LISTAS

Proyecto 2021/2022
Teoría de Lenguajes

Jorge del Castillo Gómez y Raúl Hormigo Cerón

Índice

1. Introducción

Con el objetivo de llevar a cabo la evaluación de la asignatura, el profesorado ha decidido proponer la construcción de un traductor para el lenguaje llamado “LISTAS”. Este lenguaje permite trabajar con diferentes tipos de listas con mucha facilidad, dividiendo los programas en bloques donde se definen listas, variables e instrucciones que trabajen con todos ellos.

El presente documento busca realizar un repaso a todos los detalles de diseño del lenguaje generado por los alumnos, además de dar pautas y aclarar restricciones a la hora de utilizar dicho lenguaje.

2. Planificación del desarrollo

Siguiendo la recomendación del profesorado de la asignatura, el desarrollo del proyecto ha sido dividido en cuatro grandes fases:

a) Fase 1: generación de un lenguaje formal que permita el reconocimiento de un programa escrito en el lenguaje LISTAS. Tras finalizar esta fase, el programa será capaz de reconocer estos ficheros, pero no podrá realizar ninguna acción adicional con ellos.

b) Fase 2: creación de las Estructuras de Datos y control de la semántica del lenguaje. Así, el programa podrá almacenar y trabajar con listas y variables declaradas en los ficheros de entrada.

c) Fase 3: control de posibles errores y situaciones excepcionales. Al acabar, el programa será capaz de generar un fichero en lenguaje C++ como resultado de las operaciones indicadas en los ficheros de entrada.

d) Fase 4: redacción de la documentación y aclaración de las decisiones de diseño. Como resultado, se obtiene el presente documento que permite aclarar cualquier duda acerca del lenguaje LISTAS creado por el alumnado.

Bajo esta planificación, el desarrollo del proyecto puede dividirse en hitos fácilmente alcanzables y que permiten medir el nivel de progreso que se ha alcanzado durante la creación del lenguaje. Además, permite a los autores conocer aproximadamente el tiempo a invertir en cada fase.

2.1 Fase 1: generación de un lenguaje formal

Para comenzar a desarrollar un traductor que permita el reconocimiento de ficheros escritos en LISTAS, es necesario primero identificar las palabras reservadas del lenguaje. Utilizando un análisis léxico, algunas de estas palabras reservadas son “LISTAS”, “VARIABLES”, “INICIO”, “FIN”, “Si”, “Si_no”, “NuevaLinea”, etc.

La creación del lenguaje formal sigue un enfoque basado en bloques, donde los ficheros están divididos en una sección de listas, otra de variables y otra de código que debe ser ejecutado. Cada uno de estos bloques, a su vez, estará dividido en otras secciones, como son una o varias listas, una serie de declaraciones de variables, un conjunto de instrucciones, etc. Siguiendo este análisis, el lenguaje se irá desglosando poco a poco hasta llegar a los tokens obtenidos del estudio léxico anterior.

En este contexto, es importante conocer el concepto de “secuencia”. Una secuencia se refiere a un conjunto de conceptos normalmente ordenados que puede estar vacío, contener un solo concepto o una secuencia de estos. Algunos de los ejemplos ofrecidos por el profesorado muestra:

```
secuenciaCosas:
    | secuenciaCosas cosa
    ;

secuenciaCosas: cosa
    | secuenciaCosas cosa
    ;
```

Figura 1: Ejemplos de secuencias dados por el profesorado.

Con todos estos conceptos en mente, el lenguaje se ha ido desarrollando buscando siempre la mayor simpleza y eficiencia posibles. En este contexto, son necesarios aclarar una serie de restricciones impuestas por los autores de esta versión de LISTAS:

a) La sección de listas puede estar vacía, ya que es posible escribir programas en LISTAS sin contar con ninguna lista declarada.

b) Una lista no puede estar vacía, ya que no tiene sentido trabajar con una lista sin elementos.

c) La sección de variables puede estar vacía, ya que es posible escribir programas en LISTAS sin contar con ninguna variable declarada.

d) Toda variable declarada debe tener un tipo, y todo tipo declarado debe estar asociado a un identificador de variable. Esto es, las expresiones 'Entero;' o 'i' no tienen sentido y, por tanto, son consideradas errores sintácticos. Una versión correcta sería, por ejemplo, 'Entero i;'.

e) La sección de código debe contar con, al menos, una instrucción; ya que no tiene sentido crear un programa que no realice ninguna acción.

f) La instrucción 'Escribir ()' debe contener obligatoriamente algún parámetro (aunque sea una cadena vacía), ya que no tiene sentido no escribir nada por pantalla.

