Lógica para ciencias de la computación

Nombre:

* Ricardo vergara Toloza
* Carrera: Ingeniería civil informática
* Fecha: 21-12-2019
* Rut : 20.199.732-1

Introducción

La programación lógica es uno de los paradigmas de la programación, este tipo de paradigma pertenece a las de tipo declarativa en contraposición a la [programación imperativa](https://www.ecured.cu/index.php?title=Programaci%C3%B3n_imperativa&action=edit&redlink=1), es un [paradigma de programación](https://www.ecured.cu/index.php?title=Paradigma_de_programaci%C3%B3n&action=edit&redlink=1) que está basado en el desarrollo de [programas](https://www.ecured.cu/Software) especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan solo se le indica a la [computadora](https://www.ecured.cu/Computadora) qué es lo que se desea obtener o qué es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con [transparencia referencial](https://www.ecured.cu/index.php?title=Transparencia_referencial&action=edit&redlink=1). (Ecured, 2019).

Uno de estos es lenguajes es Prolog , es un lenguaje de programación lógica cuya primera versión fue desarrollada a principios de la década de 1970 por Colmerauer en la universidad de Marsella. Contrariamente a otros lenguajes de programación basados es estructuras de control y definición de funciones para calcular resultados, Prolog está orientado a la especificación de relaciones para responder consultas. En ese sentido Prolog es similar a un sistema de base de datos, aunque en el contexto de la inteligencia artificial se prefiere hablar de bases de conocimiento, enfatizando la complejidad estructural de los datos y de las deducciones que se pueden obtener de ellos (uchile, 2019).

Problema 1

**Genere una base de conocimiento a elección con información real ,realice las consultas pertinentes a su base de conocimiento y demuestre que puede ser utilizado para fuente de conocimiento real**

La base de conocimiento se basa en el registro de producto de la tienda Pc Factory y el cómo podemos consultar nombres, tipos de productos (CPU, GPU, etc.) , precios, marcas , garantía de los distintos tipos de producto de la tienda.

Algunos datos fueron sacados de las siguientes páginas:

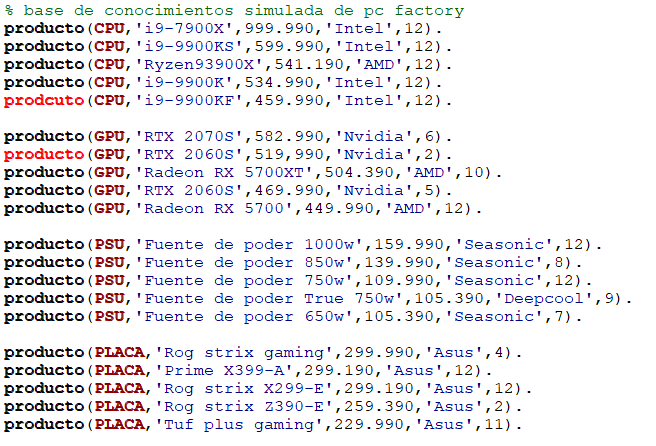
<https://www.pcfactory.cl/procesadores?categoria=272&papa=633>

<https://www.pcfactory.cl/tarjetas-graficas?categoria=334&papa=633>

<https://www.pcfactory.cl/fuentes-de-poder-psu-?categoria=54&papa=633>

<https://www.pcfactory.cl/placas-madres?categoria=292&papa=633>

Base de conocimientos



Menor a : Este hecho permite mostrar productos con un precio menor al número dado.



Mayor a: Es el hecho opuesto del mencionado anteriormente.



Garantía: Muestra productos con “N” meses de garantía.



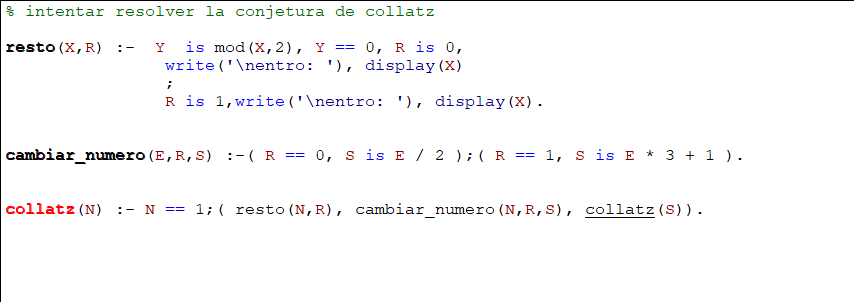
Problema 2

**Genere una base de conocimiento que resuelva la Conjetura de Collatz**

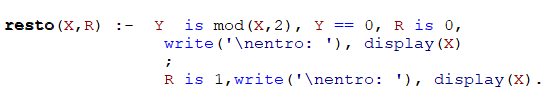
La conjetura de Collatz fue propuesta por el matemático Lothar Collatz en 1937 la cual dice que al realizar cierta operación a los números naturales siempre se obtendrá la unidad.

