

PES University, RR Campus, Bangalore - 560085

Automata Formal Languages and Logic

UE23CS242A

Mini Project

3rd Semester, Academic Year 2024

Date: 12/11/2024

To describe and write Grammar for the constraints of a programming language

Language : Lisp

Constraints : Variable assignment, if-else conditions and function declaration

Name of team member:	SRN of team member:	Section :
Anirudh Sudhir	PES1UG23CS917	B

Lexer

```
1 from ply import lex
2
3 reserved = {"if": "IF", "defun": "DEFUN", "setq": "SETQ", "t": "T", "nil": "NIL"}
4 tokens = (
5     "LPAREN",
6     "RPAREN",
7     "IDENTIFIER",
8     "NUMBER",
9     "STRING",
10    "EQUALS",
11    "PLUS",
12    "MINUS",
13    "MULTIPLY",
14    "DIVIDE",
15    "GREATER",
16    "LESS",
17    "GREATEREQUAL",
18    "LESSEQUAL",
19 ) + tuple(reserved.values())
20
21 t_LPAREN = r"\("
22 t_RPAREN = r"\)"
23 t_EQUALS = r"="
24 t_PLUS = r"\+"
25 t_MINUS = r"\-"
26 t_MULTIPLY = r"\*"
27 t_DIVIDE = r"/"
28 t_GREATER = r">"
29 t_LESS = r"<"
30 t_GREATEREQUAL = r">="
31 t_LESSEQUAL = r"<="
32
33 t_ignore = " \t\n"
34
35 def t_IDENTIFIER(t):
36     r"[a-zA-Z][a-zA-Z0-9\-\_]*" # identifiers start with a letter and can contain digits and hyphens
37     t.type = reserved.get(t.value.lower(), "IDENTIFIER")
38     return t
39
40 def t_NUMBER(t):
41     r"[d\.\d+]" # accepting integer as well as floating point numbers
42     if "." in t.value:
43         pass
44
45 NORMAL main sem-3 lisp_lex.py { 8 Top 1:1 11:02
```

```

7
6
5 def t_IDENTIFIER(t):
4     r"[a-zA-Z][a-zA-Z0-9~]*" # identifiers start with a letter and can contain digits and hyphens
3     t.type = reserved.get(t.value.lower(), "IDENTIFIER")
2     return t
1
41
1 def t_NUMBER(t):
2     r"[d*\.?d+]" # accepting integer as well as floating point numbers
3     if "." in t.value:
4         t.value = float(t.value)
5     else:
6         t.value = int(t.value)
7     return t
8
9
10 def t_STRING(t):
11     r"\"[^\"]*" # accepting quoted strings, and matching any character which is not a double quote
12     t.value = t.value[1 : len(t.value) - 1] # removing the double quotes
13     return t
14
15
16 def t_error(t):
17     print("Illegal character: ", t.value[0])
18     t.lexer.skip(1)
19
20
21 lexer = lex.lex()

```

NORMAL y main sem-3 > lisp_lex.py

{ < 8 66% 41:1 11:03

Parser

```

lisp_lex.py X lisp_yacc.py X
1 from ply import yacc
2
3 from lisp_lex import tokens
4
5 def p_program(p):
6     """program : form
7     | program form"""
8     if len(p) == 2:
9         p[0] = [p[1]]
10    else:
11        p[0] = p[1] + [p[2]]
12
13 def p_form(p):
14     """form : function_definition
15     | variable_definition
16     | function_call
17     | if_condition
18     | literal
19     | operation"""
20     p[0] = p[1]
21
22
23 def p_operation(p):
24     """operation : LPAREN operator arguments RPAREN"""
25     p[0] = {"type": "operation", "operator": p[2], "arguments": p[3]}
26
27
28 def p_operator(p):
29     """operator : EQUALS
30     | PLUS
31     | MINUS
32     | MULTIPLY
33     | DIVIDE
34     | GREATER
35     | LESS
36     | GREATEREQUAL
37     | LESSEQUAL"""
38     p[0] = p[1]
39
40
41 def p_function_definition(p):

