UT1. EJECUCIÓN DE UN PROGRAMA. LENGUAJES DE PROGRAMACIÓN.

Índice de contenido

1 EJECUCIÓN DE UN PROGRAMA INFORMÁTICO

1.1 INSTRUCCIONES Y DATOS

Las instrucciones son el conjunto de órdenes elementales que un ordenador puede ejecutar, deben estar almacenadas en memoria principal para que puedan ser leídas y ejecutadas.

Además de las instrucciones, un ordenador necesita almacenar los datos sobre los que las instrucciones trabajan.

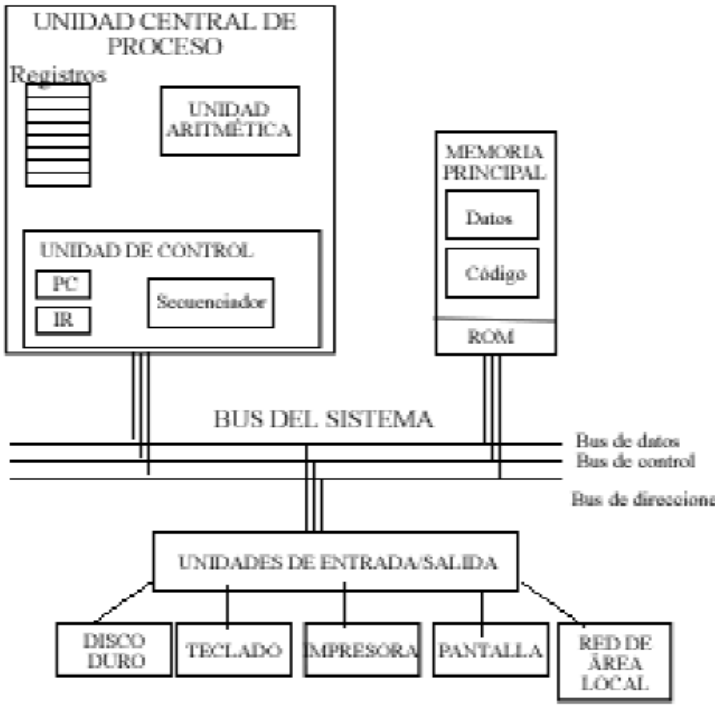
1.2 HARDWARE Y SOFTWARE

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

Un sistema informático está sostenido por dos elementos básicos:

- El elemento físico, conocido por el nombre de hardware.
- El elemento lógico conocido por el nombre de software

Además, existe un elemento humano, conocido como personal informático que se encarga de operar con el sistema informático.



1.3 ESTRUCTURA FUNCIONAL DE UN ORDENADOR.

Todos los computadores personales actuales tienen una estructura similar, en la cual destacan cinco componentes básicos: CPU, memoria, dispositivos de almacenamiento, unidades de entrada/salida y los buses.

1.3.1 MEMORIA PRINCIPAL

La memoria principal es la unidad donde se almacenan los *datos e instrucciones* necesarios para realizar un determinado proceso. Es rápida, y está estrechamente ligada a las unidades funcionales más rápidas dentro de la computadora (la UC y la ALU).

Aquí es donde deben ser cargados los programas para poder ejecutarse.

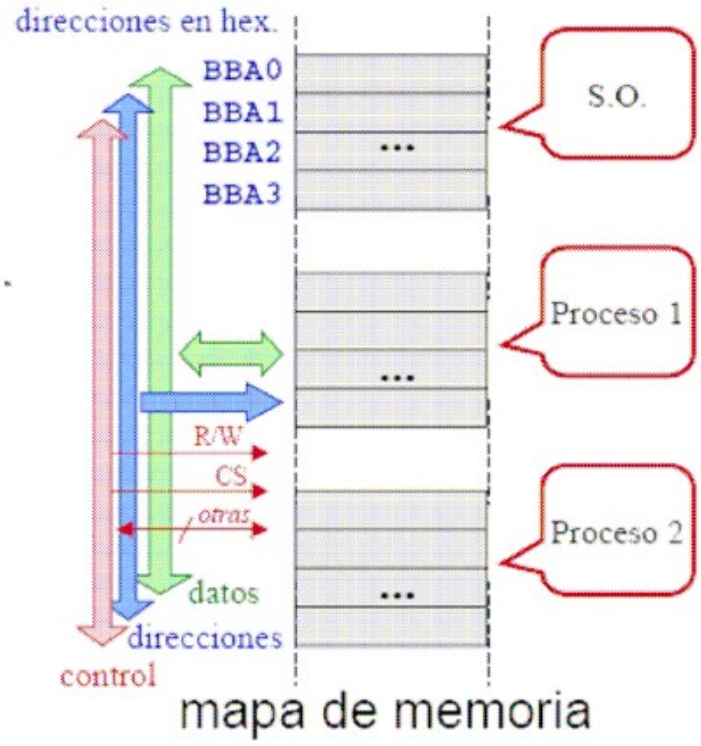
La memoria principal está formada por circuitos electrónicos capaces de almacenar sólo dos valores (0 ó 1) en cada elemento o celda de memoria. Sin embargo no accedemos a cada una de las celdas de la memoria individualmente, sino que estas están agrupadas en palabras de memoria que suelen ser de 32 o 64 bits.

Así, independientemente de la memoria física utilizada, la UCP ve lógicamente laMP como un conjunto deposiciones o *celdas* donde leer y escribir que se llama *mapa de*

memoria y cada posición del mapa corresponde a una *palabra* y se numera consecutivamente mediante una *dirección*. Como se puede ver en la figura.

Existen dos tipos principales de memoria central:

- La memoria ROM, de lectura y permanente, donde se almacena la BIOS (Basic Input-Output System) es el conjunto de programas que se ejecuta al encender el ordenador: chequea el sistema y carga el SO en la RAM.
- La memoria RAM para almacenamiento temporal de la información. Desde el punto de vista del programador ésta es la más interesante. En ella se guardan todos los datos, tanto de entrada como resultados intermedios y definitivos de las operaciones realizadas durante la ejecución de programas, así como las instrucciones que forman los propios programas utilizando los siguientes segmentos: de código, de datos, de pila y el montículo o heap para almacenar los datos dinámicos.



1.3.2 UNIDAD CENTRAL DE PROCESO.

La unidad central de proceso (CPU) es el elemento principal del computador, porque es donde se ejecutan las instrucciones que se leen de la memoria RAM. Tiene tres elementos principales:

- **Unidad de Control**: Encargado de controlar las acciones del resto de las acciones del resto de las unidades, interpretando y ejecutando las instrucciones en la secuencia adecuada
- **Unidad Aritmético Lógica (ALU):** Su misión consiste en realizar todas las operaciones aritméticas elementales (suma, resta, multiplicación, división) y las operaciones lógicas (por ejemplo, la comparación de dos valores)
- **Registros del microprocesador:** Desde el punto de vista del programador es interesante conocer la existencia de bancos de memoria de alta velocidad muy especializados conocidos como registros del microprocesador, donde se almacenan algunos datos e instrucciones del programa mientras se ejecuta.

