LENGUAJE ALGORÍTMICO EN ESPAÑOL: LES

LUIS ALEJANDRO BERNAL ROMERO *

RESUMEN

El lenguaje algorítmico en español LES es principalmente una herramienta didáctica para facilitar el aprendizaje de la algoritmia y fomentar las buenas prácticas de la programación. Es un lenguaje estructurado que permite la abstracción procedimental y de datos y forma parte de la Metodología para la solución de problemas algorítmicos que está siendo desarrollada en el programa de Ingeniería de Sistemas de la Escuela de Administración de Negocios (E.A.N.). Se describen en este artículo el por qué de su creación, las características, algunos ejemplos, la formalización del lenguaje y finalmente el estado actual.

Palabras clave: Lenguajes de programación, algoritmia, aprendizaje de la programación.

*Ingeniero de Sistemas, Universidad Central.
Magister (candidato) en Ingeniería de Sistemas y Computación, Universidad de Los Andes.
Docente Tiempo completo, Escuela de Administración de Negocios E.A.N.
E-mail: lbernal@uvirtual.ean.edu.co

INTRODUCCIÓN

El equipo de profesores del Departamento de Ciencias de la Computación de la Escuela de Administración de Negocios (EAN), está desarrollando una Metodología para la resolución de problemas algorítmicos (ver [Amador98]). Esta metodología se compone de dos aspectos: de una serie de pasos sucesivos que llevan a la solución de problemas mediante algoritmos, y una serie de acuerdos de cómo se deben presentar los resultados de cada paso.

En uno de los pasos de la metodología, El diseño del algoritmo, es necesario una representación de éste mediante algún formalismo[1] que permita definir de manera precisa cómo es el algoritmo, si realmente resuelve el problema para el que fue escrito, y lo más importante: que se pueda programar una máquina algorítmica[2] con este formalismo.

En la metodología se han definido dos formalismos: los diagramas de flujo (ver [Amador98]) y un lenguaje algorítmico en español; LES. Este artículo trata sobre dicho lenguaje y en particular lo concerniente a: qué problemas pretende solucionar, cuáles son sus características y su forma.

1. ¿Por qué un lenguaje algorítmico en Español?

Uno de los problemas más complejos al que se enfrenta el estudiante de computación es el de la programación y en particular la resolución de problemas mediante métodos algorítmicos. Es muy difícil introducir al estudiante en esta forma de pensar, primero por la dificultad inherente a la algoritmia y segundo porque los lenguajes de programación y los pseudoalgorítmicos están basados en el idioma inglés. El cambio a un lenguaje que no es el nativo dificulta al estudiante, en forma innecesaria, el aprendizaje de la programación. Por ello se ha desarrollado un lenguaje algorítmico en español, que facilite su aprendizaje.

Existe una clase particular de máquinas programables que se ubican dentro del paradigma definido por la Máquina de Turing. Este paradigma responde a una lógica que resuelve problemas paso por paso mediante operaciones mecánicas y empleando un número finito de pasos. El computador es una máquina de Turing, esto quiere decir, que todo lo que puede hacer una máquina de Turing lo puede hacer un computador y viceversa.

Esta forma de funcionar, esta lógica, es diferente a la que utiliza el cerebro en la vida cotidiana. En el caso particular del aprendizaje de los algoritmos, ésta es una lógica ajena a la forma de pensar de los estudiantes. Entonces, es necesario entrenar una parte del cerebro del estudiante para que "piense" utilizando dicha lógica, esto es, que forme un modelo mental de máquina algorítmica, punto clave en el aprendizaje de la algoritmia. Pero esto no es nada fácil, si además, tenemos en cuenta que para programar estas máquinas normalmente se usan lenguajes artificiales basados en el inglés, lengua en la cual no piensan los estudiantes.

Por lo anterior, LES es un lenguaje algorítmico en español diseñado para facilitar la formación de ese modelo mental de máquina algorítmica como una extensión o particularización realizada en el lenguaje materno.

Pero éste no es el único problema que pretende solucionar el lenguaje. Se busca también fomentar las buenas prácticas de programación ya que el lenguaje obliga a escribir algoritmos estructurados, bien documentados, limpios, fáciles de entender y por tanto de mantener.

Otro aspecto que resuelve el lenguaje es que proporciona mecanismos de abstracción, encapsulamiento y reutilización de código, tal y como exigen los grandes proyectos de Ingeniería de Software. Veamos estos conceptos en forma más detallada:

---

[1] Formalismo se refiere a representaciones lingüísticas (lenguajes artificiales) literales o gráficas. Estas representaciones tienen bases matemáticas. Más adelante se explica cuales deben ser los elementos para que un lenguaje artificial, como es el caso de LES, se considere formalmente especificado.

[2] Se considera máquina algorítmica a cualquier máquina que sea reducible a la máquina de Turing. En estas están incluidas las computadoras.

⇒ La abstracción es un mecanismo que permite agrupar todo lo relacionado con un concepto, en una unidad que puede ser usada como una unidad atómica en el lenguaje.

⇒ El encapsulamiento consiste en dividir la implementación de estos conceptos en dos partes: la interfaz y la implementación. La interfaz es la parte de comunicaciones de la unidad, define lo que hace, pero no cómo lo hace. La implementación, como su nombre lo indica, define el cómo lo hace.

⇒ La reutilización de código, como su nombre lo indica, es volver a usar el código ya escrito anteriormente, primero para no reinventar la rueda y segundo, porque reutilizar el código tiene el efecto de reducir el número de errores en él.

Todo lo anterior se logra mediante los mecanismos de Tipos Abstractos de Datos (TAD) y las Clases de Objetos, características básicas sobre las que está implementado el lenguaje LES. Todas estas características y otras se detallarán a continuación.

2. Características del lenguaje LES

Las características del lenguaje son: es una herramienta didáctica, es estructurado, orientado hacia la abstracción de datos y fomenta las buenas prácticas de programación. A continuación se especifican en detalle cada una de estas características.

2.1 LES es una herramienta didáctica

Su orientación didáctica es la principal característica del lenguaje, ya que nació de la necesidad de enseñar a programar. De ahí que se ha diseñado teniendo en cuenta algunas condiciones deseables en el proceso de enseñanza y aprendizaje, por ejemplo: Está basado en el español, pues es más fácil entender los algoritmos que están escritos en la lengua en que se piensa. Al estar basado en la lengua materna es mucho más fácil formar un modelo mental de como funciona la máquina algorítmica y de esa forma pasar a un lenguaje de expresión de algoritmos que tiene como base el inglés o cualquier otro lenguaje. Por otro lado, no utiliza abreviaturas ni atajos que oscurezcan el código; prefiere en todo momento la claridad a la eficiencia expresiva[3].

