– Typeset by GMNI & FoilTEX **–**

Lenguajes de programación en ingeniería:
EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

F. Navarrina, J. París & GMNI

GMNI — GRUPO DE MÉTODOS NUMÉRICOS EN INGENIERÍA

Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos
Universidad de A Coruña, España

e-mail: {fermin.navarrina,jose.paris}@udc.es
página web: http://caminos.udc.es/gmni

ÍNDICE

PRINCIPALES PARADIGMAS (en Ingeniería):

▶ Lenguaje Máquina:	Hardware	($<$ 1940 aprox.)
▶ Ensamblador:	Software	(1940–1950 aprox.)
▶ FORTRAN/II/IV:	Compiladores	(1950–1960 aprox.)
▶ BASIC:	Intérpretes	(1960–1970 aprox.)
▶ PASCAL:	Prog. Estructurada	(1970–1980 aprox.)
▶ FORTRAN 77:	(influencia del Pascal)	(1977)
▶ Lenguaje C:	Portabilidad, S.O. UNIX	(1980–1990 aprox.)
▶ Lenguaje C++:	Prog. orientada a objeto	(1990–2000 aprox.)
▶ Fortran 90/95/2003:	(influencia del C)	(1990,1995,2003)
▶ Java:	Internet	($>$ 2000 aprox.)

Lenguaje Máquina (I)

LENGUAJE MÁQUINA

- Secuencia de bits (agrupados en bytes) que corresponden a las instrucciones que debe ejecutar el procesador y a los datos fijos del programa (esencialmente constantes numéricas y/o alfanuméricas) .
- La codificación directa de un programa en lenguaje máquina es
  - ♠ muy costosa,
  - ♠ muy farragosa, y
  - ♣ ABSOLUTAMENTE DEPENDIENTE DEL **HARDWARE**.
- Ejemplos: (véase la carpeta EjemplosDeArchivosDeTextoYBinarios)
  - El archivo hello.exe es un archivo binario que contiene un programa ejecutable.
  - El archivo hello.exe.txt muestra los bytes que forman el archivo hello.exe,
  - y el archivo hello.exe.b.txt muestra los correspondientes bits.
- En la actualidad sólo se utiliza en dispositivos muy sencillos.

Ensamblador (I)

ENSAMBLADOR

- Lenguaje de **MUY BAJO NIVEL**. (*)
- El programador dispone de un repertorio de instrucciones elementales sencillas, relativamente fáciles de recordar. Por ejemplo:

STO N	$\longrightarrow$ Guardar (“‘store”) en la posición de memoria N.
RCL N	$\longrightarrow$ Leer (“recall”) el contenido de la posición de memoria N.
+,-,*,/	$\longrightarrow$ Sumar, restar, multiplicar, dividir.
=	$\longrightarrow$ Calcular el resultado de las operaciones anteriores.
JMP #L	$\longrightarrow$ Si el resultado anterior es 0, ir (“jump”) a la línea L.
STP	$\longrightarrow$ Parar el programa (“stop”).

- Un programa (**ASSEMBLER**) se encarga de **ENSAMBLAR** (**) el código y construir el correspondiente programa ejecutable

---

(*) Muy próximo al lenguaje máquina.

(**) Traducir el programa a lenguaje máquina y añadir lo que sea necesario para que pueda ejecutarse.

Ensamblador (II)

- La programación en **ENSAMBLADOR** sigue siendo
  - ♠ muy costosa,
  - ♠ muy farragosa, y
  - ♣ ABSOLUTAMENTE DEPENDIENTE DEL **HARDWARE**.
- En la actualidad sólo se utiliza cuando se necesita sacar el máximo partido al hardware.

Ensamblador (III)

- Ejemplo: Programa para calcular factoriales realizado en un hipotético **ENSAMBLADOR**

```
#01: STO 01
#02: 1
#03: STO 02
#04: RCL 01
#05: JMP #16
#06: *
#07: RCL 02
#08: =
#09: STO 02
#10: RCL 01
#11: -
#12: 1
#13: =
#14: STO 01
#15: GTO #04
#16: RCL 02
#17: STP
```

FORTRAN (I)

FORTRAN / II / IV

- Lenguaje de **ALTO NIVEL**. (*)
- Diseñado por John Backus para IBM en 1953, como alternativa al Lenguaje Máquina de un ordenador IBM 704.
- El primer compilador fue distribuido en 1957.

John Backus (1924–2007).
Creador del FORTRAN (1953) en IBM.
(Fuente: IBM)