Pasaremos a continuación a estudiar por su interés más detalladamente la unidad de control

Unidad de Control.

Su funcionamiento es el siguiente:

- Lee de la Memoria la instrucción del programa que se ejecutará en cada momento e identifica la operación a realizar.
- Busca los operandos que intervienen en la operación.
- Ejecuta la operación informando a la unidad responsable, por ejemplo: si es una operación aritmética o lógica a la ALU, si es una operación de entrada o de salida a las unidades de entrada/salida, etc.
- Guarda en la memoria el resultado de la operación realizada.

Para realizar todas estas operaciones la unidad de control necesita una serie de registros. Los más importantes son:

- ***El registro Contador de Programa***. Es un registro especial que almacena, en todo momento, la dirección de memoria donde reside la instrucción que se

ejecutará a continuación. Cuando se inicia la ejecución de una instrucción (operación), el contenido del registro contador se incrementa siempre en una unidad.

- ***El registro de Instrucción*.** Almacena la instrucción que se está ejecutando en cada momento. El registro puede dividirse (al menos a nivel conceptual) en dos partes bien diferenciadas: *Parte de código de operación y Parte de dirección(s) de operando(s)*

- ***Secuenciador***. Es un dispositivo que se encarga de interpretar la información contenida en el registro de instrucción. Una vez interpretada ésta, se encarga de activar en orden los circuitos apropiados para que la operación se complete con éxito.

Una vez identificados las unidades funcionales básicas de la unidad de control, veamos esquemáticamente en qué forma interactúan sus componentes, y en qué orden:

- El secuenciador activa los circuitos necesarios para leer de memoria el contenido de la celda cuya dirección indica el registro contador de programa.

- El contenido de dicha celda de memoria se transfiere al registro de instrucción.

- El contador de programa se incrementa en una unidad

- El secuenciador interpreta la parte de código de operación, del contenido del registro de instrucción, localiza los datos que intervendrán en la operación, utilizando la parte del registro de dirección de operandos, y hace que dichos datos se transfieran, junto con el tipo de operación que se debe realizar, a la unidad correspondiente.

- El proceso vuelve a comenzar

1.3.3 UNIDADES DE ENTRADA/SALIDA

Está formado por los dispositivos de entrada/salida.

- Los dispositivos o periféricos de entrada son los componentes hardware encargados de introducir la información desde el exterior para su posterior

proceso. Un ejemplo de dispositivo que se utiliza para la entrada es el teclado, conocido comúnmente como dispositivo estándar de entrada.

- Dispositivos o periféricos de salida. Son los componentes hardware encargados de hacer llegar al exterior los resultados procedentes de los procesos realizados en el sistema informático. Un ejemplo de dispositivo utilizado para la salida es el monitor, conocido comúnmente como dispositivo estándar de salida.

- Existen dispositivos que permiten la comunicación en ambos sentidos, por ejemplo el módem.

1.3.4 DISPOSITIVOS DE ALMACENAMIENTO AUXILIAR

También llamado dispositivo de almacenamiento secundario o memoria auxiliar. Son unidades de almacenamiento masivo de información mucho más lentas que la memoria central y con una capacidad mayor. Estas memorias son utilizadas para guardar datos y programas de forma permanente a diferencia de la memoria RAM, que se borra cuando se apaga el ordenador. Cuando la CPU requiera estos datos, deberán transferirse desde el dispositivo de almacenamiento a la memoria central para su proceso. El principal elemento de almacenamiento auxiliar de información es el disco duro.

1.3.5 BUSES

Es evidente que todos los dispositivos hardware del ordenador deben comunicarse de algún modo para que la información pueda transmitirse de uno a otro. Se necesitan vías por donde circule dicha información. A los distintos circuitos encargados de la conexión entre los microprocesadores, unidades de memoria y el resto de los elementos del ordenador se les denomina buses.

Existen tres tipos buses:

- **Bus de datos:** Es un conjunto de líneas especiales que permiten la transmisión de datos en paralelo y en ambos sentidos.

- **Bus de direcciones:** Es un conjunto de líneas especiales que contiene la dirección a la que van o de la que viene los datos que, en ese momento circulan por las líneas del bus de datos. Es un bus que tiene un único sentido, del procesador a la memoria, unidad de entrada salida o dispositivos de

almacenamiento externo

- **Bus de control:** Por este bus viaja información de control.

1.3.6 TIPOS DE SOFTWARE: BIOS, SISTEMAS Y APLICACIONES.

El software de un sistema informático es el conjunto de elementos lógicos, programas, dato, información, etc, que hacen posible el uso y funcionamiento de los ordenadores.

Se puede decir que los elementos básicos del software son los datos y las órdenes o instrucciones. Si el software forma parte del sistema informático, deberá almacenarse en un soporte físico como la memoria central o la memoria secundaria.

Se puede clasificar el software como software básico y software de aplicación.

- La BIOS es el software más básico del ordenador. Se utiliza para reconocer el hardware básico y cargar el sistema operativo en memoria RAM para ejecutarlo. Viene integrado en la placa base de cada ordenador.

- El software básico, o software de sistema, es el conjunto de programas imprescindibles para gestionar el hardware del sistema informático, sistema operativo, software de comunicaciones, etc.

- El software de aplicación está formado por un conjunto de programas diseñados con el objetivo de que los ordenadores realicen trabajos específicos, facilitando al usuario la realización de sus actividades. Son software de aplicación las aplicaciones ofimáticas, programas de dibujo, compiladores, etc.

