Cool Compiler - Tokenization

Disciplina: Compiladores

Docente: Carlos Bazílio Martins

Discente: Júlia Miranda Rodrigues

Identifiers

```python
from enum import Enum, auto
import re

class Ids(Enum):
    CLASS_ID            = auto(), "class"
    SEMICOLON_ID        = auto(), ";"
    QUOTE_ID            = auto(), '"'
    DOT_ID              = auto(), "."
    COMMA_ID            = auto(), ","
    COLON_ID            = auto(), ":"
    INHERITS_ID         = auto(), "inherits"
    IF_ID               = auto(), "if"
    ELSE_ID             = auto(), "else"
    FI_ID               = auto(), "fi"
    WHILE_ID            = auto(), "while"
    LOOP_ID             = auto(), "loop"
    POOL_ID             = auto(), "pool"
    LET_ID              = auto(), "let"
    IN_ID               = auto(), "in"
    CASE_ID             = auto(), "case"
    OF_ID               = auto(), "of"
    ESAC_ID             = auto(), "esac"
    NEW_ID              = auto(), "new"
    ISVOID_ID           = auto(), "isvoid"
    PLUS_ID             = auto(), "+"
    MINUS_ID            = auto(), "-"
    ASTERISK_ID         = auto(), "*"
    F_SLASH_ID          = auto(), "/"
    TIDE_ID             = auto(), "~"
    LESS_THAN_ID        = auto(), "<"
    LESS_THAN_EQUAL_TO_ID = auto(), "<="
    EQUAL_TO_ID         = auto(), "="
    NOT_ID              = auto(), "not"
    O_BRACKETS          = auto(), "{"
    C_BRACKETS          = auto(), "}"
    O_PARENTHESIS       = auto(), "("
    C_PARENTHESIS       = auto(), ")"
    ATT_ID              = auto(), "<-"
    ID_ID               = auto(), ">"
    TRUE_ID             = auto(), "true"
    FALSE_ID            = auto(), "false"
    STRING_ID           = auto(), "..."
    DIGITS              = auto(), "..."
```

Identifiers

```
45      @classmethod
46      def match(self, str='oi'):
47          if(str.isdigit()):
48              return self.DIGITS
49          if(str[0] =='"' and str[-1::] == '"'):
50              return self.STRING_ID
51          elif(str == "false"):
52              return self.FALSE_ID
53          elif("^[-+]?[0-9]+$" in str):
54              return self.DIGIT
55          elif(str == "true"):
56              return self.TRUE_ID
57          else:
58              for i in self:
59                  if (i.value[1] == str):
60                      return i
61              return self.ID_ID
```

Class Token

```python
from Id import Ids
import re

class Token():
    def __init__(self, str_, line):
        self.token = self.tiraT(str_)
        self.id = self.classify()
        self.line = line

    def __str__ (self):
        return f"{self.token} : {self.id.name}, line {self.line}\n"

    def tiraT(self, str):
        if(str!="\t"):
            return str.replace("\t","")
        else:
            return str

    def classify(self):
        if( self.token == "false" or self.token == "true"):
            return Ids.match(self.token)
        else:
            return Ids.match(self.token.lower())
```

Class Token

```python
from Id import Ids
import re

class Token():
    def __init__(self, str_, line):
        self.token = self.tiraT(str_)
        self.id = self.classify()
        self.line = line

    def __str__ (self):
        return f"{self.token} : {self.id.name}, line {self.line}\n"

    def tiraT(self, str):
        if(str!="\t"):
            return str.replace("\t","")
        else:
            return str

    def classify(self):
        if( self.token == "false" or self.token == "true"):
            return Ids.match(self.token)
        else:
            return Ids.match(self.token.lower())
```

Lexical Analyzer

```
102  def readNtokenize(fileName):
103      code = []
104      with open(fileName, 'r') as file:
105          program = file.readlines()
106
107          for num, line in enumerate(program, start=0):
108
109              line = spaceSymbols(line)
110              temp = splitLine(line)
111              temp = removeNone(temp)
112              temp = isolateString(temp)
113              temp = removeComments(temp)
114              if len(temp)!=0:
115                  code.append([class_.Token(str, num+1) for str in temp if str.strip()])
116
117          if(openComment):
118              raise Exception("Error! Unclosed block comment")
119      return code
120
121  def printTokens(tokensID):
122      for i in tokensID:
123          for j in i:
124              print(j)
```

Space Symbols - Split Line

```
5   spaceSymbols =  lambda line : line.replace("\n", " ").replace("@", " @ ").replace("*", " * ")\
6                   .replace(",", " , ").replace('"', ' " ').replace("(", " ( ").replace("/", " / ")\
7                   .replace(")"," ) ").replace(";", " ; ").replace("}", " } ")\
8                   .replace("{", " { ").replace(":", " : ").replace(".", " . ").replace("--"," -- ")
```

```
70    splitLine = lambda line: line.split(" ")
```

```
3    removeNone = lambda line : list(filter(None, line))
```

Catch Strings

```
72  stringOpen = False
73  def isolateString(line):
74      global stringOpen
75      newline = []
76      tempStr = ""
77
78      for token in line:
79
80          if(token != '"'):
81              if(not stringOpen):
82                  newline.append(token)
83              else:
84                  tempStr     = tempStr + " " + token
85          else:
86              if (not stringOpen):
87                  tempStr     = tempStr + token
88                  stringOpen  = True
89              else:
90                  if(stringOpen):
91                      stringOpen  = False
92                  tempStr     = tempStr + " " + token
93                  newline.append(tempStr)
94                  tempStr = ""
95      if(stringOpen):
96          if(tempStr[-1::]=="\\"):
97              newline.append(tempStr)
98          else:
99              raise Exception(r'Error!Missing one ". Did you mean "...\"? ')
100     return newline
```

Remove Comments (line and block)

```
68  removeComments = lambda line : removeLineComment(removeBlockComments(line))
```

```
19  searchAsterisk  = False
20  searchClose     = False
21  openComment     = False
22  def removeBlockComments(line):
23      global searchAsterisk
24      global searchClose
25      global openComment
26      newList = []
27      ant = ""
28
29      for token in line:
30          if (searchClose):
31              if(token != ")"):
32
33                  ant = token
34                  openComment = True
35              else:
36                  if(ant == "*"):
37
38
39                      searchClose = False
40                      openComment = False
41                  else:
42                      ant = token
```

```
43          elif(searchAsterisk):
44              if(token == "*"):
45
46                  searchClose     = True
47                  searchAsterisk  = False
48              else:
49                  if(ant=="("):
50                      newList.append("(")
51                  newList.append(token)
52                  searchAsterisk  = False
53          else:
54              if(token=="("):
55                  searchAsterisk = True
56                  ant = token
57              else:
58                  newList.append(token)
59
60      if(searchClose):
61          if (openComment):
62              pass
63          else:
64              openComment = True
65
66      return newList
```

```
10  removeLineComment = lambda line : line.partition('--')[0]
```

Main

```
1   import lexicalAnalyzer as LA
2   import sys
3
4   fileName    =  sys.argv[1]
5
6   tokens = LA.readNtokenize(fileName)
7
8   LA.printTokens(tokens)
```