UACH

**INFORME FINAL - TRABAJO SEMESTRAL**

*Traductor dirigido por sintaxis usando JFLEX-CUP*

Asignatura: INFO165 - Compiladores

Profesor: María Eliana de la Maza

Integrantes:   -Daniel Berlien

            -Ignacio Huichal

            -Patricio Suazo

Fecha: 1 de diciembre, 2016

Introducción

En el presente informe daremos a conocer el proceso que llevamos a cabo para la implementación de un traductor dirigido por sintaxis. donde su funcionamiento consiste en aceptar una secuencia de instrucciones del lenguaje AZ2016 y efectuar las acciones a medida que se realiza el análisis sintáctico.

Para ello las herramientas ocupadas son:

-Lenguaje utilizado es Java, junto las librerías:

-JFLEX (Analizador léxico)

-CUP (Analizador sintáctico)

Dentro de los objetivos que se desean abordar y cumplir, se encuentra dar una solución al problema planteado, entender el funcionamiento de un traductor dirigido por sintaxis, entender el concepto de reglas semánticas y comprender la especificación de una gramática utilizando los complementos de java JFLEX Y CUP.

Desarrollo

Primera gramática entregada:

Inst -> inicio inst fin | inst inst | leer(id) | escribir(mostrar) | inst\_asig | inst\_if

inst\_if -> si cond entonces inst

inst\_asig -> id = exp

cond -> exp oprel exp

exp -> exp opar exp | id

mostrar -> id

opar -> + | - | \* | /

id -> 0|1|…|8|9

variable -> A|B|…|Y|Z

oprel-> == | != | > | <

Gramática Final:

Go -> inicio enter instruc fin

instruc -> inst instruc | inst

inst -> inst\_si | inst\_asig | inst\_leer | inst\_escribir

inst\_escribir ->exp

inst\_leer ->digito

inst\_asig -> exp = exp

inst\_si ->si cond entonces inst

cond -> exp menor exp | exp mayor exp | exp igual exp

exp -> exp mas exp | exp menos exp | exp por exp | exp div exp | digito | variable

fin -> fin

Se implementó la variable ‘instruc’ para eliminar recursividad que generaba la variable ‘inst’.

Para la gramática final se consideró especificar directamente las operaciones relacionales en el caso de las condiciones.

También se modificó la producción ‘mostrar’ por ‘inst\_escribir’.

De la misma forma se modificó ‘id’ por ‘digito’.

Así, la gramática final implementada con la herramienta JFLEX de Java, queda de la siguiente manera:

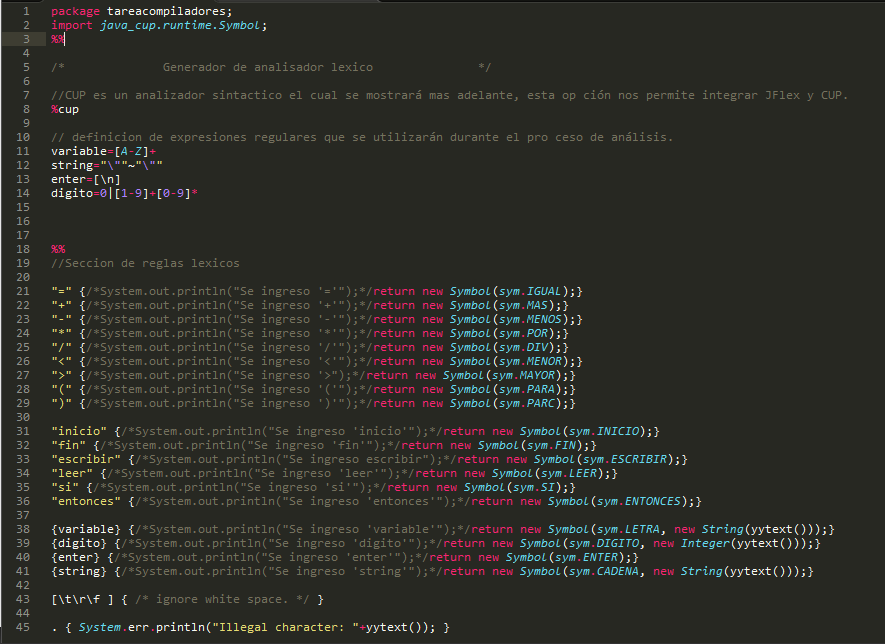


Figura 1: Código alexico.flex

Con “inicio” obtenemos la expresión regular para iniciar la ejecución.

Con “fin” obtenemos la expresión regular para finalizar la ejecución.

Con “escribir” obtenemos la expresión regular para poder imprimir texto en pantalla.