2 LENGUAJES DE PROGRAMACIÓN

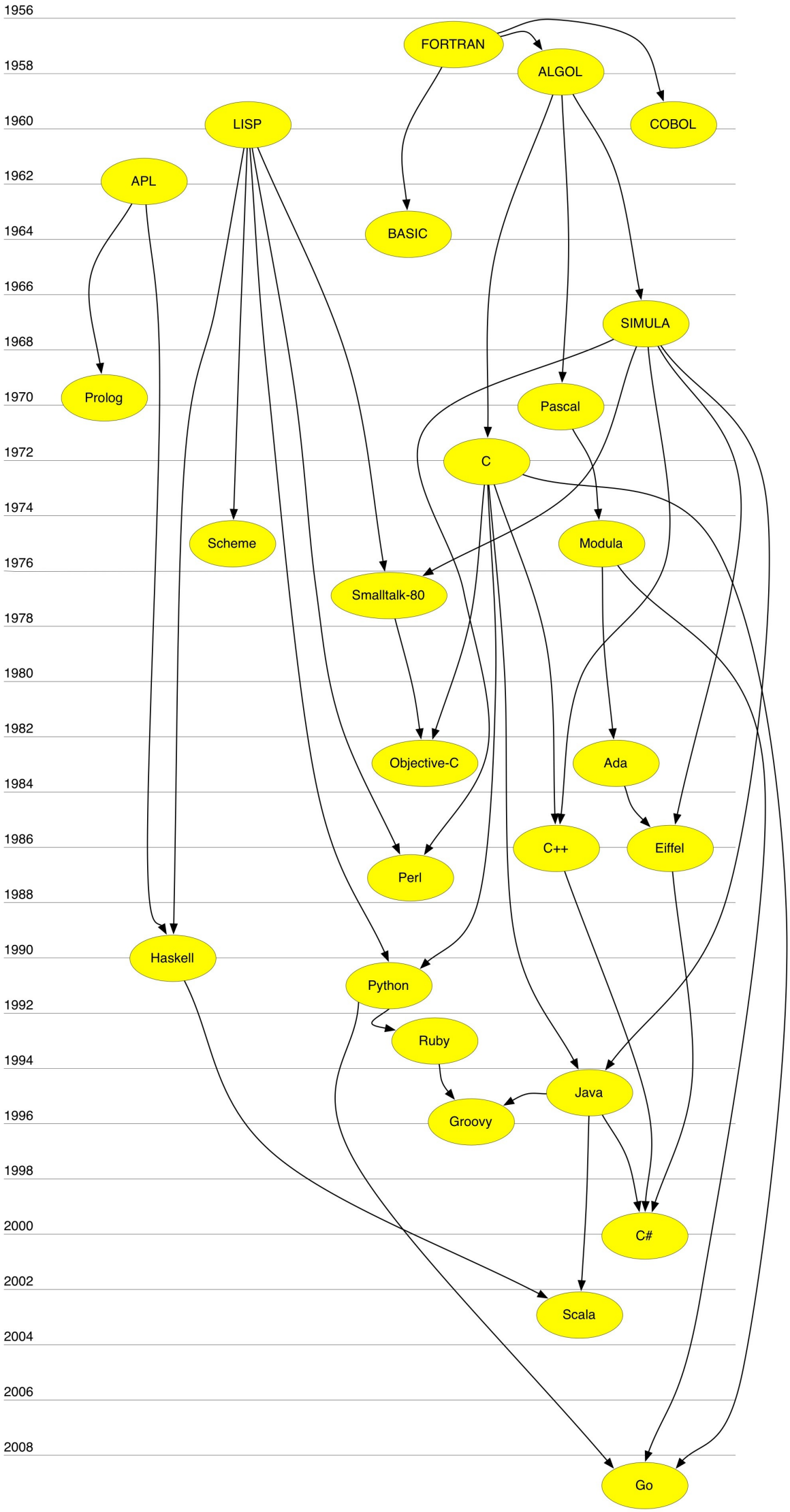
Un lenguaje de programación es una notación para describir algoritmos y estructuras de datos para resolver problemas concretos en un ordenador.

Los lenguajes de programación se pueden clasificar

- Según la evolución histórica
- Según su nivel de abstracción
- Según la forma de ejecución
- Según el paradigma de programación

3 CLASIFICACIÓN SEGÚN SU EVOLUCIÓN HISTÓRICA

Ver vídeo y presentación


1956
1958
1960
1962
1964
1966
1968
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
FORTRAN
ALGOL
COBOL
LISP
APL
BASIC
SIMULA
Prolog
Pascal
C
Scheme
Modula
Smalltalk-80
Objective-C
Ada
C++
Eiffel
Perl
Haskell
Python
Ruby
Java
Groovy
C#
Scala
Go

4 CLASIFICACIÓN SEGÚN SU NIVEL DE ABSTRACCIÓN

4.1 LENGUAJES DE BAJO NIVEL

Son lenguajes de programación dependientes de arquitectura de la máquina que los soporta. Los programadores están obligados a conocer perfectamente la máquina, ya que estos lenguajes manejan directamente recursos del sistema: memoria, registros del microprocesador ...

4.1.1 LENGUAJE MÁQUINA

Es el lenguaje usado directamente por el procesador y está formado por un conjunto de instrucciones codificadas en binario.

La tecnología de la máquina tan sólo le permite entender en 0 y 1 (bits) y es de esa forma, como se codificaban las órdenes en los comienzos de la informática.

El segmento de código en lenguaje Java es:

```
int counter = 0;

counter = counter + 1;
```

podría ser traducido a lenguaje de máquina como

```
00010100010001000100010000100010101011110

00000111010100011111000010001000001010101O
```

Antes de los años 50, los programadores eran los encargados de introducir los códigos binarios correspondientes a las distintas operaciones que debía realizar la computadora. Los programas eran totalmente dependientes de la máquina.

Los programas en código máquina eran difíciles de leer e interpretar, por ello surgen los lenguajes de programación.

4.1.2 LENGUAJE ENSAMBLADOR

Para paliar el uso del lenguaje máquina y como evolución de los mismos aparecen los lenguajes ensambladores. La idea surge del uso de palabras mnemotécnicas, en lugar

de las largas secuencias de ceros y unos, para referirse a las distintas operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.

```
;name of the program:one.asm
;
.model small
.stack
.code
        mov AH,1h     ;Selects the 1 D.O.S. function
        Int 21h       ;reads character and return ASCII
                              ; code to register AL
        mov DL,AL     ;moves the ASCII code to register DL
        sub DL,30h    ;makes the operation minus 30h to
                              ; convert 0-9 digit number
        cmp DL,9h     ;compares if digit number it was
                              ; between 0-9
        jle digit1    ;If it true gets the first number
                              ; digit (4 bits long)
        sub DL,7h     ;If it false, makes operation minus
                              ; 7h to convert letter A-F digit1:
        mov CL,4h     ;prepares to multiply by 16
        shl DL,CL     ;multiply to convert into four bits upper
        int 21h               ;gets the next character
        sub AL,30h    ;repeats the conversion operation
        cmp AL,9h     ;compares the value 9h with the content
                              ; of register AL
        jle digit2    ;If true, gets the second digit number
        sub AL,7h     ;If no, makes the minus operation 7h
                      ; digit2:
        add DL,AL     ;adds the second number digit
        mov AH,4CH
        Int 21h       ;21h interruption
        End     ;finish the program code
```

El ordenador no puede ejecutar directamente un programa escrito en lenguaje ensamblador, por lo que es indispensable un programa capaz de realizar la traducción al lenguaje máquina.

Este programa se conoce como *Ensamblador* siendo también específico de cada tipo de procesador.