2.2 Es un lenguaje estructurado

Un lenguaje estructurado tiene que cumplir con unas características específicas. Estas se basan en lo que establecen [Dijkstra72] y [Hoare72] sobre la programación estructurada:

- **Estructuración en sentencias.**

♦ No uso de la sentencia «ir a» en inglés «go to». Dijkstra hace una excelente demostración sobre lo nocivo de su uso en [Dijkstra68].

♦ Las sentencias deben poder anidarse unas en otras, ésto es lo que se conoce como estructuración.

♦ Las sentencias básicas son:

Sentencias simples

⇒ Asignación.
⇒ Llamado a procedimientos.

Sentencias repetitivas (ciclos):

⇒ Condicional: Mientras que (while).
⇒ Repita .... hasta (repeat ... until).
⇒ Para. (for).

Sentencias condicionales:

⇒ Si .... entonces (if ... then).
⇒ En caso de (case of).

Constructores de sentencias

⇒ Declaración de funciones.
⇒ Declaración de procedimientos.

---

[3] Esto también se conoce en el medio como taquigrafías, esta forma de escribir hace sentencias muy cortas que son al mismo tiempo muy poderosas, pero con el problema que son difíciles de leer.

• **Estructuración en datos.**

♦ Las estructuras de datos deben poder anidarse para formar datos complejos.
♦ Los tipos de datos son:

Datos simples.

Constuctores de datos.

⇒ Arreglos (array).
⇒ Estructuras (struct).

En síntesis, el lenguaje LES es estructurado porque permite la creación de nuevos tipos de datos y de sentencias que se pueden usar de igual forma que las sentencias originales del lenguaje. El lenguaje LES fue diseñado teniendo en cuenta todas las anteriores características.

2.3 Orientado a la abstracción de datos

La abstracción de datos es un mecanismo mediante el cual se encapsulan las sentencias y los datos asociados a un concepto. Esto facilita pensar en forma unificada sobre un tipo de dato. A esto se le llama un TAD (o TDA) Tipo Abstracto de Dato. Las características y la formalización matemática de un TAD están en el artículo [Cardoso86] y en [Cardoso87]. Estas características se pueden sintetizar en los siguientes puntos:

• Es un conjunto de objetos: Todos los objetos que pertenecen al TAD deben cumplir con una condición, la cual es una afirmación que debe estar escrita en lógica formal. Tal afirmación se conoce como Invariante de TAD.

• Tiene un conjunto de operaciones: Se definen una serie de operaciones que deben cumplir con dos condiciones: primero, deben involucrar al menos un objeto del tipo; y segundo, no deben violar el invariante de TAD.

En resumen, "Un TAD es una estructura algebraica, o sea, un conjunto de objetos con ciertas operaciones definidas sobre ellos." [Villalobos90, pag57]. El lenguaje LES implementa este concepto.

Otras de las herramientas conceptuales para la abstracción de datos es el objeto. Este es una evolución del concepto de TAD. El objeto nació del mundo de la simulación en computador, pues el objeto se basa en modelar el mundo real en el dominio de la máquina. Los principios del paradigma de objetos se encuentran en [Booch94] y en [Rumbaugh91], se pueden resumir en los siguientes puntos:

• Objeto: la unidad básica de abstracción y representación del mundo.

• Atributos: las características del objeto.

• Mensajes: las operaciones que se pueden realizar sobre los objetos.

• Clase: la generalización de los objetos, la plantilla que dice como serán los objetos que pertenecen a esa clase.

• Herencia: la base de la reutilización de código. Una clase que hereda de otra toma todos sus atributos y mensajes como si fueran propios.

El lenguaje LES tiene las constructoras necesarias para trabajar con objetos y cumple con todo lo anterior.

2.4 Fomenta las buenas prácticas de programación

El lenguaje promueve las buenas prácticas de programación mediante las siguientes características:

• Documentación de cada segmento significativo del algoritmo, como las declaraciones de programa, funciones, procedimientos, TADs, clases, variables y objetos. En el caso de la documentación de programa, procedimiento, función, operación, mensaje, TDA y Clase el lenguaje exige adicionalmente: autor, fecha y especificación.

• Documentación en lenguaje natural y en una línea del propósito de: funciones, procedimientos, TADs y clases.

• Especificación formal mediante precondición y postcondición de cada módulo a saber: el pro-

grama como un todo, las partes del programa, las funciones, procedimientos, las operaciones de TADs y los mensajes de clase. Exige definir también los invariantes de TAD y de clase. Exige también el invariante para los ciclos, y permite definir aserciones en medio del código.

• No permite usar variables globales.

• Las instrucciones de control de flujo son estructuradas, esto es:

1. No permite el uso del "ir a"[4].

2. Tiene las constructoras básicas y sus derivadas que son: sentencias atómicas, sentencia de agrupación, sentencia "si", sentencia "mientras que"; funciones y procedimientos y sus derivadas que son: sentencia "haga....mientras_que", "para" y "en caso de".

• Las declaraciones de datos son también estructuradas. Se tienen los siguientes estructuradores: los tipos básicos, la estructura, la unión, el TAD y la clase.

• Permite la abstracción y encapsulamiento mediante el uso de TADs y clases.

2.4 Orientado a TADs y Clases

El lenguaje posee mecanismos, ya explicados anteriormente, de abstracción de datos que son los Tipos Abstractos de Datos y las clases. Se trabajan estos por razones didácticas, ya que es más fácil introducir en primera instancia los TADs y luego los objetos como un refinamiento de los TADs.

3. ¿Cómo es el lenguaje?

Se describirá el lenguaje partiendo de tres conceptos: los tipos de datos, las sentencias y los constructores del lenguaje, para luego presentar algunos ejemplos de su implementación.

3.1 Tipos de datos del lenguaje

El lenguaje tiene los tipos básicos que se encuentran en los lenguajes de programación. Para darle una estructura a éstos se ha organizado una jerarquía de clases.

3.2 Sentencias del lenguaje

El lenguaje incluye las instrucciones estructuradas que son comunes a todos los lenguajes. Se dividen en dos tipos básicos: las sentencias atómicas que son las básicas del lenguaje, y las de control de flujo que permiten variar la forma como transcurre el flujo de control. Las sentencias son las siguientes:

• Sentencias atómicas.

⇒ Asignación.
⇒ Leer datos de la entrada estándar.
⇒ Escribir datos a la salida estándar.

• Sentencias de control de flujo.

⇒ Agrupación de sentencias.
⇒ Sentencia "si".
⇒ Sentencia "mientras que".
⇒ Sentencia "repita ... hasta".
⇒ Sentencia "para".
⇒ Sentencia "en caso de".
⇒ Llamada a procedimiento.

3.3 Constructores del lenguaje

El lenguaje provee formas de construir nuevas funciones, procedimientos, tipos y clases. Para ello dispone de dos tipos de constructores:

• Constructores procedimentales:

⇒ Declaración de funciones.
⇒ Declaración de procedimientos.

