National University of Computer and Emerging Sciences
Fast School of Computing Spring 2025

CS-4031-Compiler Construction
Assignment #02

Instructions

1. The following assignment must be completed in the same pairs as Assignment 1.
2. The assignment must be attempted in **C programming language**.
3. Your code must be generic and reusable.
4. Usage of built-in functions or libraries related to compiler construction is **not allowed**.
5. Ensure proper indentation and comments in the code in addition to a presentable output of the program.
6. Plagiarism in any form (copying from others, copying from the internet, etc.) is strictly prohibited. If your work is found to be plagiarized, you will be awarded zero marks for the assignment.

Submission Guidelines

1. Combine all your work into one folder and name it as follows: **RollNumber1-RollNumber2-Section.zip** (e.g., `22i1234-22i5678-A.zip`).
2. Submit the .ZIP file on Google Classroom before the deadline. Submissions through other means (e.g. email) will not be accepted.

Assignment Overview

In this assignment, you are required to design and implement a C program that processes a Context-Free Grammar (CFG) provided as input from a file. Your program must perform the following steps:

1. **Left Factoring:** Transform the grammar to remove any common prefixes among the productions. Left factoring is a transformation used to rewrite grammars in order to prepare them for predictive (LL(1)) parsing. For example, given a production such as

   $$A \to \alpha\beta_1 \mid \alpha\beta_2,$$

   it should be rewritten as:

   $$A \to \alpha A', \quad A' \to \beta_1 \mid \beta_2.$$

2. **Left Recursion Removal:** Eliminate left recursion from the grammar to avoid infinite recursion during top-down parsing. For a production of the form:

   $$A \to A\alpha \mid \beta,$$

   it should be rewritten as:

   $$A \to \beta A', \quad A' \to \alpha A' \mid \varepsilon.$$

3. **First Set Computation:** Compute the `FIRST` set for each non-terminal in the grammar. The `FIRST` set of a symbol is the set of terminals that begin the strings derivable from the symbol.

4. **Follow Set Computation:** Compute the `FOLLOW` set for each non-terminal. The `FOLLOW` set contains terminals that can appear immediately to the right of the non-terminal in some "sentential" form.

5. **LL(1) Parsing Table Construction:** Using the computed `FIRST` and `FOLLOW` sets, construct the LL(1) parsing table.

Program Requirements

1. Your program must read a generic CFG from a file. The input file should contain productions in a simple format (e.g., one production per line, using a specific delimiter for the arrow such as `->`).

2. Perform left factoring on the CFG to remove non-determinism, if detected.

3. Process the resultant CFG from step 2 for left recursion removal.

4. Compute the First & Follow sets for each non-terminal in the CFG.

5. Construct a LL(1) parse table using the processed CFG, First and Follow sets.

6. Display the result obtained after completion of each of the steps mentioned above.
7. Ensure that your program is well modularized, with clear functions for each of the above tasks.
8. Use appropriate data structures to represent the grammar, the sets, and the parsing table.

Example CFG

Consider the following generic CFG as an example:

```
E -> E + T | T
T -> T * F | F
F -> ( E ) | id
```

- **Left Factoring:** If there is a production like `A -> aB | aC`, then it should be factored as:

  ```
  A -> aA'
  A' -> B | C
  ```

- **Left Recursion Removal:** In the given example, productions such as `E -> E + T` and `T -> T * F` exhibit left recursion and need to be restructured.
- **First and Follow Sets:** Calculate the sets for each non-terminal. For instance, the `FIRST` set of `F` might be { '(', id } and the corresponding `FOLLOW` set would depend on the context in which `F` appears.
- **LL(1) Parsing Table:** Build the table using the computed sets, ensuring that for each non-terminal and terminal pair, the correct production is indicated.

Deliverables

1. Source code files in C.
2. A sample input file containing the CFG.
3. A text file or output log showing the output at each stage of the processing:
   - CFG after left factoring.
   - CFG after left recursion removal.
   - `FIRST` sets for all non-terminals.
   - `FOLLOW` sets for all non-terminals.

- LL(1) parsing table.

4. A brief report (in PDF format) describing your approach, any challenges faced, and how you verified the correctness of your program.

Good luck!