LECTURAS SOBRE COMPUTADORAS DIGITALES –LECTURA Nº7
MATERIA: ARQUITECTURA DE LAS COMPUTADORAS

LENGUAJES DE ALTO NIVEL

1.- Introducción

Los lenguajes hasta ahora analizados obligaban a que el programador conociera cómo la máquina resuelve el problema y, además, preguntarse cómo resolver el problema.

Eran lenguajes orientados a la máquina y, entre otros inconvenientes, implicaban la migración de lenguaje cada vez que el equipo cambiaba su arquitectura, no había compatibilidad entre las distintas estructuras ni entre los distintos lenguajes de máquina.

Por los motivos expuestos, se crearon los llamados lenguajes orientados al problema (Problem Oriented Language(POL es la sigla en inglés) que podían ser usados en distintas máquinas como Programa y con un programa compilador o traductor propio de cada máquina obtenerse el Programa Objeto, también propio de cada máquina.

La idea básica de estos lenguajes subyace en el concepto de Librería, que es una colección de palabras reservadas, macroinstrucciones o sentencias que convocan o son un programa en lenguaje de máquina.

A partir de aquí podemos decir que un Lenguaje de Computación es una colección de palabras reservadas que se encuentran sintetizadas en la Librería o Biblioteca del Lenguaje.

Al contrario de los lenguajes humanos que sirven para comunicar cualquier idea o sentimiento, los lenguajes de alto nivel están orientados a resolver un tipo de problema específico.

2.- Lenguaje FORTRAN

El primer lenguaje de alto nivel es el FORTRAN (FORmula TRAslator – Traductor de Fórmulas) que fue pensado para resolver cálculos numéricos o científicos, aunque puede usarse en otras áreas del conocimiento.

El FORTRAN fue desarrollado por John Backus entre 1954 y 1957 y pensado para una máquina en particular, la IBM 704, pero significó un paso conceptual significativo en la evolución de los lenguajes de computación pues significó alejarse de la arquitectura de la computadora y concentrarse en el problema.

El FORTRAN I, evolucionó a través del tiempo incorporando nuevas funciones a su Librería y nuevos procedimientos que incrementaron su utilidad, conoció versiones poco exitosas como el FORTRAN II y III y se estabilizó en el FORTRAN IV, llamado el Standard, y luego aparecieron nuevas versiones como el Fortran 2000.-

La librería del Fortran incluye funciones matemáticas tales como:

SQR(x) que convoca el programa objeto que resuelve la Raíz cuadrada de x
SIN(x) que convoca el programa objeto que resuelve el Seno de x
COS(x) que convoca el programa objeto que resuelve el Coseno de x
ABS(x) que convoca el programa objeto que resuelve el Valor absoluto de x
LOG(x) que convoca el programa objeto que resuelve el Logaritmo neperiano de x
EXP(x) que convoca el programa objeto que resuelve la potencia x del número e
INT(x) que convoca el programa objeto que resuelve el hallar la parte entera de x

Además, incluye los operadores lógicos como AND, OR y NOT, así como otras funciones.

En general, en un lenguaje de alto nivel hay que definir, en el momento del diseño, la sintaxis, la semántica, el modo de llegar a ejecutar un programa, el modo en el cuál se ingresan los datos e instrucciones y las etapas operativas.

El FORTRAN I, requería que la entrada de datos e instrucciones ingresaran a través de tarjetas perforadas, soporte físico en papel con posibilidades de contener un registro lógico de 80 caracteres.

Esto implica que el ingreso de datos e instrucciones se hacía mediante una lectora de tarjetas.

El programador escribía el programa en papel(ver figura 1) y un perfoverificador generaba un lote de tarjetas con el programa, una tarjeta por cada dato o instrucción.

La sintaxis del lenguaje puede resumirse en el texto extraído del libro "Lenguajes de Programación para micros" de G.Marshall, Ed.Paraninfo, Madrid, 1985:

"Los nombres de variables pueden constar de hasta seis caracteres: deben comenzar con una letra, pero los caracteres siguientes pueden ser letras o números. Los números que comienzan con cualquier letra de la I a la N, indican, por defecto, variables enteras y el resto son nombres de variables reales. Así la asignación:

CINCO =5,0

Asigna un valor real a una variable real, pero

IDOS=2

Asigna un valor entero a una variable entera…"

El sentido semántico está dado por la Librería, si SQR(x) es la raíz cuadrada de x, en el momento de la traducción se deberá sustituir SQR(x) por un Programa Objeto que, en el lenguaje de máquina, resuelve el problema de la raíz cuadrada de x.

El FORTRAN trabaja en el Modo Operativo COMPILADOR, es decir, el programa Fuente se Compila y se obtiene el Programa Módulo Objeto Reubicable, el mismo se Link-Edita y se obtiene el Programa Objeto. De la ejecución de este programa se obtiene la solución del problema.

Es de señalar, que todos los Compiladores de Lenguaje incluyen en su estructura la Librería, un programa que busca errores de sintaxis y semánticos que se llama del Debbuger(Desbichador).

La Compilación incluye tres etapas o análisis:

- Análisis sintáctico, que determina si una instrucción está bien escrita. El Debbugger detecta las sentencias mal escritas y lo indica en un listado al terminar la compilación.
- Análisis semántico que determina si las palabras que se usan en el programa son palabras que pertenecen al lenguaje. Si hay errores se listan al terminar la compilación.
- Análisis lexicográfico que, podríamos decir, es la traducción del lenguaje fuente al lenguaje objeto, sin direcciones absolutas.

Para una mayor claridad del modo operativo se brinda a continuación la Figura 1.

