Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–3661 – Laboratorio de Lenguajes de Programación.
Septiembre–Diciembre 2012

# Proyecto II: Bacon Pancakes!

Después de un largo día de Aventuras, Jake el perro y Finn el humano se proponen preparar sus famosos *Bacon Pancakes* (Panquecas de Tocineta). Pero hay un problema. ¡No quedan tocinetas! Y el único lugar en todo el mundo donde pueden conseguirse más es en el lejano mercado astral, en la cumbre de la gran montaña de Skolem.

Cruzando el bosque de los predicados perdidos, más allá del desierto de la unificación, luego de trepar la gran montaña, finalmente llegan Finn y Jake a las puertas del antigüo mercado. Hacen un intento por abrir las puertas, pero no ceden. Siguen intentándolo, cada vez con más fuerza, pero aún sin éxito alguno. Repentinamente, su frustración es interrumpida por una pequeña criatura computista de la aldea de MYS. La criatura se acerca a ellos y les dice que el mercado ha estado cerrado por meses, por que el sistema que maneja las puertas arrojó un *Segmentation Fault* y no han podido arreglarlo. Las criaturas computistas, guardianes del mercado astral, son excelentes programadores, pero solo saben programar en el lenguaje `Brainf**k` . El mercado es mucho más antigüo que ellos y fue programado por los mismos Dioses; y por tanto, solo puede entender programas escritos en el antiguo y olvidado lenguaje: `ProLog`.

Finn y Jake no saben nada de computadoras ni programación, pero si conocen a alguien que puede haber vivido lo suficiente como para haber conocido a los antiguos Dioses: ¡El elefante guerrero psíquico ancestral! Después de varios días de búsqueda, logran encontrarlo y piden su ayuda. El elefante no conoce de programación tampoco pero, siendo psíquico, logra comunicarse con los Dioses y les pide consejo. Los Dioses escuchan la plegaria de su amigo elefante y deciden que debe implementarse entonces un intérprete de `Brainf**k` en `ProLog`, dejando así el mercado en manos de sus guardianes. Pero siendo Dioses tan ocupados, no tienen tiempo para tales proezas de implementación, por lo que dejan tal tarea a sus hábiles estudiantes del Laboratorio de Lenguajes de Programación.

## Definición del Lenguaje: `Brainf**k`

`Brainf**k` es muy similar a una Maquina de Turing y está diseñado para tener un conjunto muy pequeño de instrucciones sencillas. Se tiene una cinta de tamaño $K$, llena completamente de ceros. Al iniciar un programa, se tiene un apuntador a la primera posición de esa cinta. A partir de ese estado inicial, las posibles instrucciones que tiene el lenguaje son las siguientes:

- **Incremento:** Aumenta en 1 el valor de la casilla apuntada actualmente. Se escríbe usando únicamente el símbolo `'+'` (sin las comillas).
- **Decremento:** Disminuye en 1 el valor de la casilla apuntada actualmente. Se escríbe usando únicamente el símbolo `'-'` (sin las comillas).
- **Avance:** Aumenta en 1 el valor del apuntador (ahora apunta a la siguiente casilla). Se escríbe usando únicamente el símbolo `'>'` (sin las comillas). *Nota: Si se está en la última casilla de la cinta, esta instrucción no tiene efecto alguno.*
- **Retroceso:** Dsiminuye en 1 el valor del apuntador (ahora apunta a la casilla anterior). Se escríbe usando únicamente el símbolo `'<'` (sin las comillas). *Nota: Si se está en la primera casilla de la cinta, esta instrucción no tiene efecto alguno.*

- **Lectura:** Lee un caracter de la entrada y almacena, en la casilla apuntada actualmente, su valor ASCII. Se escríbe usando únicamente el símbolo `‘,’` (sin las comillas).
- **Escritura:** Imprime el valor de la casilla apuntada actualmente a la salida, interpretado como un caracter ASCII. Se escríbe usando únicamente el símbolo `‘.’` (sin las comillas).
- **Iteración:** Si el valor de la casilla apuntada actualmente es cero (0), entonces la instrucción no tiene efecto alguno. De lo contrario, se ejecuta la cadena de instrucciones que encierra y al finalizar vuelve a consultar el valor de la casilla apuntada. Se escríbe usando la sintaxis `‘[` $<Instrs>$ `]’` (sin las comillas), donde $<Instrs>$ es la cadena de instrucciones a ejecutar potencialmente.
- **Comentarios:** Cualquier otro caracter que no esté listado anteriormente es considerado un comentario y, por tanto, debe ser ignorado.