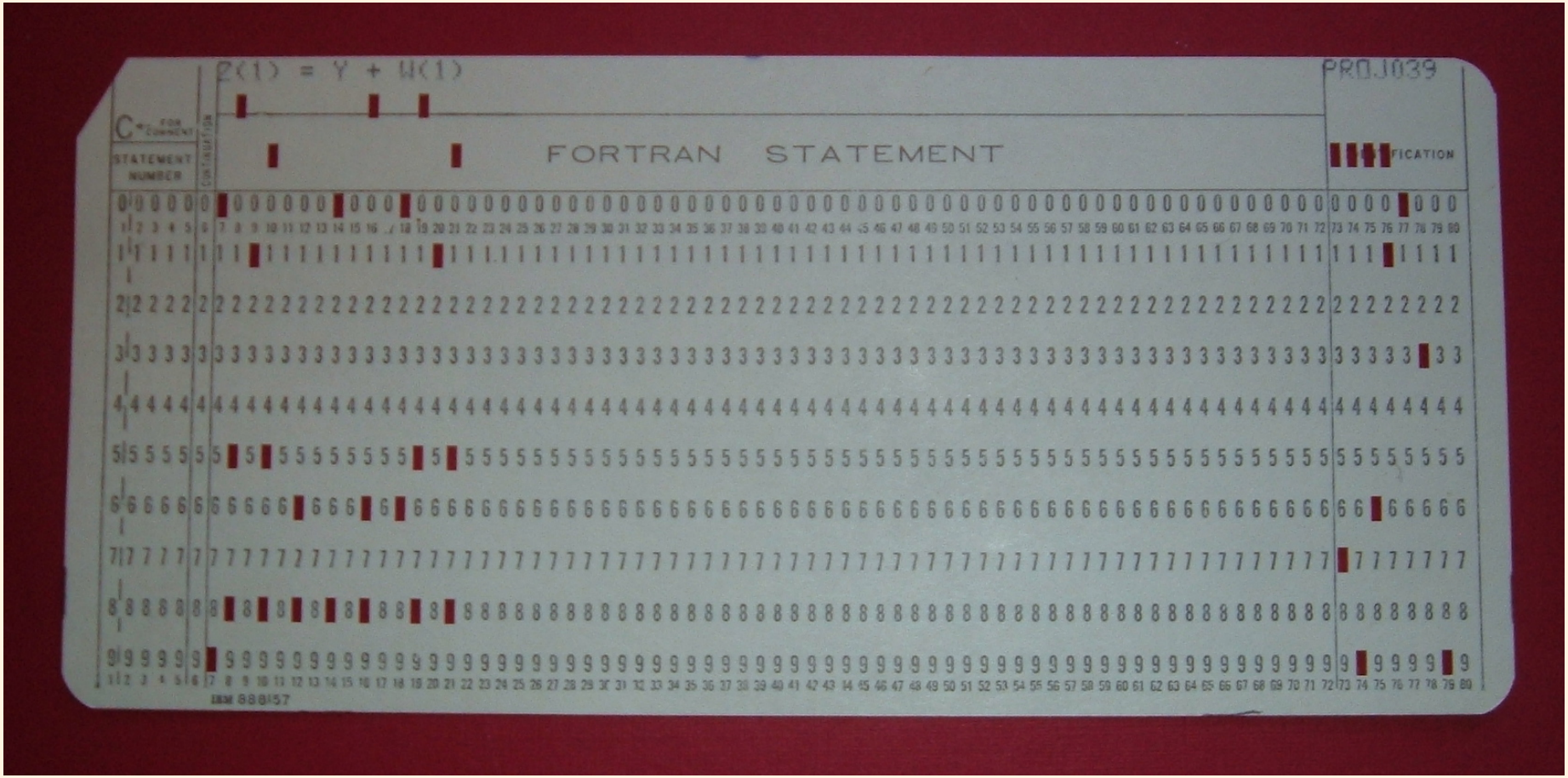
(*) Alejado del lenguaje máquina.

FORTRAN (II)

- **FORTRAN** proviene de **FORmula TRANslator** (“traductor de fórmulas”).
- La codificación de las fórmulas es sencilla.
- La sintaxis viene condicionada por el soporte (tarjetas perforadas).

En efecto . . .

- Origen: **Tarjetas Perforadas** (”punch cards”)



Tarjeta Perforada. (Fuente: <http://commons.wikimedia.org/wiki/Image:FortranCardPROJ039.agr.jpg>)

- Origen: **Perforadora de Tarjetas**

Perforadora de Tarjetas. (Fuente: <http://www.chilton-computing.org.uk/acl/technology/atlas/p013.htm>)

- Origen: **Programa FORTRAN en tarjetas perforadas**

Programa FORTRAN en tarjetas perforadas. (Fuente: <http://www.staff.ncl.ac.uk/roger.broughton/museum/iomedia/pc.htm>)

FORTRAN (IIId)

- Origen: **Lectora de Tarjetas**

Programa FORTRAN en tarjetas perforadas. (Fuente: <http://www.staff.ncl.ac.uk/roger.broughton/museum/iomedia/pc.htm>)

FORTRAN (IV)

- La gestión de memoria es rígida pero predecible. (*)
- Las instrucciones de control son muy primitivas. Básicamente...
  - **GOTO** incondicional
  - **IF aritmético**
  - **DO – CONTINUE**
  - **CALL**
- Modelo de implementación: **PROGRAMA PRINCIPAL + SUBRUTINAS**
  - **SUBRUTINA** = **subprograma**
    - **CALL** transfiere el control a un subprograma
    - que forma parte del **programa ejecutable** o
    - cuya localización conoce el **programa ejecutable** (ejemplo: librerías del sistema)
  - La transfencia de argumentos se realiza **POR REFERENCIA**.

(*) En principio (siempre que el programa esté bien hecho) cuando comienza la ejecución de un programa debe haber memoria suficiente para la realización de todos los cálculos, por lo que todo debería funcionar correctamente (sin detenciones por falta de recursos, "cuelgues" del sistema, etc.)