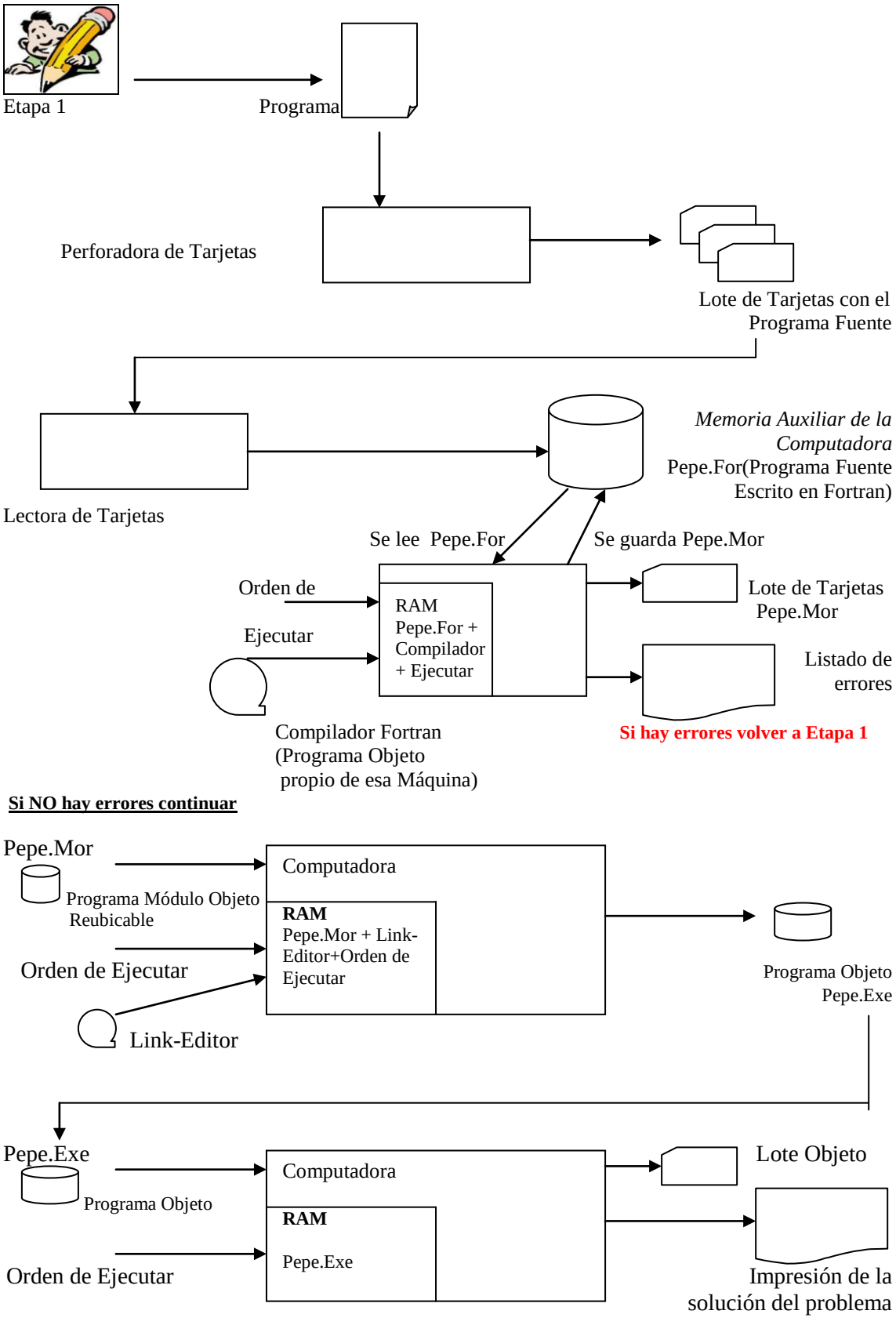

Figura 1: Etapas de operación de un programa escrito en lenguaje de alto nivel en los años cincuenta

En síntesis: El FORTRAN tiene una gran capacidad de cálculo por la gran cantidad de funciones matemáticas que incluye en su Librería, pero es pobre en lo que respecta a la E/S de datos y en la representación de la información de salida.

Es de destacar que los lenguajes de alto nivel valen para distintas máquinas pero para que la ellas lo puedan convertir en programa objeto, deben tener compiladores que sean propios de esas máquinas y que puedan tomar el Fuente genérico y transformarlo en Objeto acorde al lenguaje de máquina de esa máquina.-

3.- **Lenguaje COBOL**

El COBOL(**CO**mputer **B**usiness **O**riented **L**anguage - Lenguaje de Computación Orientado a los Negocios) es un lenguaje diseñado entre 1952 y 1959 por la capitana de navío de la Armada Norteamericana Grace Hope y está orientado a problemas administrativos y contables. A partir de 1960 se hizo cargo del proyecto un comité llamado CODASYL(Conference Data Systems Languages)

Es un lenguaje plenamente vigente en versiones tales como el ACU-COBOL, el ANSI-COBOL, el COBOL 97 y el COBOL 2000.

Como lenguaje comercial, el COBOL se destaca en el manejo de datos alfanuméricos, de forma que permite la realización de tareas tales como la lectura y actualización de registros y la cumplimentación automática de formularios.

Su Librería es pobre en funciones matemáticas complejas, lo que lo hace menos poderoso que el FORTRAN.

Hay que tener en cuenta que los problemas administrativos y contables presentan poco proceso y mucha entrada/salida de información y datos. El COBOL se adapta perfectamente a estas exigencias.

