Teoría de Lenguajes
Segundo Obligatorio - 2017

El propósito del obligatorio es construir una gramática para un lenguaje que se describirá a continuación. Para esto se utilizará un programa Python entregado por el equipo docente y la biblioteca NLTK. Con dicha biblioteca podemos a partir de una especificación de una gramática libre de contexto generar un parser, el cual permite determinar si una tira (o texto) es reconocido por dicha gramática.

Presentación del Lenguaje

Para presentar el lenguaje se utilizará una gramática escrita en EBNF. Son símbolos del metalenguaje:

*	indica cero o más repeticiones
+	indica una o más repeticiones
|	indica OR
?	indica 0 o 1 ocurrencia
()	se usa para agrupar dos o más términos a los que se le aplica un operador.

Los símbolos terminales se describen entre comillas y en negrita.

```
program ::= "program" "identif" ";" declarations "begin" statementSequence "end" "."
declarations ::= constants ? types ? variables ?
constants ::= "const" "identif" "=" constant ";" ( "identif" "=" constant ";" ) *
types ::= "type" "identif" "=" type ";" ( "identif" "=" type ";" ) *
variables ::= "var" identifList ":" type ";" ( identifList ":" type ";" ) *
statementSequence ::= statement ( ";" statement ) *
statement ::= variable ":=" expression
            | "while" expression "do" statement
            | "repeat" statementSequence "until" expression
            | "for" variableIdentif ":=" expression ("to" | "downto") expression "do" statement
            | "begin" statementSequence "end"
            | "if" expression "then" statement ( "else" statement ) ?
expression ::= simpleExpression ( relationalOperator simpleExpression ) ?
simpleExpression ::= ( "+" | "-" )? term ( additionOperator term ) *
term ::= factor ( multiplicationOperator factor ) *
factor ::= variable | "unsignednumber" | "(" expression ")" | "not" factor
relationalOperator ::= "=" | "<>" | "<" | "<=" | ">" | ">="
additionOperator ::= "+" | "-" | "or"
multiplicationOperator ::= "*" | "/" | "div" | "mod" | "and"
variable ::= variableIdentif | variableArray
variableIdentif ::= "identif"
variableArray ::= "identif" "[" expressionList "]"
expressionList ::= expression ( "," expression ) *
type ::= simpleType | arrayType | "identif"
simpleType ::= constant ".." constant | "(" identifList ")"
arrayType ::= "array" "[" simpleType ( "," simpleType ) * "]" "of" type
identifList ::= "identif" ( "," "identif" ) *
constant ::= ("+" | "-" ) ? ("identif" | "unsignednumber")
```

Construcción del reconocedor gramatical

Se le entregará al estudiante un programa que contendrá el main del programa, código para reconocer los símbolos y la invocación al reconocedor gramatical.

El estudiante deberá completar el programa con la gramática libre de contexto.

En este al referirse a los terminales **deberá usar** los siguientes nombres, pues son los que se reconocen previamente en el reconocedor de símbolos por el programa main.py.

Terminal	Símbolo para la gramática
"program"	PROGRAM
"begin"	BEGIN
"end"	END
"const"	CONST
"type"	TYPE
"var"	VAR
"."	DOT
".."	DOTDOT
";"	SEMICOLON
":"	COLON
","	COMMA
"="	EQUAL
"<>"	DIFFERENT
"<"	LT
">"	GT
"<="	LTE
">="	GTE
":="	ASSIGN
"+"	ADD
"-"	SUB
"or"	OR
"*"	ASTER
"/"	BARRA
"div"	DIV
"mod"	MOD
"and"	AND
"while"	WHILE
"do"	DO
"repeat"	REPEAT
"until"	UNTIL
"for"	FOR
"to"	TO
"downto"	DOWNTO
"if"	IF
"then"	THEN
"else"	ELSE
"("	OPAR
")"	CPAR
"["	OSQRBRA
"]"	CSQRBRA
"not"	NOT
"array"	ARRAY
"of"	OF
"ident"	IDENT
"number"	NUMBER

Los siguientes terminales (o tokens):

- **"ident"** representa un identificador de la siguiente forma: [a-zA-Z] ([a-zA-Z]|[0-9])*
- **"number"** representa un número de la siguiente forma: [0-9]+

El reconocedor gramatical deberá trabajar directamente con IDENT y NUMBER (estos habrán sido reconocidos ya por el reconocedor de símbolos).

Entorno de trabajo

Se entregará un archivo comprimido (.zip) con un directorio que contienen lo siguiente:

- Los archivos con las salidas "oficiales" para cada uno de los archivos de entrada donde se indica cuales deberán ser esas salidas (t1.out, t2.out, etc.).
- Un programa Python (main.py) contiene el analizador léxico e invoca al analizador sintáctico (o parser) que se pide realizar. Este programa deberá quedar intacto.
- El esqueleto del código fuente del analizador sintáctico (o parser) que se debe desarrollar en el laboratorio.
- Varios scripts (*.bat) para facilitar las tareas de ejecutar y comparar las salidas.
- El programa diff.exe para comparar las salidas.

Desarrollo del trabajo

Los trabajos se realizarán en grupos de 2 a 4 personas.

Se sancionará con la pérdida del obligatorio si se detectan trabajos copiados, o si se hace una mala utilización del EVA.

Entrega

Se realizará vía web, se comunicarán los detalles oportunamente. La misma **se habilitará un par de días antes del vencimiento, que es el día jueves 22 de junio a las 23:55 hs.**

Se deberán entregar los siguientes 2 archivos:

- **grammar.txt**
- **integrantes.txt** (contendrá la CI (7 dígitos) y el nombre de cada uno de los integrantes del grupo (sin usar tildes), y a continuación si desean hacer algún comentario sobre la ejecución.

Formato de ejemplo del archivo integrantes.txt:

```
3123456, Gonzalez, Pablo
4567123, Martinez, Veronica
3444555, Garcia, Pablo

Comentarios, solo si para algún test no funcionó bien.
```