

Universidad LaSalle Arequipa



Facultad de Ingenieria

Escuela Profesional de Ingenieria de Software

EXAMEN FINAL

Curso:

Compiladores

Semestre:

VI

Integrantes:

Elmerson 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.

Analizador Lexico

```

1  import ply.lex as lex
2
3
4  reserved = {
5      'true': 'true',
6      'false': 'false',
7      'if': 'if',
8      'else': 'else',
9      'return': 'return',
10     'int': 'int',
11     'bool': 'bool',
12     'def': 'def'
13 }
14 tokens = [
15     'id',
16     'num',
17     'addition',
18     'subtract',
19     'multiplication',
20     'division',
21     'equal',
22     'mayor',
23     'minor',
24     'lparem',
25     'rparem',
26     'ini_llave',
27     'fin_llave',
28     'dotcomma',
29     'comma',
30 ] + list(reserved.values())
31
32 # Regular expression rules for simple tokens
33 t_addition = r'\+'
34 t_subtract = r'\-'
35 t_multiplication = r'\*'
36 t_division = r'\/'
37 t_equal = r'\='
38 t_mayor = r'\>'
39 t_minor = r'\<'
40 t_lparem = r'\('
41 t_rparem = r'\)'
42 t_ini_llave = r'\{'
43 t_fin_llave = r'\}'
44 t_dotcomma = r'\;'
45 t_comma = r','
46
47 # A regular expression rule with some action code
48
49 def t_num(t):
50     r'\d+'
51     t.value = int(t.value) # guardamos el valor del lexema
52     #print("se reconocio el numero")
53     return t
54
55 if __name__ == "__main__":
56     fp = open("ProyectoParcial\practica.txt")
57     tokens = get_tokens(fp)
58     # print(tokens)
59 }

```

Gramatica

```

1 # GRAMMAER
2
3 PROGRAM -> FUNCTIONS STATEMENTS
4 FUNCTIONS -> FUNCTION FUNCTIONS
5 FUNCTIONS -> ''
6
7 FUNCTION -> def id lparem PARATYS rparem ini_llave STATEMENTS return EXP dotcomma
      fin_llave
8 PARATYS -> TYPE id PARATY
9 PARATYS -> ''
10 PARATY -> comma TYPE id PARATY
11 PARATY -> ''
12
13 STATEMENTS -> STATEMENT STATEMENTS
14 STATEMENTS -> ''
15 STATEMENT -> DECLARATION
16 STATEMENT -> ASSIGN
17 STATEMENT -> IF
18 STATEMENT -> ELSE
19
20 DECLARATION -> TYPE id equal EXP dotcomma
21 TYPE -> int
22 TYPE -> bool
23 ASSIGN -> id equal EXP dotcomma
24
25 EXP -> T E'
26 E' -> OPER T E'
27 E' -> ''
28 T -> TERM
29 T -> lparem EXP rparem
30 TERM -> id INVOCACION
31 TERM -> num
32 TERM -> BOOLEAN
33 BOOLEAN -> true
34 BOOLEAN -> false
35 INVOCACION -> lparem PARAMS rparem
36 INVOCACION -> ''
37 OPER -> addition
38 OPER -> subtract
39 OPER -> division
40 OPER -> multiplication
41
42 PARAMS -> EXP PARAM
43 PARAMS -> ''
44 PARAM -> comma EXP PARAM
45 PARAM -> ''
46
47
48 IF -> if lparem CONDITION rparem ini_llave STATEMENTS fin_llave
49 ELSE -> if ini_llave STATEMENTS fin_llave
50 CONDITION -> EXP OPERCON EXP
51 OPERCON -> minor
52 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 array 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 insertarlos 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

```

crearAssembly

```

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    $t0," + "    var_" + variable_ar[i].lexeme)
31                         file.write("\n\nlw    " + "$a0," + "    0($t0)")
32                         break
33                 # si la primera variable de VALOR_RES no es IGUAL a VARIABLE_AR -> se
34                 Creaera la variable
35             else:
36                 file.write("\n\nli $a0,    " + str(valor_res[j]) + "\n")
37             if j >= 1:
38                 for i in range(len(variable_ar)):
39                     if valor_res[j] == variable_ar[i].lexeme:
40                         file.write("\n\nsw    " + "$a0,    " + "0($sp)")
41                         file.write("\n\naddiu   " + "$sp,    " + "$sp,    " + "-4" + "\n")
42                         file.write("\n\nla    $t0, var_" + variable_ar[i].lexeme)
43                         file.write("\n\nlw    " + "$a0," + "    0($t0)")
44                         break
45             else:
46                 file.write("\n\nsw    " + "$a0,    " + "0($sp)")
47                 file.write("\n\naddiu   " + "$sp,    " + "$sp,    " + "-4" + "\n")
48                 file.write("\n\nli $a0,    " + str(valor_res[j]) + "\n")
49
50             if index_signo < index:
51                 if signo_res[index_signo] == "+":
52                     file.write("\n\nlw    " + "$t1,    " + "4($sp) \n")
53                     file.write("\n\nadd    " + "$a0,    " + "$a0,    " + "$t1 \n")
54                     file.write("\n\naddiu   " + "$sp    " + "$sp    " + "4")
55                     index_signo += 1
56
57     file.write("\n\nla    " + "$t0,    " + "var_" + variable.lexeme)
58     file.write("\n\nsw    " + "$a0,    " + "0($t0)")
59     file.close()
60
61 else:
62     file.write("\n\nli $a0,    " + str(valor_res[0]) + "\n")
63     file.write("\n\nla    $t0," + "    var_" + variable.lexeme)
64     file.write("\n\nsw    " + "$a0," + "    0($t0)\n")
65     file.close()

```

Ejemplos

```
1 int x = 3;  
2 if ( x > 20){  
3     x = 10;  
4 }  
5 else{  
6     x = 5;  
7 }  
8 int z = x + 8;
```

Output

Rpta: 13

```
1 int x = 3;  
2 if ( 20 > x){  
3     x = 10;  
4 }  
5 else{  
6     x = 5;  
7 }  
8 int z = x + 8;
```

Output

Rpta: 18

```
1 def mostrar (int f){  
2     return f;  
3 }  
4 int x = mostrar(5);
```

Output

Rpta: 5

```
1 def add (int p){  
2     return p + 7;  
3 }  
4 int x = add(5);
```

Output

Rpta: 12

```
1 int x = 4;  
2 int y = 10;  
3 int z = x - y;
```

Output

Rpta: 6

```
1 int x = 4;  
2 int y = 10;  
3 int z = x + y;
```

Output

Rpta: 14

Repositorio del trabajo

Aqui esta el repositorio Link

Conclusiones

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.