Universidad Centroamericana “José Simeón Cañas”

Teoría de Lenguajes de Computación

DOCUMENTO DE PROYECTO FINAL

Catedrático:

Jaime Roberto Clímaco Navarrete

Integrantes:

Douglas Antonio Hernandez Torres	00004516
Karla Beatriz Morales Alfaro	00022516
Erika Stephanie Ramírez Mirón	00157516
Carmen María Solano García	00029613

Antiguo Cuscatlán, 26 de noviembre del 2020

Introducción

Un parser es una pieza fundamental en el proceso de compilación de un programa. El presente proyecto pretende desarrollar un analizador lexicográfico y sintáctico para evaluar la correcta escritura de un subset del lenguaje de programación C.

En este se han contemplado definición, declaración y uso de variables, definición de funciones,, tipos de datos (int, float, char), operaciones aritméticas y lógicas, operaciones de bits, instrucciones condicionales (if - else) y de iteración (while), directivas de inclusión y definición de macros, manejo de comentarios.

Para ello se ha hecho uso de la librería PLY para python con sus implementaciones particulares de Lex y Yacc. Con la ayuda de estas herramientas se ha establecido un conjunto de tokens para el Lex y una gramática para Yaac (Luego se detallará cada producción del lenguaje).

Además, como requisito del proyecto se construyó un analizador sintáctico para la estructura del bucle while, creando a partir de la gramática utilizada en Yaac, otra cumpliera con los requisitos para una parser descendente LL(1). Se utilizaron procesos como factorización, eliminación de recursividad por la izquierda, análisis del First Y Follow de cada producción, etc.

De igual manera, en el directorio del código, se encuentran dos archivos:

- Uno con la gramática para la estructura
- El otro con la evaluación de First, Follow y la tabla de parseo

Guía de uso

Para hacer uso de esta pieza de software, se necesita tener instalado python en su versión 3.9, y poseer una copia del código fuente. Este puede encontrarse en el siguiente repositorio de github: https://github.com/DouglasHdezT/C_Tokenizer_PLY

Una vez se cumplan estos requisitos, solo hace falta correr ambos programas. El método de entrada es a partir de un archivo en el root con el código a evaluar..

Para ejecutar el parser del subset completo de C, se debe ejecutar el archivo main.py:

```
python3.9 main.js
```

A continuación, el prompt de la consola pedirá ingresar el nombre del archivo, procediendo a evaluar sintácticamente el contenido del mismo

Para el parser manual de la estructura while, se debe ejecutar el archivo main_pata.py. El método de ingreso es el mismo, con un archivo, solo que para este caso se debe tener en cuenta que el símbolo final del archivo debe de ser un $.

```
python3.9 main_pata.js
```

De igual forma, el prompt de la consola pedirá ingresar el nombre del archivo.

Gramática para estructura While

A continuación se muestra la gramática utilizada para la estructura While, en la cual los números representan un correlativo utilizado en la tabla de parseo.

while_stmt	01 : WHILE LPAREN condition RPAREN statement
statement	02 : expression_stmt 03 | compound_stmt
compound_stmt	04 : BLOCK_START local_instructions BLOCK_END
local_instructions	05 : local_instructions'
local_instructions'	06 : local_instruction local_instructions' 07 | empty
local_instruction	08 : types var_declaration EOI 09 | statement
expression_stmt	10 : expression EOI 11 | EOI
expression	12 : expression' simple_expression
expression'	13 : ID ASSIGN 14 | empty
var_declaration	15 : var_definition var_declaration'
var_declaration'	16 : COMMA var_definition var_declaration' 17 | empty
var_definition	18 : ID var_definition'
var_definition'	19 : ASSIGN simple_expression 20 | empty
types	21 : INT 22 | FLOAT 23 | CHAR
condition	24 | condition_member condition'
condition'	25 | LOGIC_OP condition_member condition'

	26 | empty
condition_member	27 : NEGATION simple_expression 28 | simple_expression
simple_expression	29 : additive_operation simple_expression'
simple_expression'	30 : RELATION_OP additive_operation simple_expression' 31 | empty
additive_operation	32 : prod_operation additive_operation'
additive_operation'	33 : ARITMETIC_OP_ADD prod_operation additive_operation' 34 | empty
prod_operation	35 : factor prod_operation'
prod_operation'	36 : ARITMETIC_OP_PROD factor prod_operation' 37 | empty
factor	38 : LPAREN simple_expression RPAREN 39 | ID 40 | NUMBER 41 | CHARACTER

