دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

درس کامپایلر

تکلیف عملی دوم

تاریخ تحویل: ۱۵ اردیبهشت ۱۴۰۲

You have to upload all of your works to the Quera platform, so join the class's Quera webpage ASAP (the password is Compiler_140102!)

Question 1: Designing a Grammar for Recursive Descent Parsing

In this programming question, you are required to write a Python program that takes a given context-free grammar (CFG) as input and transforms it into a grammar that can be parsed by a recursive descent parser. The input grammar will be provided as a dictionary, where the keys represent non-terminal symbols, and the values are lists of possible productions for each non-terminal.

Your program should perform the following tasks:

1. Eliminate left recursion from the input grammar.
2. Perform left factoring on the resulting grammar.

In the end, your program should output the transformed grammar in the same dictionary format as the input grammar.

Input:

- A dictionary input_grammar, where the keys are non-terminal symbols (strings) and the values are lists of productions (lists of strings).

Output:

- A dictionary output_grammar, where the keys are non-terminal symbols (strings) and the values are lists of productions (lists of strings) for a grammar that can be parsed by a recursive descent parser.

Constraints:

- The non-terminal symbols will consist of uppercase alphabets (A-Z).
- The terminal symbols will consist of lowercase alphabets (a-z), digits (0-9), and special characters (e.g., +, -, *, /).
- The input grammar will have at most 20 non-terminal symbols and 50 productions in total.
- The length of each production will not exceed 100 characters.

Example:

Suppose the input grammar is as follows:

```
{
   'E': ['E+T', 'T'],
   'T': ['T*F', 'F'],
   'F': ['(E)', 'id']
}
```

Your program should output the following transformed grammar:

```
{
   'E': ['T', 'E`'],
   'E`': ['+TE`', ''],
   'T': ['F', 'T`'],
   'T`': ['*FT`', ''],
   'F': ['(E)', 'id']
}
```

Grading Policy:

- If your code only removes left recursion from the input grammar, you will receive 30 points. (1)
- If your code only performs left factoring on the resulting grammar, you will receive 30 points. (2)
- If your code completes both of these tasks at the same time and correctly outputs the transformed grammar, you will receive 100 points! (3)

- Note that only **one of these conditions** will be valid for grading your code. The combination of these conditions may not result in a score greater than 100
- Kindly note that your submission for this question will be evaluated based on a maximum 5-minute video that you must provide.
    - At the start of the video, please introduce yourself and
    - indicate the type of answer you are providing (1,2,3).
    - Additionally, you must provide a detailed explanation of the code you have written,
    - run precisely three test cases for the question, and
    - evaluate the results.
- It is important to note that your grade will be based solely on the video you submit and not on the mandatory Python code.

Question 2: Drawing the Parse Tree for a Recursive Descent Parser-Compatible Grammar

In this question, you are asked to write a Python program that takes a grammar that can be parsed by a recursive descent parser and an input string. The program should then draw the parse tree for the given string using the provided grammar.

The input grammar will be provided as a dictionary, where the keys represent non-terminal symbols, and the values are lists of possible productions for each non-terminal.

Input:

- A dictionary input_grammar, where the keys are non-terminal symbols (strings) and the values are lists of productions (lists of strings) for a grammar that can be parsed by a recursive descent parser.
- A string input_string, which is the input string to be parsed.

Output:

- A graphical representation of the parse tree can be saved as an image file (e.g., PNG, JPEG).

Constraints:

- The non-terminal symbols will consist of uppercase alphabets (A-Z).
- The terminal symbols will consist of lowercase alphabets (a-z), digits (0-9), and special characters (e.g., +, -, *, /).
- The input grammar will have at most 20 non-terminal symbols and 50 productions in total.
- The length of each production will not exceed 100 characters.
- The length of the input string will not exceed 100 characters.

Example:

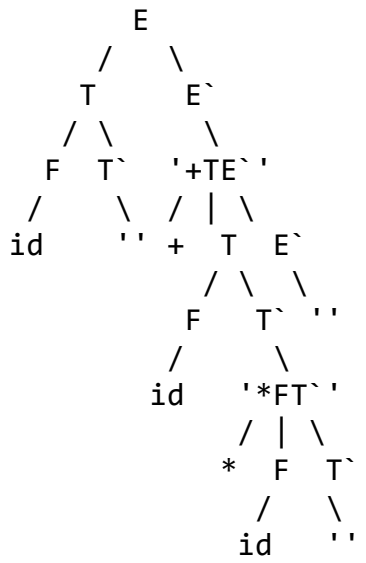
Suppose the input grammar is as follows:

```
{
   'E': ['T', 'E`'],
   'E`': ['+TE`', ''],
   'T': ['F', 'T`'],
   'T`': ['*FT`', ''],
   'F': ['(E)', 'id']
}
```

And the input string is:

id+id*id

Your program should output a parse tree representation of the input string using the given grammar.
In this case, the parse tree should be:



Grading Policy:

- If your code only generates the First and Follow sets of the given grammar, you will receive 20 points. (1)
- If your code only constructs the LL(1) table for the given grammar, you will receive 30 points. (2)
- If your code only serves as a parser for the input grammar and outputs the derivations (not in the form of a tree), you will receive 70 points. (3)
- If your code correctly prints out the parse tree of the grammar in the terminal as described in the question, you will receive 90 points. (4)
- If your code correctly generates and saves the parse tree of the grammar as a PNG or JPEG file, you will receive 100 points. (5)

- Note that only **one of these conditions** will be valid for grading your code. The combination of these conditions may not result in a score greater than 100.
- Kindly note that your submission for this question will be evaluated based on a maximum 5-minute video that you must provide.
    - At the start of the video, please introduce yourself and
    - indicate the type of answer you are providing (1, 2, etc.).
    - Additionally, you must provide a detailed explanation of the code you have written,
    - run precisely three test cases for the question, and
    - evaluate the results.
- It is important to note that your grade will be based solely on the video you submit and not on the mandatory Python code.