Universidad del Valle de Guatemala
Diseño de Lenguajes de Programación
Luis Diego Fernández
16344

Documentación – Proyecto 3

I. Introducción

Este documento demuestra el proceso de creación de un parser y scanner utilizando archivos de cocol y el proyecto generado durante el semestre, todo bajo las especificaciones requeridas para la clase de Diseño de Lenguajes de Programación. Este documento se dividide en cuatro partes:

- Código, estructura y ejecución del proyecto.
- Documentación de corrida con archivo de prueba Aritmetica.ATG.
- Documentación de corrida con archivo de prueba DobleAritmetica.ATG.
- Documentación de corrida con archivo de prueba LittleCoCol.ATG

II. Código, estructura y ejecución del proyecto

Código

El repositorio del proyecto se encuentra en GitHub bajo el repositorio:

https://github.com/LuisDiego19FV/Proyecto_Compiladores

Librerías

Para correr el programa se recomienda utilizar Python 3.7 o superior, y se requiere tener las siguientes librerías instaladas:

- PySimpleGUI (Para utilizar la GUI de la aplicación)
- PySimpleAutomata (Para imprimir el DFA en un archivo)

Ejecución

Para correr el archivo para la producción del parser y scanner únicamente se necesita correr:

```
python .\my_cocor.py
```

Esto desplegará un explorador de archivos que permita seleccionar el ATG a utilizar para la creación del compilador.

Para correr el parser y scanner luego de que se abran generado únicamente se necesita correr:

```
python .\run_scanner.py
```

Esto desplegará un explorador de archivos que permita seleccionar el de pruebas que el parser estará leyendo.

III. Pruebas con Aritmetica.ATG

El parser y scanner generados, además del archivo cocol utilizado para esta prueba se han copiado en la carpeta del proyecto “Documentación /Aritmetica” (Además en la entrega del proyecto se ha entrega una carpeta con solo la documentación). Aquí también se encuentra un archivo utilizado para hacer la prueba del parser y scanner generados, el cual es mismo que se estará enseñando a continuación.

Screenshot del parser y scanner generado:

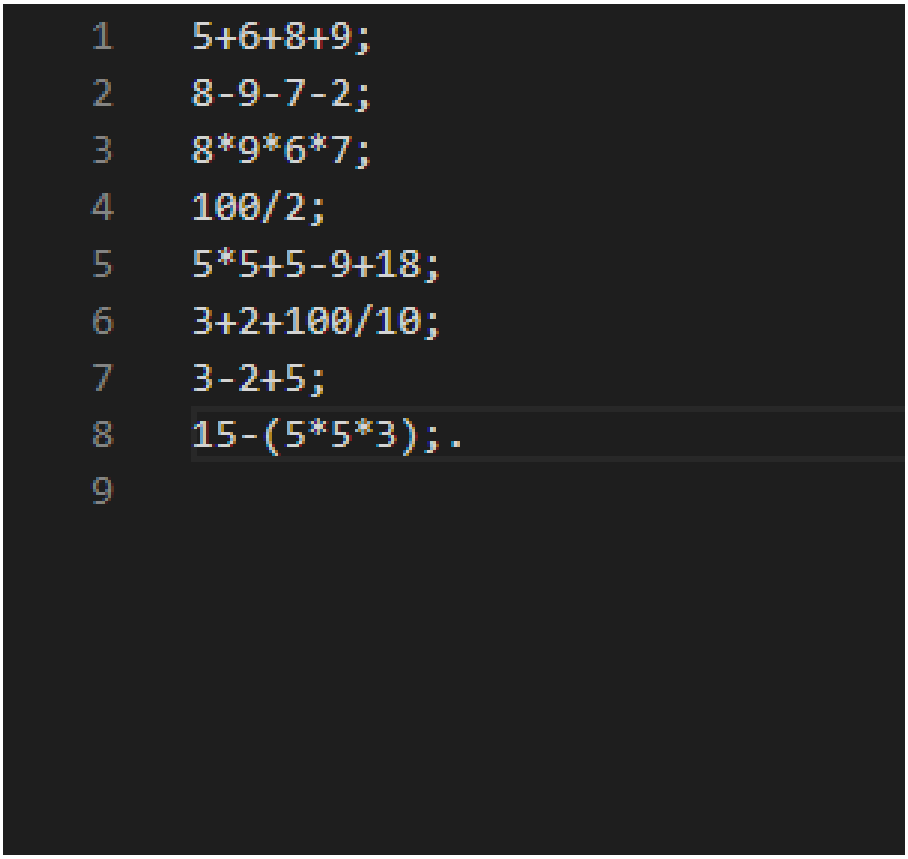
Parser

```
import Lexer.lexer as lex
import Lexer.automataPrinter as autPrint
from my_scanner import Scanner as scanner

class Parser():

    def __init__(self, filename):
        self.filename = filename
        self.sc = scanner(filename)

        self.t = self.sc.scan()
        self.la = self.t

        self._EOF = 0

        self._white = 1
        self._number = 2
        self._pt7 = 3
        self._pt6 = 4
        self._pt5 = 5
        self._pt4 = 6
        self._pt3 = 7
        self._pt2 = 8
        self._pt1 = 9
        self._pt0 = 10
        self._switch = 11
        self._do = 12
        self._while = 13
        self._ident = 14
        self.maxT = 15

        self.Aritmetica()


    def Aritmetica(self):
        while(self.la.get_tok_type() == self._pt3 or self.la.ge
            self.Stat()
            self.Expect(self._pt0)
            while(self.la.get_tok_type() == self._white):
                self.Expect(self._white)
        self.Expect(self._pt1)
```

