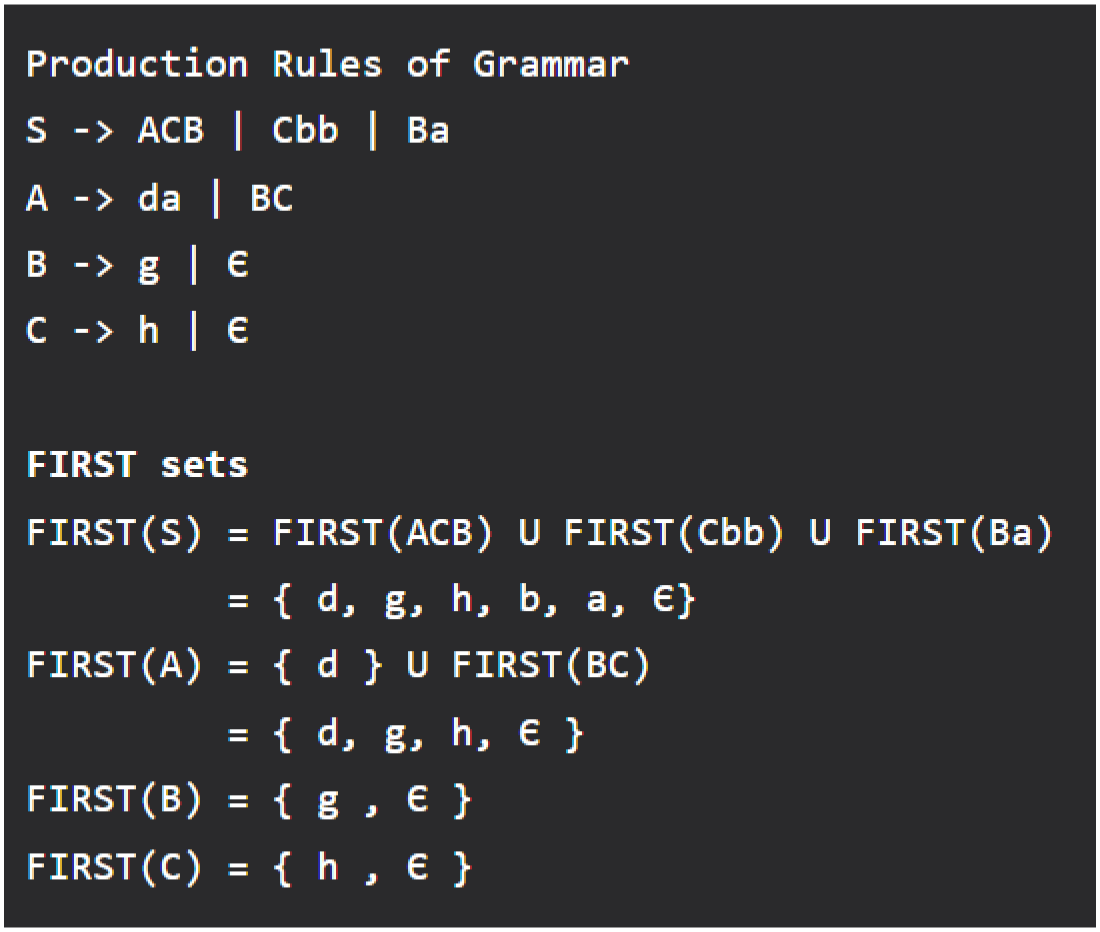
Github link:

→https://github.com/Badetto/FLCD_Lab2 (In the folder "Lab_Teams")

Parser:

Description: The first step into creating a Parser algorithm with the parsing method LL(1) is creating the methods: FIRST and FOLLOW. The second step into creating a Parser algorithm with the parsing method LL(1) is creating the parsingTable.

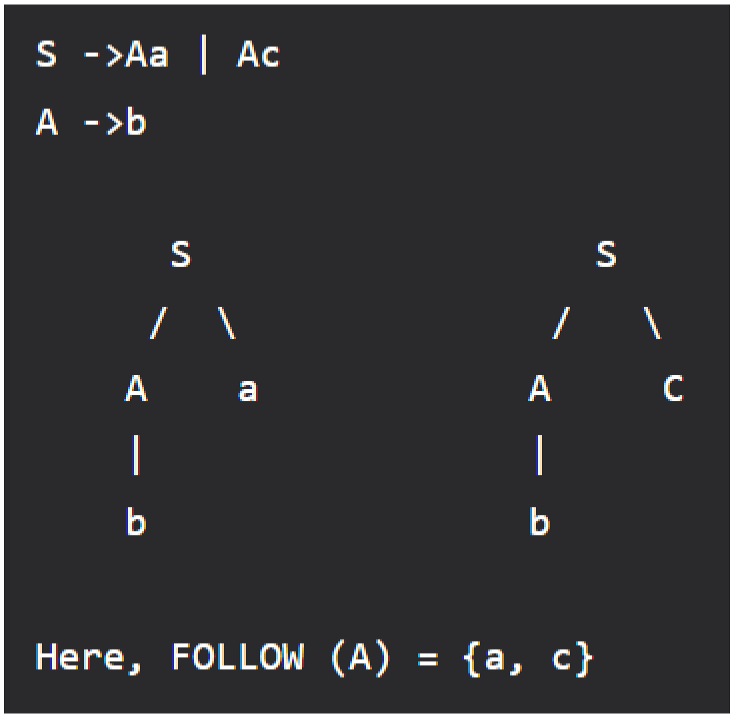
→**FIRST(symbol: string):HashSet<string>:** It calculates the FIRST set for a given symbol in a context-free grammar, which includes all possible terminals that can appear at the start of strings derived from that symbol.



```
Production Rules of Grammar
S -> ACB | Cbb | Ba
A -> da | BC
B -> g | Є
C -> h | Є

FIRST sets
FIRST(S) = FIRST(ACB) U FIRST(Cbb) U FIRST(Ba)
         = { d, g, h, b, a, Є}
FIRST(A) = { d } U FIRST(BC)
         = { d, g, h, Є }
FIRST(B) = { g , Є }
FIRST(C) = { h , Є }
```

→**FOLLOW(nonTerminal: string, inProgress:HashSet<string> | null):HashSet<string>:** It computes the FOLLOW set for a given non-terminal in a context-free grammar, which includes all terminals that can appear immediately after the non-terminal in any derivation from the start symbol.



```
S ->Aa | Ac
A ->b

     S                S
    / \              /  \
   A   a            A    C
   |                |
   b                b

Here, FOLLOW (A) = {a, c}
```

→**InitializeParsingTable( ):void:** First it initialize the parsing table (each nonTerminal-terminal pair + nonTerminal-$ pair) with an empty list. Then it goes thourgh each production inside the grammar and systematically populates the table. For each production, it computes the FIRST set of its symbols. If this set doesn't contain 'epsilon', the production is added to the table entries corresponding to these FIRST set symbols. In cases where 'epsilon' is found, the production is added to the entries for each symbol in the FOLLOW set of the non-terminal. Finally, the function checks for and reports any conflicts, indicating ambiguities where a single table entry contains multiple productions. These conflicts must be resolved for the grammar to be LL(1) compatible.

→**PrintParsingTable( ):void:** It prints the parsing table (each nonTerminal-terminal pair + nonTerminal-$ pair), it should be calles after the function InitialParsingTable was called.