Compiler

Student number: 20166450
Deadline 2019.04.28.
Student name: Young Min Kim

■Source Code

```python
#define TOKENS based on requirements.
keyword = ['return','RETURN', 'WHILE','while','ELSE','else','IF', 'if']
type1 = ['int','INT']
type2 = ['char','CHAR']
number=['0','1','2','3','4','5','6','7','8','9']
alphabet=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
operator=['=','<','>','<=','>=','==','!=']
arith_operator=['+','-','*','/']
terminate_symbol=[';']
white_space=['\n', '\t',' ']
pair_symbol=['{','}','(',')']
comma=[',']

#Combine all of Tokens which i defined.
KEYWORDS = arith_operator+terminate_symbol+pair_symbol+keyword+type1+comma+type2+operator+white_space
```

1~15: Define Tokens.

```python
# Function of True, False based on following dfa which is accepted. or not.
def accepts(transitions,initial,accepting,s):
    state = initial
    for c in s:
        if c in transitions[state].keys():
            state = transitions[state][c]
        else:
            return bool(0)
    b= state in accepting
    return b
```

21~30: Function of performing DFA

```python
#Integer DFA
Integer_dfa = {0:{'0':3,'-':1, '1':2,'2':2,'3':2,'4':2,'5':2,'6':2,'7':2,'8':2,'9':2},
               1:{'1':2,'2':2,'3':2,'4':2,'5':2,'6':2,'7':2, '8':2,'9':2},
               2:{'0':2,'1':2,'2':2,'3':2,'4':2,'5':2,'6':2,'7':2, '8':2,'9':2},
               3:{}}
#Literal DFA
Literal_dfa={0:{'"':1},
             1:{'a':1,'b':1,'c':1,'d':1,'e':1,'f':1,'g':1,'h':1,'i':1,'j':1,'k':1,'l':1,'m':1,
                'n':1,'o':1,'p':1,'q':1,'r':1,'s':1,'t':1,'u':1,'v':1,'w':1,'x':1,'y':1,'z':1,
                '0':1,'1':1,'2':1,'3':1,'4':1,'5':1,'6':1,'7':1,'8':1,'9':1,'A':1,'B':1,'C':1,
                'D':1,'E':1,'F':1,'G':1,'H':1,'I':1,'J':1,'K':1,'M':1,'N':1,'O':1,'P':1,'Q':1,
                'R':1,'S':1,'T':1,'U':1,'V':1,'W':1,'X':1,'Y':1,'Z':1,'\n':1,'\t':1,' ':1,'"':2},
             2:{}}

#Identifier DFA
Identifier_dfa={0:{'a':1,'b':1,'c':1,'d':1,'e':1,'f':1,'g':1,'h':1,'i':1,'j':1,'k':1,'l':1,'m':1,
                   'n':1,'o':1,'p':1,'q':1,'r':1,'s':1,'t':1,'u':1,'v':1,'w':1,'x':1,'y':1,'z':1},
                1:{'a':1,'b':1,'c':1,'d':1,'e':1,'f':1,'g':1,'h':1,'i':1,'j':1,'k':1,'l':1,'m':1,
                   'n':1,'o':1,'p':1,'q':1,'r':1,'s':1,'t':1,'u':1,'v':1,'w':1,'x':1,'y':1,'z':1,
                   '0':1,'1':1,'2':1,'3':1,'4':1,'5':1,'6':1,'7':1,'8':1,'9':1,'A':1,'B':1,'C':1,
                   'D':1,'E':1,'F':1,'G':1,'H':1,'I':1,'J':1,'K':1,'M':1,'N':1,'O':1,'P':1,'Q':1,
                   'R':1,'S':1,'T':1,'U':1,'V':1,'W':1,'X':1,'Y':1,'Z':1}}
```

32~53: In case of Integer, Literal, Identifier DFA, I used dictionary type.

```
55  #open the file
56  file = open('test4.c','r')
57  str=file.read()
58
59  #save it on the test.out
60  fileout= open('test4.out','w')
61  print("    <Lexical Analyzer Table>    ")
62  print("_____________________________")
63  fileout.write("    <Lexical Analyzer Table>    ₩n")
64  fileout.write("_____________________________₩n")
```

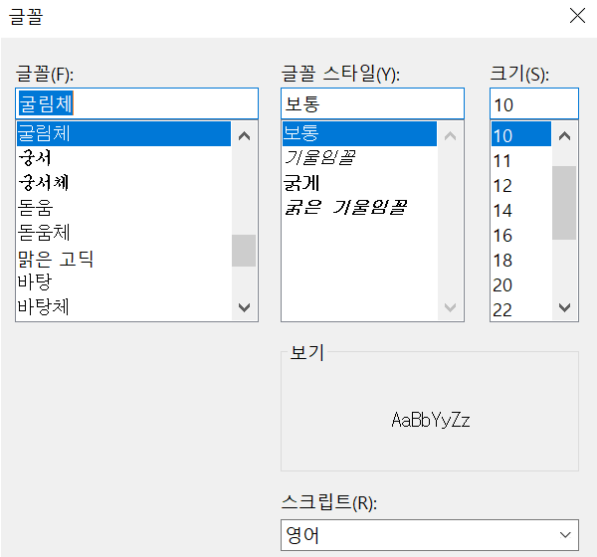
55~64: Open Input file. (test~.c) and Write Output file. (test.out)

```
67  #i is index of 'str' and char is value of 'str' about index
68  for i,char in enumerate(str):
69      #Not to show whit space in lexical table
70      if char not in white_space:
71          lexeme += char
72
73      #to prevent an error
74      if (i+1) < len(str):
75          #This is not to confuse about each comparison operators
76          if (str[i] in operator) and str[i + 1] == '=':
77              continue
78          #Keyword is all of tokens.
79          elif str[i+1] == blank or str[i+1] in KEYWORDS or lexeme in KEYWORDS :
```

68~184: Using for statement and if statement, Print and save in output file DFA tokens.

