MANUAL TÉCNICO

201602782 - SERGIO FERNANDO OTZOY GONZALEZ

PRIMER PROYECTO DE LABORATORIO

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

CONTENIDO

OBJETIVOS

GENERALES

1. Ilustrar las gramáticas utilizadas en el intérprete
2. Ilustrar las estructuras utilizadas por el intérprete

ESPECÍFICOS

1. Mostrar la gramática utilizada para el analizador LALR(1)
2. Mostrar la gramática utilizada para simular el analizador LL(1)
3. Dar una breve explicación de las estructuras de datos utilizadas por el intérprete

ALCANCE

Este manual técnico está escrito para dar a conocer la estructura interna del interprete. Es decir, las producciones de las gramáticas utilizadas, así como las estructuras de datos utilizadas por el intérprete para lograr su objetivo, generar un resultado dado un texto de entrada.

La aplicación está escrita en Python 3.8. Se utilizaron las librerías de PyQt5, graphviz y ply.

PANORAMA GENERAL DE LA APLICACIÓN

La aplicación es un IDE el cual cuenta con el lenguaje Augus. El lenguaje Augus es una *combinación* de lenguajes de programación PHP y el ensamblador MIPS. El uso de esta aplicación y su lenguaje es con meros fines didácticos, es decir, para demostrar el uso de las gramáticas LALR(1) y LL(1) en ply de Python, así como el aprovecho del análisis dirigido por la sintaxis.

REQUISITOS TÉCNICOS

Para el correcto funcionamiento y ejecución de la aplicación es necesaria la instalación de Python 3.8, ***pip*** el sistema de gestión de paquetes de Python. Una vez instalados, es necesario instalar las librerías de las cuales depende directamente la aplicación:

- PyQt5
- graphviz
- ply

El programa fue exitosamente ejecutado en un computador con las siguientes especificaciones:

- Windows ® 10 x64
- 8Gb RAM
- Intel ® Core™ i7-8550U

ANALIZADOR LALR (ASCENDENTE)

La gramática utilizada para este analizador es la siguiente:

Grammar

```
INIT →LIST
     | .
LIST →LIST IST
     | IST.
IST →  IDT assign OPR scolon
     | goto lbs scolon
     | if parizq OPR parder goto lbs scolon
     | lbs colon
     | NTV scolon.
NTV →  unset parizq IDT parder
     | print parizq OPN parder
     | exit.
IDT →  VRN DML
     | VRN.
VRN →  tvar
     | avar
     | vvar
     | rvar
     | svar
     | spvar.
DML →  DML DMN
     | DMN.
DMN →  corizq int_val corder
     | corizq string_val corder
     | corizq VRN corder.
```

```
OPR → OPN plus OPN
    | OPN minus OPN
    | OPN times OPN
    | OPN quotient OPN
    | OPN remainder OPN
    | OPN and OPN
    | OPN xor OPN
    | OPN or OPN
    | OPN andbw OPN
    | OPN orbw OPN
    | OPN xorbw OPN
    | OPN shl OPN
    | OPN shr OPN
    | OPN eq OPN
    | OPN neq OPN
    | OPN gr OPN
    | OPN gre OPN
    | OPN ls OPN
    | OPN lse OPN
    | read parizq parder
    | array parizq parder
    | minus OPN
    | not OPN
    | notbw OPN
    | andbw idt
    | abs parizq OPN parder
    | parizq int parder IDT
    | parizq float parder IDT
    | parizq char parder IDT.
OPN → float_val
    | int_val
    | string_val
    | IDT.
```

ANALIZADOR LL (DESCENDENTE)

