Universidad LaSalle Arequipa

Facultad de Ingenieria

Escuela Profesional de Ingenieria de Software

Examen Final

Curso:

Compiladores

Semestre:

VI

Integrantes:

Elmerson Ramith Portugal Carpio

2022

Introduccion

La idea del lenguaje es hacer un hibrido de python que tome la sintaxis detallada de java, esto se hizo con el fin mejorar la experiencia de los nuevos programadores en lenguaje fuertemente tipados, ya que al principio para algunos programadores se les hace complicado entender la sintaxis de python.

Analizardor Lexico

```
import ply.lex as lex


reserved = {
    'true': 'true',
    'false': 'false',
    'if': 'if',
    'else': 'else',
    'return': 'return',
    'int': 'int',
    'bool': 'bool',
    'def': 'def'
}
tokens = [
    'id',
    'num',
    'addition',
    'subtract',
    'multiplication',
    'division',
    'equal',
    'mayor',
    'minor',
    'lparem',
    'rparem',
    'ini_llave',
    'fin_llave',
    'dotcomma',
    'comma',
] + list(reserved.values())

# Regular expression rules for simple tokens
t_addition = r'\+'
t_subtract = r'\-'
t_multiplication = r'\*'
t_division = r'\/'
t_equal = r'\='
t_mayor = r'\>'
t_minor = r'\<'
t_lparem = r'\('
t_rparem = r'\)'
t_ini_llave = r'\{'
t_fin_llave = r'\}'
t_dotcomma = r'\;'
t_comma = r'\,'



# A regular expression rule with some action code

def t_num(t):
    r'\d+'
    t.value = int(t.value)  # guardamos el valor del lexema
    #print("se reconocio el numero")
    return t


def t_id(t):
    r'[a-zA-Z]+([a-zA-Z0-9]*)'
    t.type = reserved.get(t.value, 'id')
    return t

```

```
   # Define a rule so we can track line numbers


def t_nuevalinea(t):
    r'\n+'
    t.lexer.lineno += len(t.value)


   # A string containing ignored characters (tabs)
t_ignore = ' \t'

# Error handling rule


def t_error(t):
    print("Caracter Ilegal '%s'" % t.value[0])
    t.lexer.skip(1)


# Build the lexer
lexer = lex.lex()

# Traenis la informacion de un txt


def get_tokens(fp):

    data = fp.read()
    print(data)
    fp.close()

    # Give the lexer some input
    lexer.input(data)
    # Guarda la informacion

    guardar_token = []

    while True:
        tok = lexer.token()
        if not tok:
            break      # No more input
        # print(tok)
        #print(tok.type, tok.value, tok.lineno, tok.lexpos)
        guardar_token.append({'type': tok.type.lower(), 'lexeme': str(
            tok.value).lower(), 'line': tok.lineno})

    guardar_token.append(
        {'type': '$', 'lexeme': '$', 'line': guardar_token[-1]['line']})
    return guardar_token


if __name__ == "__main__":
    fp = open("ProyectoParcial\practica.txt")
    tokens = get_tokens(fp)
    # print(tokens)
}
```

Gramatica

```
# GRAMMAER

PROGRAM -> FUNCTIONS STATEMENTS
FUNCTIONS -> FUNCTION FUNCTIONS
FUNCTIONS -> ''

FUNCTION -> def id lparem PARATYS rparem ini_llave STATEMENTS return EXP dotcomma
    fin_llave
PARATYS -> TYPE id PARATY
PARATYS -> ''
PARATY -> comma TYPE id PARATY
PARATY -> ''

STATEMENTS -> STATEMENT STATEMENTS
STATEMENTS -> ''
STATEMENT -> DECLARATION
STATEMENT -> ASSIGN
STATEMENT -> IF
STATEMENT -> ELSE

DECLARATION -> TYPE id equal EXP dotcomma
TYPE -> int
TYPE -> bool
ASSIGN -> id equal EXP dotcomma

EXP -> T E'
E' -> OPER T E'
E' -> ''
T -> TERM
T -> lparem EXP rparem
TERM -> id INVOCACION
TERM -> num
TERM -> BOOLEAN
BOOLEAN -> true
BOOLEAN -> false
INVOCACION -> lparem PARAMS rparem
INVOCACION -> ''
OPER -> addition
OPER -> subtract
OPER -> division
OPER -> multiplication

PARAMS -> EXP PARAM
PARAMS -> ''
PARAM -> comma EXP PARAM
PARAM -> ''


IF -> if lparem CONDITION rparem ini_llave STATEMENTS fin_llave
ELSE -> if  ini_llave STATEMENTS fin_llave
CONDITION -> EXP OPERCON EXP
OPERCON -> minor
OPERCON -> mayor
```