---

[4] En la mayoría de lenguajes de programación, que están en inglés, la sentencia "ir a" se escribe "go to". El uso de la instrucción "go to" en programación es considerada mala práctica. Cuando se utilizan, los programas de computador toman un aspecto tan desordenado que se le ha llamado programación spaghetti. La sustentación de este problema se encuentra en el artículo clásico [Dijkstra68].

- Constructores de tipos:

  ⇒ Declaración de Tipos Abstractos de Datos.

  ⇒ Declaración de Clases de Objetos.

En cada una de estas declaraciones es obligatoria la documentación mediante la descripción de la construcción con una frase en lenguaje natural. Además, las funciones y procedimientos deben especificarse mediante precondición y postcondición. Por su parte los TADs y las clases deben tener su invariante, además de la documentación de sus operaciones.

3.4 Un ejemplo sencillo

La mejor forma de ver cómo es el lenguaje es mediante un ejemplo[5]. El enunciado del problema es el siguiente:

Se busca encontrar la suma de las áreas de un determinado número de cuartos.

El algoritmo en el lenguaje es el siguiente:

```
programa sumatoria
        autor «John Alexander Rojas Montero»
        código 79451716
        fecha 1998.02.22
        propósito «Halla la sumatoria de las áreas de los cuartos
        de una oficina»
        precondición pertenece(n,naturales) y n <> 0 y pertene
        ce (x,reales) y pertenece(y_,reales)
        postcondición pertenece(z,reales) y z =
        sumatoria(1,n,x*y_)
        nivel básico
inicio
        variables
                n: entero;  /* Cantidad de cuartos. */
                x: real;    /* Ancho de cada cuarto. */
                y_: real;   /* Largo de cada cuarto. */
                c: entero;  /* Contador. */
                z: área;    /* Suma de las areas de los cuar
                            tos. */
        algoritmo
                /* Paso 1: Leer cuantos cuartos tiene la ofici
                na
                ( verificado la precondicion sobre n) */
                repita
                        invariante verdadero
                        leer(N);
                hasta n > 0;

                /* Paso 2: leer las medidas de cada cuarto. */
                c:= 0;
                y_:= 0;
                z:= 0;
                mientras_que c < n
                        invariante z = sumatoria(0, c, c*y_ )
                haga
                        leer(x);
                        leer(y_);

                /* Paso 3: Sumar las areas de los cuartos. */
                        z:= z + x * y_;
                        c:= c + 1;
                fin_mientras_que;

                /* Paso 4: Mostrar el total de la suma de los
                 cuartos. */
                escribir(z);
fin_programa sumatoria
```

Este ejemplo es muy sencillo, pero en él se pueden apreciar las características básicas de un algoritmo escrito en este lenguaje.

Lo primero que se nota es un encabezado de programa en donde está el identificador de programa, una descripción de lo que hace el programa, pero no cómo lo hace y una especificación en forma de precondición o postcondición.

Sigue una indicación de nivel, éste permite activar y desactivar ciertas características del lenguaje que dependen del nivel que va desarrollando el estudiante.

Ahora tenemos el bloque principal, que a su vez tiene dos bloques: el de declaración de variables y el algoritmo.

En el bloque de declaración de variables, como su nombre lo indica, se especifica cuáles son las variables que se van a utilizar en el algoritmo y de qué tipo son.

En el bloque de algoritmo se ponen las sentencias que componen al algoritmo que soluciona el problema.

Se puede observar en el algoritmo, aparte de las sentencias simples, dos tipos de ciclos distintos,

[5] El ejemplo fue realizado por el Ing. John Alexander Rojas Montero.

cada uno tiene especificado su invariante y cada ciclo es cerrado con una palabra específica del tipo de ciclo. Finalmente, se tiene el fin de programa que debe incluir el identificador de éste.

3.5 Otro ejemplo

Ahora se presenta otro ejemplo un poco más complejo, esta vez con declaración de funciones[6]:

```
programa millas_por_galón
        autor «John Rojas»
        código 79451716
        fecha 1999.01.19
        propósito «Encontrar el promedio de millas por galón en
        un automóvil»
        precondición pertenece(g,reales) y pertenece(m,reales)
        y g <> -1 y i>=1 y i <= n
        postcondición pertenece(p,reales) y p = sumatoria(1,n,g)
        nivel básico
subprogramas
        función acumular (entrada x,y_:real) : real
                autor «John Rojas»
                código 79151716
                fecha 1999.01.19
                propósito "Acumula millas por galón y con
                tabiliza el numero de tanqueadas"
                precondicion pertenece(x,reales) y
                pertenece(y_,reales) y x <> -1 y i >= 1 y i <= n
                postcondición z=sumatoria(1,n,x/y_) y
                pertenece(z,reales) y w=n y i>=1 y i<=n
        inicio
                algoritmo
                        /* Paso 1: Calcular las millas por
                        galón */
                        z := z + (y_ / x);
                        escribir(z);

                        /* Paso 2: calcular el numero de
                        tanqueadas */
                        w := w + 1;
        fin_función acumular;
inicio
        variables
                g : real; /* galones usados */
                m : real; /* millas conducidas */
                mg : real; /* millas por galón */
                nt : real; /* numero de tanqueadas */
                p : real; /* promedio millas por galón */
        algoritmo
                /* Paso 1: Leer los galones y las millas */
                mg := 0;
                nt := 0;
                repita
                        invariante verdadero
                        escribir("Digite el numero del ga
                        lones usados: ");
                        leer(g);
                        si g <> -1 entonces
                                escribir("Digite el nu
                                mero de millas condu
                                cidas");
                                leer(m);

                        /* Paso 2: Acumular el promedio de galones
                        y el numero de tanqueadas */
                                acumular(g,m);
                        fin_si;
                hasta g = -1;

                /* Paso 3: Calcular el promedio de millas por
                galón */
                p := mg / nt;

                /* Paso 4: Mostrar el promedio de millas por
                galón */
                escribir(p);
fin_programa millas_por_galón
```

Como se puede ver, una función es documentada con los mismos items que el programa: autor, código, fecha, propósito y la especificación, que en este caso corresponden a la precondición y la postcondición. Después viene el bloque y se finaliza con **fin_función** que además exige el identificador de función. Esta es una forma muy didáctica y muy sencilla para no perderse en programas demasiado anidados.

