

Cool Compiler - Tokenization

Disciplina: Compiladores

Docente: Carlos Bazílio Martins

Discente: Júlia Miranda Rodrigues

Identifiers

```
1  from enum import Enum, auto
2  import re
3
4  class Ids(Enum):
5      CLASS_ID          = auto(), "class"
6      SEMICOLON_ID      = auto(), ";"
7      QUOTE_ID          = auto(), '"'
8      DOT_ID            = auto(), "."
9      COMMA_ID          = auto(), ","
10     COLON_ID           = auto(), ":"
11     INHERITS_ID        = auto(), "inherits"
12     IF_ID              = auto(), "if"
13     ELSE_ID            = auto(), "else"
14     FI_ID              = auto(), "fi"
15     WHILE_ID           = auto(), "while"
16     LOOP_ID            = auto(), "loop"
17     POOL_ID            = auto(), "pool"
18     LET_ID             = auto(), "let"
19     IN_ID              = auto(), "in"
20     CASE_ID            = auto(), "case"
21     OF_ID              = auto(), "of"
```

```
22     ESAC_ID           = auto(), "esac"
23     NEW_ID             = auto(), "new"
24     ISVOID_ID          = auto(), "isvoid"
25     PLUS_ID            = auto(), "+"
26     MINUS_ID           = auto(), "-"
27     ASTERISK_ID        = auto(), "*"
28     F_SLASH_ID         = auto(), "/"
29     TIDE_ID            = auto(), "~"
30     LESS_THAN_ID       = auto(), "<"
31     LESS_THAN_EQUAL_TO_ID = auto(), "<="
32     EQUAL_TO_ID        = auto(), "="
33     NOT_ID              = auto(), "not"
34     O_BRACKETS         = auto(), "{"
35     C_BRACKETS         = auto(), "}"
36     O_PARENTHESIS      = auto(), "("
37     C_PARENTHESIS      = auto(), ")"
38     ATT_ID             = auto(), "<-"
39     ID_ID              = auto(), ">"
40     TRUE_ID            = auto(), "true"
41     FALSE_ID           = auto(), "false"
42     STRING_ID          = auto(), "..."
43     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

```
1  from Id import Ids
2  import re
3
4  class Token():
5      def __init__(self, str_, line):
6          self.token = self.tiraT(str_)
7          self.id = self.classify()
8          self.line = line
9
10     def __str__(self):
11         return f"{self.token} : {self.id.name}, line {self.line}\n"
12
13     def tiraT(self, str):
14         if(str!="\t"):
15             return str.replace("\t","")
16         else:
17             return str
18
19     def classify(self):
20         if( self.token == "false" or self.token == "true"):
21             return Ids.match(self.token)
22         else:
23             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("*", " * ")\n6     .replace(",", " , ").replace("'", " ' ").replace("(", " ( ").replace("/", " / ")\n7     .replace(")", " ) ").replace(";", " ; ").replace("}", " } ")\n8     .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
44     elif(searchAsterisk):
45         if(token == "**"):
46
47             searchClose    = True
48             searchAsterisk = False
49         else:
50             if(ant=="("):
51                 newList.append("(")
52                 newList.append(token)
53                 searchAsterisk = False
54             else:
55                 if(token=="("):
56                     searchAsterisk = True
57                     ant = token
58                 else:
59                     newList.append(token)
60
61     if(searchClose):
62         if (openComment):
63             pass
64         else:
65             openComment = True
66
67     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)
```