1-6-2021

Gramática

Proyecto JPR

Eduardo Ixen
INGENIERIA USAC

Contenido

Expresiones regulares

Decimales

Expresión formada por uno o más dígitos seguidos de punto seguidos de uno o más dígitos.

```
def t_DECIMAL(t):
    r'\d+\.\d+'
    try:
        t.value = float(t.value)
    except ValueError:
        print("Float value too large %d", t.value)
        t.value = 0
    return t
```

Enteros

Expresión formada por uno o más dígitos.

```
def t_ENTERO(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print("Integer value too large %d", t.value)
        t.value = 0
    return t
```

Cadenas

Expresión que comienza con comillas dobles seguido del carácter especial comilla doble o cualquier carácter una o muchas veces y finaliza con comillas dobles.

```
def t_CADENA(t):
    r'\"([^\\\"]|\\.)*\"'
    t.value = t.value[1:-1] # remuevo las comillas
    return t
```

Carácter

Expresión que comienza con comilla simple seguido de cualquier carácter especial con \ o cualquier carácter, una sola vez y termina con comilla simple.

```
def t_CARACTER(t):
    r'(\'([^\\\"]|\\.)\')'
    t.value = t.value[1:-1]
    return t
```

Identificador

Cualquier carácter de a-z seguido de a-z, _ o números del 0 al 9 cero o muchas veces.

```
def t_ID(t):
    r'[a-zA-Z][a-zA-Z_0-9]*'
    t.type = reservadas.get(t.value.lower(),'ID')
    return t
```

Comentario multilínea o simple

Comentario multilínea: Comienza con numeral seguido de asterisco seguido de cualquier carácter o salto de línea cero o muchas veces y termina con asterisco numeral.

Comentario simple: Comienza con numeral seguido de cualquier carácter y termina en salto de línea.

```
def t_COMENTARIO_MULTI(t):
    r'\#\*(.|\n)*\*\#'
    t.lexer.lineno += t.value.count("\n")

# Comentario simple // ...
def t_COMENTARIO_SIMPLE(t):
    r'\#.*\n'
    t.lexer.lineno += 1
```

Producción inicial

Consta de la lista de instrucciones que pueden venir dentro del lenguaje, utilizando recursividad para admitir una o varias instrucciones.

```
def p_init(t) :
    'init            : instrucciones'

    t[0] = t[1]

def p_instrucciones_instrucciones_instruccion(t) :
    'instrucciones    : instrucciones instruccion'
    if t[2] != "":
        t[1].append(t[2])
    t[0] = t[1]

#////////////////////////////////////INSTRUCCIONES////////

def p_instrucciones_instruccion(t) :
    'instrucciones    : instruccion'
    if t[1] == "":
        t[0] = []
    else:
        t[0] = [t[1]]
```

Instrucciones

Es una lista de instrucciones que el lenguaje acepta.

```
def p_instruccion(t) :
    '''instruccion      : imprimir_instr finins
                        | declaracion_instr finins
                        | asignacion_instr finins
                        | if_instr
                        | while_instr
                        | break_instr finins
                        | main_instr
                        | incremento_instr finins
                        | decremento_instr finins
                        | for_instr
                        | switch_instr
                        | funcion_instr
                        | llamada_instr finins
                        | return_instr finins
                        | continue_instr finins
                        | read_inst_exp finins
                        | declArr_instr finins
                        | modlArr_instr finins
    '''
```

Final de instrucción

Producción que permite finalizar o no cada instrucción con un punto y coma.

```
def p_finins(t) :
    '''finins       : PUNTOCOMA
                    |'''
```

Instrucción imprimir

Producción que comienza con la palabra reservada “prima” seguida de paréntesis abierto, seguido de una expresión y finaliza con paréntesis cerrado.

```
def p_imprimir(t) :
    'imprimir_instr     : RPRINT PARA expresion PARC'
```

Declaración de variables

Producción que puede comenzar con la palabra reservada “Var” seguida de un identificador seguida del signo igual y finaliza con una expresión o puede ser únicamente la palabra reservada “var” seguida de un identificador.

```
def p_declaracion(t):
    '''declaracion_instr : RVAR ID IGUAL expresion
                         | RVAR ID
    '''
```

Declaración de arreglos

Producción que puede comenzar con un tipo de dato seguido de una lista de dimensiones seguido de un identificador seguido del signo igual seguido de la palabra “new” seguido de una lista de

expresiones o puede ser de un tipo de dato seguido de una lista de dimensiones seguido de un identificador seguido del signo igual seguido de un identificador.

```
def p_tipo1(t):
    'tipo1 : tipo lista_Dim ID IGUAL RNEW tipo lista_expresiones'
    t[0] = DeclaracionArr1(t[1], t[2], t[3], t[6], t[7], t.lineno(3), find_column(input, t.slice[3]))

def p_tipo1Id(t):
    'tipo1 : tipo lista_Dim ID IGUAL ID'
    t[0] = DeclaracionReferencia(t[1], t[2], t[3], t[5], t.lineno(3), find_column(input, t.slice[3]))
```

