Assignment #2

Date Due: March 19, 2026

Total: 100 marks

INSTRUCTIONS

- Submit all program code and any relevant output from program testing.
- Combine all your files into a single compressed file. Please use a filename which includes your UPEI username and is of the form: username_ast2.tar.gz; submit the targzip file via moodle.

  Each program includes beside the source code an executable Readme file which will run all examples automatically and a run.txt file containing the demo of the execution, for each test.

Requirements

Step 1

Assume your grammars have $V_N = \{A, B, \ldots, Z\}$ and $V_T = \{a, b, c, \ldots, z\}$. Productions $A \to \alpha$, of a given grammar are stored in a file as
$A\ \alpha$

The task is to develop the following tools:

1. (10 marks) Write programs/methods/functions to compute $FIRST(\alpha)$, for a given string $\alpha \in (V_n \cup V_T)^*$.
2. (10 marks) Write programs/methods/functions to compute $FOLLOW(A)$, for any nonterminal of the grammar.
3. (10 marks) Write programs/methods/functions to compute $LEFT(A)$, for any nonterminal of the grammar.
4. (10 marks) Write programs/methods/functions to compute $RIGHT(A)$, for any nonterminal of the grammar.

Step 2

1. (20 marks) Write a program that will build the precedence table and decide if your grammar is

   (a) Simple precedence grammar;
   (b) Weak precedence grammar;
   (c) Not a precedence grammar.

2. (10 marks) Given a grammar and a word, use your program to find a rightmost derivation by implementing the shift-reduce algorithm. In case the word is not in the language, produce appropriate error messages.

Step 3

- (20 marks)Modify the program such that terminals can be any sequence printable characters and they are stored in a symbol table while reading the grammar. Non-terminals are written between angle parenthesis < and >. If < or > must be used as terminals they should be escaped in the description of the production they appear, e.g. if the production is $< GT > \to >$, we will write it as $< GT > \to \backslash >$
- (20 marks)Use an ad-hoc lexical analyzer to feed the parser, so we can apply it to BNF rules and a programming language code.