2.2 Fase 2: creación de las Estructuras de Datos

Para poder trabajar con este lenguaje, el traductor debe ser capaz de almacenar listas y variables, conocer los diferentes tipos de datos posibles y saber las relaciones que existen entre ellos. Con esto en mente, la autoría de esta versión del lenguaje LISTAS ha creado dos Estructuras de Datos bien diferenciadas: el Conjunto de Listas, que se encarga de crear, almacenar y gestionar todas las listas del programa; y la Tabla de Símbolos, que tiene la función de registrar y controlar las variables declaradas.

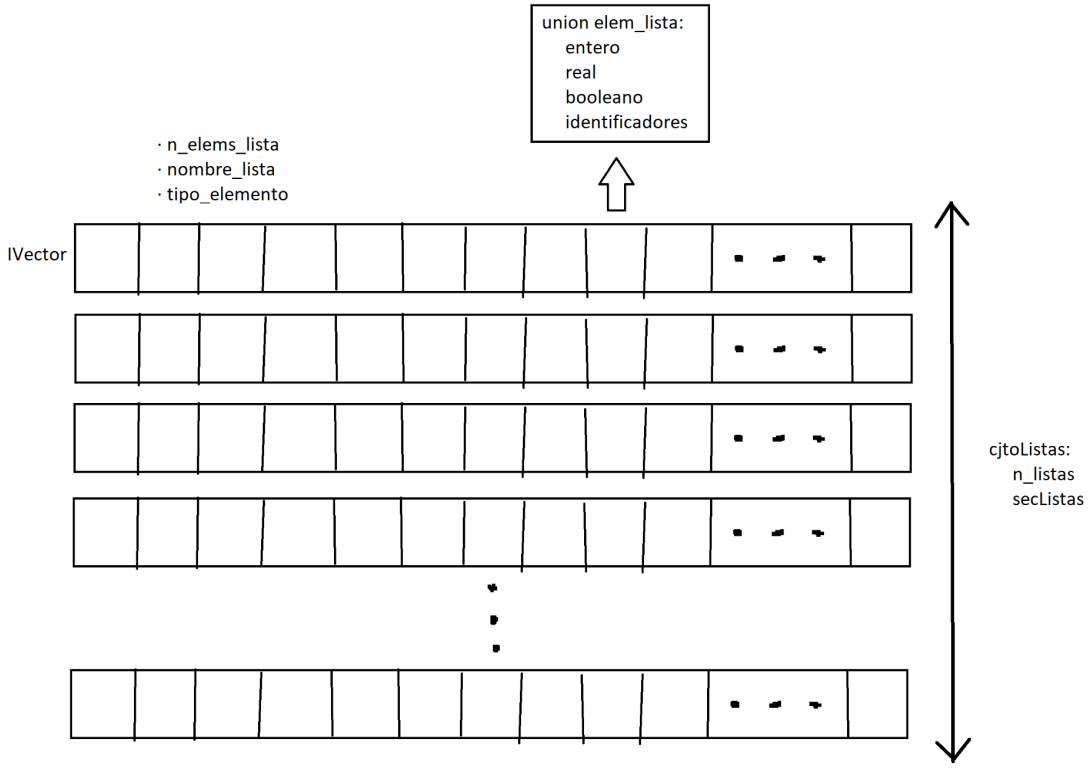

Figura 2: Esquema del Conjunto de Listas de esta versión del lenguaje LISTAS.

La construcción del Conjunto de Listas se basa en la combinación de los diferentes aspectos que lo forman. Estos aspectos son:

a) Los datos que las listas pueden contener son números enteros o reales, valores booleanos o identificadores. Así, las listas contienen uniones llamadas “elem_lista” con los tipos “int”, “float”, “bool” y “characters” (este último tipo es un vector de datos tipo “char” de 25 posiciones).

b) La estructura de una lista se forma como un Vector de Ocupación Variable (“IVector”) de tipo “elem_lista”, además de contener otros datos como su nombre (“nombre_lista”, de tipo characters), el número de elementos que contiene (“n_elems_lista”, de tipo int) y el tipo de los elementos que forman la lista (“tipo_elemento”). Todo ello constituye una única lista (“vectorLista”).

c) El tipo de los elementos de una lista (“tipo_elemento”) se concibe como un enumerado que asocia a cada tipo posible un número único. Este enumerado llamado “type_code” relaciona el

tipo entero (“e_integer”) con el número 0, el tipo real (“e_real”) con el número 1, el tipo booleano (“e_boolean”) con el número 2 y el tipo identificadores (“e_chars”) con el número 3. Este enumerado contempla también el tipo lista (“e_list”) asociado al número 4 que, aunque no es necesario para las listas del lenguaje, sí que resulta esencial para la declaración de variables y la Tabla de Símbolos.

d) Por último, el Conjunto de Listas se genera como un Vector de Ocupación Variable (“secListas”) de tipo “vectorLista”, junto al número de listas del programa (“n_listas”), de tipo int.