Su sintaxis hace que el lenguaje sea más legible que el FORTRAN y soporta instrucciones del tipo:

```
MOVE X TO Y
```

que hace que se puedan mover valores singulares o estructuras completas. El COBOL tiene algunas posibilidades aritméticas así como puede escribir operaciones matemáticas en forma muy legible. Una fórmula como:

$$x = a + 2b/c$$

puede programarse como:

```
COMPUTE X = A + 2 * B/C
```

O como:

```
DIVIDE C INTO B GIVING D
MULTIPLY 2 BY D
ADD D TO A GIVING X
```

En este último caso, el programador debe situar las instrucciones en el orden correcto para obtener el resultado deseado.

El COBOL dispone de instruciones READ y WRITE para entrada y salida, e instruciones condicionales como:

```
IF PEDIDO IS GREATER THAN 100
MULTIPLY  DESCUENTO BY PRECIO
```

Serán palabras reservadas del lenguaje:

Palabra Reservada	Sentido Semántico
COMPUTE	define un algortimo matemático
DIVIDE	define la División
INTO	señala operador
WRITE	egreso de datos
READ	ingreso de datos
ADD	define la suma
GIVING	indica la obtención de un resultado
Etc.	Etc.

El sentido semántico está dado por la Librería que transforma cada palabra reservada en su programa equivalente en lenguaje de máquina en el momento de obtener el programa objeto.

El COBOL tenía entrada por tarjetas perforadas en su primera versión y trabajaba en el Modo Operativo Compilador, por lo tanto la Figura 1 indica también el modo operativo de trabajo del COBOL, además del FORTRAN

La aparición de los lenguajes de alto nivel hizo que apareciera una divisikón en el trabajo del personal de informática, estaban los Programadores de Aplicaciones y los Programadores de Sistemas de Base(SO, Link-Editor y Compiladores).

La aparición del COBOL hizo que apareciera una Informática Administrativa, en contraposición de la Informática Científica que tenía su lenguaje en el FORTRAN.

Se puede hacer una pequeña comparación entre ambos programas a traves del siguiente ejemplo de ordenación de una secuencia de 10 números:

PROGRAMA EN FORTRAN

```
C    Ordenación de una secuencia de 10 números
     PROGRAMA BURBUJA
     INTEGER N(10)
     INTEGER  I,  J,  AUX
     DO  10  I=1,  10
          READ (* , *)  N(I)
10   CONTINUE
     DO 20  I = 9,  1,  -1
        DO  20,   J = 1,    I
           IF  (N(J) .GT.N(J + 1) ) THEN
                AUX=N(J)
                N(J) =N(J+1)
                N(J+1) = AUX
          END  IF
20    CONTINUE
     DO  30   I=1,  10
          WRITE (*, *)   N(I)
30   CONTINUE
     STOP
     END
```

PROGRAMA EN COBOL

```
      * Ordenación de una secuencia de 10 números
      IDENTIFICACTION DIVISION.
      PROGRAM-ID. BURBUJA.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      01       TABLA-N.
               05    N   PIC  9(4)  OCCURS 10  TIMES.
      01       VARIABLES
               05   I  PIC        9(4).
               05   J  PIC        9(4).
               05   AUX   PIC   9(4).
      01       SALIDA     PIC  Z(3) 9.
      PROCEDURE  DIVISION.
      PROCESO - PRINCIPAL.
    PERFORM VARYING  I  FROM   BY  1 UNTIL  I > 10
        ACCEPT   N   (I)
        END - PERFORM
PERFORM  ORDENAR
VARYING  I  FROM  9  BY  -1  UNTIL  I <  1
 AFTER  J  FROM  1  BY  1  UNTIL   J  >  I
 PERFORM VARYING I FROM 1 BY 1  UNTIL I  > 10
 MOVE N(I)  TO  SALIDA
  DISPLAY   SALIDA
  END-PERFORM
  STOP RUN
  ORDENAR
       IF  N(J)   >  N(J + 1)   THEN
           MOVE  N(J)    TO AUX
           MOVE  N(J+1)  TO  (J)
           MOVE  AUX     TO N (J+1)
     END-IF
```

4.- **Lenguaje BASIC: LA GRAN REVOLUCIÓN**

El Lenguaje BASIC significa un re enfoque del proceso operativo de en la programación, creado en 1965 por los profesores John G.Kemeny y Thomas E.Kurtz, del Dartmouth College(Estados Unidos), con el propósito de proporcionar a los principiantes un lenguaje fácil de aprender, como se indica en su nombre Beginner's All-purpose Symbolic Instruction Code(Código de instrucción simbólico de propósito general para principiantes).

Este lenguaje introduce una nueva forma operativa: el Modo Intérprete y la Edición directa de los programas fuente.

En la estructura del lenguaje existen distintos programas, el Editor, que es un programa que permite escribir y modificar programas fuente, el Traductor con su Librería, el Debbugger para buscar errores en forma dinámica y el Rodador que es un programa que permite correr los programas fuente.

Veamos el proceso de creación y ejecución de un programa mediante el diagrama operativo de la Figura 2.