■Input Output Capture

If you use .txt. Please change Font Style following the picture.



Problem!!

```c
1 int func(int a, char b){
2     char test;
3     int c = -11112;
4
5     return 0;
6 }
```

This is input 〈test.c〉. Please note Line 3.

```
C:₩Users₩newmi₩PycharmProjects₩untitled3₩venv₩Scripts₩python.exe C:/Users/newmi/PycharmProjects/untitled3/Lexer.py
    <Lexical Analyzer Table>

INT             |  int
IDENTIFIER      |  func
LPAREN          |  (
INT             |  int
IDENTIFIER      |  a
COMMA           |  ,
CHAR            |  char
IDENTIFIER      |  b
RPAREN          |  )
LBRACE          |  {
CHAR            |  char
IDENTIFIER      |  test
SEMI            |  ;
INT             |  int
IDENTIFIER      |  c
ASSIGN          |  =
OPERATOR        |  -
INTEGER         |  11112
SEMI            |  ;
IDENTIFIER      |  return
INTEGER         |  0
SEMI            |  ;
RBRACE          |  }
```

And this is output of test.c. When I put -11112 as a integer, My lexer cannot recognize that as a integer. It shows '–' as a operator.
So I tried to solve this problem.
Next pic is code that I've solved.

```
        #This is additional code for an error that I've found.
        elif str[i] == '-':
            d = i
            while str[d] == blank:
                d -= 1
            if str[d] in operator or arith_operator:
                continue
```

Here is what I've solved. If there are any operator just before '–', it becomes integer except blank.

```
C:₩Users₩newmi₩PycharmProjects₩untitled3₩venv₩Scripts₩python.exe C:/Users/newmi/PycharmProjects/untitled3/Lexer.py
    <Lexical Analyzer Table>
_____________________________
INT             |  int
IDENTIFIER      |  func
LPAREN          |  (
INT             |  int
IDENTIFIER      |  a
COMMA           |  ,
CHAR            |  char
IDENTIFIER      |  b
RPAREN          |  )
LBRACE          |  {
CHAR            |  char
IDENTIFIER      |  test
SEMI            |  ;
INT             |  int
IDENTIFIER      |  c
ASSIGN          |  =
INTEGER         |  -11112
SEMI            |  ;
IDENTIFIER      |  return
INTEGER         |  0
SEMI            |  ;
RBRACE          |  }
```

This is the same input I used. After I changed my code , my lexer recognized '–11112' as a integer. Because of this, I complete all of requirement.

〈input.2〉

```
test2.c
Project63        (전역 범위)        operationfunc1(int a, int b)
int operationfunc1(int a, int b)
{
    char c= (char)a - b;
    int d = -2134;
    d = 0;
    while (b<=1)
        a = 1;

    return 2;
}
```

〈ouput.2 of input.2〉

test2.out - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
    <Lexical Analyzer Table>
_____________________________
INT             |  int
IDENTIFIER      |  operationfunc1
LPAREN          |  (
INT             |  int
IDENTIFIER      |  a
COMMA           |  ,
INT             |  int
IDENTIFIER      |  b
RPAREN          |  )
LBRACE          |  {
CHAR            |  char
IDENTIFIER      |  c
ASSIGN          |  =
LPAREN          |  (
CHAR            |  char
RPAREN          |  )
IDENTIFIER      |  a
OPERATOR        |  -
IDENTIFIER      |  b
SEMI            |  ;
INT             |  int
IDENTIFIER      |  d
ASSIGN          |  =
INTEGER         |  -2134
SEMI            |  ;
IDENTIFIER      |  d
ASSIGN          |  =
INTEGER         |  0
SEMI            |  ;
KEYWORD         |  while
LPAREN          |  (
IDENTIFIER      |  b
COMPARISON      |  <=
INTEGER         |  1
RPAREN          |  )
IDENTIFIER      |  a
ASSIGN          |  =
INTEGER         |  1
SEMI            |  ;
RETURN          |  return
INTEGER         |  2
SEMI            |  ;
```

〈input 3〉

input4.c

Project63 (전역 범위) foofunc(int a, int b)

```c
int foofunc(int a, int b) {
    int c = a * b;
    c = a - b;
    c = a + b;
    c = a / b;
    return 0;
}
```

〈output 3 of input3〉

test4.out - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
    <Lexical Analyzer Table>
_________________________________
INT             |  int
IDENTIFIER      |  foofunc
LPAREN          |  (
INT             |  int
IDENTIFIER      |  a
COMMA           |  ,
INT             |  int
IDENTIFIER      |  b
RPAREN          |  )
LBRACE          |  {
INT             |  int
IDENTIFIER      |  c
ASSIGN          |  =
IDENTIFIER      |  a
OPERATOR        |  *
IDENTIFIER      |  b
SEMI            |  ;
IDENTIFIER      |  c
ASSIGN          |  =
IDENTIFIER      |  a
OPERATOR        |  -
IDENTIFIER      |  b
SEMI            |  ;
IDENTIFIER      |  c
ASSIGN          |  =
IDENTIFIER      |  a
OPERATOR        |  +
IDENTIFIER      |  b
SEMI            |  ;
IDENTIFIER      |  c
ASSIGN          |  =
IDENTIFIER      |  a
OPERATOR        |  /
IDENTIFIER      |  b
SEMI            |  ;
RETURN          |  return
INTEGER         |  0
SEMI            |  ;
```