TDT4205 Problem Set 3
Spring 2012

PART 1 - Theory due Wed Feb. 22^{th}, 20:00.
PART 2 - Programming due Wed Feb. 29^{th}, 20:00.

Part 1 needs to be passed in order for Part 2 to be evaluated! ALL answers are to be submitted to *itslearning*

ALL OF THIS ASSIGNMENT IS TO BE DONE INDIVIDUALLY. Cheating ("koking"), including using partial solutions from other students, will cause a failing grade for the whole course. We will be checking for this using a plagerism detecting as well as looking for suspiciously similar submissions.

All submitted source code MUST be able to compile and run on asti. **This assignment counts towards your final grade.** Please read the assignment guidelines on itslearning before starting to work on the assignment. Requests for clarifications can be posted on the itslearning forum.

What to turn in
When turning in assignments, please turn in two files:

- (your_username)_code.zip,tar.gz,tgz : All your code for this assignment, including makefiles and other necessary code (Part 2)

PART 2 - Programming: Simplifying VSL syntax trees

The provided archive *ps3_skeleton.tar.gz* contains a working parser which constructs syntax trees, mirroring the grammar from problem set 2. Implement the function
`simplify_tree ( node_t **simplified, node_t *root )`,
such that it recursively traverses the tree rooted at `*root`, and rewrites the tree to implement the following simplifications:

Task 2.1: Redundant node pruning (10%)

`STATEMENT`, `PRINT_ITEM`, `PARAMETER_LIST` and `ARGUMENT_LIST` nodes are only conveniences for treating two or more nonterminals as the same when writing the grammar. When parsing completes, these nodes add no information beyond what is evident from the type of their only child, and they can therefore be eliminated entirely.

Task 2.2: List flattening (20%)

The list nonterminals are recursively defined for the sake of accepting arbitrary length lists. After parsing, the length of the list they encode is known from the depth of the subtree below, and list subtrees can therefore be flattened (i.e. encoded as a single list node, with all list elements directly attached as children). This should be performed for all the recursively defined list types, i.e. `DECLARATION_LIST`, `FUNCTION_LIST`, `STATEMENT_LIST`, `PRINT_LIST`, `EXPRESSION_LIST`, and `VARIABLE_LIST`. *(Note that declaration lists have a slight structural difference from the rest, as they do not have a separate 'wrapper' nonterminal to cover the case when they are empty and produce ϵ.)*

Task 2.3: Print statements (10%)

`PRINT_STATEMENT` nodes only serve to ensure that the `PRINT` keyword is correctly placed, and always have a single `PRINT_LIST` as a child. After flattening the `PRINT_LIST`, this permits the print statement node to attach all list items directly, eliminating print lists entirely.

Task 2.4: Folding of constant expressions (20%)

Arithmetic (sub-)expressions which depend only on constants can be evaluated at compile time. These should be rewritten into integer nodes containing the value of the expression.