```
# CS3342 Lab2A - Lexical Analysis - Lexer
# This program will read an input file.
# Take the input character stream and convert it into tokens.
# Input file: Need file name and its path.  Be sure the path is correct.
import re


class Token:
    """ A simple Token structure. Token type, value and position.
    """

    def __init__(self, type, val, pos):
        self.type = type
        self.val = val
        self.pos = pos

    def __str__(self):
        return '%s(%s) at %s' % (self.type, self.val, self.pos)


class Lexer:
    """ A simple regex-based lexer/tokenizer.
    """

    def __init__(self, rules, skip_whitespace=True):
        """ Create a lexer.
            rules:
                A list of rules. Each rule is a `regex, type`
                pair, where `regex` is the regular expression used
                to recognize the token and `type` is the type
                of the token to return when it's recognized.
            skip_whitespace:
                If True, whitespace (\s+) will be skipped and not
                reported by the lexer. Otherwise, you have to
                specify your rules for whitespace, or it will be
                flagged as an error.
        """
        self.rules = []
        for regex, type in rules:
            self.rules.append((re.compile(regex), type))
        self.skip_whitespace = skip_whitespace
        self.re_ws_skip = re.compile('\S')

    def input(self, buf):
        """ Initialize the lexer with a buffer as input.
        """
        self.buf = buf
        self.pos = 0

    def token(self):
        """ Return the next token (a Token object) found in the
            input buffer. None is returned if the end of the
            buffer was reached.
            In case of a lexing error (the current chunk of the
            buffer matches no rule), a LexerError is raised with
            the position of the error.
        """
        if self.pos >= len(self.buf):
            return None
        if self.skip_whitespace:
            m = self.re_ws_skip.search(self.buf, self.pos)
            if m:
                self.pos = m.start()
            else:
                return None
        for regex, type in self.rules:
            m = regex.match(self.buf, self.pos)
            if m:
                tok = Token(type, m.group(), self.pos)
                self.pos = m.end()
                return tok
        # if we're here, no rule matched
        print("invalid token on this line at ", self.pos, ":", self.buf)

    def tokens(self):
        """ Returns an iterator to the tokens found in the buffer.
        """
        while 1:
            tok = self.token()
            if tok is None: break
            yield tok


# Rules to categorize the tokens
rules = [
    ('\d+', 'NUMBER'),
    ('[a-zA-Z_]\w*', 'IDENTIFIER'),
    ('\+', 'PLUS'),
    ('\-', 'MINUS'),
    ('\*', 'MULTIPLY'),
    ('\/', 'DIVIDE'),
    ('\(', 'LP'),
    ('\)', 'RP'),
    ('=', 'EQUALS'),
    ('\{', 'LBRACE'),
    ('\}', 'RBRACE'),
    ('\,', 'COMMA'),
    ('\;', 'SEMICOLON'),
]
data = ""
lx = Lexer(rules, skip_whitespace=True)
header = ['Lexeme', 'Token']
print("{: >20} {: >20}".format(*header))
keywords = ['int', 'float', 'return']
for line in open('c:\\users\\pc\\downloads\\lab2_test.c', 'r'):
    li = line.strip()
    # The following will skip the comments.
    if not li.startswith("//"):
        lx.input(line.rstrip())
        for tok in lx.tokens():
            # remove right part of token string
            st = str(tok).rstrip('at 0123456789')
            tokens = st.split('(')
            tokens[1] = tokens[1].rstrip(')')
            if tokens[0] == 'LP':
                print("{: >20} {: >20}".format('(', 'LPAREN'))
            elif tokens[0] == 'RP':
                print("{: >20} {: >20}".format(')', 'RPAREN'))
```

```
elif tokens[1] in keywords:
    print("{: >20} {: >20}".format(tokens[1], 'KEYWORD'))
else:
    print("{: >20} {: >20}".format(tokens[1], tokens[0]))
```