Intérprete sencillo utilizando Java, Jlex y Cup

En los cursos de compiladores de la universidad, es bastante común que se solicite al estudiante desarrollar un intérprete, una herramienta que reciba como entrada cierto lenguaje de programación y lo ejecute, pero la mayoría de documentación al respecto solo muestra ejemplos de cosas sencillas, como una calculadora o un lenguaje que imprime cadenas en consola. Pero qué pasa si lo que deseamos es que se ejecuten sentencias de control como el IF o ciclos como la sentencia WHILE y que además estas sentencias soporten muchos niveles de anidamiento, que se declaren variables y se asigne valores a estas variables, que se tenga control de los ámbitos de las variables, en fin, que tenga las funciones básicas de un lenguaje de programación. No es común encontrar este tipo de ejemplos, en lo personal, puedo asegurar que nunca encontré un tutorial en el que se mostrara un ejemplo documentado y bien explicado sobre esto. Es por eso que les traigo este ejemplo, espero que les sea útil.

Funcionamiento de la aplicación

En este tutorial se desarrolla un intérprete que recibe como entrada un archivo de texto que contiene varias sentencias en un lenguaje programación diseñado especialmente para esta aplicación, primero se hace análisis léxico y sintáctico de dicha entrada, durante el análisis sintáctico se carga en memoria un Árbol de Sintaxis Abstracta (AST) que se utiliza posteriormente para ejecutar las sentencias. Los analizadores se generan con Jlex y Cup. Para desarrollar el proyecto se utilizó Ubuntu 14.04 y Netbeans 8.0. El proyecto completo del ejemplo puede descargarse del siguiente enlace:

Intérprete sencillo utilizando Java, Jlex y Cup

Todo el código dentro del proyecto está documentado con comentarios que contienen explicaciones sobre su funcionamiento.

Si desean, una pequeña introducción al uso de Jlex y Cup pueden visitar mi post: Mi primer proyecto utilizando Jlex y Cup.

El lenguaje de entrada

Dentro de la carpeta del proyecto, hay un archivo de entrada llamado "entrada.txt", en él se muestran ejemplos de todas las funciones del lenguaje diseñado para esta aplicación, al leerlo se puede tener una idea clara de las funciones con las que el lenguaje cuenta, este archivo contiene lo siguiente:

```
/*****************************************
 * Ejemplo desarrollado por Erick Navarro *
 * Blog: e-navarro.blogspot.com           *
 * Septiembre - 2015                      *
 *****************************************/

//Se imprime el encabezado
imprimir("Tablas de" & " multiplicar");

//Se declara la variable a, de tipo numero
numero a;
//Se asigna a la variable a el valor 0
a=0;
//Se declara la variable c, de tipo numero
numero c;
//Se asigna a la variable c el valor 0
c=1;
//Se imprime un separador
imprimir("----------------");
/**
```

Como se puede observar, el lenguaje acepta:

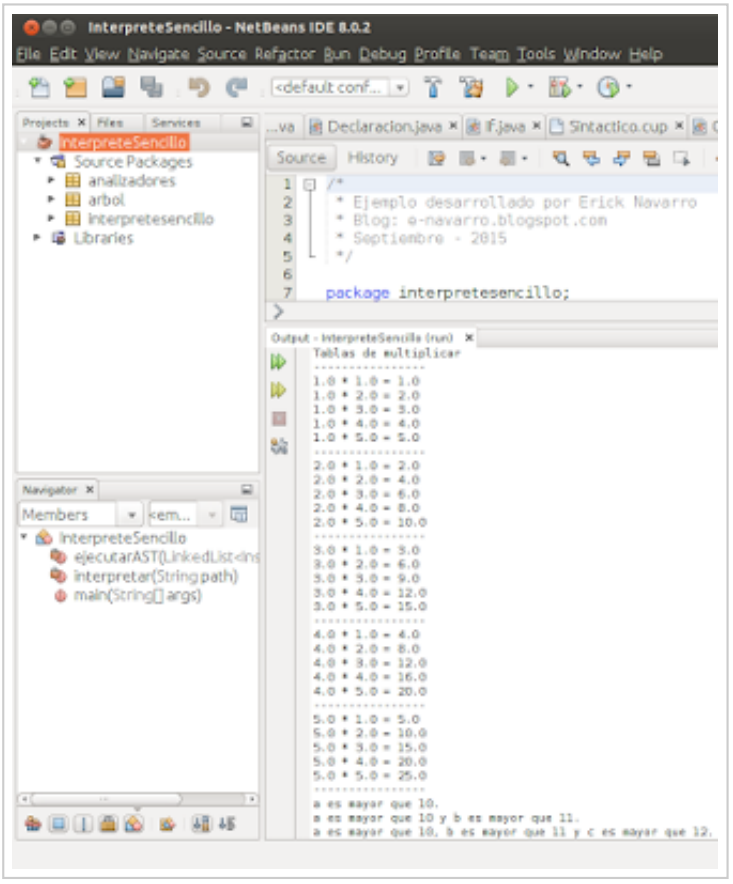
- Comentarios de muchas líneas (/*<Contenido del comentario>*/).
- Comentarios de una línea (//<Contenido del comentario>).
- Concatenación de cadenas, mediante el operador "&".

