**Al-Maaref University**

**Faculty of Sciences**

CSC 340 Programming Project Report

Grammar Analyzer

**Students**

Zeinab Kassir-10120191

**Instructor Name**

Fatima Abdallah

01/01/2024

Our project focused on creating a program that reads a grammar and a string and then determines if the grammar can generate that string. To scope it up, we focused on specific grammar. the following ones that we will deal with:

* A = {an#bn| n > 0}
* B = {w#wR | w ∈ {0, 1}∗}
* C = {ai#bj#ck|j = k and i, j, k > 0}
* D= {w# | w ∈ {0, 1}∗}

# **Encoding the Grammar**

In this part, we convert the above grammar into a set of rules stored in a text file for later use. We encode each grammar separated by “;” and each one is composed of a set of rules. Each rule is written on a separate line composed of the left side and right side separated by the “ -> “. variables are located on the LS rule denoted by capital letters and RS rule is composed of a list, separated by ”|”, of combinations of variables and symbols; Grammars A, B, C, and D are represented as follows in the rule.txt file respectively.

# **Structure of the project**

* Reader Class: Reads grammars from a file (rules.txt) and stores them in a list. Contains a method readfile() that returns a list of grammar objects.
* Main Class: Using the Reader class read grammars from the file. Provides a menu for the user to choose a grammar (A, B, C, D) or exit (E). For a chosen grammar, it allows the user to enter strings and check if they are accepted by the grammar by using the parser method.
* grammar Class: Represents grammar with a list of rules. Provides methods to add rules, check if a symbol is a variable, and find a matching rule for a given variable and input symbol.as well as, a method parseString() that checks if the grammar accepts a given string.
* rule Class: Represents a production rule in a grammar. Contains a left-hand side (lS) and a list of right-hand side options (RS). Provides methods to get and set the left-hand and right-hand sides, and a toString() method for easy printing. created a class grammar composed of a list of rules (type class rule), method add rule() to add to the list is a variable method to check if the to store each grammar in it and rule class (to avoid multiple file reading)

**Reading the Grammar**

To read the grammar from the text file we use a buffer in the read file method that uses a buffer reader to read from the rules.txt by looping on each line, skipping empty ones and creating rules from non-empty ones separating those on the right side (as a list separated split on “|”) and left side as well as splitting finalising the grammar and storing it in the list when “;” line is detected. When all lines are read in the file the list of grammars is returned.

**Parsing the String**

The parsing process began by pushing the start symbol onto the stack. The parser then entered a loop where it continually popped an element from the stack and checked if it was a terminal or a non-terminal. If it's a terminal, it checks if it matches the next character in the input. If it's a non-terminal, it looks at the next character in the input and uses that to determine which rule to apply. It then pushes the right-hand side of that rule onto the stack. If no rule matches the next input, the string is rejected. If the stack is empty and the input is fully consumed, the string is accepted. Otherwise, it is rejected.

**Testing**

To test the validity of the program several accepted and rejected strings are taken.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| Accepted | a#b  aa#bb | #  0#0 | a#b#c  a#bb#cc | #  1010# |
| Rejected | aab#bb  # | 010#011  #01 | #b#c  ax#b#  a#bcc#b  ## | 00#1 |

# **Analysis**

1. The two conditions set as grammar criteria are placed to ensure that when parsing the string and analyzing the current symbol, you can easily identify the corresponding right-side rule to apply. Since the first symbol on the right side is always terminal, it directly matches the current input symbol
2. If those weren't settled multiple scenarios must be taken into consideration such as the first symbol on the right-hand side could be either a terminal or a non-terminal which complicates the process of finding the right match.
3. Any ambiguous grammar will lead to different meanings or interpretations for a given input and lead to difficulties in choosing the correct derivation path.
4. Multiple approaches can be used other than that taken in this project such as pumping lemma and semantic analysis.