|  |  |
| --- | --- |
| **lenguajes de programación esotéricos**  TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES | Descripción breve  Un trabajo de investigación sobre el funcionamiento de los lenguajes de programación esotéricos, e implementación práctica del lenguaje COW.  Diego Lopez Reduello  2ºC Ingeniería del Software |

ÍNDICE

[Introducción 2](#_Toc124778309)

[Definición 2](#_Toc124778310)

[Ejemplos destacables 2](#_Toc124778311)

[INTERCAL 2](#_Toc124778312)

[Piet 4](#_Toc124778313)

[Lenguaje COW: Lógica y funcionamiento 5](#_Toc124778314)

[Lenguaje COW: Instalación y preparación 6](#_Toc124778315)

[Implementación de programas en COW 7](#_Toc124778316)

[Repositorio GitHub del proyecto 7](#_Toc124778317)

[Repositorio original de COW 7](#_Toc124778318)

[Bibliografía 7](#_Toc124778319)

# Introducción

En este trabajo de investigación para la asignatura Teoría de Autómatas y Lenguajes Formales, explicaré qué es un lenguaje de programación esotérico, mostraré algunos ejemplos de lenguajes de programación esotéricos destacables tanto por su complejidad como por su originalidad y, por último, una implementación práctica de un lenguaje esotérico a mi elección. En este caso, he escogido el lenguaje esotérico COW.

# Definición

Un lenguaje de programación esotérico es un lenguaje de programación minimalista que tiene como principal función demostrar que se pueden crear lenguajes pocos intuitivos, o incluso a veces absurdos, que son Turing-computables, es decir, lenguajes que pueden computar todas las funciones que podría computar una Máquina de Turing.

Su utilidad en casos prácticos o en proyectos de gran escala es bastante dudosa, debido a su complejidad de código y otras características poco comunes en lenguajes de programación. Son programas que suponen un reto para el que lo crea, y también un reto para los que intenten programar cualquier cosa con ellos.

Actualmente existe una comunidad en internet bastante activa que se dedica a crear todo tipo de lenguajes de programación esotéricos y escribir programas en ellos por diversión, además de que también debaten sus propiedades computacionales.

# Ejemplos destacables

## INTERCAL

INTERCAL es uno de los lenguajes de programación esotéricos más conocidos, extendidos y antiguos de la historia. Su diseño se basa en la siguiente mecánica: debemos pedir por favor la ejecución de ciertas sentencias. En vez de usar la instrucción mítica de los lenguajes de programación de bajo nivel *“GO TO”* para saltar a una instrucción concreta, usaremos una instrucción llamada *“COME FROM”* en la instrucción concreta a la que queremos llegar. También tiene otras peculiaridades, como evitar que se ejecuten ciertas sentencias con *“PLEASE ABSTAIN FROM”*, entre otras peculiaridades. Este lenguaje de programación se creó con la intención de crear algo totalmente distinto. A continuación, tenemos un programa escrito en INTERCAL que lee dos enteros de 32 bits, tratándolos como enteros con signo en formato de complemento a dos, y escribiendo su valor absoluto.

DO (5) NEXT

(5) DO FORGET #1

PLEASE WRITE IN :1

DO .1 <- 'V":1~'#32768$#0'"$#1'~#3

DO (1) NEXT

DO :1 <- "'V":1~'#65535$#0'"$#65535'

~'#0$#65535'"$"'V":1~'#0$#65535'"

$#65535'~'#0$#65535'"

DO :2 <- #1

PLEASE DO (4) NEXT

(4) DO FORGET #1

DO .1 <- "'V":1~'#65535$#0'"$":2~'#65535

$#0'"'~'#0$#65535'"$"'V":1~'#0

$#65535'"$":2~'#65535$#0'"'~'#0$#65535'"

DO (1) NEXT

DO :2 <- ":2~'#0$#65535'"

$"'":2~'#65535$#0'"$#0'~'#32767$#1'"

DO (4) NEXT

(2) DO RESUME .1

(1) PLEASE DO (2) NEXT

PLEASE FORGET #1

DO READ OUT :1

PLEASE DO .1 <- 'V"':1~:1'~#1"$#1'~#3

DO (3) NEXT

PLEASE DO (5) NEXT

(3) DO (2) NEXT

PLEASE GIVE UP

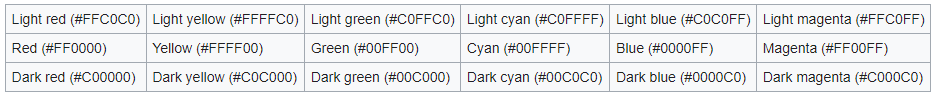
## Piet

Piet fue el lenguaje de programación que personalmente hizo que me interesara por el tema de los lenguajes esotéricos. Se trata de un lenguaje de programación basado en stacks (una estructura de datos usada frecuentemente en las memorias de los lenguajes de programación esotéricos) cuyos programas parecen pinturas de arte abstracto. El programa que imprime en consola Hello World es el siguiente:

Imagen digital de colores

Descripción generada automáticamente con confianza baja

Piet usa 20 colores en sus programas, de los cuales 18 tienen dos propiedades: *hue* y *lightness* (se podría traducir al español como matiz y dureza). En la siguiente tabla podemos ver las diferentes variaciones de colores:



Las instrucciones se escriben estableciendo la diferencia de propiedades entre el un color y otro, siguiendo la siguiente tabla de comandos:

Tabla

Descripción generada automáticamente

Por ejemplo, de *Light read* a *Yellow* hay un paso de *Hue change* y un paso de *lightness* (se cuenta en la tabla cuántos pasos de diferencia hay) y eso significaría que tenemos que ejecutar la instrucción *Substract*.

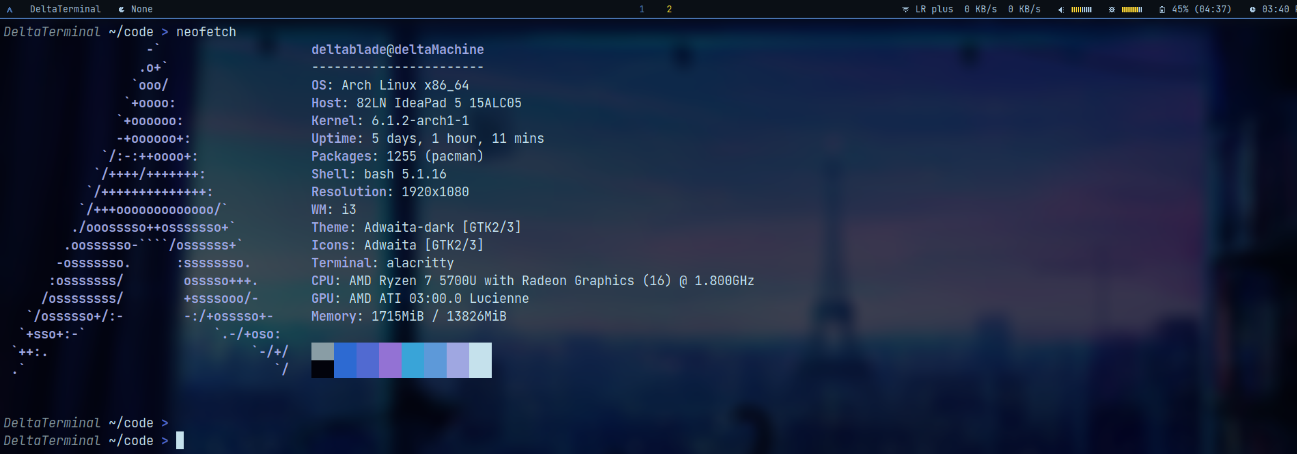
# Lenguaje COW: Lógica y funcionamiento

COW fue creado a partir de Brainfuck, uno de los lenguajes esotéricos más conocidos. Funciona con un vector de memoria, un puntero que señala un bloque de ese vector, un flujo de entrada y un flujo de salida. Las posibles instrucciones son las siguientes:

|  |  |  |
| --- | --- | --- |
| **ID** | **Instrucción** | **Descripción** |
| 0 | moo | Esta instrucción está conectada con la instrucción MOO. Sirve para crear bucles. Cuando se encuentra durante una ejecución, busca en el programa (hacia atrás) un comando MOO y empieza a ejecutar las instrucciones desde ahí. |
| 1 | mOo | Mueve el puntero de memoria al bloque anterior. |
| 2 | moO | Mueve el puntero de memoria al bloque siguiente. |
| 3 | mOO | Lee el valor del bloque de memoria actual y ejecuta la instrucción asociada a ese valor. No funciona para el 3, ya que provocaría un bucle infinito, ni para números estrictamente mayores a 11. |
| 4 | Moo | Si el valor del bloque de memoria actual es 0, lee del flujo de entrada un carácter ASCII y almacena su valor en el bloque.  En otro caso, imprime en la salida el carácter ASCII que corresponde al valor del bloque de memoria. |
| 5 | MOo | Decrementa en uno el valor del bloque de memoria actual. |
| 6 | MoO | Incrementa en uno el valor del bloque de memoria actual. |
| 7 | MOO | Si el valor del bloque de memoria actual es 0, salta al siguiente comando moo y sigue ejecutando desde la instrucción posterior a moo.  En otro caso, continua con el siguiente comando. |
| 8 | OOO | Establece a 0 el valor del bloque de memoria actual. |
| 9 | MMM | Si no hay ningún valor almacenado en el registro, copia el valor del bloque de memoria actual.  Si hay un valor almacenado, se pega el valor en el bloque de memoria actual y se resetea el registro. |
| 10 | OOM | Imprime en la salida el valor del bloque de memoria como un Integer. |
| 11 | oom | Lee de la entrada un número entero y lo escribe en el bloque de memoria actual. |

# Lenguaje COW: Instalación y preparación

Primero, inicié el sistema operativo Arch Linux en mi máquina local para trabajar de forma más cómoda con COW.



