

Assessment 1 (Regular Expressions)

Organisation: This coursework assessment may be completed individually or in pairs - there is no penalty for working in a pair. You will hand work in as individuals but the hand-in procedure will allow you to say who you collaborated with (and you must do so!).

Hand in: The deadline and the electronic hand-in 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.

Weighting: This component is worth 30% of the coursework marks.

Task 1 - Regular Expressions

Use JavaCC regular expressions to define precisely **integer literals**, **floating point literals** and **comments** 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). You are to implement and test your expressions using JavaCC (make your expressions readable and understandable).

Integer Literals. An integer literal may be expressed as *binary*, *decimal*, *hexadecimal*, or *octal* numerals. Each may be suffixed with an 'L' to denote an integer of type long, and may be prefixed with a + or a - character to indicate sign.

1. A binary numeral consists of the leading characters b or B followed by one or more of the digits 0 or 1.
2. 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.
3. A hexadecimal numeral consists of leading characters Z or z 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.
4. An octal numeral consists of a leading characters o or O followed by one or more of the digits 0 to 7. Examples of integer literals: 0 1996 0372 ZDadaCafe O777L ZC0B0L b0110 426355690003133711121133114641.

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* (in that order). 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 or E followed by an optionally signed number. At least one digit in either the whole number or the fraction part is required. Also, at least a decimal point, an exponent, or a float type suffix is required. All other parts are optional.

Subject to the above constraints, the fractional-part and the number in the exponent are sequences of digits from 0 to 9. The whole-number part is a sequence of digits from 0 to 9 and may optionally be prefixed with a + or a - character to indicate sign.

Examples: 1e1f 2.f .3f 0f 3.14f 6.022137E23f 1e1 2. 0.3 0.0 3.14 1e-9d 1e137 -5.56e4263 -42f

Comments. The language accepts multiple- and single-line comments. Multiple-line comments must start with the string `/*` and must end with the string `*/`. Single-line comments start with the `//` character.

The Program.

- The lexical analyzer should report on all tokens found. You should also define a special token for errors i.e. strings that can't be matched should be reported.
- Convert all **Integer Literals** to their decimal representation and report their value. For example, token `B110` should be displayed as 6 (a partial implementation will be provided).
- A sample run:

Input: `34 3.14e12 /* dgd */ b3b b111`

Output: `INT(34) FLOAT(3.14e12) COMMENT ERROR INT(7)`

Advice:

- Create regular expressions for each useful case (so perhaps one for binary integer literals, another for decimal integer literals, etc.), and build up larger expressions from smaller ones.
- Test your expressions regularly! Don't forget, an acceptable test plan must be provided to achieve full marks.
- Aim for a smaller number of entirely correct cases rather than lots of broken ones.