Con “leer” obtenemos la expresión regular que nos ayudara almacenar una variable para utilizarla en ejecución.

Con “si” y “entonces” obtendremos las expresiones regulares que nos ayudaran establecer sentencias de control.

Al haber manejado la herramienta JFLEX e implementar la gramática, se inicia con el análisis sintáctico, con la utilización de la herramienta CUP de Java. Para ello se implementa y definen el action code, parser code, terminales, no terminales y las producciones de la gramática de la siguiente forma:

**Action code:**

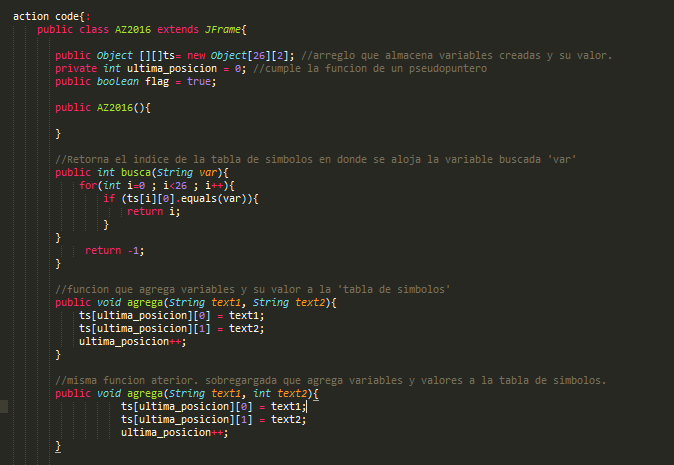
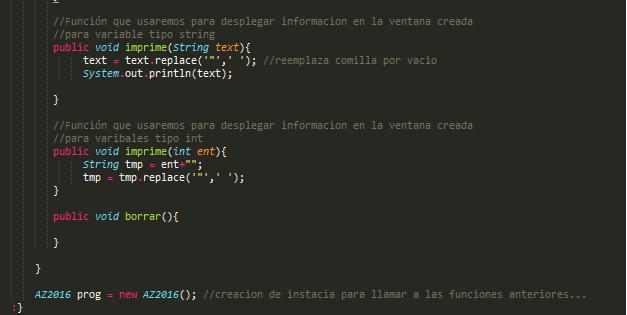
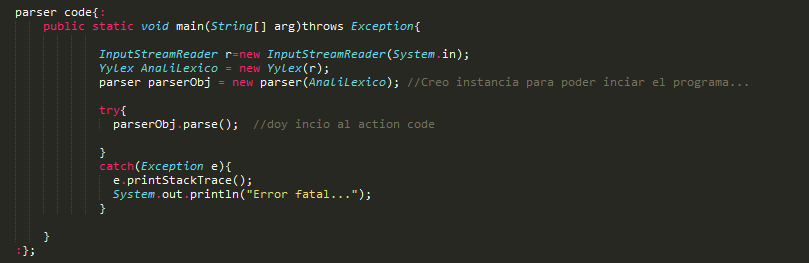


Figura 2: sección de código correspondiente a action code, esto ubicado en el archivo asintactico.cup

**Perser code:**

Figura 3: sección de código correspondiente a perser code, esto ubicado en el archivo asintactico.cup



**Terminales y no terminales:**

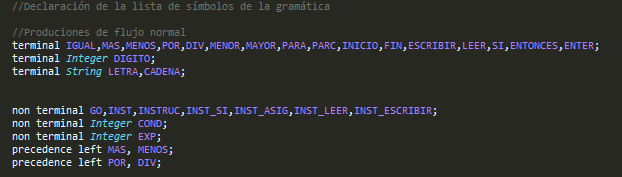
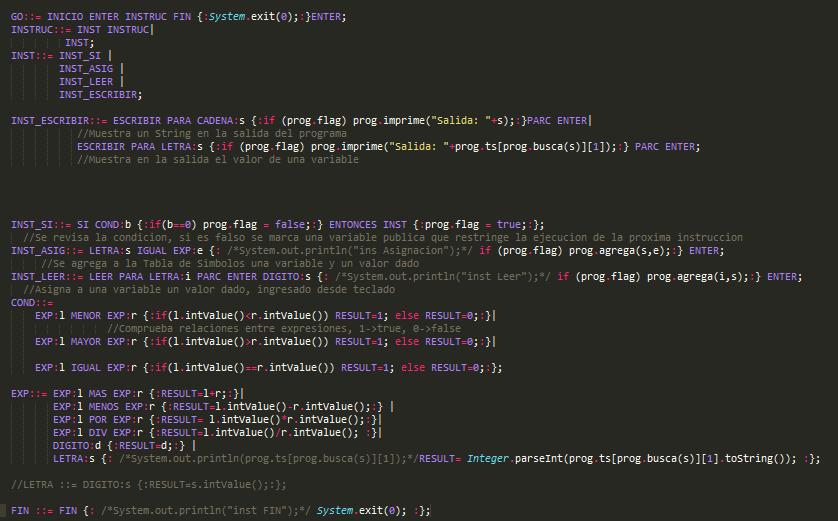


