Historia de los Lenguajes de Programación

AÑOS 1960-1969

MANUEL ÁNGEL RUBIO JIMÉNEZ

Historia de los Lenguajes de Programación

Años 1960-1969

Manuel Ángel Rubio Jiménez

ISBN 978-84-124520-4-4

Capítulo 22. FORTH y la cuarta generación de lenguajes

> FORTH es un amplificador. Un buen programador hará un trabajo fantástico con FORTH; un mal programador hará uno desastroso.
>
> — Charles H. Moore

En 1993 se celebró una reunión para hablar sobre la Historia de los Lenguajes de Programación. Una de las referencias (Rather etal., 1993) fue la sesión número 13 (XIII) presentada por sus creadores, Elizabeth D. Rather, Donald R. Colburn y Charles H. Moore.

Su introducción señaló a FORTH como un lenguaje único porque su desarrollo y proliferación no contó con el respaldo de ninguna empresa ni institución importantes y fue desarrollado inicialmente por una sola persona: Charles H. Moore.

22.1. Charles H. Moore

Al igual que John Backus, Charles H. Moore comenzó su carrera profesional trabajando en el cálculo de efemérides, elementos orbitales y posiciones de los satélites naturales en el Observatorio Astrofísico de Smithsonian en la década de 1950.

La programación de estos elementos rellenaba su escritorio con gran cantidad de tarjetas perforadas y ante la necesidad de hacer su tarea más liviana desarrollo un programa intérprete simple. Con este

intérprete podía componer diferentes ecuaciones para diferentes satélites sin necesidad de tener que recompilar. Este intérprete disponía de diferentes comandos y conceptos como un comando para leer *palabras* separadas por espacios y uno para convertir números en formato texto a código máquina además de un bloque condicional `IF..ELSE`.

Figura 37. Charles Moore

Moore descubrió que la entrada de texto en formato libre era más eficiente produciendo código más pequeño y rápido y más confiable que la forma de programar en FORTRAN empleando columnas fijas.

En 1961, Moore se licenció en Física en el MIT y entró en la escuela de postgrado de Stanford donde también aceptó un trabajo a tiempo parcial en el Acelerador Lineal de Stanford [SLAC], escribiendo código para optimizar la dirección del haz del acelerador de partículas de 2 millas.

Este nuevo puesto de trabajo fue ideal para poder continuar su trabajo con el intérprete. En el SLAC disponían de un programa llamado CURVE, desarrollado en ALGOL para ajustar los datos de corrección diferencial no lineal de propósito general. Su intérprete le ayudó a manejar este programa. Para ello mejoró su intérprete para permitir el uso de

parámetros a través de una pila de llamadas, variables, operadores aritméticos y de comparación y la definición e interpretación de procedimientos.

Una vez terminó su postgrado (1965) se trasladó a Nueva York donde comenzó a trabajar como autónomo programando en FORTRAN, ALGOL, JOVIAL, PL/I y varios ensambladores mientras seguía mejorando y trabajando en su intérprete. De hecho, llevaba a todas partes su *baraja de cartas*, una colección de tarjetas perforadas donde tenía el código de su intérprete usándolo como base para recodificarlo cuando fuese necesario.

A finales de la década de 1960 aparecieron las minicomputadoras y con ellas los terminales teletipo por lo que ya no era necesario trabajar con tarjetas perforadas. Moore agregó entonces a su intérprete operaciones para gestionar la entrada y salida de caracteres.

En 1968 tras casarse y mudarse a una ciudad más pequeña, Amsterdam, en el estado de Nueva York, entró a trabajar para la empresa Mohasco Industries donde desarrolló programas de gráficos informáticos para una computadora IBM 1130 con pantalla gráfica usando un compilador de FORTRAN.

Moore consiguió implementar a través de una mezcla de FORTRAN y ensamblador un editor primitivo y herramientas para la gestión de fuentes en un sistema que permitía mostrar imágenes animadas en 3D mientras que el software diseñado por IBM solo podía dibujar imágenes estáticas en 2D.

Por diversión también implementó una versión de Spacewar, uno de los primeros videojuegos y convirtió su programa ALGOL Chess (ajedrez) al nuevo lenguaje al que finalmente bautizó como FORTH.

22.2. FORTH

El lenguaje FORTH tenía como objetivo el desarrollo de software para la cuarta generación de computadoras que Moore consideraba que se caracterizarían por ser pequeñas computadoras distribuidas y por ese motivo aunque el nombre original era FOURTH tuvo que eliminar la *U* porque el nombre de los ficheros se limitaba a 5 caracteres.

Los computadores como el IBM 1130 disponían de unos programas específicos para poder ejecutar el código hecho en FORTRAN. Específicamente FORTRAN disponía del ensamblador para definir el compilador y el supervisor y a su vez el supervisor para especificar el lenguaje de control de tareas. Esta jerarquía de lenguajes queda fuera de la vista de los programadores, pero sigue ahí. Moore tomó conciencia de que FORTRAN no aprovechaba la potencia completa del IBM 1130 y fue cuando amplió su lenguaje.

Moore desarrolló FORTH para atender a las necesidades propias y por la frustración de no encontrar óptima toda esa jerarquía de lenguajes constituyendo para él una *torre de Babel* difícil de entender. Por ese motivo FORTH sustituía toda esa jerarquía para simplificar la comunicación entre el programador y la máquina.

