Índice general

Índice de figuras

Índice de cuadros

Capítulo 3

El Lenguaje de Programación Pascal

3.1. Introducción

A programming language called Pascal is described which was developed on the vasis of Algol 60. Compared to Algol 60, its rango of applicability is considerably increased due to a variety of data structuring facilities. In view of its intended usage both as convenient basis to teach programming and as an efficient tool to write large programs, emphasis was placed on keeping the number of fundamental concepts reasonably small, on a simple and systematic language structure, and on efficient implementability.A one-pass compiler has been contructed for the CDC 6000 computer family; it is expressed entirely in terms of Pascal itself. [1] [Wir71]

El Lenguaje de Programación Pascal fué creado por el profesor Niklaus Wirth[2] a finales de la decada de los sesenta del siglo XX. En 1970 fué finalmente publicado, fijando dos objetivos en su diseño arquitectónico:

i. Crear un **lenguaje claro y natural orientado a la enseñanza** de los fundamentos de la programación de computadores. Por ello se estructuran los módulos como funciones y procedimientos.

ii. Definir un lenguaje que **permita realizar programas lo más eficientes posibles**. El tipado de datos es explícito.

Pascal recibe su nombre en honor al matemático frances Blaise Pascal (ver Anexo **??**).

3.2. Influencias del Lenguaje Pascal

3.2.1. Fortran (The IBM Mathematical Formula Translating System)

Fortran, inicialmente conocido como **FORTRAN es el acrónimo de *The IBM Mathematical Formula Translating System***.

Fortan se trata del primer lenguaje de alto nivel. Es multipropósito y se basa en el paradigma de la programación estructurada.

Su origen tiene que ver con la necesidad de crear aplicaciones científicas de manera más sencilla y lógica para el entendimiento humano.

The FORTRAN language is intended to be capable of expressing any problem of numerical computation. In particular, it deals easily with problems containing large sets of formulae and

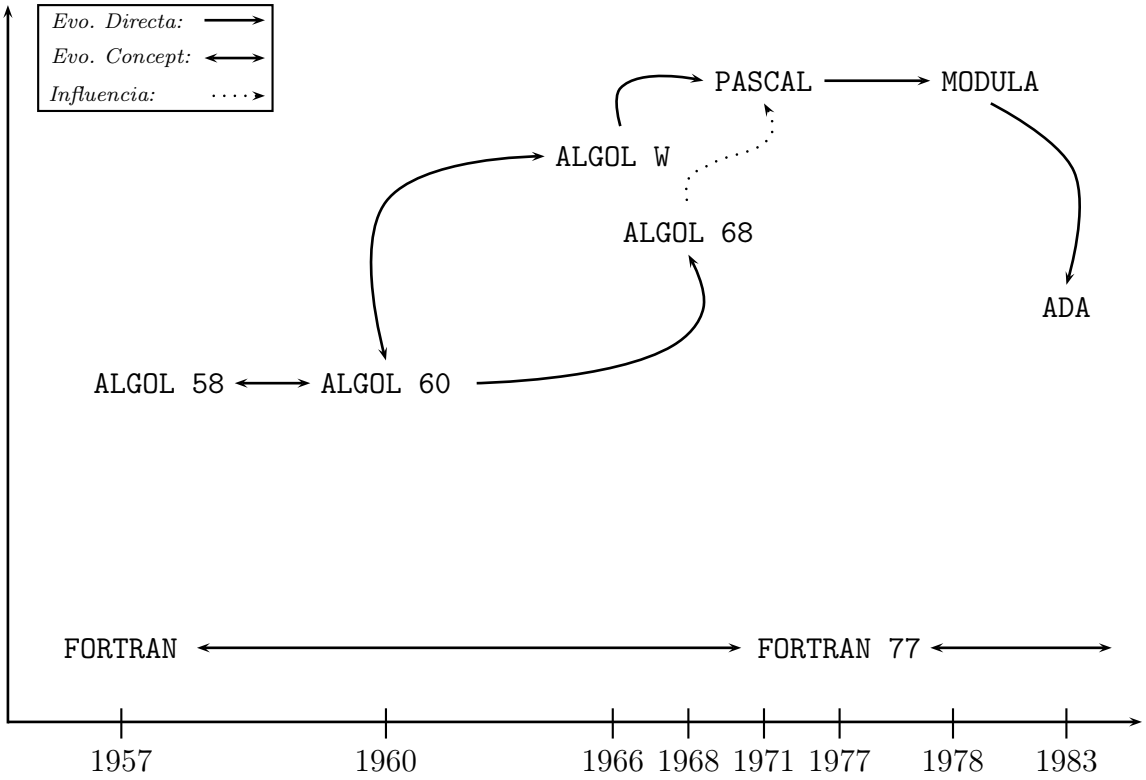


Figura 3.1: Relaciones entre los primeros lenguajes de programación.

many variables, and it permits any variable to have up to three independent subscripts. However, for problems in which machine words have a logical rather than a numerical meaning it is less satisfactory, and it may fail entirely to express some such problems. Nevertheless, many logical operations not directly expressable in the FORTRAN language can be obtained by making use of provisions for incorporating library routines. [3] [Sta66]

El primer proyecto de compilador de FORTRAN fue un `Milestone` que ocupaba `15KB` aproximadamente. Era muy rudimentario y funcionaba con rutinas muy primitivas de los SSOO de la época, prácticamente era código ensamblador.

El compilador oficial de FORTRAN fue escrito entre 1954 y 1957 a cargo de John W. Backus y grandes programadores como: Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Richard Goldberg, Lois Haibt and David Sayre. La primera ejecución del compilador se realizó sobre una máquina `IBM 704`.

Su primeros programas fueron para control energético de reactores nucleares. Demostraba ser mucho más rápido que otras soluciones tradicionales sobre Lenguaje Ensamblador.