Figura 4: sección de código correspondiente a terminales y no terminales, esto ubicado en el archivo asintactico.cup

**Gramática**

Figura 5: sección de código correspondiente a la gramática utilizada, esto ubicado en el archivo asintactico.cup



Las clases correspondiente a alexico.flex y asintactico.cup son las encargadas de generar las clases necesarias para que el compilador funcione correctamente, de ellas se obtiene la clase parser.java, Yylex y sym.java. Las que nos ayudaran a ejecutar nuestro compilador.

Especificación del programa

Lenguaje de implementación: Java, con herramientas JFlex y Cup.

Requerimientos básicos de hardware: Windows 7 o superior, Procesador de 32 o 64 bits,10 Mb de disponibilidad en disco duro.

Requerimientos básicos de software: Java Runtime Environment (JRE) 1.2 o superior.

**Cómo compilar:**

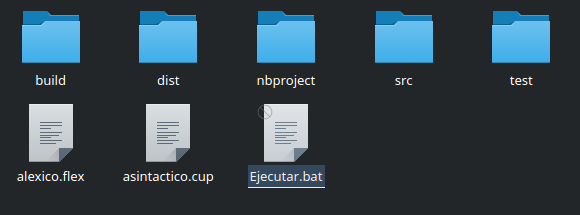
El programa viene ya compilado, pero si fuera necesario se hace de la siguiente manera: Se debe ejecutar el archivo “Generador.java” que está dentro del directorio “src/tareacompiladores” ignorando mensajes de advertencia. Nosotros ejecutamos el archivo “Generador.java” usando NetBeans (la carpeta principal “TareaCompiladores” es un proyecto de netbeans). Después de ejecutar el archivo se crean tres archivos en la misma carpeta: “parser.java”, “sym.java” y “Ylex.java”. Si existen esos tres archivos se puede ejecutar el programa.

