. Para lo cual se ideó el hecho de que hubiera una memoria donde se almacenaban esas instrucciones. Esa memoria se podía rellenar con datos procedentes del exterior. Inicialmente se utilizaron tarjetas perforadas para introducir las instrucciones.

Durante mucho tiempo esa fue la forma de programar, que teniendo en cuenta que las máquinas ya entendían sólo código binario, consistía en introducir la programación de la máquina mediante unos y ceros. El llamado código máquina. Todavía los ordenadores es el único código que entienden, por lo que cualquier forma de programar debe de ser convertida a código máquina. Utiliza el alfabeto binario, que consta de los únicos símbolos 0 y 1, denominados *bits* (abreviatura inglesa de dígitos binarios)

Sólo se ha utilizado por los programadores en los inicios de la informática. Su incomodidad de trabajo hace que sea impensable para ser utilizado hoy en día, siendo sustituido por otros lenguajes más fáciles de aprender y utilizar, que además reducen la posibilidad de cometer errores. Sin embargo cualquier programa de ordenador debe, finalmente, ser convertido a este código para que un ordenador pueda ejecutar las instrucciones de dicho programa.

Un detalle a tener en cuenta es que el código máquina es distinto para cada tipo de procesador. Lo que hace que los programas en código máquina no sean portables entre distintas máquinas.

El lenguaje máquina ofrece ciertas ventajas:

- Es directamente entendible por el ordenador.
- Es muy eficiente, ya que depende de la máquina, con lo cual se ejecuta muy rápidamente y permite aprovechar los recursos de ésta en su totalidad. No obstante, la eficiencia de un programa codificado en cualquier lenguaje de programación depende en gran medida de la habilidad del programador.

Sin embargo, presenta también algunos inconvenientes a considerar:

- Resulta complicado trabajar con el código binario.
- El programador debe conocer la arquitectura física del ordenador con cierto nivel de detalle. Fundamentalmente, las características del microprocesador (conjunto de registros y juego de instrucciones) y la programación de los diferentes dispositivos de entrada/salida.
- No posee sentencias declarativas (proporcionan información sobre determinadas circunstancias del programa, ej. definición de variables), sino que todas las sentencias son instrucciones.
- Es un lenguaje totalmente dependiente del microprocesador del ordenador. Si dos ordenadores tienen dos microprocesadores distintos, normalmente tendrán distinto lenguaje máquina. Por lo tanto, los programas escritos en este lenguaje son poco transportables de un ordenador a otro.
- No puede incluir comentarios que faciliten la legibilidad de los programas.
- Poseen un conjunto de instrucciones bastante reducido (aritméticas y lógicas, salto condicional e incondicional, carga y almacenamiento de operandos en memoria), en comparación con los lenguajes de alto nivel.

Segunda generación (2GL): Lenguaje ensamblador.

En los años 40 se intentó concebir un lenguaje más simbólico que permitiera no tener que programar utilizando código máquina. Poco más tarde se ideó el lenguaje ensamblador, que es la traducción del código máquina a una forma más textual. Cada tipo de instrucción se asocia a una palabra mnemotécnica (como SUM para sumar por ejemplo), de forma que cada palabra tiene traducción directa en el código máquina.

Tras escribir el programa en código ensamblador, un programa (llamado también ensamblador) se encargará de traducir el código ensamblador a código máquina. Esta traducción es rápida puesto que cada línea en ensamblador tiene equivalente directo en código máquina (en los lenguajes modernos no ocurre esto).

La idea es la siguiente: si en el código máquina, el número binario 0000 significa sumar, y el número 0001 significa restar. Una instrucción máquina que sumara el número 8 (00001000 en binario) al número 16 (00010000 en binario) sería:

0000 00001000 00010000

Realmente no habría espacios en blanco, el ordenador entendería que los primeros cuatro BITS representan la instrucción y los 8 siguientes el primer número y los ocho siguientes el segundo número (suponiendo que los números ocupan 8 bits). Lógicamente trabajar de esta forma es muy complicado. Por eso se podría utilizar la siguiente traducción en ensamblador:

SUM 8 16

Que ya se entiende mucho mejor.

Este lenguaje presenta la mayoría de los inconvenientes del lenguaje máquina:

- Cada modelo de computadora tiene un lenguaje ensamblador propio.
- El programador ha de conocer perfectamente el hardware del equipo, ya que maneja directamente las posiciones de memoria, registros del procesador y demás segmentos físicos.
- Todas las instrucciones son elementales, es decir, en el programa se deben escribir con el máximo detalle todas las operaciones que se han de efectuar en la máquina para la realización de cualquier proceso.

Por otro lado, al igual que el lenguaje máquina goza de la ventaja de mínima ocupación de memoria y mínimo tiempo de ejecución.

Tercera generación (3GL): Lenguajes de alto nivel.

Aunque el ensamblador significó una notable mejora sobre el código máquina, seguía siendo excesivamente críptico. De hecho para hacer un programa sencillo requiere miles y miles de líneas de código.

Para evitar los problemas del ensamblador apareció la tercera generación de lenguajes de programación, la de los lenguajes de alto nivel. En este caso el código vale para cualquier máquina pero deberá ser traducido mediante software especial que adaptará el código de alto nivel al código máquina correspondiente. Esta traducción es necesaria ya que el código en un lenguaje de alto nivel no se parece en absoluto al código máquina.

