Syntactic Analysis

Experiment 03-01: Implementing context free grammar in C

S → cAb

A → ad | a

Experiment 03-02: Syntactic Analysis of simple arithmetic expressions.

A grammar for the concrete syntax of simple arithmetic expressions.

Production rules are:-

<Exp> ::= <Term> + <Term> | <Term> - <Term> | <Term>

<Term> ::= <Factor> * <Factor> | <Factor> / <Factor> | <Factor>

<Factor> ::= (<Exp>) | ID | NUM

ID ::= [a-z]

NUM ::= [0-9]

Non-terminal symbols:

<Exp>, <Term>, <Factor>

Terminal symbols:

+, -, *, /, (,), a, b, c, 1, 2, 3...

Start symbol:

<Exp>

Note the following are meta symbols:

::= (read it as "can be replaced by")

-> is alternative notation for ::=,

| (read it as "or by")

e.g. An <Exp> can be replaced by <Term> + <Term> or by <Term> - <Term>.

Tasks

Task: 01 Consider the Context Free Grammar G. You have the set of rules as the following:

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow 0S_1 1 \mid \mathcal{E}$$

$$S_2 \rightarrow 1S_2 0 \mid \mathcal{E}$$

Implement the grammar that will accept strings accepted by this language $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$.

Task 02: Revise the grammar of simple arithmetic expressions to accept the strings like,

(abc – b2)*(-c * d3) – g

Note: You have to use the previous lab materials to implement it.

Task 03: Implement the following grammar in C.

statement → assign-stat | decision_stat | looping-stat

assign_stat → id = expn

expn→ **simple-expn** eprime

eprime→relop simple-expn | ∈

decision-stat → if (expn) stat dprime

dprime → else stat | ∈

looping-stat → while (expn) stat | for (assign_stat ; expn ; assign_stat) stat

relop → == | != | <= | >= | > | <

Note: The grammar related to simple expression is described in experiment 02.
