TDT4205 Problem Set 2
Spring 2016

Answers are to be submitted by *It's Learning*, by Feb. 15^{th}, 20:00.

1 Top-down parsing

1.1 LL(1) form

Rewrite the following grammar into LL(1) form, by left factoring it and eliminating left recursion.
$S \rightarrow sCT|sCTwB$
$C \rightarrow c$
$T \rightarrow t|\epsilon$
$B \rightarrow Ba|a$

1.2 Parsing table

Tabulate the FIRST and FOLLOW sets of the nonterminals in the resulting grammar, and construct the predictive parsing table.

2 VSL specification

The directory in the code archive ps2 skeleton.tgz 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.
- *Operators* are assignment (:=), the basic arithmetic operators '+', '-', '*', '/', and relational operators '=', '<', '>'.
- *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.tgz 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 slide 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).

$program \quad \rightarrow \quad global\_list$
$global\_list \quad \rightarrow \quad global \quad | \quad global\_list \quad global$
$global \rightarrow \quad function \quad | \quad declaration$
$statement\_list \quad \rightarrow \quad statement \quad | \quad statement\_list \quad statement$
$print\_list \quad \rightarrow \quad print\_item \quad | \quad print\_list \quad ',' \quad print\_item$
$expression\_list \quad \rightarrow \quad expression \quad | \quad expression\_list \quad ',' \quad expression$
$variable\_list \quad \rightarrow \quad identifier \quad | \quad variable\_list \quad ',' \quad identifier$
$argument\_list \quad \rightarrow \quad expression\_list \quad | \quad \epsilon$
$parameter\_list \quad \rightarrow \quad variable\_list \quad | \quad \epsilon$
$declaration\_list \quad \rightarrow \quad declaration \quad | \quad declaration\_list \quad declaration$
$function \quad \rightarrow \quad FUNC \quad identifier \quad '(' \quad parameter\_list \quad ')' \quad statement$
$statement \quad \rightarrow \quad assignment\_statement \quad | \quad return\_statement$
$statement \quad \rightarrow \quad print\_statement \quad | \quad if\_statement$
$statement \quad \rightarrow \quad while\_statement \quad | \quad null\_statement \quad | \quad block$
$block \quad \rightarrow \quad BEGIN \quad declaration\_list \quad statement\_list \quad END$
$block \quad \rightarrow \quad BEGIN \quad statement\_list \quad END$
$assignment\_statement \quad \rightarrow \quad identifier \quad ':' \quad '=' \quad expression$
$return\_statement \quad \rightarrow \quad RETURN \quad expression$
$print\_statement \quad \rightarrow \quad PRINT \quad print\_list$
$null\_statement \quad \rightarrow \quad CONTINUE$
$if\_statement \quad \rightarrow \quad IF \quad relation \quad THEN \quad statement$
$if\_statement \quad \rightarrow \quad IF \quad relation \quad THEN \quad statement \quad ELSE \quad statement$
$while\_statement \quad \rightarrow \quad WHILE \quad relation \quad DO \quad statement$
$relation \quad \rightarrow \quad expression \quad '=' \quad expression$
$relation \quad \rightarrow \quad expression \quad '<' \quad expression$
$relation \quad \rightarrow \quad expression \quad '>' \quad expression$
$expression \quad \rightarrow \quad expression \quad '+' \quad expression$
$expression \quad \rightarrow \quad expression \quad '-' \quad expression$
$expression \quad \rightarrow \quad expression \quad '*' \quad expression$
$expression \quad \rightarrow \quad expression \quad '/' \quad expression$
$expression \quad \rightarrow \quad '-' \quad expression$
$expression \quad \rightarrow \quad '(' \quad expression \quad ')'$
$expression \quad \rightarrow \quad number \quad | \quad identifier \quad | \quad identifier \quad '(' \quad argument\_list \quad ')'$
$declaration \quad \rightarrow \quad VAR \quad variable\_list$
$print\_item \quad \rightarrow \quad expression \quad | \quad string$
$identifier \rightarrow \quad IDENTIFIER$
$number \rightarrow \quad NUMBER$
$string \rightarrow \quad STRING$