Examen Parcial

Docente	Carrera	Curso
M.Sc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

Semestre	Tema	Estudiante
V	Presentación del lenguaje propuesto	Andrea del Rosario Velazco Yana

2024 - I

Índice

1. Introducción

1.1. Justificación

En el presente trabajo, se aborda el desafío de no solo comprender las especificaciones de un lenguaje de programación, sino también de reducir la complejidad inherente a la programación para hacerla más accesible a una variedad de personas. Reconociendo la diversidad de habilidades, antecedentes y contextos de los usuarios de la programación, se está desarrollando un nuevo lenguaje con el objetivo primordial de ser inclusivo y comprensible para todos, independientemente de su nivel de experiencia o dominio del tema.

Este nuevo lenguaje se concibe con una visión clara: servir como herramienta facilitadora tanto para la comprensión de los conceptos fundamentales de la programación como para la implementación práctica de compiladores. La intención es no solo enseñar a comprender el funcionamiento interno de los compiladores, sino también simplificar el proceso de desarrollo y depuración de programas, haciendo que la programación sea más accesible y menos intimidante para una audiencia más amplia.

Al ofrecer una interfaz intuitiva y simplificada, se espera que este nuevo lenguaje fomente una mayor participación en la programación, especialmente entre aquellos que podrían haberse sentido excluidos o desalentados por la complejidad de los lenguajes existentes. Al hacer que la programación sea más comprensible y menos abstracta, se busca eliminar barreras y fomentar la diversidad en la comunidad de programadores, permitiendo que personas con una variedad de antecedentes y habilidades contribuyan de manera significativa al desarrollo tecnológico.

Este proyecto no solo aspira a enseñar programación, sino también a democratizarla, promoviendo la inclusión y la accesibilidad en el mundo de la tecnología mediante un enfoque innovador y centrado en el usuario.

1.2. Objetivos

- Conocer las bases de la teoría de la computación para la implementación de un compilador.
- Entender como convertir un autómata finito no determinista a un autómata finito determinista.
- Desarrollar un nuevo lenguaje de programación que sea accesible y comprensible para una amplia gama de personas, independientemente de su experiencia previa en programación.
- Diseñar una sintaxis clara y concisa que simplifique el proceso de escritura y comprensión del código.

2. Especificación Lexica

Se detallara la especificacion lexica del programa creado. Comprende los siguientes aspectos:

2.1. Definición de los comentarios

Dentro del lenguaje "sharm", si el programador desee realizar un comentario, lo puede hacer utilizando el siguiente formato:

Listing 1: Definición de comentarios de una linea

```
1   /- Este es un comentario.
```

Suponiendo otro caso, en el que el programador desee realizar un comentario mas amplio, utilizando el tabulador para separarlo en mas de una linea, debera utilizar el siguiente formato:

Listing 2: Definición de comentarios de bloque

```
3   /- Este es un comentario de bloque.
4   Para abarcar varias lineas. -/
```

En el primer caso, basandonos en un comentario corto (de una linea) debe emplear /-"para comenzar el comentario y pasar a la siguiente linea para terminarlo.

En el segundo caso, basandonos en que el programador desee un comentario mas amplio debe emplear /-"para comenzar y de igual manera colocar al final /"para determinar el fin del comentario en otra linea.

2.2. Definición de los identificadores

En el lenguaje "sharm" los identificadores deben comenzar con una letra y tambien pueden contener dígitos y el carácter de subrayado. No se permiten caracteres especiales. Por ejemplo:

Listing 3: Definición de identificadores

```
6   nombre
7   nombre_1
```

2.3. Definición de las palabras clave

Listing 4: Definición de palabras clave

```
func => funcion
id => identificador
ikey => nombre de funcion
return => devolver
b_for => para
b_while => mientras
b_do => repetir
if => si
else => sino
print => imprimir
read => leer
t_int => numero entero
t_float => numero decimal
t_string => texto
```

```
t_bool => booleano
par_izq => (
par_der => )
llave_izq => [
llave_der => ]
coment_linea => comentario (una linea o inicio)
coment_cierre => cierre de comentario
```

2.4. Definición de los literales

Las cadenas de texto se debe realizar utilizando las comillas, así como:

Listing 5: Definición de literales

```
31  "Cadena en sharm."
```

2.5. Definición de los operadores

Los operadores que se emplean para realizar las diversas operaciones son los siguientes:

Listing 6: Definición de operadores

```
op_suma: +
op_resta: -
op_div: /
op_mult: *
op_equal: =
op_equalequal: ==
op_menor: <
op_menorequal: <=
op_mayor: >
op_mayorequal: >=
and: operador logico (y)
or: operador logico (O)
```

2.6. Expresiones regulares

A continuacion de mostrar una tabla con las expresiones reguales en el lenguaje ”sharm”:

Tabla 1: Expresiones Regulares

Token	**Expresión regular**
id	[a-z][a-Z0-9]*
num	[0-9]+
tex	[a-Z]+
op_suma	’+’
op_mult	’*’
par_izq	’(’
par_der	’)’
op_equal	’=’
op_rest	’-’
op_div	’/’
op_menor	’¡’
op_menorequal	’=’
op_mayorequal	’¿=’
op_mayor	’¿’
op_equalequal	’==’
op_com	’,’
llave_izq	’’
llave_der	’’
and	’y’
or	’o’
coment_linea	’/-’
coment_cierre	’-/’
fin_linea	’,’
return	’return’
func	’funcion’
if	’if’
else	else’
print	’print’
read	’read’
b_for	’for’
b_while	’while’
b_do	’do’
t_int	’num’
t_float	-[0-9]+ (,[0-9]+)
t_bool	’true — false’
t_string	’string’

2.7. Ejemplos de código

- Se mostraran diferentes ejemplos de como se podria implementar el lenguaje "sharm"para diferentes situaciones

2.7.1. Ejemplo 1: Hola mundo

Listing 7: Hola mundo en sharm

```
46      print ("Hola mundo")
```

2.7.2. Ejemplo 2: Factorial Iteratitivo

Listing 8: Factorial Iterativo en sharm

```
49    func factorial [int n] (
50        fact = 1,
51        for [int i = 1; i <= n; i++] (
52            fact = fact*i),
53        return fact,
54    )
```

2.7.3. Ejemplo 3: Factorial Recursivo

Listing 9: Factorial Recursivo en sharm

```
56    func fact [int n] (
57        if n==0 or n==1 (
58            return=1,
59            )
60        else (
61            result=n*fact(n-1),
62            return result,
63        )
64    )
```

3. Gramática

Listing 10: Gramática

```
MAIN       -> func par_izq STATEMENT_M par_der

STATEMENT_M -> STATEMENT STATEMENT_M
STATEMENT_M -> ''

STATEMENT  -> id OPER E
STATEMENT  -> WHILE_CTRL
STATEMENT  -> IF_CTRL
STATEMENT  -> IMPRIMIR
STATEMENT  -> FUNCION
STATEMENT  -> RETORNO
```

```
FUNCION    -> ikey llave_izq E op_com TERM llave_der


WHILE_CTRL -> while llave_izq E STATEMENT_M llave_der

IF_CTRL -> if llave_izq E IMPRIMIR IFELSE_CTRL llave_der
IFP_CTRL -> IMPRIMIR IFELSE_CTRL




IFELSE_CTRL -> else llave_izq IMPRIMIR llave_der
IFELSE_CTRL -> ''

IMPRIMIR   -> print llave_izq E llave_der

RETORNO    -> return llave_izq E STATEMENT_M llave_der E' ASIGNACION

ASIGNACION -> id OPER E


E          -> T E'
E          -> FUNCION

E'         -> OPER T E'
E'         -> ''
T          -> TERM
T          -> llave_izq E llave_der


TERM       -> id TERM_FUNC
TERM       -> num
TERM       -> bool
TERM       -> text
TERM_FUNC  -> llave_izq PARAM_M llave_der
TERM_FUNC  -> ''
OPER       -> op_suma
OPER       -> op_equal
OPER       -> op_equalequal
OPER       -> op_div
OPER       -> op_rest
OPER       -> op_menor
OPER       -> and
OPER       -> or

PARAM_M    -> E PARAM_E
PARAM_M    -> ''
PARAM_E    -> op_com E PARAM_E
PARAM_E    -> ''
```

