# COMP4403/7402 Compilers and Interpreters
# Sample Solution to Tutorial Exercise 1 (2021/1)*

School of Information Technology and Electrical Engineering, University of Queensland

February 11, 2021

For this tutorial you will need copies of the PL0 Concrete Syntax and PL0 Compiler Data Structures as used in the lectures.

1. For the following PL0 program:

```
var
  x: int; y: int; max: int;
begin
  read x; read y;
  if x <  y then
    max := y
  else
    max := x;
  write max
end
```

   (a) Describe the sequence of tokens supplied by the scanner (lexical analyser). Note that "int" is a predefined identifier, not a keyword in the language. See Figure 1 in the PL0 Concrete Syntax handout for the definitions of PL0 tokens. You don't need to do the entire program.

   **Sample solution.** The sequence of tokens will be:
   KW_VAR,
   IDENTIFIER("x"), COLON, IDENTIFIER("int"), SEMICOLON,
   IDENTIFIER("y"), COLON, IDENTIFIER("int"), SEMICOLON,
   IDENTIFIER("max"), COLON, IDENTIFIER("int"), SEMICOLON,
   KW_BEGIN, KW_READ, IDENTIFIER("x"), SEMICOLON,
   KW_READ, IDENTIFIER("y"), SEMICOLON,
   KW_IF, IDENTIFIER("x"), LESS, IDENTIFIER("y"), KW_THEN,
   IDENTIFIER("max"), ASSIGN, IDENTIFIER("y"),
   KW_ELSE, IDENTIFIER("max"), ASSIGN, IDENTIFIER("x"),
   SEMICOLON, KW_WRITE, IDENTIFIER("max"), KW_END,
   EOF

   (b) Give the parse tree that results from parsing just the **if** statement in the above program, i.e. ignore everything up to and including the reads and everything from the write onwards. The top node in the parse tree should be for the nonterminal *IfStatement* and the tree should be consistent with the PL0 Concrete Syntax (see Figure 2 in the handout).

   **Sample solution.** The parse tree is given in Figure 1, in which the IDENTIFIER token has been abbreviated to ID.

   (c) Give the abstract syntax tree for the **if** statement within the program. The tree should be consistent with the tree nodes used in the compiler (see PL0 Compiler Data Structures handout).