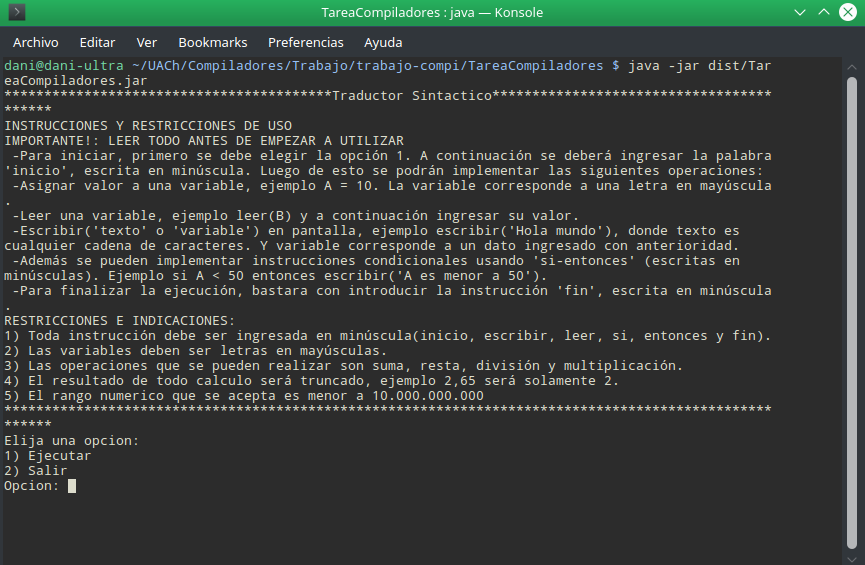
**Cómo ejecutar:**

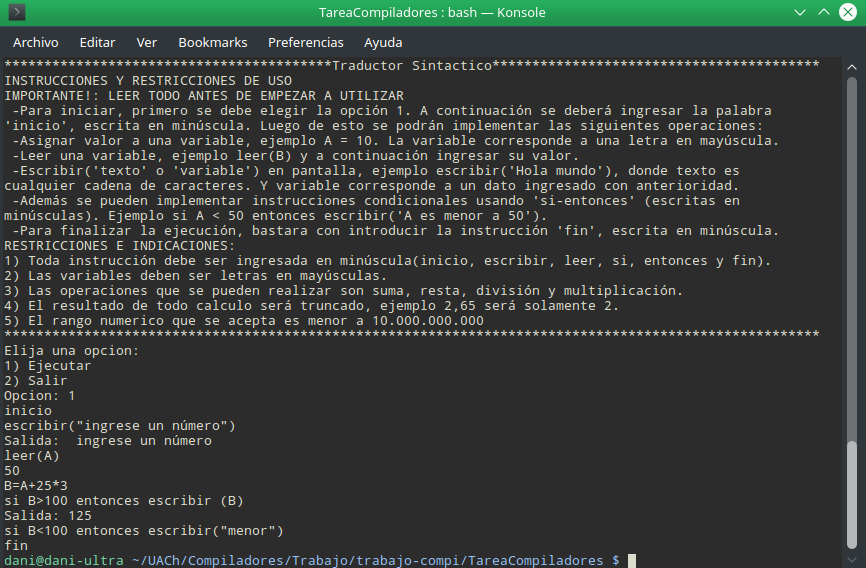
En la carpeta principal del programa se encuentra un archivo de extensión .bat llamado “Ejecutar.bat”. Al ejecutar este archivo se comienza la ejecución del programa. Es necesario tener instalado java en el sistema.

En el caso de que ese archivo no funcione bastara con ir a la carpeta ‘dist’ dentro del proyecto y en el encontrar en archivo ‘Ejecutar.bat’ para hacerlo funcionar.

**Ejemplo de funcionamiento:**

En la carpeta principal se visualizan los siguientes archivos, abrir “Ejecutar.bat”:

Abriendo el archivo se mostrará el programa en la consola:

Elegimos la opción 1 y se comienza la ejecución del compilador

En la imagen se usaron las instrucciones de ejemplo que se nos dieron con la tarea.

Se ingresaron las siguientes instrucciones:

-inicio

-escribir (“ingrese un número”)

-leer(A)

\*después de leer(A) escribimos el valor que queremos dar a A, en este caso 50

-B=A+25\*3

-si B>100 entonces escribir(B)

-si B<100 entonces escribir(“menor”)

-fin

Al ingresar la instrucción fin se acaba el proceso del programa, y es necesario abrir de nuevo el archivo Ejecutar.bat para volver a usarlo.

**Conclusión**

La construcción del traductor dirigido por sintaxis fue una oportunidad para poder conocer mejor y de manera más clara las etapas y procesos que ocurren en un computador para interpretar un lenguaje de programación.

Una de las dificultades que tuvimos fue la instrucción de asignación cuando se asignaba a una variable una expresión con otras variables, ya que al principio nos hacía conflicto ya que las variables las tomaba como string, y había incongruencia de tipos, lo pudimos solucionar extrayendo en valor real del símbolo en la tabla y usando algunos métodos para transformar los tipos.

Otra dificultad que se nos presento fue el de la recursividad, la cual finalmente pudo ser solucionada agregando otra producción.

Pero luego de buscar información sobre el tema para guiarse, se logró corregir cada error que se nos presentó y finalmente lograr una solución al problema sujeta a restricciones e indicaciones que son mostradas a la hora de efectuar su ejecución.

**Bibliografía**

**Rafael Vega Castro**

**Bibliografía:** Anon, (2016). [online] Available at: http://www.rafaelvega.info/wp-content/uploads/Articulo.pdf [Accessed 1 Dec. 2016].

**Build software better, together**

**Bibliografía:** GitHub. (2016). *Build software better, together*. [online] Available at: https://github.com/ [Accessed 1 Dec. 2016].

**Creación de un analizador sintáctico con JFlex y CUP en Java**

**Bibliografía:** YouTube. (2016). *Creación de un analizador sintáctico con JFlex y CUP en Java*. [online] Available at: https://www.youtube.com/watch?v=XQHivIfKvMk [Accessed 1 Dec. 2016].

**ditopo**

**Compilador Usando Jflex y Cup**

**Bibliografía:** ditopo, (2016). *Compilador Usando Jflex y Cup*. [online] Es.slideshare.net. Available at: http://es.slideshare.net/ditopo/compilador-jflexcup [Accessed 1 Dec. 2016].

**Marangon, J.**

**Utilizando o JFlex e Java CUP | Compiladores para Humanos**

**Bibliografía:** Marangon, J. (2016). *Utilizando o JFlex e Java CUP | Compiladores para Humanos*. [online] Johnidm.gitbooks.io. Available at: https://johnidm.gitbooks.io/compiladores-para-humanos/content/part2/using-jflex-java-cup.html [Accessed 1 Dec. 2016].