Tras varios intentos de representar lenguajes, en 1957 aparece el que se considera el primer lenguaje de alto nivel, el FORTRAN (FORmula TRANslation), lenguaje orientado a resolver fórmulas matemáticas. Por ejemplo la forma en FORTRAN de escribir el texto Hola mundo por pantalla es:

```
PROGRAM HOLA
   PRINT *, '¡Hola, mundo!'
```

Poco a poco fueron evolucionando los lenguajes formando lenguajes cada vez mejores. Así en 1958 se crea LISP como lenguaje declarativo para expresiones matemáticas. Programa que escribe Hola mundo en lenguaje LISP:

```
(format t "¡Hola, mundo!")
```

En 1960 la conferencia CODASYL se creo el COBOL como lenguaje de gestión en 1960. En 1963 se creo PL/I el primer lenguaje que admitía la multitarea y la programación modular. En COBOL el programa Hola mundo sería éste (como se ve es un lenguaje más declarativo):

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
MAIN SECTION.
DISPLAY "Hola mundo"
STOP RUN.
```

BASIC se creó en el año 1964 como lenguaje de programación sencillo de aprender y ha sido, y es, uno de los lenguajes más populares. En 1968 se crea LOGO para enseñar a programar a los niños. Pascal se creó con la misma idea académica pero siendo ejemplo de lenguaje estructurado para programadores avanzados. El creador del Pascal (Niklaus Wirdth) creo Modula en 1977 siendo un lenguaje estructurado para la programación de sistemas (intentando sustituir al lenguaje C).

Programa que escribe por pantalla Hola mundo en lenguaje Pascal:

```
PROGRAM HolaMundo;
BEGIN
    Writeln('¡Hola, mundo!');
END.
```

En 1967, Martin Richards desarrolla BCPL un lenguaje sin tipos (se basaba en palabras de máquina), y que requería en gran medida el uso de apuntadores y aritmética de direcciones contrarios al espíritu de los lenguajes estructurados hacia los que evolucionaba la codificación. En 1970 Ken Thompson implementa UNIX usando ensamblador y un lenguaje B de su propia cosecha.

C es desarrollado por Dennis Ritchie en 1972 en los Laboratorios Bell instalándose en un DEC PDP-11. Se diseñó para ser el lenguaje de los Sistemas Operativos UNIX, superando las limitaciones de su predecesor B. Su definición se publicó en 1978 en el apéndice "C Reference Manual" del libro The C Programming Language (Brian Kernighan & Dermis Ritchie),

Durante algunos años la versión de C se suministraba con UNIX versión 5. Si añadimos a esto la popularización de los microordenadores, el resultado fue la aparición de una multitud de versiones C, increíblemente compatibles, aunque con lógicas discrepancias de interpretación. Para resolver este problema, en 1983 el American National Standar Institute (ANSI) creó un comité para obtener una definición no ambigua del lenguaje C que fuera independiente de la arquitectura de cualquier máquina. De esta forma quedó definido en 1989 el estándar ANSI para el lenguaje C, conocido comúnmente como ANSI C.

Programa que escribe por pantalla Hola mundo en lenguaje C:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Hola mundo");
   return 0;
}
```

Cuarta generación (4GL): Lenguajes de alto nivel.

En los años 70 se empezó a utilizar éste término para hablar de lenguajes en los que apenas hay código y en su lugar aparecen indicaciones sobre qué es lo que el programa debe de obtener. Se consideraba que el lenguaje SQL (muy utilizado en las bases de datos) y sus derivados eran de cuarta generación. Los