El Lenguaje Fortran ha sido parte de seis estandarizaciones:

I. FORTRAN o FORTRAN 66: La característica más destacada es la separación de las fases de compilación, además de la posibilidad de enlazar con rutinas de lenguaje ensamblador.

II. FORTRAN 77: Entre sus características destacan:

   i. Bucles `DO` con variable índice de incremento y decremento.

   ii. Bloque de secuencias: `{IF...THEN...ELSE...ENDIF.}`

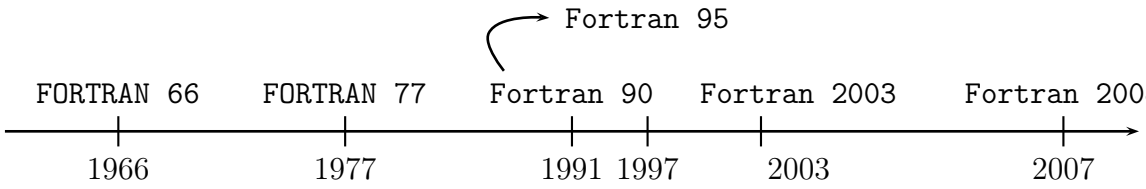


Figura 3.2: Evolución del Lenguaje Fortran.

iii. Pruebas antes de compilación de bucles `{DO}`.

iv. Tipo de dato `CHARACTER`.

v. El símbolo apostrofe (`'`) como delimitador de conjuntos de carateres.

vi. Final de un programa sin necesidad de usar la palabra `{STOP}`.

III. Fortran 90: Sus principales novedades son:

i. Nuevas estructuras de flujo: `{CASE & DO WHILE}`.

ii. Estructuras de datos tipo `RECORD`.

iii. Mejora en el manejo de `ARRAY` (nuevos operadores).

iv. Memoria dinámica.

v. Sobrecarga de operadores.

vi. Paso de argumentos por referencia.

vii. Control de precisión y rango.

viii. Módulos (paquetes de código).

IV. Fortran 95:

i. Construcciones `{FORALL}`.

ii. Procedimento `PURE` y `ELEMENTAL`.

iii. Mejoras en la inicialización de objetos.

iv. Sentencia `{DO}` para tipos de datos: `REAL` y `DOUBLE PRECISION`.

v. Sentencia `{END IF}` para terminar bloque.

vi. Sentencia `{PAUSE}`.

vii. Incorporación de `ISO/IEC 1539-1:1997` que incluye dos tipos de modulos opcionales:

a. `STRINGS` dinámicos `ISO/IEC 1539-2:2000`.

b. Compilación condicional `ISO/IEC 1539-3:1998`.

V. Fortran 2003:

i. Soporte de Programación Orientada a Objetos: Extensión de tipos, Polimorfismo y completo soporte para TADS (Tipos Abstracos de Datos) entre otras características.

ii. Mejora en la manipulación de memoria: Valores por referencia, atributo `VOLATILE`, especificación explícita de constructores para `ARRAY` y sentencia `{POINTER}`.

iii. Mejoras en Entrada/Salida: Transferencia asíncrona, acceso por flujo `(STEAM)`, especificación de operaciones de transferencia, sentencias de control y de redondeo para conversiones y sentencia `{FLUSH}`.

iv. Soporte para aritética flotante de `IEEE`.

v. Interoperabilidad con el Lenguaje de Programación C.

vi. Internacionalización `ISO 1064`.

vii. Mejora en la integración son SSOO anfitrión: Acceso a línea de comandos, variables de sistema, procesos y mensajes de error.

VI. Fortran 2008:

i. Submodulos `ISO/IEC TR 19767:2005`.

ii. Modelos de `ARRAY` para ejecución en paralelo.

iii. Contrucción `{DO CONCURRENT}`.

iv. Atributo `CONTIGUOUS`.

v. Contrucciones de tipo `BLOCK`.

vi. Componentes recursivos dinámicos.

3.2.1.1. Análisis de Fortran

Nota: Basado en: `Fortran ISO 2003`.

I. Alfabeto:

i. 26 letras[4]: `‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’| ‘p’| ‘q’ | ‘r’ | ‘s’ | ‘t’| ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’`

ii. 10 dígitos: `‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’`

iii. Carácter *Underscore*: `’_’`

iv. Símbolos especiales: ver Figura (3.3).

v. Otros símbolos: Dichos símbolos pueden ser representables pero sólamente aparecen en: comentarios, caracteres de constantes, registros de entrada/salida y descripciones.

II. Gramática: Ver bibliografía capitular [ISO04].

Programa 3.2.1. `helloProgrammer.f95`

```
1 PROGRAM HELLOPROGRAMMER
2         WRITE (*,*) "Hello programmer!"
3 END PROGRAM
```

Carácter	Nombre	Carácter	Nombre
	Blank	;	Semicolon
=	Equals	!	Exclamation point
+	Plus	"	Quotation mark or quote
-	Minus	%	Percent
*	Asterisk	&	Ampersand
/	Slash	~	Tilde
\	Backslash	<	Less than
(	Left parenthesis	>	Greater than
)	Right parenthesis	?	Question mark
[	Left square bracket	’	Apostrophe
]	Right square bracket	‘	Grave accent
{	Left curly bracket	^	Circumflex accent
}	Right curly bracket	|	Vertical line
,	Comma	$	Currency symbol
.	Decimal point or period	#	Number sign
:	Colon	@	Commercial at

Figura 3.3: Símbolos especiales de `Fortran 2003`.