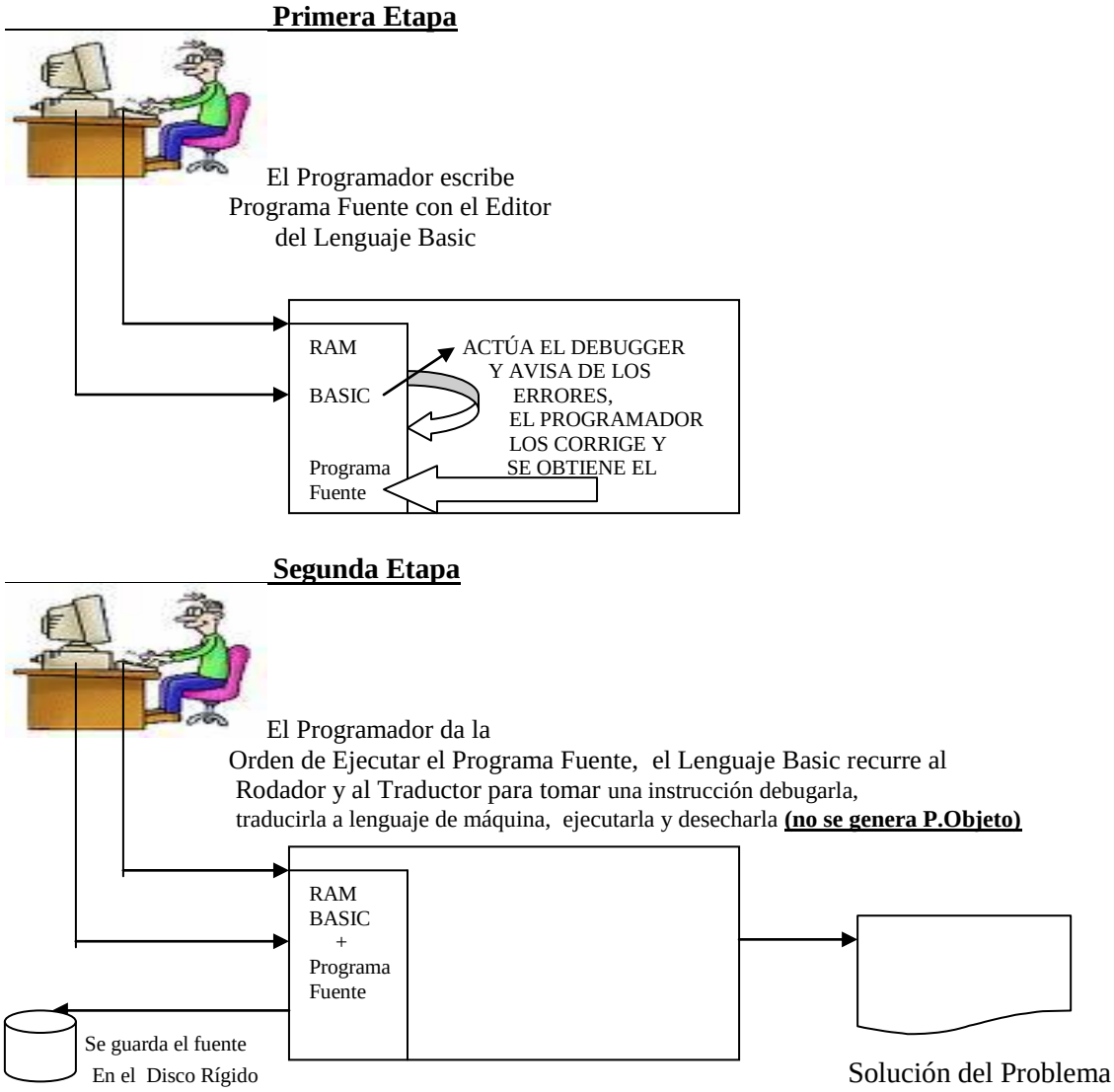

Figura 2: Trabajando en Basic en modo Intérprete

Como puede verse en el Modo Operativo Intérprete no hace falta escribir el problema en el papel, perforar las tarjetas del Programa Fuente, hacer que las lea la lectora de tarjetas y recién en es momento se encuentra en la memoria principal, sencillamente el Programador teclea el Programa Fuente, gracias a que el Lenguaje Basic incluye al programa Editor.

Esta fue el primer acierto del BASIC, ***la Edición***, la misma fue incorporada para trabajar con los lenguajes en modo operativo COMPILADOR, a través de un Editor que formó parte de los Sistemas Operativos, en primera instancia y luego la Edición se dejó en manos de los Procesadores de Texto, con los cuales se obtiene el Programa Fuente que luego será compilado.

El BASIC en modo Intérprete tiene la propiedad de buscar los errores cada vez que se digita ENTER de una sentencia del programa, esto acelera el tiempo de obtención del Programa Fuente Depurado(sin errores). Recordemos que en la Compilación había que terminar todo el Programa Fuente, luego compilarlo y allí aparecían los errores. Dicho de otra manera en el modo operativo Compilación se tarda más en la obtención del Programa Fuente Depurado.

El problema del Modo Intérprete es que en el momento de la ejecución, en memoria principal tiene que encontrarse el Lenguaje y el Programa Fuente, lo que implica tener mucha memoria, además como toma una instrucción, le busca los errores, la traduce y luego la ejecuta, es lenta la ejecución. En el Modo Operativo Compilación se obtiene el Programa Objeto y la ejecución es veloz pues solamente se ejecuta y además no se requiere la presencia del Compilador en memoria principal en el momento de la ejecución.

Cabe aclarar que los lenguajes son independientes del Modo Operativo, de hecho que la empresa Texas creó un BASIC en modo Intérprete para obtener el Programa Fuente Depurado y un Compilador BASIC para obtener a partir de ese Programa Fuente, el Programa Objeto, de esta manera se tienen las ventajas del Intérprete(obtener rápido el Fuente) y del Compilador(ser veloz en la ejecución pues se tiene un programa objeto).

Si alguien lo hace habrá un COBOL Compilador y un COBOL Intérprete o lo mismo para el FORTRAN o para otro lenguaje cualquiera. Es incorrecto decir el BASIC es Intérprete, pues también puede ser compilador…

El BASIC tiene una sintaxis propia, que puede resumirse de la manera siguiente:

- Las sentencias se numeran en forma secuencial
- Se deja un espacio en blanco y se escribe la sentencia
- En cada sentencia aparece la instrucción y el operando

El siguiente ejemplo en el cual se calcula el promedio de seis números y se lo imprime, puede aclarar los conceptos:

```
5   INPUT G,H,I,J,K,L
6   LET V1 =  (G+H+I+J+K+L)/6
7   PRINT V1
8   GO TO 5
25  END
```

Los números de sentencia pueden ser en una secuencia completa o pueden existir saltos como el de ir de 8 a 25, se aconseja dejar espacios vacantes entre los números para poder agregar sentencias.

