DETERMINACION SIMPLE DE UN NÚMERO PRIMO APLICANDO PROGRAMACION FUNCIONAL A TRAVÉS DE DRSCHEME

RESUMEN
Se presenta en este artículo una solución simple y sencilla que aprovecha los recursos conceptuales de la programación funcional y su aplicación en el Lenguaje de Programación DrScheme. Si bien determinar si un número es primo o no corresponde a uno de los enunciados más comunes en programación, la solución que se puede alcanzar aplicando los fundamentos de la programación funcional, y que se presenta en este artículo, podría considerarse como la más simple teniendo en cuenta la aparente complejidad que, desde otros paradigmas, pareciera tener este enunciado. Se basa en tres elementos claros para su implementación: adecuación de la definición de lo que es un número primo, aprovechamiento del concepto de recursión y aplicación del concepto de Funcionalidad que es la que simplifica la solución.

PALABRAS CLAVES: Funciones, Lógica, Números primos, Paradigma, Programación Funcional, programación,

OMAR I. TREJOS BURITICÁ
Ingeniero de Sistemas. Msc.
Profesor Titular
Universidad Tecnológica de Pereira
omartrejos@utp.edu.co

ABSTRACT
Presented in this article a simple and easy solution that leverages in all its fullness the conceptual resources of functional programming and its application in DrScheme Programming Language. While determining whether a number is prime or not for a common set of programming, the solution can be achieved by applying the basics of functional programming, presented in this article. The proposed solution presented in this article is based on three clear elements for its implementation: an adaptation of the definition of what is a prime number for the easy development, the concept of recursion and application the concept of functionality that simplifies the solution.

KEYWORDS: Functions, Logic, Prime Numbers, Paradigm, Functional Programming, Programming.

1. INTRODUCCIÓN

Este artículo es uno de los productos del proyecto de investigación "Desarrollo de contenidos y metodología para un curso de Introducción a la Programación basado en el paradigma de Programación Funcional para estudiantes de primeros semestres de Ingenierías utilizando actividades y técnicas de Active Learning".

Se justifica la temática de este artículo desde dos ópticas: de una parte desde una óptica conceptual al desarrollar un proceso de transformación de una definición para aproximarla a un determinado paradigma de programación, de otra parte desde una óptica práctica dado que dicha transformación se aterriza en un lenguaje de programación funcional como DrScheme. La aplicación de un paradigma de programación en la solución de problemas convencionales propios de esta área es una de las labores más importantes de apropiación de la tecnología computacional.

Se parte de la hipótesis de que siempre es posible encontrar un camino profundamente sencillo para resolver un determinado problema y, más aún, cuando este problema está dentro de los llamados "problemas computacionales", es decir, aquellos problemas que pueden ser desarrollados con tecnología de computadores. Se refuerza esta hipótesis cuando se acude al paradigma de programación funcional como forma sencilla y simple de transformar un objetivo determinado en algo fácilmente alcanzable.

Los procedimientos, funciones, unidades, módulos y programas para determinar si un número es primo o no, en diferentes lenguajes de programación, corresponde a uno de los renglones más comunes cuando se aborda el tema de los ciclos o procesos repetitivos en programación de computadores. En este caso se implementa a través del concepto de recursión.

En su más simple esencia, la solución se implementa a partir de tres principios: una adecuación de la definición de número primo llevándolo de lo matemático a lo funcional, el concepto de recursión y el principio de funcionalidad, propio e identificador del paradigma de programación funcional.

Fecha de Recepción: 17 de junio de 2010
Fecha de Aceptación: 13 de Agosto de 2010

No pretende este articulo ser un análisis riguroso de las propiedades y características matemáticas de los números primos; solo se explicaran algunos antecedentes que permitan tener una concepción más clara del concepto de numero primo. Su objetivo exponer una forma profundamente sencilla de determinar si un número es primo o no, basado en los principios de la funcionalidad y la recursión de la Programación Funcional.

Es de anotar que la solución propuesta para determinar si un número es primo o no, se encuentra en su estado natural y no en un estado óptimo de aprovechamiento de los recursos computacionales ya que ello corresponde al tema de otro artículo en el cual se aprovechan otras características de la programación funcional.