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.f95` sobre GNU, usaremos el compilador `GNU Fortran`[5].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$ gfortran -o helloProgrammer helloProgrammer.f95
$ ./helloProgrammer
 Hello programmer!
```

3.2.2. ALGOL (ALGOrithmic Language)

3.2.2.1. Definiciones

Definición 3.2.2. ALGOL inicialmente recibio el nombre de IAL *Internationa Alogrithmic Language*.

Definición 3.2.3. ALGOL se trata de una familia de Lenguajes de Programación basados todos ellos en la primera versión del Lenguaje base ALGOL 58.

Definición 3.2.4. Diseñado entre 1957 y 1960 por un comité de científicos europeos y americanos que se basaban en dos ideas principales:

i. Mejorar las deficiencias estruturales de FORTRAN (todavía sin estandar pero ampliamente usado).

ii. Crear un lenguaje altamente expresivo que sea capaz de dar una respuesta común a todos los científicos.

Corolario 3.2.5. *Unos de sus notales avances fue el de limitar unidades de código (sentencias) en bloques* `{BEGIN...END.}`

3.2.2.2. Historia

The purpose of the algorithmic language is to describe computational processes. The basic concept used for the description of calculating rules is the well known arithmetic expression containing as constituents numbers, variables, and func- tions. From such expressions are compounded, by applying rules of arithmetic composition, self-contained units of the language–explicit formulae–called assignment statements. [6] [ea60]

Como hemos dicho anteriormente, ALGOL tiene su primera espcificación formal en el año 1958. Este este documento base (ALGOL 58) fue oficialmente presentado en tres formatos:

i. Sitaxis de refencia:

ii. Sintaxis de de publicaciones:

iii. Sintaxis de implementación:

ALGOL desde su comienzo tuvo un importante nicho entre científico europeos y americanos. De igual manera, ALGOL introduce en su especificación formal la notación **Backus-Naur Form**[7] que ha sido utilizada desde entronces como método descriptivo de los Lenguajes de Programación. También es notable el hecho de que ALGOL es el primer lenguajer que combina el flujo imperativo con *Lambda-Calculus*.

La primera versión estandarizada de ALGOL es ALGOL 58 que finalmente fue mejorado y actualizado con la nueva versión ALGOL 60.

ALGOL 60 es un uno de los estándares más usados y marco de referencia básica para la creación y especificación de otros lenguajes. Ha sido por ello, base de lenguajes tan importantes como: BCPC, B, Pascal, Simula o C.

El problema que intentó soclucionar esta versión fue la de hacer de ALGOL un lenguaje con aspiraciones comerciales. Por ello, se trabajó intensamente en menjorar la Entrada/Salida y la relación con el entorno de ejecución (SSOO).

Partiendo de este estandar conceptual y muy avanzado surgieron dos nuevas propuestas:

i. ALGOL 68: Sobre ALGOL 68 destacar que añade gran cantidad de utilidades que eran comunmente utilizadas por programadores de la época, entre ellas: declaración de tipos, estructuras de unión y modelos de variables por referencia.

   La especificación de esta nueva revisión adoptó la notación de **Adriann van Wijgaarden**[8], que usaba gramáticas libres de contexto para generar infinitos conjuntos de producción.

ii. ALGOL W: ALGOL W fue un proyecto encargado al profesor Nicklaus Wirth que tras la publicación de ALGOL 68 y las enormes quejas que despertó, trataba de actualizas ALGOL 60 intentando conservar la esencia y cultura del lenguaje.

3.2.2.3. ALGOL 58

Análisis

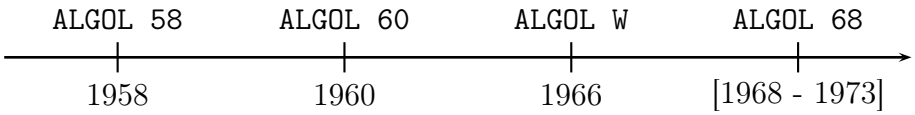


Figura 3.4: Evolución del Lenguaje ALGOL.

Nota:

I. Alfabeto:

i. asdfg

II. Gramática: Ver bibliografía capitular .

3.2.2.4. ALGOL 60

Análisis

Nota:

I. Alfabeto:

i. Letras:

```
<letter> ::= a | b | c | d | e | f | g | h | i | j | k | l |
        m | n | o | p | q | r | s | t | u | v | w | x | y | z | A |
        B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
        Q | R | S | T | U | V | W | X | Y | Z
```

ii. 10 dígitos: `<digit>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

iii. Valores lógicos: `<logical value>::= true | false`

iv. Delimitadores:

```
<delimiter> ::= <operator> | <separator> | <bracket> |
        <declarator> | <specificator>

<operator> ::= <arithmetic operator> | <relational operator> |
        <logical operator> | <sequential operator>

<arithmetic operator> ::= + | - | TIMES | / | ÷ | POWER

<relational operator> ::= < | NOTGREATER | = | NOTLESS | > | NOTEQUAL

<logical operator> ::= EQUIVALENCE | IMPLICATION | OR | AND | ¬
```

```
<sequential operator> ::= goto | if | then |
        else | for | do (2)

<separator> ::= , | . | 10 | : | ; | := | BLANK | step |
        until | while | comment

<bracket> ::= ( | ) | [ | ] | ‘ | ’ | begin | end

<declarator> ::= own | Boolean | integer |
        real | array | switch |
        procedure

<specificator> ::= string | label |
        value
```

II. Gramática: Ver bibliografía capitular .

Programa 3.2.6. `helloProgrammer.a60`

```
1 BEGIN
2    FILE F (KIND=REMOTE);
3    EBCDIC ARRAY E[0:16];
4    REPLACE E BY "HELLOW PROGRAMMER!";
5    WRITE(F,*,E);
6 END.
```

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.a60` sobre GNU, usaremos el compilador `a60c`[9].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
Hello Programmer!
```

3.2.2.5. ALGOL 68

Análisis

Nota:

I. Alfabeto:

   i. asdfg

II. Gramática: Ver bibliografía capitular .

Programa 3.2.7. `helloProgrammer.a68`

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.a68` sobre GNU, usaremos el compilador `Algol 68 Genie`[10].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$ a68g  --compile helloProgrammer.a68
$ ./helloProgrammer.sh
Hello Programmer!
```

3.2.2.6. ALGOL W

`ALGOL W`, inicialmente denominado `ALGOL X`, fue desarrollado por Nicklaus Wirth y C.A.R como sucesor directo de `ALGOL 60` a propuesta en `IFIP Working Group`. La especificación del lenguaje se vió insuficiente y fue finalmente publicada como: *"A contribution to the development of ALGOL"*.

Dicha especificacion incluía mejoras que en ningún momento querían romper la armonía original de `ALGOL 60`. Entre estas mejoras destacan:

1. Tipo de datos `STRING`.
2. Incorporación de Números Complejos ?.
3. Llamada por referencia de tipos de datos `RECORD`.
4. Sentencia `WHILE`.

3.3. El Lenguaje Pascal

Pascal se sustenta sobre dos poderosos lenguajes del ámbito científico: FORTRAN y ALGOL, de los que anteriormente hemos hablado.

Lo que trataba de hacer Wirth era descender ALGOL 60 en un nuevo lenguaje de propósito mucho más general.

El primer prototipo de esta idea fue ALGOL W programado sobre una computadora `IBM 360`. Fué una versión bastante conservadora del lenguaje lo que dio fuerza a la idea de que el nuevo lenguaje que deseaba construir Wirth tenia que soportar un repertorio de mucho más amplio.

Wirth además **quería que dicho lenguaje tuviera fines educativos**, es decir, **que enseñase la cultura de "la buena programación"**. Para ello tomo como referencia a FORTRAN y al Lenguaje Ensamblador (`ASM ?`).

En 1968 cuando empezo a implementar estos hitos en el futuro lenguaje.

Había igualmente una alta competencia con las soluciones de compilación que ofrecía FORTRAN, por ello decidio que su compilador sería de una sola pasada[11] basada en el diseño "Top-Down".

El compilador se completó a finalmente a mediados de 1970.

Pascal después llego a ser un lenguaje muy propular en circulos universitarios durante la década de los ochenta y noventa del siglo XX debido principalmente a la venta de compilares muy económicos, y un IDE de proposito general que se basaba en el mismo, hablamos de Turbo Pascal.

3.3.1. Pascal ISO 7185:1990

En 1977 BSI[12] [ISO91] produjo el estándar del Lenguaje de Programación Pascal, publicado en 1979. Ese mismo año, el organismo BSI propuso que Pascal fuese parte del programa ISO. Fué aceptado con denominación `ISO/TC97/SC5/WG4`.

En los Estados Unidos de América, `IEEE` aprobo el 10978 del proyecto 770 (Pascal).

En Diciembre, `178 X3J9` convino el resultado de `SPARCH`[13] para la resolución de US TAG[14] para la `ISO Pascal`.

En Febrero de 1979, representantes de `IEEE` combinaron los proyectos `X3` y `IEEE 770` bajo el comité `X3J9/IEEE-P770 Pascal Standards (JPC)`.

La resolución de `JFC` fué avalada en `NSI/IEEE770X3 .97-1983` por ANSI bajo `American National Standard Pascal Computer Programming Language`.

Las especifiaciones de BSI se hicieron públicas en 1982, internacionalmente conocido como `Standard 7185`.

3.3.1.1. Alfabeto

I. Unidades:

i. `letter = ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’| ‘p’| ‘q’ | ‘r’ | ‘s’ | ‘t’| ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’ .`

ii. `digit = ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’ .`

II. Símbolos:

i. `special-symbol = ‘+’ | ‘-’ | ‘*’ | ‘/’ |‘=’ | ‘<’ | ‘>’ | ‘[’ | ‘]’| ‘.’ | ‘,’ | ‘:’ | ‘;’ | ’ˆ | ‘(’ | ‘)’| ‘<>’ |‘<=’ | ‘>=’ | ‘:=’ | ‘..’ | word-symbol .`

ii. `word-symbol = ‘and’ | ‘array’ | ‘begin’ | ‘case’ | ‘const’ | ‘div’|‘do’ | ‘downto’ | ‘else’ | ‘end’ | ‘file’ | ‘for’|‘function’ | ‘goto’ | ‘if’ |‘in’ | ‘label’ | ‘mod’|‘nil’ | ‘no’ | ‘of’ | ‘or’ | ‘packed’ | ‘procedure’| ‘program’ | ‘record’ | ‘repeat’ | ‘set’ | ‘then’|‘to’ | ‘type’ | ‘until’ | ‘var’ | ‘while’ | ‘with’ .`

III. Identificadores: `identifier = letter  letter | digit .`

IV. Directivas: `directive = letter  letter | digit .`

V. Números:

i. `signed-number = signed-integer | signed-real .`

ii. `signed-real = [ sign ] unsigned-real .`

iii. `signed-integer = [ sign ] unsigned-integer .`

iv. `unsigned-number = unsigned-integer | unsigned-real .`

v. `sign = ‘+’ | ‘-’ .`

vi. `unsigned-real = digit-sequence '.' fractional-part [ 'e' scale-factor | digit-sequence 'e' scale-factor .`

vii. `unsigned-integer = digit-sequence`

viii. `fractional-part = digit-sequence .`

ix. `scale-factor = [ sign ] digit-sequence .`

x. `digit-sequence = digit  digit`

VI. Etiquetas: `label = digit-sequence .`

VII. Cadenas de caracteres:

i. `character-string = '' string-element  string-element  '' .`

