Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Organización de Lenguajes y Compiladores 2

Junio 2020

Juan Pablo Osuna de Leon

201503911

Manual de Tecnico Augus

Requerimientos mínimos:

- Python 3.8
- Windows 10

Explicación de gramáticas:

Gramáticas Ascendente:

```
S = A
A = main : SENTENCIAS
```

S produce A

A produce ‘main’ ‘:’ SENTENCIAS

SENTENCIAS produce una lista de instrucciones

SENTENCIAS produce SENTENCIAS SENTENCIA

| SENTENCIA

SENTENCIA produce DECLARACIONES

|INSTRUCCIONES

|ETIQUETA

```
SENTENCIAS = SENTENCIAS SENTENCIA
           | SENTENCIA

SENTENCIA = DECLARACIONES
          |INSTRUCCIONES
          |ETIQUETA
```

DECLARACIONES produce ID ARRAY_

ARRAY_ produce CORCHETES '=' EXPRESION

| '=' EXPRESION

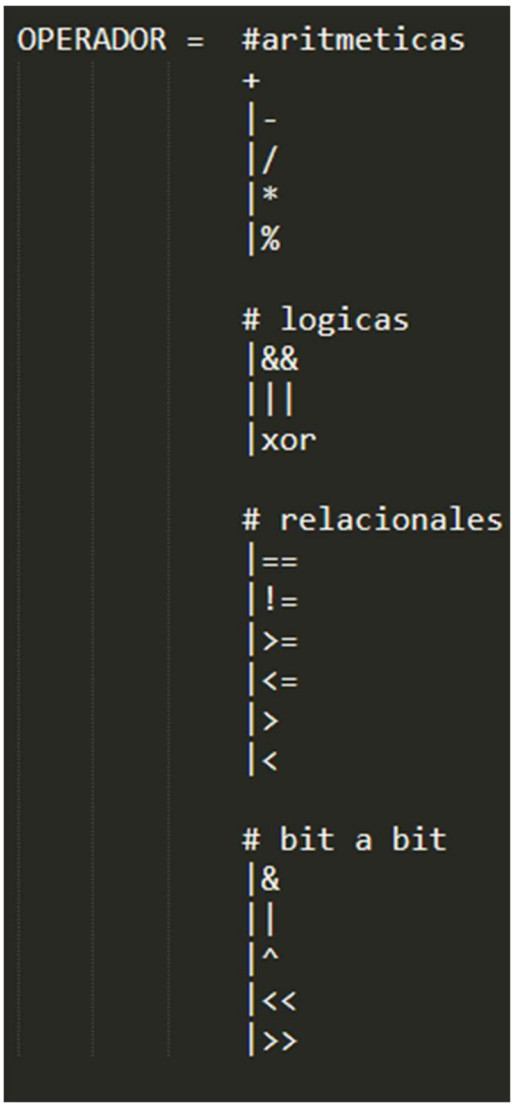
```
DECLARACIONES = ID ARRAY

ARRAY = CORCHETES igual EXPRESION ;
      | igual EXPRESION ;
```

EXPRESION = OPERACION

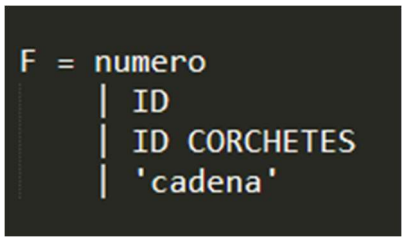
|ATOMICO

|FUNCION

OPERACIÓN PRODUCE:



ATOMICO: produce F
F produce:



Gramtoca completa:

```
S = A
A = main : SENTENCIAS

SENTENCIAS = SENTENCIAS SENTENCIA
           | SENTENCIA

SENTENCIA = DECLARACIONES
          |INSTRUCCIONES
          |ETIQUETA

ETIQUETA = LABEL :

INSTRUCCIONES = print ( EXPRESION ) ;
              | if ( EXPRESION ) goto cadena ;
              | unset( ID );
              | exit;
              | goto cadena ;

DECLARACIONES = ID ARRAY

ARRAY = CORCHETES igual EXPRESION ;
      | igual EXPRESION ;


CORCHETES = CORCHETES CORCHETE
          |CORCHETE

CORCHETE = [ F ]

EXPRESION = OPERACION
          |ATOMICO
          |FUNCION


OPERACION = F OPERADOR F            #expresion
          | -F                      #menos f
          | !F                      #notlogica f
          | ~F                      #notbit f
          | &F                      #andbit f
```

```
ATOMICO = F

FUNCION =  abs( EXPRESION )          #valor absoluto
           | ( TIPO ) ID ;           #casteo
           | read();
           | array();                #para casos como print($t1[0]);


TIPO =  int
        | float
        | char

F = numero
    | ID
    | ID CORCHETES
    | 'cadena'

ID = $ letra numero

OPERADOR =  #aritmeticas
            +
            |-
            |/
            |*
            |%

            # logicas
            |&&
            |||
            |xor

            # relacionales
            |==
            |!=
            |>=
            |<=
            |>
            |<

            # bit a bit
            |&
            ||
            |^
            |<<
```

es, 45 characters selected

Gramática descendente:

Básicamente es la misma gramática solo cambian las recursividades por la izquierda

Lista de sentencias:

```
SENTENCIAS = SENTENCIA SENTENCIAS'

SENTENCIAS' = SENTENCIA SENTENCIAS'
            | e
```

Lista de corchetes:

```
CORCHETES = CORCHETE CORCHETES'

CONSTCHETES' = CONRCHETE CORCHETES'
             |e
```

Clase Expresiones:

En esta clase se almacenarán todas las expresiones.

Por ejemplo, las expresiones aritméticas, y todas las variantes que tiene dicha expresión.

```python
class NumericExpression:
    ''' this class represent an numeric expresion'''

class BinaryExpression(NumericExpression):
        def __init__(self, op1, op2, operator, line, column):
            self.op1 = op1
            self.op2 = op2
            self.operator = operator
            self.line = line
            self.column = column

class Number(NumericExpression):
        def __init__(self, line, column, val=0):
            self.val = val
            self.line = line
            self.column = column

class Identifier(NumericExpression):
        def __init__(self, id, line, column):
            self.id = id
            self.line = line
            self.column = column
```

La clase de instrucciones servirá para crear objetos de tipo instrucción y poder hacer una abstracción de estas en el análisis

```python
class Instruction:
    '''this is an abstractab class'''

class Print_(Instruction) :
    '''print statment, recieve a string'''

    def __init__(self, cadena, line, column):
        self.cadena = cadena
        self.line = line
        self.column = column

class Declaration(Instruction):
    '''variables declarations'''

    def __init__(self, id, line, column, val = 0):
        self.id = id
        self.val = val
        self.line = line
        self.column = column

class Unset(Instruction):
    '''variables destruction'''

    def __init__(self, id, line, column):
        self.id = id
        self.line = line
        self.column = column
```

Instancia de una declaración

```python
def p_declaraciones(t):
    'DECLARACIONES  : ID ARRAY_'
    #insertion a new variable
    t[0] = Declaration(t[1], t.lineno(1), t.lexpos(1),t[2])
```

Ejecución de instrucciones:

```
def process(instructions, ts, printList, textEdit):
    global currentAmbit, pasadas, currentParams, contador

    try:
        i = 0
        while i < len(instructions):
            #isinstance verificar tipos
            b = instructions[i]
            if isinstance(b, Print_):
            elif isinstance(b, Declaration):
            elif isinstance(b, If):
            elif isinstance(b, Goto):
            elif isinstance(b, Label):
            elif isinstance(b, Exit):
            elif isinstance(b, Unset):

            i += 1
    except:
```