Grammar

```
INIT →    LIST
        | .
LIST →    IST LIST1 .
LIST1 →   IST LIST1
        | .
IST →     IDT assign OPR scolon
        | goto lbs scolon
        | if parizq OPR parder goto lbs scolon
        | lbs colon
        | NTV scolon .
NTV →     unset parizq IDT parder
        | print parizq OPN parder
        | exit .
IDT →     VRN FVRN .
FVRN →    DML
        | .
VRN →     tvar
        | avar
        | vvar
        | rvar
        | svar
        | spvar .
DML →     DMN DML1 .
DML1 →    DMN DML1
        | .
DMN →     corizq Fcorizq .
Fcorizq → int_val corder
        | string_val corder
        | VRN corder .
```

```
OPR →     OPN FOPN
        | read parizq parder
        | array parizq parder
        | minus OPN
        | not OPN
        | notbw OPN
        | andbw idt
        | abs parizq OPN parder
        | parizq Fparizq .
FOPN →    plus OPN
        | minus OPN
        | times OPN
        | quotient OPN
        | remainder OPN
        | and OPN
        | xor OPN
        | or OPN
        | andbw OPN
        | orbw OPN
        | xorbw OPN
        | shl OPN
        | shr OPN
        | eq OPN
        | neq OPN
        | gr OPN
        | gre OPN
        | ls OPN
        | lse OPN .
Fparizq →int parder IDT
        | float parder IDT
        | char parder IDT .
OPN →     float_val
        | int_val
        | string_val
        | IDT .
```

ESTRUCTURAS UTILIZADAS

CLASES DE INSTRUCCIÓN

```
class Print(Instruction):
    '''
        This class saves a value that is then used to print a message
    '''
    def __init__(self, oper, row):
        self.oper = oper
        self.row = row
```

```
class Unset(Instruction):
    '''
        This class saves a variable name that must be deleted
    '''
    def __init__(self, varn, row):
        self.varn = varn
        self.row = row
```

```
class Exit(Instruction):
    '''
        This class represents the end a procedure
    '''
```

```
class If(Instruction):
    '''
        This class make decisions upon a boolean given value
    '''
    def __init__(self, oper, name, row):
        self.oper = oper
        self.name = name
        self.row = row
```

```
class GoTo(Instruction):
    '''
        To make a jump to a given label
    '''
    def __init__(self, name, row):
        self.name = name
        self.row = row
```

```
class Assignment(Instruction):
    '''
        To set a value to a given variable
    '''
    def __init__(self, varName, varType, valExp, row):
        '''varName: name of the variable
           varType: type of the variable
           valExpe: saves the access for an array'''
        self.varName = varName
        self.varType = varType
        self.valExp = valExp
        self.row = row
```

```
class Label(Instruction):
    '''
        Sets a label name that is then used to make a jump
        inside the code
    '''
    def __init__(self, name, row):
        self.name = name
        self.row = row
```

CLASES DE OPERACIÓN

```
class OperationExpression(Expression):
    '''
    '''
    def __init__(self, op, e1 = None, e2 = None, row = -1):
        self.e1 = e1
        self.e2 = e2
        self.op = op
        self.row = row
```

```
class ValExpression(Expression):

    '''
    '''
    def __init__(self, value, type, row = -1):
        self.value = value
        self.type = type
        self.row = row
```

CLASES ENUM

```
class Operator(Enum):
    '''All operators'''
    PLUS = 1        # +
    MINUS = 2       # -
    TIMES = 3       # *
    QUOTIENT = 4    # /
    REMAINDER = 5   # %
    NEGATIVE = 6    # -
    ABS = 7         # abs
    NOT = 8         # !
    AND = 9         # &&
    XOR = 10        # xor
    OR = 11         # ||
    NOTBW = 12      # ~
    ANDBW = 13      # &
    ORBW = 14       # |
    XORBW = 15      # ^
    SHL = 16        # <<
    SHR = 17        # >>
    EQ = 18         # =
    NEQ = 19        # !=
    GR = 20         # >
    GRE = 21        # >=
    LS = 22         # <
    LSE = 23        # <=
    AMP = 24        # &
    CINT = 25       # int
    CFLOAT = 26     # float
    CCHAR = 27      # string
    READ = 28       # read()
    ARRAY = 29      # array()
```

```
class ValType(Enum):
    '''Supported types'''
    CHAR = 1
    STRING = 2
    FLOAT = 3
    INTEGER = 4
    POINTER = 5
    ARRAY = 6
    STRUCT = 7
    REFVAR = 8
```