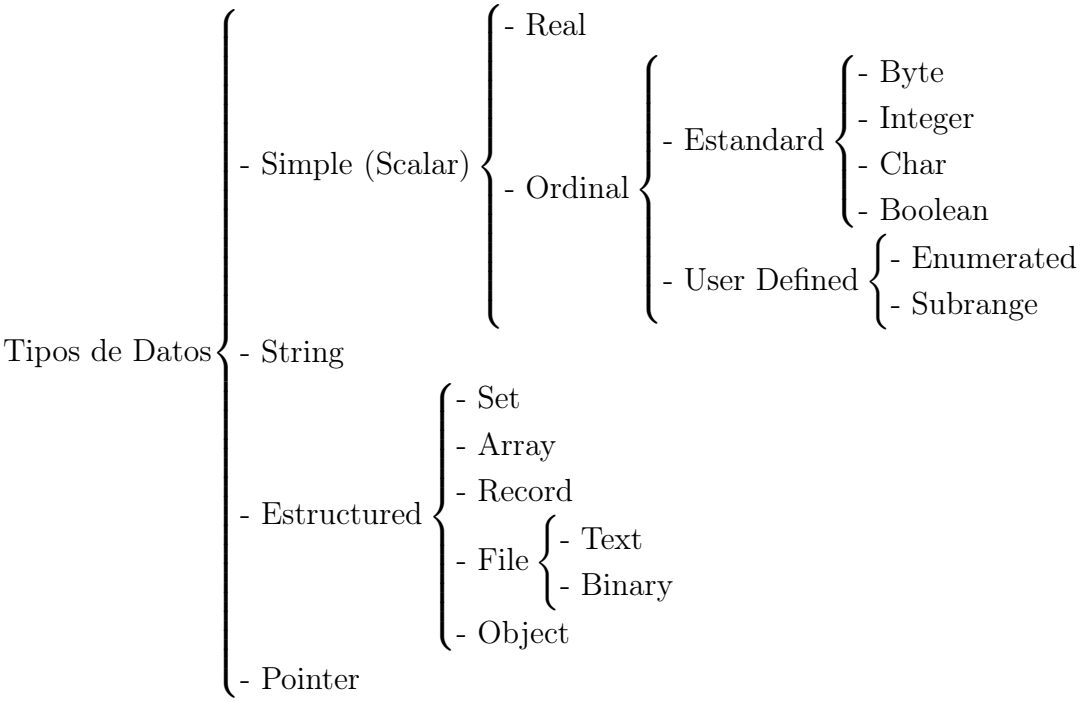
ii. `string-element = apostrophe-image | string-character .`

iii. `apostrophe-image = '' .`

iv. `string-character = one-of-a-set-of-implementation-defined-characters .`

VIII. Separadores: `( '{' | '(*' ) commentary ( '*)' | '}' )`

3.3.1.2. Tipos de datos



3.3.1.3. Biblioteca

1. Procedimientos:

i. `PROCEDURE REWRITE(`f`)` → Crea un fichero en modo escritura. En caso de existir el propio fichero es sobreescrito.

1) Precondición: `True`
2) Postcondición: `(f↑)` está indenfinido.

ii. `PROCEDURE PUT(`f`)` →

1) Precondición: `(f↑)` está denfinido.
2) Postcondición: `(f↑)` está indenfinido.

iii. `PROCEDURE RESET(`f`)` → Abre un fichero en modo lectura con el puntero de fichero sobre el comienzo del mismo.

1) Precondición: `(f↑)` está denfinido.
2) Postcondición: `(f↑)` está indenfinido.

iv. `PROCEDURE GET(`f`)` →

1) Precondición: `(f↑)` está denfinido.
2) Postcondición: `(f↑)` está indenfinido.

v. `PROCEDURE READ(`f`)` → Se encarga de leer el fichero (`var F: tipodeFichero`) y de asignar sus datos al conjunto de variables (`lista de variables`).

con la salvedad de que el (`var F: tipodeFichero`) es opcional, y en el caso de no especificarse explicitamente como parámetro se lee el fichero por defecto `input`.

Nota: `(f)` es equivalente: `begin read(ff,`v_1`); begin read(ff,`v_2`, ..., `v_n`) end`

vi. `PROCEDURE WRITE(`f`)` → Se encarga de escribir el fichero (`var F: tipodeFichero`) y de escribir en el mismo los datos de (`lista de variables`).

anterior, con la salvedad de que el (`var F: tipodeFichero`) es opcional, y en el caso de no especificarse explicitamente como parámetro se lee el fichero por defecto `input`.

Nota: `(f)` es equivalente: `begin write(ff,`v_1`); begin write(ff,`v_2`, ..., `v_n`) end`

vii. `PROCEDURE NEW(`f`)` →

viii. `PROCEDURE DISPOSE(`f`)` →

2. Funciones:

I. Funciones aritméticas:

i. `FUNCTION ABS(`x`)` → Se trata de un operador genérico para tipos de datos Entero y Real que a partir del parámetro (x:tipo) devuelve el valor absoluto de `x`.

Formalidad 3.3.1. `FUNCTION ABS(`x`:tipo): tipo;` $\equiv |x|$

ii. `FUNCTION SQR(`x`)` →Para el parámetro `x` de tipo INTEGER o REAL devuelve el valor de x^2.

Formalidad 3.3.2. `FUNCTION SQR(`x`:tipo): tipo;` $\equiv x^2$

iii. `FUNCTION SIN(`x`)` → Para el tipo de datos REAL, devuelve el valor del seno del parámetro `x`.

Formalidad 3.3.3. `FUNCTION SIN(x:REAL): REAL;` $\equiv sen(x)$

iv. `FUNCTION COS(x)` → Para el tipo de datos REAL devuelve el coseno de `x`.

Formalidad 3.3.4. `FUNCTION COS(x:REAL): REAL;` $\equiv cos(x)$

v. `FUNCTION EXP(x)` → Para el tipo de datos REAL devuelve el valor de e^x, siendo `x` el parámetro.

Formalidad 3.3.5. `FUNCTION EXP(x:REAL): REAL;` $\equiv e^x$

vi. `FUNCTION LN(x)` → Para el tipo de datos REAL devuelve Lnx, siendo `x` el parámetro.

Formalidad 3.3.6. `FUNCTION LN(x:REAL): REAL;` $\equiv Lnx$

vii. `FUNCTION SQRT(x)` → Para el parámetro `x` de tipo REAL devuelve el valor de $\sqrt{x}$.

Formalidad 3.3.7. `FUNCTION SQRT(x:REAL): REAL;` $\equiv \sqrt{x}$

viii. `FUNCTION ARCTAN(x)` → Para el tipo de dato Real devuelve el valor del arcotangente `x` en radianes.

Formalidad 3.3.8. `FUNCTION ARCTAN(x:REAL): REAL;` $\equiv arctg(x)$

II. Funciones de transferencia:

i) `FUNCTION TRUNC(x)` → Para el tipo de dato REAL, obtiene la parte entera del parámetro `x`.

Formalidad 3.3.9. `FUNCTION TRUNC(x:REAL): LONGINT;` $\equiv TRUNC(a,b) = a$

ii) `FUNCTION ROUND(x)` → Para el tipo de datos REAL, redondea el parámetro `x` al valor entero mas próximo.

III. Funciones ordinales:

i) `FUNCTION ORD(x)` → Para el tipo de datos LONGINT devuelve `true` en caso de que el parámetro `x` sea par. Siendo impar devuelve `false`.

ii) `FUNCTION CHR(x)` → Para el tipo de datos BYTE devuelve el carácter (ver tabla ASCII, Apéndice ...) del valor ordinal `x`.

iii) `FUNCTION SUCC(x)` → Para un valor ordinal devuelve el sucesor del parámetro `x`.

iv) `FUNCTION PRED(x)` → Para un valor ordinal devuelve el predecesor del parámetro `x`.

IV. Funciones booleanas:

i) `FUNCTION ODD(x)` →

ii) `FUNCTION EOF(f)` → Devuelve el valor `true` en caso de que sea final de fichero (el puntero de lectura/escritura se encuentra en el carácter de final de fichero). Caso contrario `false`.

iii) `FUNCTION EOLN(f)` → Devuelve el valor `true` en caso de que sea final de línea (el puntero de lectura/escritura se encuentra en el carácter de final de línea). Caso contrario `false`.

3.3.1.4. Estructura de un programa

Un programa en Pascal se divide en tres partes bien diferenciadas:

I. *Program Heading*:

II. *Declaration Part*:

III. *Statement Part*:

Programa 3.3.10. `Plantilla de programa en Pascal`