Asociadas al Conjunto de Listas existen una serie de operaciones necesarias para su gestión. Estas operaciones pueden clasificarse en dos grandes grupos:

a) Aquellos módulos destinados a la creación y gestión de una sola lista.

b) Los módulos creados para controlar el Conjunto de Listas en su totalidad.

Las operaciones relacionadas con las listas como elementos individuales son:

Nombre	Operación
iniciarVectorLista	Inicializa la lista sin ningún elemento.
insertarENLista	Inserta un nuevo elemento en la lista. En caso de que el tipo del nuevo elemento no coincida con el tipo de los elementos de la lista, indica un error semántico.
estaVacia	Indica si la lista dada por parámetro está vacía.
mostrarLista	Muestra por pantalla los elementos de la lista dada.
cambiarNombreLista	Actualiza el nombre de la lista con el nuevo nombre dado.
crearListaRangos	Crea una lista nueva dados los límites del rango.

Figura 3: Tabla que indica las operaciones relacionadas con las listas como elementos individuales.

Las operaciones relacionadas con el Conjunto de Listas en su totalidad son:

Nombre	Operación
iniciarCjtoListas	Inicializa el conjunto de listas a cero.
insertarLista	Inserta una nueva lista en el conjunto de listas en caso de que no exista. Si la lista ya existe, indicará un error semántico.
buscaLista	Devuelve por el parámetro de entrada-salida la lista identificada por el nombre, siempre que exista; y devuelve 'false'. En caso de que no exista, el módulo devuelve 'true'.
mostrarCjtoLista	Muestra por pantalla todas las listas almacenadas en el conjunto.

Figura 4: Tabla que indica las operaciones relacionadas con el Conjunto de Listas en su totalidad.

Pasando a la Tabla de Símbolos, se encarga de generar y gestionar todas las variables posibles en el lenguaje LISTAS.

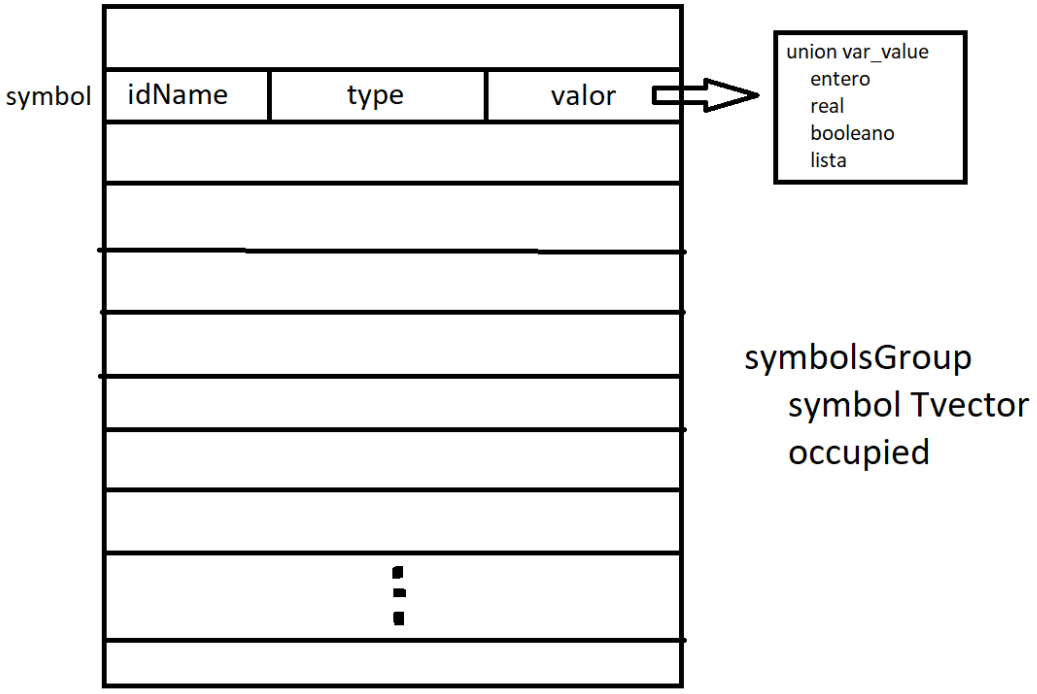

Figura 5: Esquema de la Tabla de Símbolos de esta versión del lenguaje LISTAS.