FORTRAN (V)

- Ejemplo: Programa **FORTRAN IV** para calcular factoriales

```
C     PROGRAMA PARA CALCULAR FACTORIALES
      READ(5,100) N
  100 FORMAT(I5)
      NFAC=1
  500 CONTINUE
      IF (N) 1000, 1000, 600
  600 CONTINUE
      NFAC=NFAC*N
      N=N-1
      GOTO 500
 1000 WRITE(6,110) NFAC
  110 FORMAT(I10)
      STOP
      END
```

LENGUAJES COMPILADOS

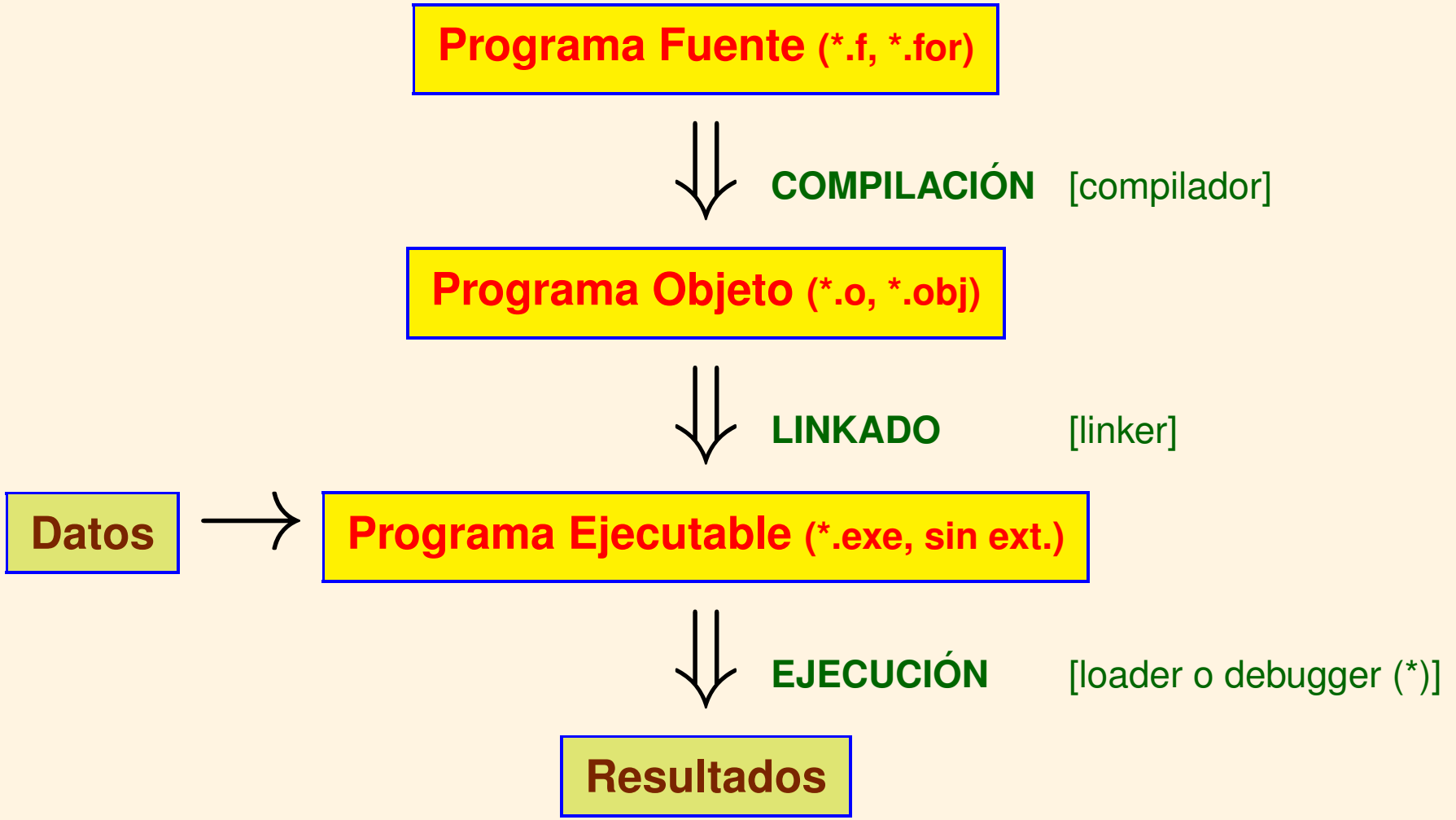
▶ FORTRAN ES EL PROTOTIPO DE LENGUAJE COMPILADO

- El programador escribe el **programa fuente** (uno o varios archivos *.f, *.for).
- Un programa (**compilador FORTRAN**) se encarga de **COMPILAR** (*)
  el **programa fuente** (uno o varios archivos *.f, *.for) y
  crear los correspondientes **programas objeto** (archivos *.o, *.obj)
- Otro programa (**linker**) se encarga de **LINKAR** (**)
  el/los **programa/s objeto** (archivos *.o, *.obj) y
  crear el correspondiente **programa ejecutable** (archivo *.exe o sin extensión).
- Una vez creado, el **programa ejecutable** (archivo *.exe o sin extensión)
  puede utilizarse cuantas veces sea necesario.

(*) Traducirlo/s a lenguaje máquina.

(**) Dimensionar la memoria necesaria y
unir (“link”) todas las partes.

Lenguajes Compilados (II)



(*) Debugger (depurador): loader que permite ejecutar paso a paso un programa y que tiene herramientas que permiten localizar dónde se producen los errores.

Lenguajes Compilados (III)

- **ERRORES:**
  - ♣ de **Planteamiento**

    Errores conceptuales o de diseño que impiden que el programa funcione correctamente.
  - ♣ de **Compilación**

    Errores gramaticales que impiden **compilar** el **programa fuente** y generar el correspondiente **programa objeto**.
  - ♣ de **Linkado**

    Errores estructurales que impiden **linkar** el **programa objeto** y generar el correspondiente **programa ejecutable**.
  - ♣ de **Ejecución** (“run–time”) (*)

    Errores de tipo general que impiden **ejecutar** el **programa ejecutable** y obtener los resultados.

(*) Bug: error de ejecución difícil de encontrar.

Errores de Planteamiento

Errores conceptuales o de diseño que impiden que el programa funcione correctamente:

- uso de algoritmos inadecuados o incorrectos,
- errores en los datos.