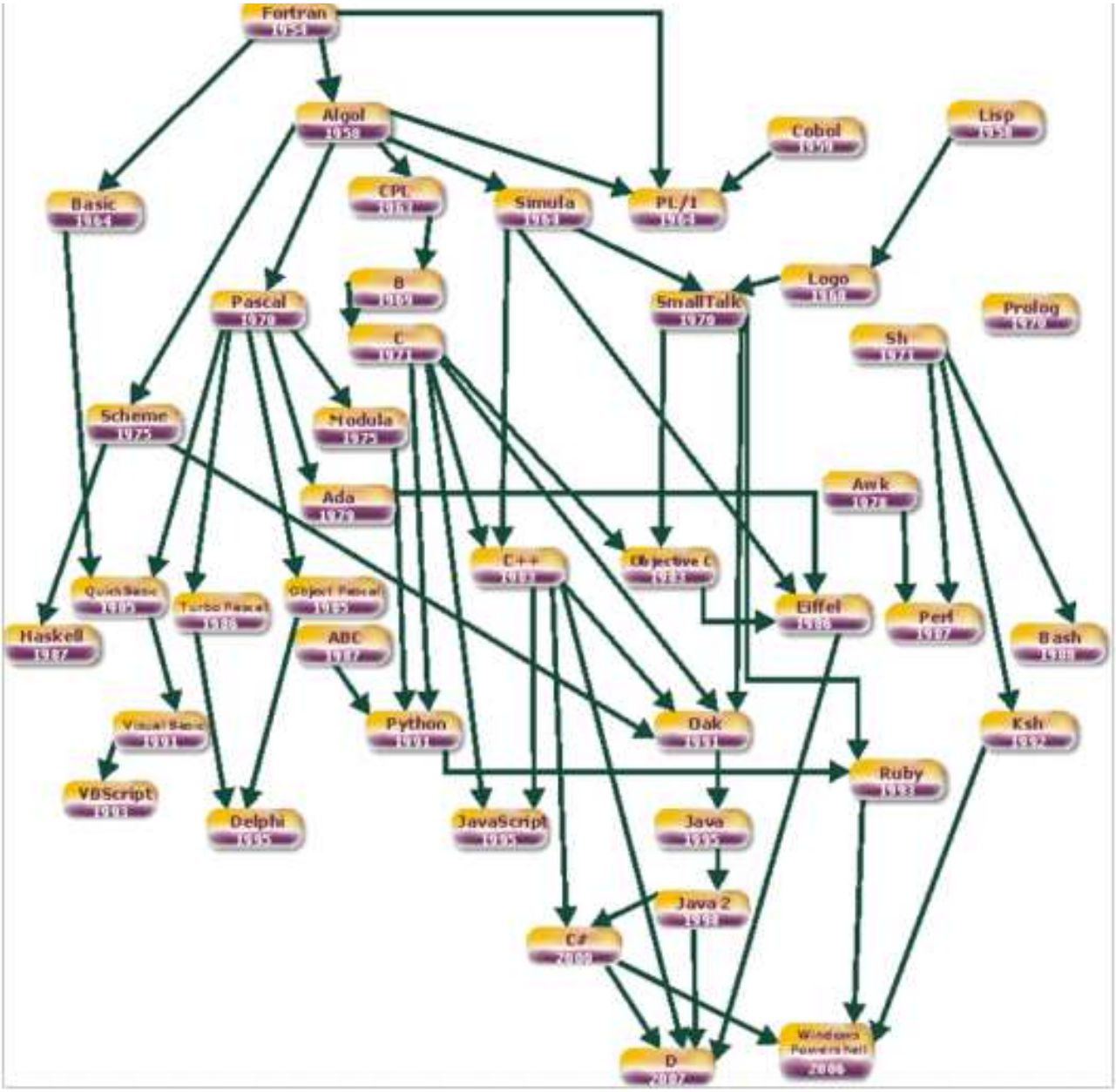
lenguajes de consulta de datos, creación de formularios, informes,... son lenguajes de cuarto nivel. Aparecieron con los sistemas de base de datos

Actualmente se consideran lenguajes de éste tipo a aquellos lenguajes que se programan sin escribir casi código (lenguajes visuales), mientras que también se propone que éste nombre se reserve a los lenguajes orientados a objetos.

Lenguajes orientados a objetos

En los 80 llegan los lenguajes preparados para la programación orientada a objetos todos procedentes de Simula (1964) considerado el primer lenguaje con facilidades de uso de objetos. De estos destacó inmediatamente C++, otros fueron Smalltalk, Eifell, Actor, etc.

A partir de C++ aparecieron numerosos lenguajes que convirtieron los lenguajes clásicos en lenguajes orientados a objetos (y además con mejoras en el entorno de programación, son los llamados lenguajes visuales): Visual Basic, Delphi (versión orientada a objetos de Pascal), Visual C++,...



En 1995 aparece Java como lenguaje totalmente orientado a objetos y en el año 2000 aparece C# un lenguaje que toma la forma de trabajar de C++ y del propio Java. El programa Hola mundo en C# sería:

```
using System;

class MainClass
{
    public static void Main()
    {
        Console.WriteLine("¡Hola, mundo!");
    }
}
```

Lenguajes para la web

La popularidad de Internet ha producido lenguajes híbridos que se mezclan con el código HTML con el que se crean las páginas web. HTML no es un lenguaje en sí sino un formato de texto pensado para crear páginas web. Estos lenguajes se usan para poder realizar páginas web más potentes.

Son lenguajes interpretados como JavaScript o VB Script, o lenguajes especiales para uso en servidores como ASP, JSP o PHP. Todos ellos permiten crear páginas web usando código mezcla de página web y lenguajes de programación sencillos.

Ejemplo, página web escrita en lenguaje HTML y JavaScript que escribe en pantalla “Hola mundo” (de color rojo aparece el código en JavaScript):

```
<html>
<head>
   <title>Hola Mundo</title>
</head>
<body>
   <script type="text/javascript">
      document.write("¡Hola mundo!");
   </script>
</body>
</html>
```

4.2. Clasificación de los lenguajes de programación

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 cómo realizar construcciones del lenguaje y una *semántica*, que determina el significado de cada construcción correcta.

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). Atendiendo a este criterio se pueden establecer los tres grupos siguientes:

- **Lenguajes de bajo nivel** (máquina).
- **Lenguajes intermedios** (ensambladores).
- **Lenguajes de alto nivel** (evolucionados).

