

**Clase:**

**Compiladores 1**

**Alumnos:**

Erick Carrasco - 11641166

Milton Martínez - 11511162

**Profesor:**

Ing. Carlos Vallejo

**Tarea:**

**Manual de Usuario de NxR**

**Fecha:**

28 de septiembre del 2020

**Tegucigalpa M.D.C, Honduras C.A.**

**Contenido**

[**Introducción** 3](#_Toc52295583)

[**Sobre NxR** 4](#_Toc52295584)

[**Instrucciones de uso** 5](#_Toc52295585)

[**Declaración de variables** 8](#_Toc52295586)

[**Operadores relacionales** 9](#_Toc52295587)

[**Operadores Lógicos** 10](#_Toc52295588)

[**Operadores aritméticos** 11](#_Toc52295589)

[**Bloques de decisión** 12](#_Toc52295590)

[**Iteración** 13](#_Toc52295591)

[**Switch** 14](#_Toc52295592)

[**Lectura y escritura** 15](#_Toc52295593)

[**Reglas de Comparación** 16](#_Toc52295594)

[**Funciones** 17](#_Toc52295595)

## **Introducción**

Este manual de usuario ha sido elaborado con la intención de ofrecer la información necesaria para el uso correcto del lenguaje NxR como parte del proyecto de la clase de **Compiladores 1,** en donde se muestra cómo realizar la declaración de variables, el uso de operadores relacionales, el uso de operadores lógicos, operadores aritméticos, bloques de código para decisiones y bloques de código para iteraciones.

## **Sobre NxR**

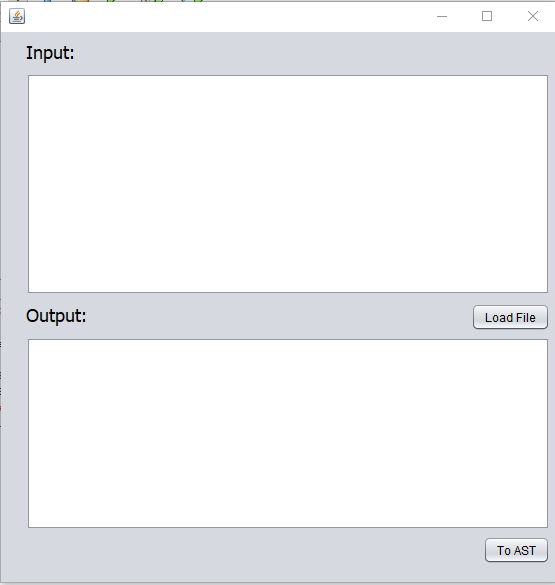
NxR es un lenguaje fuertemente influenciado por Ruby, con estructura de C, y también un poco de Java, en el lenguaje existen las variables como los **Int, Char y Faker,** pero también tenemos los arreglos o listas **vector**, que pueden almacenan cualquier tipo de dato, en el lenguaje se pueden crear funciones con el tipo de retorno, y también existe la recursividad.

## **Instrucciones de uso**

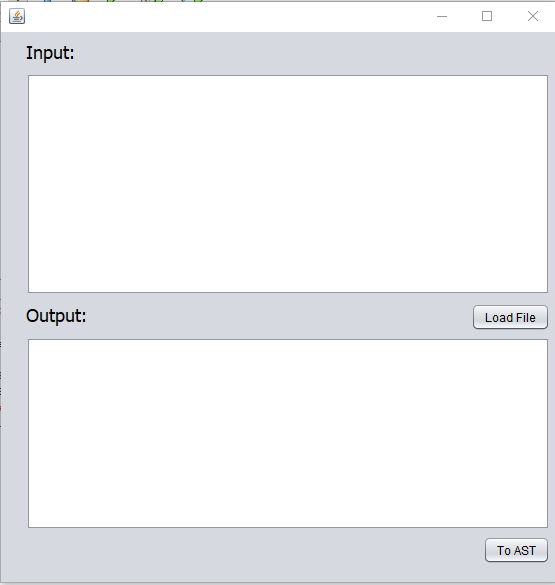
Para hacer uso del analizador léxico y sintáctico del lenguaje NxR, primero se debe de ejecutar el .java que se llama **Compilar\_Flex\_Cup.java** una vez creado los archivos:

1. **AnalizadorSintactico.java**
2. **Léxico.java**

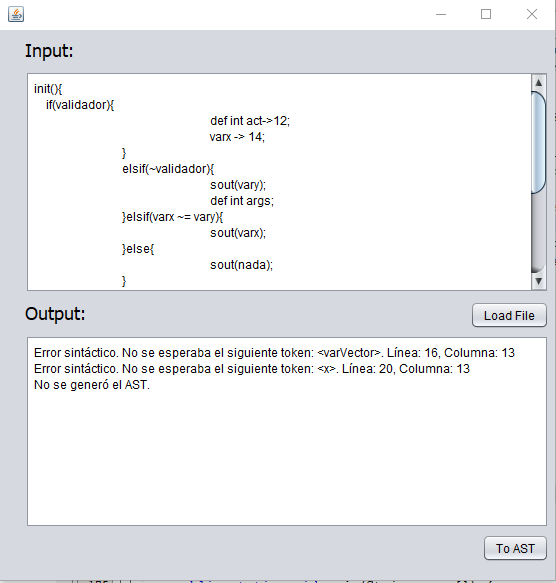
Se puede ejecutar el archivo .java que se llama GraphVIS.java, que es el programa que contiene la parte gráfica con la que se muestra el resultado de los analizadores. Se muestra la siguiente la pantalla:



Se puede cargar un archivo de texto, con un programa escrito en NxR, haciendo clic en el botón **“Load File”** y se muestra el código en el área de texto o se puede escribir directamente en el área de texto, y se autoguardara un archivo temporal del programa escrito.