Después, en la carpeta correspondiente, cloné el repositorio original de cow con el comando “git clone <https://github.com/BigZaphod/COW.git>” y moví la carpeta source y algunos ejemplos de código a mi carpeta cowInvestigation. Creé el repositorio (al que se puede acceder con este enlace: <https://github.com/Deinigu/cowInvestigation>) y añadí el contenido de la carpeta.

En la carpeta source, hay dos archivos que nos interesan:

* cow.k: Es simplemente un archivo que contiene las traducciones de las instrucciones de COW a C++.
* cow.cpp: Es el programa escrito en C++ que, al compilarse, se puede usar para interpretar archivos escritos en el lenguaje COW.

Compilé cow.cpp con el comando “gcc cow.cpp -o compiladorcow” y moví el archivo de salida a la carpeta cow, donde están también los diferentes ejemplos.

A continuación, explicaré algunos ejemplos de programas escritos en cow y escribiré alguno propio.

# Implementación de programas en COW

## Helloworld.cow

## Fibonacci.cow

Este programa viene en la carpeta de ejemplos del repositorio original. Imprime en pantalla en bucle los números de la sucesión de Fibonacci. El código (comentado con su modo de funcionar) es el siguiente:

|  |
| --- |
| ; Inicialización de las variables |
| MoO |
| moO |
| MoO |
| mOo |
| ; Loop |
| MOO |
| ; Imprime el primer número |
| OOM |
| ; Copia en el registro el primer número y lo lleva a un bloque |
| ; al que llamaremos temp |
| MMM |
| moO |
| moO |
| MMM |
| mOo |
| mOo |
| ; Pone el segundo número en la primera posición |
| moO |
| MMM |
| mOo |
| MMM |
| ; Volvemos al bloque temp |
| moO |
| moO |
| ; Usamos el bloque temp para sumarlo al primer número y lo guardamos |
| ; en el segundo y volvemos a comenzar el bucle |
| MOO |
| MOo |
| mOo |
| MoO |
| moO |
| moo |
|  |
| mOo |
| mOo |
| moo |

Al ejecutar el programa, imprime lo siguiente:

|  |
| --- |
| DeltaTerminal ~/code/cowInvestigation/cow > ./cow.out |
| 1 |
| 1 |
| 2 |
| 3 |
| 5 |
| 8 |
| 13 |
| 21 |
| 34 |
| 55 |
| 89 |
| 144 |
| 233 |
| 377 |
| 610 |
| 987 |
| 1597 |
| 2584 |
| 4181 |
| 6765 |
| 10946 |
| 17711 |
| 28657 |
| 46368 |
| 75025 |
| 121393 |
| 196418 |
| 317811 |
| 514229 |
| 832040 |
| 1346269 |
| 2178309 |
| 3524578 |
| 5702887 |
| 9227465 |
| 14930352 |
| 24157817 |
| 39088169 |
| 63245986 |
| 102334155 |
| 165580141 |
| 267914296 |
| 433494437 |
| 701408733 |
| 1134903170 |
| 1836311903 |
| -1323752223  ^C |

El número negativo no es problema del programa, sino del tipo de datos signed int, que es de 4 bytes y tiene un rango de valores desde -2,147,483,648 hasta 2,147,483,647. Como el siguiente número, que sería 2971215073, no se encuentra en este rango, devuelve un número negativo.

# Repositorio GitHub del proyecto

<https://github.com/Deinigu/cowInvestigation>

# Repositorio original de COW

<https://github.com/BigZaphod/COW>

# Bibliografía

* <https://esolangs.org/wiki/>
* <http://www.formauri.es/personal/pgimeno/compurec/LenguajesEsotericos.php>
* <https://es.wikipedia.org/wiki/Lenguaje_de_programación_esotérico>
* <https://bigzaphod.github.io/COW/>
* <https://frank-buss.de/cow.html>
* <https://es.wikipedia.org/wiki/Brainfuck>
* <https://r-knott.surrey.ac.uk/Fibonacci/fibtable.html>
* <https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/>
* <https://www.learncpp.com/cpp-tutorial/signed-integers/>