Lab Assignment 5

Syntax Analysis/Parsing- (LL(1) predictive parser)

NOTE: Refer lecture notes, Chapter 4 (until sec 4.4).

For each of the grammars, perform the following steps and implement a predictive LL(1) parser.

STEP (1) Manually transform the grammar to LL(1) (without affecting/changing the language) by (remove left-recursion etc). As discussed, you may have to introduce new non-terminals in the process of transforming the grammar to LL(1). Write the **LL(1)** grammar in a file called "grammarLL.txt".

OPTIONAL: You may implement functions/programs to perform some of the transformation steps such as elimination of left-recursion (which takes any grammar as input and returns the transformed grammar as output).

STEP (2) For the LL(1) grammar obtained from step 1, manually compute the **FIRST()** set of all symbols in the grammar, and also compute the **FOLLOW()** set for all non-terminals. Store this information in another file e.g., file named "First-Follow.txt".

OPTIONAL: You may implement a program for computing FIRST(), FOLLOW() sets for any grammar given as input.

STEP (3) Write a C program to implement a table driven predictive parser for the LL(1) grammar obtained in step 1. Your implementation should contain two different modules/functions. The first module should construct the LL(1) parsing table from the (a) input grammar ("GrammarLL.txt") and (b) computed FIRST, FOLLOW sets ("First-Follow.txt"). You may represent the parse table either in a file or in a temporary data structure. The second module, take an input expression and parses it using the parse table. In case the input expression satisfies the given grammar, the output of the program should print "**Accepted**" along with the ordered sequence of productions (leftmost derivation). In the event of errors, provide reporting of both lexical and syntactic errors.

NOTE: Use Lex/Flex to generate a lexical analyzer for recognizing tokens.

Submission: Along with the source-code (step 3), for each example input grammar, submit files "grammarLL.txt". and "First-Follow.txt". Provide additional README file with other information (e.g., instructions to run, sample test inputs considered).

Example grammars are given below. Please note that only the steps that need to be done manually (steps 1, and 2) should be redone when you consider a different grammar.

Grammar 1:

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

Grammar 2:

For the following grammar,
the non-terminals are
N = { AE, BE, D, DL, E, F, ES, IOS, IS, NE, P, PE, RE, S, SL, T, TY, VL, WS },

the terminals are
T = { + , −, ∗, /, =, <, >, (,), {, }, :=, ; , and, else, end, **ic**, **id**, if, int, do, **fc**, float, not, or, print, prog, scan, **str**, then, while },

Most of the terminals are self-explanatory (id is an (identifier), ic is an integer constant, fc is a floating-point constant, str is a string of characters, := is an assignment etc.

The production rules are given in the next page (the start symbol is **P**).

P → prog DL SL end

DL → D DL | ε

D → TY VL ;

TY → int | float

VL → id VL | id

SL → S SL | ε

S → ES | IS | WS | IOS

ES → id := E ;

IS → if BE then SL end |

if BE then SL else SL end

WS → while BE do SL end

IOS → print PE | scan id

PE → E | str

BE → BE or AE | AE

AE → AE and NE | NE

NE → not NE | { BE } | RE

RE → E = E | E < E | E > E

E → E + T | E − T | T

T → T ∗ F | T / F | F

F → (E) | id | ic | fc