CSE 351- PROGRAMMING LANGUAGES TERM PROJECT REPORT

AIM: In the project, our aim was to convert a given grammar into a recursive-descent parser code. There were four different semantic checks that my code have to perform.
I wrote a lex, yacc and Makefile codes to achieve the aim. Which are added in the zip file along with this report.

SEMANTIC ANALYSIS

Analysis 1:

In this first analysis, I have checked if there is an inconsistency between this rule definition line and the given grammar. As it is seen in the example below, the rule definition line claims that there are three rules of the non-terminal "E", but in the grammar, there are only two rules. In this case, my code prints the error message **The number of rules for E non-terminal is 2 but in the rule definition line, it is 3[Error].**

```
%rules E 3 T 1;
E -> T
| T + E;
T -> int
```

Analysis 2:

The line "%rules E 2 T 3;" in the input also means there are grammar rules with E and T non-terminals. I have given an error message if there is no grammar rule with E or T. **T does not have grammar rule[Error2].**

While searching for semantic errors if my code encounter with a semantic error it immediately prints the error message and exits. In this case, it can not print the all semantic error message. It prints the one which encountered the first so my code cannot say if there is no grammar rule with E AND T.

On the contrary, if a there is a grammar rule for non-terminal X but it is not defined in the rule definition line, again, In this case, my code prints the error message **There is grammar rule for non-terminal T but it is not defined in the rule definition line [Error1].**

```
Error(1)                    Error(2)
%rules E 3;                   %rules E 3 T 1;
E -> T                          E -> T
| T + E;                       | T + E;
T -> int;
```

Analysis 3:

I have given an error message if a non-terminal is referenced but it is not defined as a rule. As it is seen in the example, non-terminal T is referenced in the second rule of non-terminal E, but it does not have any dedicated grammar rule. **E does not have any dedicated grammar rule [Error].**

Error example

```
%rules E 3;
E -> T
| T + E;
```

Analysis 4:

I have given an error message if there is a useless rule. As it is seen in the example below, non-terminals F and G are never referenced in the right-hand side of any grammar rule, and, thus, they are useless. My code prints the error message **F never referenced in the right-hand side of any grammar rule, thus useless rule[Error1].**
Error example

```
%rules E 3;
E -> T
| T + E;
T -> int;
F -> E + G
G -> int
```

My code reads the input.txt which contains the grammar.The grammar %rules definition order and the given order of rules for each non-terminal does not have to be in the same order. First, it checks if there is a syntax error. If there is it gives an error which shows the syntax error line. If not it checks if there is a semantic errors which are given above. The code semantic analysis order is, analysis2, analysis1, analysis3, and analysis 4.So if there are multiple semantic errors the code will only give the first encounter semantic error message. After the user corrects the first semantic error it will continue on the others until it is none. If there is no syntax and semantic errors the code will not print any output on the terminal. It will write an output.c file according to given input grammar.My code can produce a working C code. Only it request the **TOKEN input[50] = ... l**ine manually copy paste. After that, it rejects the inputs which are invalid compared to the grammar and accepts the inputs which are valid compared to the grammar.I have tried the valid/invalid inputs that our teacher provided us and I got the expected output.

Finally, as much I see my code prevents the required statements of this project. The only problem is that I added "." token to my lex code which prevents printing the none recognize tokens. For example

```
%rules E 2 T 1;
E -> T input
| T + E;
T -> int
```

This is a valid grammar except for the lowercase "input", I excepted that this will give a syntax error according to my lex and yacc but it does not. Without the "." token in lex it just prints the "input" text on the terminal. This is the only problem that I found out in my project.

Aleyna Buse Hışım 20160702072