Lexical-Analyzer-Generator

(Compiler)

Project Report

Mohamed Essam (64)

Mohamed Murad(66)

Mohamed Raafat (62)

Wessam Mohamed (81)

Table of contents

[1] Used data structure](#_Toc509262469)

2] algorithms and techniques used…………………………………………………………………….

3] The resultant PARSING TABLE OF CONTEXT FREE GRAMMAR…………………………...

4] EXPLANATION OF USED FUNCTION………………………………………………………………………....

**5] ANY ASSUMPTIONS MADE AND THEIR JUSTIFICATION……………………………………………**

**1] Used data structure**

1] Rules Parser:

2] Infix to Postfix Handler:

3] NFA Builder:

4] DFA Builder:

5] First:

Holds the value of the first.

6] Follow:

Holds the value of the follow.

7] Production:

Holds the production of the non-terminal as vector of string each element in this vector is an element in this production, ex: E -> ‘+’ T E` | P F then this is a non-terminal E that has 2 productions ‘+’ T E` and P F , then there would be 2 production objects each of which held one production and save in form of elements. Then the result would be: P1 = {‘+’, T, E`} and P2 = {P, F}

8] First\_Production\_Pair:

Holds the non\_terminal, its first and its corresponding production, that would be substituted in case we found that first in the file while parsing.

9] Non\_Terminal\_Info:

We made this data structure that holds the info of each non-terminal which are: its value, Its First and corresponding production, and Its follow. In this Data structure we used another data structure such as First\_Production\_Pair, Production, First, and Follow that are mentioned above.

10] Parsing\_Table:

Holds an unordered\_map of an unordered\_map of productions, the first map holds the non-terminal which provide us quick access, the next map holds the pair of first of this non-terminal which points to production object that holds the correct production would be substituted in case we found the correct first of the current non-terminal.

**2] algorithms and techniques used**

1] //Raffat:

First (Infix to Postfix Converter) *"According to Geeks for Geeks"*:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,  
   …3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack (or the stack is empty), push it.  
   …3.2 Else, Pop the operator from the stack until the precedence of the scanned operator is less-equal to the precedence of the operator residing on the top of the stack. Push the scanned operator to the stack.
4. If the scanned character is an ‘(‘, push it to the stack.
5. If the scanned character is an ‘)’, pop and output from the stack until an ‘(‘ is encountered.
6. Repeat steps 2-6 until infix expression is scanned.
7. Pop and output from the stack until it is not empty.

Second (Postfix Expression Evaluator) *"According to Geeks for Geeks"*:

1. Create a stack to store operands (or values).
2. Scan the given expression and do following for every scanned element.  
   …2.1 If the element is an Operand, push it into the stack  
   …2.2 If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
3. When the expression is ended, the value in the stack is the final answer.

2] //Murad:

3] Parsing\_Table\_Generator:

We start iterating on each element in the Non\_Terminal\_Info vector and start building the parsing table if we found lambda as first of any non-terminal we put the follow too in the table with lambda as production, else we put and empty production with non-terminal value called sync that indicates that is a sync entry of the table. Else of those cases we don’t insert any entry.

4] Scanner:

This object was used last time to scan the file and collect the tokens, now we made him print the left most derivation of the grammar while collecting the tokens by adding a method called match token, that take the token, parser stack and the parsing table as input, and output the effect of this token taken from the file.

This method matches the token by popping the non-terminals of the stack that has this token as a first and put its production instead until we find the matching token then we pop it from the stack and return. In case we found any kind of error we print error message on the terminal and continue parsing the file.

**3] The resultant PARSING TABLE OF CONTEXT FREE GRAMMAR**

**// ToDo**

**4] EXPLANATION OF USED FUNCTION**

1] Raffat

2] Murad

3] Generate\_Table:

This method is used in the Parsing\_table\_generator, that is used to generate the parsing table of the input grammar file, by iterating over the data structure and continue filling the table as described above in the data structure.

4] match\_token:

Used to match the token given from the lexical analyzer using the parsing stack and the parsing table that give us the corresponding production to this token, if an error found during matching an error message prints on the terminal, that define the error type and the token and the non-terminal that caused this error.

5] split\_line:

Used for the first part of the project that take the grammar file path, then open the file and start to get the rules from the file and splitting it to non-terminals and productions.

6] package\_non\_terminal:

Used to convert the first and follow maps to vector of Non\_Terminal\_Info data structure to be used to construct the table.

7] grammer\_table:

Used to generate the parsing table from the vector of the non\_Terminal\_info by calling the Parsing\_Table\_Generator that generates the table.

**5] ANY ASSUMPTIONS MADE AND THEIR JUSTIFICATION**

We assumed that the keywords defined first then punctuation, then definitions then expressions which is not used in the LEX they define the ordering and the priority of the tokens by the order of the reading we made the 2 implementations, but we are using ours assumption as it’s logical that the keywords token would have higher priority than the expressions.

We assumed that we will not use the #, ?,% and the @ as we are using them in evaluating the regular expression.

**6] BONUS – BUILDING LEXICAL ANALYZER USING FLEX**