```
1  program {name of the program}
2  uses {comma delimited names of libraries you use}
3  const {global constant declaration block}
4  var {global variable declaration block}
5
6  function {function declarations, if any}
7  { local variables }
8  begin
9  ...
10 end;
11
12 procedure { procedure declarations, if any}
13 { local variables }
14 begin
15 ...
16 end;
17
18 begin { main program block starts}
19 ...
20 end. { the end of main program block }
```

Programa 3.3.11. `helloProgrammer.pas`

```
1 PROGRAM HelloProgrammer;
2 BEGIN
3 WRITELN('Hello Programmer!');
4 END.
```

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.pas` sobre GNU, usaremos el compilador `GNU Pascal Compiler`[15].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$ gpc -o helloProgrammer helloProgrammer.pas
$ ./helloProgrammer
Hello Programmer!
```

3.4. Evoluciones del Lenguaje Pascal

3.4.1. Modula/Modula-2

Modula-2 grew out of a practical need for a general, efficiently implementable systems programming language for minicomputers. Its ancestors are Pascal and Modula. From the latter it has inherited the name, the important module concept, and a systematic, modem syntax, from Pascal most of the rest. This includes in particular the data structures, i.e. arrays, records, variant records, sets, and pointers. Structured statements include the familiar if, case, repeat, while, for, and with statements. Their syntax is such that every structure ends with an explicit termination symbol.[16] [Wir80]

Modula/Modula-2 es un lenguaje multipropósito pensado originalmente para ser un lenguaje mucho más eficiente que Pascal. **De alguna manera Wirth ha buscado siempre la mejora en el rendimiento de todos sus lenguajes.**

Se presenta como un lenguaje basado en importantes conceptos de Programación Orientada a Objetos (POO), aunque no los implementa todos. La idea más destacada de POO en Modula/Modula-2 es la de encapsulación. Como sabemos crea una estructura de datos modular en base a unas propiedades y métodos (operaciones sobre los datos). La idea de cápsula[17] deriba de que por sí las propiedades del módulo nunca son accesibles directamente como ocurre en la programación estructurada. Por contra, son los métodos (las acciones) las que dan acceso al contenido de estas variables bien sea para su lectura, escritura o ambas.

Como decimos, este es el aspecto más destacado de Modula/Modula-2 ya que, conceptos universales como la Herencia son inexistentes dentro del lenguaje. Dado que su propósito era más general y profesional que Pascal, Modula/Modula-2 tuvo cierto auge en entornos profesionales en los años ochenta del siglo XX. Una vez más sus "limitaciones por definición" lo han convertido en un "lenguaje para aprender a programar".

3.4.1.1. Símbolos y Gramática

1. Tokens:

```
letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j'
       | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't'
       | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' .

digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
```

2. Símbolos especiales:

```
special-symbol = '+' | '-' | '*' | '/' | '=' | '<' | '>' | '[' | ']'
               | '.' | ',' | ':' | ';' | '^' | '(' | ')'
               | '<>' | '<=' | '>=' | ':=' | '..' | word-symbol .

word-symbol = 'and' | 'array' | 'begin' | 'case' | 'const' | 'div'
            |'do' | 'downto' | 'else' | 'end' | 'file' | 'for'
            |'function' | 'goto' | 'if' | 'in' | 'label' | 'mod'
```

```
                |'nil' | 'not' | 'of' | 'or' | 'packed' | 'procedure'
                |'program' | 'record' | 'repeat' | 'set' | 'then'
                |'to' | 'type' | 'until' | 'var' | 'while' | 'with' .
```

3. Gramatica: (ver Anexo **??** sección 2.)

Programa 3.4.1. `helloProgrammer.mod`

```
1 MODULE helloProgrammer;
2
3 FROM InOut IMPORT WriteString;
4
5 BEGIN
6         WriteString('Hello programmer!');
7 END helloProgrammer.
```