Su objetivo se centró en **reducir el esfuerzo requerido y aumentar la calidad del programa producido** (Moore, 1970):

> He escrito muchos programas a lo largo de los años. He intentado escribir buenos programas y he observado la forma en que los escribo de manera bastante crítica. Mi objetivo ha sido reducir el esfuerzo requerido y aumentar la calidad del producto. En el transcurso de estas observaciones, me he encontrado cometiendo los mismos errores una y otra vez. Errores que son obvios en retrospectiva, pero difíciles de reconocer en contexto. Pensé que si escribía una receta para la programación, al menos podría recordarme los problemas. Y si el resultado es valioso para mí, debería serlo para otros [...]

Los principios clave de Moore pueden resumirse en los siguientes puntos:

Simplicidad

Uno de sus principios clave. El gran problema de los lenguajes de programación es su evolución, cada vez que se agrega una nueva característica debe revisarse su conjunto y estimar si el conjunto funciona y no se ha hecho más complejo, si mantiene una compatibilidad y coherencia entre todas y cada una de sus partes.

No especular

Uno de los grandes enemigos de Moore era la especulación y consideraba a este enemigo un gran problema entre los programadores porque siempre intentan especular sobre necesidades futuras y satisfacerlas.

Flexible

Es esencial contar con un sistema flexible. Un sistema que puede realizar cualquier cosa lo cual es muy diferente al enfoque de dejar abiertas ciertas partes del propio código en espera de cambios. Mientras que el primero, el enfoque utilizado por Moore, permite hacer de *todo*, el segundo termina generando un sistema enorme de propósito general poco flexible y difícil de manejar.

Hazlo tú mismo

Las generalidades no eran del agrado de Moore. Moore consideraba óptimo realizar el código que necesites al momento de necesitarlo sin agregar generalidades que no se llegarán a usar más allá de lo realizado. Es más, aseguraba que escribir mil veces la misma subrutina ayuda al programador a hacerlo cada vez mejor.

Este último punto se contradice con el uso de librerías estándares pero Moore en realidad habla en este punto de la elaboración del lenguaje de programación, de su compilador y su implementación.

Moore llevó de forma recta y sin desviaciones su propia doctrina implementando desde cero y sin copiar código de una plataforma de computación a otra su código de FORTH. Realizó la implementación de 18 plataformas diferentes y para cada una implementó su propio compilador, sus propios controladores de disco, terminal e incluso sus propias subrutinas de multiplicación y división para sistemas que no

dispusieran de estas en su código ensamblador. De hecho eran la mayoría.

Este era un reflejo de su inconformismo y no estar completamente satisfecho con su trabajo anterior de modo que siempre buscaba formas de hacerlo mejor en la siguiente vez y por eso prefería hacer una implementación desde cero de cada código hecho por sí mismo y ver si podía mejorar algo. Este inconformismo sería una fuente de frustración para Elizabeth Rather en FORTH, Inc. pero no nos adelantemos.

FORTH resultó ser gracias a estas iteraciones en código un sistema pequeño, simple, limpio y extremadamente flexible.

22.3. Elisabeth Rather

La primera implementación de FORTH se llevó a cabo en 1971 para un radiotelescopio de 11 metros del Observatorio Nacional de Radioastronomía [NRAO] en Kitt Peak, Arizona. Este telescopio funcionaba con dos minicomputadoras de [DEC] enlazadas por puerto serie: una DDP-116 de 16KB y una H316 de 32KB.

Las actividades del telescopio eran:

- Apuntar y rastrear el telescopio.
- Recolectar datos y registrarlos en una cinta magnética.
- Brindar soporte a una terminal gráfica interactiva para analizar los datos registrados previamente.

El sistema era único para la época porque el programa también se almacenaba en la cinta magnética e incluso se auto-abastecía leyendo más código en caso de ser indicado. Esto era posible porque FORTH estaba escrito en FORTH, era un lenguaje que podía compilarse a sí mismo con la ayuda de una unidad mínima inicial.

Las capacidades de multiusuario que permitía a los astrónomos de NRAO para analizar gráficamente los datos mientras un operador

controlaba el telescopio y los datos fluían en vivo era algo inaudito en la época. Pero el problema de Edward K. Conklin, jefe de la división de Tucson de NRAO era que el software era desarrollado por Moore desde Charlottesville, Virginia. Así que contrató a Elizabeth Rather.

Elizabeth Rather era una analista de sistemas de la Universidad de Arizona que aceptó un trabajo a tiempo parcial en el NRAO. Cuando Rather escuchó que el telescopio era operado con un lenguaje poco o nada documentado que tan solo conocía una persona, su creador, se horrorizó y su primer impulso fue reescribirlo todo en FORTRAN, pero no había ni tiempo ni dinero por lo que se decidió a aprender y documentar el sistema lo mejor que pudo.

Después de dos meses, Rather comenzó a darse cuenta de algo asombroso. A pesar de la rareza del lenguaje, a pesar de la falta de expertos o recursos locales, en tan solo la mitad de la semana utilizando FORTH podía hacer mucho más que en el resto de la semana utilizando mainframes mucho más potentes a los que tenía acceso. La respuesta al porqué parecía obvia, la atención del programador no se ve interrumpida en ningún momento por la necesidad de abrir y cerrar archivos, cargar y ejecutar: compiladores, enlazadores, cargadores y depuradores. Todo es más flexible en FORTH y está siempre disponible.