* Si n es par divide entre 2
* Si n es impar multiplicar por 3 y suma 1 al resultado

El programa se encargara de recibir un numero “n” y aplicar el procedimiento recién mencionado hasta que el numero “n” sea 1.

****

Lo primero que se realizo para hacer el programa fue crear un hecho que determine si un numero es par o impar, Para esto debemos tener un parámetro “X” que almacena que numero queremos evaluar, es decir “X” será el numero a preguntar si es par o impar y un segundo parámetro “R” que valdrá un 0 en caso de que el numero “X” sea par y 1 si “X” es impar, el procedimiento para determinar si R debe ser 0 o 1 será dividir el numero “X” en 2 y guardar el resto de esta división en una variable llamada “Y”, para finalizar si la variable “Y” es igual a 0 sabemos que “X” es un numero par por lo cual asignaremos a el parámetro “R” el valor 0, en caso de no cumplir esta condición , es decir “Y” es igual a 1 asignaremos 1 al parámetro “R”.



El segundo paso será hacer la operación al numero, como en el paso anterior ya determinamos si el numero era par o impar necesitamos un parámetro que reciba a que numero vamos a realizar la operación al cual nombramos “E”, un segundo parámetro “R”, que será el mismo “R” del paso anterior y un tercer parámetro llamado “S” que almacenara la operación que le tengamos que hacer a la variable “E”. Para decidir que hacer preguntaremos si “R” es 0, en ese caso “S” es el numero “E” dividido en dos, sino eso implica que “R” es 1 y “S” es “E” multiplicado por 3 y a este resultado le sumaremos 1.



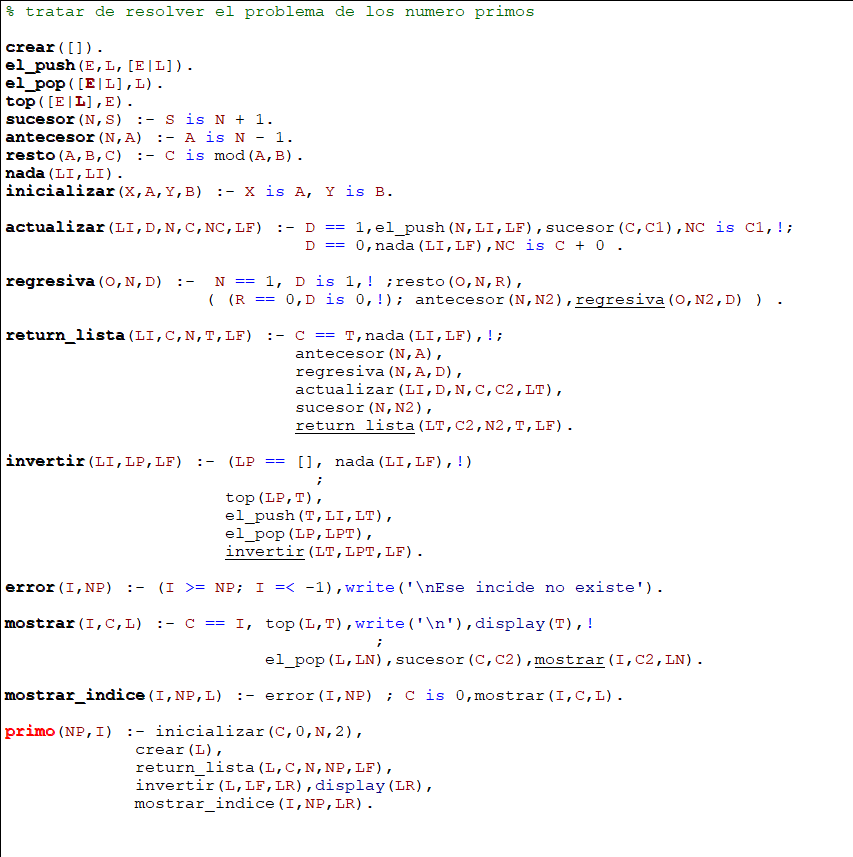
El ultimo paso será tratar de realizar los dos pasos anteriores hasta el numero sea la unidad, por lo cual sabemos que si el numero llega a uno nos detendremos, en caso de no ser la unidad repetiremos los dos pasos anteriores y volver a evaluar nuestro nuevo numero hasta que sea la unidad, entonces lo único que falta es pedir el numero al usuario para ello solo se usara un parámetro llamado “N” que registrara que numero queremos probar en la conjetura de collatz.