En cuanto a los lenguajes de **alto nivel,** existen varios cientos de ellos, siendo bastante difícil establecer una clasificación general de los mismos, ya que en cualquiera que se realice habrá lenguajes que pertenezcan a más de uno de los grupos establecidos. Por agruparlos de alguna forma podríamos hacer la siguiente clasificación:

- **Lenguajes imperativos o procedurales:** representa el método tradicional de programación. Un lenguaje imperativo es un conjunto de instrucciones que se ejecutan una por una, de principio a fin, de modo secuencial, excepto cuando intervienen instrucciones de salto de secuencia o control. son lenguajes basados en la asignación de valores. 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 eran de este tipo: Pascal, ADA, C, Modula-2, Basic, Cobol, etc.
- **Lenguajes declarativos:** están basados en la definición de funciones o relaciones. Los programas están formados por sentencias que ordenan "que es lo que se quiere hacer", no teniendo el programador que indicar al ordenador el proceso detallado (algoritmo) de "cómo hacerlo". No utilizan instrucciones de asignación (sus variables no almacenan valores). Los programas están formados por una serie de definiciones de funciones (**lenguajes funcionales**, como pueden ser Lisp o Haskell) o de predicados (**lenguajes lógicos**, como Prolog).
- **Lenguajes orientados a objetos:** el diseño de los programas se basa más en los datos y su estructura. La unidad de proceso es el objeto y en él se incluyen los datos (variables) y las operaciones que actúan sobre ellos (por ejemplo Smalltalk, C++, Java, Eifell, etc).

Los lenguajes imperativos y orientados a objetos, son mucho más utilizados que los lenguajes declarativos su uso está mucho más extendido debido a varias razones, entre ellas se podrían destacar las siguientes: se utilizan para desarrollar aplicaciones reales, son los que primero se enseñan en las universidades, institutos, academias, etc., y además son muy eficientes. Sin embargo, la programación imperativa también tiene sus problemas o inconvenientes como son, el aprendizaje de estructuras de datos fijas, el manejo de punteros, la gran variedad de sentencias de control que existen, etc.

La evolución de los lenguajes ha producido un alejamiento de la máquina y un acercamiento a las personas, lo cual ha tenido como consecuencia que los programas no pueden ser directamente interpretados por el ordenador, siendo necesario realizar previamente su traducción a lenguaje máquina. Hay dos tipos de traductores de lenguajes de programación: los compiladores y los intérpretes.

4.3. Traductores de lenguajes

Los traductores son programas que, como su nombre indica, traducen un programa escrito en un lenguaje de alto nivel a su correspondiente en lenguaje máquina. Los traductores pueden clasificarse en: Ensambladores, Compiladores e Intérpretes. Un mismo lenguaje puede tener Compilador e Intérprete.

4.3.1. Compiladores

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 vez traducido el programa, su ejecución es independiente del compilador, así, por ejemplo, cualquier interacción con el usuario sólo estará controlada por el sistema operativo.

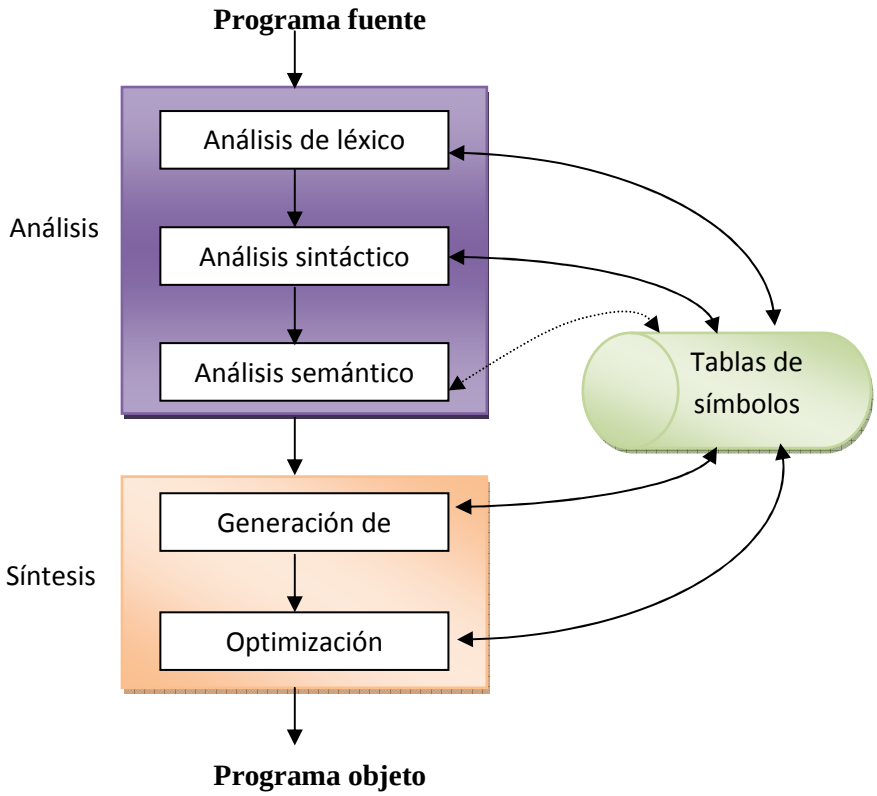
Un compilador, además del proceso de traducción, realiza una serie de funciones que en su mayoría están enfocadas a la detección de errores en la escritura del programa fuente. Por lo general está constituido por los siguientes módulos:

1. Análisis lexicográfico o "escáner". Consiste en descomponer el programa fuente en sus elementos constituyentes o **símbolos** ("tokens"). Los símbolos de un lenguaje son caracteres o secuencias de caracteres que tienen un significado concreto en el lenguaje, por ejemplo palabras reservadas, símbolos de operadores, identificadores de variables, etc.
   El analizador lexicográfico aísla los símbolos, identifica su tipo y almacena en las tablas de símbolos la información del símbolo que será necesaria durante el proceso de traducción. Así por ejemplo, convierte los números o constantes, en código de E/S (ASCII, por ejemplo), a su representación interna (entero, como flotante, etc.), ya que esta información será necesaria a la hora de generar código.