   **Sample solution.** The abstract syntax tree is given in Figure 2.

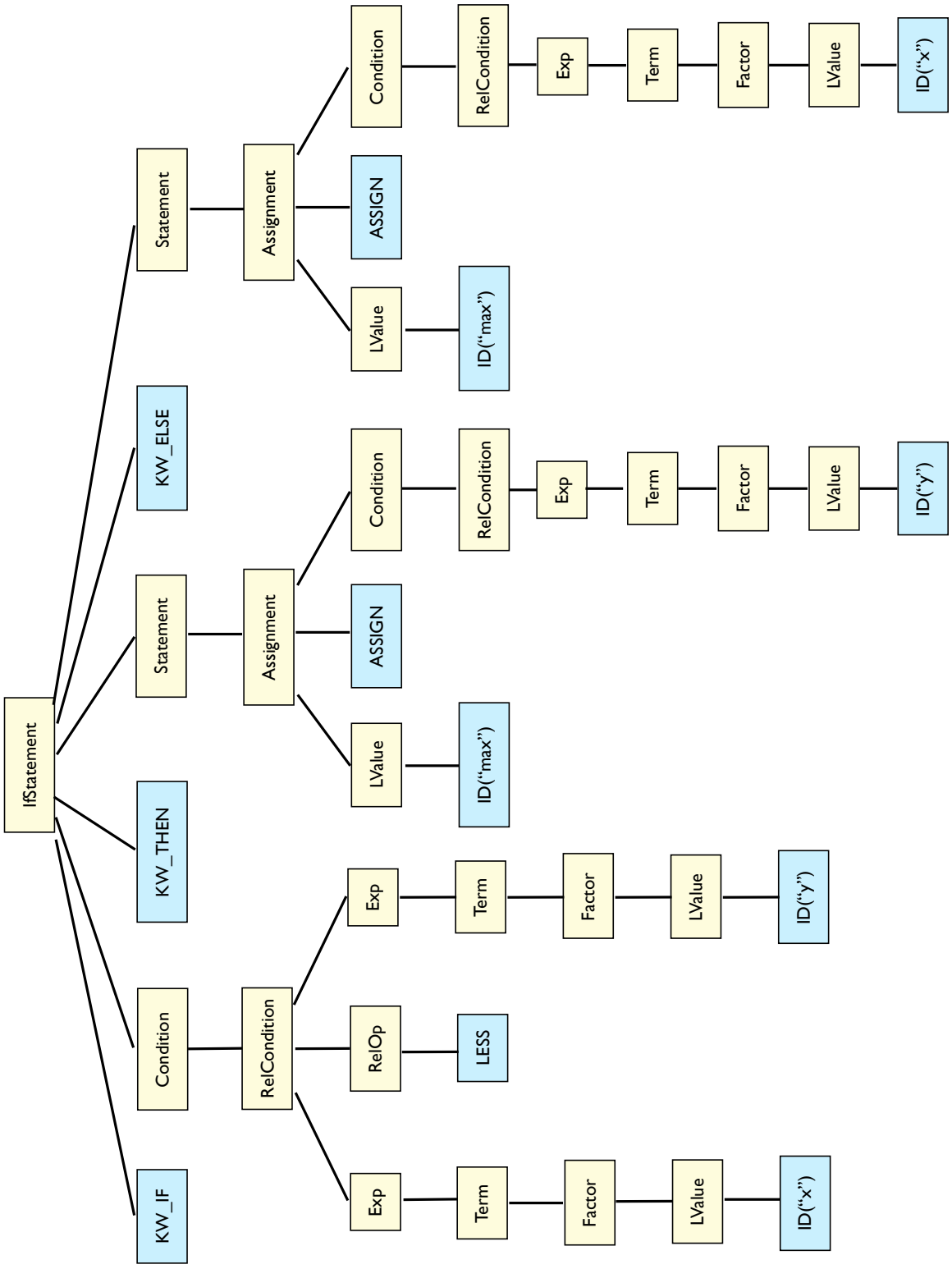


Figure 1: Parse tree for if x < y then max := y else max := x

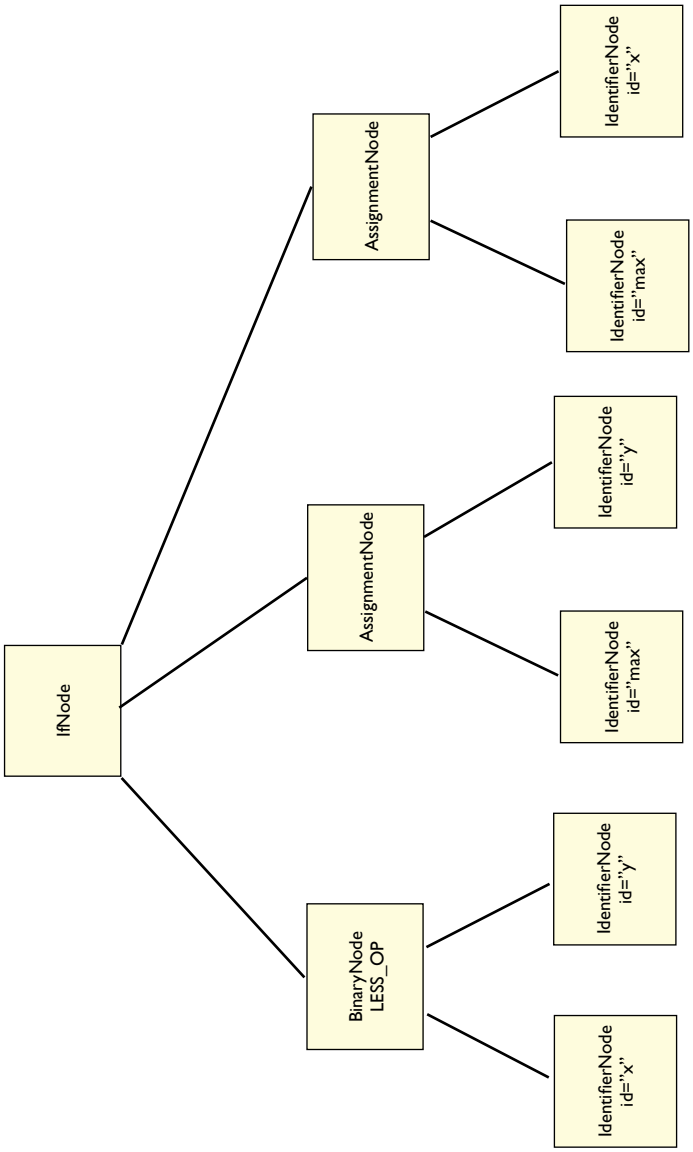


Figure 2: Abstract syntax tree for if x $<$ y then max := y else max := x

2. In which compiler module would you expect the following to occur:

   (a) An undeclared identifier is detected.

   **Sample solution.** Note that the answers to these questions give a bit more detail than we expect of you at this stage.

   Conceptually in the static semantics analyser (using symbol table) but in the PL0 compiler this is done as the symbol table (and abstract syntax tree) is being set up during the parsing phase.

   (b) An illegal character in the source is detected.

   **Sample solution.** Lexical analyser (or scanner).

   (c) The use of ']' instead of ')' is detected.

   **Sample solution.** Parser.

   (d) Unreachable code is eliminated.

   **Sample solution.** (Source code) optimiser. *Here we distinguish between optimisations that are based only on the source code and optimisations that depend on the target machine but we don't expect that in your answers.*

   (e) Run time checks for range overflow are inserted.

   **Sample solution.** (Intermediate) code generation.

   (f) Run time checks for range overflow are eliminated where it can be shown from the context that they are redundant.

   **Sample solution.** (Source code) optimiser.

   (g) The use of a ';' instead of a ',' in a parameter list is detected.

   **Sample solution.** Parser.

   (h) The insertion appropriate for a C language '#include file' macro command (which results in textual substitution of the contents of file at this point) is performed. (Warning: this is something of a trick question!)

