# INF-341: Compiladores
# Tarea #2
# "Le seguimos haciendo al léxico"

Anghelo Carvajal
201473062-4

21 de octubre de 2018

# 1. Introducción

En nuestro analizador léxico para archivos `.ini`, definimos 8 tipos de tokens y 4 estados distintos de modo de poder identificar correctamente este archivo, para luego almacenar cada token en una estructura tipo diccionario.

El código mostrado son fragmentos del código completo final.

# 2. Analizador léxico

## 2.1. Tokens

Los tokens están definidos de la siguiente manera:

- `NEWLINE`. `r"(\n|\r|(\r\n))"`
  Calza los saltos de linea e indica que debemos entrar al estado inicial (`INITIAL`).

  Indicamos que retorna `None` para especificar que no nos interesa esto.

- `COMMENT`. `r"[;|#].*"`
  Calza los comentarios que empiecen con `;` o con `#`.

  Retorna nada, de modo que ignore los comentarios.

- `KEY`. `r"[^=;\#\[\]]+"`
  Calza una llave o identificador. Este identificador esta definido como cualquier carácter que no sea un ‘`=`’ (separador), ‘`;`’, ‘`#`’ (comentarios) ‘`[`’ o ‘`]`’ (reservado para las secciones).

  Además indicamos que debemos entrar al estado `KEY`.

  Finalmente, tal como especifica la documentación proveída, tanto los identificadores como las secciones son *case insensitivity*, de modo que retornamos el valor transformado a minúsculas.

- `KEY_COMMENT`. `r"[ \t]*[;|\#].*"`
  Si estando en el estado `KEY` nos encontramos con un comentario, nos devolvemos al estado inicial.

  Si llegamos a este token, significa que el identificador fue *seteado* sin un valor correspondiente, y tan solo se escribió un comentario al costado.

- `KEY_NEWLINE`. `r"[ \t]*(\n|\r|(\r\n))"`.
  Similar al anterior, con la diferencia de que no hay comentarios al costado.

- `KEY_SEPARATOR`. `r"[ \t]*=[ \t]*"`
  Estando en el estado `KEY`, calza el símbolo ‘`=`’ pudiendo estar rodeado de espacios o tabulaciones.

  Después de esto entramos al estado `SEPARATOR`.

- `SEPARATOR_VALUE`. `r"[^;\#\n\r]+"`
  El valor asociado a un identificador.

  Solo entramos aquí si es que antes hemos calzado un identificador y un separador.

  Puede ser cualquier carácter que no sean los reservados para comentarios, ni el salto de linea.

  Además, se quitan todos los espacios sobrantes al inicio o al final del valor.

  También, si el valor es un numero entero positivo, este es transformado a una variable del tipo `int`, y si es un valor booleano (`true` o `false`, `case insensitive`) lo convierte a una variable del tipo `bool`.

  Finalmente indica de que debemos devolvernos al estado `INITIAL`.

- `SECTION`. `r"\["`
  Calza el caracter [. Luego indicamos que entramos al estado `SECTION`

- `SECTION_DATA`. `r"[a-zA-Z\d\-\._]+"`
  El nombre de la sección. Solo entramos aquí si ya calzamos un ‘`[`’ con anterioridad.

  Esto calza caracteres alfanuméricos, guiones (-), puntos (.) y guiones bajos (_).

  Luego entramos al estado `DATA`.

- `DATA_END`. `r"\]"`
  El final de una sección.

  Solo entramos aquí si es que ya encontramos un nombre para la sección. Esto es útil para evitar secciones con nombres vacíos.

  Luego nos vamos al estado `INITIAL`.

## 2.2. Estados

Se definieron 4 estados, complementando a `INITIAL`. Se detallan a continuación:

- `KEY`. Indica que hemos calzado un identificador.

  Estando en este estado, solo podemos consumir los símbolos de comentarios o nueva linea, lo cual nos llevaría al estado `INITIAL`; o consumir el símbolo separador, lo cual nos lleva al estado `SEPARATOR`.

- `SEPARATOR`. Indica que hemos calzado el separador.

  Solo podemos entrar a este estado si es que ya nos encontramos en el estado `KEY`.

  En este estado, solo podemos consumir lo correspondiente al valor de un identificador, lo cual nos llevara al estado inicial nuevamente. Consumir cualquier otra cosa es un error.

- `SECTION` Indica que hemos calzado el símbolo de inicio de sección.

  En este estado solo podemos calzar el nombre de la sección, lo cual debería llevarnos al estado `DATA`

- `DATA`. Indica que hemos calzado el nombre de la sección.

  Estando en este estado, solo podemos calzar el final de la sección, llevándonos al estado `INITIAL`.

