**Laboratorio**

**Analizador Sintactico 1 con JAVACC**

Vamos a realizar un analizador sintáctico para un lenguaje que permite declarar variables booleanas y enteres y realizar operaciones matemáticas entre ellas.

La gramática para el lenguaje es:

<Programa> ::= <Sentencias> <EOF>

<Sentencias> ::= <SimpleStm>";"[<Sentencias>]

<SimpleStm> ::= <expr> | <declaracion>

<declaracion> ::= int <id> | bool <id>

<expr> ::= <Term> (<AddOp> <Term>)\*

<Term> ::= <Factor> (<MulOp> <Factor>)\*

<Factor> ::= "("<expr>")" | <number> | <identifier>

<MulOp> ::= "\*"|"/"

<AddOp> ::= "+"|"-"

Un ejemplo de código con el lenguaje descrito es:

int a;

int b;

int c;

a\*b+3;

b+(4\*9/8+(4-c));

Una vez que tenemos claro el lenguaje y sobre todo la gramática, podemos desarrollar el analizador semántico en JAVACC muy fácilmente para eso siga los siguientes pasos:

Usando JAVACC en eclipse.

1. Cree un proyecto en JAVA (Java Project)
2. Una vez que el proyecto esta creado, seleccionar la carpeta src y dar clic derecho sobre ella.
3. Seleccione New->Other.
4. En la ventana que sale escoja JavaCC->JavaCC Template File y de click en Next.
5. Escriba el nombre de la clase por ejemplo comp
6. Por defecto el crea un código fuente de prueba. Borrenlo todo y escriban en siguiente código

/\*\*

\* JavaCC template file created by SF JavaCC plugin 1.5.28+ wizard for JavaCC 1.5.0+

\*/

**options**

**{**

**static** = **true**;

**}**

**PARSER\_BEGIN(comp)**

**public** **class** comp

{

**public** **static** **void** main(String args []) **throws** ParseException

{

**try**

{

System.out.println("Ingrese el código a analizar:");

comp analizador = **new** comp(System.in);

analizador.Programa();

System.out.println("Se ha compilado con éxito");

}

**catch**(ParseException e)

{

System.err.println("Se han encontrado errores");

System.err.println(e.getMessage());

}

}

}

**PARSER\_END(comp)**

**SKIP**: /\* Whitespace \*/

**{**

"\t"

**|** "\n"

**|** "\r"

**|** " "

**}**

**TOKEN**:

**{**

**<**LPAREN: "("**>**

**|** **<**RPAREN: ")"**>**

**|** **<**ADD\_OP: "+" **|** "-"**>**

**|** **<**MULT\_OP: "\*" **|** "/"**>**

**|** **<**INT: "int"**>**

**|** **<**BOOL: "bool"**>**

**|** **<**NUMBER: **([**"0"**-**"9"**])+>**

**|** **<**ID: **([**"a"**-**"z","A"**-**"Z"**])+>**

**|** **<**SEMIC: ";"**>**

**|** **<**FINARCH: "?"**>**

**}**

**void** Programa():

**{}**

**{**

Sentencias() <FINARCH>

**}**

**void** Sentencias(): **{}**

**{**

**(**SimpleStm() <SEMIC> **[**Sentencias()**]** **)**

**}**

**void** SimpleStm(): **{}**

**{**

expresion() **|** declaracion()

**}**

**void** declaracion(): **{}**

**{**

<INT> <ID> **|** <BOOL> <ID>

**}**

**void** expresion(): **{}**

**{**

term() **(**<ADD\_OP> term()**)\***

**}**

**void** term() : **{}**

**{**

factor()**(**<MULT\_OP> factor()**)\***

**}**

**void** factor() : **{}**

**{**

<LPAREN> expresion() <RPAREN>

**|** <NUMBER>

**|** <ID>

**}**

1. Para compilar, clic derecho sobre el archivo .jj y darle compilar con JAVACC. Esto generará varias clases .java.
2. Para probarlo vayan a la clase comp.java y dar clic derecho Run as -> Java Aplication

Código para probar el programa

**Código sin errores:**

int a;

int b;

int c;

a\*b+3;

b+(4\*9/8+(4-c));?

Código con errores:

int 3a;

int b;

int c;

a\*b+3;

a%b;

b+(4\*9/8+(4-c));?

1. Adicione que permita en vez de expresiones asignaciones, es decir en el código no puede aparecer la línea b+4; como una sentencia, en lugar de ello deberá permitirse la asignación ejemplo a=b+4;
2. Modifique la gramática para que al declarar una variable, permita declarar varias al tiempo.

Como resultado de este laboratorio debe enviar un archivo que tenga la gramática BNF modificada con los cambios solicitados y el archivo .jj con los cambios solicitados.