Figura 38. Elizabeth Rather

Rather abandonó inmediatamente la Universidad y se dedicó en exclusiva al NRAO y durante los dos años siguientes (1973) escribió el manual de FORTH llegando a propagarse su fama y su uso en sitios como el Observatorio Steward, el MIT, el Imperial College (Londres), el Observatorio Interamericano de Cerro Tololo (Chile) y la Universidad de Utrecht (Países Bajos). Su uso fue tal que se adoptó oficialmente como lenguaje estándar por la Unión de Astronómica Internacional (1976).

Pero antes de este último evento, Moore, Rather y Conklin formaron la compañía FORTH, Inc. para explorar los usos comerciales del lenguaje. En 1976 desarrollaron versiones comerciales del lenguaje para la mayoría de minicomputadoras y tomaron como clientes a todos sus usuarios, observatorios principalmente.

Cuando un cliente adquiría una máquina, inicialmente algunas de ellas venían con muchas limitaciones y las que proporcionaban un sistema operativo de fábrica estaban aún así muy limitadas a nivel multiusuario degradándose enormemente con tan solo 8 usuarios. Cuando Moore y sus colegas instalaban FORTH muchas de estas limitaciones desaparecían o se ampliaban permitiendo a los usuarios utilizar un editor de alto nivel con posibilidad de escribir código en FORTH y Ensamblador soportando hasta 64 usuarios.

Agregar FORTH a una nueva máquina le llevaba a Moore unas dos semanas porque gran parte del código de FORTH estaba escrito en FORTH, solo las partes críticas del sistema o las que requerían un gran rendimiento estaban en ensamblador.

A lo largo de 1970, FORTH Inc. fue creciendo y se comenzaron a portar incluso a los primeros microprocesadores.

22.4. El lenguaje

FORTRAN se basa en el álgebra mientras que FORTH se basa en la prosa inglesa, aunque hay quien piensa que más bien podría ser a la prosa alemana por su notación posfija. Sus elementos (palabras) son datos (sustantivos) y procedimientos (verbos), permitiendo definir nuevos verbos a través de palabras reservadas e incrustar código ensamblador.

Las palabras de FORTH son equivalentes a las subrutinas, funciones o procedimientos de otro tipo de lenguajes. Las palabras también pueden constituir comandos porque FORTH difumina la distinción entre elementos lingüísticos y elementos funcionales.

FORTH identifica las palabras como cualquier cadena de caracteres rodeada por espacios. No hay caracteres especiales. Las palabras pueden ser de tres tipos: definidas por Forth, números y no definidas.

Veamos un programa en FORTH para ilustrar lo que llevamos comentado (Brodie, 1981):

```
: STAR 42 EMIT ; ①
ok ②

CR STAR CR STAR CR STAR ③
*
*
*ok ④

: MARGIN CR 5 SPACES ; ⑤
ok

MARGIN STAR MARGIN STAR ⑥
     *
     *ok ⑦

: STARS 0 DO STAR LOOP ; ⑧
```

```
ok

5 STARS
*****ok ⑨
```

① Definimos una nueva palabra usando : para que se emita el carácter 32 cada vez que usemos STAR.

② Siempre que ejecutemos un comando obtendremos una salida y el resultado, en este caso ok.

③ Así indicamos que queremos ejecutar palabras CR que es el retorno de carro y STAR.

④ Como siempre obtenemos nuestro retorno tras la salida de la ejecución de esos 3 CR y 3 STAR combinados.

⑤ Definimos ahora MARGIN que nos dará 5 espacios (SPACES).

⑥ Probamos a ejecutar 2 MARGIN y 2 STAR combinados.

⑦ Nos retorna 5 espacios, una estrella, 5 espacios y una estrella acabando con ok.

⑧ Definimos ahora STARS como un bucle para imprimir estrellas comenzando desde 0 el bucle.

⑨ Indicamos el valor máximo del bucle al emplear la palabra.

Este lenguaje como ves guarda relación más con otros lenguajes como BASIC y LOGO pero presenta una forma muy simple. En el caso del bucle puedes ver que no está definido completamente sino que su valor máximo se proporciona cuando se emplea la definición.

La forma de funcionamiento como decíamos se basa en un diccionario donde se van almacenando las palabras de que dispone FORTH y las palabras que vamos definiendo. Si una palabra no está en el diccionario se genera un error.

Los cálculos sin embargo son diferentes, es decir, como hemos visto las palabras se procesan y los números son un tipo específico de palabras

que no hace falta definir. Se emplean principalmente como parámetros, pero ¿Cómo hacemos los cálculos?

```
3 4 + . ①
7 ok ②

: FOUR-MORE 4 + ; ③
ok

3 FOUR-MORE . ④
7 ok
```

① En este caso se usa la pila poniendo en ella el 3, luego el 4 y finalmente se ejecuta la suma (+). El comando del punto (.) es el encargado de dar por finalizada la operación y retornar lo que haya en la pila.

② Como vemos, el resultado es 7.

③ La pila es general por lo que una función puede empujar un elemento en la pila y acto seguido realizar la suma como función.

④ Al ejecutar vemos que inicialmente ponemos 3 en la pila y después ejecutamos FOUR-MORE que se encarga de poner 4 y después ejecutar la suma obteniendo exactamente el mismo resultado.