```

NORMAL y main sem-3 > 1 1 lisp_yacc.py

^ [< 8 1% 2:1 11:05

```
lisp_lex.py x lisp_yacc.py x
4 | LESS
3 | GREATEREQUAL
2 | LESSEQUAL"""
1 | p[0] = p[1]
41 |
1 |
2 |
3 |
4 |
5 | def p_function_definition(p):
6 |     """function_definition : LPAREN DEFUN IDENTIFIER parameter_list body RPAREN"""
7 |     p[0] = {
8 |         "type": "function_definition",
9 |         "name": p[3],
10 |         "parameters": p[4],
11 |         "body": p[5],
12 |     }
13 |
14 | def p_parameter_list(p):
15 |     """parameter_list : LPAREN parameters RPAREN"""
16 |     p[0] = p[2]
17 |
18 | def p_parameters(p):
19 |     """parameters :
20 |     | IDENTIFIER parameters"""
21 |     if len(p) == 1:
22 |         p[0] = []
23 |     else:
24 |         p[0] = [p[1]] + p[2]
25 |
26 | def p_body(p):
27 |     """body : form
28 |     | form body"""
29 |     if len(p) == 2:
30 |         p[0] = [p[1]]
31 |     else:
32 |         p[0] = [p[1]] + p[2]
33 |
34 |
35 | def p_variable_definition(p):
36 |     """variable_definition : LPAREN SETQ IDENTIFIER form RPAREN"""
37 |     p[0] = {"type": "variable_definition", "name": p[3], "value": p[4]}
38 |
NORMAL | main | sem-3 | 1 1 | lisp_yacc.py | 26% 41:1 | 11:05
```

```
lisp_lex.py x lisp_yacc.py x
4 | p[0] = [p[1]] + p[2]
3 |
2 |
1 |
77 | def p_variable_definition(p):
78 |     """variable_definition : LPAREN SETQ IDENTIFIER form RPAREN"""
79 |     p[0] = {"type": "variable_definition", "name": p[3], "value": p[4]}
80 |
81 |
82 | def p_function_call(p):
83 |     """function_call : LPAREN IDENTIFIER arguments RPAREN"""
84 |     p[0] = {"type": "function_call", "name": p[2], "arguments": p[3]}
85 |
86 |
87 | def p_arguments(p):
88 |     """arguments :
89 |     | form arguments"""
90 |     if len(p) == 1:
91 |         p[0] = []
92 |     else:
93 |         p[0] = [p[1]] + p[2]
94 |
95 |
96 | def p_if_condition(p):
97 |     """if_condition : LPAREN IF test_form then_form RPAREN
98 |     | LPAREN IF test_form then_form else_form RPAREN"""
99 |     if len(p) == 6:
100 |         p[0] = {"type": "if_condition", "test": p[3], "then": p[4], "else": None}
101 |     else:
102 |         p[0] = {"type": "if_condition", "test": p[3], "then": p[4], "else": p[5]}
103 |
104 |
105 | def p_test_form(p):
106 |     """test_form : form"""
107 |     p[0] = p[1]
108 |
109 |
110 | def p_then_form(p):
111 |     """then_form : form"""
112 |     p[0] = p[1]
113 |
114 |
115 | def p_else_form(p):
116 |     """else_form : form"""
117 |
NORMAL | main | sem-3 | 1 1 | lisp_yacc.py | 50% 77:37 | 11:05
```

```
lisp_lex.py X lisp_yacc.py X
10 | p[0] = p[1]
9 |
8 |
7 | def p_then_form(p):
6 |     "then_form : form"
5 |     p[0] = p[1]
4 |
3 |
2 |
1 | def p_else_form(p):
116 |     "else_form : form"
1 |     p[0] = p[1]
1 |
1 |
1 |
3 | def p_literal(p):
4 |     """literal : NUMBER
5 |     | STRING
6 |     | T
7 |     | NIL
8 |     | IDENTIFIER"""
9 |     if isinstance(p[1], (int, float)):
10 |         p[0] = {"type": "number", "value": p[1]}
11 |     elif p[1] == "t":
12 |         p[0] = {"type": "boolean", "value": True}
13 |     elif p[1] == "nil":
14 |         p[0] = {"type": "boolean", "value": False}
15 |     elif isinstance(p[1], str):
16 |         if p.slice[1].type == "STRING":
17 |             p[0] = {"type": "string", "value": p[1]}
18 |         else:
19 |             p[0] = {"type": "identifier", "value": p[1]}
20 |
21 |
22 | def p_error(p):
23 |     if p:
24 |         print("Syntax error at: ", p.value)
25 |     else:
26 |         print("Syntax error at EOF")
27 |
28 |
29 | parser = yacc.yacc()
30 | while True:
31 |     try:
32 |         s = input("CL > ")
NORMAL | p main | sem-3 | 1 1 | lisp_yacc.py | f p_else_form | 75% 116:15 | 11:05
```

```
lisp_lex.py X lisp_yacc.py X
4 | | IDENTIFIER"""
3 | if isinstance(p[1], (int, float)):
2 |     p[0] = {"type": "number", "value": p[1]}
1 | elif p[1] == "t":
128 |     p[0] = {"type": "boolean", "value": True}
1 | elif p[1] == "nil":
2 |     p[0] = {"type": "boolean", "value": False}
3 | elif isinstance(p[1], str):
4 |     if p.slice[1].type == "STRING":
5 |         p[0] = {"type": "string", "value": p[1]}
6 |     else:
7 |         p[0] = {"type": "identifier", "value": p[1]}
8 |
9 |
10 | def p_error(p):
11 |     if p:
12 |         print("Syntax error at: ", p.value)
13 |     else:
14 |         print("Syntax error at EOF")
15 |
16 |
17 | parser = yacc.yacc()
18 | while True:
19 |     try:
20 |         s = input("CL > ")
21 |     except EOFError:
22 |         break
23 |     if not s:
24 |         continue
25 |     result = parser.parse(s)
26 |     print(result)
NORMAL | p main | sem-3 | 1 1 | lisp_yacc.py | f p_literal | 83% 128:37 | 11:06
```