Como puede verse en el ejemplo, el BASIC es fácil de entender y de usar. Un segundo ejemplo muestra algunas otras palabras reservadas del lenguaje:

```
20  LET Y = 10
30   READ Z
40    IF Z <  = Y THEN 70
50   PRINT "Z ES MAYOR  QUE"; Y
60   GO TO 90
70    PRINT "Z ES MENOR O IGUAL QUE"; Y
90   STOP
```

En el ejemplo, que compara un valor Z con el número 10, puede verse la sintaxis de la impresión y de la lectura y además los saltos.

Acorde a los dos programas vistos, el sentido semántico o contenido de la Librería abarca**, al menos:**

LET Nombre de la variable, para definir el valor de la misma.
READ Nombre de la variable, para definir la variable a leerse.
IF Nombre de la variable símbolo/s variable THEN, para hacer una comparación entre variables y un salto condicional.
PRINT "MENSAJE"; variable, para imprimir un mensaje que implique convocar una variable.
STOP, para fin de programa.
END, para fin de programa
GO TO número de sentencia, para el salto incondicional
Variable + variable, para sumar variables.
/, indica la división

Del BASIC existen numerosas versiones entre ellas una escrita por el mismo Bill Gates cuando todavía no era el hombre más rico del mundo…

5.- **Ciclo de vida del software**(Fuente "Introducción a la Informática de Albarracin-Alcalde Lancharro y García López-McGraw Hill-Santiago de Chile-1996-página 202 a 204)

Desde el planteamiento de un problema o tarea hasta que se tiene el correspondiente programa o aplicación informática para su realización por medio de una computadora, instalado en la misma y en funcionamiento mientras sea reutilidad, se sigue una serie de etapas que, en conjunto, denominamos ciclo de vida del software. Cada una de las etapas tiene un objetivo bien determinado, ha de llevarse a cabo cuando se ha terminado completamente la anterior, es decir, se han de abordar en forma estrictamente secuencial.

Las etapas de que consta el ciclo de vida del software pueden agruparse en dos bloques según el esquema de la Figura 3 para el modo operativo compilador y la figura 4 para el modo operativo intérprete..

Etapas de Análisis y diseño de programas

Es de destacar que la realización de estas etapas no necesita el uso de la computadora. No obstante, desde hace algún tiempo se utiliza ésta como apoyo mediante las denominadas herramientas CASE (Computer Arded Software Engineering)

- Análisis: Consiste en el estudio detallado del problema con el fin de obtener una serie de especificaciones en los que quede totalmente definido el proceso de la automatización.
- Programación: Consiste en la realización de una solución o ***algoritmo*** del problema planteado. Esta solución se diseña utilzando una notación intermedia(pseudocódigo) o bien mediante alguna de las notaciones gráficas como los flujogramas, sin tener

necesariamente el lenguaje de programación(aunque es conveniente tenerlo en cuenta) de la siguiente etapa.