3.6 Un ejemplo con TDA's

Veamos ahora un ejemplo con declaración de un TDA y su utilización:

```
programa  prueba_de_tda
        autor "L. Alejandro Bernal R."
        código 79302400
        fecha 1999.02.16
        propósito "Ilustrar el uso de TDA"
        precondicion verdadero
        postcondición verdadero
        nivel básico
tipos
        tda tipo_tda
                autor «L. Alejandro Bernal R.»
                código 79302428
                fecha 1999.02.26
                propósito «tad de prueba»
                invariante min < MAX y min >= 0 y max >= 0
                 y max < MAX
                constantes
                        MAX=100;
                        MIN=0;
                tipos
                        tipo_cadena= cadena[MAX];
                        tipo_arreglo=
                        arreglo[MIN](entero);
                variables
```

```
                    min: entero;
                    max: real;
            operaciones
                    procedimiento inicializa ()
                            autor "L. Alejandro
                            Bernal R."
                            código 79302428
                            fecha 1999.02.24
                            propósito "probar el
                            tda"
                            precondición verda
                            dero
                            postcondición verda
                            dero
                    inicio
                            algoritmo
                                min:= MIN;
                                max:= MAX;
                    fin_procedimiento inicializar;

                    procedimiento adicionar(entrada
                    inc:entero)
                            autor "L. Alejandro
                            Bernal R."
                            código 79302428
                            fecha 1999.02.26
                            propósito "Prueba de
                            operación de tda"
                            precondición verda
                            dero
                            postcondición verda
                            dero
                    inicio
                            algoritmo
                               max := max + inc;
                               min := min - inc;
                    fin_procedimiento adicionar;
        fin_tda tipo_tda;

inicio
        variables
                ap : tipo_tda;
        algoritmo
                ap.inicializar();
                ap.adicionar(20);
fin_programa prueba_tda
```

Se aprecia en particular la exigencia, por parte del lenguaje, del invariante de TDA, importante para la especificación de TDAs. Además, se exigen los items de documentación que tiene el programa, las funciones y procedimientos.

En los tres ejemplos anteriores se pueden apreciar varias características:

- Exige especificación formal de cada módulo del lenguaje. En otros lenguajes esto no es obligatorio y los estudiantes lo omiten o lo dejan para el final, y en últimas lo olvidan originando muchos errores.

- Cada bloque del programa con funciones distintas es encabezado con una palabra en español que indica su función (constantes, tipos, variables, algoritmo, etc.). De esta forma las distintas partes de un programa quedan claramente delimitadas y en consecuencia es más fácil encontrar algún segmento específico de código.

- Se exige documentar cada segmento significativo de código. Esto también tiene la virtud de que el estudiante no deje estos importantes detalles para después, o peor aún que no lo haga.

- Exige, también, que cada vez que se cierra un módulo (con fin) se especifique de qué clase es (fin_función, fin_procedimiento, fin_clase, fin_tda, etc), y además cuál específicamente, es decir, se debe indicar el nombre del módulo. También es una forma de documentación necesaria en especial cuando hay un nivel de anidamiento muy profundo o una gran cantidad de módulos.

- A lo largo de todo el código se tienen palabras en español que explican claramente lo que se está haciendo. Esto facilita la lectura del programa por una persona de habla española.

Todo lo anterior hace de un programa escrito en LES más fácil de entender, verificar y mantener, logrando ser una herramienta didáctica más efectiva que los lenguajes tradicionales de programación.

4. Formalización del lenguaje

El lenguaje LES es un lenguaje artificial libre del contexto[7]. Para que un lenguaje de este tipo esté perfectamente formalizado se requieren dos condiciones: una gramática y una semántica asociada a dicha gramática.

[7] En un lenguaje libre del contexto, la determinación de si una frase pertenece o no al lenguaje es independiente de los caracteres que lo rodean.

4.1 Gramática del lenguaje

Una buena definición formal de lo que es una gramática se encuentra en [Berstel90] y es como sigue:

> *Una gramática libre de contexto G = (V,A,P,S) está compuesta de un alfabeto finito V, un subconjunto A de V llamado alfabeto terminal, un conjunto finito de producciones P contenidas en (V-A), un elemento distinguido S perteneciente a V-A llamado axioma. Una letra en V-A es un símbolo no terminal o variable.* [Berstel90, pag 62]

En otras palabras, una gramática requiere de dos conjuntos: un alfabeto terminal y un conjunto de producciones. Veamos en que consiste cada uno:

4.1.1 Alfabeto terminal (A)

El alfabeto terminal es el conjunto de todas las palabras y símbolos básicos de un lenguaje, a éstos se les conoce como lexemas. A las reglas que dicen cuáles lexemas pertenecen al lenguaje se les llaman reglas léxicas. Una forma de expresar estas reglas léxicas es mediante expresiones regulares, una explicación detallada sobre esto se puede encontrar en [Aho90, Capítulo 3].

A continuación se presentan los símbolos utilizados en la especificación del lenguaje de expresiones regulares; los lexemas de LES, que son el alfabeto terminal del lenguaje, y finalmente una nota sobre la validez de esta especificación.

4.1.1.1 El lenguaje de expresiones regulares

Aquí se ha adoptado el lenguaje de expresiones regulares que utilizan Lex[8] y Flex[9], dos generadores de analizadores léxicos muy utilizados en la construcción de compiladores. La descripción detallada del lenguaje de expresiones regulares se encuentra en el man[10] de Flex que es [Paxon95], el documento oficial del Flex. La tabla No.1 nos muestra los símbolos que se utilizaron en la especificación de LES.

4.1.1.2 Los lexemas de LES

El Alfabeto terminal A del LES es una serie de lexemas que cumplen con las siguientes expresiones regulares:

Lexema	Expresión
DIGITO	[0-9]
LETRA	[a-zA-Z_]
CONSTANTE_ENTERA	{DIGITO}+
CONSTANTE_REAL	{CONSTANTE_ENTERA}("."{CONSTANTE_ENTERA})?(("e"|"E"){CONSTANTE_ENTERA})?
CONSTANTE_FECHA	{DIGITO}{4}"."{DIGITO}{2}"."{DIGITO}{2}
CONSTANTE_HORA	{DIGITO}{2}":"{DIGITO}{2}":"{CONSTANTE_REAL}
IDENTIFICADOR	{LETRA}({LETRA}|{DIGITO})*
CONSTANTE_CADENA	\"[^\n]*\"
COMENTARIO_INICIO	"/*"
COMENTARIO_FIN	"*/"
PROGRAMA	"programa"
FIN_PROGRAMA	"fin_programa"
AUTOR	"autor"
CODIGO	"codigo"
FECHA	"fecha"
HORA	"hora"
PRECONDICION	"precondicion"
POSTCONDICION	"postcondicion"
NIVEL	"nivel"
USAR	"usar"
INICIAL	"inicial"
BASICO	"basico"
MEDIO	"medio"
AVANZADO	"avanzado"
CONSTANTES	"constantes"
TIPOS	"tipos"
VARIABLES	"variables"
ALGORITMO	"algoritmo"
CARACTER	"caracter"
ENTERO	"entero"
REAL	"real"
BOOLEANO	"booleano"
APUNTADOR	"apuntador"

[8] Lex quiere decir "Lexical Analyzer Generator" generador de analizadores léxicos.