Los programadores están obligados a conocer perfectamente la máquina, ya que estos lenguajes manejan directamente recursos del sistema: memoria, registros del microprocesador …

Los programas son totalmente dependientes de la máquina

4.2 LENGUAJES DE ALTO NIVEL

Los programadores de la etapa anterior (hasta el 1959 no surge el manual de

programación del UNIVAC) seguían obligados a pensar a la hora de diseñar los algoritmos en términos de instrucciones de máquina básicas, aún muy alejados de la manera natural en que nos comunicamos las personas.

Siguiendo en el camino de acercamiento hombre- máquina y en el intento de paliar los problemas derivados del uso de ensambladores, se desarrollaron los llamados lenguajes de alto nivel.

Están orientados al programador, en lugar de a la máquina, por lo que utilizan palabras del lenguaje natural (en inglés), por lo que son más fáciles de leer, escribir y modificar que los lenguajes de bajo nivel.

Son independientes de la máquina (aunque pueden incorporar instrucciones no estándares) y por tanto, el programador no necesita conocer el hardware específico de la máquina., C++, C#, Java, Logo, Python, Pascal, C...

5 CLASIFICACIÓN SEGÚN LA FORMA DE EJECUCIÓN

5.1 LENGUAJES COMPILADOS

Los algoritmos escritos en un lenguaje de alto nivel necesitan para su ejecución un programa llamado compilador, capaz de realizar la traducción al lenguaje máquina (Turbo Pascal, Turbo C, Borland C++, Visual C++).

La traducción se efectúa de manera que cada instrucción escrita en lenguaje de alto nivel se transforma en una o varias instrucciones en lenguaje máquina.

El compilador traduce completamente el texto escrito en lenguaje de alto nivel y una vez acabada la traducción, informa de los posibles errores. El programador deberá corregir esos errores y, sólo entonces, se genera la traducción lista para ejecutar.

5.2 LENGUAJES INTERPRETADOS

Existen lenguajes de alto nivel cuyos traductores, llamados Intérpretes, realizan lo que se podría llamar traducción simultánea, es decir, cada instrucción escrita en un lenguaje interpretado es analizada, traducida y ejecutada tras su verificación.

La principal diferencia con los lenguajes compilados, es que una vez traducida cada instrucción, se ejecuta y a continuación se elimina. No se guarda una versión del

programa traducido a código máquina.

Suelen ser más lentos que los compilados, ya que han de ser traducidos mediante se ejecutan.

Se necesita tener el intérprete ejecutándose en la máquina para ejecutar el programa, en los compilados no se necesita tener el compilador.

Ejemplos de lenguajes interpretados: PHP, JavaScript, Python, Perl, Logo Rubi, ASP, Basic, etc.

Actualmente hay lenguajes en el mercado que se podrán llamar pseudo-compilados o pseudo- interpretados, por ejemplo Java. El código fuente se compila para obtener un código binario en forma de byte-codes, que son estructuras similares a las instrucciones de máquina, con la característica de no ser específicas de ninguna máquina en particular. Este código es interpretado por la máquina virtual Java, que es en definitiva quien lo traduce al código máquina del procesador.

5.3 MÁQUINAS VIRTUALES DE PROCESO

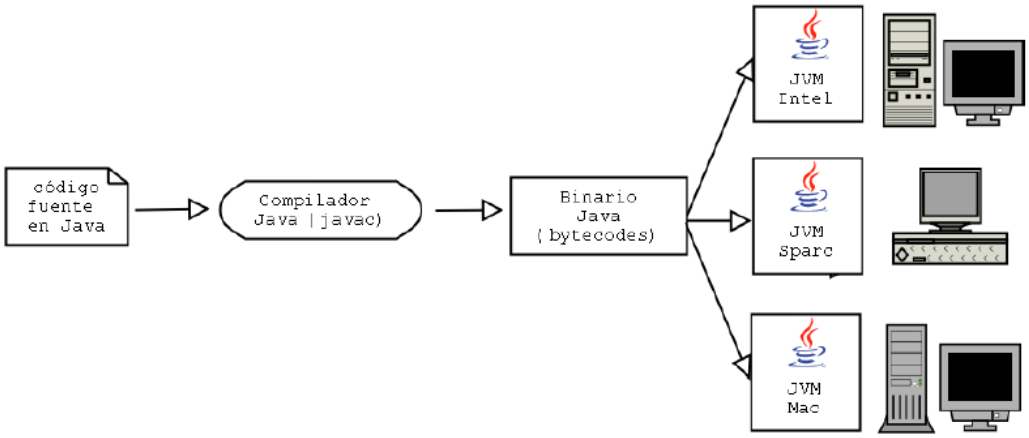
Las máquinas virtuales de proceso se utilizan para independizar la ejecución de un proceso del hardware subyacente, el más conocido es la máquina virtual Java.

Es importante no confundir las máquinas virtuales de proceso con las máquinas virtuales de sistema, ejemplos de software para crear máquinas virtual de sistema son Vmware Workstation y Virtual Box.

Una vez que escribamos y compilemos nuestro codigo fuente, podremos llevar el archivo binario a cualquier ordenador que tenga instalado una **Máquina Virtual de Java** (JVM, *Java Virtual Machine*) y se ejecutará exactamente igual, independientemente de la arquitectura software y hardware de ese ordenador.

¿Y qué es la JVM?. Observemos la figura, en la que se muestra la compilación de un programa Java. Comenzamos escribiendo nuestro código fuente Java. No nos tenemos que preocupar de las peculiaridades del S.O. ni del ordenador en el que se vaya a ejecutar ese código.

Una vez escrito, lo compilamos con el compilador de Java, que nos genera un archivo de ***bytecodes***. Los *bytecodes* son una especie de *código intermedio*, un conjunto de instrucciones en un lenguaje máquina independiente de la plataforma.



Compilación de programas en Java.

Cuando queramos ejecutar nuestro programa, la JVM instalada en el ordenador leerá el archivo de *bytecodes* y lo interpretará **en tiempo de ejecución**, traduciéndolo al código nativo de la máquina en la que se está ejecutando en ese momento. Por tanto, la JVM es un intérprete de *bytecodes*.

Para instalar el eatorno de ejecución de la máquina virtual Java hemos de acceder a la web de Oracle y donde aparece JRE (Java Runtime Enviroment) descargar el fichero que nos interese según el sistema operativo del ordenador: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

Investiga cuál es la diferencia con el JDK.

Requisitos Java en https://www.java.com/es/download/help/sysreq.xml

6 CLASIFICACIÓN SEGÚN EL PARADIGMA DE PROGRAMACIÓN

Existe una enorme variedad de lenguajes de programación, no sólo en cuanto a su sintaxis, sino también en cuanto a su comportamiento o semántica.

