APL - A Programing Language

Roldán Galán Lafarga
NIP:539891
rolgalan@gmail.com

Asignatura de Lenguajes de programación.
Profesor: Luis Montesano

Índice

Contenido

APL

- APL es un lenguaje de programación funcional orientado a arrays, por lo que también se le conoce como *Array Processing Language*.
- Es el lenguaje más expresivo que existe, implementando en una línea lo que en otros lenguajes requiere una docena.
- Se caracteriza por su conjunto de caracteres no incluidos en ASCII, que lo hacen más similar a las matemáticas que a la programación.

Historia

- En 1957, Kenneth E. Iverson diseñó una notación matemática potente y concisa con la que estudiar y representar algoritmos y funciones relacionados con la informática.
- En 1962 la publicó en un libro titulado A programming Language, y tres años más tarde se realizó la primera implementación (parcial) en una computadora.
- Finalmente, en 1967 se implementó completo para el IBM 1130.
- El gigante azul ha sido el principal impulsor de este lenguaje durante su historia (especialmente en mainframes al inicio), terminando con el desarrollo del APL2 en los 80, que todavía comercializa.

Contenido

Funciones y operadores

- Se compone de datos, funciones y operadores: las funciones modifican a los datos y los operadores a las funciones.
- Tiene un amplio conjunto de funciones primitivas (más de 100), y gracias a los operadores se consigue una increible expresividad sin necesidad de codificar demasiadas líneas.
- El usuario también puede definir nuevas funciones como en otros lenguajes, así como también operadores.

Codificación

- Es principalmente un lenguaje interpretado, aunque existe la opción de compilar los programas.
- Al conseguir codificar tantas instrucciones en tan pocos comandos, a veces es utilizado para el diseño de prototipos.
- Por su apariencia "jeroglica"suele tachársele de *lenguaje de ślolo escritura*.
- Por contra, al disponer de poderosas funciones, el código resultante es más reducido y resulta mucho más sencillo de mantener y actualizar.

Contenido

Orden de ejecución

- Todas las funciones se ejecutan de acuerdo a su posición en la expresión.
- La función más a la derecha cuyos argumentos estén disponibles, se ejecuta primero.
- No hay niveles de precedencia ni jerarquías entre las funciones.
- Entonces, 2x5+4, es **18**, y no 14.

Niveles de precedencia

- No existe una jerarquía porque sería muy complejo tratar de establacer un orden entre todas las funciones primitivas del sistema. Además, las funciones que crea el usuario deben comportarse del mismo modo que las primitivas, lo cual sería imposible estableciendo jerarquías predefinidas.
- Para escapar a esto, se permite el uso de paréntesis con el comportamiento habitual, resolviéndose sus funciones interiores antes de ejecutar el resto.

Nombres

- Los nombres son las representaciones simbólicas de los objetos de APL, es decir, arrays, funciones y operadores.
- Los nombres **primitivos** están asociados siempre al mismo objeto y forman parte de la definición del lenguaje.
- Los nombres **construidos** pueden estar asociados a diferentes objetos cada vez.
- Para ser válidos han de comenzar por una letra, y el resto serán caracteres alfanuméricos, incluído el guión bajo.
- Para asociar un nombre a un objeto se hace mediante $\leftarrow$

Nombres distinguidos

También existen los nombres distinguidos, los cuales vienen precedidos del carácter □ y están reservados para configuración del lenguaje.
Si corresponden a un array, se llaman variables del sistema; los asociados a funciones son llamados funciones del sistema. Algunos son:

- □IO *Index Origin*. Define si los vectores se indexan desde 0 ó 1.
- □PP *Print precision*. Establece el número de decimales que se imprimirán por pantalla.

Símbolos de construcción sintáctica I

Algunos símbolos no están asociados a ningún objeto, teniendo usos diversos:

- [] A la derecha de un array, indica indexado. A la derecha de una función, indica el eje sobre el que actúa.
- → Seguida de una expresión, indica la siguiente línea a ser ejecutada. Sola, limpia el indicador de estado de una operación suspendida y su llamada.
  - ◦ Parámetro de sustitución para el operando izquierdo de un producto exterior.
- ( ) Utilizado para agrupar. Las expresiones entre paréntesis son evaluadas primero.