[9] Flex significa "Fast Lexical Analyzer Generator" generador de analizadores léxicos rápidos. Es una versión más rápida de su antecesor el Lex. Es un programa GNU, esto es, una versión particular de software libre, es decir, sin ningún costo y se encuentra para múltiples plataformas (unix, DOS,etc.).

[10] man es un comando que se encuentra en todos los sistemas Unix; significa manual y despliega la documentación sobre un comando Unix, en nuestro caso Flex.

REFERENCIA	"referencia"
CADENA	"cadena"
CONJUNTO	"conjunto"
ARREGLO	"arreglo"
ARCHIVO	"archivo"
ESTRUCTURA	"estructura"
FIN_ESTRUCTURA	"fin_estructura"
TDA	"tda"
FIN_TDA	"fin_tda"
CLASE	"clase"
FIN_CLASE	"fin_clase"
PROPOSITO	"proposito"
INVARIANTE	"invariante"
HERENCIA	"herencia"
OPERACIONES	"operaciones"
MENSAJES	"mensajes"
SUBPROGRAMAS	"subprogramas"
FUNCION	"funcion"
FIN_FUNCION	"fin_funcion"
PROCEDIMIENTO	"procedimiento"
FIN_PROCEDIMIENTO	"fin_procedimiento"
ASERCION	"asercion"
INICIO	"inicio"
FIN	"fin"
SI	"si"
ENTONCES	"entonces"
SI_NO	"si_no"
FIN_SI	"fin_si"
MIENTRAS_QUE	"mientras_que"
FIN_MIENTRAS_QUE	"fin_mientras_que"
HAGA	"haga"
PARA	"para"
HASTA	"hasta"
REPITA	"repita"
QUE	"que"
INCREMENTO	"incremento"
FIN_PARA	"fin_para"
ES	"es"
CASO	"caso"
OTROS	"otros"
FIN_OTROS	"fin_otros"
FIN_CASO	"fin_caso"
ENTRADA	"entrada"
SALIDA	"salida"
FALSO	"falso"
VERDADERO	"verdadero"
OP_ASIGNACION	":="
OP_COMPARACION	"="
OP_MAYOR	">"
OP_MAYOR_IGUAL	">="
OP_MENOR	"<"
OP_MENOR_IGUAL	"<="
OP_DIFERENTE	"<>"
OP_SUMA	"+"
OP_RESTA	"-"
OP_MULTIPLICACION	"*"
OP_DIVISION	"/"
OP_DIVISION_ENTERA	"div"
OP_MODULO	"mod"
OP_POTENCIACION	"^"
OP_AND	"y"
OP_OR	"o"
OP_NOT	"no"

4.1.1.3 Validez

La corrección y validez de las anteriores expresiones regulares ha sido verificada computacionalmente mediante el programa Flex y al mismo tiempo se construyó un verificador sintáctico de programas escritos en LES.

4.1.2 Conjunto de producciones (P)

Como se dijo la especificación de la gramática de un lenguaje artificial requiere de un alfabeto terminal y un conjunto de producciones. El conjunto de producciones P determinan la sintaxis del lenguaje. Esta sintaxis requiere de una metasintaxis o notación BNF para su especificación, como se verá en el siguiente punto. Asimismo, se presentarán en seguida las producciones BNF para el lenguaje LES, y por último se incluye una nota sobre la validez de dichas producciones.

4.1.2.1 Notación BNF

Para la especificación de la sintaxis del lenguaje, se utiliza la notación o meta-sintaxis BNF (Backus-Naur Form, ver [Naur63], [Aho90, pag 284] o [Wirth84, pag 86]). El dialecto particular que se usa aquí es el programa Bison[11], una descripción detallada se puede encontrar en [Donnely91].Esta notación se resume en el cuadro No.2.

[11] Bison:"The YACC-compatible Parser Generator". Bison es un generador de analizadores sintácticos. Se basa en la descripción gramatical del lenguaje. Los tipos de lenguaje que pueden manejar son los independientes de contexto del tipo LALR(1). Bison está basado en una versión anterior llamada Yacc, que quiere decir Yet Another Compiler-Compiler, otro compilador de compiladores más. Bison es un software GNU, que es un tipo especial de software libre, es decir, que está disponible para muchas plataformas (Unix, DOS, etc.) y no tiene costo.

CUADRO No.1

Metacaracter	Significado	Ejemplos	Explicación
x	El carácter x se toma literalmente, a menos que sea un metacaracter.	a c b f	Toma las letras literalmente
"cadena"	La cadena entre comillas se toma literalmete	"programa"	Toma la cadena programa literalmente
\x	El caracter x se toma literalmente, muy utilizado cuando se tiene que utilizar un metacaracter como parte de la especificación	* \+ \.	Los caracteres* + y . son operadores de las expresiones regulares, para usarse tienen que ir precedidos de \ o ponerse entre comillas.
.	Cualquier caracter excepto nueva línea	.	Acepta todos los caracteres menos nueva línea (\n).
[xyz]	Cualquiera de los caracteres en los corchetes.	[0123456789]	La expresión léxica para un dígito.
[^xyz]	Negación del anterior. Ninguno de los caracteres en los corchetes.	[^1234]	Cualquier caracter que no sea 1, 2, 3 ó 4.
[a-z]	Especifica un rango, es válido cualquiera de los caracteres en ese rango.	[a-zA-Z] [0-9]	Cualquier caracter alfabético Cualquier dígito.
r*	Cero o más veces el caracter r	[0-9]*	Expresión regular para número entero sin signo.
r+	Una o más veces el caracter r	[a-z]+	Expresión regular para palabra de sólo minúsculas.
r?	Cero o una vez el caracter r	+?	El + opcionalmente.
r|m	O el caracter r ó el m, exclusivo	+|-	O el más o el menos, pero no los dos a la vez.
( )	Utilizado para cambiar la prioridad de los operadores	(+|-)?[0-9]+	Expresión regular para entero con signo opcional.
{macro}	Macroexpansión del identificador entre corchetes.	DIGITO [0-9] {DIGITO}+	Equivalente a entero sin signo

CUADRO No.2

Simbolo	Significado	Ejemplos	Explicación
;	Punto y coma. Termina una producción.	asignación : referencia OP_ASIGNACION Expresión;	Aquí la definición de la categoría gramatical **Asignación** comienza con **asignación** y termina en punto y coma.
:	Se lee **se define como**. Lo que está a la izquierda del símbolo se define sintácticamente como la cadena de símbolos de la derecha.	declaración_de_variable : IDENTIFICADOR ':' tipo ';' ;	La declaración de variable se define como un **identificador** seguido de dos puntos seguido de un **tipo** y terminado en punto y coma.
identificador	Un identificador nombra una categoría gramatical. Si aparece a la de derecha del símbolo : quiere decir que se refiere a otra producción en la cual debe aparecer a la izquierda. Si está escrito en minúsculas se refiere a otra regla sintactica y si está en mayúsculas se refiere a una regla léxica.	sentencia : asignación | aserción ; asignación : referencia OP_ASIGNACION expresión ;	En la primera producción se está definiendo **Sentencia**, porque está a la izquierda de :, en términos de **asignación** y **aserción**, porque están a la derecha.
|	Selección. Se puede escoger entre cualquiera de los símbolos que aparecen antes o después. Esta escogencia es un ó exclusivo, o sea, sólo se puede seleccionar una de las alternativas.	<Sentencia> : <Asignación> | <Aserción>;	Aquí una sentencia puede ser o una **Asignación** o una **Aserción.**
'a'	El caracter entre comillas se toma literalmente.	';'	Aquí se toma literalmente el punto y coma y no su significado en la meta-sintaxis BNF el de terminar producción.