Tabla de parseo:

NT/T	WHILE	LPAREN	RPAREN	BLOCK_START	BLOCK_END	EOI	ID	ASSIGN	COMMA	INT	FLOAT	CHAR	NEGATION	RELATION_OP	ARITMETIC_OP_ADD	ARITMETIC_OP_PROD	LOGIC_OP	NUMBER	CHARACTER	$
while_stmt	01																			
statement		02		03		02	02											02	02	
compound_stmt				04																
local_instructions					05															05
local_instructions_p		06		06	07	06	06			06	06	06						06	06	07
local_instruction		09		09		09	09			08	08	08						09	09	
expression_stmt		10				11	10											10	10	
expression		12					12											12	12	
expression_p		14					13											14	14	14
var_declaration							15													
var_declaration_p						17			16											17
var_definition							18													
var_definition_p						20		19	20											20
types										21	22	23								
condition		24					24						24					24	24	
condition_p			26														25			26
condition_member		28					28						27					28	28	
simple_expression		29					29											29	29	
simple_expression_p			31			31			31					30			31			31
additive_operation		32					32											32	32	
additive_operation_p			34			34			34					34	33		34			34
prod_operation		35					35											35	35	
prod_operation_p			37			37			37					37	37	36	37			37
factor		38					39											40	41	

Gramática del subset completo

Cuerpo principal

p_program
program ::= include_directives declaration_l

p_empty
empty :: =

Directivas de inclusión y declaración

p_include_directives
include_directives ::= include_directives include_directive
| include_directive
| empty

p_include_directive
include_directive ::= HASH INCLUDE STRING
| HASH DEFINE ID simple_expression

p_declaration_l
declaration_l ::= declaration_l declaration
| declaration

p_declaration
declaration ::= type var_declaration EOI
| fun_declaration

p_var_declaration
var_declaration ::= var_declaration COMMA var_definition
| var_definition

p_var_definition
var_definition ::= ID
| ID ASSIGN simple_expression

p_fun_declaration
fun_declaration ::= type ID LPAREN params RPAREN compound_stmt
| VOID ID LPAREN params RPAREN compound_stmt

p_params
params ::= params_l

| VOID
| empty

p_params_l

params_l ::= params_l COMMA param
| param

p_param

param ::= type ID

p_type

type ::= INT
| FLOAT
| CHAR

Expresiones y bloques de código

p_expression

expression ::= ID ASSIGN simple_expression
| simple_expression

p_expression_stmt

expression_stmt ::= expression EOI
| EOI

p_compound_stmt

compound_stmt ::= BLOCK_START local_instructions BLOCK_END

p_local_instructions

local_instructions ::= local_instructions type var_declaration EOI
| local_instructions statement
| empty

p_statement

statement ::= expression_stmt
| compound_stmt
| if_stmt
| while_stmt
| return_stmt

p_return_stmt

return_stmt ::= RETURN EOI
| RETURN expression EOI

p_if_stmt
if_stmt ::= IF LPAREN condition RPAREN statement
| IF LPAREN condition RPAREN statement ELSE statement

p_while_stmt
while_stmt ::= WHILE LPAREN condition RPAREN statement

Condiciones

p_condition
condition ::= NEGATION simple_expression
| simple_expression LOGIC_OP simple_expression
| simple_expression

Expresiones

p_simple_expression
simple_expression ::= simple_expression RELATION_OP bit_operation
| bit_operation

p_bit_operation
bit_operation ::= bit_operation BIT_OP additive_operation
| additive_operation

p_additive_operation
additive_operation ::= additive_operation ARITMETIC_OP_ADD prod_operation
| prod_operation

p_prod_operation
prod_operation ::= prod_operation ARITMETIC_OP_PROD factor
| factor

p_factor
factor ::= LPAREN simple_expression RPAREN
| call
| ID
| NUMBER
| CHARACTER

Llamadas a funciones

p_call
call ::= ID LPAREN args RPAREN

p_args

```
args : args_l
        | empty
```

p_args_l

```
args_l : args_l COMMA simple_expression
         | simple_expression
```