



TRADUCTOR DE PROGRAMAS: PASCAL a C

**Introducción**

El objetivo de la práctica es transformar un lenguaje parecido a PASCAL a un lenguaje parecido a C usando un traductor. Para lograrlo se ha divido el trabajo en dos partes, un analizador léxico (jflex) y uno sintáctico (cup) unidos por un programa java. Para compilarlos usamos el cmd de Windows.

**Analizador Léxico**

La función del AL es, a partir de un conjunto de reglas léxicas, reconocer las cadenas que cumplan dichas reglas.

Primero se definen los macros que vamos a usar para reconocer las cadenas, estos son muy generales como *Blank* para reconocer espacio, tabuladores y saltos de líneas; o *Identifier* para reconocer palabras formadas por letras, números o signos.

A continuación, empezamos a poner los posibles símbolos terminales sencillo que se han de reconocer y las acciones que se van a tomar con ellos, que en este caso será retornar el symbol asociado a ellos.

Después reconocemos las expresiones regulares definidas en los macros. Por ejemplo, *Identifier* y *Numeric\_* se devuelven tal cual están en el lenguaje fuente. Si se pasa un **‘**, pasamos a un estado STRING donde se controla que haya otro **‘** para cerrar el anterior. Si se detectan dos **‘‘** seguidos se transforman en uno solo como indica el enunciado. Los espacios y comentarios se ignoran.

Finalmente, si no se reconoce ninguna de las anteriores, se manda un error donde se incluye el carácter no reconocido y su ubicación.

**Analizador Sintáctico**

La función del AS es introducir acciones semánticas al traductor. El nuestro se trata de una definición dirigida por la sintaxis porque que las acciones se asignan a consecuentes completos.

Primero definimos los terminales y no terminales, luego añadimos los precedentes. En un resumen rápido. hemos definido las acciones semánticas de manera que; empezando desde los no terminales más básicos, como **OPCOMP** o **OPARTIT** y vamos subiendo sus valores hacia arriba en el árbol, acumulando los valores hasta llegar a **PRG**.

Se han usado tres clases para guardar los valores; los más básicos son los Símbolos, estos se guardan en Sentencias, las cuales se guardan en dos listas en la clase Función, cada Sentencia en las listas representa una línea. Las dos listas se llaman Declaraciones y Bloque, pensado así para que todas las funciones y métodos se guarden en Declaraciones según va subiendo el traductor. Esto nos permite saber que parte del programa ira dentro del *main*, ya que ira dentro de la lista Bloque.

Una vez llegamos a **PRG** en el traductor, utilizamos el método reestructurar de la clase Función para imprimir las sentencias traducidas como deseamos. Este método recorre ambas listas, imprimiendo cada símbolo. Al principio de cada sentencia añade un numero de tabulaciones, el cual está determinado por si la sentencia actual es un cierre (baja el numero) o una apertura (lo sube).

Para saber cuando debemos transformas una asignación en un *return*, metemos en un *String* el nombre de la función actual y si alguna asignación dentro de él repite ese nombre lo cambiamos por *return* y ponemos un *boolean* a false para avisar al programa que durante lo que queda de sentencia no debe imprimir nada salvo lo que se está retornando.

Un problema encontrado es que el lenguaje no funciona como debería, al traducir no distingue entre **PROC\_CALL** y **FACTOR**, lo cual no seria un problema si no fuera porque cuando se llama a un método sin ningún valor no detecta el **SUBPARAMLIST** y no devuelve los paréntesis para indicar que una llamada.

Para solucionar esto se ha optado por meter una lista que se va llenando con los nombres de todos los métodos a los que no se les pasa ningún valor y si se vuelve a detectar esa ID se pone el “()”. Esto se hace si falla el *if* que comprueba si debe ponerse un *return* para así evitar confusiones.

**Ejemplos**