IN2009: Language Processors

2006/2007 Coursework 1

Organisation: This coursework assessment may be completed individually, or in pairs, and there is no penalty for working in a pair. You will hand work in as individuals but the handin procedure will allow you to say who you collaborated with (and you must do so!). *Pairs* only – not threesomes!

Hand in: The deadline and the electronic handin procedure, and exactly what you should hand-in, are documented online. Obviously you should change file and directory permissions while you are working so that your work is not visible to others – remember plagiarism carries severe penalties.

Use JavaCC regular expressions to define precisely integer literals and floating point literals as described below. In this context, 'literal' means the piece of text that appears in a program to denote a number (for example, the text '`3.142`' denotes the number 3.142.

Task 1 - Regular Expressions

Integer Literals An integer literal may be expressed as decimal, hexadecimal, or octal numerals. Each may be suffixed with an 'L' to denote an integer of type `long`. A decimal numeral is either the single character `0`, or consists of a digit from `1` to `9`, optionally followed by one or more digits from `0` to `9`. A hexadecimal numeral consists of leading characters `0X` or `0x` followed by one or more hexadecimal digits. A hexadecimal digit is a digit from `0` to `9` or a letter from `a` through `f` or `A` through `F`. An octal numeral consists of a digit `0` followed by one or more of the digits `0` to 7. Examples of integer literals: `0 1996 0372 0xDadaCafe 0L 0777L 0xC0B0L 0x00FF00FF`

Floating point literals A floating point literal has the following parts: a whole-number part, a decimal point (represented by a period character), a fractional part, an exponent, and a type suffix. A type suffix is either the letter `d` (denoting `double` type) or `f` (denoting `float` type). The exponent, if present, is indicated by the letter `e` followed by an optionally signed number. A least one digit, in either the whole number or the fraction part, and either a decimal point, an exponent, or a float type suffix are required. All other parts are optional. Subject to the above constraints, the whole-number part, the fractional-part and the number in the exponent are sequences of digits from `0` to `9`. Examples: `1e1f 2.f .3f 0f 3.14f 6.022137e23f 1e1 2. 0.3 0.0 3.14 1e-9d 1e137`

Implement and test your expressions using JavaCC (make your expressions readable and understandable).

Task 2 - MiniJava modification

Copy the MiniJava implementation directory to where you want to work, using:

`cp -R /soi/sw/courses/daveb/IN2009/minijava/chap4` (or download it from Cityspace)

Files `README` in the various directories give a brief description of the structure of the implementation. The MiniJava BNF and reference manual have been handed out and are online.

1. Add a Java-like `for`-statement to the MiniJava implementation

$$< Statement > \; - > < for > \; (< Exp >; < Exp >; < Exp >) \{ < Statement > \}$$

2. In a similar way, add the Java-like try and throw statements defined below to the implementation:

$$< Statement > \; - > < try > \; \{ < Statement >^* \} \; < Catch >^* < finally > \; \{ < Statement >^* \}$$

$$< Statement > \; - > < throw > < Exp > ;$$

$$< Catch > \; - > < catch > \; (< Type > < id >) \{ < Statement >^* \}$$

Note that the $< Catch >$s sequence may be represented as a list in your abstract syntax (and hence programmed like the other lists).

You will need to add the new statements to the JavaCC specification (including appropriately adjusting the token regular expressions), so that it builds the correct abstract syntax trees, and write new appropriate abstract syntax tree classes. The pretty-printer will also need to be updated to appropriately print the new abstract syntaxes you have introduced.

Note: remember that these exercises deal only with lexical and syntax analysis and producing appropriate abstract syntax trees: you do not yet have to worry about how MiniJava programs are type-checked, execute or have code produced for them!