Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Diciembre - Marzo 2014-2015

Setlan - Etapa II

Análisis Sintáctico y Árbol Sintáctico Abstracto (AST)

Especificación de la entrega

En la primera etapa de desarrollo del interpretador para el lenguaje `Setlan` se implementó el analizador lexicográfico que permite reconocer todos y cada uno de los token que componen un programa de `Setlan`, o que se detiene en caso de conseguir un error lexicográfico en éste. Este segundo paso corresponde al diseño de la gramática libre de contexto del lenguaje y la implementación de un reconocedor para ella. Además, durante el reconocimiento, se deberá crear el Árbol Sintáctico Abstracto (AST) e imprimirlo de forma legible por salida estándar.

Su gramática debe ser lo suficientemente completa para que puedan ser reconocidas diferentes combinaciones de expresiones e instrucciones en distintos programas. Para esta segunda entrega *no es necesario* que sean reportados errores de tipos o variables no declaradas. Su analizador sintáctico deberá *sólo* verificar la correctitud de la sintaxis del programa.

Para la construcción del AST deberán crear las clases necesarias para la representación de cada instrucción y expresión del lenguaje. Válgase del uso de tipos recursivos de datos y herencia de clases. El árbol sintáctico abstracto deberá ser impreso por salida estándar cuando sea analizado un programa *sin errores*.

Ejecución

Para la ejecución del interpretador su programa deberá llamarse `setlan` y recibirá como primer argumento el nombre del archivo con el código en `Setlan` a analizar.

Primero se hace el análisis lexicográfico, donde en caso de haber errores, éstos se deben reportar al igual que en la primera entrega y detener la ejecución; si el análisis lexicográfico no tiene errores, se usa la lista de *tokens* generada por éste para el análisis sintáctico.

Por salida, se debe mostrar el árbol abstracto sintáctico siguiendo el modelo de impresión a continuación. Note que ya **no** se deberá imprimir la lista de *tokens* que se imprimía en la primera entrega.

En caso de encontrar un error de sintaxis se reportará por salida estándar **sólo el primer error encontrado**, indicando el *token* causante del problema con su número de fila y columna respectivo. A pesar de que las herramientas permitidas para el desarrollo del interpretador de `Setlan` dispongan de recuperación de errores sintácticos, no es el objetivo para este curso. No implementen recuperación de errores sintácticos.

Al encontrar el primer error, se deberá abortar la ejecuión.

Ejemplo de un programa correcto en `Setlan`:

```
program {
    using
        set s;
        int a,b;
    in
    s = {1,1,2,3,5,8};
    a = 2;
    b = a*3;

    for i min s do {
        println i*a;
        a = a+b;
    }
}
```

y su respectiva salida correspondiente:

```
# El AST será publicado más adelante
```

Un ejemplo de programa con errores en su código:

```
program {
    usign
        int a,b,c;
        int x,y;
    in

    x = b*b - 2*a*c;
    y = ;

    if (x >= 0)
        println "Raiz real", y
    else
        println "Raiz imaginaria", y
    ;
}
```

y su salida correspondiente:

```
Error de sintaxis: Token ';' inesperado en línea 8, columna 8.
```

Implementación

Para la implementación del interpretador del lenguaje `Setlan`, pueden escoger uno (1) de los tres (3) lenguajes de programación a continuación. Para cada uno de ellos se indica las herramientas disponibles para el desarrollo de un interpretador de código. Recuerden que para esta etapa de desarrollo deberán seguir empleando **el lenguaje de**

programación utilizado en la primera etapa así como deberán seguir con él en etapas posteriores.

- *Python:*
  - Interpretador *python* 2.7.
  - *Python Lex-Yacc (PLY).* Para esta primera etapa de desarrollo utilizarán el submódulo **Lex** de *PLY*. Sin embargo, el submódulo **Yacc** será empleado en siguientes etapas.
- *Ruby:*
  - Interpretador *ruby* 1.9.
  - Para esta etapa de desarrollo no existe una herramienta en *Ruby* que permita el análisis lexicográfico de un archivo de determinado código, por lo tanto, el trabajo para esta entrega se debe realizar a través del manejo de las expresiones regulares del lenguaje. Para entregas posteriores se utilizará *Racc*.
- *Haskell:*
  - Compilador *ghc* 7.6.3 ó 7.8.3.
  - *Alex* y *Happy* . Para esta etapa de desarrollo utilizarán el generador de analizadores lexicográficos *Alex*. Posteriores entregas requerirán del *Happy* para el analizador sintáctico.

Formato de Entrega:

Deben enviar un correo electrónico a **todos los preparadores** con el asunto: `[CI3725]eXgY` donde `X` corresponde al número de la entrega e `Y` al número del equipo. El correo debe incluir un archivo comprimido **.zip** con el nombre `eXgY.zip` siguiendo los valores para el asunto del correo; este **.zip** debe contener:

- Código fuente debidamente documentado.
- En caso de utilizar Haskell, deben incluir un archivo Makefile. Si su proyecto no compila, el proyecto no será corregido.
- Un archivo con el nombre `gramatica.txt` donde se especifique la gramática libre de contexto propuesta por el equipo.
- Un archivo de texto con el nombre `LEEME.txt` donde **brevemente** se expliquen:
  - Estado actual del proyecto.
  - Problemas presentes.
  - Cualquier comentario respecto al proyecto que consideren necesario.
  - Este archivo debe estar identificado con los nombres, apellidos y carné de cada miembro del equipo de trabajo.

Fecha de entrega:

La fecha límite de entrega del proyecto es el día **viernes 06** de febrero de 2015 (semana 7) *hasta* las **11:50pm**, entregas hechas más tarde tendrán una **penalización del 20%** de la nota. Esta penalización aplica por cada día de retraso.