Scanner

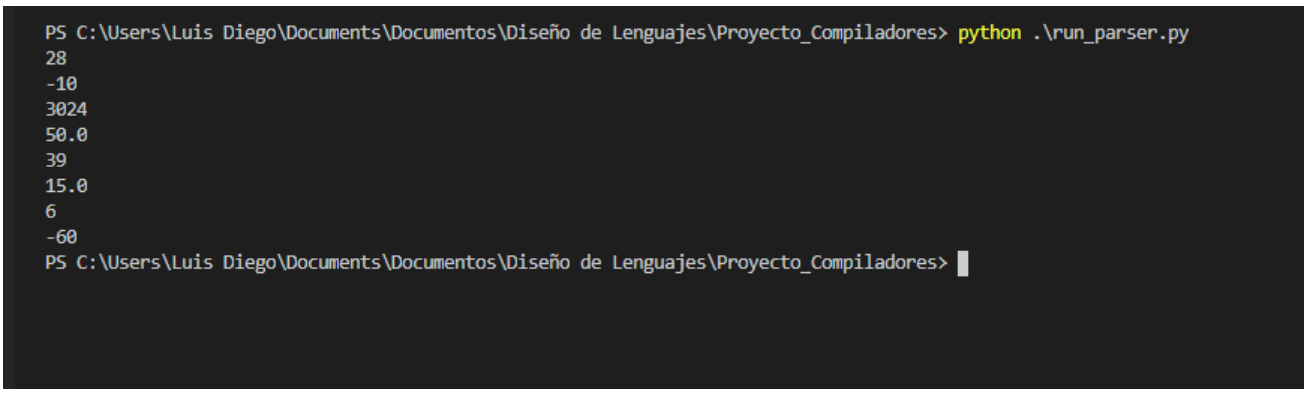
```
import Lexer.lexer as lex
from Lexer.nodes import Token as cls_token

class Scanner():

    def __init__(self, filename):

        # COMPILER VARIABLES
        self.tag = ""
        self.filename = filename
        self.pointer = 0
        self.buffer = ""
        self.end_buffer = False
        self.ignore_chars = ""

        # ADD FILE TO BUFFER
        file = open(filename)

        for i in file:
            self.buffer += i

        # AUTOMATA VARIABLES
        self.separateDFAs = []
        self.mainDFA = None

        self.process_tokens()

    def process_tokens(self):

        # AUTOMATAS TOKENS

        nodes0 = lex.regexToDFA("(\r|\n|\t| )((\r|\n|\t| ))*","white
        self.separateDFAs.append(nodes0)

        nodes1 = lex.regexToDFA("(0|1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|
        self.separateDFAs.append(nodes1)

        nodes2 = lex.regexToDFA(")","pt7",3)
        self.separateDFAs.append(nodes2)

        nodes3 = lex.regexToDFA("(","pt6",4)
        self.separateDFAs.append(nodes3)

        nodes4 = lex.regexToDFA("/","pt5",5)
        self.separateDFAs.append(nodes4)

        nodes5 = lex.regexToDFA("*","pt4",6)
        self.separateDFAs.append(nodes5)
```