   Como resultado del análisis lexicográfico, se obtiene una representación del programa formada por la descripción de las tablas, y una secuencia de símbolos junto con la referencia a la ubicación del símbolo en la tabla. Esta representación contiene la misma información que el programa fuente, pero en una forma más compacta, no estando el código ya como una secuencia de caracteres, sino de símbolos. La información almacenada en las tablas de símbolos se completa, y utiliza en las fases posteriores de traducción.

   El analizador de léxico realiza, además, otras funciones secundarias:

   - Identifica, y salta, comentarios.
   - Identifica, y pasa, espacios en blanco y tabulaciones.
   - Informa de posibles errores de léxico.



2. Análisis sintáctico o "Parser". Busca los posibles errores sintácticos. 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 orden correcto. Se han definido varios sistemas para definir la sintaxis de los lenguajes de programación, destacan la notación BNF y los diagramas sintácticos.

3. Análisis semántico. Localiza los posibles errores de significado que aparezcan en el programa fuente como, por ejemplo, intentar sumar números con letras o combinar datos (constantes o variables) de distinto tipo dentro de la misma expresión.

En cualesquiera de los tres análisis, todo error detectado se comunica al programador por medio de un Listado de Compilación, en el que se indica el tipo de error y su localización en el programa fuente. En ocasiones, pueden existir determinados errores los cuales no perjudican al resto del proceso de compilación, e incluso permiten el funcionamiento del programa final. Los mensajes correspondientes a este tipo de errores se denominan **Advertencias** o **Warnings**.

4. 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 generado puede ser (dependiendo del compilador) directamente ejecutable, o necesitar de otros pasos previos como el ensamblado, encadenado y carga. En algunas ocasiones se utiliza un código intermedio para 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.). También se realiza la asignación de memoria a los datos definidos en el programa.

   En la fase de optimización se mejora el código intermedio, así por ejemplo si existe un bucle que da 10000 vueltas, y en su interior se le asigna a una variable un valor constante (B = 7.5), no alterándose dicho valor (B) en el bucle, el optimizador sacaría esa instrucción del bucle.

Durante muchos años, los lenguajes potentes han sido compilados. El uso masivo de Internet ha propiciado que esta técnica a veces no sea adecuada y haya lenguajes modernos interpretados o semi-interpretados, mitad se compila hacia un código intermedio y luego se interpreta línea a línea (esta técnica la siguen Java y los lenguajes de la plataforma .NET de Microsoft).

4.3.2. Intérpretes

Los intérpretes, a diferencia de los compiladores, traducen el programa fuente instrucción a instrucción. Cuando leen una instrucción fuente ejecutan las instrucciones en lenguaje máquina correspondientes a la primera. Por lo tanto, los traductores de este tipo interpretan las instrucciones simbólicas del programa fuente. No generan ningún fichero con el programa objeto en memoria auxiliar para su posterior uso.

Los intérpretes poseen las siguientes características:

- Las sentencias que hay dentro de un bucle se interpretan tantas veces como se ejecute el bucle.
- La optimización, si existe, se lleva a cabo dentro de cada sentencia, no del programa completo, a diferencia de lo que sucede en el compilador.

- Cada vez que se desea ejecutar un programa, éste se debe volver a interpretar. Ello redunda en una considerable pérdida de tiempo.

Al igual que ocurría en el proceso de compilación, el intérprete debe asignar memoria a las variables y constantes que aparecen en el programa fuente.

A la hora de depurar un programa (corregir sus errores una vez escrito), el intérprete es mucho más cómodo, pues la traducción a lenguaje máquina se lleva a cabo DURANTE la ejecución del programa fuente. En caso de error, se modifica la instrucción que lo ha producido y se vuelve a ejecutar el programa. Sin embargo, si se trata de un programa compilado, en el caso de que se produzca un error hay que volver a compilar el programa fuente corregido.

No obstante, un programa compilado se ejecuta mucho más rápidamente que un programa interpretado. Esto se debe a que en el intérprete, la ejecución de las instrucciones de un programa lleva consigo el proceso de traducción de las mismas. Esta traducción ya se ha realizado en su totalidad cuando se va a ejecutar el programa objeto que resulta de una compilación.

Hoy en día la mayoría de los lenguajes integrados en páginas web son interpretados, la razón es que como la descarga de Internet es lenta, es mejor que las instrucciones se vayan traduciendo según van llegando en lugar de cargar todas en el ordenador. Por eso lenguajes como JavaScript (o incluso, en parte, Java) son interpretados.