Errores de Compilación

Errores gramaticales que impiden **compilar** el **programa fuente** y generar el correspondiente **programa objeto**:

- errores ortográficos (instrucciones mal escritas) o
- errores sintácticos (proposiciones mal construidas).

Errores de Linkado

Errores estructurales que impiden **linkar** el **programa objeto** y generar el correspondiente **programa ejecutable**:

- dimensionamiento incorrecto de la memoria o
- falta de alguna de las subrutinas/funciones.

Errores de Ejecución (“run–time”)

Errores de tipo general que impiden **ejecutar** el **programa ejecutable** y obtener los resultados:

- errores ocultos que no fueron detectados ni al **compilar** ni al **linkar**,
- incapacidad del ordenador para ejecutar el programa
  - el programa requiere más memoria de la que hay disponible o
  - falta alguna de las subrutinas/funciones del sistema,
- errores ocultos de cualquiera de los tipos anteriores que impiden la ejecución del programa,
  - conducen a una operación prohibida
    (ej: división por 0, raíz cuadrada de un número negativo) o
  - se producen errores en la lectura de datos.

BASIC (I)

BASIC

- ♡ Más ordenadores, más rápidos, más pequeños, más baratos. . .
- ♡ Interacción más cómoda:
  MONITORES CRT frente a **tarjetas perforadas**.
- ♣ La programación se pone al alcance de personas sin experiencia con poca formación técnica y sin mentalidad analítica. (*)
- ♣ Algunas opiniones de nuevo cuño sobre el **FORTRAN**
  - ♠ Es muy **¡difícil!** programar en este lenguaje.
  - ♠ Es muy **¡fastidioso!** tener que **compilar** y **linkar** antes de ejecutar.
  - ♣ El estilo del lenguaje se ha quedado **anticuado**.
  - ♣ La gestión de memoria es demasiado **rígida**.

(*) Carecer de estas capacidades no tiene por qué ser malo (en general),
pero dificulta la realización de algunas actividades
(como calcular integrales, diseñar sistemas que funcionen o programar ordenadores).

BASIC (II)

- **BASIC** proviene de . . . **¿a ver quién lo adivina?** (*)
- La codificación de las fórmulas es tan sencilla como en **FORTRAN**.
- La sintaxis es muy parecida a la del **FORTRAN**
  con algunas novedades que se deben a la nueva forma de trabajar. (**)

(*) BASIC $\Longrightarrow$ Beginner's All-purpose Symbolic Instruction Code.

(**) Cada línea tiene ahora un número.
Los números de línea establecen el orden en el que deben interpretarse las instrucciones. Se pueden intercalar una nueva línea en cualquier parte del programa: basta con escribir la llínea mediante el teclado, comenzando con un número intermedio entre los de otras dos líneas correlativas.

BASIC (III)

- La gestión de memoria es flexible, pero impredecible. (*)
- Las instrucciones de control son similares a las del **FORTRAN**. Básicamente. . .
  - **GOTO** incondicional
  - **IF – THEN**
  - **FOR – NEXT**
  - **GOSUB**
- Modelo de implementación: **PROGRAMA PRINCIPAL**

  SUBRUTINA = **parte del programa principal**
  - **GOSUB** equivale a un **GOTO** con retorno.

(*) La memoria se dimensiona sobre la marcha. Por tanto, se pueden producir detenciones por falta de recursos, “cuelgues” del sistema, etc.

BASIC (IV)

- Ejemplo: Programa **BASIC** para calcular factoriales

```
10 REM PROGRAMA PARA CALCULAR FACTORIALES
20 READ N
30 NFAC=1
40 IF (N<=0) THEN GOTO 80
50 FOR I = 1 TO N
60 NFAC=NFAC*I
70 NEXT I
80 PRINT NFAC
90 END
```

Lenguajes interpretados (I)