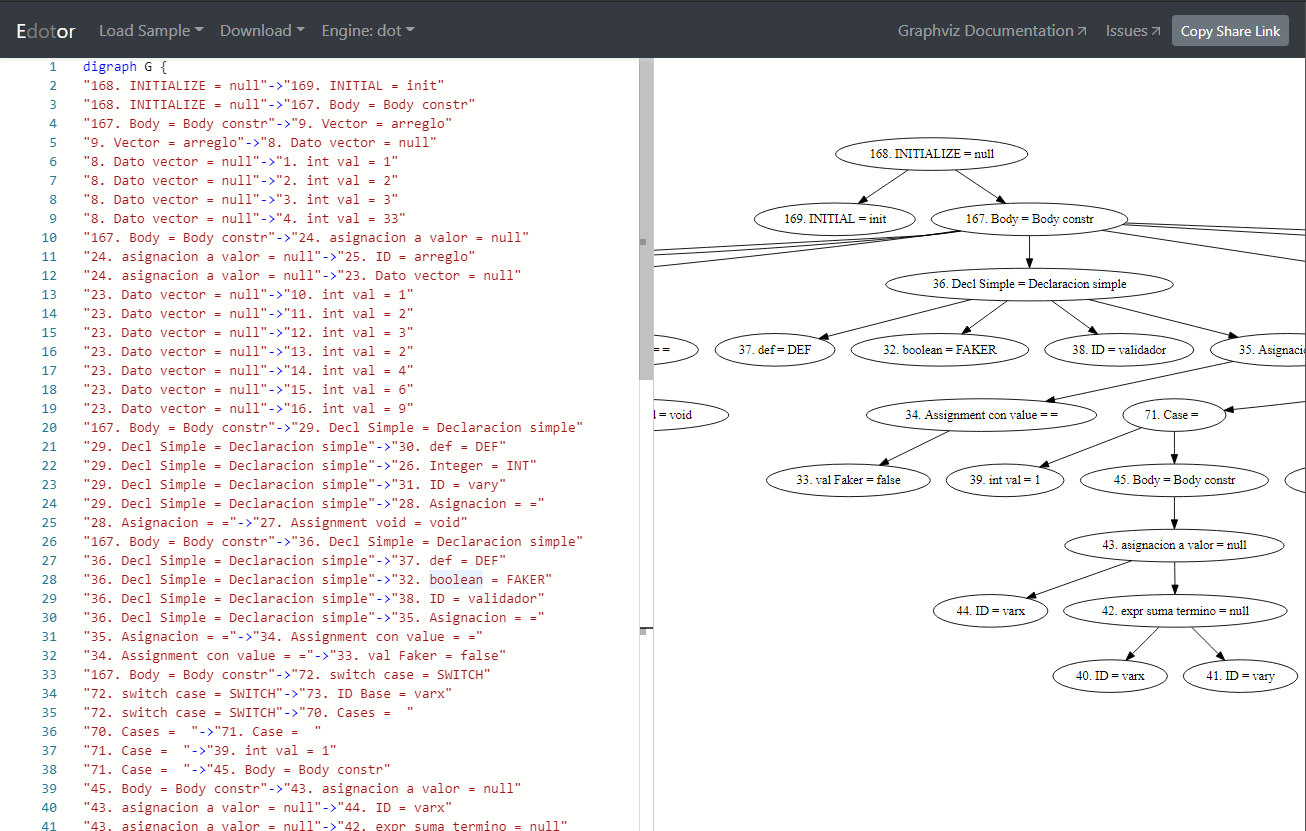


Luego, se puedo realizar un clic en el botón **“To AST”** y se mostraría el resultado del análisis sintáctico, en el área de texto, y además, en la carpeta del proyecto se crearía un archivo .**dot** que es el que contiene el Árbol Sintáctico Abstracto, pero solo en caso de que no exista un error se creara el árbol.



Para visualizar el árbol creado, se abre el archivo .**dot** llamado **“AbstractSyntaxTree.dot”** y se copia el contenido del documento, luego nos dirigimos al sitio web:  
<https://edotor.net/>

Y se pega el contenido e inmediatamente se mostraría el árbol generado.



Luego se puede descargar la imagen del árbol, haciendo clic en el menú de **“Download”.**

## **Declaración de variables**

def int ID -> Valor numérico;

def char ID -> Carácter;

def faker ID -> Estado Booleano;

def vector ID-> [] ;// Una dimensión.

def vector ID-> [[]] ;//Segunda dimensión.

## **Operadores relacionales**

~= // distinto de

=== // igual a

> // mayor que

< // menor que

>= //mayor o igual a

<=//menor o igual a

-> //asignación

## **Operadores Lógicos**

and //Operador lógico “Y”

or //Operador lógico “O”

~ //Operador lógico “negación”

## **Operadores aritméticos**

+ //suma

- //Resta

\* //Multiplicación

/ //división

% //Residuo de

## **Bloques de decisión**

Caso **Base**:

if (ExpresionCondicional) {

Expresion

}

Caso **Else if:**

if(ExpresionCondicional){

Expresion

}

elsif(ExpresionCondicional){

Expresion

}

Caso **Else**:

if(ExpresionCondicional){

expresion

}

...

Else{

expresion

}

## **Iteración**

loop{

Expresión

}each( cantRepeticiones )

While(ExpresiónCondicional){

Expresión

//Es necesario que el valor cambie durante la secuencia del while antes del “}”,

//en otro caso el while será infinito

}

## **Switch**

Switch(ValorCondicional){

//valor condicional puede ser int o char.

Case Contenido\_ValorCondicional:

Expresión

break;

//este es un caso por defecto

Case default:

Expresión

break;

}

## **Lectura y escritura**

Scan(tipoDeDato); //Lectura de variables

Sout(Variable); //escritura

Soutln(variable); //Escritura con salto de línea

## **Reglas de Comparación**

Comparación mayor o menor, mayor igual o menor igual, distinto e igual se puede usar para los **Int** sin problema.

Int < Int, int === int, int ~= int

Tipo **boolean** solo acepta comparación de igual o distinto

bool === bool, bool~= bool

Al igual que tipo chars

char === char, char ~= char

## **Funciones**

func int < nombre de función > ( parametro 1, parametro 2, ...parametro n){

Expresion

payback variable;

}

**Ejemplo**

INIT(){

def int x -> 0;

def char z -> 0;

def Vector j -> [1,2,3,4];

loop{

z -> scan(Char);

sout(“hola”);

soutln(“hola x2”);

}each(j);

def int y -> 1;

def int a-> 0;

if(a===y){

sout(“valido”);

}elsif (a>y){

sout(“Valor muy grande”);

}else{

sout(“Desconocido”);

}

}