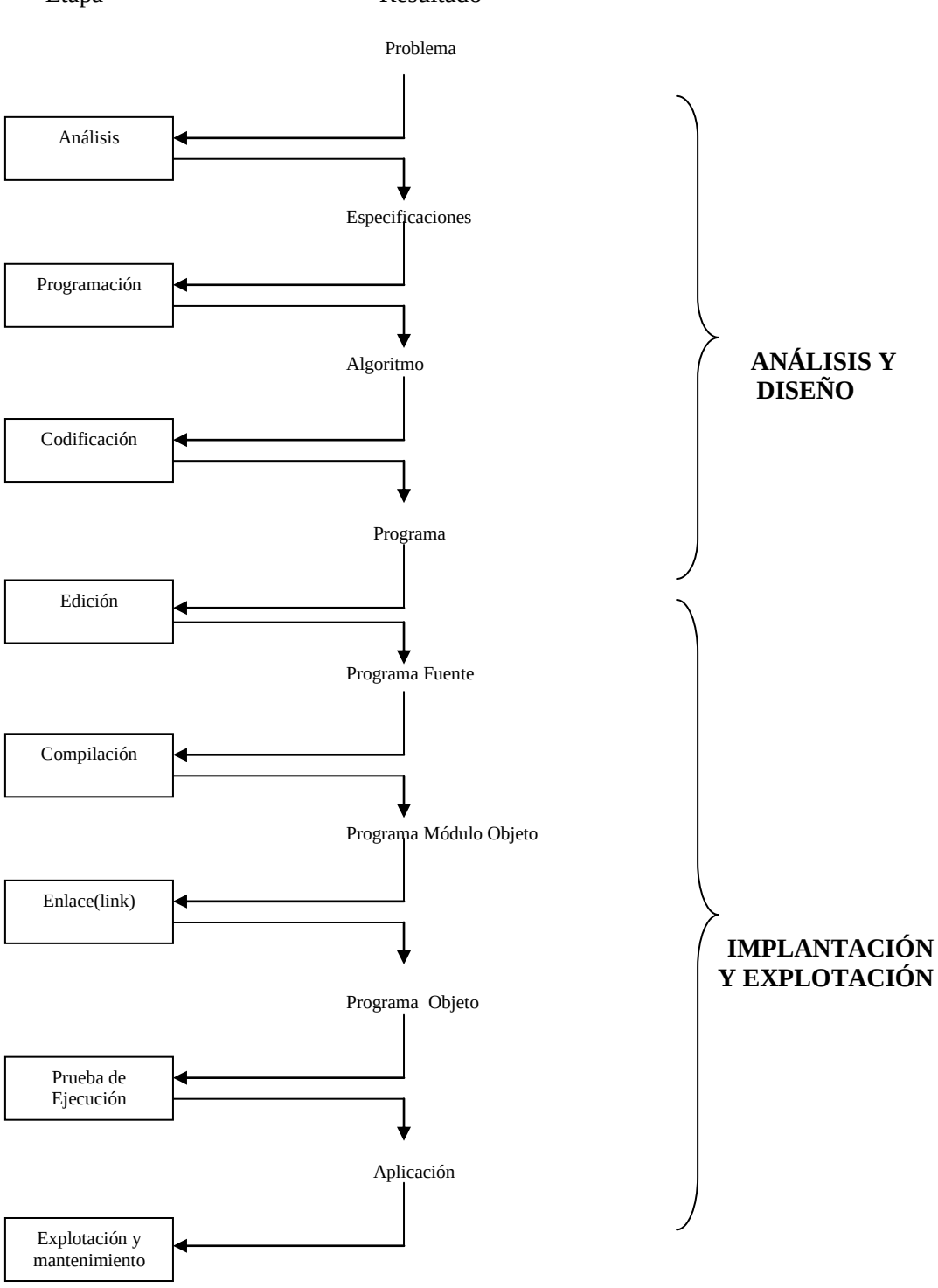

Figura 3: Ciclo de vida del software en modo operativo compilación

En esta etapa es donde tiene cabida fundamentalmente la tarea del programador y la utilización de técnicas como la programación estructurada o de la programación orientada a objetos.

- Codificación: escritura en un lenguaje de programación de alto nivel de los algoritmos obtenidos en la etapa anterior. Esta etapa puede hacerse en planillas o directamente refundirla con la etapa siguiente de…
- Edición: En esta fase se escribe el programa a la computadora, grabándose el mismo en la memoria auxiliar, por medio de un procesador de textos. Se obtiene el Programa fuente.
- Compilación: Se obtiene el programa módulo objeto reubicable y el listado de errores. De existir errores se vuelve sobre la edición, sino…
- Enlace(linkedición) En esta fase se incluyen determinadas rutinas internas de las librerías del lenguaje que sean necesarias para el programa y si la aplicación consta de varios programas o módulos se enlaza entre ellos, obteniéndose lo que denominamos programa objeto.
- Prueba de ejecución. El programa ejecutable obtenido en la etapa anterior se somete a un juego de datos de prueba capaz de detectar las posibles incorrecciones en su funcionamiento. Se pueden encontrar errores de lógica que hacen re-enfocar el problema.
- Explotación y mantenimiento. Una vez comprobada la corrección del programa y realizada su instalación en un sistema informático, la aplicación queda a disposición del usuario, que la utilizará hasta tanto se decida abandonarla o cambiarla por otra. Es lo que denominamos explotación de aplicación. Paralelamente al uso de la aplicación se realiza el mantenimiento de la misma, que consiste en su evaluación periódica por parte del personal informático, así como la inclusión de las adaptaciones y modificaciones necesarias para mantenerla actualizada.

