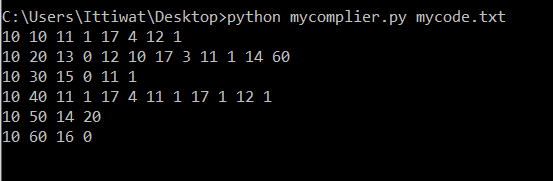
**5931077721 นาย อิทธิวัฒน์ สุขุมธนากุล**

**Mycomplier**

หลักการทำงานเบื่องต้นของ complier อันนี้คือการแบ่ง input ออกมาเป็นบรรทัดแล้วทำการแปลง input แต่ละบรรทัดเป็น B-code โดยใช้ recursive function ที่รับ input เป็น list of language ของบรรทัดนั้น, index, line number, state ที่ gramma เราคาดหวังสำหรับคำถัดไป ถ้าเกิดเจอ คำที่ไม่เป็นไปถามหลัก gramma (ถ้าผิดจาก state ที่ควรจะเป็นจะทำการแสดงผล error ออกมา)



ภาพที่ 1 ตัวอย่างการใช้งานโปรแกรม

**วิเคราะห์การทำงาน scanner**

การทำงานของ scanner นั้นจะทำงานโดย scan input หลังจากการแบ่งบรรทัดแล้ว ไปทีละคำโดยที่ระหว่าง scan จะทำการวิเคราะห์ไปด้วยว่าตำแหน่งนี้ควรจะเป็นค่าอะไร โดยแยกจาก state ที่คำก่อนหน้าให้มา ยกตัวอย่างเช่น 10 อาจจะเป็นได้ทั้ง line num หรือ const เราจะแยกมันจาก state ถ้าเป็นคำแรก state เป็น line num แต่ถ้ามาต่อท้าย operator จะคิดว่า state มันจะเป็น const

**วิเคราห์การทำงาน parser**

สำหรับโค้ดที่เขียนขึ้นการวิเคราะห์ความถูกต้องของ gramma สามารถทำได้ระหว่างการ scan ได้เลย โดยที่ดูจาก state ถ้าเกิดคำที่เราสนใจไม่ตรงกับ state ที่เราคาดหวังไว้จะถือว่าเป็น error และจะทำการแสดงผล error ออกมาทางหน้าจอ

**สามารถเข้าไปดูการทำงานได้ที่**

[**https://github.com/ongss/my-simple-complier/**](https://github.com/ongss/my-simple-complier/)

**ตัวอย่างโค้ดอยู่หน้าถัดไป >>**

**Code : python**

import sys

MAX\_LINE\_NUM = 1000

ID = {'A':'1','B':'2','C':'3','D':'4','E':'5','F':'6','G':'7','H':'8','I':'9','J':'10','K':'11','L':'12','M':'13','N':'14','O':'15','P':'16','Q':'17','R':'18','S':'19','T':'20','U':'21','V':'22','W':'23','X':'24','Y':'25','Z':'26' }

MAX\_CONST = 100

COMMAND = {'IF','GOTO','PRINT','STOP'}

OP = {'+':'1','-':'2','<':'3','=':'4'}

ans = ""

error = False

error\_msg = ''

openfile = False

example\_code = "10 A = 1\n20 IF 10 < A 60\n30 PRINT A\n40 A = A + 1\n 50 GOTO 20\n60 STOP"

def conv\_to\_list(s):

temp = s.split('\n')

for i in range(len(temp)):

temp[i] = temp[i].split()

return temp

def read\_file(file\_name):

temp = ''

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

for line in file:

temp = temp + line

return temp

def write(word):

global ans

ans = ans + word

def translater(temp,idx,state,line):

global error

global error\_msg

if(idx < len(temp)):

if(state == 'line\_num'):

if(temp[idx].isdigit() and int(temp[idx]) < 1000):

write('10 '+temp[idx]+' ')

translater(temp,idx+1,'stmt',line)

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected line\_num but has '+temp[idx]+'\n'

elif(state == 'stmt'):

if(temp[idx] in ID.keys() and temp[idx+1] == '='):

write('11 '+ID[temp[idx]]+' 17 4 ')

translater(temp,idx+2,'exp',line)

elif(temp[idx] == 'IF'):

write('13 0 ')

translater(temp,idx+1,'cond',line)

elif(temp[idx] == 'GOTO' and temp[idx+1].isdigit() and int(temp[idx+1]) < 1000):

write('14 '+temp[idx+1]+' ')

translater(temp,idx+2,'end',line)

elif(temp[idx] == 'PRINT'):

write('15 0 ')

translater(temp,idx+1,'id',line)

translater(temp,idx+2,'end',line)

elif(temp[idx] == 'STOP'):

write('16 0 ')

translater(temp,idx+1,'end',line)

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected stmt but has '+temp[idx]+'\n'

elif(state == 'exp'):

if(len(temp) > idx+2 and (temp[idx+1] == '+' or temp[idx+1] == '-')):

translater(temp,idx,'term',line)

translater(temp,idx+1,'op',line)

translater(temp,idx+2,'term',line)

translater(temp,idx+3,'end',line)

else:

translater(temp, idx,'term',line)

elif(state == 'cond'):

translater(temp,idx,'term',line)

translater(temp,idx+1,'com',line)

translater(temp,idx+2,'term',line)

translater(temp,idx+4,'end',line)

write('14 '+temp[idx+3]+' ')

elif(state == 'term'):

if(temp[idx] in ID.keys()):

translater(temp,idx,'id',line)

elif(temp[idx].isdigit()):

translater(temp,idx,'const',line)

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected term but has '+temp[idx]+'\n'

elif(state == 'op'):

if(OP[temp[idx]] == '1' or OP[temp[idx]] == '2'):

write('17 '+OP[temp[idx]]+' ')

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected oparator but has '+temp[idx]+'\n'

elif(state == 'com'):

if(OP[temp[idx]] == '3' or OP[temp[idx]] == '4'):

write('17 '+OP[temp[idx]]+' ')

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected comparator but has '+temp[idx]+'\n'

elif(state == 'id'):

if(temp[idx] in ID.keys()):

write('11 '+ID[temp[idx]]+' ')

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected id but has '+temp[idx]+'\n'

elif(state == 'const'):

if(temp[idx].isdigit() and int(temp[idx]) <= 100):

write('12 '+temp[idx]+' ')

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected const but has '+temp[idx]+'\n'

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' expected end but has '+temp[idx]+'\n'

elif(state == 'end'): return True # FINISH

else:

error = True # ERROR

error\_msg += 'at line '+str(line)+' has too much arguments\n'

def complie\_with\_file(file\_name):

global error

error = False

s = read\_file(file\_name)

temp = conv\_to\_list(s)

for i in range(len(temp)):

translater(temp[i],0,'line\_num',i)

if(error): break

ans.strip()

write('\n')

ans.strip()

return error

def complie(s):

global error

error = False

temp = conv\_to\_list(s)

for i in range(len(temp)):

translater(temp[i],0,'line\_num',i)

if(error): break

ans.strip()

write('\n')

ans.strip()

return error

def main():

if(openfile):

if(len(sys.argv) != 2): print("invalid arguments")

else:

file\_name = sys.argv[1]

complie\_with\_file(file\_name)

if(error): print("syntax error :",error\_msg)

else: print(ans)

else:

complie(example\_code)

if(error): print("syntax error :",error\_msg)

else: print(ans)

main()