Documentation for Lab5

LINK TO GIT: https://github.com/radutalaviniaelena/FLCD

REQUIREMENT:

Statement: Implement a parser algorithm

1. One of the following parsing methods will be chosen (assigned by teaching staff):

   1.a. recursive descendent

   1.b. ll(1)

   1.c. lr(0)

2. The representation of the parsing tree (output) will be (decided by the team):

   2.a. productions string (max grade = 8.5)

   2.b. derivations string (max grade = 9)

   2.c. table (using father and sibling relation) (max grade = 10)

PART 1: Deliverables

1. *Class Grammar* (required operations: read a grammar from file, print set of nonterminals, set of terminals, set of productions, productions for a given nonterminal, CFG check)
2. Input files: *g1.txt* (simple grammar from course/seminar), *g2.txt* (grammar of the minilanguage - syntax rules from Lab 1b)

For implementing the laboratory, I have the following:

1) **Grammar** class – contains all fields and methods necessary for working with a grammar

   I. **Fields**:

      1. **private final Set<String> terminals** – the set of all terminals;
      2. **private final Set<String> nonTerminals** – the set of all nonterminals;
      3. **private final Map<String, Set<String>> productions** – for each nonterminal we keep a set of all its productions;
      4. **private String startSymbol**;
      5. **private Boolean isCFG** – this variable is false if the grammar is not a context free grammar and true otherwise;

II. **Methods**:

1. /**
   * This function reads from a given file all the elements of a grammar: terminals, nonterminals, productions and start symbol. Before reading such a sequence (e.g.: the sequence of terminals), we read the number of items in that sequence (for this we use the function **private int getNumber(BufferedReader reader, int number)**). At the same time, for each production read we check if the grammar is CFG up to that moment or not (storing the result in the isCFG variable).
   * **@param** filePath: a string representing the location of the file
   */
   public void readGrammarFromFile(String filePath)

2. /**
   * This function returns the set of productions for a given nonterminal.
   * **@param** nonterminal: a String value representing the nonterminal
   * **@return:** the set of all productions for the given nonterminal
   */
   public Set<String> getProductionForNonTerminal(String nonterminal)

3. **Two print functions** which displays both the set of all productions and the set of productions for a given nonterminal in a proper manner.

2) **Main** class – the **main function** displays a menu from which you can choose to access any element of the grammar (the set of terminals/nonterminals/productions/productions for a given nonterminal, start symbol) or to check if the grammar is CFG.

The EBNF of the input files (g1.txt, g2.txt):

nz_digit := “1” | ”2” | .. | ”9”
digit := “0” | ”1” | ”2” | .. | ”9”
number := nz_digit {digit}

letter := “a” | “b” | .. | “z” | “A” | “B” | .. | “Z”
character := letter | digit
string := character {character}

first_line := number (* it represents the number of nonterminals *)
second_line := {string} (* it represents the nonterminals *)
third_line := number (* it represents the number of terminals *)
fourth_line := {string} (* it represents the terminals *)
fifth_line := number (* it represents the number of productions *)
sixth_line := {string “->” string} (* it represents the productions *)
seventh_line := string (* it represents the start symbol *)

inputFile := first_line second_line third_line fourth_line fifth_line sixth_line seventh_line