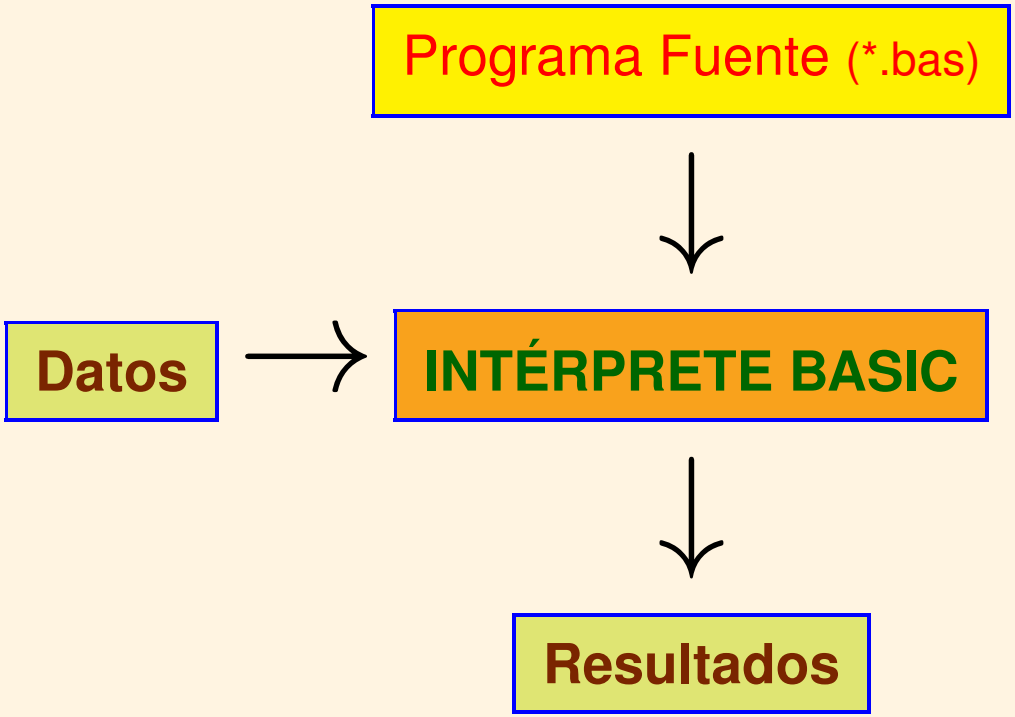
LENGUAJES INTERPRETADOS

▶ BASIC ES EL PROTOTIPO DE LENGUAJE INTERPRETADO

- El programador escribe el **programa fuente** (un archivo *.bas).
- Un programa (**intérprete BASIC**) se encarga de **INTERPRETAR** (*) el **programa fuente** (un archivo *.bas).
- El programa se ejecuta a medida que se interpreta, y se interpreta a medida que se ejecuta.
- No hay **compilación** ni **linkado**, por lo que
  - ▷ un programa puede ejecutarse aunque tenga errores de sintaxis hasta que el procesador no entienda la instrucción y el programa se pare, y
  - ▷ la memoria se va dimensionando sobre la marcha mientras no se agote y el programa se pare.

(*) Se traduce cada instrucción a lenguaje máquina y se ejecuta a continuación.

Lenguajes interpretados (II)



Lenguajes interpretados (III)

- El lenguaje **BASIC**
  - ♡ es fácil de aprender,
  - ♣ tiene un **intérprete** “amigable”,
  - ♣ fue moderno y
  - ♣ es flexible.
- Pero. . .
  - ♠ cada vez que se ejecuta una instrucción es necesario traducirla, (por lo que los programas son mucho más lentos), y
  - ♠ al arrancar un programa no se sabe
    - ♣ si el programa está correctamente escrito, ni
    - ♣ si habrá memoria suficiente para ejecutarlo.

PROGRAMACIÓN ESTRUCTURADA

- ♡ programas estructurados en bloques autocontenidos, con un flujo fácil de entender (con pocos GOTO's).
- ♡ Preocupación por la estandarización (ANSI).
- ♣ Preocupación por la declaración de variables.
- ♣ El **PASCAL** compendia las nuevas ideas.

FORTRAN 77 (I)

FORTRAN 77

- El **FORTRAN** se acomoda a las nuevas tendencias...
  - el lenguaje existe como standard ANSI a partir de 1966,
  - se evita el uso del **GOTO** incondicional,
  - se cambia profundamente el estilo de programación para que la lógica de los programas sea más evidente, y
  - se introducen muchas extensiones (no standard) en los compiladores (*)
- **FORTRAN 77** incorpora nuevas instrucciones de control más sofisticadas como...
  - **DO – ENDDO** (**)
  - **DO WHILE – ENDDO**
  - **IF – THEN– ELSE – ENDIF**

(*) Finalmente muchas de estas extensiones han acabado formando parte del standard.

(**) ENDDO fue una extensión (no standard) que acabó formando parte del standard.

FORTRAN 77 (II)

- Ejemplo: Programa **FORTRAN 77** para calcular factoriales

```
C     Programa para calcular factoriales

      read(5,*) n

      nfac=1

      do while (n.gt.0)
        nfac=nfac*n
        n=n-1
      enddo

      write(6,*) nfac

      stop
      end
```

Lenguaje C

- Lenguaje de **BAJO NIVEL**. (*)
- Diseñado por Dennis Ritchie para Bell Telephone Laboratories entre 1969 y 1973, a partir de trabajos previos de Ken Thompson como complemento del **Unix**.
- En 1978 se publicó el libro *The C Programming Language* de Brian Kernighan y Dennis Ritchie. (**)

Ken Thompson y Dennis Ritchie, Bell Telephone Laboratories (1972).
Creadores del **Lenguaje C** y del sistema operativo **Unix**.
(Fuente: <http://www.encyclopaedia.es/es/wiki/Unix.html>)

(*) Próximo al lenguaje máquina.

(**) Kernighan no participó en el desarrollo del **Lenguaje C**, aunque sí en el del **Unix**. ¿Cómo se las arregló para figurar como primer autor del libro?

Lenguaje C (II)

- El nombre del **Lenguaje C** se debe a que Dennis Ritchie lo desarrolló a partir del **Lenguaje B**, que había sido desarrollado previamente por Ken Thompson a partir del . . . ¿**Lenguaje A**? (*)

Dennis Ritchie y Ken Thompson.
Creadores del C (1972) y del sistema operativo **Unix** en Bell Telephone Laboratories.
(Fuente: <http://penguin.dcs.bbk.ac.uk/academic/unix/linux/slides/>)

(*) ¡Pues no! Pero fue algo parecido.

Lenguaje C (III)

- ♠ Antes del desarrollo del sistema operativo **Unix**, cada fabricante desarrollaba un sistema operativo propio para **¡cada modelo de ordenador!**
- ♢ **Unix** fue desarrollado con la intención de disponer de un sistema operativo universal **capaz de funcionar en cualquier ordenador**.
  - ♣ La primera versión de Unix se programó en **ENSAMBLADOR**.
  - ♡ La segunda versión de Unix se programó en **Lenguaje C**

    . . . **¡Y FUE UN ÉXITO!**
- ♡ Ya que **Unix** y **C** deben funcionar en máquinas pequeñas, se realiza un diseño minimalista $\Rightarrow$ máxima eficacia, minimos requisitos.
- ♡ Ya que **Unix** y **C** deben tener carácter universal, se abandona toda dependencia respecto a un tipo particular de hardware.
  - ♣ Se abandona el concepto de una instrucción por línea.