## 2.3. La implementación

A continuación, se puede ver el código del analizador léxico. Se han omitido las funciones que manejan errores.

Código 1: `analizador_lexico.py`

```
1  import ply.lex as lex
2
3  tokens = (
4     "KEY",
5     "SEPARATOR",
6     "VALUE",
7     "COMMENT",
8     "NEWLINE",
9     "SECTION",
10    "DATA",
11    "END",
12 )
13
14 states = (
15      ("KEY", "exclusive"),
16      ("SEPARATOR", "exclusive"),
17      ("SECTION", "exclusive"),
18      ("DATA", "exclusive"),
19 )
20
21 t_ignore                = "      "
22 t_KEY_ignore            = t_ignore
23 t_SEPARATOR_ignore      = t_ignore
24 t_SECTION_ignore        = t_ignore
25 t_DATA_ignore           = t_ignore
26
27
28 def t_NEWLINE(t):
29      r"(\n|\r|(\r\n))"
30      t.lexer.begin('INITIAL')
31      return
32
33
34 def t_COMMENT(t):
35      r"[;|\#].*"
36      return
37
38
39 def t_KEY(t):
40      r"[^=;\#\[\]\s]+"
41      t.lexer.begin("KEY")
42      t.value = t.value.lower()
```

```
43      return t
44
45  def t_KEY_COMMENT(t):
46      r"[ \t]*[;|\#].*"
47      t.lexer.begin("INITIAL")
48      return
49
50  def t_KEY_NEWLINE(t):
51      r"[ \t]*(\n|\r|(\r\n))"
52      t.lexer.begin("INITIAL")
53      return
54
55  def t_KEY_SEPARATOR(t):
56      r"[ \t]*=[ \t]*"
57      t.value = t.value.strip().lower()
58      t.lexer.begin("SEPARATOR")
59      return t
60
61
62  def t_SEPARATOR_VALUE(t):
63      r"[^;\#\n\r]+"
64      t.value = t.value.strip()
65      aux = t.value.lower()
66      if(t.value.isdigit()):
67          t.value = int(t.value)
68      elif(aux in ("false", "true")):
69          t.value = aux == "true"
70      t.lexer.begin("INITIAL")
71      return t
72
73
74  def t_SECTION(t):
75      r"\["
76      t.lexer.begin("SECTION")
77      return t
78
79  def t_SECTION_DATA(t):
80      r"[a-zA-Z\d\-\._]+"
81      t.lexer.begin("DATA")
82      return t
83
84  def t_DATA_END(t):
85      r"\]"
86      t.lexer.begin("INITIAL")
87      return t
```

## 3. Hash

La función que se encarga de transformar los datos de entrada a una estructura `hash`, en este caso un diccionario, es `parseIniFile(fullFile)`, donde `fullFile` es toda la entrada de texto.

El diccionario retornado tiene la siguiente estructura:

- Las llaves son los nombres de las secciones en tipo `string`.

  Si un identificador esta definido al comienzo del documento sin una sección, este sera almacenado dentro de la sección (string vació).

  Si todos los identificadores están dentro de alguna sección, la sección no existirá dentro del diccionario.

- Los valores son diccionarios que contienen los identificadores de la sección correspondiente.

  Esta estructurado de la siguiente forma:

  - La llave es el nombre del identificador. Tipo `string`.
  - El valor es el valor del identificador correspondiente.

    Puede ser de distintos tipos, siendo `bool` si se encontró un `true` o un `false`, `int` si se encontró únicamente un numero positivo, `None` si el identificador fue *seteado* sin un valor acompañante, o tipo `string` en caso contrario a los otros.

Si la definición de una función se encuentra repetida, se elimina la definición anterior y se usa la nueva. Lo mismo ocurre con los identificadores que se encuentren dentro de la misma sección.

Código 2: `parser_ini.py`

```
import sys

def check_cmd():
    if(len(sys.argv) == 1):
        print("Usage: tarea2.py file.ini")
        exit(1)
    return

def getFullFile():
    check_cmd()
    fullFile = ""
    with open(sys.argv[1]) as openedFile:
        fullFile = openedFile.read()
    return fullFile

def parseIniFile(fullFile):
    lexer = lex.lex()
    lexer.input(fullFile)

    lastKey = ""
    lastSection = ""
    hashed = {lastSection: dict()}
    for tok in lexer:
        if(tok.type == "KEY"):
            lastKey = tok.value
            if(lastKey not in hashed):
                hashed[lastSection][lastKey] = None

        if(tok.type == "VALUE"):
            hashed[lastSection][lastKey] = tok.value

        if(tok.type == "DATA"):
            lastSection = tok.value
            hashed[lastSection] = dict()
    if(len(hashed[""]) == 0):
        del hashed[""]
    return hashed
```