Archivo de prueba utilizado (prueba test_aritmetica.txt):



```
1   5+6+8+9;
2   8-9-7-2;
3   8*9*6*7;
4   100/2;
5   5*5+5-9+18;
6   3+2+100/10;
7   3-2+5;
8   15-(5*5*3);.
9
```

Resultado de corrida:



```
PS C:\Users\Luis Diego\Documents\Documentos\Diseño de Lenguajes\Proyecto_Compiladores> python .\run_parser.py
28
-10
3024
50.0
39
15.0
6
-60
PS C:\Users\Luis Diego\Documents\Documentos\Diseño de Lenguajes\Proyecto_Compiladores>
```

IV. Pruebas con DobleAritmetica.ATG

El parser y scanner generados, además del archivo cocol utilizado para esta prueba se han copiado en la carpeta del proyecto “Documentación /DobleAritmetica” (Además en la entrega del proyecto se ha entrega una carpeta con solo la documentación). Aquí también se encuentra un archivo utilizado para hacer la prueba del parser y scanner generados, el cual es mismo que se estará enseñando a continuación.

Screenshot del parser y scanner generado:

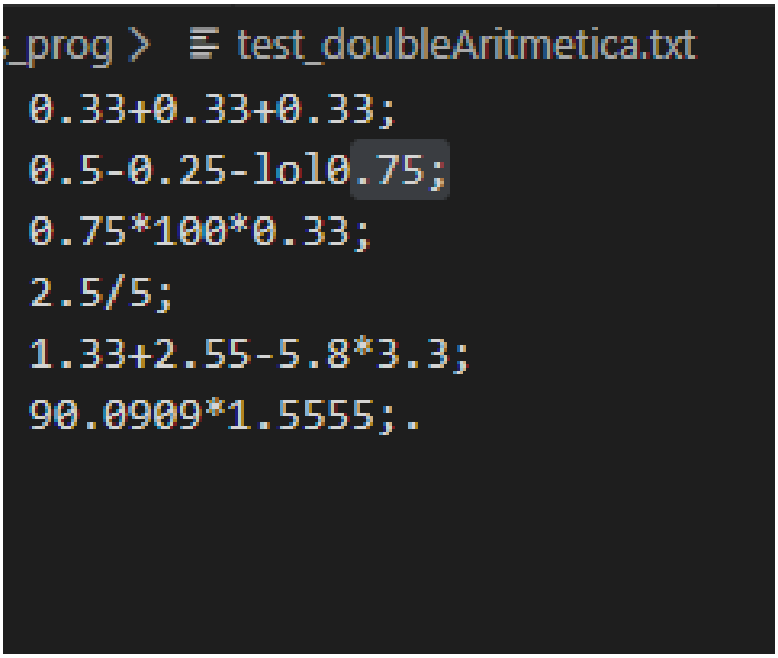
Parser

```
import Lexer.lexer as lex
import Lexer.automataPrinter as autPrint
from my_scanner import Scanner as scanner

class Parser():

    def __init__(self, filename):
        self.filename = filename
        self.sc = scanner(filename)

        self.t = self.sc.scan()
        self.la = self.t

        self._EOF = 0

        self._white = 1
        self._decnumber = 2
        self._number = 3
        self._pt7 = 4
        self._pt6 = 5
        self._pt5 = 6
        self._pt4 = 7
        self._pt3 = 8
        self._pt2 = 9
        self._pt1 = 10
        self._pt0 = 11
        self._do = 12
        self._while = 13
        self.maxT = 14

        self.Double()


    def Double(self):
        while(self.la.get_tok_type() == self._pt3 or self.la.
            self.Stat()
            self.Expect(self._pt0)
            while(self.la.get_tok_type() == self._white):
                self.Expect(self._white)
        while(self.la.get_tok_type() == self._white):
            self.Expect(self._white)
        self.Expect(self._pt1)
```