Lenguaje C (IV)

- La gestión de memoria es completamente abierta: **PUNTEROS**. (*)
- Las instrucciones de control son muy sofisticadas. Principalmente...
  - **if** () {} **else** {}
  - **for** (; ; ;) {}, **while** () {}, **do** {} **while** (); **break;** y **continue;**
  - **switch** () {**case C: ; default: ;** } ...y también el indestructible
  - **goto** incondicional, (porque a veces sigue siendo lo mejor).
- Modelo de implementación: **PROGRAMA PRINCIPAL + FUNCIONES**
  - **FUNCIÓN** = **subprograma**
    - que se incorpora al **programa ejecutable** o
    - cuya localización conoce el **programa ejecutable** (ejemplo: librerías del sistema)
  - La transfencia de argumentos se realiza **POR VALOR** (por defecto) o **POR REFERENCIA** (si se indica).

(*) El lenguaje permite manipular la memoria con total libertad:
es posible dimensionar una parte de la memoria en el momento de la compilación;
y también es posible dimensionar memoria adicional durante la ejecución (lo que puede provocar problemas).

Lenguaje C (V)

- Es un lenguaje (nuevamente) de **BAJO NIVEL**: (*)
  - formado por instrucciones elementales,
  - que operan sobre entidades sencillas
    (no hay tratamiento de strings, ni operador exponenciación, etc.).
- No contempla instrucciones I/O
  (scanf() y printf() son funciones).
- Compacidad de código (un tanto críptico).
- Mayúsculas y minúsculas son letras distintas.
- Es preciso declarar todas las variables.
- Nuevos tipos de variables (a medida): **struct**, **union**.
- Se activa un PREPROCESADOR antes de la compilación.

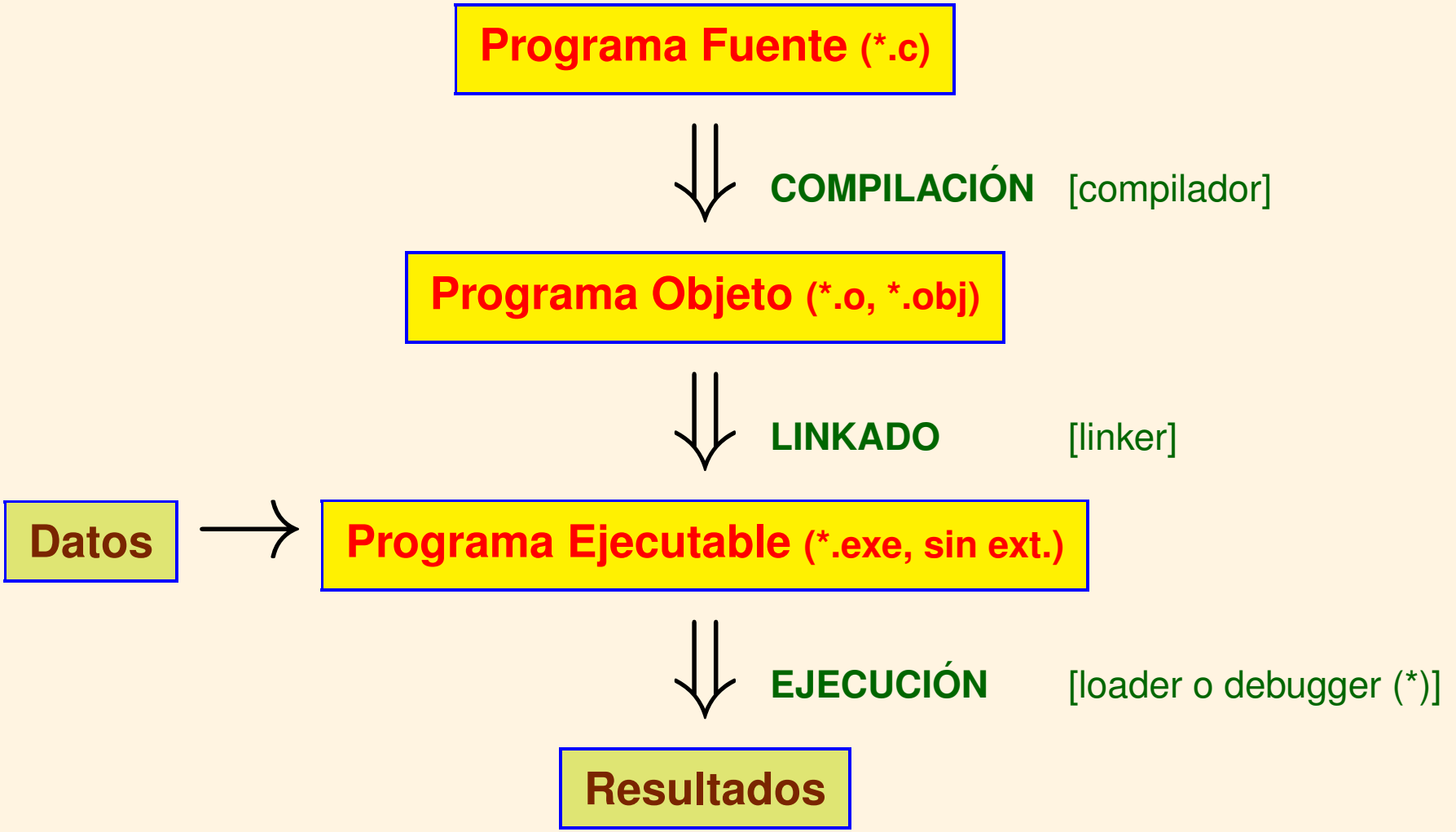
(*) Próximo al lenguaje máquina.

