**Retro Basic Project Report**

Present To

Prabhas Chongstitvatana

Submitted By

Paphawit Boonyapongnukul

(5931036021)

This project is parted of Programming Languages Principles (2110316)

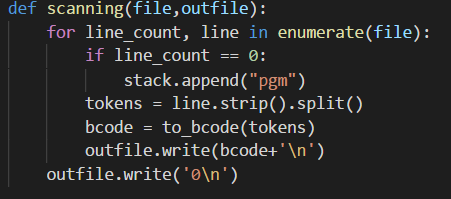
Section 33 Semester 1 Year 2018

Chulalongkorn University

Retro Basic (implemented in Python)

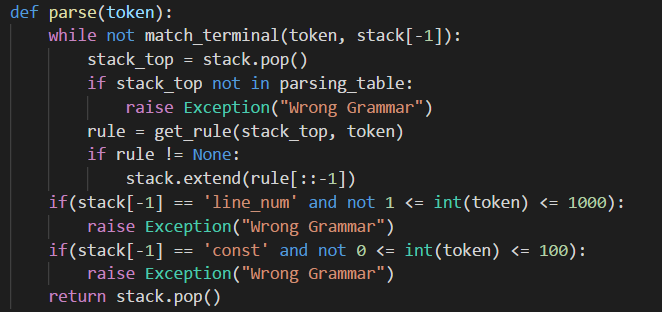
**Part I: Scanner**

Tokenize streaming of character from each line of file by using spilt method in python and get list of tokens considering “Blank Space” as delimiter.



**Part II: Parser**

Put each token into stack and pop for check rule by parsing table if it’s not available raise an exception but if it does push all entry of parsing into stack and loop till terminal (reference algorithm form lecture slide).



|  |  |
| --- | --- |
| pgm := line pgm | EOF line := line\_num stmt stmt := asgmnt | if | print | goto | stop asgmnt := id = exp exp := term exp’exp’ := + term | - term | λ | term := id | const if := IF cond line\_num cond := term cond’cond’ := < term | = term print := PRINT id goto := GOTO line\_num stop := STOP |

**Grammar**

**First set & Follow set**

|  |  |  |
| --- | --- | --- |
| **Non-terminal** | **First** | **Follow** |
| pgm | line\_num, EOF | $ |
| line | line\_num | line\_num, EOF |
| stmt | id, IF, PRINT, GOTO, STOP | line\_num, EOF |
| asgmnt | id | line\_num, EOF |
| exp | id, const | line\_num, EOF |
| exp’ | +, -, λ | line\_num, EOF |
| term | id, const | +, -, line\_num, EOF |
| if | IF | line\_num, EOF |
| cond | id, const | line\_num |
| cond’ | <, = | line\_num |
| print | PRINT | line\_num, EOF |
| goto | GOTO | line\_num, EOF |
| stop | STOP | line\_num, EOF |

**Split Grammar**

1. pgm := line pgm
2. pgm := EOF
3. line := line\_num stmt
4. stmt := asgmnt
5. stmt := if
6. stmt := print
7. stmt := goto
8. stmt := stop
9. asgmnt := id = exp
10. exp := term exp’
11. exp’ := + term
12. exp’ := - term
13. exp’ := λ
14. term := id
15. term := const
16. if := IF cond line\_num
17. cond := term cond’
18. cond’ := < term
19. cond’ := = term
20. print := PRINT id
21. goto := GOTO line\_num
22. stop := STOP

**Parsing Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | line\_num | id | IF | PRINT | GOTO | STOP | + | - | const | < | = | EOF | $ |
| pgm | 1 |  |  |  |  |  |  |  |  |  |  | 2 |  |
| line | 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| stmt |  | 4 | 5 | 6 | 7 | 8 |  |  |  |  |  |  |  |
| asgmnt |  | 9 |  |  |  |  |  |  |  |  |  |  |  |
| exp |  | 10 |  |  |  |  |  |  | 10 |  |  |  |  |
| exp’ | 13 |  |  |  |  |  | 11 | 12 |  |  |  | 13 |  |
| term |  | 14 |  |  |  |  |  |  | 15 |  |  |  |  |
| if |  |  | 16 |  |  |  |  |  |  |  |  |  |  |
| cond |  |  |  |  |  |  |  |  | 17 |  |  |  |  |
| cond’ |  |  |  |  |  |  |  |  |  | 18 | 19 |  |  |
| print |  |  |  | 20 |  |  |  |  |  |  |  |  |  |
| goto |  |  |  |  | 21 |  |  |  |  |  |  |  |  |
| stop |  |  |  |  |  | 22 |  |  |  |  |  |  |  |

**Code**

import sys

parsing\_table = {

"pgm": {"line\_num": ["line", "pgm"], "EOF": ["EOF"]},

"line": {"line\_num": ["line\_num", "stmt"]},

"stmt": {"id": ["asgmnt"], "IF": ["if"], "PRINT": ["print"], "GOTO": ["goto"], "STOP": ["stop"]},

