MANUAL TÉCNICO

Información del sistema:

- OS: Ubuntu 22.04.2 LTS
- OS Type: 64-bit
- CPU: Intel core i5 2.50 GHz
- GPU: Mesa Intel HD Graphics 4400
- Versión de java: 17.0.8
- RAM: 6 GB
- IDE: IntelliJ idea
- Control de versiones: git 2.34.1
- Github: https://github.com/Hatsune02/LenguajesFormales1.git

Descripción del proyecto:

En esta segunda fase se continua con el desarrollo de parser-py, para esta práctica es necesario diseñar, desarrollar e implementar un analizador sintáctico que pueda pedirle al analizador léxico los tokens reconocidos, este analizador debe ser capaz de reconocer si el código fuente sigue las reglas y estructura sintáctica correcta del lenguaje, recuerde que parser-py está inspirado en python.

Validar que el analizador sintáctico sea capaz de detectar errores sintácticos en el código fuente y proporcionar mensajes de error claros y precisos al usuario.
Implementar mecanismos para recuperarse de errores y continuar el análisis, evitando que un único error provoque la interrupción del proceso completo.

Para esto se ha tenido que definir una gramática, para posteriormente crear un autómata que pueda seguir analizar sintácticamente el texto leído. Para eso necesitaremos una estructura sintáctica la cual se compondrá de:

- Expresiones.
- Declaración y asignación de variables.
- Condicionales (if, elif else).
- Operador Ternario.
- Ciclos (for, for-else, while).
- Funciones/Métodos.
- Bloques de código.

Gramática

G = (N, T, S, P) donde:

N es un conjunto de símbolo no terminales, llamados variables.

T es un conjunto de símbolo terminales, llamados constantes.

S es un símbolo inicial o axioma gramatical, S pertenece a N.

P es un conjunto de reglas de producción.

N (S, A, Asignación, Asignaciones, Asignación', Bloque, OperadorTernario, Condicional, Elif, Else, For, For', While, Función, Parámetros, Parámetro, Parámetro', Bloque, Cuerpo, Cuerpo', Expresiones, Expresiones', CuerpoDiccionario, CuerpoDiccionario', Expresión, ExpresiónFor, Arreglo, Operación, Operación', OperaciónComparación, OperaciónComparación', SumaResta, SumaResta', MultiDiv, MultiDiv', Exponente, Exponente', ExpresiónC, LlamarF, ExpresiónPrimitiva)

T ($, ε, id, fin, (,), asignación, ,, if, else, :, elif, for, not, in, while, def, *, indent, dedent, return, break, [,], {, }, lógicos, and, or, is, comparación, +, -, /, //, %, **, número, cadena, booleano)

Reglas de Producción / Gramática.

S -> Instrucción $

Instrucción -> id **A** fin **Instrucción**

| **Condicional Instrucción**

| **For Instrucción**

| **While Instrucción**

| **Función Instrucción**

| ε

A -> **Asignación** | (**Expresiones**)

Asignación -> **Asignación' Expresión Asignaciones OperadorTernario**

Asignaciones -> asignación **Expresión Asignaciones** | ε

Asignación' ->, id **Asignación' Expresión**, | asignación

OperadorTernario -> if **O** else **Expresión** | ε

Condicional -> if **O**: fin **Bloque Elif Else**

Elif -> elif **O**: fin **Bloque** | ε

Else -> else: fin **Bloque** | ε

For -> for **O For'**

For' -> not in **Arreglo**: fin **Bloque Else**

| in **Arreglo**: fin **Bloque Else**

While -> while **O**: fin **Bloque**

Función -> def id (**Parámetros**): fin **Bloque**

Parámetros -> **Parámetro** | *id | ε

Parámetro -> id **Parámetro'**

Parámetro' ->, id **Parámetro'** | ε

Bloque-> indent **Cuerpo** dedent

Cuerpo -> **Instrucción Cuerpo'**

Cuerpo' -> return **Expresión** fin

| break fin

| ε

Expresiones -> **Expresión Expresiones'** | ε

Expresiones' ->, **Expresión Expresiones'** | ε

CuerpoDiccionario -> cadena: **Expresión CuerpoDiccionario'** | ε

CuerpoDiccionario' ->, cadena: **Expresión CuerpoDiccionario'** | ε

Expresión -> [**Expresiones**] | {**CuerpoDiccionario**} | **O ExpresiónFor**

ExpresiónFor -> in **Arreglo** | not in **Arreglo** | ε

Arreglo -> id **LlamarF** | [**Expresiones**]

Operación -> **OperaciónComparación Operación'**

Operación' -> lógicos **OperaciónComparación Operación'**

| and **OperaciónComparación Operación'**

| or **OperaciónComparación Operación'**

| is **OperaciónComparación Operación'**

| ε

OperaciónComparación -> **SumaResta OperaciónComparación**'

OperaciónComparación' -> comparación **SumaResta OperaciónComparación**' | ε

SumaResta -> **MultiDiv SumaResta'**

SumaResta' -> + **MultiDiv SumaResta'** | - **MultiDiv SumaResta'** | ε

MultiDiv -> **Exponente MultiDiv'**

MultiDiv' -> * **Exponente MultiDiv'**

| / **Exponente MultiDiv'**

| // **Exponente MultiDiv'**

| % **Exponente MultiDiv'**

| ε

Exponente -> **Primero Exponente'**

Exponente' -> ** **Primero Exponente'** | ε

Primero -> not **ExpresiónC** | **ExpresiónC**

ExpresiónC -> (**O**) | **ExpresiónPrimitiva**

LlamarF -> (**Expresiones**) | ε

ExpresiónPrimitiva -> -número | número | cadenas | booleanos | id **LlamarF**

Primeros:

P(S) = {id, $, if, for, while, def}

P(I) = {id, if, for, while, def}

P(A) = {(, ,, asign}

P(Asign) = {,, asign}

P(Asigns) = {asign}

P(Asign') = {,, asign}

P(OT) = {if}

P(Con) = {if}

P(Elif) = {elif}

P(Else) = {else}

P(For) = {for}

P(For’) = {not, in}

P(While) = {while}

P(Fun) = {def}

P(Ps) = {*, id}

P(P) = {id}

P(P’) = {,}

P(B) = {indent}

P(C) = {id, if, for, while, def, return, break}

P(C’) = {return, break}

P(Es) = {[, {, not, (, -, num, string, bool, id}

P(Es’) = {,}

P(CD) = {string}

P(CD’) = {,}

P(E) = {[, {, not, (, -, num, string, bool, id}

P(Ef) = {in, not}

P(Arr) = {id, [}

P(O) = {not, (, -, num, string, bool, id}

P(O’) = {log, and, or, is}

P(Oc) = {not, (, -, num, string, bool, id}

P(Oc’) = {comp}

P(SR) = {not, (, -, num, string, bool, id}

P(SR’) = {+, -}

P(MD) = {not, (, -, num, string, bool, id }

P(MD') = {*, /, //, %}

P(Exp) = {not, (, -, num, string, bool, id }

P(Exp') = {**}

P(First) = {not, (, -, num, string, bool, id}

P(Ec) = {(, -, num, string, bool, id}

P(L) = {(}

P(Ep) = {-, num, string, bool, id}

Siguiente:

S(S) = {}

S(I) = {dedent, return, break, $}

S(A) = {fin}

S(Asign) = {fin}

S(Asigns) = {fin, if}

S(Asign') = {[, {, not, (, -, num, string, bool, id}

S(OT) = {fin}

S(Con) = {id, if, for, while, def, dedent, return, break, $}

S(Elif) = {id, if, for, while, def, else, dedent, return, break, $}

S(Else) = {id, if, for, while, def, dedent, return, break, $}

S(For) = {id, if, for, while, def, dedent, return, break, $}

S(For') = {id, if, for, while, def, dedent, return, break, $}

S(While) = {id, if, for, while, def, dedent, return, break, $}

S(Fun) = {id, if, for, while, def, dedent, return, break, $}

S(Ps) = {)}

S(P) = {)}

S(P') = {)}

S(B) = {id, if, for, while, def, else, elif, dedent, return, break, $}

S(C) = {dedent}

S(C') = {dedent}

S(Es) = {),]}

S(Es') = {),]}

S(CD) = {}}

S(CD') = {}}

S(E) = {fin, if, asign, ,,),], }}

S(Ef) = {fin, if, asign, ,,),], }}

S(Arr) = {:, fin, if, asign, ,,),], } }

S(O) = {else, :, not, in, fin, if, asign, ,,),], } }

S(O') = {else, :, not, in, fin, if, asign, ,,),], } }

S(Oc) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is,], } }

S(Oc') = {else, :, not, in, fin, if, asign, ,,), log, and, or, is,], } }

S(SR) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp,], } }

S(SR') = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp,], } }

S(MD) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -,], } }

S(MD') = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -,], } }

S(Exp) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -, *, /, //, %,], } }

S(Exp') = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -, *, /, //, %,], } }

S(First) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -, *, /, //, %, **,], } }

S(Ec) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -, *, /, //, %, **,], } }

S(L) = {:, fin, if, asign, ,,), else, not, in, log, and, or, is, comp, +, -, *, /, //, %, **,], } }

S(Ep) = {else, :, not, in, fin, if, asign, ,,), log, and, or, is, comp, +, -, *, /, //, %, **,], } }

Tabla:

Tabla de transiciones

Símbolos No Terminales

S	Inicial	Es'	Expresiones’
I	Instrucción	CD	CuerpoDiccionario
A	A	CD'	CuerpoDiccionario’
Asign	Asignación	E	Expresión
Asigns	Asignaciones	Ef	ExpresiónFor
Asign'	Asignación’	Arr	Arreglos
OT	OperadorTernario	O	Operación
Con	Condicional	O'	Operación’
Elif	Elif	Oc	OperaciónComparación
Else	Else	Oc'	OperaciónComparación’
For	For	SR	SumaResta
For'	For’	SR'	SumaResta’
While	While	MD	MultiDiv
Fun	Función	MD'	MultiDiv’
Ps	Parámetros	Exp	Exponente
P	Parámetro	Exp'	Exponente’
P'	Parámetro’	First	Primero
B	Bloque	Ec	ExpresionC
C	Cuerpo	L	LlamarF
C'	Cuerpo’	Ep	ExpresiónPrimitiva
Es	Expresiones		

Símbolos Terminales

$	Inicial	break	break
id	identificador	[	[
fin	fin	]	]
(	(	{	{
)	)	}	}
asign	asignación	log	lógicos
,	,	and	and
if	if	or	or
else	else	is	is
:	:	comp	comparación
elif	elif	+	+
for	for	-	-
not	not	/	/
in	in	//	//
while	while	%	%
def	def	**	**
*	*	num	número
indent	indent	string	cadena
dedent	dedent	bool	booleano
return	return		