Lenguaje C (VI)

- **C ES UN LENGUAJE COMPILADO**
  - Mismo esquema **compilación–linkado–ejecución** que el **FORTRAN**.
  - Mismo tipo de errores que el **FORTRAN**.
- En comparación con el **FORTRAN**
  - El **Lenguaje C** es —claramente— más adecuado que el **FORTRAN** para programación de sistemas y de aplicaciones informáticas.
  - El **Lenguaje C** es más adecuado que el **FORTRAN** para interactuar con aplicaciones informáticas (AutoCAD, bases de datos, etc.).
  - El **Lenguaje C** es normalmente más adecuado que el **FORTRAN** para realizar programas de propósito general.
  - El **FORTRAN** compite favorablemente con el **Lenguaje C** cuando se trata de realizar programas de cálculo científico en Ingeniería, porque
    - es igual de eficaz,
    - y mucho más sencillo conceptualmente (*).

(*) Digamos que para realizar un programa **FORTRAN** no es imprescindible entender desde un punto de vista puramente informático qué es lo que se está haciendo, como sucede con el **Lenguaje C**.

Lenguaje C (VII)



Lenguaje C (VIIIa)

- Ejemplo: Programa en **Lenguaje C** para calcular factoriales

```
/* Programa para Calcular Factoriales */
#include <stdio.h>
main()
{
  int n, nf;

  scanf("%d", &n);

  for (nf = 1; n > 0; nf *= n--);

  printf("%d", nf);
}
```

Lenguaje C (VIIIb)

- Ejemplo: Programa en **Lenguaje C** (compactado) para calcular factoriales

```
#include <stdio.h>
main(){
int n,nf;scanf("%d",&n);for(nf=1;n>0;nf*=n--);printf("%d",nf);
}
```

Tendencias actuales (I)

PROGRAMACIÓN ORIENTADA A OBJETOS

- Paradigma de programación que se basa en la utilización de
  - objetos y de
  - sus interacciones.
- Se basa en varias técnicas, entre las que se incluyen:
  - herencia,
  - modularidad,
  - polimorfismo, y
  - encapsulamiento.

Tendencias actuales (II)

LENGUAJE C++

- Lenguaje de **NIVEL MEDIO**. (*)
- Diseñado por Bjarne Stroustrup
  para Bell Laboratories
  hacia 1980,
  a partir del **Lenguaje C**.
- En esencia es una de las extensiones del **Lenguaje C** (**)
  (tal vez la más conocida)
  que incluye herramientas de programación orientada a objetos.

(*) Su organización combina elementos característicos de lenguajes tanto de alto como de bajo nivel.

(**) **C++** es muy habitual en entornos **Linux**.
C++ y **C#** son muy habituales en entornos **Windows**.
Objective C es muy habitual en entornos **Mac OS**.

Tendencias actuales (III)

JAVA

- En esencia es otra de las extensiones del **Lenguaje C** que incluye herramientas de programación orientada a objetos y de la que se han eliminado operaciones de bajo nivel (para evitar que se cometan errores).
- Fue desarrollada por Sun Microsystems a principios de los años 90.
- Está especialmente orientado a la ejecución de aplicaciones y a la visualización a través de Internet.

Tendencias actuales (IV)

Fortran 90/95/2003

- El **Fortran** se acomoda a las nuevas tendencias (una vez más)...
  - El lenguaje existe como standard ANSI a partir de 1992.
  - Se incorporan conceptos propios de otros lenguajes y estilos de programación:
    - Declaración **ALLOCATABLE** e instrucción **ALLOCATE**. (*)
    - Archivos fuente tipo **Free-form** (más de 80 carácteres por línea).
    - Punteros, recursividad, operaciones vectoriales, prog. orientada a objetos, etc.
  - Se procura eliminar inconsistencias internas y causas de errores frecuentes:
    - En Fortran 90 se declaran **obsoletas** algunas instrucciones, aunque se permite su uso. En Fortran 95 se prohíbe su utilización. (**)
  - El nuevo lenguaje es más potente y versátil que sus precedentes, pero también
    - menos intuitivo y más difícil de aprender, [$\Longrightarrow$ no gusta a programadores noveles]
    - menos atractivo que otros lenguajes modernos. [$\Longrightarrow$ no gusta a programadores profesionales]

(*) Es posible dimensionar memoria adicional en tiempo de ejecución.

(**) Algunos programas antiguos no pueden ser recompilados directamente.

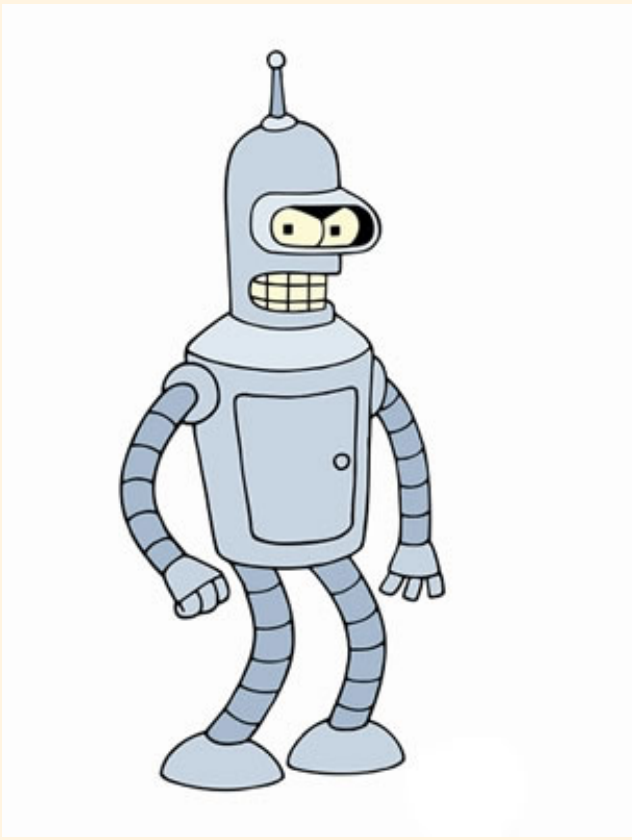
Curiosidades (I)

