

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 construir 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.

<i>prog</i>	→	program <i>id</i> <i>opt_decls</i> <i>opt_fun_decls</i> begin <i>opt_stmts</i> end
<i>opt_decls</i>	→	<i>decl_lst</i> ε
<i>decl_lst</i>	→	<i>decl</i> ; <i>decl_lst</i> <i>decl</i>
<i>decl</i>	→	let <i>id_lst</i> : <i>tipo</i>
<i>id_lst</i>	→	<i>id</i> , <i>id_lst</i> <i>id</i>
<i>tipo</i>	→	integer real
<i>opt_fun_decls</i>	→	<i>fun_decls</i> ε
<i>fun_decls</i>	→	<i>fun_decl</i> , <i>fun_decls</i> <i>fun_decl</i>
<i>fun_decl</i>	→	fun <i>id</i> (<i>opt_params</i>) : <i>tipo</i> <i>opt_decls</i> begin <i>opt_stmts</i> end
<i>opt_params</i>	→	<i>param_lst</i> ε
<i>param_lst</i>	→	<i>param</i> , <i>param_lst</i> <i>param</i>
<i>param</i>	→	id : <i>tipo</i>
<i>stmt</i>	→	id := <i>expr</i> if <i>expresion</i> then <i>stmt</i> if <i>expresion</i> then <i>stmt</i> else <i>stmt</i> while <i>expresion</i> do <i>stmt</i> read <i>id</i> print <i>expr</i> begin <i>opt_stmts</i> end
<i>opt_stmts</i>	→	<i>stmt_lst</i> ε
<i>stmt_lst</i>	→	<i>stmt</i> ; <i>stmt_lst</i> <i>stmt</i>

$expr$	\rightarrow	$expr + term$ $expr - term$ $term$
$term$	\rightarrow	$term * factor$ $term / factor$ $factor$
$factor$	\rightarrow	$(expr)$ id intnum realnum id (opt_args)
opt_args	\rightarrow	$arg_lst \mid \varepsilon$
arg_lst	\rightarrow	$expr, arg_lst \mid expr$
$expresion$	\rightarrow	$expr < expr$ $expr > expr$ $expr = expr$ $expr \leq expr$ $expr \geq expr$