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.adb` sobre GNU, usaremos el compilador `GNU Modula-2`[1].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$ gm2 -o helloProgrammer helloProgrammer.mod
$ ./helloProgrammer
Hello Programmer!
```

3.4.2. Ada

La idea conceptual del lenguaje de programación **Ada nace por los enormes gastos que generaban: compiladores, editores y otras herramientas de sistemas embebidos en el Departamento de Defensa de EEUU**. En 1974 da comienzo un estudio que desbela el gran problema de tener un sistema de desarrollo muy caótico dónde eran frecuentes aplicaciones sobre un lenguaje determinado para un tipo de sistema concreto. Se llegó a la conclusión de que era necesaria una estandarización de desarrollo (lo que inlcuía un nuevo lenguaje general para estos sitemas).

En 1975 se elaboró un documento técnico (`Strawman`) con las primeras especificaciones. Dicho documento sufrió distintas modificaciones gracias a la participación y comentarios de muchos desarrolladores. En el año 1976 se tenía una versión muy robusta del lenguaje que querían a nivel conceptual. Se hizó un concurso público para que disitas empresas dieran una forma Software a dichas especificaciones. Finalmente y tras un duro proceso de selección en 1979 se público ganadora la empresa *CII Honeywell Bull*. Al mismo tiempo se dio nombre al lenguaje que empezaba a ser una realidad. **Se denomino Ada en honor a la primera programadora de la historia, Augusta Ada King (Ada Loverlace)**[18], asistente y mecenas de Charles Babbage a su vez, creador de la primera máquina analítica.

En 1980 se publica la versión definitva del lenguaje y es propuesta para su estadarización en ANSI. **En 1893 se tuvo la primera versión de Ada estandar (`ANSI/MIL-STD`**

[1] http://www.nongnu.org/gm2/homepage.html

1815A) **conocido como Ada 83**. El modelo se perfecciono y por fin fue publicado por ISO (ISO-8652:1987).

El mismo Ada 83 desde el principio tuvo deficiencias prácticas por lo que se trabajo en una nueva versión (Ada 9X) que entre otras ideas, incorporaba el mecanismo de herencia. La nueva versión se llamó Ada 95 (ISO-8652:1995) y es uno de los estándares más usados hoy en día.

Ya en el año 2012, en el Congreso Ada-Europe celebrado en Stockholm, los organismos: Ada Resource Association (ARA) y Ada-Europe anunciaron el diseño de una nueva versión pendiente actualmente de aprobación por parte de ISO/IEC.

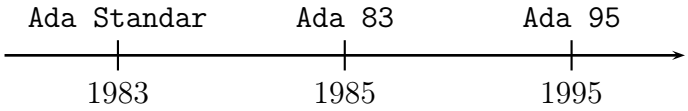


Figura 3.5: Evolución del Lenguaje Ada.

El lenguaje de programación Ada destaca sobre otros por lo siguiente:

i. Legibilidad: Ada en mayor medida que Pacal insiste en que los programas deben ser legibles(cualquier programador puede ser capaz de comprender el codigo fuente de un programa). Por ejemplo, un valor en aritmética flotante es expresado como: `(PART INT).(PART DECIMAL)`.

ii. Tipificación fuerte: Es necesario y sabido que cualquier estado de datos (variable o propiedad) debe estar perfectamente tipificado es decir, desde el principio se debe aclarar a que familia de datos pertenece

iii. Programación en gran escala: Es parte del concepto de ADA es ser un lenguaje que atienda a necesidades de computación masiva. Por ello se utilizan técnicas de encapsuláción, unidades lógicas, parrametrización de procesos con el objetivo final de que el programa sea siempre un conjuto de partes que se puedan auditar individualmente.

iv. Manejo de escepciones: Las expceciones son imprescindibles para este tipo de lenguajes que puesto que trabajan muy cercanos al Hardware. ADA es un lenguaje ampliamante usado aplicaciones de alto redimiento y es por esto, que hace muy necesario conocer como se comporta el programa en tiempo de ejecución.

v. Unidades genéricas: Lo hemos comentado anteriormente y es que, la lógica de programación nos dice *Divide and Conquer* (D&C)[2] Tiende siempre a unidades lógicas (cápsulas) y funcionales. Con esto ADA siempre ha pretendido que los errores se corrijan desde lo más básico y procurando que afecte lo menos posible al resto de módulos (Alta coherencia y baja cohesión ?).

Programa 3.4.2. `helloProgrammer.adb`

[2]Divide y Venceras.

```
1 with Ada.Text_IO; use Ada.Text_IO;
2 procedure helloProgrammer is
3 begin
4   Put_Line ("Hello programmer!");
5 end helloProgrammer;
```

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.adb` sobre GNU, usaremos el compilador `GNAT (GNU NYU Ada Translator)`[3].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$ gnatmake helloprogrammer.adb
$ ./helloprogrammer
Hello programmer!\begin{verbatim}
```

3.4.3. Oberon

Oberon is a general-purpose programming language that evolved from Modula-2. Its principal new feature is the concept of type extension. It permits the construction of new data types on the basis of existing ones and to relate them.[19] [Wir88]

Oberon es un lenguaje de programación orientado a objetos y procedimental creado por Niklaus Wirth (autor también de Pascal, Modula y Modula-2) y sus colaboradores del ETHZ (Suiza).

Oberon puede considerarse una evolución de Modula-2 con un soporte completo de orientación a objetos. De este lenguaje y de sus antecesores hereda buena parte de la sintaxis y de la filosofía. Wirth siempre ha intentado simplificar los lenguajes sin que por ello se pierda en potencia. **También está diseñado con la seguridad en mente**: tiene chequeos de rango en arrays, recolector de basura y es fuertemente tipado. Sin embargo, por su intento de simplicidad carece de enumeraciones y enteros restringidos en rango, los cuales pueden implementarse como objetos.

La sintaxis de orientación a objetos de Oberon no se parece a la de otros lenguajes más populares como C++ o Java, pero sí guarda similitud con la de Ada 95.