4.1.2.2 Producciones BNF para el lenguaje LES

Las siguientes son las producciones que definen la sintaxis para el lenguaje LES:

```
programa:encabezado_de_programa
        bloque_de_declaración_de_librerias
        bloque_de_declaración_de_constantes
        bloque_de_declaración_de_tipos
        bloque_de_declaración_de_subprogramas
        INICIO
                bloque_de_declaración_de_variables
                bloque_algoritmo
        FIN_PROGRAMA IDENTIFICADOR
        FIN_ARCHIVO
        ;
encabezado_de_programa:identificación_de_programa
                       documentación
                       especificación
                       NIVEL nivel
                      ;
documentación:AUTOR CONSTANTE_CADENA
              CÓDIGO CONSTANTE_ENTERA
              FECHA CONSTANTE_FECHA
              PROPÓSITO CONSTANTE_CADENA
               ;
identificación_de_programa:PROGRAMA IDENTIFICADOR
                         |     PROGRAMA IDENTIFICADOR
'('declaración_de_parámetros')'
                          ;
declaración_de_parámetros:declaración_de_parámetros_de_entrada
                                declaración_de_
                                parámetros_de_salida
                          ;
declaración_de_parámetros_de_entrada:ENTRADA lista_de_
                                declaración_de_parámetros
                                  |
                                  ;
declaración_de_parámetros_de_salida:SALIDA lista_de_
                                declaración_de_parámetros
                                  |
                                  ;
lista_de_declaración_de_parámetros:lista_de_declaracion_de_variables
                                  ;
nivel:    INICIAL
      |   BÁSICO
      |   MEDIO
      |   AVANZADO
      ;
especificación:     PRECONDICIÓN expresión_lógica
                    POSTCONDICIÓN expresión_lógica
              ;
```

```
bloque_de_declaración_de_librerias:USAR lista_de_
                                identificadores_cualificados ';'
                                  |
                                  ;
lista_de_identificadores_cualificados:lista_de_identificadores_
                                cualificados ';'identificador_
                                cualificado
                                      |  identificador_
                                         cualificado
                                ;
identificador_cualificado:      identificador_cualificado '.'
                                IDENTIFICADOR
                                |       IDENTIFICADOR
                                ;
bloque_de_declaración_de_constantes:CONSTANTES lista_de_
                                declaración_de_
                                 constantes ';'
                                 |
                                 ;
lista_de_declaración_de_constantes: lista_de_declaracion_de_
                                constantes ';' declaración_
                                de_constante
                                 |      declaración_de_
                                        constante
                                 ;
declaración_de_constante:       IDENTIFICADOR OP_
                                COMPARACIÓN constante
                                ;
bloque_de_declaración_de_tipos:TIPOS lista_de_
                                declaración_de_tipos ';'
                                |
                                ;
lista_de_declaración_de_tipos:  lista_de_declaración_de_tipos
                                ';' declaración_de_tipo
                                | declaración_de_tipo
                                        ;
declaración_de_tipo:        IDENTIFICADOR OP_
                            COMPARACIÓN tipo
                            |  tipo_estructurado
                            ;
bloque_de_declaración_de_variables:     VARIABLES lista_
                                        de_declaración_
                                        de_variables ';'
                                        |
                                        ;
lista_de_declaración_de_variables:      lista_de_
                                        declaración_de_
                                        variables ';'
                                        declaración_de_
                                        variable
                                        |       declaración
                                                _de_variable
                                        ;
declaracion_de_variable     :   lista_de_identificadores
                                ':' tipo
                            ;
lista_de_identificadores    :   lista_de_identificadores ','
                                IDENTIFICADOR
                                |       IDENTIFICADOR
                                ;
tipo        :       CARACTER
            |       ENTERO
            |       REAL
            |       FECHA
            |       HORA
```

```
            |   BOOLEANO
            |   tipo_apuntador
            |   tipo_referencia
            |   tipo_cadena
            |   tipo_arreglo
            |   tipo_archivo
            |   tipo_conjunto
            |   tipo_estructurado
            |   IDENTIFICADOR
            ;
tipo_estructurado   :   tipo_estructura
                    |   tipo_tda
                    |   tipo_clase
                    ;
tipo_apuntador      :   APUNTADOR '(' tipo ')'
                    ;
tipo_referencia     :   REFERENCIA '(' tipo ')'
                    ;
tipo_cadena         :   CADENA '[' expresión_
                        constante ']'
                    ;
tipo_conjunto       :   CONJUNTO '(' tipo ')'
                    ;
constante_conjunto  :   "{" lista_de_elementos "}"
                        ;
lista_de_elementos  :   lista_de_elementos ',' expresión
                    |   expresión
                        ;
expresión_constante :   expresión
                        ;
tipo_arreglo        :   ARREGLO '[' expresión_
                        constante ']' '(' tipo ')'
                    ;
tipo_archivo        :   ARCHIVO '(' tipo ')'
                    ;
tipo_estructura     :   ESTRUCTURA
                        IDENTIFICADOR
                            lista_de_
                            declaracion_de_
                            variables ';'
                        FIN_ESTRUCTURA
                        IDENTIFICADOR
                    ;
tipo_tda    :       TDA IDENTIFICADOR
                            encabezado_de_tda
                            bloque_de_
                            declaración_de_
                            constantes
                            bloque_de_
                            declaración_de_tipos
                            bloque_de_
                            declaración_de_
                            variables
                            bloque_de_
                            declaración_de_
                            operaciones
                        FIN_TDA IDENTIFICADOR
                    ;
tipo_clase :        CLASE IDENTIFICADOR

                                encabezado_de_clase
                                  bloque_de_
                                  declaración_de
                                  _herencia
                                   bloque_de_
                                        declaración_de
                                        _relaciones
                                        bloque_de_
                                        declaración_de_
                                        atributos
                                        bloque_de_
                                        declaracion_de_
                                        mensajes
                                    FIN_CLASE IDENTIFICADOR
                                ;
encabezado_de_clase:        documentación
                                    invariante
                            ;
invariante :            INVARIANTE expresión_lógica
                            ;
encabezado_de_tda   :       encabezado_de_clase
                            ;
bloque_de_declaración_de_herencia:HERENCIA lista_de_
                                  identificadores ';'
                                  |
                                  ;
bloque_de_declaración_de_relaciones:RELACIONES lista_de_
                                  declaración_de_variables ';'
                                  |
                                  ;
bloque_de_declaración_de_atributos:ATRIBUTOS lista_de_
                                  declaración_de_variables ';'
                                  |
                                  ;
bloque_de_declaración_de_mensajes:MENSAJES lista_de_
                                  declaración_de_
                                  subprogramas ';'
                                  |
                                  ;
bloque_de_declaración_de_operaciones:OPERACIONES lista_
                                    de_declaración_de_
                                    subprogramas ';'
                                  |
                                  ;
bloque_de_declaración_de_subprogramas:SUBPROGRAMAS
                                    lista_de_declaración_
                                    de_subprogramas ';'
                                  |
                                  ;
lista_de_declaración_de_subprogramas:lista_de_declaración_
                                    de_subprogramas ';'
                                   declaración_de_