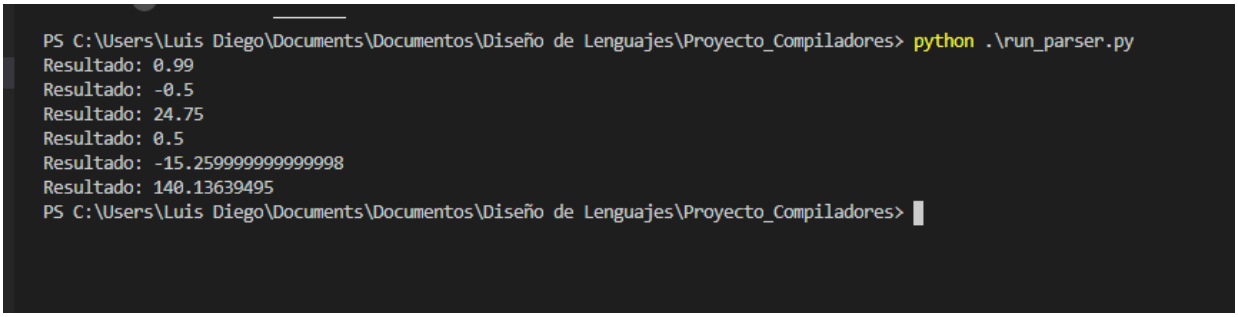
Scanner

```
import Lexer.lexer as lex
from Lexer.nodes import Token as cls_token

class Scanner():

    def __init__(self, filename):

        # COMPILER VARIABLES
        self.tag = ""
        self.filename = filename
        self.pointer = 0
        self.buffer = ""
        self.end_buffer = False
        self.ignore_chars = ""

        # ADD FILE TO BUFFER
        file = open(filename)

        for i in file:
            self.buffer += i

        # AUTOMATA VARIABLES
        self.separateDFAs = []
        self.mainDFA = None

        self.process_tokens()

    def process_tokens(self):

        # AUTOMATAS TOKENS

        nodes0 = lex.regexToDFA("(\n|\r|\t)((\n|\r|\t))*","white",1)
        self.separateDFAs.append(nodes0)

        nodes1 = lex.regexToDFA("(0|1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|
        self.separateDFAs.append(nodes1)

        nodes2 = lex.regexToDFA("(0|1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|
        self.separateDFAs.append(nodes2)

        nodes3 = lex.regexToDFA(")","pt7",4)
        self.separateDFAs.append(nodes3)

        nodes4 = lex.regexToDFA("(","pt6",5)
        self.separateDFAs.append(nodes4)
```

Archivo de prueba utilizado (prueba test_doubleAritmetica.txt):



Resultado de corrida:



V. Pruebas con LittleCoCoL.ATG

El parser y scanner generados, además del archivo cocol utilizado para esta prueba se han copiado en la carpeta del proyecto “Documentación / LittleCoCoL” (Además en la entrega del proyecto se ha entrega una carpeta con solo la documentación). Aquí también se encuentra un archivo utilizado para hacer la prueba del parser y scanner generados, el cual es mismo que se estará enseñando a continuación.

Screenshot del parser y scanner generado:

Parser

my_parser.py > Parser > SymbolProd

```python
import Lexer.lexer as lex
import Lexer.automataPrinter as autPrint
from my_scanner import Scanner as scanner

class Parser():

    def __init__(self, filename):
        self.filename = filename
        self.sc = scanner(filename)

        self.t = self.sc.scan()
        self.la = self.t

        self._EOF = 0

        self._endarg = 1
        self._startarg = 2
        self._endcode = 3
        self._startcode = 4
        self._charnumber = 5
        self._string = 6
        self._char = 7
        self._pt18 = 8
        self._pt17 = 9
        self._pt16 = 10
        self._pt15 = 11
        self._pt14 = 12
        self._pt13 = 13
        self._pt12 = 14
        self._pt11 = 15
        self._pt10 = 16
        self._pt9 = 17
        self._pt8 = 18
        self._pt7 = 19
        self._pt6 = 20
        self._pt5 = 21
        self._pt4 = 22
        self._pt3 = 23
        self._pt2 = 24
        self._pt1 = 25
        self._pt0 = 26
        self._ident = 27
        self.maxT = 28

        self.MyCOCOR()
```