5. CICLO DE VIDA DEL SOFTWARE

El proceso que se sigue desde el planteamiento de un problema, hasta que se tiene una solución instalada en el ordenador y lista para su ejecución, se compone de varias fases:

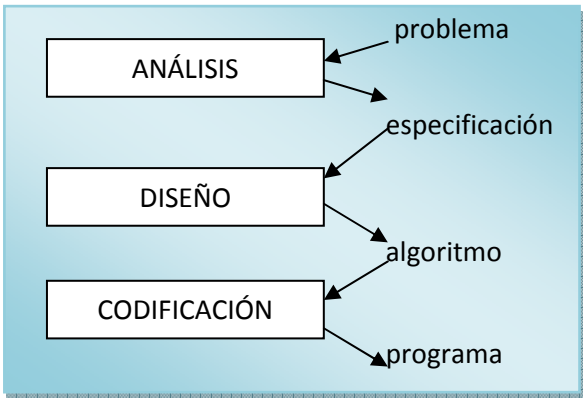
1. Análisis detallado del problema planteado.
2. Diseño del algoritmo o descomposición de la solución en tareas elementales.
3. Codificación o programación del algoritmo.
4. Obtención del programa ejecutable.
5. Prueba, verificación y depuración del programa.
6. Documentación del programa y elaboración de manuales.

5.1. Análisis del problema

El punto de partida es determinar el problema que se desea que resuelva el ordenador. Al analizar el problema que se desea solucionar deben considerarse los siguientes factores:

- ✓ El equipo que se va a utilizar (hardware y software).
- ✓ Los datos iniciales o de entrada.
- ✓ El tratamiento que debe realizarse con los datos.
- ✓ Los resultados o datos de salida.
- ✓ La posibilidad de descomponer el problema en módulos.

El resultado de esta fase es lo que se denomina **especificación** del problema, formada por el conjunto de documentos elaborados a partir de los aspectos citados.



Esquema de las tres primeras fases

5.2. Diseño del algoritmo

El **algoritmo** resultante describe, sin ambigüedades, la forma de realizar una tarea compleja mediante operaciones básicas, sencillas y elementales, desde el punto de vista del ordenador.

Esta es la fase fundamental de la resolución del problema, es la fase de programación propiamente dicha, en la cual se utilizan las técnicas de la metodología de la programación (programación estructurada, diseño descendente,...)

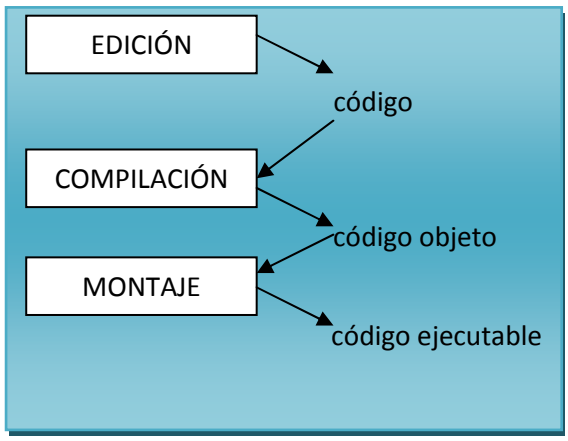
5.3. Codificación del algoritmo

Es la fase de traducción del algoritmo al lenguaje de programación elegido de acuerdo con sus reglas sintácticas. Se trata de un proceso puramente mecánico.

5.4. Obtención del programa

Obtener el programa consiste en crear el software que resolverá el problema de partida. Precisa de tres fases: edición, compilación y montaje.

- Edición.
  La fase de edición es en la que tiene lugar la creación del **código fuente** o escritura del programa, en el lenguaje de programación elegido y dentro del sistema operativo o entorno de programación adecuado. El resultado es un fichero de texto.



- Compilación.
  La fase de compilación sirve para obtener el programa en lenguaje máquina (instrucciones que la CPU sea capaz de procesar). El resultado que se obtiene son los ficheros de **código objeto** o **código máquina** que formarán parte del programa final. El proceso de compilación es realizado por los programas compiladores o intérpretes, según el lenguaje de programación utilizado.

- Montaje o linkado.
  En los programas compilados es necesario añadir al programa objeto algunas rutinas del sistema o, en algunos casos, subprogramas externos que se hayan compilado separadamente. De ello se encarga el montaje o linkado (enlazado) que es un proceso de enlace o unión entre distintas partes de código máquina para obtener un fichero o programa ejecutable.

5.5. Depuración del programa

Una vez elaborado el programa y probado su funcionamiento, llega la fase de depuración del programa, en la cual debe estudiarse su comportamiento en todas las situaciones límites que puedan plantearse, procediendo a retocar tanto el algoritmo como el código, hasta conseguir un resultado óptimo.

5.6. Documentación del programa

Un programa debe tener dos niveles de documentación: interna y externa.

La **documentación interna** forma parte del propio programa y tiene el mismo soporte y en ella se incluyen los comentarios en el propio fichero fuente (necesarios para las fases de codificación y depuración) y la autodocumentación, que se consigue cuando se emplean identificadores y expresiones que clarifiquen el modo de actuar del programa.

La documentación externa, que no forma parte del programa y su soporte puede ser el papel, debe incluir la descripción del problema, de los datos de entrada y salida, del algoritmo, del programa completo y los manuales de usuario y de mantenimiento.