                                   subprograma
                                  |     declaración_de_
                                        subprograma
                                   ;
declaración_de_subprograma:  declaración_de_función
                             |  declaración_de_
                                procedimiento
                                    ;
declaración_de_función:      FUNCIÓN IDENTIFICADOR '('
                             declaración_de_parametros ')'
                             ':' tipo
                             documentación
                             especificación
                             bloque_de_declaración_
                             de_constantes
                             bloque_de_declaración_
                             de_tipos
                             bloque_de_declaración_
```

```
                        de_subprogramas
                        INICIO
                                bloque_de_
                                declaración_de_
                                variables
                                bloque_
                                algoritmo
                        FIN_FUNCIÓN
                        IDENTIFICADOR
                ;
declaración_de_procedimiento: PROCEDIMIENTO
                        IDENTIFICADOR
                        '(' declaración_de
                        _parametros ')'
                                documentación
                                especificación
                                bloque_de_
                                declaración_de_
                                constantes
                                bloque_de_
                                declaración_de_
                                tipos
                                bloque_de_
                                declaración_de_
                                subprogramas
                        INICIO
                                bloque_de_
                                declaración_de_
                                variables
                                bloque_algoritmo
                        FIN_PROCEDIMIENTO
                        IDENTIFICADOR
                ;
bloque_algoritmo:   ALGORÍTMO lista_de_sentencias ';'
                    |
                    ;
lista_de_sentencias:  lista_de_sentencias ';' sentencia
                    |       sentencia
                    |
                    ;
sentencia:          asignación
                    |   aserción
                    |   llamada_a_procedimiento
                    |   si
                    |   si_multiple
                    |   mientras_que
                    |   repita_hasta
                    |   para
                    |   bloque_de_sentencias
                    ;
bloque_de_sentencias:       INICIO
                                bloque_
                                de_declaración_
                                de_variables
                                bloque_algoritmo
                            FIN
                    ;
asignación:         referencia OP_ASIGNACIÓN expresión
                    ;
aserción    :       ASERCIÓN expresión_lógica
                    ;
llamada_a_procedimiento     :       llamada_a_función
                            |       llamada_a_operación
                                    _o_mensaje
                            ;
```

```
si          :       SI expresión_booleana ENTONCES
                            lista_de_sentencias ';'
                    FIN_SI
                |   SI expresión_booleana ENTONCES
                            lista_de_sentencias ';'
                    SI_NO
                            lista_de_sentencias ';'
                    FIN_SI
            ;
mientras_que        :       MIENTRAS_QUE
                            expresión_booleana
                            invariante
                            HAGA
                                    lista_de_sentencias
                                    ';'
                            FIN_MIENTRAS_QUE
                ;
repita_hasta        :       REPITA
                                    invariante
                                    lista_de_sentencias
                                    ';'
                            HASTA expresión_booleana
                            ;
para        :       PARA referencia OP_ASIGNACIÓN
                    expresión HASTA expresión
                    INCREMENTO expresión
                    invariante
                    HAGA
                            lista_de_sentencias ';'
                    FIN_PARA
                ;
si_multiple:                SI referencia ES
                                    lista_de_casos ';'
                            FIN_SI
                    |       SI referencia ES
                                    lista_de_casos ';'
                                    OTROS ENTONCES
                                            lista_de_
                                            senten
                                            cias ';'
                                    FIN_OTROS ';'
                            FIN_SI
                ;
lista_de_casos      :       lista_de_casos ';' caso
                    |       caso
                    ;
caso        :       CASO expresión ENTONCES
                            lista_de_sentencias ';'
                    FIN_CASO
                ;
expresión_booleana  :       expresión_disyuntiva
                            ;
expresión   :       expresión_disyuntiva
                    ;
expresion_lógica    :       expresión_disyuntiva
                            ;
expresión_disyuntiva :      expresión_disyuntiva OP_OR
                            expresión_conjunción
                            |       expresión_
                                    conjunción
                            ;
expresión_conjunción        :       expresión_
                                    conjunción OP_AND
                                    expresión_de_
                                    comparación
```

```
                        |       expresión_
                                de_comparación
                        ;
expresión_de_comparación    :   expresión_aditiva
                                OP_COMPARACIÓN
                                expresión_aditiva
                            |   expresión_
                                aditiva OP_
                                MAYOR
                                expresión_aditiva
                            |   expresión_
                                aditiva OP_
                                MAYOR_
                                IGUAL
                                expresión_aditiva
                            |   expresión_aditiva
                                OP_MENOR
                                expresión_aditiva
                            |   expresión_aditiva
                                OP_MENOR_
                                IGUAL
                                expresión_aditiva
                            |   expresión_aditiva
                                OP_DIFERENTE
                                expresión_aditiva
                            |   expresión_aditiva
                                ;
expresión_aditiva       :   expresión_aditiva operador_
                            aditivo término
                            |       término
                            ;
operador_aditivo        :   OP_SUMA
                            |       OP_RESTA
                            ;
término     :   término operador_multiplicativo factor
            |   factor
            ;
operador_multiplicativo     :   OP_MULTIPLICACIÓN
                            |   OP_DIVISIÓN
                            |   OP_DIVISIÓN_
                                ENTERA
                            |   OP_MÓDULO
                            ;
factor      :   OP_NOT factor
            |   OP_RESTA factor
            |   constante
            |   '(' expresión ')'
            |   llamada_a_función
            |   llamada_a_operación_o_mensaje
            |   referencia
            ;
constante   :   CONSTANTE_ENTERA
                |       CONSTANTE_REAL
                |       CONSTANTE_FECHA
                |       CONSTANTE_HORA
                |       CONSTANTE_CADENA
                |       constante_conjunto
                |       FALSO
                |       VERDADERO
                ;
referencia  :   IDENTIFICADOR '[' expresión ']'
                |       IDENTIFICADOR '.'
IDENTIFICADOR
                |       '*' referencia
                |       '&' referencia
                |       IDENTIFICADOR
                ;
llamada_a_función   :   IDENTIFICADOR '(' lista_
                        parámetros_actuales ')'
                        ;
llamada_a_operación_o_mensaje       :IDENTIFICADOR
                                    '.' IDENTIFICADOR
                                    '(' lista_parámetros_
                                    actuales ')'
                                    ;
lista_parámetros_actuales   :   lista_parámetros_
                                actuales ',' expresión
                            |       expresión
                            |
                            ;
```

4.1.2.3 Validez

La validez de la sintaxis se ha verificado computacionalmente mediante el uso del programa Bison y como se indicó, al mismo tiempo se construyó un analizador sintáctico que permitirá verificar programas escritos en LES.

