Retro Basic Complier

1. Scanner

The input can be scanned by using function open(). The Retro Basic program is keep in the input file and each line is strip() and split() into a token and stored as list() of tokens as shown in python code below.

```
path = str(sys.argv[1])
fil = open(path,'r')
out = open('output.txt', 'w')
for count,line in enumerate(fil):
    if count == 0 :
        stack.append("pgm")
    tokens = line.strip().split()
```

There are 4 types of token

1. id , a character which can be both uppercase of lowercase

2. line_num , a number in [0,1000] attached in front of each instruction line

3. const , a number in [0,100]

4. + , - , < , = , IF , PRINT , GOTO , STOP , EOF

2. Parser

With the grammar given, they are splitted into the rule with left factor considered.

The grammar are

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' -> empty
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

Then, create a first set and follow set for each Non Terminal

Non Terminal	First Set	Follow Set
pgm	{EOF, line_num}	{EOF}
line	{line_num}	{EOF, line_num}
stmt	{id, IF, PRINT, GOTO, STOP}	{EOF, line_num}

asgmnt	{id}	{EOF, line_num}
exp	{id, const}	{EOF, line_num}
exp’	{+, -, empty}	{EOF, line_num}
term	{id, const}	{+, -, EOF, line_num, <, =}
if	{IF}	{EOF, line_num}
cond	{id, const}	{line_num}
cond’	{<, =}	{line_num}
print	{PRIN T}	{EOF, line_num}
goto	{GOTO}	{EOF, line_num}
stop	{STOP}	{EOF, line_num}

and their parsing table

	id	const	line_num	+	-	<	=	IF	PRINT	GOTO	STOP	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	17										
cond’						18	19					
print									20			
goto										21		
stop											22	

To check if the input is correct according to the given grammar and rules, each token is pushed into the stack sequentially, EOF is pushed firstly, then follow the method until the top of stack is EOF.

3. Code

Link to fullcode : https://github.com/patipon1998/Retro_Basic_Compiler.git

```
import sys

grammar ={
    1 : ["line", "pgm"], #pgm ->
line pgm
    2 : ["EOF"], #pgm -> EOF
    3 : ["lineNum", "stmt"], #line -
> lineNum stmt
    4 : ["asgmnt"], #stmt -> asgmnt
    5 : ["if"], #stmt -> if
    6 : ["print"], #stmt -> print
    7 : ["goto"], #stmt -> goto
    8 : ["stop"], #stmt -> stop
    9 : ["id", "=", "exp"], #asgmnt
-> id = exp
    10 : ["term", "exp'"], #exp ->
term exp'
    11 : ["+", "term"], #exp' -> +
term
    12 : ["-", "term"], #exp' -> -
term
    13 : None, #exp' -> empty
    14 : ["id"], #term -> id
    15 : ["const"], #term -> const
    16 : ["IF", "cond", "lineNum"],
#if -> IF cond lineNum
    17 : ["term", "cond'"], #cond ->
term cond'
    18 : ["<", "term"], #cond' -> <
term
    19 : ["=", "term"], #cond' -> =
term
    20 : ["PRINT", "id"], #print ->
PRINT id
    21 : ["GOTO", "lineNum"], #goto
-> GOTO lineNum
    22 : ["STOP"] #stop -> STOP
}

parsing_table = {
    "pgm": {"lineNum": 1, "EOF": 2},
    "line": {"lineNum": 3},
    "stmt": {"id": 4, "IF": 5,
"PRINT": 6, "GOTO": 7, "STOP": 8},
    "asgmnt": {"id": 9},
    "exp": {"id": 10, "const": 10},
    "exp'": {"+": 11, "-": 12,"EOF":
13, "lineNum": 13},
    "term": {"id": 14, "const": 15},
    "if": {"IF": 16},
    "cond": {"id": 17, "const": 17},
    "cond'": {"<": 18, "=": 19},
    "print": {"PRINT": 20},
    "goto": {"GOTO": 21},
    "stop": {"STOP": 22},
}

id = [chr(e) for e in
range(ord('A'), ord('Z')+1)]
lineNum = [str(e) for e in range(1,
1001)]
const = [str(e) for e in range(0,
101)]
terminal = ["+", "-", "IF" ,"<","=",
"PRINT", "GOTO", "STOP", "EOF"]
bCodeType = {
    "#line": 10,
    "#id": 11,
    "#const": 12,
    "#if": 13,
    "#goto": 14,
    "#print": 15,
    "#stop": 16,
    "#op": 17,
}

stack = ["EOF"]

def getTerminalType(token) :
    if token.isdigit():
        return "num"
    if token in terminal:
        return token
    if token in id:
        return "id"
    print("Invalid Input")
    exit()

def pairTerminal(token, top):
    termType =
getTerminalType(token)
    if termType != "num":
        return termType == top
    else :
        return top == "lineNum" or
top == "const"

def discipline(top,token):
    termType =
getTerminalType(token)
    if termType != "num" and
termType in parsing_table[top]:
        return
parsing_table[top][termType]
```

```
    if "lineNum" in
parsing_table[top]:
        return
parsing_table[top]["lineNum"]
    if "const" in
parsing_table[top]:
        return
parsing_table[top]["const"]
    print("Invalid Grammar")
    exit()

def parser(token):
    while not pairTerminal(token,
stack[-1]):
        top = stack.pop()
        if top not in parsing_table
:
            print("Invalid
Grammar555")
            exit()
        principle =
discipline(top,token)
        if grammar[principle] !=
None :

stack.extend(grammar[principle][::-
1])
    if(stack[-1] == 'lineNum' and
not 1 <= int(token) <= 1000):
        print("Invalid Grammar")
        exit()
    if(stack[-1] == 'const' and not
0 <= int(token) <= 100):
        print("Invalid Grammar")
        exit()
    return stack.pop()

def getBCode(term,val) :
    if(term == "+"):
        return ("#op", 1)
    if(term == "-"):
        return ("#op", 2)
    if(term == "<"):
        return ("#op", 3)
    if(term == "="):
        return ("#op", 4)
    if(term == "lineNum"):
        return ("#line", int(val))
    if(term == "id"):
        return ("#id", ord(val) -
ord('A') + 1)
    if(term == "const"):
        return ("#const", int(val))
    if(term == "IF"):
        return ("#if", 0)
    if(term == "GOTO"):
        return ("#goto", int(val))
    if(term == "PRINT"):
        return ("#print", 0)
    if(term == "STOP"):
        return ("#stop", 0)

def genBCode(parsed) :
    bCodeList = list()
    for i in range(len(parsed)):
        if(parsed[i][0] not in
["GOTO","lineNum"] or i == 0):

bCodeList.append(getBCode(parsed[i][
0], parsed[i][1]))
        else:
            if(parsed[i][0] ==
'lineNum' and i != 0):

bCodeList.append(getBCode("GOTO",par
sed[i][1]))
    return bCodeList

def toBCode(token):
    parsed = list()
    for tok in token :

parsed.append((parser(tok),tok))
    bCodeList = genBCode(parsed)
    bCodeLine = ""
    for types, value in bCodeList:
        bCodeLine = bCodeLine +
str(bCodeType[types]) + " " +
str(value) + " "
    return bCodeLine.strip()

path = str(sys.argv[1])
fil = open(path,'r')
out = open('output.txt', 'w')
for count,line in enumerate(fil):
    if count == 0 :
        stack.append("pgm")
    tokens = line.strip().split()
    bcode = toBCode(tokens)
    print(bcode)
    out.write(bcode + "\n")
print("0")
out.write("0\n")
fil.close()
out.close()
```