   **Sample solution.** This is actually done in a separate preprocessor, which is run before the compiler. If not done separately, it would have to be done early, in the scanner.

   (i) A type conflict in an expression is detected.

   **Sample solution.** Static semantics analyser.

   (j) A constant too large for the target machine is detected.

   **Sample solution.** If the programming language defines the maximum size for an integer value then it makes sense to do the check early on in the lexical analyser (scanner) but for some languages (like C) the maximum size of an integer may be dependent on target machine (e.g. 16-bit versus 32-bit versus 64-bit machines) and then there is a strong argument for deferring it to target machine code generation. Note that a compiler for a language may have options to generate code for different target machines.

   (k) A branch (jump) instruction that has as its destination an unconditional branch instruction is changed to go directly to the destination of the second branch.

   **Sample solution.** Source code optimiser (or peephole optimiser).

   (l) Whether the language has name equivalence or structural equivalence of types is an influence.

   **Sample solution.** Static semantic analyser.

   (m) Whether ';'s are used as separators or terminators for statements.

   **Sample solution.** Parser.

   (n) Code for coercing an integer to a real where this is required in a mixed mode expression (an expression with both integers and reals) is created.

**Sample solution.** This is handled in the static semantic analyser which transforms the expression to include a coercion node, the code generation for which will convert the integer to a real.

(o) The appropriate instance of an overloaded method required for a particular call is identified. Overloading of methods is said to occur when the same name is used for more than one method. The appropriate instance is identified using the types of the parameters.

**Sample solution.** Static semantic analyser.