4.2 Semántica del lenguaje

Retomando lo anterior, la formalización del lenguaje requiere de una gramática y una semántica asociada a dicha gramática. La semántica del lenguaje define el significado de las declaraciones y sentencias del lenguaje. Esto se puede hacer de varias formas: una informal mediante sentencias en lenguaje natural[12]; una pseudoformal indicando la traducción a otro lenguaje o formalismo, como los diagramas de flujo y otra formal, mediante el uso de semántica denotacional (ver esto en detalle en [Mosses90]).

En cuanto a las realizaciones adelantadas en este componente, se está trabajando la notación de la semántica en notación UML[13], utilizandola para definir una máquina virtual LES que define sin ambigüedades el comportamiento de un programa escrito en el lenguaje. Más adelante, implementando ésta máquina se tendría un medio para ejecutar programas LES.

---

[12] Para nuestro caso el lenguaje natural es el español.

[13] UML: Unified Modeling Language. Lenguaje de modelamiento unificado. Este lenguaje es una notación gráfica basada en el paradigma de objetos. Ver en más detalles en la dirección http://www.rational.com/uml.

5. Estado actual y trabajo futuro

Hasta el momento el lenguaje se ha aplicado a cuatro cursos de programación, dos iniciales y uno avanzado. La aplicación de éste ha inspirado mejoras en el lenguaje, como por ejemplo la eliminación de las variables globales. El lenguaje está sometido permanentemente a pruebas y críticas por parte de los docentes e investigadores de la EAN, por lo que se mantiene en permanente evolución.

En la actualidad, se tiene formalizada la totalidad de la sintaxis del lenguaje. Se tiene listo un verificador sintáctico de programas escritos en LES, que como su nombre lo indica, verifica que un programa particular esté escrito en lenguaje LES y se utiliza como primer paso en la corrección de proyectos y trabajos realizados por los estudiantes. En el momento no hace verificaciones de tipo semántico como es el caso determinar si un identificador ha sido previamente declarado o la verificación de tipos. Se está trabajando en esto último.

También se está desarrollando la documentación del lenguaje, específicamente el manual del lenguaje. Asimismo, los profesores de Ciencias de la computación están creando un banco de problemas cuyas soluciones están implementadas en el lenguaje.

En el momento se está desarrollando la especificación de la máquina virtual LES en UML. A mediano plazo, cuando se haya decantado el lenguaje, se abordará la construcción de un interpretador basado en la definición de ésta máquina virtual. A largo plazo, se piensa en un interpretador, que basado en el anterior, pueda por ejemplo realizar corridas gráficas de los algoritmos.

Bibliografía

[Aho90] Aho V., Alfred; Sethi, Ravi; Ullman, Jeffrey D. Compiladores : Principios, técnicas y herramientas. Addison-Wesley Iberoamericana, Wilmington, Delawere, E.U.A. 1990, primera edición en español.

[Amador98] Amador Montaño, José Francisco. Metodología para la solución de problemas algorítmicos. En revista EAN, Diciembre 1998, No. 35. pags. 64-76.

[Berstel90] Berstel, J.; Boasson, L. Context-Free Languages. En Handbook of Theoretical Computer Science. Volumen B: Formal Models and Semantics. The MIT Press 1990.

[Cardoso86] Cardoso, R. Diseño e implementación de Tipos abstractos de datos Memo de investigación No 9, UniAndes, 1986.

[Cardoso87] Cardoso, R. Práctica en Tipos abstractos de datos Memo de investigación No 71, UniAndes, 1986.

[Booch94] Booch, G. Object Oriented Design, Benjamin/Cummings, New York.

[Dahl73] Dahl, O. J.; Dijkstra; Hoare C. A. R. Structured Programming. Academic Press, 1973.

[Dijkstra68] Dijkstra, E. W. Go to statement considered harmful. Comm. Assoc. Comp. Mach., 11 (1968, pp. 147'148).

[Dijkstra72] Dijkstra, E. W. Notes on Structured Programming. Structured Programming. Academic Press, New York, 1972.

[Dijkstra76] Dijkstra, E. W. A discipline of programming.Prentice Hall 1976.

[Donnelly91] Donnelly, Charles; Stallman, Richard. Bison : The YACC-compatible Parser Generator. Dic. 1991, Bison Version 1.16, manual electrónico que viene con el software.

[Paxson95] Paxson, Vern. flex - fast lexical analyzer generator. Manual electrónico en los sistemas Unix y similares, Abril 1996, ver 2.5

[Tucker94] Tucker, Allen B.; Cupper, Robert D.; Bradley, W. James; Garnick, David K. Fundamentos de Informática: Lógica, resolución de problemas, programas y computadoras. México: Mc Graw Hill, 1994. 392 p.

[Jensen86] Jensen, K; Wirth, Niklaus. Pascal. Manual del usuario e informe. El Ateneo, Buenos Aires, 1985.

[Hoare72] Hoare, C. A. R. Notes on Data Structuring. Structured Programming. Academic Press, New York, 1972.

[Knuth87] Knuth, D. E. The art of Computer Programming. Vol. 1, Fundamental Algorithms, 1996. Vol. 2, Sorting and Searching. 1972. Addison Wesley.

[Mosses90] Mosses, Peter D.Denotacional Semantics. En Handbook of Theoretical Computer Science. Volumen B: Formal Models and Semantics. The MIT Press 1990.

[Naur63] Naur P. Revised report on the algorithmic language Algol 60. En Comm. ACM 6:1, págs. 1-17.

[Rumbaugh91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. Object-oriented Modelling and Design. Prentice-Hall, Englewood Cliffs, N.J.

[Villalobos90] Villalobos, Jorge. Estructuras de Datos : Un enfoque desde los tipos abstractos. Ediciones uniandes, Bogotá, 1990.

[Wirth84] Wirth, Niklaus; Jensen, Kathleen. Pascal : Manual del usuario e informe. De El Ateneo, 2a edición, Buenos Aires, 1984.

[Wirth87] Wirth, Niklaus, Algoritmos + Estructuras de datos = Programas. Prentice Hall, 1987.