Hemos visto que cada año el número de lenguajes se incrementa, de forma que para los informáticos es prácticamente imposible conocer cada nuevo lenguaje que aparece. Pero eso no es un problema, ya que todos esos lenguajes tienen características comunes y se pueden agrupar en cuatro grandes grupos o modelos computacionales llamados paradigmas.

Todos los lenguajes pertenecen a uno de esos cuatro paradigmas. De forma que, si se conocen las características de los paradigmas de programación, es muy sencillo

aprender a programar en un nuevo lenguaje, porque tendrá las características del paradigma de programación al que pertenezca.

El origen de la palabra paradigma entendida como un marco general en el que se desarrollan teorías científicas se encuentra en el trabajo de 1962 del filósofo e historiador de la ciencia Thomas S. Kuhn, La estructura de las revoluciones científicas. Esa palabra ha sido después adoptada por el mundo de la computación para definir un conjunto de ideas y principios comunes de grandes grupos de lenguajes de programación.

La definición de la palabra paradigma más cercana a lo que se quiere decir en la expresión paradigma de programación es la siguiente:

Un marco filosófico y teórico de una escuela o disciplina científica en el que se formulan teorías, leyes y generalizaciones y los experimentos realizados en soporte de ellas.

Un paradigma define un conjunto de reglas, patrones y estilos de programación que son usados por un grupo de lenguajes de programación.

Podemos distinguir cuatro grandes paradigmas de programación:

- Paradigma funcional
- Paradigma lógico
- Paradigma estructurado
- Paradigma orientado a objetos

Una reflexión importante es que la separación entre los paradigmas y los lenguajes no es estricta. Existen ideas comunes a distintos paradigmas, así como lenguajes de programación que soportan más de un paradigma.

Otros paradigmas de programación menos comunes:

- Paradigmas de programación paralela y concurrente
- Paradigmas basados en restricciones
- Paradigmas visuales

Veamos algunas características importantes de los paradigmas más importantes.

6.1 Paradigma funcional

El programa es una colección de funciones matemáticas cada una de ellas con su entrada y su resultado. Las funciones interactúan y se combinan las unas con las otras utilizando condicionales, recursivas y composición funcional.

Lenguajes: LISP, Scheme, Haskell, Scala, Clojure.

Ejemplo de código (LISP):

```
(define (factorial x)
        (if (= x 0)
                1
                (* x (factorial (- x 1)))))
(factorial 8)
40320
(factorial 30)
265252859812191058636308480000000
```

6.2 Paradigma lógico (declarativo)

El programa es una colección de declaraciones lógicas sobre el resultado que debería conseguir una función, en lugar de cómo debería obtenerse dicho resultado. La ejecución del programa aplica estas declaraciones para obtener una serie de soluciones posibles a un problema. La programación lógica ofrece un vehículo natural para la expresión no determinista, ya que las soluciones a muchos problemas no son únicas, sino múltiples.

Lenguajes: Prolog, Mercury, Oz.

Ejemplo de código (Prolog):

```
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'debora').
hijode(A,B) :- padrede(B,A).
abuelode(A,B) :- padrede(A,C), padrede(C,B).
hermanode(A,B) :- padrede(C,A) , padrede(C,B), A \== B.
familiarde(A,B) :- padrede(A,B).
familiarde(A,B) :- hijode(A,B).
```

```
familiarde(A,B) :- hermanode(A,B).
?- hermanode('juan', 'marcela').
yes
?- hermanode('carlos', 'juan').
no
?- abuelode('pablo', 'maria').
yes
?- abuelode('maria', 'pablo').
no
```

6.3 Paradigma estructurado

El programa es una serie de pasos, cada uno de cuales realiza un cálculo, recupera una entrada o tiene como resultado una salida. La abstracción procedimental es un bloque de creación esencial en la programación estructurada, al igual que las sentencias condicionales, asignaciones, bucles y secuencias.

Lenguajes de programación estructurada son Cobol, Fortran, C.

```
type
        tDimension = 1..100;
        eMatriz(f,c: tDimension) = array [1..f,1..c] of real;
        tRango = record
                f,c: tDimension value 1;
        end;
        tpMatriz = ^eMatriz;
procedure EscribirMatriz(var m: tpMatriz);
var filas,col : integer;
begin
        for filas := 1 to m^.f do begin
                for col := 1 to m^.c do
                        write(m^[filas,col]:7:2);
                writeln(resultado);
                writeln(resultado)
        end;
```

```
end;
```

6.4 Paradigma orientado a objetos

El programa es una colección de objetos que interactúan los unos con los otros trasladándose mensajes que transforman su estado. Las clases y la herencia de objetos son bloques de creación esenciales de la programación OO.

Lenguajes orientados a objetos son Smalltalk, Java, C++, Eiffel.

Ejemplo (Java):

```
public class Bicicleta {
	public int marcha;
	public int velocidad;
	public Bicicleta(int velocidadInicial, int marchaInicial) {
		marcha = marchaInicial;
		velocidad = velocidadInicial;
	}
	public void setMarcha(int nuevoValor) {
		marcha = nuevoValor;
	}
	public void frenar(int decremento) {
		velocidad -= decremento;
	}
	public void acelerar(int incremento) {
		velocidad += incremento;
	}
}
public class MountainBike extends Bicicleta {
	public int alturaSillin;

	public MountainBike(int alturaInicial, int velocidadInicial, int
                                                        marchaInicial) {
		super(velocidadInicial, marchaInicial);
		alturaSillin = alturaInicial;
	}

	public void setAltura(int nuevoValor) {
		alturaSillin = nuevoValor;
```

```
        }
}
public class Excursion {
        public static void main(String[] args) {
                MountainBike miBicicleta = new MoutainBike(10,10,3);
                miBicicleta.acelerar(10);
                miBicicleta.setMarcha(4);
                miBicicleta.frenar(10);
        }
}
```

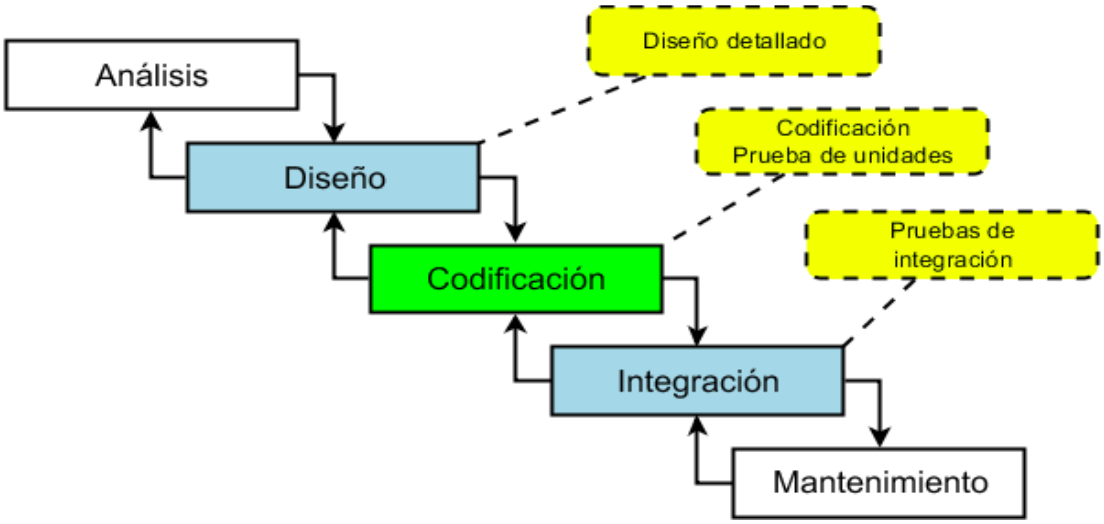
7 HERRAMIENTAS Y ENTORNOS PARA EL DESARROLLO DE PROGRAMAS.