Símbolos de construcción sintáctica II

' Delimita un string de caracteres

; Entre corchetes, separa los índices a lo largo de cada eje. Al comienzo de una operación definida, separa la lista de nombres locales del resto y de la sintaxis de la operación.

◊ Separa múltiples expresiones que aparecen en una sola línea.

← Asocia un nombre con un objeto, o modifica los valores de una posición seleccionada en un array.

Expresiones

Una expresión consiste en un nombre (primitivo o construído) y, opcionalmente, uno o más símbolos de construcción sintáctica. Pueden ser:

Expresiones de array Son aquellas que contienen una función cuyos argumentos son evaluados, siendo el resultado un array.

Expresión de función Cuando la expresión tiene sólo una función, sin argumentos.

Expresión de operador Si es sólo un operador sin operandos.

Las dos últimas sólo pueden ser evaluadas dentro de una expresión de array.

Declaraciones

- Son una unidad de trabajo ejecutable.
- Se compone de tres partes. Cualquiera de ellas puede ser omitida, pero si se incluye, debe aparecer en el orden mostrado a continuación.
- etiqueta : expresiones (upshoejot) comentario
- Las **expresiones** pueden ser ninguna, una o varias (separadas por $\Diamond$).

Expresiones sintácticamente válidas I

Notación de vectores (más adelante)

Colocación de operaciones

- Una función u operador diádico es escrito entre sus argumentos.
- Una función monádica es escrita a la izquierda de su argumento.
- Un operador monádico es escrito a la derecha de su operando.

Expresiones sintácticamente válidas II

Símbolos de construcción sintáctica

- Paréntesis, comillas y corchetes deben emparejarse.
- Los paréntesis se pueden colocar alrededor de expresiones de array, función y operadores. No deben cortar un nombre o un grupo de funciones u operadores.
- Los dos puntos sólo están permitidos siguiendo a una etiqueta que está totalmente a la izquierda de una línea.

Expresiones sintácticamente válidas III

Símbolos de construcción sintáctica

- La expresión a la derecha de $\leftarrow$ ha de ser de array. El objeto de la izquierda puede ser el nombre de un array, un nombre no asociado a ningún objeto, una lista de nombres o una expresión que selecciona una posición de un array.
- Una $\rightarrow$ ha de estar a la izquierda del todo, o justo a la derecha de una etiqueta. Cualquier expresión a la derecha de la flecha debe ser una expresión de array.
- El punto y coma sólo está permitido dentro de corchetes.

Expresiones sintácticamente válidas IV

Espacios Son necesarios para separar nombres construídos de otros símbolos.

Espacios y paréntesis redundantes Están permitidos, y a menudo son empleados para hacer una expresión más legible. No cambian el significado de una expresión y no producen errores.

Contenido

Arrays

APL está orientado a trabajar con colecciones de números o caracteres, estas colecciones se denominan arrays, los cuales tienen dos propiedades: estructura y datos.

- La primera define características del array utilizado.
- Los datos son la información en sí, los cuales se consideran escalares (o más arrays).

Arrays: estructura

Rango Mide la dimensión: 0 para un escalar, 1 para un vector, 2 para una matriz...

Forma Es la cantidad de elementos en cada dimensión.

Profundidad Indica la cantidad de arrays anidados que componen un dato, incluyendo él mismo. Un escalar tiene profundidad cero, mientras que un vector tendría profundidad 1. Un vector en el que al menos uno de sus elementos sea otro vector, tendrá profundidad 2. También puede interpretarse que un array con profundidad n tiene al menos un array con profundidad n-1 entre sus elementos.

Arrays: datos

- Los datos son numéricos o cadenas de texto.
- Todos los números introducidos o mostrados son en decimal o en formato científico: 2,56E3, representaría 2563.
- Los booleanos se incluyen en este grupo, utilizando 1 y 0.
- También soporta números complejos, expresados de la forma: 2J3, siendo $2 + 3i$.
- Pueden definirse en su forma polar, tanto en grados como en radianes, mediante 2R3.1 y 2D90 respectivamente.
- Los strings son introducidos entre comillas simples.

