Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3725 - Traductores e Interpretadores

Bitiondo - Etapa II

Análisis Sintáctico y Árbol Sintáctico Abstracto (7%)

Especificación de la entrega

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

La gramática a desarrollar 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 estructuras de datos 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 `bitiondo` y recibirá como primer argumento el nombre del archivo con el código en *Bitiondo* 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 *Bitiondo* 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 *Bitiondo*:

```
begin
    int a;
    bits c[4] = 0b0001;

    a = 2;
    c = c << a;
    outputln c;
    c = c >> 2;
    outputln c, "Done!";
end
```

y su respectiva salida correspondiente:

```
BEGIN
    DECLARE
        type: int
        variable: a
    DECLARE
        type: bits
        variable: c
        size:
            const_int: 4
        value:
            const_bits: 0b0001
    ASSIGN
        variable: a
        value:
            const_int: 2
    ASSIGN
        variable: c
        value:
            BIN_EXPRESSION:
                operator: LEFTSHIFT
                left operand:
```

```
                    variable: c
                right operand:
                    variable: a
    OUTPUTLN
        elements:
            variable: c
    ASSIGN
        variable: c
        value:
            BIN_EXPRESSION
                operator: RIGTHSHIFT
                left operand:
                    variable: c
                right operand:
                    const_int: 2
    OUTPUTLN
        elements:
            variable: c
            string: "Done!"
END
```

Nota: estos ejemplos de salida **no** son un formato obligatorio. Sólo se espera que su salida sea de fácil lectura y la impresión haga entendible la estructura del programa. Sí es importante que el AST impreso por su interpretador mantenga una "estructura de árbol" como la aquí mostrada.

Un ejemplo de programa con errores en su código:

```
begin
    int a = 2;;
    bool b = 3;

    output a == 3 && b;
end
```

y su salida correspondiente:

```
ERROR: unexpected token ';' at line 2, column 16
```

Implementación

Para la implementación de esta etapa del interpretador del lenguaje `Bitiondo`, es **requerido que se continúe el desarrollo en el lenguaje de programación escogido para la primera etapa**. Así como debe seguirse usando para etapas posteriores.

- *Ruby:*
  - *ruby* 2.3.0

    - Para esta etapa de desarrollo se utilizará el generador de parsers *Racc*.
- *C++:*
    - Usando *GCC* 5.3.1
    - Para esta etapa de desarrollo se utilizará el generador de parsers *Bison*.
- *Haskell:*
    - *GHC* 7.10.3
    - Para esta etapa de desarrollo se utilizará el generador de parsers *Happy*.

Formato de Entrega:

- Código fuente debidamente documentado.
- En caso de utilizar Haskell o C++, deben incluir un archivo Makefile o un archivo de configuración para *Cabal*. **Si su proyecto no compila, no será corregido**.
- Un archivo de texto con el nombre `README.md` donde **brevemente** se explique:
    - Decisiones de implementación
    - 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.

Para la fecha de la entrega, se revisará el último *commit* de fecha y hora previas a la fecha y hora límite, con la etiqueta `v0.2.x` donde `x` es de libre uso por los integrantes del equipo para sus versiones internas.

En caso de que no existir etiquetas, se usará el último *commit* de fecha y hora previas a la fecha y hora límite.

Fecha de entrega:

La fecha límite de entrega del proyecto es el día **miércoles 8** de noviembre de 2017 (semana 8) *hasta* las **11:50pm**, entregas concretadas más tarde tendrán una **penalización del 20%** de la nota, esta penalización aplica por cada día de retraso. En caso de que deseen que se revise un *commit* que incurra en penalizaciones, deben hacerlo saber por correo a cualquiera de los encargados de la materia.