Un **entorno de desarrollo de software** es una combinación de herramientas que automatiza o soporta al menos una gran parte de la tareas (o fases) del desarrollo: análisis de requisitos, diseño de arquitectura, diseño detallado, codificación, pruebas de unidades, pruebas de integración y validación, gestión de configuración, mantenimiento, etc. Las herramientas deben estar bien integradas, pudendo interoperar unas con otras.

Están formados por el conjunto de instrumentos (*hardware*, *software*, procedimientos, ...) que facilitan o automatizan las actividades de desarrollo.

- **CASE**: *Computer-Aided Software Engineering*
    - Con este término genérico se denominan los productos software que dan soporte informático al desarrollo
    - Sería deseable automatizar todo el desarrollo, pero normalmente se automatiza sólo en parte
    - Productos CASE: son cada uno de los instrumentos o herramientas software de apoyo al desarrollo

Las actividades mejor soportadas por herramientas de desarrollo son normalmente la centrales: codificación y pruebas de unidades. El conjunto de herramientas que soportan estas actividades constituyen lo que se llama un **entorno de programación**. A veces se utilizan las siglas **IDE** (*Integrated Development Environment*) para designar estos entornos, aunque no son un entorno de desarrollo completo, sino sólo una parte de él.



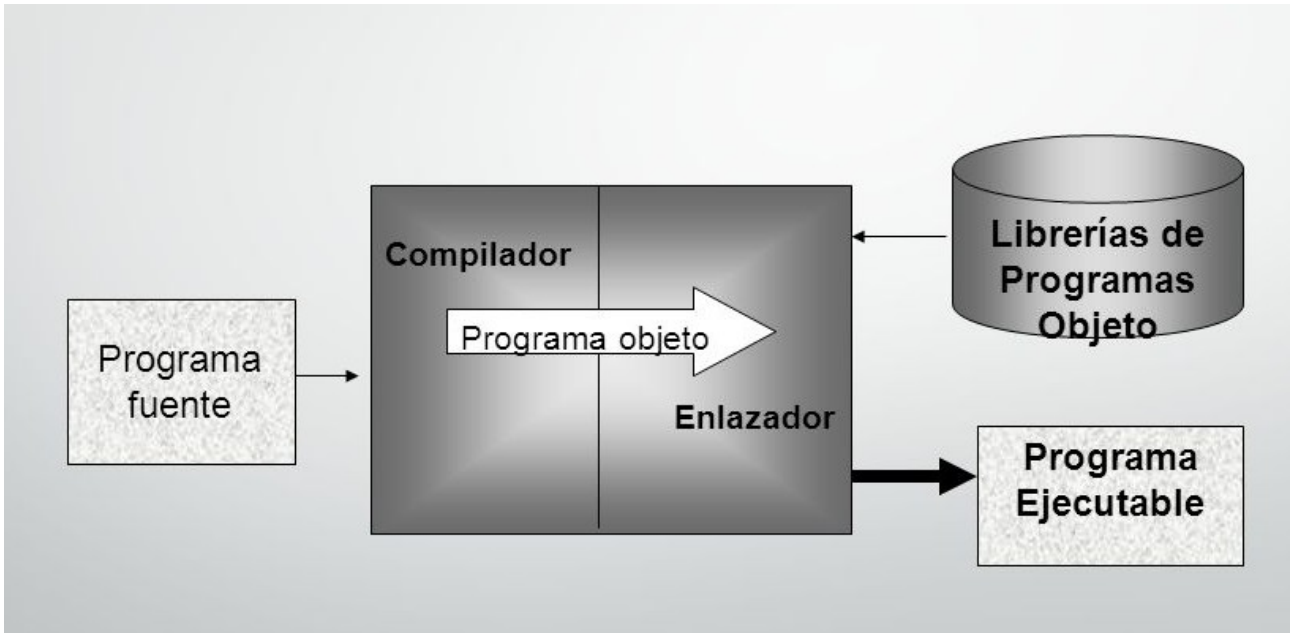
7.1 Funciones de un Entorno de Programación

Como se ha dicho, la misión de un Entorno de Programación es dar soporte a la preparación de programas, es decir, a las **actividades de codificación y pruebas**.

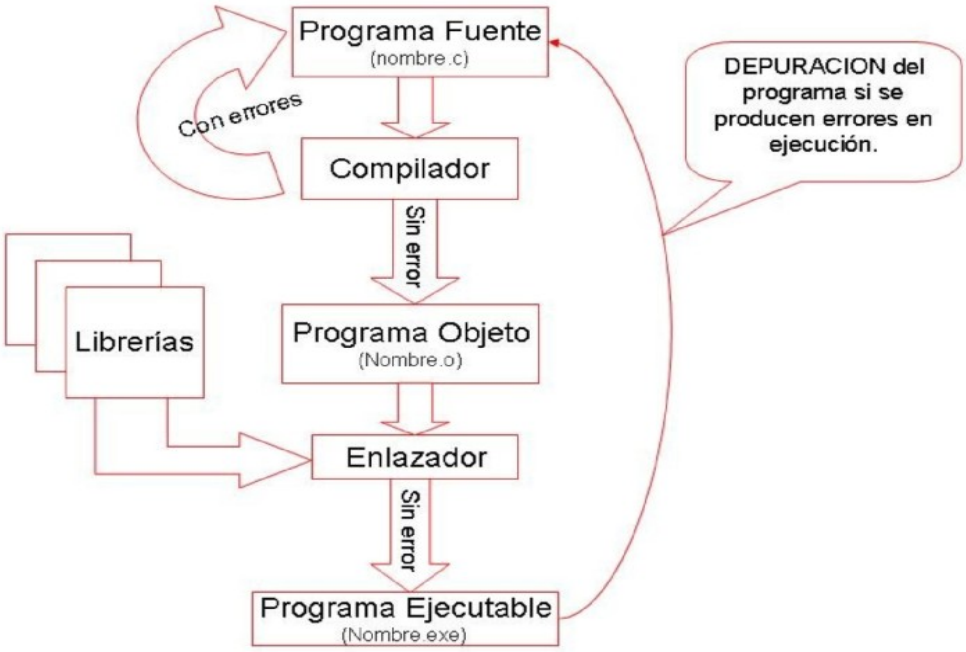
- Las tareas esenciales de la fase de codificación son:
  - Edición (creación y modificación) del código fuente
  - Proceso/ejecución del programa
    - Interpretación directa (código fuente)
    - Compilación (código máquina) - montaje - ejecución
    - Compilación (código intermedio) - interpretación
- Otras funciones:
  - Examinar (hojear) el código fuente
  - Analizar consistencia, calidad, etc.
  - Ejecutar en modo depuración
  - Ejecución automática de pruebas
  - Control de versiones
  - Generar documentación, reformar código
  - ... y otras muchas más ...

