McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3E03
Exercise Sheet 2

Design and Selection of Programming Languages

20 September 2006, updated 21 September 2006

Exercise 2.1 — Context-Free Syntax: Exponents (Midterm 1, 2005)

For this question, the **abstract syntax** of expressions is defined by the following grammar:

$$\begin{array}{lll} \mathit{Expression} & \rightarrow & \mathit{Number} \ \mid \ \mathit{Expression}\ \mathit{Op}\ \mathit{Expression} \\ \mathit{Op} & \rightarrow & \texttt{*} \ \mid \texttt{/} \mid \texttt{\^{}} \end{array}$$

Define a **concrete syntax** for these expressions by giving a **context-free grammar (e.g. in EBNF)** such that

- the grammar is unambiguous,
- multiplication `*` and division `/` associate to the left,
- exponentiation `^` has higher precedence and associates to the right.

For example, the two strings “`2 / 3 ^ 4 ^ 5 / 7`” and “`(2 / (3 ^ (4 ^ 5))) / 7`” represent the same expression.

Exercise 2.2 — Expression Manipulation in Java

Substitution $e_1[v \mapsto e_2]$ of an expression e_2 for a variable v in an expression e_1 is defined as follows:

$$\begin{array}{lcll} v[v \mapsto e] & = & e & \\ w[v \mapsto e] & = & w & \text{if } v \neq w \\ k[v \mapsto e] & = & k & \text{for } k \in \mathit{Num} \\ (e_1 \oplus e_2)[v \mapsto e] & = & (e_1[v \mapsto e]) \oplus (e_2[v \mapsto e]) & \text{for } \oplus \in \mathit{Op} \end{array}$$

This exercise further modifies the expression classes of Exercise 1.2.

(a) Add an instance method *substituteVariable* that takes as arguments a variable, and an expression to be substituted into that variable, and **returns the result of the substitution** into the expression for which the method is called.

(b) Add an instance method *destructivelySubstituteVariable* that takes as arguments a variable, and an expression to be substituted into that variable, and **modifies the expression object for which the method is called** by performing the substitution.

(c) Discuss the difference between these two methods!

Exercise 2.3 — Expression Parsing and Manipulation in C

Extend the C datatype for expressions and the simple bison-based calculator presented in the lecture (source files are available on the course page) with the following functionality — carefully define and document the interfaces:

(a) Add a function for producing string representations from expressions.

(b) Add an exponentiation operator.

(c) Add destructive and non-destructive substitution functions as in Exercise 2.2.

(d) Further modify the simple calculator presented in class so that it accepts definitions of variables, introduced by the keyword “let”:

```
let x = 4
let y = 5
x+y
 = 9
```

(e) Further modify the simple calculator presented in class so that it produces step-wise evaluation traces:

```
(4+3) * 8 - 2*7
 = (4 + 3) * 8 - 2 * 7
 = 7 * 8 - 2 * 7
 = 56 - 2 * 7
 = 56 - 14
 = 42
```