Oberon es también el nombre de un sistema operativo,escrito con y para este lenguaje. Oberon se ha portado a otros sistemas (incluyendo a Windows y sistemas Unix) e incluso se puede compilar en `bytecodes` para la máquina virtual de Java. También existe un proyecto para crear un compilador para la plataforma .NET.

3.4.3.1. Símbolos y Gramática

1. Tokens:

2. Símbolos especiales:

3. Gramatica: (ver Anexo **??** sección 3.)

[3] http://www.adacore.com/

Programa 3.4.3. `helloProgrammer.mod`

```
1  MODULE Hello;
2            IMPORT Oberon, Texts;
3    VAR W: Texts.Writer;
4
5    PROCEDURE World*;
6    BEGIN
7      Texts.WriteString(W, "Hello World!");
8      Texts.WriteLn(W);
9      Texts.Append(Oberon.Log, W.buf);
10   END World;
11
12 BEGIN
13   Texts.OpenWriter(W);
14 END Hello.
```

Notas sobre compilación: Para compilar el archivo fuente `helloProgrammer.adb` sobre GNU, usaremos el compilador `Oberon for GNU/Linux`[4].

Las ordenes para compilarlo y ejecutarlo son las siguientes:

```
$  helloprogrammer.mod
$ ./helloprogrammer
Hello programmer!
```

[4]http://olymp.idle.at/tanis/oberon.linux.html

Notas del capítulo

[1] *El Lenguaje de Programación Pascal es descrito a partir del desarrollo de una versión de Algol 60. Comparado con Algol 60, el rango de aplicación es considerablemente mayor gracias a la variedad de estructura de datos. En principio es un intento para fundamentar las bases para enseñar programación con una herramienta eficiente para escribir grandes programas enfatizando en usar conceptos razonable sencillos, sistematicos a la programación structurada, y la eficiencia de la implementación. El compilador de es una sola pasada y ha sido construido para la familia CDC 6000.*

[2] El profesor **Niklaus Wirth** nació en Winterthur Suiza el 15 de febrero de 1934.

Se gradua en 1959 como Ingeniero en Electrónica por la Escuela Politécnica Federal de Zúrich (ETH) en Suiza. Un año más tarde, se doctora (Ph.D.) en la Universidad de Berkley, California.

Trabajó a mediados de la decada de los sesenta del siglo XX en la Universidad de Stanford y en la Universidad de Zúrich. Finalmente en 1968 se convierte en profesor de Informática en la ETH en Suiza.

[3] *El lenguaje de programación FORTRAN intenta hacer posible la expresión de cualquier problema numérico. En particular, es ideal para formular conjuntos de múltiples variables, permitiendo que cualquier variable sea tratada desde un plano libre de contexto. Sin embargo, este tipo de ejercicios persenta inconsistencias con el repertorio de palabras de cada máquina de computo y su lógica numérica, lo que puede llevar a problemas a la hora de expresar algunos problemas numéricos. Mucha de la lógica que emplea FORTRAN no es directamente expresable por lo que se puede obtener incorporando bibliotecas.*

[4] La relación entre mayúsculas (upper-case) y minúsculas (lower-case) se delega en el fabricantes del compilador.

[5] http://gcc.gnu.org/fortran/

[6] *El propósito de ALGOrithmic Language es la de describir procesos computacionales. El concepto básico usado en la descripción de reglas de cálculo es conocido expresiones aritméticas constituidas por: números, variables y funciones. Las expresiones son compuestas aplicando reglas de composición aritmética, constituyendo unidades del lenguaje, explicitamente formuladas, llamadas asignación de recursos.*

[7] Backus-Naur Form...

[8] **Adriann van Wijgaarden**...

[9] webSite

[10] jmvdveer.home.xs4all.nl

[11] Single-Pass.

[12] *British Standards Institution*: http://www.bsigroup.com/

[13] *Standards Planning and Requirements Committee*

[14] *Technical Advisory Group*: http://technicaladvisorygroup.com/

[15] http://www.gnu-pascal.de/gpc/

[16] Modula-2 en un lenguaje de programación de propósito general, eficientemente implementado para para minicomputadoras. Sus antecesores son Pascal y Modula. Suscrito a su nombre está el concepto de módulo y el trato sistemático con una sintaxis moderna, por ejemplo: matrices, registros, registros variables, conjuntos y punteros. Sus estructuras incluye los familiares: if, case, repeat, while y símbolos de terminación explicita.

[17] El concepto de cápsula como unidad estrutural se de debe a David Lorge Parnas...

[18] Augusta Ada King (1815-1852) hija del poeta Lord Byron...

[19] Oberon is un lenguaje de propósito general envuelto sobre Modula-2. Es principalmente nueva la característica de extensión de tipo.- Esto permite la construcción de nuevos tipos de datos basados en otros existentes.

Bibliografía

[ea60] J. W. Backus et al. Report on the algorithmic language ALGOL 60. *Numerische Mathematik Volume 2, Number 1, 106-136, DOI: 10.1007/BF01386216*, 1960.

[ISO91] ISO/IEC. Pascal ISO 90 (ISO/IEC 7185:1990). *ISO/IEC*, 1991.

[ISO04] ISO/IEC. Fortran ISO 2003 (ISO/IEC DIS 1539-1:2004). *ISO/IEC*, 2004.

[Sta66] American National Standar. FORTRAN 66 (USA Standar FORTRAN). *American National Standar*, March 7, 1966.

[Wir71] Nickaus Wirth. The Programming Language Pascal. *Acta Informatica Volume 1, Number 1, 35-63, DOI: 10.1007/BF00264291*, 1971.

[Wir80] Nickaus Wirth. Modula-2. *Institut für Informatik ETH Zürich Technical Report 36*, 1980.

[Wir88] Nickaus Wirth. *The programming language Oberon.* Software: Practice and Experience Volume 18, Issue 7, pages 671–690, July 1988.

Índice alfabético