2. DEFINICION MATEMATICA DE NÚMERO PRIMO

En matemática, se considera como número primo a todo número natural que tiene únicamente dos divisores exactos: él mismo y el número 1. Tal es el caso, por ejemplo, del número 13 cuyos únicos divisores exactos son el número 1 y el mismo número 13. Si un número, además del mismo número y del número 1, tiene otros divisores entonces se llama número compuesto. Un ejemplo de ello podría ser el número 15 cuyos divisores exactos son los números 1, 3, 5 y 15.

Se conoce como ***primalidad*** la propiedad que tiene un número de ser, precisamente, un número primo. Debido a que el único número primo par es el número 2 (dado que cumple con la definición) entonces, con frecuencia, se utiliza el término *número primo impar* para referirse a cualquier número primo mayor que 2.

En relación con el número 1 se tiene en la escena matemática una discusión acerca de si ha de considerarse como tal o no. Puede decirse que ambas posturas matemáticas pueden ser, según el caso, inconvenientes o, según otras situaciones, puede ofrecer ciertas ventajas. Hasta el siglo XIX, la comunidad matemática, en su mayor parte, consideraban el 1 como un número primo. La lista de números primos que publicó Derrick Norman Lehmer en 1956 iba desde el número 1 hasta el número 10.006.721 lo cual evidencia que este matemático consideraba al 1 como un número primo.

En la actualidad, la comunidad matemática tiene cierta inclinación a no considerar el número 1 en la lista de los primos. Ello lleva a pensar que principios como el teorema fundamental de la aritmética o algunas propiedades no se cumplen plenamente cuando se trata el número 1. Para efectos de este artículo se considerará el 1 como número primo aunque ha de aclararse que solo se necesita cambiar un valor por otro para que éste quede excluido de la lista, ajustando así la solución que aquí se plantea a la tendencia moderna en cuanto a la no concepción del 1 como número primo.

3. ACERCA DE LOS NUMEROS PRIMOS

Durante la historia de la humanidad (y especialmente de la matemáticas), los números primos han estado presente de una u otra forma en el desarrollo del pensamiento y de diferentes teorías. Una pequeña muestra de ello la constituyen los llamados huesos de Ishango que parecen tener más de 20.000 años de antigüedad (paleolítico superior). Esta pieza arqueológica consiste en un largo hueso que, por sus características en los extremos, parecía ser una herramienta para grabar o escribir.

Inicialmente se pensó que se usaba para realizar conteos, dado que tenía una serie de muescas talladas y organizadas en tres columnas a lo largo de toda la pieza. Sin embargo, debido a la presencia de de los números primos 11, 13, 17 y 19 al hacer una revisión minuciosa de las muescas, se ha sugerido por parte de la comunidad académica que el hombre de esos tiempos pareciera tener un entendimiento matemático que iba más allá del conteo. Estas teorías se han perdido entre el mito y la realidad pues no se tienen mayores elementos que permitan probar o negar lo planteado.

Otro ejemplo interesante se puede inferir de las, llamadas, matemáticas egipcias en las cuales el cálculo de fracciones requería unos buenos conocimientos sobre tres tópicos propios de la matemática: las operaciones, la división de los naturales y las formas de factorización. Las fracciones egipcias se le llamaban a aquellas fracciones en la cuales el numerador siempre es 1.

De esta forma son fracciones egipcias $\frac{1}{3}$, $\frac{1}{2}$ y para escribir una fracción como $\frac{1}{6}$ la expresaban como suma de inversos naturales lo cual hace suponer que los egipcios tenían una noción de los números primos ya que estos aparecen cuando se simplifican números, tal como se hace en la actualidad con la descomposición de un número en sus factores primos.

Euclides, hacia el año 300 a.C. plantea en sus escritos una prueba irrefutable del conocimiento de los números primos y en su libro Elementos los define, demuestra que hay infinitos de ellos y proporciona un método para determinarlos conocido, en una definición muy moderna, como el algoritmo de Euclides. Entre estos métodos se hace imposible omitir la famosa criba de Eratóstenes que no es más que un método sencillo que permite encontrar números primos.

Esta preocupación y conocimiento acerca de los números primos tuvo pocos avances en los siglos siguientes. Fue solo hasta 1640 cuando Pierre de Fermat planteó que todos los números de la forma $2^{2n}+1$ eran números primos (conocido también como el teorema de Fermat). Este teorema fue controvertido, en años posteriores, por

Leibniz y Euler. Este último demostró que el número $2^{32} + 1$ era un número compuesto y por lo tanto no era un número primo. El teorema perdió validez pero su preocupación al respecto dinamizó la matemática alrededor de los números primos. Uno de los que siguió trabajando en estas teorías fue el matemático Euler.