3.1. Ejemplos

1. Ejemplo 1

```
func par_izq id op_suma id par_der
```

Trace		
Stack	**Input**	**Rule**
$ MAIN	func par_izq id op_suma id par_der $	
$ par_der STATEMENT_M par_izq func	func par_izq id op_suma id par_der $	MAIN -> func par_izq STATEMENT_M par_der
$ par_der STATEMENT_M par_izq	par_izq id op_suma id par_der $	
$ par_der STATEMENT_M	id op_suma id par_der $	
$ par_der STATEMENT_M STATEMENT	id op_suma id par_der $	STATEMENT_M -> STATEMENT STATEMENT_M
$ par_der STATEMENT_M E OPER id	id op_suma id par_der $	STATEMENT -> id OPER E
$ par_der STATEMENT_M E OPER	op_suma id par_der $	
$ par_der STATEMENT_M E op_suma	op_suma id par_der $	OPER -> op_suma
$ par_der STATEMENT_M E	id par_der $	
$ par_der STATEMENT_M E' T	id par_der $	E -> T E'
$ par_der STATEMENT_M E' TERM	id par_der $	T -> TERM
$ par_der STATEMENT_M E' TERM_FUNC id	id par_der $	TERM -> id TERM_FUNC
$ par_der STATEMENT_M E' TERM_FUNC	par_der $	
$ par_der STATEMENT_M E'	par_der $	TERM_FUNC -> ''
$ par_der STATEMENT_M	par_der $	E' -> ''
$ par_der	par_der $	STATEMENT_M -> ''
$	$	

Figura 1: Ejemplo1

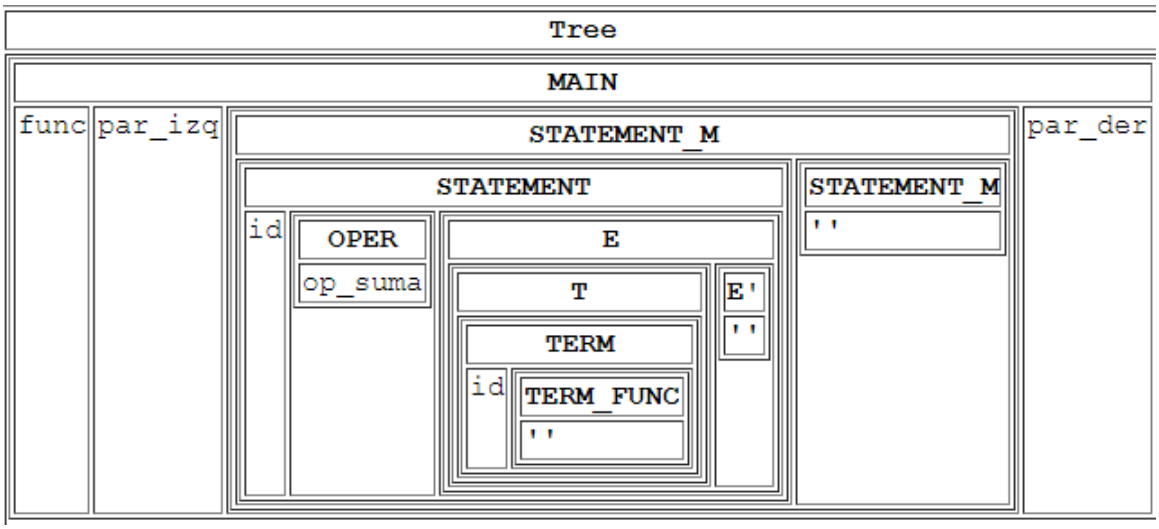

Figura 2: Ejemplo1

1. Ejemplo 2

```
func par_izq
    id op_equal bool
    if llave_izq
        id op_equalequal bool
        print llave_izq text llave_der
    llave_der
par_der
```