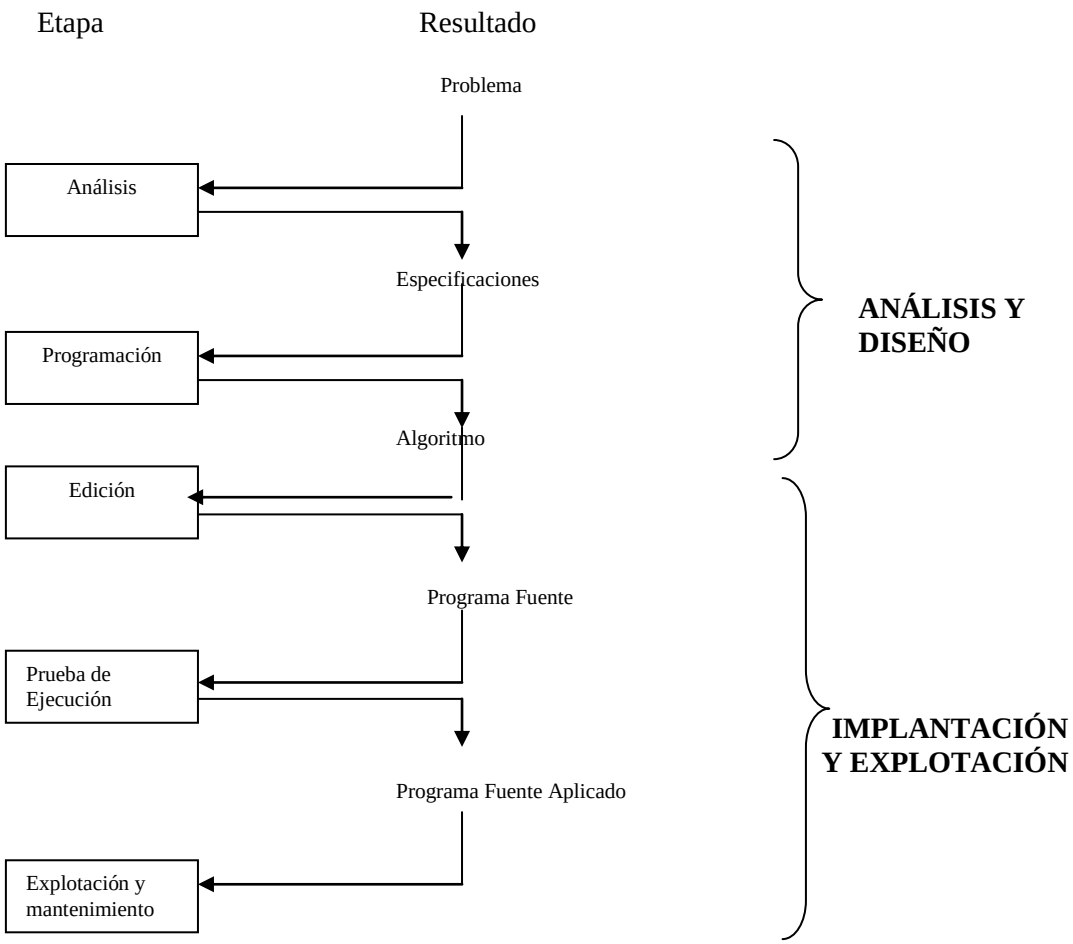

Figura 4: Ciclo de vida del software en modo operativo EDICIÓN

Nótese que en el modo Compilador se está entregando al usuario un Programa Objeto que no se puede modificar, le queda al programador el Fuente para futuras ampliaciones, esto es lo que comercialmente hay que hacer, a menos que se trabaje en relación de dependencia y el programador renuncie expresamente a sus derechos de autor.

En el modo Intérprete se termina entregando el Programa Fuente que puede ser modificado por otro programador, comercialmente esto no es conveniente.

6.- **Una Babel de Lenguajes**

Con el correr de los años se fueron creando nuevos lenguajes para nuevas aplicaciones, haciendo un rápido listado, se tiene:

- ALGOL: **ALGO**rithmic **L**anguage fue creado por J.Backus y P.Naur entre 1958 y 1960. Se lo considera el primer lenguaje de propósito general para aplicaciones industriales y científicas. La última versión fue Algol68.
- SIMULA: El Simula I fue el primer lenguaje orientado a objetos, estructurado mediante objetos y clase(ver punto 7). Fue creado por Ole Dahl y Kristen Nygaard del Centro de Computación de Noruega(NCC, Oslo) en 1962. El lenguaje fue implementado por primera vez en 1964, diseñado como una extensión a Algol 60, haciendo posible simular eventos discretos. En 1967 se introdujo el lenguaje más general Simula 67 con un número mayor de tipos de datos, además de apoyo a objetos. Simula se estandarizó en 1977.
- PL/I: Programming Language I, fue un lenguaje bastante complejo diseñado en IBM a partir de 1962 para el sistema \360. PL/I quería ser el programa principal de desarrollo para los sistemas y aplicaciones de gran tamaño. Fue utilizado principalmente en la década de 1980, aún se utiliza.
- APL: A Programming Language, fue diseñado por Ken Iverson a partir de 1962 y utilizado por IBM. El objetivo principal de este lenguaje era el apoyo a la programación matemática y estadística. La versión actual es APL 2000.
- Pascal: Este lenguaje fue inventado por Niklaus Wirth del Instituto Tecnológico Federal de Zurich entre 1968 y 1971. Pascal evolucionó el diseño de Algol, por lo que muchos años fue el lenguaje más utilizado como introducción a la programación en muchas universidad. Entre sus dialectos más modernos se incluye TurboPascal y ObjectPascal como parte de desarrollo Delphi de Borland.
- Smalltalk: Creado por Alan Kay en Xerox Park, también diseñador de la primera computadora personal basada en programación orientada a objetos, FLEX de 1967 a 1968. La primera versión fue conocida como Smalltalk 72, cuyas raices fueron Simula 67, siguiendo Smaltalk 76, versión totalmente orientada a objetos. En 1980, Smalltalk 80, fue la primera versión comercial del lenguaje, incluyendo un ambiente de programación orientada a objetos. El lenguaje Smalltalk ha influido sobre otros muchos lenguajes como C++ y Java. Smalltalk es aún un lenguaje muy utilizado.
- PROLOG: **PRO**gramming in **LOG**ic fue el origen de la programación lógica. Lo diseño Robert Kowalsi de la Universidad de Edimburgo, Reino Unido, junto con Alain Colmerauer, de la Universidad de Aix-Marsella, Francia; en 1972.-
- C: El Lenguaje C fue diseñado por Ritchie y Thompson entre 1969 y 1973. Este lenguaje fue diseñado en forma paralela a los primeros desarrollos del sistema operativo UNIX. Una segunda etapa de desarrollo fue hecha por Kermighan y Ritchie entre 1977 1979, como adaptación a la extensión del UNIX a múltiples plataformas. Es uno de los lenguajes más utilizados en la actualidad.
- CLU: CLUster, es un lenguaje diseñado por Bárbara Liskov del MIT entre los años 1974 y 1977. El lenguaje utilñiza conceptos básicos de orientación a objetos aunque no es propiamente considerado como tal.
- MODULA: La versión original de este lenguaje se llamó Modula-2 y fue desarrollado por Niklaus Wirth a mediados de la década de 1970 como descendiente directo de Pascal. Este lenguaje incluía ciertos aspectos de la orientación a objetos. La última versión conocida

como Modula-3 fue diseñada por Luca Cardelli. Dada su simpleza se desconoce por qué no ha tenido mayor éxito en su utilización.