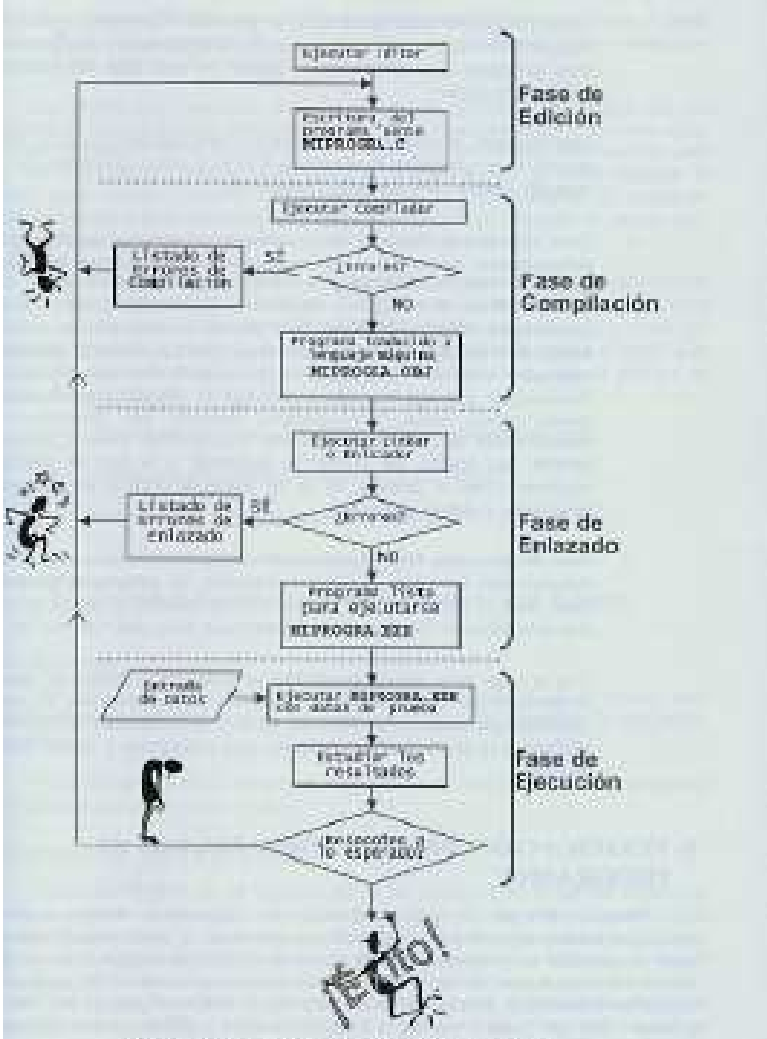


Figura 1.9. Fases en la creación de un programa

6. UML

UML es la abreviatura de *Universal Modelling Language* (Lenguaje de Modelado Universal). No es una metodología, sino una forma de escribir esquemas con pretensiones de universalidad (que ciertamente ha conseguido).

La idea es que todos los analistas y diseñadores utilizaran los mismos esquemas para representar aspectos de la fase de diseño. UML no es una metodología sino una forma de diseñar (de ahí lo de lenguaje de modelado) el modelo de una aplicación.

Su vocación de estándar está respaldada por ser la propuesta de OMG (*Object Management Group*) la entidad encargada de desarrollar estándares para la programación orientada a objetos. Lo cierto es que sí se ha convertido en un estándar debido a que une las ideas de las principales metodologías orientadas a objetos: OMT de Rumbaugh, OOD de Grady Booch y Objectory de Jacobson (conocidos como Los Tres Amigos).

Los objetivos de UML son:

(1) Poder incluir en sus esquemas todos los aspectos necesarios en la fase de diseño.

(2) Facilidad de utilización.

(3) Que no se preste a ninguna ambigüedad, es decir que la interpretación de sus esquemas sea una y sólo una.

(4) Que sea soportado por multitud de herramientas CASE. Objetivo perfectamente conseguido, por cierto.

(5) Utilizable independientemente de la metodología utilizada. Evidentemente las metodologías deben de ser orientadas a objetos.

Los tres amigos llegaron a la conclusión de que las metodologías clásicas no estaban preparadas para asimilar las nuevas técnicas de programación. Por ello definieron el llamado **Proceso Unificado de Rational.** Rational es una empresa dedicada al mundo del Análisis. De hecho es la empresa en la que recabaron Rumbaugh, Booch y Jacobson. Dicha empresa fabrica algunas de las herramientas CASE más populares (como Rational Rose por ejemplo).

En definitiva el proceso unificado es una metodología para producir sistemas de información. Los modelos que utiliza para sus diagramas de trabajo, son los definidos por UML.

Es una metodología iterativa y por incrementos, de forma que en cada iteración los pasos se repiten hasta estar seguros de disponer de un modelo UML capaz de representar el sistema correctamente.

Lo que realmente define UML es la forma de realizar diagramas que representen los diferentes aspectos a identificar en la fase de diseño. Los diagramas a realizar con esta notación son:

❒ **Diagramas que modelan los datos**.

- **Diagrama de clases**. Representa las clases del sistema y sus relaciones.
- **Diagrama de objetos**. Representa los objetos del sistema.
- **Diagrama de componentes**. Representan la parte física en la que se guardan los datos.
- **Diagrama de despliegue**. Modela el hardware utilizado en el sistema
- **Diagrama de paquetes**. Representa la forma de agrupar en paquetes las clases y objetos del sistema.

❒ **Diagramas que modelan comportamiento** (para el modelo funcional del sistema).

- **Diagrama de casos de uso**. Muestra la relación entre los usuarios (actores) y el sistema en función de las posibles situaciones (casos) de uso que los usuarios hacen.

