Documentation

GitHub: https://github.com/917tapoimarius/LFTC_Parser

Team members:

Taravinas Iannis

Tapoi Marius-Stefan

Class Grammar: A class representing the grammar of the programming language:

Attributes:

nonTerminals: List<String> – a list of Strings representing the nonterminals of the grammar.

terminals: List<String> - a list of Strings representing the terminals of the grammar.

startSymbol: String – a String which represents the starting symbol of the grammar.

productions: Map<String, List<List<String>>> – a table like representation of the productions of the grammar; the key is represented by a nonterminal, and for each key, the value is represented by a list of lists, so that a nonterminal can be represented in multiple ways by one or more symbols.

Methods:

Grammar (): Constructor of the class; initializes the nonTerminals List as an ArrayList, the terminals List as an ArrayList and the productions table as a HashMap.

params: -

returns: -

readFile (String filename): Method used to read through each line of the input file. The first line represents the nonterminals. When it is read, the strings are stored in the *nonTerminals* list. The second line represents the terminals which are stored in the *terminals* list. The third line represents the *startSymbol*. The following lines represent the *productions.* Each line is split into a left side and a right side, separated by an arrow. The left side represents a nonterminal, while on the right side there can be another nonterminal, a terminal, or a list of nonterminals. If the left side already exists in the productions list as a key, then the right side is added at the key's address, otherwise the left side is stored as a key, and the right side as an ArrayList at the key's address.

params:

filename: String – the location of the file from which the data is read.

returns: -

checkCFG(): Method used to check if the read grammar is a context free grammar. At first, it checks if all the keys (represented by nonterminals) are stored in the productions table. If one is missing, then the grammar is not context free. Then, it checks all the values from the table. It checks every element from each production list and if the one of the elements is not found in the nonterminals or terminals lists then the grammar is not context free. If these two checks pass, then the grammar is context free.

params: -

returns: true if the grammar is context free or false otherwise.

printNonTerminals(): Method used to print the nonterminals list.

params: -

returns: -

printTerminals(): Method used to print the terminals list.

params: -

returns: -

printProductions(): Method used to print the productions list in a structured manner. Each key (nonterminal) is followed by an arrow and the list of all possible representations of the nonterminal, each representation being separated by the '|' character.

printStartSymbol(): Method used to print the start symbol.

params: -

returns: -

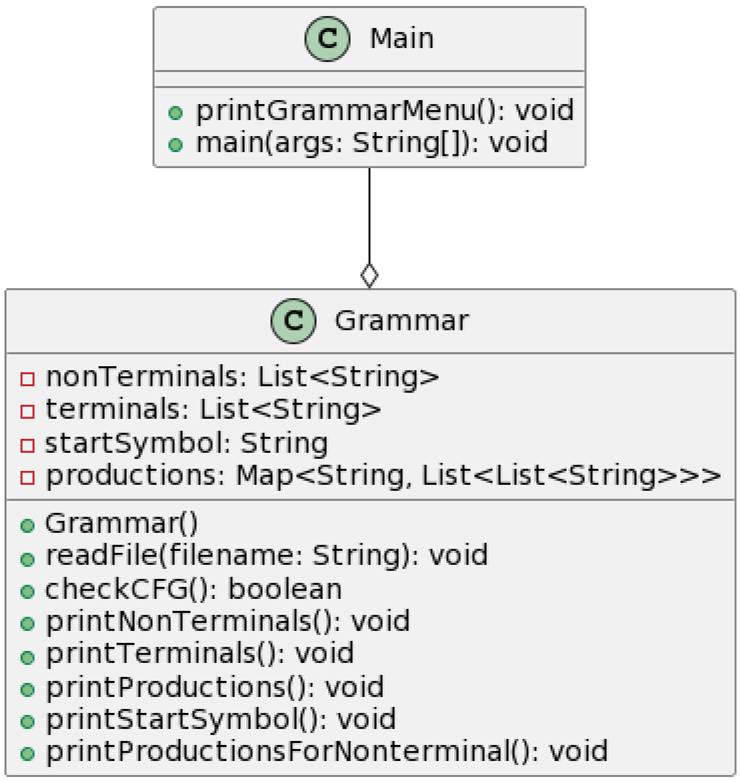
printProductionsForNonterminal(String nonTerminal): Method used to print all the productions for a given nonterminal. The nonterminal is printed (key), followed by the production (value), separated by an arrow. If the production is represented by multiple lists, the key is printed with each of its productions.

params:

nonTerminal: String – given string representing a nonterminal for which all the productions are printed.

returns: -

Class Diagram:


Main
printGrammarMenu(): void
main(args: String[]): void
Grammar
nonTerminals: List<String>
terminals: List<String>
startSymbol: String
productions: Map<String, List<List<String>>>
Grammar()
readFile(filename: String): void
checkCFG(): boolean
printNonTerminals(): void
printTerminals(): void
printProductions(): void
printStartSymbol(): void
printProductionsForNonterminal(): void