22.5. Problemas con FORTH

Tal como reza la frase de apertura. El creador de FORTH consideraba al lenguaje como un gran lenguaje para buenos programadores permitiendo escribir programas en muy poco tiempo pero también una muy mala opción para malos programadores porque los resultados podrían ser desastrosos. Esto llevo a la idea de pensar de FORTH que era inmanejable con algo de publicidad negativa en la década de 1980 con el desastre del proyecto VALDOCS de Epson.

En realidad, los problemas de estos proyectos hubiesen sido similares con otros lenguajes porque eran comunes: definición inadecuada, una administración del proyecto pobre y expectativas poco realistas.

Aunque FORTH es un lenguaje que se puede seguir encontrando en la actualidad, su auge fue principalmente en la década de 1970 y 1980 y ahora está muy restringido a proyectos específicos, principalmente embebidos, robótica, control industrial y la exploración espacial. Al igual que LOGO constituye el ejemplo de un lenguaje que no ha desaparecido pero que es muy difícil de encontrar hoy en día.

Glosario

Este glosario incluye algunas palabras empleadas a lo largo del libro y cuya definición sea importante mantener localizada.

ACE

Automatic Computing Engine o Máquina de Computación Automática. Fue la Máquina desarrollada en Reino Unido por Alan Turing.

ACM

Association for Computing Machinery o Asociación de Maquinaria de Computación.

AED-0

Automated Engineering Design o Diseño de Ingeniería Automatizado, aunque las siglas también se entendieron como *ALGOL Extended for Design* o ALGOL Extendido para el Diseño.

AIMACO

Air Material Compiler o Compilador de Material Aéreo.

ALGOL

ALGOrithmic Language o Lenguaje Algorítmico fue un lenguaje creado por un conjunto de científicos dedicados a la computación de EE.UU. y Europa. Ver el Capítulo 3.

ANSI

American National Standards Institute o Instituto de Estándares Nacional Americano. Una institución de estándares estadounidense.

ASCII

American Standard Code for Information Interchange o Código Estándar Americano para el Intercambio de Información fue desarrollado por los Laboratorios Bell en 1968.

BASIC

Beginners All-Purpose Symbolic Instruction Code o Código de Instrucciones Simbólico para Todo-Propósito para Principiantes.

BBN

Bolt, Beranek y Newman, empresa creada por los Richar Bolt, Leo Beranek y Robert Newman.

BCD

Binary-Coded Decimal o Decimal Codificado en Binario. Desarrollado y usado principalmente por IBM en código binario de 6 bits para codificar números, letras y caracteres especiales.

BCPL

Basic CPL o CPL Básico. Fue una simplificación de [CPL].

BEMA

Business Equipment Manufacturers Association o Asociación de Fabricantes de Equipos de Negocio.

BNF

Backus Normal Form o *Backus-Naur Form*, Forma Normal de Backus o Forma de Backus-Naur. Ver la Sección 3.2 en el Capítulo 3.

CAI

Computer-Aided Instruction o Instrucción Asistida por Computador.

CAL

Conversational Algebraic Language o Lenguaje Algebraico Conversacional.

CISL

Cambridge Information Systems Laboratory o Laboratorio de Sistemas de Información de Cambridge.

COBOL

Common Business Oriented Language o Lenguaje Común Orientado a los Negocios.

CODASYL

Conference On Data Systems Languages o Conferencia sobre Lenguajes de Sistemas de Datos.

COMTRAN

Comercial Translator o Traductor Comercial. Un lenguaje desarrollado por IBM para ser amistoso al mundo de los negocios.

CORAL

Computer On-line Real-time Applications Language o Lenguaje de Aplicaciones de Computador en Línea y en Tiempo-Real y muy posiblemente en sus primeras versiones *Computer On-line Radar Applications Language* o Lenguaje de Aplicaciones de Computador en Línea para Radar.

COWSEL

COntrolled Working SpacE Language o Lenguaje del Espacio de Trabajo Controlado.

CPL

Combined Programming Language o Lenguaje de Programación Combinado. En verdad el nombre sugiere muchas más posibles interpretaciones (ver Capítulo 12).

CTSS

Compatible Time-Sharing System o Sistema de Tiempo-Compartido Compatible. También se conoce como *Caltech Time-Sharing System* o Sistema de Tiempo-Compartido de Caltech por el Instituto de Tecnología de California donde fue creado.

DEC

Digital Equipment Corporation o Corporación de Equipos Digitales presidida inicialmente por Ken Olsen. Una gran empresa de computación entre la década de 1960 y 1990. En 1998 se fusionó a Compaq perdiendo su nombre en 2002 en otra fusión con Hewlett-Packard (HP).

DOPE

Dartmouth Oversimplified Programming Experiment o Experimento de Programación Sobre-simplificado de Dartmouth.

DSL

Domain Specific Language o Lenguaje de Dominio Específico. Son lenguajes que se crean para un propósito específico y su sintaxis y su semántica se diseñan para facilitar la expresión del dominio al que pertenecen.

ECMA

European Computer Manufacturers Association o Asociación Europea de Fabricantes de Computadores.

EPL

Early PL/I o PL/I temprano. Fue un subconjunto del lenguaje de programación PL/I desarrollado por McIlroy usando TMG para disponer del lenguaje PL/I en los sistemas operativos CTSS y GECOS empleados para crear Multics.