Arrays: notación de vectores

Una sucesión de valores separados únicamente por espacios entre sí (o paréntesis), es tratado como un vector.

Construcciones válidas

```
1 2 3 4
10 (15+5) 30
'a' 'b' 'c'
4 5 (1 2) 6
A<-1 2
4 5 A 6
```

Obsérvese que las dos últimas líneas y la anterior consiguen el mismo resultado.

Contenido

Tipos de funciones por número de parámetros

Se dividen en monádicas y diádicas y se representan e invocan mediante símbolos.

Monádicas Un único parámetro, siempre a la derecha del símbolo.

Diádicas Dos parámetros, uno a cada lado de la función.

Niádicas El usuario puede definir funciones que se ejecuten sin argumentos (no existen funciones primitivas con estas características).

La mayoría de símbolos representan distintas funciones si se utilizan con uno o dos parámetros, aunque generalmente tienen características similares.

Símbolos con distinta función según parámetros

-

-A Negativo de A.

B-C B menos C.

!

!A Devuelve el factorial de A.

B!C Total de combinaciones de B elementos que pueden realizarse sobre C elementos $\binom{B}{C}$

⍟

⍟A Logaritmo neperiano de A.

B ⍟ C Logaritmo en base B de C.

Determinar la función

- Puesto que la mayoría pueden ser de uno o dos operadores, hay que establecer una regla clara para diferenciarlas a la hora de escribir un código.
- Si el objeto a la izquierda es un array, entonces la función es diádica.
- Si el objeto a la izquierda es un operador, u otra función, entonces es monádica.

Algunas funciones son sólo monádicas o sólo diádicas. Si se escriben con número incorrecto de argumentos, se lanza un *VALENCE ERROR*, y no un *SYNTAX ERROR*, puesto que es sintácticamente correcto escribir una función sin argumentos.

Tipos de funciones según operación

Escalares Se aplican a los escalares contenidos en un array de cualquier estructura, produciendo un resultado con la misma estructura.

Estructurales Se aplican sobre la estructura que contiene los datos, sin modificar a estos últimos.

Información Devuelven información acerca de los arrays o su contenido, pero sin devolver el dato en sí mismo.

Transformación de no escalares Transforman datos en ciertas estructuras, o devuelven estructuras de datos diferentes a las de los argumentos.

Ejemplos de funciones por operación I

De Escalares

$\lceil R$ Función techo a cada elemento de R.

```
⌈1.3 6.8
2 7
```

$L\,b\,R$ Elementos de L elevados a correspondientes de R.

```
(3 4)b(2 2)
9 16
```

Estructurales

ϵR Genera un vector simple con los escalares de R.

```
ϵ1 2 (30 40 50)
6
1 2 3 4 5 6
```

$L\rho R$ Redimensiona el vector R en las dimensiones de L

```
(2 2)ρ(1 2 3 4)
1 2
3 4
```

Ejemplos de funciones por operación II

De Información

ρR Devuelve los elementos en cada eje de R

```
1 2 3
4 5 6
```

ρR: 2 3

LιR La posición de la primera aparición en L de cada elemento de R.

```
(1 2 3 4)ι(6 3 1 2)
5 3 1 2
```

de Transformación

ιR Todos los enteros de 1 a R.

```
ι7
1 2 3 4 5 6 7
```

L?R L enteros aleatorios del vector ιR.

```
3?10
6 7 2
```

Trig

asdf

Contenido

Descripción

Un operador toma funciones (u ocasionalmente datos) como operandos y devuelve una nueva función, la cual es llamada función derivada. El comportamiento de esta función derivada es el mismo que el de cualquier otra función.

Tipos

Monádicos escritos a la derecha de la función que modifican.

Diádicos escritos entre ambos operandos.

Descripción

Un operador toma funciones (u ocasionalmente datos) como operandos y devuelve una nueva función, la cual es llamada función derivada. El comportamiento de esta función derivada es el mismo que el de cualquier otra función.

Tipos

Monádicos escritos a la derecha de la función que modifican.

Diádicos escritos entre ambos operandos.