Gauss fue otro matemático que incorporó la preocupación sobre los primos como suya e hizo algunas demostraciones al respecto. Fue hacia 1859 cuando el matemático alemán Bernhard Riemann estableció un camino que presentaría la demostración del teorema de los números primos que capitaliza las características de los logaritmos neperianos para encontrar una cantidad finita de números primos. El teorema fue demostrado hacia 1896 por Hadamard.

A partir de estas preocupaciones son muchos los algoritmos que la matemática ha proporcionado para demostrar la primalidad de un determinado número Algunos nombres como Proth (1878), Pocklington (1914), Brillahart, Lehmer y Selfridge (19759 y, más recientemente, Konyagin y Pomerance (1997) aparecen en la escena matemática proponiendo algoritmos y nuevos conceptos matemáticos que proporcionan, desde una óptica puramente matemática, elementos para demostrar si un número es primo o no.

El más reciente test de primalidad se conoce como algoritmo AKS y fue formulado hacia 2002 el cual parte de una complejidad polinómica y que exige tecnología de alta velocidad computacional para que el tiempo de respuesta sea más breve.

La utilidad que en la actualidad se le encuentra a los números primos corresponde al área de la criptografía y específicamente a la definición de claves públicas, en la cual los números primos forman parte de la base de los algoritmos de seguridad. Debe anotarse que, desde 1951, el mayor número primo ha sido descubierto con la ayuda de computadores.

Este pequeño recorrido histórico pretende demostrar que la caracterización de los números primos ha sido una preocupación del mundo matemático y que hoy ha rebasado las fronteras de las matemáticas incorporando en su análisis a profesionales de la informática y de otras áreas que también han hecho aportes al respecto.

4. DEFINICION FUNCIONAL DE NÚMERO PRIMO

La solución que se presenta en este artículo busca aprovechar al máximo la capacidad computacional moderna para obtener la respuesta acerca de si un número es primo o no esperando que el tiempo de respuesta sea el más breve posible y sabiendo que el guarismo ha de ser introducido por el usuario.

Dado que la definición de número primo establece que éste es todo aquel número natural que tiene solamente dos divisores exactos y que son el número 1 y el mismo número en cuestión, entonces es claro que un número primo podríamos definirlos, para efectos de construir el programa solución, de la siguiente forma: *Un número primo es aquel entero que tiene solamente, y no mas que, dos divisores exactos en el rango [1, n] siendo n el número que se quiere evaluar.*

Esta definición tan simple, y tan próxima a la definición matemática pues deriva de ella, nos permitirá entonces pensar en que si contamos la cantidad de divisores exactos que tiene un número cualquiera y si notamos que esa cantidad es igual a 2, entonces podemos concluir que el número es primo. Es aquí en donde la definición de primalidad en el número 1 podría aparecer en escena y la resolvemos de una forma muy sencilla: tomar el número 1 como número primo para lo cual la definición que se presenta en el párrafo anterior es suficiente y no requiere ninguna modificación ó excluir el número 1 (y de paso excluir, por redundancia, el mismo número que se quiere evaluar).

En este último caso la definición de número primo podría tener la siguiente definición: *Un número primo es aquel entero que no tiene ningún divisor exacto en el rango [2, n-1] siendo n el número que se quiere evaluar.* Y entonces podríamos ser mucho más eficientes y determinar que, sabiendo que de la mitad + 1 de un número n hasta (n-1) no existen divisores exactos. Por ejemplo, el número 1000 no tiene divisores exactos entre el número 501 y el número 999. De forma que una definición más eficiente, en términos computacionales, podría ser: *Un número primo es aquel entero que no tiene ningún divisor exacto en el rango [2,* $\frac{n}{2}$ *] siendo n el número que se quiere evaluar.*

Para efectos de este artículo se utilizará la primera definición que se planteó en esta sección. A manera de optimización, es de aclarar que con el simple hecho de que se encuentre un solo divisor exacto en ese rango es suficiente para que el número no sea primo.

5. PRINCIPIOS DE PROGRAMACION FUNCIONAL

La programación funcional basa todas sus soluciones en la buena organización e interconexión de una unidad de trabajo que, para este paradigma, corresponde al núcleo del mismo: la función. La función podríamos definirla como un “pequeño programita” que cumple de manera específica y clara con un “pequeño objetivo” y que hace un aporte concreto a la solución de un problema. El desarrollo de varias funciones y su correcta interconexión (o también conocidos como llamados) permite que sea posible computacionalmente cualquier solución.