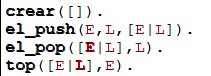
Problema 3

**Genere una base de conocimiento que al ingresar un número, nos devuelve el elemento de una lista localizado en la posición indicada. La lista debe contener los primeros 20 números primos y debe ser llenada de manera automática**

Un número primo es un número entero mayor que cero, que tiene exactamente dos divisores positivos. También podemos definirlo como aquel número entero positivo que no puede expresarse como producto de dos números enteros positivos más pequeños que él, o bien, como producto de dos enteros positivos de más de una forma. Conviene observar que con cualquiera de las dos definiciones el 1 queda excluido del conjunto de los números primos (mimosa, 2019).



El primer paso será realizar todas las acciones que necesitaremos ejecutar en nuestra estructura de dato, para este problema se decidió hacer uso de la pila, entre ellas están crear una pila, agregar elementos a una pila (push), quitar elementos de la pila (pop) y saber cuál es la sima de la pila (top).



El segundo paso será hacer hechos que permitan sacar el sucesor y el antecesor de un numero para algunos otros hechos que se usaran en el programa.



El tercer paso será poder obtener el resto entre un numero “A” y un numero “B” y almacenando su resto en un variable “C”.



El cuarto paso será hacer un hecho que permita determinar si un numero “O” es múltiplo de alguno otro número “N”, ya que si recordamos un numero es primo si es solo divisible por uno y por si mismo. Para contestar esta pregunta sabemos que si el “resto” de un numero “A”, con otro numero “B” es 0, implica que “A” y “B” son múltiplos, en cualquier otro caso no son múltiplos, el proceso consistirá en determinar si existe algún antecesor al numero “O” que sea múltiplo, siendo “N” los antecesores a “O”, como “N” siempre decrece si llega a ser la unidad significa que solo era múltiplo por si mismo y por 1, en tal caso guardaremos en una variable “D” un 1, por el contrario si “N” no es la unidad sacaremos el resto de “O” con “N” y su resto será almacenado en “R”, a continuación preguntaremos si “R” es 0,si se cumple este caso significa que “O” y “N” son múltiplos por lo cual asignaremos a “D” un 0 y detendremos el proceso, en el último caso de que “N” no sea ni la unidad ni un múltiplo repetiremos el mismo proceso recursivamente con el antecesor de “N” hasta que “N” sea la unidad o un múltiplo de “O “.



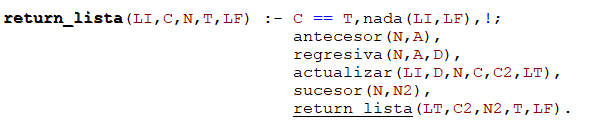
El quinto paso será crear un hecho que recibirá una pila y no le hará ningún cambio.



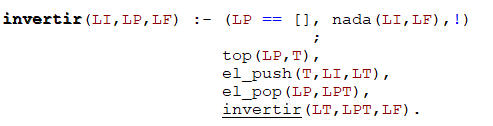
El sexto hecho será para actualizar una pila en base a la variable “D” obtenida en el cuarto paso, este hecho recibirá una pila guardada en “LI”, la variable “D” del cuarto paso, un numero “N”, una variable “C” que lleva la cantidad de elementos en la pila, “NC” será la nueva cantidad de elementos en la pila que se mantendrá o aumentara en uno dependiendo de la variable “D” del cuarto paso, y la variable “LF” que almacenar la nueva pila que saldrá de este hecho. En este hecho el numero “N” y el numero “O” son iguales, recordando el cuarto paso retornamos un 1 en “D” si “O” era primo, en caso contrario retornaba 0, en caso de que “D” sea 1 agregaremos “N” en la pila “Li”, siendo “LF” el resultado se agregar “N” en la pila “LI” y como la pila tiene un elemento más debemos sumar uno a la cantidad de elementos en la pila, la nueva cantidad de elementos será el contador actual, es decir “C” más uno y este nuevo contador será registrado en “NC”, en caso de que “N” no sea primo es decir “D” es 0 , no le haremos ningún cambio a la pila y “NC” será “C” más cero.