- Función "imprimir": que recibe como parámetro una cadena e imprime en consola dicha cadena.
- Declaración de variables: el único tipo de variables que el lenguaje soporta es "numero", que es una variable de tipo numérico que suporta números enteros o con punto decimal (Dentro del rango del tipo Double de Java).
- Asignación de variables, a cualquier variable se le puede asignar cualquier expresión que tenga como resultado un número.
- Instrucción "mientras": tiene el comportamiento clásico del ciclo while, ejecuta el ciclo mientras la expresión booleana que recibe sea verdadera. Esta instrucción soporta anidamiento.
- Instrucción "if" e instrucción "if-else": si la expresión booleana que recibe es verdadera entonces ejecuta las instrucciones contenidas en el "if", si es falsa y la instrucción tiene un "else" entonces se ejecutan las instrucciones contenidas en el "else". Esta instrucción soporta anidamiento.
- Expresiones aritméticas: Estas expresiones soportan sumas, restas, divisiones, multiplicaciones, expresiones negativas y paréntesis para agrupar operaciones. Tiene la precedencia habitual de las expresiones aritméticas.
- Expresiones booleanas: comparan dos expresiones que tengan como resultado un número y soportan únicamente los operadores mayor que y menor que (<, >).

El resultado de la ejecución

Al ejecutar el archivo de entrada mostrado anteriormente se obtiene el siguiente resultado en consola:

```
run:
Tablas de multiplicar
----------------
1.0 * 1.0 = 1.0
1.0 * 2.0 = 2.0
1.0 * 3.0 = 3.0
1.0 * 4.0 = 4.0
1.0 * 5.0 = 5.0
----------------
2.0 * 1.0 = 2.0
2.0 * 2.0 = 4.0
2.0 * 3.0 = 6.0
2.0 * 4.0 = 8.0
2.0 * 5.0 = 10.0
----------------
3.0 * 1.0 = 3.0
3.0 * 2.0 = 6.0
3.0 * 3.0 = 9.0
3.0 * 4.0 = 12.0
3.0 * 5.0 = 15.0
```



Sobre la tabla de símbolos

La tabla de símbolos es una parte importante en el proceso de ejecución del código, es en esta estructura de datos en donde guardamos información de las variables como su tipo, identificador y valor. A esta estructura podemos pedirle el valor de una variable, o pedirle que le asigne cierto valor a una variable.
Es importante mencionar que en el proceso de ejecución la tabla de símbolos va cambiando de forma dinámica, esto con el objetivo de manejar los ámbitos, por ejemplo, la instrucción WHILE tiene su propio ámbito, lo que significa que su tabla de símbolos contiene información de las variables declaradas en ámbitos superiores y la información de las variables declaradas en el ámbito local de la instrucción, al terminar de ejecutar la instrucción, todas las variables declaradas en el ámbito local se eliminan de la tabla de símbolos que almacena la información de los ámbitos superiores, de tal manera que los ámbitos superiores no tendrán acceso a las variables declaradas dentro del WHILE.

La magia detrás de todo esto: Árbol de sintaxis abstracta (AST)

Un árbol de sintaxis abstracta (AST) es una representación simplificada de la estructura sintáctica del código fuente. A nivel de programación un AST es una estructura de datos que se genera durante el proceso de análisis sintáctico.
En este ejemplo el AST es la pieza más importante porque al recorrerlo pueden ejecutarse las acciones del código de entrada y ese es el principal objetivo de la aplicación.
En el código fuente de Cup se observa que la mayoría de las acciones se enfocan en cargar el AST, básicamente es lo único que hace el analizador, además de verificar que la sintaxis de la entrada sea correcta
La estructura en este caso es un tanto compleja ya que cada nodo puede tener muchos hijos, en el caso de las instrucciones IF-ELSE y WHILE, el número de hijos es incierto ya que estas instrucciones pueden contener muchas otras instrucciones dentro, lo cierto es que el árbol se acopla muy bien al lenguaje de programación porque en el árbol se tiene bien claro qué instrucciones están contenidas dentro de otras instrucciones, porque cada nodo esta directamente ligado a sus hijos, entonces la ejecución de instrucciones anidadas no representa mayor problema.
Hacemos análisis sintáctico una sola vez para cargar el árbol, posteriormente recorremos ese árbol para ejecutar el código.
El árbol es una representación exacta de lo que el código de entrada contiene. Las únicos tres paquetes del proyecto son:

- analizadores: que contiene los archivos de Cup y JLex y los analizadores que con estas

herramientas se generaron.

- arbol: que contiene todas las clases que forman parte del AST, que se utiliza como estructura primaria en la aplicación.
- interpretesencillo: que contiene la clase principal de la aplicación.