La construcción de la Tabla de Símbolos se basa en la combinación de los diferentes aspectos que la forman. Estos aspectos son:

a) Las variables pueden ser de tipo entero, real, booleano o de tipo lista. Así, las variables se definen como uniones llamadas "var_value" con los tipos "int", "float", "bool" y "vectorLista" (este último siendo una lista individual del Conjunto de Listas previamente definido).

b) Cada variable tiene asociada un tipo indicado en el campo "type" del tipo "type_code" previamente explicado (lo que justifica la presencia del campo "e_list"). Además, también tienen un nombre asociado ("idName" de tipo characters) y, todo ello, forma una variable (el tipo "symbol").

c) Por último, la Tabla de Símbolos se construye como un Vector de Ocupación Variable ("Tvector", de tipo symbol) y el número de posiciones de la tabla ocupadas ("occupied"), de tipo int. En su conjunto, constituye un "symbolsGroup".

Asociadas a la Tabla de Símbolos existen una serie de operaciones necesarias para su gestión. Estas operaciones son:

Nombre	Operación
iniciar	Inicializa la Tabla de Símbolos con cero variables.
insert	Inserta un nuevo símbolo en la Tabla de Símbolos. En caso de que ya exista, su valor se ve actualizado.
searchVariable	Busca en la Tabla de Símbolos la variable dado su nombre y lo devuelve por el parámetro de salida en caso de encontrarlo; devolviendo 'false'. Si no lo encuentra, devuelve 'true'.
selectEnum	Devuelve la interpretación en std::string del tipo indicado por parámetro.
showIDs	Muestra la Tabla de Símbolos por el flujo de salida indicado.
showID2s	Muestra la Tabla de Símbolos por pantalla.
eliminaSimbolo	Elimina la variable de la Tabla de Símbolos dado su nombre, en caso de que exista. Si la elimina, devolverá 'false'. Si no la encuentra, devolverá 'true'.

Figura 6: Tabla que indica las operaciones relacionadas con la Tabla de Símbolos.

2.3 Fase 3: control de posibles errores

Para que el traductor pueda trabajar correctamente, es necesario que las producciones tengan asociadas ciertas acciones que permitan el manejo de las Estructuras de Datos y de la correcta interpretación de los ficheros de entrada. Además, el resultado del

traductor es un fichero escrito en C++ que permite mostrar los resultados del programa redactado en LISTAS.

Con todo esto en mente, se han generado una serie de módulos que permiten ejecutar ciertas acciones relacionadas con la salida del traductor. Estos módulos son:

Nombre	Operación
removeChar	Elimina los caracteres " de la cadena dada por parámetro.
traducirBooleano	Devuelve una traducción del valor booleano del parámetro a std::string.
funcionesEspeciales	Ejecuta las funciones "primero (Lista)", "ultimo (Lista)" y "enesimo (Lista)" dependiendo del parámetro "posicion".
evaluarCondicional	Permite evaluar qué parte del condicional se ejecutará.
codigo_c_antes	Muestra el código en C++ que debe escribirse antes de ejecutar el módulo "parse ()".
codigo_c_despues	Muestra el código en C++ que debe escribirse después de ejecutar el módulo "parse ()".
funcionNuevaLinea	Permite ejecutar la función "NuevaLinea" en el fichero de salida.
funcionEscribir	Permite ejecutar la función "Escribir" en el fichero de salida.
copiarLista	Copia el contenido de la lista en la cadena.

Figura 7: Tabla que indica las operaciones relacionadas con la salida del traductor.

Para revisar las acciones asociadas a las producciones que definen esta versión del lenguaje LISTAS, es necesario revisar el código fuente del archivo “sintactico.y”.

2.4 Fase 4: redacción de la documentación

Para la redacción de la presente documentación, la autoría se ha servido de la documentación interna presente en los archivos relacionados con el proyecto, además de revisar las decisiones de diseño tomadas a lo largo del proceso de creación del lenguaje.

3. Ampliaciones

Al momento de la redacción del presente documento, la creación de esta versión del lenguaje LISTAS no cuenta con ninguna ampliación propuesta por el profesorado de la asignatura.

4. Conclusiones

Tanto la parte de contenidos teóricos como la de contenidos prácticos de la asignatura ha permitido al alumnado comprender cómo se pueden desarrollar lenguajes formales siguiendo técnicas matemáticas y, por ende, cómo crear un lenguaje de programación desde el principio. También ha permitido entender la complejidad del proyecto y las dificultades a las que se enfrentaron las primeras personas que desarrollaron lenguajes de programación durante el siglo pasado.

Por último, e investigando un poco más en el tema, se puede concebir la dificultad que resulta desarrollar este tipo de lenguajes; conociendo sobre todo que lo explicado a lo largo de la asignatura no abarca todo el proceso de creación de lenguajes de programación.