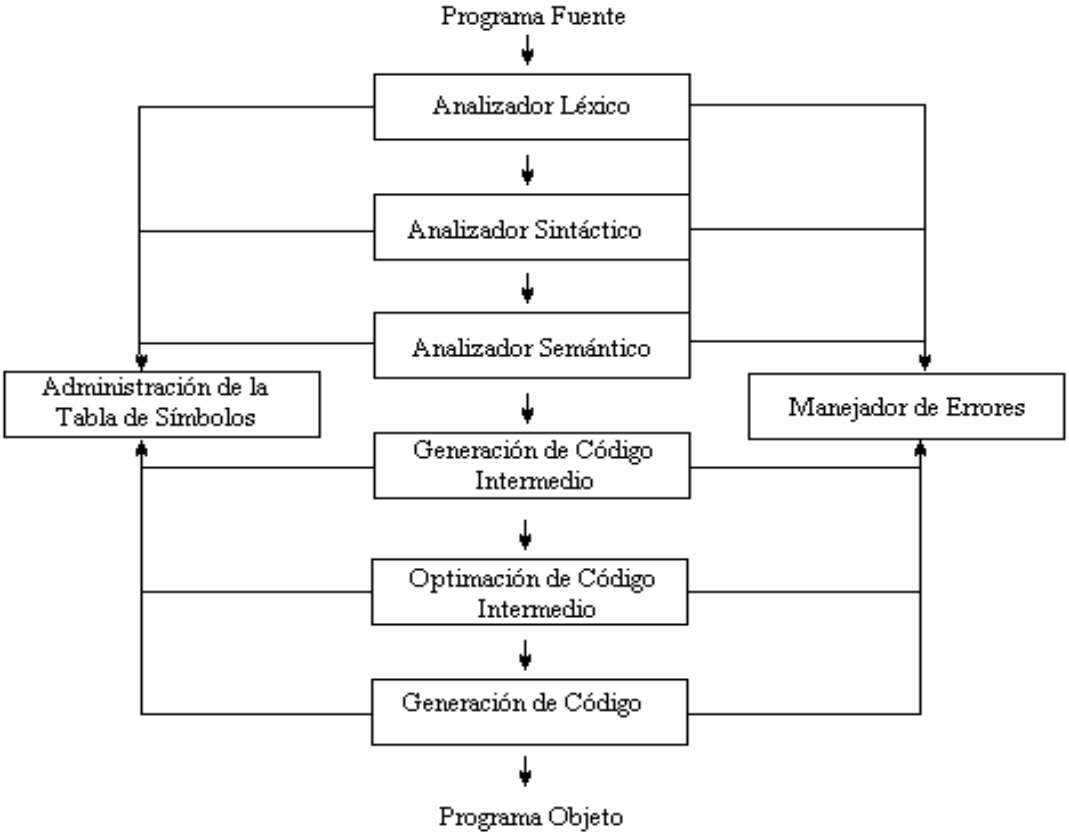
7.1.1 COMPILADORES

El proceso de traducción de un programa a código máquina se lleva a cabo mediante dos programas compilador y enlazador. Si el compilador en el proceso de traducción devuelve algún error (falta algún signo de puntuación, sentencias mal escritas, tipos de datos no compatibles, variables no definidas, etc) no se generará el programa objeto. Será necesario modificar el programa fuente y pasarlo de nuevo al compilador.

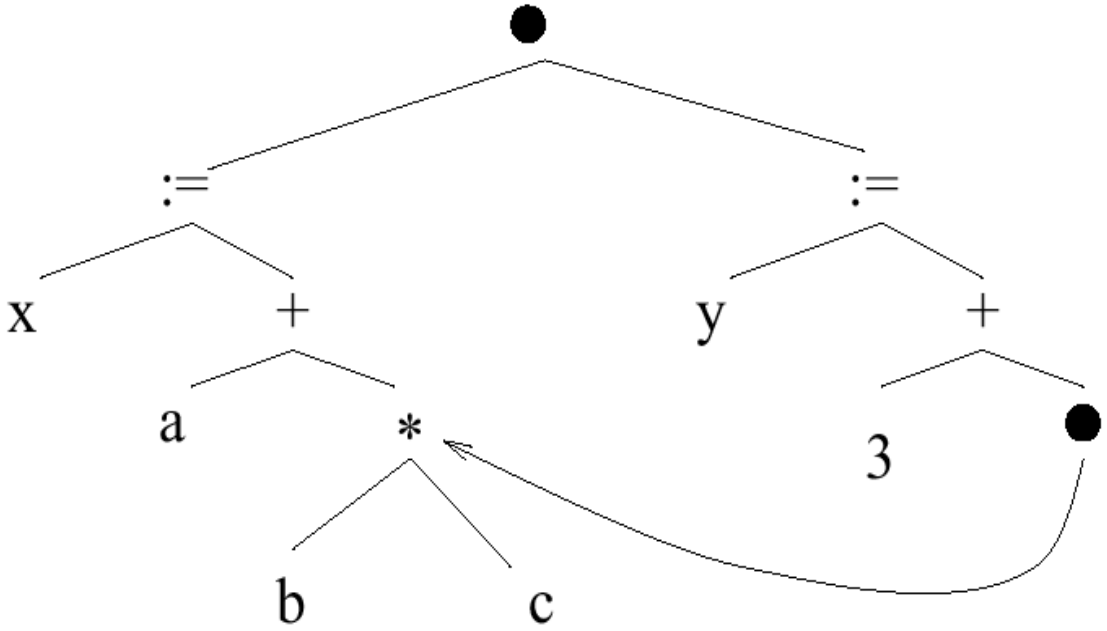


El compilador se compone internamente de varias etapas o fases que realizan distintas operaciones.