FARGO

Fourteen-o-one Automatic Report Generation Operation u Operación de generación automática de informes 1401. Es considerada la versión [RPG] 0.

FLPL

FORTRAN List Processing Language o Lenguaje de Procesamiento de Listas FORTRAN.

FOCAL

Formulating On-line Calculations in Algebraic Language o Formulación de Cálculos en línea en Lenguaje Algebraico. Aunque sus siglas también pudieron ser para *FOrmula CALculator* o Calculadora de Fórmulas.

FORMAC

FORmula MAnipulation Compiler o Compilador de Manipulación de Fórmulas.

GAMM

Gesellschaft für Angewandte Mathematik und Mechanik o Asociación de Matemática Aplicada y Mecánica.

GAT

Generalized Algebraic Translator o Traductor Algebraico Generalizado.

GE

General Electric.

GPM

General Purpose Macro-generator o Macro-generador de Propósito General. Ver la Sección 2.4.1 en el Capítulo 2.

GUIDE

Guidance for Users of Integrated Data-Processing Equipment u Orientación para Usuarios de Equipos Integrados de Procesamiento de Datos. Un grupo de usuarios de voluntarios de computadores y sistemas de IBM.

IA

Inteligencia Artificial.

IAL

International Algebraic Language o Lenguaje Algebraico Internacional. Fue el primer nombre otorgado al lenguaje que sería después nombrado como [ALGOL].

IBM

International Business Machines o Máquinas de Negocios Internacionales. Una empresa surgida a principios del siglo XX por la evolución de las empresas dedicadas a las máquinas de tabulación.

IFIP

International Federation for Information Processing o Federación Internacional para el Procesamiento de Información. Fue la federación a cargo de la definición de ALGOL 60 y ALGOL 68.

IPL

Information Processing Language o Lenguaje de Procesamiento de Información.

ISWIM

If you See What I Mean, Si ves lo que quiero decir. En el Capítulo 18.

IT

Internal Translator o Traductor Interno.

IVSYS

IVerson SYStem o Sistema de Iverson.

JOHNNIAC

John von Neumann Numerical Integrator and Automatic Computer o Integrador Numérico y Computadora Automática John von Neumann.

JOSS

JOHNNIAC Open Shop System o Sistema de Tienda Abierta JOHNNIAC fue un servicio de tiempo-compartido experimental de Rand Corporation diseñado para novatos en el mundo de la computación.

MAC

Proyecto MAC por sus siglas *Mathematics and Computation* o Matemáticas y Computación. Un proyecto conducido en la Universidad de Massachusetts en 1966 y del que se originaron algunas piezas clave como MacLisp.

MAD

Michigan Algorithm Decoder o Decodificador Algorítmico de Michigan.

MGH

Massachusetts General Hospital o Hospital General de Massachusetts.

MIT

Massachusetts Institute Technology o Instituto de Tecnología de Massachusetts.

MPPL

MultiPurpose Programming Language o Lenguaje de Programación MultiPropósito.

MUMPS

Massachusetts General Hospital Utility Multi-Programming System o Sistema de Programación-Múltiple de Servicios Públicos del Hospital General de Massachusetts.

NCSS

National Computer Software Systems o Sistemas de Software Computacional Nacionales.

NDRE

Norwegian Defense Research Establishment o Establecimiento Noruego de Investigación de Defensa.

NIH

National Institute of Health o Instituto Nacional de Salud.

NPL

National Physics Laboratory o Laboratorio de Física Nacional. En Reino Unido.

NRAO

National Radio Astronomy Observatory o Observatorio Nacional de Radioastronomía.

NSF

National Science Foundation o Fundación Nacional de Ciencias.

PCM

Pulse-Code Modulation o Modulación por Impulsos Codificados.

PILOT

Programmed Inquiry, Learning or Teaching o Investigación, aprendizaje o enseñanza programada.

PLATO

Programmed Logic for Automatic Teaching Operations o Lógica Programada para Operaciones de Enseñanza Automática.

POP

Package for Online Programming o Paquete para Programación en Línea. El nuevo nombre para el lenguaje [COWSEL].

REFAL

Recursive Functions Algorithmic Language o Lenguaje Algorítmico de Funciones Recursivas.

REPL

Read, Eval, Print and Loop o Lee, Evalúa, Imprime y Bucle. Indica la interacción de la máquina con el usuario, lee la instrucción, la evalúa, imprime el resultado y vuelta a comenzar.

RPG

Report Program Generator o Generador de Programas de Informes.

Ver el Capítulo 1.

SAP

Symbolic Assembly Program o Programa Ensamblador Simbólico. Ver el Capítulo 2.

SET

SET Language o Lenguaje de Conjuntos.

SCALP

Self Contained ALgol Processor o Procesador ALGOL Autocontenido.

SHARE

Grupo de usuarios voluntarios para computadores mainframe de IBM fundado en 1955.

SIL

SNOBOL Implementation Language o Lenguaje de Implementación de SNOBOL.

SLAC

Stanford Linear Accelerator o Acelerador Lineal de Stanford.

SQL

Structured Query Language o Lenguaje de Consultas Estructurado. Un lenguaje que podemos considerar de cuarta generación, declarativo o incluso de dominio específico aunque en nuestros días haya evolucionado hasta ser Turing completo.

STC

Standard Telephones and Cables o Cables y Teléfonos Estándares. Fue una empresa de telecomunicaciones de Reino Unido pionera en tecnologías como la fibra óptica.

