TDT4205 Compiler Construction
Assignment 1

1 LL(1) parsing table construction

The following grammar abstracts the CREATE SCHEMA construct in SQL:
S → c s a U O
U → n
O → L | ϵ
L → E | L E
E → b | v | p

1. Modify it so that it becomes suitable for LL(1) parsing.
2. Tabulate the FIRST and FOLLOW sets for all nonterminals of the modified grammar, including which nonterminals are nullable (*i.e.* can derive the empty string).
3. Construct the LL(1) parsing table of the modified grammar.

2 VSL specification

The directory in the code archive ps2_skeleton.zip begins a compiler for a slightly modified 64-bit version of VSL ("Very Simple Language"), defined by Bennett (Introduction to Compiling Techniques, McGraw-Hill, 1990).

Its lexical structure is defined as follows:

- Whitespace consists of the characters '\t', '\n', '\r', '\v' and ' '. It is ignored after lexical analysis.
- Comments begin with the sequence '//', and last until the next '\n' character. They are ignored after lexical analysis.
- Reserved words are FUNC, BEGIN, END, RETURN, PRINT, CONTINUE, IF, THEN, ELSE, WHILE, DO, and VAR.
- Basic operators are assignment (:=), the basic arithmetic operators '+', '-', '*', '/', and relational operators '=', '<', '>'. In addition are the following bitwise operators : '>>' (rightshift), '<<' (leftshift), '~' (NOT), '&' (AND), ' ^' (XOR) and '|' (OR).
- Numbers are sequences of one or more decimal digits ('0' through '9').
- Strings are sequences of arbitrary characters other than '\n', enclosed in double quote characters '"'.

- Identifiers are sequences of at least one letter followed by an arbitrary sequence of letters and digits. Letters are the upper- and lower-case English alphabet ('A' through 'Z' and 'a' through 'z'), as well as underscore ('_'). Digits are the decimal digits, as above.

The syntactic structure is given in the context-free grammar on the last page of this document.
Building the program supplied in the archive ps2_skeleton.zip combines the contents of the src/ subdirectory into a binary src/vslc which reads standard input, and produces a parse tree.
The structure in the vslc directory will be similar throughout subsequent problem sets, as the compiler takes shape. See the notes set from the PS2 recitation for an explanation of its construction, and notes on writing Lex/Yacc specifications.

2.1 Scanner

Complete the Lex scanner specification in src/scanner.l, so that it properly tokenizes VSL programs.

2.2 Tree construction

A node_t structure is defined in include/ir.h. Complete the auxiliary functions node_init, and node_finalize so that they can initialize/free node_t-sized memory areas passed to them by their first argument. The function destroy_subtree should recursively remove the subtree below a given node, while node_finalize should only remove the memory associated with a single node.

2.3 Parser

Complete the Yacc parser specification to include the VSL grammar, with semantic actions to construct the program's parse tree using the functions implemented above. The top-level production should assign the root node to the globally accessible node_t pointer 'root' (declared in src/vslc.c).

$program \rightarrow global\_list$

$global\_list \quad \rightarrow global \quad | \quad global\_list \quad global$

$global \rightarrow function \quad | \quad declaration$

$statement\_list \quad \rightarrow statement \quad | \quad statement\_list \quad statement$

$print\_list \quad \rightarrow print\_item \quad | \quad print\_list \quad ',' \quad print\_item$

$expression\_list \quad \rightarrow expression \quad | \quad expression\_list \quad ',' \quad expression$

$variable\_list \quad \rightarrow identifier \quad | \quad variable\_list \quad ',' \quad identifier$

$argument\_list \quad \rightarrow expression\_list \quad | \quad \epsilon$

$parameter\_list \quad \rightarrow variable\_list \quad | \quad \epsilon$

$declaration\_list \quad \rightarrow declaration \quad | \quad declaration\_list \quad declaration$

$function \rightarrow FUNC \quad identifier \quad '(' \quad parameter\_list \quad ')' \quad statement$

$statement \rightarrow assignment\_statement \quad | \quad return\_statement$

$statement \rightarrow print\_statement \quad | \quad if\_statement$

$statement \rightarrow while\_statement \quad | \quad null\_statement \quad | \quad block$

$block \rightarrow BEGIN \quad declaration\_list \quad statement\_list \quad END$

$block \rightarrow BEGIN \quad statement\_list \quad END$

$assignment\_statement \rightarrow identifier \quad ':' \quad '=' \quad expression$

$return\_statement \rightarrow RETURN \quad expression$

$print\_statement \rightarrow PRINT \quad print\_list$

$null\_statement \rightarrow CONTINUE$

$if\_statement \rightarrow IF \quad relation \quad THEN \quad statement$

$if\_statement \rightarrow IF \quad relation \quad THEN \quad statement \quad ELSE \quad statement$

$whilestatement \rightarrow WHILE \quad relation \quad DO \quad statement$

$relation \quad \rightarrow expression \quad '=' \quad expression$

$relation \quad \rightarrow expression \quad '<' \quad expression$

$relation \quad \rightarrow expression \quad '>' \quad expression$

$expression \rightarrow expression \quad '|' \quad expression$

$expression \rightarrow expression \quad '\wedge' \quad expression$

$expression \rightarrow expression \quad '\&' \quad expression$

$expression \rightarrow expression \quad '>>' \quad expression$

$expression \rightarrow expression \quad '<<' \quad expression$

$expression \rightarrow expression \quad '-' \quad expression$

$expression \rightarrow expression \quad '*' \quad expression$

$expression \rightarrow expression \quad '/' \quad expression$

$expression \rightarrow \quad '-' \quad expression$

$expression \rightarrow \quad ' \;\tilde{}\; ' \quad expression$

$expression \rightarrow \quad '(' \quad expression \quad ')'$

$expression \rightarrow \quad number \quad | \quad identifier \quad | \quad identifier \quad '(' \quad argument\_list \quad ')'$

$declaration \rightarrow VAR \quad variable\_list$

$printitem \rightarrow expression \quad | \quad string$

$identifier \quad \rightarrow IDENTIFIER$

$number \rightarrow \quad NUMBER$

$string \rightarrow \quad STRING$