- **El analizador léxico**, también conocido como scanner, lee los caracteres uno a uno desde la entrada y va formando grupos de caracteres con alguna relación entre sí (tokens), que constituirán la entrada para la siguiente etapa del compilador. Cada token representa una secuencia de caracteres que son tratados como una única entidad. Por ejemplo, en Pascal un token es la palabra reservada BEGIN, en C: WHILE, etc.
- **El analizador sintáctico,** también llamado parser, recibe como entrada los tokens que le pasa el Analizador Léxico (el analizador sintáctico no maneja directamente caracteres) y comprueba si esos tokens van llegando en el orden correcto (orden permitido por el lenguaje). La salida "teórica" de la fase de análisis sintáctico sería un árbol sintáctico.



- **El análisis semántico** es posterior al sintáctico y mucho más difícil de formalizar que éste. Se trata de determinar el tipo de los resultados intermedios, comprobar que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, etc. En definitiva, comprobará que el significado de lo que se va leyendo es válido.
- **Generación de código intermedio:** después del análisis se genera una representación intermedia similar al código máquina con el fin de facilitar la tarea de traducir al código objeto.
- **Optimización de código:** trata de mejorar el código intermedio generado en la fase anterior, de tal forma que el código resultante sea más fácil y rápido de interpretar por la máquina.
- **Generación de código:** genera el código objeto de nuestro programa.

El programa enlazador inserta en el código objeto las funciones de librería necesarias para producir el programa ejecutable.

7.1.2 INTÉRPRETES

Mientras que el objetivo de los compiladores es obtener una traducción del programa fuente a otro lenguaje, los intérpretes tienen como objeto la obtención de los resultados del programa.

Para ello deben realizar dos tareas: analizar su entrada y llevar a cabo las acciones especificadas por ella.

La parte de análisis puede realizarse de manera idéntica a como se lleva a cabo en los

compiladores. Es la parte de síntesis la que se diferencia sustancialmente. En el caso de la interpretación, se parte del análisis y el árbol de sintaxis abstracta y se recorre, junto con los datos de entrada, para obtener los resultados.

Actualmente es habitual encontrar híbridos entre la compilación y la interpretación que consisten en compilar a un lenguaje intermedio para una máquina virtual y después interpretar este lenguaje. Esta aproximación es la que se sigue, por ejemplo, en Java, o la plataforma .NET. para obtener los resultados.

8 ERRORES Y CALIDAD DE LOS PROGRAMAS

La detección, corrección y prevención de errores suponen tareas muy importantes desarrolladas en todas las fases del proceso de desarrollo de programas correctos.

En el desarrollo de un programa, según la etapa donde se producen se pueden considerar diferentes tipos de errores.

8.1 ERRORES DE ESPECIFICACIÓN Y DISEÑO

Se produce por el uso de algoritmos diseñados a partir de especificaciones incorrectas del problema y tiene como consecuencia que el programa no haga lo que debe hacer. Son errores difíciles y costosos de corregir puesto que suponen repetir una gran parte del proceso de desarrollo del software.

8.2 ERRORES EN TIEMPO DE COMPILACIÓN

Son errores detectados por el compilador cuando intenta traducir el programa fuente. Se pueden dividir en los siguientes tipos:

- **Errores lexicográficos**: Son producidos por la aparición de elementos no reconocibles por el compilador. Ejemplo: maim en lugar de main
- **Errores sintácticos:** Son provocados por el incumplimiento de las reglas de sintaxis del lenguaje y detectados por el compilador al no reconocer una tira de caracteres como una secuencia con formato válido de instrucción. Ejemplo: S=A+B/
- **Errores semánticos:** El compilador avisa de una incorrección por falta de significado Por ejemplo, la advertencia del uso en el programa de una variable

no declarada o If (a=b)

8.3 ERRORES DE ENLAZADO

Son debidos, principalmente, a que el compilador no encuentra los módulos de enlace porque:

- No se ha indicado correctamente la ruta donde se encuentran los módulos
- Los módulos de enlace no se encuentran definidos en ese lugar.

Estos errores son relativamente fáciles de detectar y su corrección pasa por configurar el compilador para que encuentre correctamente dichos módulos o por colocar los módulos creados por el programador en el lugar adecuado.

8.4 ERRORES EN TIEMPO DE EJECUCIÓN

Son fallos del programa difíciles de detectar y solucionar. Algunos de ellos provocan la terminación anormal del programa o su caída (el sistema se ha colgado). Generalmente están relacionados con desbordamientos o con operaciones indeterminadas, por ejemplo, bucles infinitos, división por cero, cálculo de la raíz cuadrada de un número negativo.

Frecuentemente, se producen errores de ejecución por no haber prevenido en el programa los posibles fallos que pudieran cometer las personas que realizan la entrada de datos para dicho programa. Por ejemplo, si el programa espera un dato numérico y el usuario introduce una secuencia de caracteres que, por tanto, no se corresponde con el número, se provocará un conflicto entre tipos de datos y un error en tiempo de ejecución.

Un buen método para la prevención de errores provocados en la entrada de datos, en especial si ésta se produce desde el teclado consiste en comprobar explícitamente en el programa que los datos se encuentran en los rangos correctos. Será el propio programa quien tenga los mecanismos para actuar si no fuera así, en lugar de dejar que el sistema responda erróneamente en tiempo de ejecución. Este proceso se conoce como validación de los datos de entrada.

Existen muchas situaciones en los programas donde pueden y deben prevenirse errores potenciales. Corresponde a los programadores añadir a los programas el código necesario para el manejo y prevención de estos errores, evitando que se

produzcan condiciones que provoquen la parada del programa durante su ejecución.

A la capacidad de un programa para recuperarse cuando se ha producido una situación e un error se le denomina robustez. Existen algunos campos de aplicación de la informática sonde la robustez de los algoritmos no es una característica sólo deseable, sino absolutamente imprescindible, e incluso crítica. Por ejemplo en sistemas de control en naves espaciales, de misiles, pilotaje automático de aviones y barcos...

Otros errores producidos en tiempo de ejecución no provocan la terminación anormal, ni caída del sistema, simplemente producen resultados erróneos.

8.5 CALIDAD DEL SOFTWARE

No es suficiente que el programa funcione para que sea bueno, esto es sólo necesario. Para que sea bueno además debe estar hecho con calidad.

> El comienzo de la sabiduría de un ingeniero del software es reconocer la diferencia entre que un programa funcione y que lo haga correctamente.
>
> M.A. Jackson

Esta frase debe exhibirse en todas las aulas donde se enseñe a programar.

El software considerado de calidad debe reunir una serie de características que serán objetivos a la hora de realizar programas. Estas características son las siguientes:

- El programa funciona de acuerdo a los requisitos establecidos.
- Legibilidad y comprensión.
- Modificable sin invertir excesivo tiempo y esfuerzo
- Su realización debe atenerse al tiempo y presupuesto establecidos.