"asgmnt": {"id": ["id", "=", "exp"]},

"exp": {"id": ["term", "exp'"], "const": ["term", "exp'"]},

"exp'": {"EOF": None, "line\_num": None, "+": ["+", "term"], "-": ["-", "term"]},

"term": {"id": ["id"], "const": ["const"]},

"if": {"IF": ["IF", "cond", "line\_num"]},

"cond": {"id": ["term", "cond'"], "const": ["term", "cond'"]},

"cond'": {"<": ["<", "term"], "=": ["=", "term"]},

"print": {"PRINT": ["PRINT", "id"]},

"goto": {"GOTO": ["GOTO", "line\_num"]},

"stop": {"STOP": ["STOP"]},

}

bcode\_type = {

"#line": 10,

"#id": 11,

"#const": 12,

"#if": 13,

"#goto": 14,

"#print": 15,

"#stop": 16,

"#op": 17,

}

id = set([chr(e) for e in range(ord('A'), ord('Z')+1)])

line\_num = set([str(i) for i in range(1, 1001)])

const = set([str(i) for i in range(0, 101)])

terminal = set(["+", "-", "IF", "<", "=", "PRINT", "GOTO", "STOP", "EOF"])

stack = ["EOF"]

def get\_terminal\_type(token):

if token.isdigit():

return "num"

if token in id:

return "id"

if token in terminal:

return token

raise Exception('Wrong Input Grammar')

def match\_terminal(token, top\_stack):

terminal\_type = get\_terminal\_type(token)

if terminal\_type != "num":

return terminal\_type == top\_stack

else:

return top\_stack == "line\_num" or top\_stack == "const"

def get\_rule(stack\_top, token):

terminal\_type = get\_terminal\_type(token)

if terminal\_type != "num" and terminal\_type in parsing\_table[stack\_top]:

return parsing\_table[stack\_top][terminal\_type]

if "line\_num" in parsing\_table[stack\_top]:

return parsing\_table[stack\_top]["line\_num"]

if "const" in parsing\_table[stack\_top]:

return parsing\_table[stack\_top]["const"]

raise Exception("Wrong Grammar")

def parse(token):

while not match\_terminal(token, stack[-1]):

stack\_top = stack.pop()

if stack\_top not in parsing\_table:

raise Exception("Wrong Grammar")

rule = get\_rule(stack\_top, token)

if rule != None:

stack.extend(rule[::-1])

if(stack[-1] == 'line\_num' and not 1 <= int(token) <= 1000):

raise Exception("Wrong Grammar")

if(stack[-1] == 'const' and not 0 <= int(token) <= 100):

raise Exception("Wrong Grammar")

return stack.pop()

def get\_bcode(terminal\_symbol, value):

if(terminal\_symbol == "line\_num"):

return ("#line", int(value))

if(terminal\_symbol == "id"):

return ("#id", ord(value) - ord('A') + 1)

if(terminal\_symbol == "const"):

return ("#const", int(value))

if(terminal\_symbol == "IF"):

return ("#if", 0)

if(terminal\_symbol == "GOTO"):

return ("#goto", int(value))

if(terminal\_symbol == "PRINT"):

return ("#print", 0)

if(terminal\_symbol == "STOP"):

return ("#stop", 0)

if(terminal\_symbol == "+"):

return ("#op", 1)

if(terminal\_symbol == "-"):

return ("#op", 2)

if(terminal\_symbol == "<"):

return ("#op", 3)

if(terminal\_symbol == "="):

return ("#op", 4)

def gen\_bcode(parsed\_list):

bcode\_list = []

for i in range(len(parsed\_list)):

if(parsed\_list[i][0] not in ["GOTO", "line\_num"] or i == 0):

bcode\_list.append(get\_bcode(parsed\_list[i][0], parsed\_list[i][1]))

else:

if(parsed\_list[i][0] == 'line\_num' and i != 0):

bcode\_list.append(get\_bcode("GOTO", parsed\_list[i][1]))

return bcode\_list

def to\_bcode(tokens):

parsed\_list = []

for token in tokens:

parsed\_list.append((parse(token), token))

bcode\_list = gen\_bcode(parsed\_list)

bcode\_string = ''

for types, value in bcode\_list:

bcode\_string = bcode\_string + \

str(bcode\_type[types]) + ' ' + str(value) + ' '

return bcode\_string.strip()

def scanning(file,outfile):

for line\_count, line in enumerate(file):

if line\_count == 0:

stack.append("pgm")

tokens = line.strip().split()

bcode = to\_bcode(tokens)

outfile.write(bcode+'\n')

outfile.write('0\n')

file\_input = str(sys.argv[1])

file = open(file\_input, 'r')

file\_output = open('result\_of\_'+file\_input+'.txt', 'w')

scanning(file,file\_output)

file.close()

file\_output.close()

**Repository of code and document**



<https://github.com/chinyyyyyyyy/RetroBasic2018>