Output

Variable assignment

- Valid syntax

```
afll [main] X python3 lisp_yacc.py
CL > (setq a 5)
[{'type': 'variable_definition', 'name': 'a', 'value': {'type': 'number', 'value': 5}}]
CL > █
```

- Invalid syntax (error)

```
afll [main] X python3 lisp_yacc.py
CL > setq a 5
Syntax error at: setq
[{'type': 'identifier', 'value': 'a'}, {'type': 'number', 'value': 5}]
CL > (setq a 5
Syntax error at EOF
None
CL >
```

If condition

- Valid syntax

```
afll [main] X python3 lisp_yacc.py
CL > (if (> a 5) "greater than 5" "less than or equal to 5")
[{'type': 'if_condition', 'test': {'type': 'operation', 'operator': '>', 'arguments': [{'type': 'identifier', 'value': 'a'}, {'type': 'number', 'value': 5}]}, 'then': {'type': 'string', 'value': 'greater than 5'}, 'else': {'type': 'string', 'value': 'less than or equal to 5'}}]
CL > █
```

- Invalid syntax (error)

```
afll [main] X python3 lisp_yacc.py
CL > (if (> a 5) "greater than 5" "less than or equal to 5
Illegal character: "
Syntax error at:  than
[{'type': 'identifier', 'value': 'or'}, {'type': 'identifier', 'value': 'equal'}, {'type': 'identifier', 'value': 'to'}, {'type': 'number', 'value': 5}]
CL > (if (> a 5) "greater than 5" "less than or equal to 5"
Syntax error at EOF
None
CL > █
```

Function declaration

- Valid syntax

```
afll [main] X python3 lisp_vacc.py
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5" ) )
[{'type': 'function_definition', 'name': 'foo', 'parameters': ['a'], 'body': [{'type': 'if_condition', 'test': {'type': 'operation', 'operator': '>', 'arguments': [{'type': 'identifier', 'value': 'a'}, {'type': 'number', 'value': 5}]}, 'then': {'type': 'string', 'value': 'greater than 5'}, 'else': {'type': 'string', 'value': 'less than or equal to 5'}}]}]
CL >
```

- Invalid syntax (error)

```
afll [main] X python3 lisp_vacc.py
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to
Illegal character: "
Syntax error at:  than
[{'type': 'identifier', 'value': 'or'}, {'type': 'identifier', 'value': 'equal'}, {'type': 'identifier', 'value': 'to'}]
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5" )
Syntax error at EOF
None
CL > defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5" )
Syntax error at: defun
Syntax error at: )
None
CL > █
```