Para lograrlo la programación funcional se basa en tres principios que, a la postre, resuelven los problemas más grande que se tienen en programación de computadores y que retan a los paradigmas cuando un programador se enfrenta a ellos:

a) **Simplificación del Objetivo**.- Antes de comenzar a escribir un programa, lo más importante es que se tenga muy claro cuál es el OBJETIVO que se quiere lograr o sea qué es lo que se quiere hacer con el programa, cuáles son los resultados que se quieren obtener y qué elementos intervienen para que podamos alcanzar ese objetivo. Muchas veces nos encontramos con que el Objetivo que se quiere lograr tiene un nivel de complejidad se dificulta concebirlo como un todo.

   Es allí en donde la programación funcional nos permite dividir el objetivo general en pequeños objetivos (o sea en funciones) de manera que cada función logre un pequeño objetivo y de esta forma el gran objetivo sea logrado por la articulación de todos los pequeños objetivos (y que valgan todas estas redundancias!).

   Es claro que cuando se simplifica un objetivo complejo y se divide en pequeños objetivos más simples, es mucho más fácil lograr cada uno de los pequeños objetivos y posteriormente articularlos para lograr el objetivo complejo, que querer hacer todo de una vez. Esa es precisamente una de las grandes fortalezas de la programación funcional. Cada función permite lograr un pequeño objetivo y, si se hace apropiadamente, podremos, a través de esta técnica, lograr objetivos mucho más complejos de lo que nosotros mismos nos hemos imaginado. Con cierta justicia matemática y computacional esta estrategia se conoce como “Divide y Vencerás”.

b) **Detección de Errores**.- Cuando se hace un programa que tiene un objetivo amplio y complejo, encontrar los errores lógicos que tenga el programa es una tarea que no siempre es tan fácil puesto que, si partimos de que un programa es un conjunto amplio de líneas de código, los errores fácilmente se pueden sumergir entre todas las líneas y encontrarlo se hace bastante dispendioso, aunque no imposible del todo.

   Sin embargo, cuando se trabaja bajo la lógica de la programación funcional se hace muy fácil encontrar errores ya que como cada función cumple con un pequeño objetivo y es una unidad relativamente pequeña e independiente de programación, es fácil hacer un seguimiento a la lógica con que fue construida y, de esta forma, hallar los errores (especialmente los lógicos) que se presenten. Sin lugar a dudas es mucho más fácil encontrar errores lógicos en un “programita” que en un programa.

c) **Reutilización de Código**.- También es conocida como reusabilidad de código. Otras de las grandes dificultades que se presentan en la programación de computadores es poder reutilizar algo que ya hayamos construido. Cuando nosotros desarrollamos un programa y vemos que, posteriormente, ese programa puede ser parte de otro objetivo más grande no siempre es tan fácil articular el nuevo programa con el programa que ya construimos para que no tengamos que rehacer de nuevo lo que ya hicimos. Aunque el juego de palabras pareciera enredado, el concepto es muy sencillo.

   La programación funcional nos permite, por sus características, que cada función sea una unidad que logre un determinado objetivo y, como todo está concebido a la luz del mismo concepto, entonces es fácil hacer llamados a funciones que ya tengamos construidas sin tener que repetir el código o tener que volver a desarrollarlo. Debe tenerse en cuenta que las funciones son unidades independientes y que precisamente eso es lo que más facilita el llamado entre unas y otras.

6. LOGICA DE SOLUCION

El enunciado del problema tendrá el siguiente texto: Construir un programa, basado en paradigma de programación funcional, que permita leer un entero y determinar si es un número primo. La simplicidad a la que se ha hecho alusión a lo largo de este artículo, y que constituye el gran aporte del paradigma de Programación Funcional, radica en que, a partir de la simplificación de la definición de lo que es un numero primo, se estructure una solución con tres funciones (cuatro con la función que muestra los resultados) en donde cada una es profundamente sencilla y pareciera cumplir con un pequeño objetivo insignificante. Sin embargo, la interconexión entre ellas es lo que permite que la solución sea posible.

