Retro Basic grammar

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

But need to rewrite the rule “exp” and “cond” to make it LL(1) parsable.

Rewritten Retro Basic grammar

```
pgm := line pgm | λ
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
```

This can generate First Set and Follow Set from the grammar which show in the table below.

First Set

First(pgm)	line_num, λ
First(line)	line_num
First(stmt)	id, PRINT, STOP, IF, GOTO
First(asgmnt)	id
First(exp)	id, const
First(exp')	+, -, λ
First(term)	id, const
First(if)	IF
First(cond)	id, const
First(cond')	<, =
First(print)	PRINT
First(goto)	GOTO
First(stop)	STOP

Follow Set

Follow(pgm)	$
Follow(line)	line_num, $
Follow(stmt)	line_num, $
Follow(asgmnt)	line_num, $
Follow(exp)	line_num, $
Follow(exp')	line_num, $
Follow(term)	<, =, +, -, line_num, $
Follow(if)	line_num, $
Follow(cond)	line_num
Follow(cond')	line_num
Follow(print)	line_num, $
Follow(goto)	line_num, $
Follow(stop)	line_num, $

Then we separate the grammar to make **LL(1) Parsing Table**

1. pgm := line pgm
2. pgm := λ
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
```

LL(1) Parsing Table

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

1. A scanner, scan the source and separate characters into token by using split command in python to split the tokens and insert it into the list.
2. A parser, check the sequence of token that it is correct according to the grammar by using parsing algorithm shown below.

Push the start symbol into the stack

WHILE stack is not empty ($ is not on top of stack) and the stream of tokens is not empty (the next input token is not $)

SWITCH (Top of stack, next token)

CASE (terminal a, a):

Pop stack; Get next token

CASE (nonterminal A, terminal a):

IF the parsing table entry M[A, a] is not empty THEN

Get A -> X1 X2 ... Xn from the parsing table entry M[A, a] Pop stack;

Push Xn ... X2 X1 into stack in that order

ELSE Error

CASE ($,$): *Accept*

OTHER: *Error*

Code (Written by Python language)

This code can be download from https://goo.gl/JQAC8A or see it below. (read from top-left to the bottom-left and move to the top-right to the bottom-right)

Remark: Need to change source file name to “retro_basics_source.txt” (without quote) and place it in the same folder of this program.

```
import sys


##########################
#NON-TERMINAL DECLARATION#
##########################
n_pgm = 1
n_line = 2
n_stmt = 3
n_asgmnt = 4
n_exp = 5
n_expp = 6
n_term = 7
n_if = 8
n_cond = 9
n_condp = 10
n_print = 11
n_goto = 12
n_stop = 13


######################
#TERMINAL DECLARATION#
######################
t_line_num = 14
t_id = 15
t_IF = 16
t_PRINT = 17
t_GOTO = 18
t_STOP = 19
```

```
t_plus = 20
t_minus = 21
t_const = 22
t_less = 23
t_equal = 24
t_goto_num = 25

#########################
#CONSTRAINTS DECLARATION#
#########################
def isLineNum(s):
    i = int(s)
    if 1 <= i <= 1000:
        return 1
    return 0

def isConst(s):
    i = int(s)
    if 0 <= i <= 100:
        return 1
    return 0

def isId(s):
    if len(s) != 1:
        return 0
    if 'A' <= s[0] <= 'Z':
        return 1
    return 0

##############
#MAIN PROGRAM#
##############
file = open("retro_basics_source.txt")
tokens = []
stack = []
tp = 0
check = 1

for line in file:
    for x in line.strip().split(" "):
        tokens.append(x)
stack.append(n_pgm)
file.close()

###################
#PARSING ALGORITHM#
###################
while len(stack) != 0 and tp <= len(tokens):

    if check == 0:
        print("\nFailed to parse: Incorrect
syntax")
        sys.exit()
    top = stack[len(stack)-1]
    stack.pop()
    if tp == len(tokens):
        break

#NON-TERMINAL
    if top == n_pgm:
        if tp == len(tokens):
            check = 1
        if isLineNum(tokens[tp]):
            stack.append(n_pgm)
            stack.append(n_line)
            check = 1
        continue

    if top == n_line:
        if isLineNum(tokens[tp]):
            stack.append(n_stmt)
            stack.append(t_line_num)
            check = 1
```