Scanner

my_scanner.py > Scanner > process_tokens

```python
import Lexer.lexer as lex
from Lexer.nodes import Token as cls_token

class Scanner():

    def __init__(self, filename):

        # COMPILER VARIABLES
        self.tag = ""
        self.filename = filename
        self.pointer = 0
        self.buffer = ""
        self.end_buffer = False
        self.ignore_chars = ""

        # ADD FILE TO BUFFER
        file = open(filename)

        for i in file:
            self.buffer += i

        # AUTOMATA VARIABLES
        self.separateDFAs = []
        self.mainDFA = None

        self.process_tokens()

    def process_tokens(self):

        # AUTOMATAS TOKENS

        nodes0 = lex.regexToDFA("(>)","endarg",1)
        self.separateDFAs.append(nodes0)

        nodes1 = lex.regexToDFA("(<)","startarg",2)
        self.separateDFAs.append(nodes1)

        nodes2 = lex.regexToDFA("(.))","endcode",3)
        self.separateDFAs.append(nodes2)

        nodes3 = lex.regexToDFA("((.)","startcode",4)
        self.separateDFAs.append(nodes3)

        nodes4 = lex.regexToDFA("(CHR)(0|1|2|3|4|5|6|7|8|9)((0
        self.separateDFAs.append(nodes4)

        nodes5 = lex.regexToDFA("(\")(A|B|C|D|E|F|G|H|I|J|K|L|
```

Archivo de prueba utilizado (prueba prueba_cocol.ATG):

```
COMPILER NumberOrWord

CHARACTERS

letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".
digit = "0123456789".

KEYWORDS

while = "while".
do = "do".
switch = "switch".

TOKENS

ident = letter{letter|digit} EXCEPT KEYWORDS.
number = digit{digit}.
white = whitespace{whitespace}.

PRODUCTIONS

Suma= {Stat ";"}.

Stat =  (.value = 0.) Expression.

Expression =  (Number | Word).

Number =  number (.print(self.t.get_val()).) .

Word=  ident (.print(self.t.get_val()).) .

END NumberOrWord
```

Resultado de corrida:

```
PS C:\Users\Luis Diego\Documents\Documentos\Diseño de Lenguajes\Proyecto_Compiladores> python .\run_parser.py
Nombre Inicial del Compilador: NumberOrWord
LEYENDO CHARACTERS
Char Set 1: letter
Char Set 2: digit
LEYENDO KEYWORDS
KeyWord 1: while
KeyWord 2: do
KeyWord 3: switch
LEYENDO TOKENS
Token 1: ident
Identificador en Token:  letter
Identificador en Token:  letter
Identificador en Token:  digit
Token 2: number
Identificador en Token:  digit
Identificador en Token:  digit
Token 3: white
Identificador en Token:  whitespace
Identificador en Token:  whitespace
LEYENDO PRODUCTIONS
Production 1: Suma
Identificador en Production:  Stat
Production 2: Stat
Identificador en Production:  Expression
Production 3: Expression
Identificador en Production:  Number
Identificador en Production:  Word
Production 4: Number
Identificador en Production:  number
Production 5: Word
Identificador en Production:  ident
Nombre Final del Compilador: NumberOrWord
PS C:\Users\Luis Diego\Documents\Documentos\Diseño de Lenguajes\Proyecto_Compiladores>
```

VI. Reproducir las pruebas

Para reproducir los resultados vistos en este documento únicamente es necesario tomar el parser y scanner en la sección de Documentación, remplezar los que se tienen en el root del proyecto por estos y correr *run_parser.py* y buscar el archivo de prueba que se quiere utilizar. Para lograr esto de manera correcta será necesario instalar las librerías indicadas en el principio del documento.

Si se quiere volver a rehacer los scanners y parsers vistos en esta documentación se pueden utilizar los archivos ATG que se tienen en la carpeta Pruebas_ATG y seguir las instrucciones al principio del documento para correr correctamente el proyecto.