Tarea 1 CC41A Lenguajes de Programación
5 de abril 2004
Plazo: 2 semanas (inmodificable) - Individual
No se aceptan tareas fuera de plazo

1. Objetivos

Esta tarea persigue construir un analizador sintáctico basado en Lex/Yacc, que construya el árbol de la expresión leida en la entrada estándar. El programa debe escribir en su salida el árbol generado, en una sintáxis tipo Scheme.

El lenguaje de entrada es una calculadora de expresiones con asignación a variables, grupos de instrucciones encerrados entre llaves y la ejecución condicional `if`.

Por ejemplo:

```
a = 1 + 2 * 3;
-> (set! a (+ 1 (* 2 3)))
if(a > 1) a = 1; else b = 1; endif;
-> (if (> a 1) (set! a 1) (set! b 1))
```

Esta salida, al entregársela de entrada a un intérprete de scheme, debiera dar el resultado adecuado. Sin embargo, algunas libertades como asignar resultados de asignaciones y no definir variables antes son aceptadas.

2. Expresiones

Las expresiones aceptadas son valores enteros, variables y operadores:

```
+ - * / = > >= < <= ! == != && ||
```

Los espacios en blanco son ignorados y los nombres de variable comienzan por una letra minúscula y luego siguen con letras minúsculas o dígitos. Nada más es válido. Una expresión termina con punto y coma.

Se aceptan comentarios de dos tipos: un doble slash hasta el fin de línea y un comentario encerrado entre `[ ]` que puede ser multilínea.

La gramática de un programa es:
$L \rightarrow IL|\epsilon$
$I \rightarrow ;|E;|\{L\}$

La gramática de las expresiones es:
$E \rightarrow E+E|E-E|E*E|E/E|id=E|id|constante|(E)$
$E \rightarrow E\&\&E|E||E|E<E|E>E|E<=E|E>=E$
$E \rightarrow E==E|E!=E|!E$
$E \rightarrow if(E)Iendif|if(E)IelseIendif$

Las prioridades de los operadores son (de mayor a menor):

```
!
* /
```

```
- +
=
< <= > >= == !=
&& ||
```

3. Implementación

Ustedes deben hacer un archivo para flex, para que entregue los tokens léxicos y un archivo para bison para implementar la gramática. El objetivo es generar el árbol de la expresión e imprimirlo como expresión scheme (es equivalente a un recorrido en prefijo). Por ejemplo la expresión:

```
a = (2+3) - 2 * 5;
```

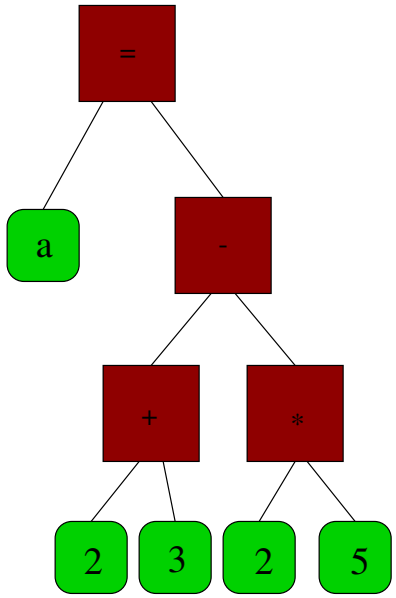


Figura 1: árbol de expresión

Debe generar el árbol de la figura 3 y luego el código Scheme:

```
(setq! a (- (+ 2 3) (* 2 5)))
```

Las condiciones se consideran expresiones normales:

```
if(a+1<2 && b>3)
    a=a+1;
else {
    b=b+1; a=a-1;
```

```
}
endif;
```

genera:

```
(if (and (< (+ a 1) 2) (> b 3)) (set! a (+ a 1)) (begin (set! b (+ b 1))
(set! a (- a 1))))
```

Después de imprimir el árbol en la salida estándar, hagan `fflush(stdout)` o tendrán que ingresar varias expresiones antes de ver cualquier resultado.

4. Datos

Se les dejó en la página web del curso un archivo lex y una yacc más un Makefile para que los usen de ejemplo y sobre ellos construyan lo necesario. Ese ejemplo construye una calculadora de expresiones con sólo constantes.

También hay un ejecutable de demo de la tarea.

5. Entrega

No se aceptan tareas fuera de plazo. Una tarea que tiene el análisis léxico correcto solamente tiene un 2.0, una tarea que acepta expresiones básicas sin condiciones (if) tiene un 4.0. Una tarea que no compila o no ejecuta tiene un 1.0.