THE

Technische Hogeschool Eindhoven o Universidad de Tecnología de Eindhoven. Se empleó como el nombre del sistema operativo

liderado por Edsger W. Dijkstra.

TMG

TransMoGrifiers un lenguaje creado por McClure en 1965 para escribir compiladores.

UMMPS

University of Michigan Multi-Programming Supervisor o Supervisor de Multi-Programación de la Universidad de Michigan.

URSS

Unión de Repúblicas Socialistas Soviéticas también conocida como Unión Soviética (US) era un conjunto de hasta 15 Repúblicas Socialistas Soviéticas (RSS) formadas desde 1922 hasta su disolución en 1991 que tuvieron un papel importante en la Segunda Guerra Mundial contra la Alemania Nazi y en la Guerra Fría contra Estados Unidos (EEUU).

VDM

Vienna Development Method o Método de Desarrollo de Viena.

Bibliografía

Mucha de la información obtenida para la realización de este libro ha partido de una búsqueda en Internet y principalmente la lectura de entradas del proyecto Wikipedia. También se han adquirido algunos documentos, tesis y publicaciones de sitios como ACM Digital Library para contrastar información contradictoria recurriendo directamente a las fuentes.

Muchos de esos artículos revisados, sitios web o libros consultados se encuentran a continuación.

Abrahams, P. W., Barnett, J. A., Book, E., Firth, D., Kameny, S. L., Weissman, C., Hawkinson, L., Levin, M. I., & Saunders, R. A. (1966). The LISP 2 programming language and system. *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, 661-676. doi.org/10.1145/1464291.1464362

Abrams, P. S. (1966). *An interpreter for "Iverson notation"*. Stanford University.

Abrams, P. S. (1975). What's wrong with APL? *Proceedings of Seventh International Conference on APL*, 1-8. doi.org/10.1145/800117.803777

Arden, B. W., Galler, B. A., & Graham, R. M. (1961). MAD at Michigan: its function & features. *Datamation, 7*(12), 27-28.

Arden, B., & Graham, R. (1959). On GAT and the Construction of Translators. *Commun. ACM, 2*(7), 24-26. doi.org/10.1145/368370.368373

Bachelor, G. A., Dempster, J. R. H., Knuth, D. E., & Speroni, J. (1961). SMALGOL-61. *Commun. ACM*, *4*(11), 499-502. doi.org/10.1145/366813.366843

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., Woodger, M., & Naur, P. (1960). Report on the Algorithmic Language ALGOL 60. *Commun. ACM*, *3*(5), 299-314. doi.org/10.1145/367236.367262

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. *IFIP Congress*. api.semanticscholar.org/CorpusID:44764020

Barron, D. W., Buxton, J. N., Hartley, D. F., Nixon, E., & Strachey, C. (1963). The Main Features of CPL. *The Computer Journal*, *6*(2), 134-143. doi.org/10.1093/comjnl/6.2.134

Beechhold, H. F. (1993). *Vanilla PILOT*. *5*(37), 67-69.

Berner, R. W. (1957). How to consider a computer. *Automatic Control Magazine*, 66-69.

Berner, R. W. (1971). A View of the History of COBOL. *Honeywell Computer Journal*, *5*.

Beyer, K. W. (2012). *Grace Hopper and the Invetion of the Information Age*. Smithsonian Institution.

Boytchev, P. (2010). *Logo Tree Project*. www.edtechpolicy.org/cyberk12ARCHIVE/Resources/Microworlds/LogoTreeProject.pdf

Brock, D. C. (2020). *Discovering Dennis Ritchie's Lost Dissertation*. computerhistory.org/blog/discovering-dennis-ritchies-lost-dissertation/

Brodie, L. (1981). *Starting FORTH*. www.forth.com/wp-content/uploads/2018/01/Starting-FORTH.pdf

Caine, S. H., & Gordon, E. K. (1968). *TTM: An Experimental Interpretive Language*. California Institute of Technology.

Chatley, R., Donaldson, A., & Mycroft, A. (2022). The Next 7000 Programming Languages. En *Computing and Software Science* (pp. 250-282). Springer-Verlag. doi.org/10.1007/978-3-319-91908-9_15

Christopher, T. W. (1996). *EULER: An Experiment in Language Definition*. Illinois Institute of Technology.

Cocke, J., & Schwartz, J. T. (1970). *Programming Languages and Their Compilers*. Courant Institute of Mathematical Sciences.

Corbató, F. J., Merwin-Daggett, M., & Daley, R. C. (1962). An Experimental Time-Sharing System. *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*, 335-344. doi.org/10.1145/1460833.1460871

Corporation, C. D. (1977). *RPG II Version 2 Reference Manual*. doi.org/96768710

Corporation, I. B. M. (1964). *FARGO for IBM 1401* [Form]. Systems Reference Library. bitsavers.org/pdf/ibm/1401/C24-1464-3_1401_fargo.pdf

Davis, M., & Schonberg, E. (2011). *Jacob Theodore Schwartz*. www.settheory.com/Schwartz_Jacob_Martin_and_Ed.pdf

Duncan, F. G. (1967). ALGOL Bulletin No. 26. *SIGPLAN Not.*, *2*(11), 1-49. doi.org/10.1145/1139498.1139500

Evans, A. (1968). PAL—a language designed for teaching programming linguistics. *Proceedings of the 1968 23rd ACM National Conference*, 395-403. doi.org/10.1145/800186.810604

Farber, D. J. (1964). *635 Assembly System - GAP*. Bell Telephone Laboratories Computation Center.