createType

```
1
2  def identificado(root):
3      # Aqui  se envia el papa  -> root
4      #  root.children -> saca los hijos de papa
5      stack = root.children
6      # creamos un array para  comparar
7      arr = []
8      valor = []
9      signo = []
10
11     while len(stack) > 0:
12
13         if stack[0].symbol.symbol == "OPER":
14             signo.append(stack[0].children[0].lexeme)
15         # En este caso --- buscamos el papa donde se encuentra las variables
16         if stack[0].symbol.symbol == 'TERM':
17             # agregamos e los hijos  al arrray creado
18             arr.append(stack[0].children[0].symbol.symbol)
19             valor.append(stack[0].children[0].lexeme)
20         temp = stack[0].children
21         stack.pop(0)
22
23         # vamos a iterrar sobre los valores para insertalos en el stack
24         for i in temp:
25             stack.insert(0, i)
26     ty = arr[0]
27
28     flag = False
29     for j in arr:
30         if j != ty:
31             flag = True
32             break
33     if flag:
34         return "Error", valor, signo
35     return ty, valor, signo
36
37 def pack(root):
38     stack = root.children
39     valor_id = []
40     while len(stack) > 0:
41         if stack[0].symbol.symbol == 'DECLARATION':
42             valor_id.append(stack[0].children[1].lexeme)
43         temp = stack[0].children
44         stack.pop(0)
45         for i in temp:
46             stack.insert(0, i)
47     inicializarVar(valor_id)
48
49 def pack_if(root):
50     stack = root.children
51     valor_id = []
52     valor_oper = []
53     arr = []
54     while len(stack) > 0:
55         if stack[0].symbol.symbol == 'TERM':
56             valor_id.append(stack[0].children[0].lexeme)
57         if stack[0].symbol.symbol == 'OPERCON':
58             valor_oper.append(stack[0].children[0].lexeme)
59         if stack[0].symbol.symbol == 'id':
60             arr.append(stack[0].lexeme)
61         temp = stack[0].children
62         stack.pop(0)
63         for i in temp:
64             stack.insert(0, i)
65     return valor_id, valor_oper, arr
```

crearAsembly

```
1
2  def inicioD():  # inicio del assembly
3      file = open("ProyectoParcial/variables.txt", "a")
4      file.write(".data\n")
5      file.close()
6
7
8  def inicioT():  # inicio del assembly
9      file = open("ProyectoParcial/variables.txt", "a")
10     file.write(".text\n")
11     file.close()
12
13
14 def convertir(variable, signo, valor):
15     file = open("ProyectoParcial/variables.txt", "a")
16     print(variable.lexeme, signo, valor)
17     valor_res = valor[::-1]
18     pack.append(valor_res)
19     signo_res = signo[::-1]
20     index = len(signo_res)
21     index_signo = 0
22     variable_ar.append(variable)
23
24     if len(signo_res) > 0:
25         for j in range(len(valor_res)):
26             if j == 0:
27                 # Si la primera posicion de Valor_res == variable_ar -_ solo se asigna
28                 for i in range(len(variable_ar)):
29                     if valor_res[j] == variable_ar[i].lexeme:
30                         file.write("\n\nla   $t0," + "   var_" + variable_ar[i].lexeme)
31                         file.write("\nlw  " + "$a0," + "  0($t0)")
32                         break
33                 # si la primera varible de VALOR_RES no es IGUAL a VARIABLE_AR -> se
    Creaera la variable
34                 else:
35                     file.write("\nli $a0,    " + str(valor_res[j]) + "\n")
36             if j >= 1:
37                 for i in range(len(variable_ar)):
38                     if valor_res[j] == variable_ar[i].lexeme:
39                         file.write("\n\nsw  " + "$a0,  " + "0($sp)")
40                         file.write("\naddiu  " + "$sp,  " + "$sp,  " + "-4" + "\n")
41                         file.write("\n\nla $t0, var_"+variable_ar[i].lexeme)
42                         file.write("\nlw  " + "$a0," + "  0($t0)")
43                         break
44
45                 else:
46                     file.write("\n\nsw  " + "$a0,  " + "0($sp)")
47                     file.write("\naddiu  " + "$sp,  " + "$sp,  " + "-4" + "\n")
48                     file.write("\nli $a0,   " + str(valor_res[j]) + "\n")
49
50                 if index_signo < index:
51                     if signo_res[index_signo] == "+":
52                         file.write("\nlw   " + "$t1,   " + "4($sp) \n")
53                         file.write("add  " + "$a0,  "   "$a0,  " + "$t1 \n")
54                         file.write("\naddiu  " + "$sp  " + "$sp  " + "4")
55                         index_signo += 1
56
57         file.write("\n\nla   " + "$t0,  " + "var_"+variable.lexeme)
58         file.write("\nsw  " + "$a0,  " + "0($t0)")
59         file.close()
60
61     else:
62         file.write("\nli $a0,    " + str(valor_res[0]) + "\n")
63         file.write("\nla   $t0," + "   var_"+variable.lexeme)
64         file.write("\nsw  " + "$a0," + "  0($t0)\n")
65         file.close()
```

Repositorio del trabajo

Aqui esta el repositorio Link

Concluciones

Gracias a este proyecto se puedo mejorar la logica en el lenguaje python y poder implementar dicha logica en el Asembly. Este prototipo de lenguaje cumple con los requisitos planteados al inicio del proyecto. Se implemento las siguientes funcionalidades, operaciones matematicas, funciones y condicionales. Se logro fusinar la logica y funcionalidad de python con la estructura de C++, dando por resultado Python Old Time.