Tecnológico de Monterrey

Tarea 8
Diseño de Compiladores

Esta tarea debe ser entregada vía Blackboard el martes 21 de mayo de 2019 antes de la media noche.

Problema 1

El objetivo del ejercicio es que construyan un intérprete de lenguaje descrito más abajo usando bison y flex. Su intérprete tendrá dos partes, un reconocedor sintáctico y un intérprete del lenguaje. El intérprete debe hacerse recorriendo el árbol sintáctico reducido generado por el reconocedor sintáctico.

Lo que en la gramática aparece en ***negritas*** son los símbolos terminales, y obviamente se refiere a lo que debe reconocer el reconocedor léxico. Las expresiones regulares que determinan **id** y **intnum** y **realnum** deben ser definidas de manera compatible a los identificadores, los números enteros y números de punto flotante en Java respectivamente.

El intérprete debe leer de la línea de comandos (consola) el nombre del archivo que contiene el programa que se va a interpretar. El reconocedor debe costruir el árbol sintáctico reducido del programa que se le pase como entrada, haciendo la revisión de tipos correspondiente. Si no hay errores de sintaxis o de tipos, el reconocedor debe pasar el control al intérprete del árbol sintáctico. Si hay algún error de sintaxis, un error con alguna variable no declarada o un error en los tipos (recuerden que se trata de un sistema de tipos fuerte) el reconocedor debe enviar el mensaje adecuado de error y terminar su ejecución. Este lenguaje tiene llamadas a función, las cuales deben implementarse tal y como discutimos en clase.

Como recordatorio: el paso de parámetros es por valor y las llamadas a función siempre devuelven un valor numérico. Los argumentos deben evaluarse antes de llevar a cabo la llamada a la función, esto es, no se usará evaluación perezosa. Al llamar a una función es necesario crear el ambiente para su ejecución, que puede hacerse copiando de alguna manera la tabla de símbolos.

$$
\begin{array}{lcl}
prog & \rightarrow & \textbf{program id } opt\_decls\ opt\_fun\_decls \textbf{ begin } opt\_stmts \textbf{ end} \\
opt\_decls & \rightarrow & decl\_lst \mid \varepsilon \\
decl\_lst & \rightarrow & decl \textbf{ ; } decl\_lst \mid decl \\
decl & \rightarrow & \textbf{let } id\_lst\textbf{: } tipo \\
id\_lst & \rightarrow & id\textbf{, } id\_lst \mid id \\
tipo & \rightarrow & \textbf{integer} \mid \textbf{real} \\
opt\_fun\_decls & \rightarrow & fun\_decls \mid \varepsilon \\
fun\_decls & \rightarrow & fun\_decl\textbf{, } fun\_decls \mid fun\_decl \\
fun\_decl & \rightarrow & \textbf{fun id (}opt\_params\textbf{) : } tipo\ opt\_decls \textbf{ begin } opt\_stmts \textbf{ end} \\
opt\_params & \rightarrow & param\_lst \mid \varepsilon \\
param\_lst & \rightarrow & param\textbf{, } param\_lst \mid param \\
param & \rightarrow & \textbf{id : } tipo \\
stmt & \rightarrow & \textbf{id} := expr \\
 & \mid & \textbf{if } expresion \textbf{ then } stmt \\
 & \mid & \textbf{if } expresion \textbf{ then } stmt \textbf{ else } stmt \\
 & \mid & \textbf{while } expresion \textbf{ do } stmt \\
 & \mid & \textbf{read } id \\
 & \mid & \textbf{print } expr \\
 & \mid & \textbf{return } expr \\
 & \mid & \textbf{begin } opt\_stmts \textbf{ end} \\
opt\_stmts & \rightarrow & stmt\_lst \mid \varepsilon \\
stmt\_lst & \rightarrow & stmt \textbf{ ; } stmt\_lst \mid stmt
\end{array}
$$

$$
\begin{array}{lcl}
expr & \rightarrow & expr + term \\
 & | & expr \texttt{ - } term \\
 & | & term \\
 & & \\
term & \rightarrow & term \texttt{ * } factor \\
 & | & term \texttt{ / } factor \\
 & | & factor \\
 & & \\
factor & \rightarrow & \texttt{(} expr \texttt{)} \\
 & | & \textbf{id} \\
 & | & \textbf{intnum} \\
 & | & \textbf{realnum} \\
 & | & \textbf{id (}opt\_args\textbf{)} \\
 & & \\
opt\_args & \rightarrow & arg\_lst \mid \varepsilon \\
 & & \\
arg\_lst & \rightarrow & expr\textbf{,}\ arg\_lst \mid expr \\
 & & \\
expresion & \rightarrow & expr < expr \\
 & | & expr > expr \\
 & | & expr = expr \\
 & | & expr <= expr \\
 & | & expr >= expr
\end{array}
$$