Farber, D. J. (1971). A Survey of the Systematic Use of Macros in

Systems Building. *SIGPLAN Not.*, *6*(9), 29-36. doi.org/10.1145/942596.807057

Fonsecai, P., & Casanovas, C. J. (2009). JGPSS, An open source GPSS framework to teach simulation. *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 256-267. doi.org/10.1109/WSC.2009.5429335

Forest, B. (1961). BALGOL at Stanford: a fast compiler on a slow computer. *Datamation*, *7*(12), 24-26.

Gough, J. (1993). *Watching the Skies: History of Groud Radar for the Defence of the United Kingdom by the Royal Air Force from 1946 to 1975*. Stationery Office Books.

Graham, R. M. (1958). Translation between Algebraic Coding Languages. *Preprints of Papers Presented at the 13th National Meeting of the Association for Computing Machinery*, 1-2. doi.org/10.1145/610937.610964

Harvey, B. (1997). *Computer Science Logo Style*. The MIT Press.

Hoare, C. A. R. (1961). Algorithm 63: Partition. *Commun. ACM*, *4*(7), 321. doi.org/10.1145/366622.366642

Hoare, C. A. R. (1962). Quicksort. *The Computer Journal*, *5*(1), 10-16. doi.org/10.1093/comjnl/5.1.10

Hoare, C. A. R. (1965). *A Programming Language for Processor Construction*.

Hoare, T. (2009). *Null References: The Billion Dollar Mistake*. www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

Holbrook, B. D., & Brown, W. S. (1982). *A History of Computing Research at Bell Laboratories (1937-1975)* [Computing Science Technical Report]. AT&T Bell Laboratories.

Holmevik, J. R. (1994). Compiling SIMULA: a historical study of technological genesis. *IEEE Annals of the History of Computing*, *16*(4), 25-37. doi.org/10.1109/85.329756

Höltgen, S., & Baranovska, M. (2022). *Hello, I'm Eliza: Fünfzig Jahre Gespräche mit Computern*.

Igarashi, S., Iwamura, T., Sakuma, K., Simauti, T., Simuzu, T., Takasu, S., Wada, E., & Yoneda, N. (1969). ALGOL N. *ALGOL Bull.*, *30*, 38-85.

Irons, E. T. (1970). Experience with an extensible language. *Commun. ACM*, *13*(1), 31-40. doi.org/10.1145/361953.361966

Iverson, K. E. (2000). A personal view of APL. *SIGAPL APL Quote Quad*, *30*(3), 4-13. doi.org/10.1145/360487.360477

Iverson, K. E. (1962). *A programming language*. John Wiley & Sons, Inc.

Kelly, B. (2009). *IBM RPG: A Great Language with a Greater History*. www.nicklitten.com/a-brief-history-of-the-ibm-rpg-programming-language/

Kurtz, T. E. (1978). BASIC Session. En *History of Programming Languages* (pp. 515-550). Association for Computing Machinery. doi.org/10.1145/800025.1198360

Lampson, B. (2011). *A Culture of Innovation* (D. Walden & R. Nickerson, eds.). Waterside Publishing.

Landin, P. J. (1966). The next 700 programming languages. *Commun. ACM*, *9*(3), 157-166. doi.org/10.1145/365230.365257

Ltd, F. (1968). *FM1600B Microcircuit Computer*. www.sba.unipi.it/sites/default/files/2015_05_29_08_44_13.pdf

McCarthy, J. (1959). *Memorandum to P. M. Morse Proposing Time-Sharing*. jmc.stanford.edu/computing-science/timesharing-memo.html

McCarthy, J. (1960). Recursive functions of symbolic expressions and

their computation by machine, Part I. *Commun. ACM, 3*(4), 184-195. doi.org/10.1145/367177.367199

McCarthy, J. (1962). Time Sharing Computer Systems. *Management and the Computer of the Future*, Chapter 6.

McCarthy, J. (1978). History of LISP. En *History of Programming Languages* (pp. 173-185). Association for Computing Machinery. doi.org/10.1145/800025.1198360

McIlroy, M. D. (1960). Macro Instruction Extensions of Compiler Languages. *Commun. ACM, 3*(4), 214-220. doi.org/10.1145/367177.367223

Mills, D. L. (1968). *The Syntactic Structure of MAD/I*. University Michigan. apps.dtic.mil/sti/citations/AD0671683

Mooers, C. N., Deutsch, L. P., & Floyd, R. W. (1965). Programming Languages for Non-Numeric Processing—1: TRAC, a Text Handling Language. *Proceedings of the 1965 20th National Conference*, 229-246. doi.org/10.1145/800197.806048

Moore, C. H. (1970). *Programming a Problem-oriented Language*. colorforth.github.io/POL.html

Nelson, T. (1974). *Computer Lib/Dream Machines*.

of California, O. A. (1998). *Register of the John A. Starkweather Papers, 1965-1985*. oac.cdlib.org/view?docId=tf2d5nb1xg;style=oac4;doc.view=entire_text

Papert, S. (1980). *Mindstorms*. Harvester Press.