Trace

Stack	Input	Rule
$ MAIN	func par_izq id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M par_izq func	func par_izq id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	MAIN -> func par_izq STATEMENT_M par_der
$ par_der STATEMENT_M par_izq	par_izq id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M	id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M STATEMENT	id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	STATEMENT_M -> STATEMENT STATEMENT_M
$ par_der STATEMENT_M E OPER id	id op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	STATEMENT -> id OPER E
$ par_der STATEMENT_M E OPER	op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M E op_equal	op_equal bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	OPER -> op_equal
$ par_der STATEMENT_M E	bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M E' T	bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	E -> T E'
$ par_der STATEMENT_M E' TERM	bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	T -> TERM
$ par_der STATEMENT_M E' bool	bool if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	TERM -> bool
$ par_der STATEMENT_M E'	if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M	if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	E' -> ''
$ par_der STATEMENT_M STATEMENT	if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	STATEMENT_M -> STATEMENT STATEMENT_M
$ par_der STATEMENT_M IF_CTRL	if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	STATEMENT -> IF_CTRL
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E llave_izq if	if llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	IF_CTRL -> if llave_izq E IMPRIMIR IFELSE_CTRL llave_der
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E llave_izq	llave_izq id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E	id op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' T	id op_equalequal bool print llave_izq text llave_der llave_der par_der $	E -> T E'
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' TERM	id op_equalequal bool print llave_izq text llave_der llave_der par_der $	T -> TERM
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' TERM_FUNC id	id op_equalequal bool print llave_izq text llave_der llave_der par_der $	TERM -> id TERM_FUNC
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' TERM_FUNC	op_equalequal bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E'	op_equalequal bool print llave_izq text llave_der llave_der par_der $	TERM_FUNC -> ''
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' T OPER	op_equalequal bool print llave_izq text llave_der llave_der par_der $	E' -> OPER T E'
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' T op_equalequal	op_equalequal bool print llave_izq text llave_der llave_der par_der $	OPER -> op_equalequal
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' T	bool print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' TERM	bool print llave_izq text llave_der llave_der par_der $	T -> TERM
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E' bool	bool print llave_izq text llave_der llave_der par_der $	TERM -> bool
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR E'	print llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL IMPRIMIR	print llave_izq text llave_der llave_der par_der $	E' -> ''
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E llave_izq print	print llave_izq text llave_der llave_der par_der $	IMPRIMIR -> print llave_izq E llave_der
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E llave_izq	llave_izq text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E	text llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E' T	text llave_der llave_der par_der $	E -> T E'
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E' TERM	text llave_der llave_der par_der $	T -> TERM
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E' text	text llave_der llave_der par_der $	TERM -> text
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der E'	llave_der llave_der par_der $	
$ par_der STATEMENT_M llave_der IFELSE_CTRL llave_der	llave_der llave_der par_der $	E' -> ''
$ par_der STATEMENT_M llave_der IFELSE_CTRL	llave_der par_der $	
$ par_der STATEMENT_M llave_der	llave_der par_der $	IFELSE_CTRL -> ''
$ par_der STATEMENT_M	par_der $	
$ par_der	par_der $	STATEMENT_M -> ''
$	$	

Figura 3: Ejemplo2

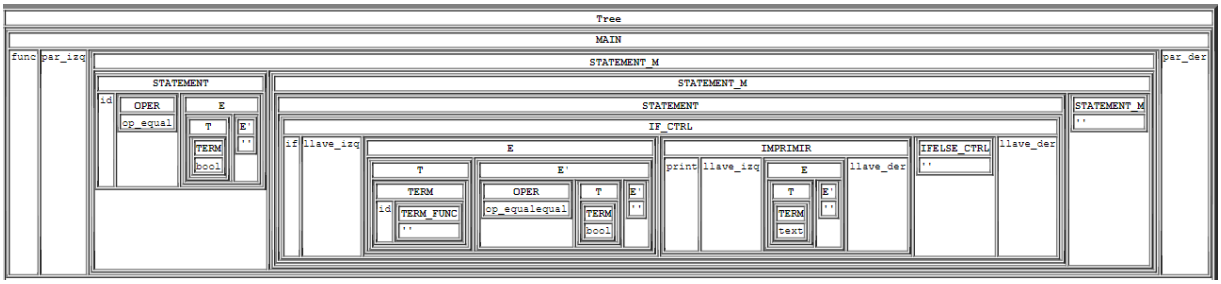

Figura 4: Ejemplo2

1. Ejemplo 3

```
func par_izq
    ikey llave_izq id op_com id llave_der
        return llave_izq id op_suma id llave_der op_div num

    id op_equal num
    while llave_izq
        id op_menor num
        id op_equal ikey llave_izq id op_com num llave_der
        if llave_izq
            id
            print llave_izq text llave_der
        else llave_izq
            print llave_izq text llave_der
        llave_der
    llave_der
par_der
```