Operadores monádicos

Al ejecutarse de derecha a izquierda, si el intérprete encuentra un operador, busca a su izquierda para un operando:

- Si es un array, utiliza la **notación de vectores** y usa el array resultante como operando.
- Si encuentra una función, ésta será el operando.
- Si hay otro operador, APL sabe que tiene que producir una función derivada con él, y ésta será el operando requerido.

Todo (/)

Toma una función escalar y la aplica al array en conjunto.

Con la función suma

- La función + con dos operandos suma todos uno a uno.Así:
  `2+1 2 3`, devolvería `3 4 5`.
  `1 2 3+4 5 6`, daría `5 7 9`
- Si queremos obtener la suma de un vector, aplicaremos el operador todo, obteniendo la suma completa:
  `+/1 2 3`, imprimiría por pantalla el valor 6.
- Sumar todos los valores de dos vectores sería:
  `+/1 2 3 + +/4 5 6`, devolviendo el valor 21.

Otras aplicaciones de /

Replicación

```
1 2 3 4/'ABCD'
ABBCCCDDDD
```

Subarray

Caso específico del anterior, utilizando booleanos. `1 0 0 1 1/10 20 30 40 50`

```
10 40 50
```

Cada (¨)

Aplica una función que actúa sobre todo el array, a cada uno de sus elementos.

Con la función de concatenación

- El comportamiento primitivo de esta función es concatenar dos arrays así:

```
1 2 3, 4 5 6
1 2 3 4 5 6
```

- Mediante el operador, indicamos que se aplique recursivamente sobre cada elemento de ambos arrays.

```
1 2 3,¨4 5 6
1 4 2 5 3 6
```

Diádicos

- APL sólo incluye un operador diádico primitivo, pero el usuario puede definir los que desee.
- Este operador es el Producto de Arrays (.), mediante el cual es posible producir una infinidad de funciones derivadas de producto interior (correspondiente a $+.\times$).
- Esta operación multiplica pares de valores de los datos a izquierda y derecha y los suma en grupo.

Diádicos: Producto interior

Definición variables

```
A<-2 3⍴⍳6
1 2 3
4 5 6
```

```
B<-3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
```

Uno a uno

```
+/1 2 3 × 1 5 9
38
+/1 2 3 × 2 6 10
44
```

Con operador

```
A+.×B
38  44  50  56
83  98 113 128
```

Diádicos: Producto interior

Definición variables

```
A<-2 3⍴⍳6
1 2 3
4 5 6
```

```
B<-3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
```

Uno a uno

```
+/1 2 3 × 1 5 9
38
+/1 2 3 × 2 6 10
44
```

Con operador

```
A+.×B
38  44  50  56
83  98 113 128
```

Diádicos: Producto interior

Definición variables

```
A<-2 3⍴⍳6
1 2 3
4 5 6
```

```
B<-3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
```

Uno a uno

```
+/1 2 3 × 1 5 9
38
+/1 2 3 × 2 6 10
44
```

Con operador

```
A+.×B
38  44  50  56
83  98 113 128
```

Producto exterior

Mediante el anterior operador, también puede conseguirse la operación de producto exterior.

$(\iota) \circ . \times \iota 5$

```
 1  2  3  4  5
2  4  6  8  10
3  4  9  12 15
4  8  12 16 20
```

Cálculo de números primos

Una aplicación directa es calcular los números primos.
Una forma bruta de hacerlo en APL es:
$(\sim R \epsilon R \circ . \times) / R \leftarrow 1 \downarrow \iota R$

Producto exterior

Mediante el anterior operador, también puede conseguirse la operación de producto exterior.

```
(⍳) ∘.× ⍳5
 1  2  3  4  5
2  4  6  8  10
3  4  9  12 15
4  8  12 16 20
```

Cálculo de números primos

Una aplicación directa es calcular los números primos.
Una forma bruta de hacerlo en APL es:
$(\sim R \in R \circ . \times) / R \leftarrow 1 \downarrow \iota R$

Bibliografía

- APL2 Language Summary, IBM Silicon Valley Laboratory.
- APL2 User's Guide, IBM Silicon Valley Laboratory.
- APL2 Programming: Language Reference, IBM Silicon Valley Laboratory.