Perlis, A. J. (2008). *ALGOL, More than just ALGOL*. api.semanticscholar.org/CorpusID:47672972

Popplestone, R. (1999). *The Early Development of POP*. www-robotics.cs.umass.edu/Popplestone/pop_development.html

Pouzin, L. (1965). *The SHELL: A Global Tool for Calling and Chaining Procedures in the System*. Massachusetts Institute of Technology. people.csail.mit.edu/saltzer/Multics/Multics-Documents/MDN/MDN-4.pdf

Pouzin, L. (2000). *The Origin of the Shell*. multicians.org/shell.html

Radin, G. (1978). The early history and characteristics of PL/I. *SIGPLAN Not.*, *13*(8), 227-241. doi.org/10.1145/960118.808389

Rather, E. D., Colburn, D. R., & Moore, C. H. (1993). The evolution of Forth. *The Second ACM SIGPLAN Conference on History of Programming Languages*, 177-199. doi.org/10.1145/154766.155369

Rawlings, N. (2014). The History of NOMAD: A Fourth Generation Language. *IEEE Annals of the History of Computing*, *36*(1), 30-38. doi.org/10.1109/MAHC.2014.10

Richards, M. (2012). How BCPL Evolved from CPL. *The Computer Journal*, *56*(5), 664-670. doi.org/10.1093/comjnl/bxs026

Ritchie, D. M. (1968). *Program Structure and Computacional Complexity*. archive.computerhistory.org/resources/access/text/2020/05/102790971/Ritchie_dissertation.pdf

Ritchie, D. M. (1993). The development of the C language. *The Second ACM SIGPLAN Conference on History of Programming Languages*, 201-208. doi.org/10.1145/154766.155580

Rubio Jiménez, M. Á. (2021). *Historia de los Lenguajes de Programación: años 1940-1959*. Altenwald Books.

Sammet, J. E. (1969). *Programming Languages: History and Fundamentals*. Prentice-Hall, Inc. doi.org/10.5555/1096897

Sammet, J. E. (1978). The early history of COBOL. En *History of Programming Languages* (pp. 199-243). Association for Computing Machinery. doi.org/10.1145/800025.1198367

Serrão, R. G. (2022). *Why APL is a language worth knowing*. mathspp.com/blog/why-apl-is-a-language-worth-knowing

Shapiro, J. S. (2004). Extracting the Lessons of Multics. *login Usenix Mag.*, *29*(6). www.usenix.org/publications/login/december-2004-volume-29-number-6/extracting-lessons-multics

Sherwood, B. (1974). *The TUTOR Language*.

Starkweather, J. A. (1967). *Computer-Assisted Learning in Medical Education*. *97*.

Steele, G. L., & Gabriel, R. P. (1996). The evolution of Lisp. En *History of Programming Languages—II* (pp. 233-330). Association for Computing Machinery. doi.org/10.1145/234286.1057818

Strachey, C. (1965). A general purpose macrogenerator. *The Computer Journal*, *8*(3), 225-241. doi.org/10.1093/comjnl/8.3.225

Strachey, C. S. (1959). Time sharing in large, fast computers. *IFIP Congress*. api.semanticscholar.org/CorpusID:5144680

Thompson, K. (1972). *Users' Reference to B*. Bell Telephone Laboratories. www.bell-labs.com/usr/dmr/www/kbman.pdf

Turchin, V. F. (1979). A supercompiler system based on the language REFAL. *SIGPLAN Not.*, *14*(2), 46-54. doi.org/10.1145/954063.954069

van Wijngaarden, A. (1963). *Generalized Algol* (Número MR 57/63/R).

Ware, ed., Willis H. (1960). *Soviet Computer Technology - 1959*. RAND Corporation.

Wayne, H. (2020). *10 Most(ly Dead) Influential Programming Languages*. www.hillelwayne.com/post/influential-dead-languages/

Wirth, N., & Weber, H. (1966). EULER: A generalization of ALGOL and its formal definition: Part 1. *Commun. ACM*, *9*(1), 13-25. doi.org/10.1145/365153.365162

Wirth, N., & Weber, H. (1966). EULER: a generalization of ALGOL, and its formal definition: Part II. *Commun. ACM, 9*(2), 89-99. doi.org/10.1145/365170.365202

Woodie, A. (2020). *Is it time to rename RPG?*. www.itjungle.com/2020/08/24/is-it-time-to-rename-rpg/

Woodward, P. M., Wetherall, P. R., & Gorman, B. (1973). *Official Definition of CORAL 66*. archive.org/details/official-definition-of-coral-66

Yngve, V. H. (1957). A Framework for Syntactic Translation. *Mechanical Translation, 4*(3), 59-65. aclanthology.org/www.mt-archive.info/50/MT-1957-Yngve.pdf

Yngve, V. H. (1962). COMIT as an IR language. *Commun. ACM, 5*(1), 19-28. doi.org/10.1145/366243.366720

Yost, J. R. (2014). *An Interview with Butler Lampson*. Charles Babbage Institute.

Cohen, I. B., Welch, G. W., & Campbell, R. V. D. (eds.). (1999). *Makin' numbers: Howard Aiken and the computer*. MIT Press.

Early Multics Development and the MSPM. (1965-1969). multicians.org/mspmtoc.html

PL/I: Language Specifications. (1965). IBM. bitsavers.org/pdf/ibm/360/pli/C28-6571-1_PL_I_Language_Specifications_Jul65.pdf