IF

La instrucción if comienza con la palabra if seguida de una expresión entre paréntesis seguida de instrucciones entre llaves y luego de eso puede seguir la palabra reservada “else” seguida de una lista de instrucciones entre llaves o la palabra reservada else pude estar seguida de otra producción de if.

```
def p_if1(t) :
    'if_instr     : RIF PARA expresion PARC LLAVEA instrucciones LLAVEC'
    t[0] = If(t[3], t[6], None, None, t.lineno(1), find_column(input, t.slice[1]))

def p_if2(t) :
    'if_instr     : RIF PARA expresion PARC LLAVEA instrucciones LLAVEC RELSE LLAVEA instrucciones LLAVEC'
    t[0] = If(t[3], t[6], t[10], None, t.lineno(1), find_column(input, t.slice[1]))

def p_if3(t) :
    'if_instr     : RIF PARA expresion PARC LLAVEA instrucciones LLAVEC RELSE if_instr'
    t[0] = If(t[3], t[6], None, t[9], t.lineno(1), find_column(input, t.slice[1]))
```

Switch

La producción comienza con la palabra reservada switch, seguida de una expresión entre paréntesis seguida de una lista de case seguida de la instrucción default.

También pude ser la palabra reservada switch, seguida de una expresión entre paréntesis seguida de una lista de case.

También puede ser la palabra reservada switch, seguida de la instrucción default.

```
def p_switch1(t):
    'switch_instr   : RSWITCH PARA expresion PARC LLAVEA case_list default LLAVEC'
    t[0] = Switch(t[3], t[6], t[7], t.lineno(1), find_column(input, t.slice[1]))

def p_switch2(t):
    'switch_instr   : RSWITCH PARA expresion PARC LLAVEA case_list LLAVEC'
    t[0] = Switch(t[3], t[6], None, t.lineno(1), find_column(input, t.slice[1]))

def p_switch3(t):
    'switch_instr   : RSWITCH PARA expresion PARC LLAVEA default LLAVEC'
    t[0] = Switch(t[3], None, t[6], t.lineno(1), find_column(input, t.slice[1]))
```

While

La producción comienza con la palabra reservada while seguida de una expresión entre paréntesis seguida de una lista de instrucciones entre llaves.

```
def p_while(t):
    '''while_instr : RWHILE PARA expresion PARC LLAVEA instrucciones LLAVEC
    '''
    t[0] = While(t[3], t[6], t.lineno(1), find_column(input, t.slice[1]))
```

For

La producción comienza con la palabra reservada for seguida de una declaración o una asignación seguida de punto y coma seguida una expresión seguida de una actualización de variables, todo esto entre paréntesis finalizando con una lista de instrucciones entre llaves.

```
def p_for(t):
    '''
    for_instr : RFOR PARA declasigna PUNTOCOMA expresion PUNTOCOMA actualizacion PARC LLAVEA instrucciones LLAVEC
    '''
```

Main

Producción que comienza con la palabra reservada “main” seguida paréntesis abierto y cerrado seguida de una lista de instrucciones entre llaves.

Esta producción es la principal del programa, es la única que se ejecuta y las demás funciones se ejecutan a partir de las llamadas en el main.

```
def p_main(t) :
    'main_instr     : RMAIN PARA PARC LLAVEA instrucciones LLAVEC'
```

Expresiones

Es una lista de expresiones posibles en el programa, compuesta de operaciones aritméticas, lógicas y relacionales.

Utiliza recursividad para poder realizar múltiples operaciones entre cada expresión.

```
    expresion : expresion MAS expresion
              | expresion MENOS expresion
              | expresion POR expresion
              | expresion DIV expresion
              | expresion POTENCIA expresion
              | expresion MODULO  expresion
              | expresion IGUALIGUAL expresion
              | expresion DIFERENTE expresion
              | expresion MENORQUE expresion
              | expresion MAYORQUE expresion
              | expresion MENORIGUAL expresion
              | expresion MAYORIGUAL expresion
              | expresion OR expresion
              | expresion AND expresion
    '''
```

Primitivos

Son una producción de las expresiones, retornan un valor primitivo como enteros, decimales, cadenas de texto, caracteres o identificadores. Se utilizan para realizar las operaciones entre las expresiones aritméticas, lógicas y relacionales.

```python
def p_expresion_identificador(t):
    '''expresion : ID'''
    t[0] = Identificador(t[1], t.lineno(1), find_column(input, t.slice[1]))

def p_expresion_entero(t):
    '''expresion : ENTERO'''
    t[0] = Primitivos(TIPO.ENTERO,t[1], t.lineno(1), find_column(input, t.slice[1]))

def p_primitivo_decimal(t):
    '''expresion : DECIMAL'''
    t[0] = Primitivos(TIPO.DECIMAL, t[1], t.lineno(1), find_column(input, t.slice[1]))

def p_primitivo_cadena(t):
    '''expresion : CADENA'''
    t[0] = Primitivos(TIPO.CADENA, especiales(str(t[1])), t.lineno(1), find_column(input, t.slice[1]))

def p_primitivo_char(t):
    '''expresion :  CARACTE (constant) CHARACTER: Literal[TIPO.CHARACTER]
    t[0] = Primitivos(TIPO.CHARACTER, especiales(str(t[1])), t.lineno(1), find_column(input, t.slice[1]))

def p_primitivo_true(t):
    '''expresion :  RTRUE'''
    t[0] = Primitivos(TIPO.BOOLEANO, True, t.lineno(1), find_column(input, t.slice[1]))

def p_primitivo_false(t):
    '''expresion :  RFALSE'''
    t[0] = Primitivos(TIPO.BOOLEANO, False, t.lineno(1), find_column(input, t.slice[1]))
```