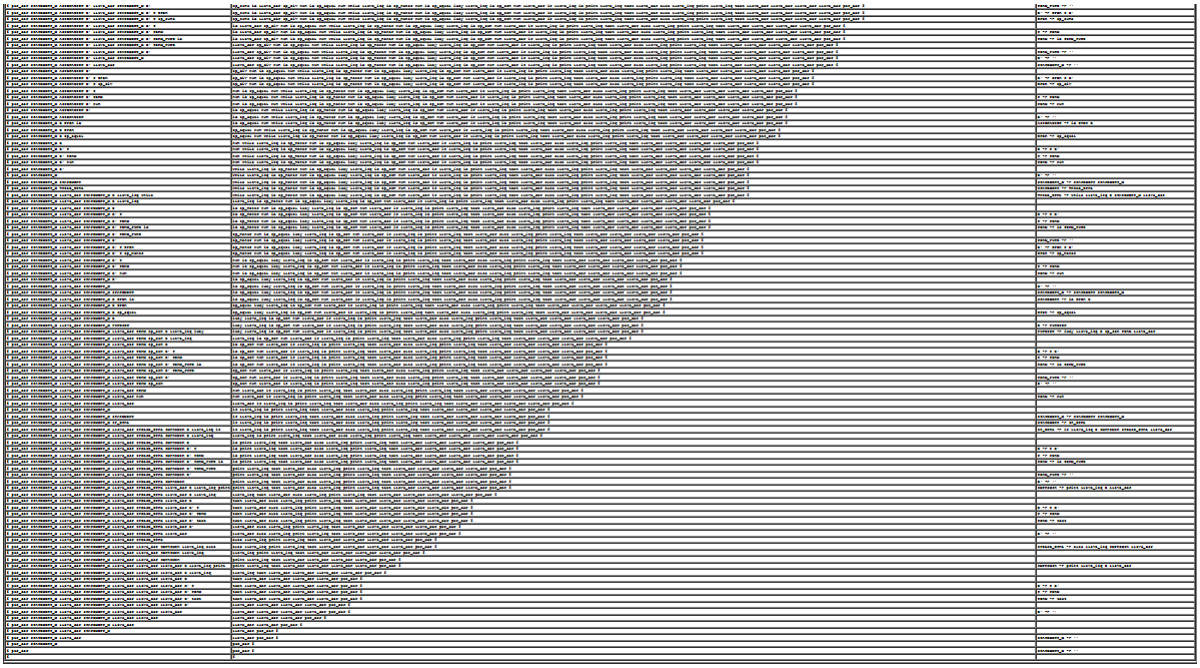

Figura 5: Ejemplo3

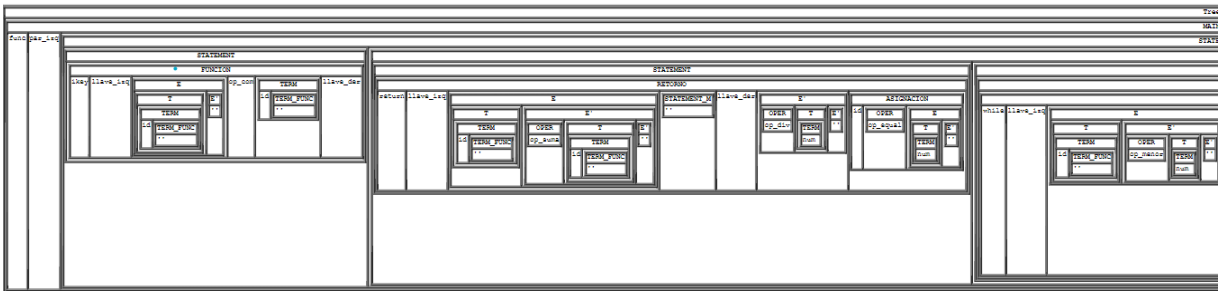

Figura 6: Ejemplo3

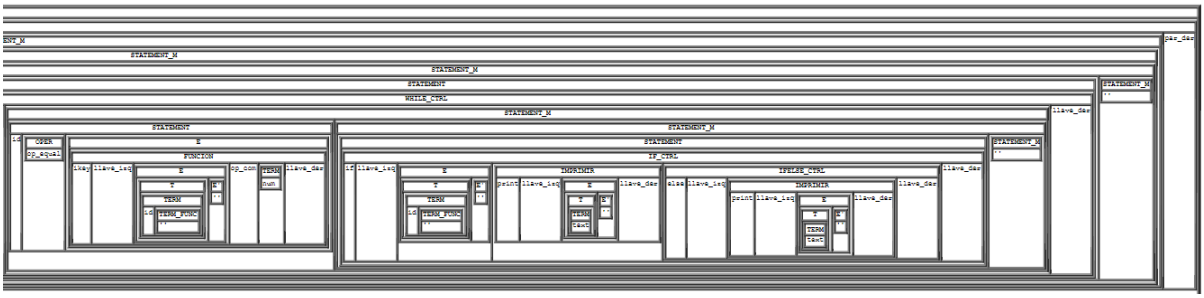

Figura 7: Ejemplo3