Project Report

Yonathan Torres

COP 4020

Prof. Anwar

This report details my project on what I found when researching a new programming language -called A Programming Language (APL). APL is an array-oriented programming language. What that means is all data consists of multi-dimensional arrays containing different types. The types are Numeric, which can be Boolean 1 or 0, integers, decimals, and even complex numbers; Character, so strings of characters; and Mixed, which is a combination of both. Since everything is an array, this allows easy integration of parallelism to optimize programs for speed, if one so chooses to use them.

APL is also a functional programming language. The difference between other functional programming languages to APL is its use of special symbols to represent primitive functions and operators in the language. For instance, the Greek letter ⍳ (iota) is the monadic primitive function that creates an array representing the indexes of an array of size n. In use,

⍳ 3  
1 2 3  
Also, in APL, code is evaluated from right to left. So the function iota takes the operand 3 on its right to produce the array below.

Primitives in APL are the basic functions and operators given to you in the APL environment, like the keywords reserved in C++ or Java. From these, there are two types of functions and operators, monadic and dyadic. Monadic means only one operand is used, and dyadic means two operands are expected. Most APL functions have both a monadic and dyadic form. For instance, - subtracts in its dyadic form (2-2 -> 0) and negates in its monadic form (-2 -> ¯2). Negative sign on numbers in APL is represented as ¯, so that you can differentiate from subtraction symbol and a negative number. Another example of a monadic and dyadic function is the Upstille ( ⌈ ) primitive. Monadically, it is the ceil function (Ex. ⌈9.1 -> 10). Dyadically, it is the max function (Ex. 2 ⌈ 3 -> 3). Most of the examples have shown the functional side of APL, but the thing about these functions is that they also perform on arrays.

A single number in APL is a scalar. But when you enter another number, it will become a two-dimensional Numeric array. The same functions applied to scalars can be applied to arrays. (Ex. 1 2 3 + 1 2 3 -> 2 4 6)  
( 1 2 – 3 4 -> ¯2 ¯2)  
( 9 2 3 ⌈ 1 2 4 -> 9 2 4)

Like how when vectors in math are added, you add the components up. Same applies to arrays in APL.