```
            else:
                check = 0
            continue


        if top == n_stmt:
            if isId(tokens[tp]):
                stack.append(n_asgmnt)
                check = 1
            elif tokens[tp] == "IF":
                stack.append(n_if)
                check = 1
            elif tokens[tp] == "PRINT":
                stack.append(n_print)
                check = 1
            elif tokens[tp] == "GOTO":
                stack.append(n_goto)
                check = 1
            elif tokens[tp] == "STOP":
                stack.append(n_stop)
                check = 1
            else:
                check = 0
            continue


        if top == n_asgmnt:
            if isId(tokens[tp]):
                stack.append(n_exp)
                stack.append(t_equal)
                stack.append(t_id)
                check = 1
            else:
                check = 0
            continue


        if top == n_exp:
            if isId(tokens[tp]) or
isConst(tokens[tp]):
                stack.append(n_expp)
                stack.append(n_term)
                check = 1
            else:
                check = 0
            continue


        if top == n_expp:
            if tokens[tp] == "+":
                stack.append(n_term)
                stack.append(t_plus)
            elif tokens[tp] == "-":
                stack.append(n_term)
                stack.append(t_minus)
            check = 1
            continue


        if top == n_term:
            if isId(tokens[tp]):
                stack.append(t_id)
                check = 1
            elif isConst(tokens[tp]):
                stack.append(t_const)
                check = 1
            else:
                check = 0
            continue


        if top == n_if:
            if tokens[tp] == "IF":
                stack.append(t_goto_num)
                stack.append(n_cond)
                stack.append(t_IF)
                check = 1
            else:
                check = 0
```

```
        continue


    if top == n_cond:
        if isId(tokens[tp]) or
isConst(tokens[tp]):
            stack.append(n_condp)
            stack.append(n_term)
            check = 1
        else:
            check = 0
        continue


    if top == n_condp:
        if tokens[tp] == "<":
            stack.append(n_term)
            stack.append(t_less)
            check = 1
        elif tokens[tp] == "=":
            stack.append(n_term)
            stack.append(n_equal)
            check = 1
        else:
            check = 0
        continue


    if top == n_print:
        if tokens[tp] == "PRINT":
            stack.append(t_id)
            stack.append(t_PRINT)
            check = 1
        else:
            check = 0
        continue


    if top == n_goto:
        if tokens[tp] == "GOTO":
            stack.append(t_line_num)
            stack.append(t_GOTO)
            check = 1
        else:
            check = 0
        continue


    if top == n_stop:
        if tokens[tp] == "STOP":
            stack.append(t_STOP)
            check = 1
        else:
            check = 0
        continue


#TERMINAL
    if top == t_line_num:
        if(isLineNum(tokens[tp])):
            print("\n10 " + tokens[tp], end
= " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_id:
        if isId(tokens[tp]):
            print("11 " +
str(ord(tokens[tp][0]) - 64), end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_IF:
        if tokens[tp] == "IF":
```

```
            print("13 0", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_PRINT:
        if tokens[tp] == "PRINT":
            print("15 0", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_GOTO:
        if tokens[tp] == "GOTO" and
isLineNum(tokens[tp+1]):
            stack.pop()
            print("14 " + tokens[tp+1], end
= " ")
            tp += 1
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_STOP:
        if tokens[tp] == "STOP":
            print("16 0\n0", end = "")
            check = 1
        else:
            check = 0
        tp += 1
        continue
```

```
    if top == t_plus:
        if tokens[tp]== "+":
            print("17 1", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_minus:
        if tokens[tp] == "-":
            print("17 2", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_const:
        if isConst(tokens[tp]):
            print("12 " + tokens[tp], end =
" ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_less:
        if tokens[tp] == "<":
            print ("17 3", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue
```

```
    if top == t_equal:
        if tokens[tp] == "=":
            print("17 4", end = " ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


    if top == t_goto_num:
        if isLineNum(tokens[tp]):
            print("14 " + tokens[tp], end =
" ")
            check = 1
        else:
            check = 0
        tp += 1
        continue


if check == 0 or len(stack) != 0 or tp !=
len(tokens):
    print("\nFailed to parse: Incorrect
syntax")
    sys.exit()
else:
    print("\nParsing done")
```