El séptimos paso será crear un hecho que regresara una pila con los numero primos haciendo uso de los seis paso anteriores, para esto necesitaremos cinco parámetros una pila “LI”, que comenzara estando vacía la primera vez que se ejecute este hecho, una variable “C” que tiene la cantidad de elementos en la pila “LI”, una variable “N” que será el numero a evaluar en el cuarto paso, una variable “T” que son la cantidad de numero primos que se quiera en la pila y una variable “LF” que será donde se guarda la pila con “T” números primos. Lo primero que se hará al ejecutar el hecho será preguntar si la cantidad de elementos en la pila, es decir “C” es igual a la cantidad de primos “T” que quiere el usuario son iguales, en caso de serlo no le hacemos nada a la pila y terminamos el proceso, de no cumplirse que “C” y “T” sean iguales obtendremos el antecesor de “N”, y entraremos al cuarto paso dando de parámetros “N” y el antecesor de “N”, es decir “N – 1”, seguido entramos al sexto paso actualizando la pila, a continuación obtendremos el sucesor de “N” y llamaremos recursivamente al mismo hecho hasta que la cantidad de elementos en la pila sea igual a la cantidad de números primos.



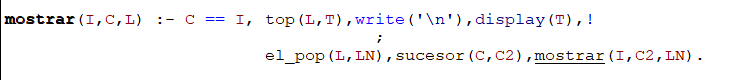
El octavo paso será invertir la pila con los numero primos obtenido en el paso anterior, para conseguir esto necesitaremos tres parámetros una pila “LI” que al principio estará vacía, “LP” que será la pila con los numero primos del paso anterior y una variable “LF” que será la nueva pila invertida, el proceso consiste en preguntar si la pila con numero primos esta vacía en caso de estarlo, no le haremos ningún cambio a la pila “LI” y “LF” será igual a esta pila y terminara el proceso, por el contrario si la lista “LP” no esta vacía obtendremos la sima de la pila “LP” y agregaremos ese elemento en la pila “LI”, seguido quitaremos la sima de la pila “LP” y llamaremos recursivamente con esta nueva pila de números primo hasta que este vacía.



El noveno paso es crear un hecho que pueda evitar un error si se desea proyectar un índice que no existe en la pila, Para esto usaremos “I” que es la posición a mostrar y “NP” que es la cantidad de numero primos que quería el usuario en la pila. Como el índice empieza desde el cero todos los naturales menores a cero no deberían existir por ende si “I” es menor a cero mostraremos un mensaje de error o si también quiere mostrar una posición mayor a la cantidad de numero primos en la pila mostraremos el mismo mensaje y por ultimo como en este programa considera el cero es decir un numero menos de la unidad también debemos restar la unidad a “NP” por ende si es mayor o igual a “NP” también mostraremos un mensaje de error.



El decimo paso se ejecuta solo si no ocurrió un error en el paso anterior, en este paso será necesario la variable “I” para saber que índice queremos mostrar, una variable “C” que será un contador que aumentara hasta que sea igual a “I” y la pila con los numero primos. Preguntaremos si el contador “C” es igual al índice “I”, en ese caso mostraremos la sima de la pila y terminaremos el programa. En caso contrario quitaremos la sima de la pila, aumentaremos el contador en uno y llamaremos recursivamente al mismo hecho hasta que el contador sea igual a “I”.



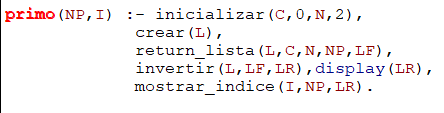
El onceavo paso sirve para mostrar un índice de la pila invertida, por lo cual guardaremos en “I” la posición que el usuario quiere proyectar, “NP” es la cantidad de números primos que quería el usuario en la pila, este parámetro servirá para evitar errores los cuales se mostraron en el noveno paso, en caso de no existir errores inicializaremos una variable “C” para llevar un contador en el hecho “mostrar”.



El doceavo paso será hacer un hecho que permite inicializar dos variables que se usaran cuando comience el programa. El objetivo de este hecho es iniciar un contador con la cantidad de elementos en la pila y el primer número primo a evaluar.



El treceavo paso seria Juntar todos los pasos anteriores en un solo hecho, es decir inicializar el contador de elementos en la pila, inicializar el primer numero de los numero primos, es decir el 2, generar “NP” números primo, invertir la pila y mostrar el índice “I” de la pila.



Conclusión

El paradigma lógico es muy eficiente al basarse en la lógica de primer orden, es decir basarse el ocurrencia o en la no ocurrencia de los hechos establecidos en la base de conocimiento creada por los programadores como lo fue demostrado en los ejercicios anteriores, Prolog era capas determinar si se cumplían o no las condiciones propuestas a través de las simples de respuestas de True o False incluso siendo capas de consultar la base de conocimiento y encontrar nuevo datos a partir de otros preexistentes.