De forma que no podrá decirse que existe una función, en esta solución, que resuelve el problema dado que el problema es resuelto por la manera como dichas funciones están interrelacionadas. Por ello, y teniendo en cuenta lo que se acaba de exponer, para construir su solución se procederá de la siguiente forma:

Bases

Se tomarán como bases para el desarrollo del programa solución las siguientes bases:

- Un ajuste a la definición de número primo a partir de su interpretación computacional y que, pare los efectos de programación, reza “*Un número primo es aquel entero que no tiene ningún divisor exacto en el rango [2, $\frac{n}{2}$] siendo n el número que se quiere evaluar”*

- Se apelará al concepto de Recursión que se define como la propiedad que tiene una función de llamarse a sí misma y que resulta útil en la implementación de procesos cíclicos
- Se construirá la solución a partir del concepto de Funcionalidad de manera que se puedan proponer funciones independientes e interdependientes

Funciones

Teniendo en cuenta todo esto, desarrollaremos las siguientes funciones para efectos de la solución:

- Una función que reciba dos argumentos y determine si uno de ellos divide exactamente al otro. Esta función retornará un 1 en caso de que un argumento sea un divisor exacto del otro, retornará 0 en caso contrario.
- Una función que determine la cantidad de divisores exactos que tiene un número entero en el rango [1, n/2]. Esta función retornará un valor entero mayor o igual a 0. Cuando esta función retorne un valor igual a 0 significa que el número que se le envió como argumento es un número primo. Cuando la función retorne un valor diferente de 0 (o sea un valor positivo) significa que el número que se envió como argumento no es un número primo.
- Una función que reciba un valor entero y lo envíe como argumento para evaluar si corresponde a un número primo.

7. CODIGO SCHEME

Se presenta a continuación el código DrScheme que desarrolla la solución planteada en el numeral anterior:

```
;; Función que determina si un valor es múltiplo de otro
;; Esta función calcula el cociente de la división de a
entre b y, ese valor, lo multiplica por b
;; De esta forma si se obtiene el mismo resultado,
entonces significa que a es múltiplo de b
;; Si no se obtiene el mismo resultado, significa que a no
es múltiplo de b

( define (divisor a b)
  (if ( = a ( * (floor (/ a b)) b ))
      1
      0
  ))

;; Calcula la cantidad de divisores de un número
;; Como esta función se basa en la función divisor
(función anterior) entonces en caso de que esta
;; división retorne el valor 1, este valor se acumula en la
suma que se expresa al final de la función
;; cuentadivisores. En caso de que la función divisor
retorne 0 entonces ese valor se suma (no
;; sumaría nada) a la misma expresión

( define (cuentadivisores num div)
  (if (= div 0)
      0
  ( + (divisor num div)(cuentadivisores num (- div 1)))
  ))

;; Determina si un número es primo
;; Esta función recibe el valor a evaluar como argumento
y determina, acorde con el valor que
;; retorne la función cuentadivisores, si el número es
primo o no. Si el valor retornado es 2 significa ;; que los
dos únicos divisores del argumento serán 1 y el mismo n,
en caso contrario (o sea
;; cuando se retorne un valor mayor que 2) significa que
el número no es primo
;; Despliega un título correspondiente dependiendo del
valor recibido

( define (esprimo n)
  (if (= (cuentadivisores n n) 2)
    (display "El numero es primo")
    (display "El numero no es primo")
  ))
```

Notas Aclaratorias

- En la función *divisor( )* la instrucción *(floor (/ a b))* se puede reemplazar por *(quotient a b)*
- El valor 1 que retorna la función *divisor( )* se puede interpretar como *Verdadero* en relación con el objetivo de la función
- En la función *cuentadivisores( )*, el valor inicial del argumento *div* es *n* dado que los posibles divisores exactos de n están entre n y 1 (haciendo un recorrido regresivo)
- Si se quiere omitir el 1 como número primo todo lo que tenemos que hacer es llamar a la función *cuentadivisores( )* enviándole n como 1º argumento y *(- n 1)* como segundo argumento y cambiando la decisión de la función *esprimo( )* de forma que en vez de evaluar frente al número 2 se evalúe frente al número 0. Ha de tenerse cuidado en hacer las evaluaciones pertinentes cuando el valor que se quiera evaluar sea el mismo número 1
- La función *esprimo( )* podría tener unos títulos más completos e ilustrativos para el usuario

8. RESULTADOS

La prueba de escritorio, realizada con valores pequeños, arroja unos resultados totalmente confiables dado que si el número que se envía como argumento es primo entonces el valor que retorna la función *cuentadivisores( )* es 2, tal como se presenta en esta solución, lo cual indica que el número efectivamente es primo (pues sus dos únicos divisores serían el 1 y el mismo número). En caso de que el número no sea primo, la prueba de escritorio realizada con valores pequeños arrojará un valor mayor que 2, dependiendo del número, y con ello indicará que el número no es primo. Sería agotador e

innecesario realizar pruebas de escritorio con números muy grandes dado que es suficiente con la confiabilidad que arrojan los resultados con valores pequeños.