- ADA: El lenguaje fue diseñado principalmente por Jean Ichibah del Departamento de Defensa de Estados Unidos en 1977, para apoyar la programación a gran escala y promover la robustez del software. Se denominó Ada en honor de Lady Ada Lovelance(1815-1852), considerada la primera programadora de la historia, amiga y confidente de Charles Babbage, considerado a su vez como el padre de la computación por su trabajo teórico hace un siglo y medio. Aunque la versión original de este lenguaje no era orientada a objetos, la última versión, Ada 95, sí lo es. Las versiones anteriores, incluyendo Ada 83, no se consideran como orientadas a objetos.
- OBJETIVE-C: Este lenguaje fue diseñado por Brad Cox como una extensiónn a C pero con orientación a objetos. Contenía muchos aspectos inspirados en Smalltalk 80. Se hizo popular debido a su utilización en la computadora NeXT, incluyendo una interface bajo el ambiente NeXTSTEP, conocida posteriormente como OpenStep , y actualmente adquirido por Apple como base para su nuevo sistema operativo MacOS X.
- BETA: Fue desarrollado por Ole Lehmann Madsen en la Universidad de Aarhus en Dinamarca. Es un lenguaje orientado a objetos inspirado en el Simula con sintaxis similar a Pascal y a C.
- ML: **M**eta **L**anguage representa a una familia de lenguajes funcionales propuesta originalmente en 1983 y diseñada entre 1984 y 1988 por Milner y Tofte. Standard ML 97 es la versión actual pero la misma es modesta y simplificada, al punto tal que no tiene las rutinas necesarias para poder trabajar en orientación a objetos.
- C++: fue diseñado por Bjarne Stoustrup de AT&T Bell Labs, entre 1982 y 1985. C++ agrega mecanismos de orientación a objetos al lenguaje C, lo que lo hace un lenguaje híbrido pero poderoso.
- Eiffel: Se llama así en honor a la famosa torre de París. Lo diseñó Bertrand Meyer como un lenguaje orientado a objetos con una sintaxis similar a C. Aunque es un lenguaje muy sencillo y poderoso nunca logró la aceptación lograda por C++ y Java. Creado en 1986..
- SELF: El lenguaje Self fue diseñado por David Ungar y Randall Smith en Sun Microsystems. Es unlenguaje cuya sintaxis es similar a Smalltalk. Creado en 1986 permite el trabajo en orientación a objetos.
- DYLAN: DYnamic LANguage es un lenguaje orientado a objetos, originalmente desarrollado por Apple. Se parece a CLUS y Écheme aunque tiene influencia de Smalltalk y Self. Fue creado n 1992.
- JAVA: Fue diseñado por Gosling de Sun Microsystems entre 1994 y 1995, es el lenguaje orientado a objetos más utilizado en la actualidad. Es sencillo y portátil, bastante similar al C++ aunque tomando ideas de Modula.3, Smalltalk y Objetive C.
- C#: Este lenguaje conocido como Sharp, fue diseñado por Microsoft. Es una extensión a C con orientación a objetos inspirado en C++ y en particular en Java. El lenguaje evita muchos de los problemas de diseño de C++.

7.- **Programación estructurada y orientación a objetos**

El comienzo de la programación fue concebir programas que eran una secuencia ordenada lógicamente de instrucciones y datos. Los progrmas que se han escrito en Assembly es un buen ejemplo de programación estructurada.

Una de las características básicas es poder llamar los datos desde cualquier ubicación de una aplicación (ver figura 5).

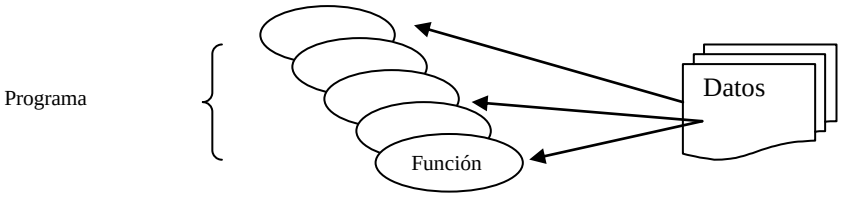

Figura 5: Programación estructurada

En la función se define el formato de los datos, por ejemplo, fecha: dia, mes y año, cada una con 2 dígitos. Debido a que los datos originalmente se escribían en las tarjetas de ochenta columnas, para definir un dato eran pocos caracteres, por ese motivo es que se acotaban La fechas como ser el ingreso a la empresa, la fecha de nacimiento, la fecha de casamiento, a seis dígitos.-

Mientras se estuvo en el mismo siglo no había ningún problema, por ejemplo, ¿qué edad tiene una persona nacida en 1971, en el año 1995?, la respuesta se obtenía de restar 95 menos 71, cuyo resultado era 24.

El problema se origina en el año 2000, llamado el problema del Y2K, por ejmplo en el año 2001 se quiere volver a calcular los años de la persona nacida en 1971:

2001 queda ⟶ 01
1971 queda ⟶ 71
Edad..... -70 (parece de 30!!!)

Para arreglar este problema hay que modificar todas las funciones.

La programación orientada a objetos parte de englobar a los datos y funciones en un mismo objeto(ver figura 6) Un cambio en la estructura de datos sólo afecta a las funciones de un objeto pero no al resto de la aplicación.

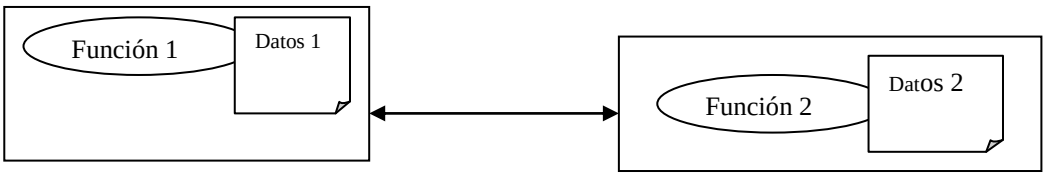

Figura 6: Programación orientada a objetos

Los lenguajes de alto nivel que trabajan con los conceptos de Orientación a Objetos son:

- SIMULA
- SMALLTALK
- MODULA
- ADA
- OBJETIVE-C
- BETA
- C++
- SELF
- DYLAN
- JAVA
- C#