- **Diagrama de actividades**. Representa los flujos de trabajo del programa.
- **Diagrama de estados**. Representa los diferentes estados por los que pasa el sistema.
- **Diagrama de colaboración**. Muestra la interacción que se produce en el sistema entre las distintas clases.
- **Diagramas de secuencia**. Muestran la actividad temporal que se produce en el sistema. Representa como se relacionan las clases, pero atendiendo al instante en el que lo hacen

Cabe decir que esto no significa que al modelar un sistema necesitemos todos los tipos de esquema.

Sin duda el diagrama más importante y utilizado es el de clases (también el de objetos). De hecho en temas posteriores veremos como crear dichos diagramas.

ACTIVIDADES

1.- ¿Qué entiendes por programa?

2.- Indica si el siguiente software es un sistema operativo, un lenguaje de programación o bien una aplicación de cualquier otro propósito:

Microsoft Word, FIFA 2008, Java, Microsoft Windows 2003 Server, VMWare, Age of Empires, Calc, Open Office, C++, Suse Linux, MySQL, Fedora, PHP, HTML.

3.- Indica el valor ascii de los siguientes caracteres: B, b, g, #, &

4.- Indica a qué caracteres corresponden los siguientes valores en ascii y en ebcdic: 151, 76, 94

5.- A) ¿Para qué sirven los interpretes y los compiladores?
B) ¿Son lo mismo?

C) Si no son lo mismo, ¿en qué se diferencian?

6.- ¿Qué son los warnings?

ANEXO I

TABLA ASCII (7 bits)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

EXTENSIÓN DE LA TABLA ASCII (8 bits)

128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179	|	195	├	211	╙	227	π	243	≤
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	169	_	185	╣	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	171	½	187	╗	203	╦	219	▉	235	δ	251	√
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	_
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	174	«	190	╛	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	
143	Å	160	á	176	▒	192	└	208	╨	224	α	240	≡		

Tabla EBCDIC

	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	**NUL** 0	**DLE** 16	**DS** 32	 48	**SP** 64	**&** 80	- 96	 112	 128	 144	 160	 176	**{** 192	**}** 208	\ 224	**0** 240
0001 1	**SOH** 1	**DCI** 17	**SOS** 33	 49	 65	/ 81	 97	 113	**a** 129	**j** 145	 161	 177	**A** 193	**J** 209	 225	**1** 241
0010 2	**STX** 2	**DC2** 18	**FS** 34	**SYN** 50	 66	 82	 98	 114	**b** 130	**k** 146	**s** 162	 178	**B** 194	**K** 210	**S** 226	**2** 242
0011 3	**ETX** 3	**TM** 19	 35	 51	 67	 83	 99	 115	**c** 131	**l** 147	**t** 163	 179	**C** 195	**L** 211	**T** 227	**3** 243
0100 4	**PF** 4	**RES** 20	**BYP** 36	**PN** 52	 68	 84	 100	 116	**d** 132	**m** 148	**u** 164	 180	**D** 196	**M** 212	**U** 228	**4** 244
0101 5	**HT** 5	**NL** 21	**LF** 37	**RS** 53	 69	 85	 101	 117	**e** 133	**n** 149	**v** 165	 181	**E** 197	**N** 213	**V** 229	**5** 245
0110 6	**LC** 6	**BS** 22	**ETB** 38	**UC** 54	 70	 86	 102	 118	**f** 134	**o** 150	**w** 166	 182	**F** 198	**O** 214	**W** 230	**6** 246
0111 7	**DEL** 7	**IL** 23	**ESC** 39	**EOT** 55	 71	 87	 103	 119	**g** 135	**p** 151	**x** 167	 183	**G** 199	**P** 215	**X** 231	**7** 247
1000 8	 8	**CAN** 24	 40	 56	 72	 88	 104	 120	**h** 136	**q** 152	**y** 168	 184	**H** 200	**Q** 216	**Y** 232	**8** 248
1001 9	**RLF** 9	**EM** 25	 41	 57	 73	 89	 105	\ 121	**i** 137	**r** 153	**z** 169	 185	**I** 201	**R** 217	**Z** 233	**9** 249
1010 A	**SMM** 10	**CC** 26	**SM** 42	 58	**cent** 74	**!** 90	| 106	**:** 122	 138	 154	 170	 186	 202	 218	 234	 250
1011 B	**VT** 11	**CU1** 27	**CU2** 43	**CU3** 59	**.** 75	**$** 91	**,** 107	**#** 123	 139	 155	 171	 187	 203	 219	 235	 251
1100 C	**FF** 12	**IFS** 28	 44	**DC4** 60	< 76	* 92	**%** 108	**@** 124	 140	 156	 172	 188	 204	 220	 236	 252
1101 D	**CR** 13	**IGS** 29	**ENQ** 45	**NAK** 61	**(** 77	**)** 93	_ 109	**'** 125	 141	 157	 173	 189	 205	 221	 237	 253
1110 E	**SO** 14	**IRS** 30	**ACK** 46	 62	+ 78	**;** 94	> 110	**=** 126	 142	 158	 174	 190	 206	 222	 238	 254
1111 F	**SI** 15	**IUS** 31	**BEL** 47	**SUB** 63	| 79	¬ 95	**?** 111	**"** 127	 143	 159	 175	 191	 207	 223	 239	 255

TABLA UNICODE

En el sitio web de *Unicode* (www.unicode.org) puedes ver secciones de la Tabla de Código Unicode. ¡La tabla completa es demasiado larga para cargarla completa en una página!