Computer Science C24
University of Toronto Scarborough

Homework Exercise # 1

Handing in and marking

For all exercises/assignments in this course, you need to submit your solutions to the pencil-and-paper questions on crowdmark and your solutions to the programming questions on MarkUs. Your pencil-and-paper solutions will be marked with respect to correctness, clarity, brevity, and readability. Your code will be marked with respect to correctness, efficiency, program design and coding style, clarity, and readability. This exercise counts for 5% of the course grade.

Question 1. (6 marks) Consider the following production rules for a context free grammar. Assume that the rule `<bool-identifier>` has been defined correctly to generate identifiers.

```
<boolean-expr> ::= <boolean-expr> AND <boolean-expr>
                 | <boolean-expr> OR <boolean-expr>
                 | NOT <boolean-expr>
                 | (<boolean-expr>)
                 | <bool-identifier> | True | False
```

- Clearly demonstrate that a CFG with these production rules is unacceptable for defining boolean expressions in a programming language.
- Provide a CFG that generates the same boolean expressions, is acceptable for defining boolean expressions in a programming language, and follows these rules:
  - The precedence order is (from highest to lowest): parentheses, negation, conjunction, disjunction.
  - Conjunction and disjunction are both left-associative.

Question 2. (20 marks) Consider the following simple programming language $\mathcal{L}$.

The expressions in $\mathcal{L}$ are:
- boolean expressions defined in the previous question,
- arithmetic expressions defined in class, and
- function calls with zero or more arguments, of the form:
  - `function-name(expr0, ... )`

Note, that an expression on its own is not a valid program in $\mathcal{L}$. Instead, valid programs in $\mathcal{L}$ are built from statements, as follows.

- $\mathcal{L}$ has assignment statements:
  `identifier := expression`
  where `expression` is any valid expression.
- The statements can be executed sequentially by using the semicolon:
  `statement; statement`
- $\mathcal{L}$ has conditional statements:
  `if ( condition ) then { statement } else { statement }`
  where `condition` is any valid boolean expression, and `statement` is any valid statement (or statements, combined sequentially), and
- indefinite loop statements:
  `while ( condition ) { statement }`
  where `condition` is any valid boolean expression, and `statement` is any valid statement (or statements, combined sequentially).

The precedence is (from highest to lowest): assignment (`:=`), sequential composition (`;`), conditional (`if-then-else`), indefinite loop (`while`).

The language also has parentheses, i.e., statements can be parenthesized, and parentheses have the highest precedence.

Provide a CFG for $\mathcal{L}$. The grammar must be unambiguous — this is a programming language! Assume that the rules for `<identifier>` and `<func-name>` have already been written (correctly!). Use the (corrected version of the) grammar from the previous question to define `condition`s.

Question 3. (4 marks) Presentation and readability of the submission.

Question 4. (2 marks bonus) Provide a CFG for the language just like $\mathcal{L}$, but the conditional statement is of the form

```
if ( condition ) then statement else statement
```

i.e., the curly braces are removed.