## Representación de Estados

Un estado estará conformado por una cinta y un apuntador. Dicho apuntador puede representarse abstractamente, separando la cinta en tres partes: La sección de cinta anterior al apuntador, la casilla apuntada y la sección de cinta posterior al apuntador. Cada sección de cinta se representará concretamente como una lista. En `ProLog` podemos describir un estado con un funcional de aridad 3:

```
estado(Anterior, Actual, Posterior)
```

Donde `Anterior` es la lista que representa la cinta anterior al apuntador, `Actual` es un valor que representa la casilla apuntada y `Posterior` es la lista que representa la cinta posterior al apuntador.

## Predicados Auxiliares

A continuación se listan los predicados que debe implementar, que serán luego invocados desde el programa principal.

- `ceros/2:` Si el primer elemento es $K$, el segundo debe unificar a una lista de tamaño $K - 1$ completamente llena de ceros (0).
- `ejecutar/4:` Su primer argumento es el nivel en el que se está ejecutando el programa (cantidad de caracteres `‘[’` menos cantidad de caracteres `‘]’`, ambos encontrados antes de la instrucción ejecutándose actualmente). Puede suponer que los programas en `Brainf**k` pasados están bien escritos, por lo que el nivel siempre será no–negativo. El segundo argumentos es un estado inicial, el tercero una lista de instrucciones bien formada y el cuarto el estado final, corresponiendte a ejecutar las instrucciones propuestas sobre el estado inicial.
- `ejecutar/2:` Su primer argumento es la capacidad/tamaño de la cinta a utilizar y el segundo argumento es una lista de instrucciones bien formada. Este predicado debe inicializar una cinta con la capacidad propuesta y ejecutar las intrucciones suministradas (Nótese que el estado final es descartado en este predicado, solo interesarán los efectos de borde).
- `cargar/2:` Su primer argumento átomo correspondiente a un nombre de archivo. El segundo argumento debe entonces unificar a una lista de caracteres con todo el contenido de dicho archivo.

Puede implementar predicados adicionales si lo cree necesario; sin embargo los predicados descritos anteriormente deben estar implementados con la cantidad de argumentos y comportamientos especificados.

## Cliente: `brainfk/0`

El programa principal será un predicado de aridad 0 (no recibe argumentos) que debe realizar las sigiuentes acciones:

- Preguntar al usuario por un nombre de archivo.
- Abrir y leer el contenido de dicho archivo.
- Ejecutar el programa en `Brainf**k` contenido en el mismo (teniendo cuidado de restaurar la entrada/salida estándar antes de comenzar la ejecución).

El predicado cliente estará en el mismo archivo que los predicados auxiliares, de nombre `bacon.pl`.

## Detalles de la Entrega

La entrega del proyecto consistirá de un único archivo `p2g<num>.tar.gz`, donde `<num>` es el número asignado a su equipo. Tal archivo debe ser un directorio comprimido que contenga únicamente los siguientes elementos:

- El archivo `bacon.pl` con los predicados descritos anteriormente.
- Un archivo `Readme` con los datos de su equipo (integrantes, carné, etc.) y los detalles relevantes de su implementación.

El proyecto deberá ser entregado al prof. Carlos Perez *unicamente* a su dirección de correo electrónico oficial: ( caperez@ldc.usb.ve ) a más tardar el Jueves 29 de Noviembre, a las 11:59pm. VET.

Para más información sobre el lenguaje `Brainf**k` pueden consultar su entrada en el Wiki de lenguaje esotéricos *esolangs*: http://esolangs.org/wiki/Brainfuck.

El soundtrack oficial del proyecto lo pueden encontrar en: http://www.youtube.com/watch?v=1eO5U_uN7DQ.