En cuanto a los resultados obtenidos con DrScheme la confiabilidad es total aunque no ha de desconocerse que el tiempo de cálculo en computadores de pequeña capacidad de procesamiento es significativamente cuestionable. Precisamente el objetivo de este artículo es mostrar una forma muy sencilla de resolver un problema que, en otras instancias, ha sido planteado y explicado a través de algoritmos complejos de entender y mucho más complejos de probar. Para la determinación del número 1000001 y su utilización con la función *esprimo( )*, el cálculo demora un poco mas de segundo y medio. De todas formas no es el objetivo de este artículo presentar una versión *óptima* para dicho cálculo sino una versión *sencilla y muy simple* para tal objetivo.

9. CONCLUSIONES

En referencia con la solución planteada al problema que se formula inicialmente, se pueden plantear las siguientes conclusiones:

- La programación funcional provee una teoría suficientemente sólida como para poder concebir y entender soluciones sencillas a problemas comunes siempre y cuando se hayan asimilado las bases propias de tal paradigma y que hacen referencia a la construcción e interconexión de funciones a partir del principio de funcionalidad y, en segunda instancia, el concepto de recursión
- En particular la solución planteada en este artículo cumple plenamente con dichos principios lo cual la hace de fácil concepción y fácil de entender
- Soluciones construidas a partir de los principios de la funcionalidad tienen la gran ventaja de ser muy fáciles de desarrollar ya que una cosa es que se conciba una solución fácil a un problema y otra es que el lenguaje permita y posibilite dicha construcción
- La lógica que se utiliza para la solución también es muy sencilla ya que se basa en tres funciones (dos operativas) cuya concepción a partir de la teoría general o de la documentación es más que simple
- Como se vio en el cuerpo del artículo, una de los principales problemas a los cuales se enfrenta un programador es la necesidad de poder hacer pruebas parciales o totales de los programas. La programación funcional facilita que se hagan pruebas a cada función (es decir pruebas parciales) y que se pueda garantizar el buen funcionamiento y el logro de los objetivos planteados inicialmente
- La utilización del programa es tan sencilla que todo lo que se necesita es enviar como argumento el valor que se quiere evaluar y el programa hará lo demás, es decir, determinará si el argumento enviado es un número primo o no
- Otra característica que tiene esta solución, cumpliendo con los principios de la programación funcional y más directamente con el paradigma funcional, es que se le pueden hacer cambios a las funciones de una manera directa y sencilla de forma que en cualquier momento se puedan hacer ajustes dependiendo de lo necesario que éstos sean
- Aún a pesar de los elementos de juicio que le aporta el paradigma de programación funcional a la solución de un problema computacional, es posible optimizarlo mucho mas aprovechando las características modernas de las tecnologías informáticas.

REFERENCIAS BIBLIOGRAFICAS

[01] M. Felleisen, R. B. Findler, M. Flatt, S. Krishnamurthi, How to design program, An Introduction to Programming and Computing, The MIT Press, Cambridge, Massachussets, 2003, p. 45-62.

[02] H. Eves, An introduction to the history of mathematics, Editorial Saunders, ISBN 0-03-029558-0, USA, 1990, p. 64

[03] A. Roger y otros, Las antiguas Ciencias de Oriente, Ediciones Orbis, Barcelona, 1988, España, p. 35.

[04] V. Roy, Concepts, Technics and models of computer programming, Swedish Institute of Computer Science, 2003, p. 103

[05] B. Brendan, A new approach to the computer science, Univ. of Massachussets, USA, 2005, p. 7.

Omar Ivan Trejos Buriticá. Ingeniero de Sistemas y Computación, Especialista en Instrumentación Física, Magister en Comunicación Educativa, estudiante del Doctorado en Ciencias de la Educación de RudeColombia.