- The Hello World Collection

  http://www.roesler-ac.de/wolfram/hello.htm

Curiosidades (IIa)

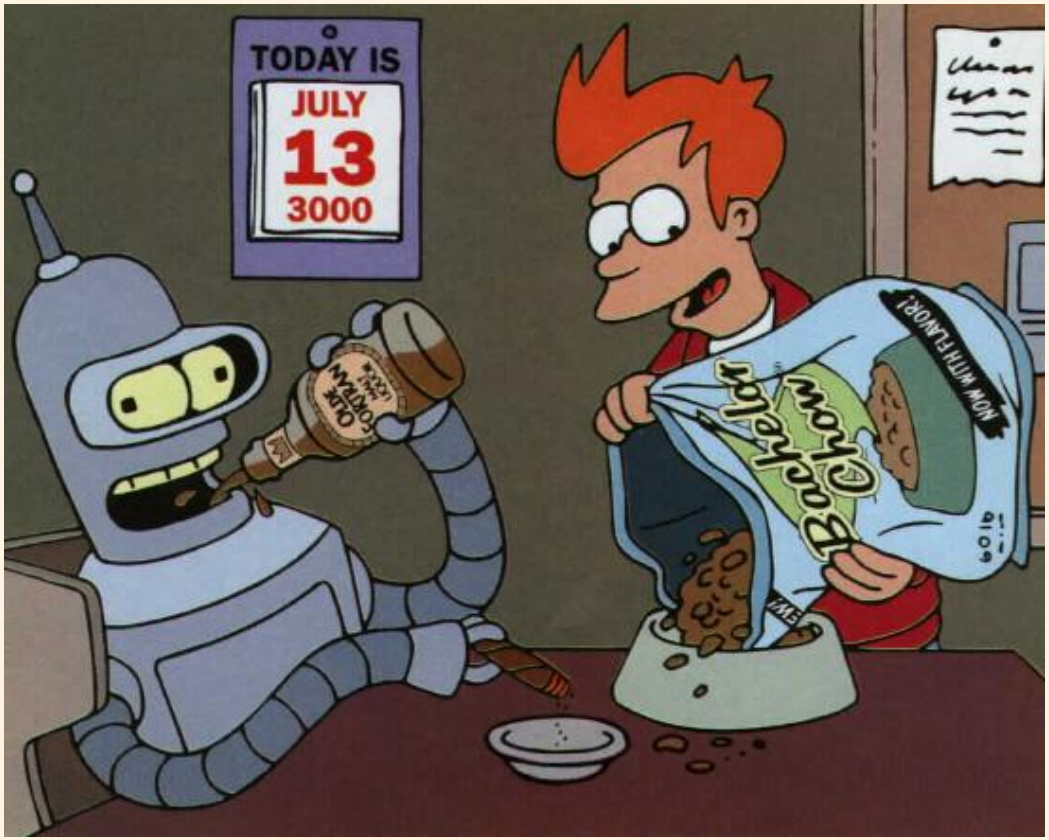
- ¿Cómo serán los lenguajes de programación en el futuro?

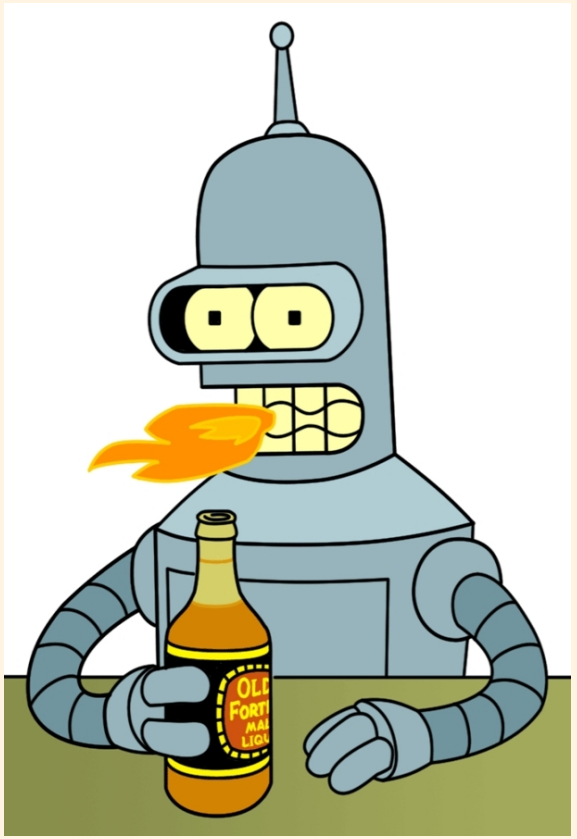


Cartel de FUTURAMA (izda.) y Robot BENDER (derecha).
[Matt Groening y David X. Cohen, FOX Broadcasting Company]

Curiosidades (IIb)

- **¿Cómo serán los lenguajes de programación en el futuro?**





BENDER con FRY (izda.) y BENDER (derecha) bebiendo.
[Matt Groening y David X. Cohen, FOX Broadcasting Company]

Curiosidades (IIc)

- **¿Cómo serán los lenguajes de programación en el futuro?**

Caja (izda.) y botella (derecha) de OLDE FORTRAN.
[Matt Groening y David X. Cohen, FOX Broadcasting Company]