Programming Assignment 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.

Task 1 - Regular Expressions

Download the file cw1.jj from the CW1 CitySpace folder. The cw1.jj JavaCC file defines a parser that recognises the INTEGER_LITERAL, IDENTIFIER and HORROR tokens. Inspect the file and generate the parser by executing the commands learned during the lab sessions.

Execute the parser. An input string "12 id12 12id 512" should give you the following output:

```
INTEGER_LITERAL> (`12')  value = 12
<IDENTIFIER> (`id12')
<HORROR> (`12id')
<INTEGER_LITERAL> (`512')  value = 512
```

Note that the token HORROR will capture any string that contains any character except whitespaces. This will force the parser to accept only tokens which are separated by spaces, besides providing a way of capturing illegal strings (you won't get the ParserError exception related to lexical errors).

TASK: Update cw1.jj in order to implement the following:

1) Integer Literals. Modify the INTEGER_LITERAL definition by introducing an optional character (+ or -) to indicate sign. For example, the input string "12 +34 -45" should return the following output:

```
INTEGER_LITERAL> (`12')  value = 12
INTEGER_LITERAL> (`+34')  value = 34
INTEGER_LITERAL> (`-45')  value = -45
```

Note that the part of the code that computes the value of the decimal string should be updated in order to reflect the sign of the number.

2) MYID Token. Add the new token **MYID** that recognises **your** College id/username. For example, since my College id is sbbc287, a run of **my** parser with input "12 sbbc287 sbbc286 86" must produce the following output:

```
<INTEGER_LITERAL> (`12')  value = 12
"sbbc287" (`sbbc287')
<IDENTIFIER> (`sbbc286')
```

<INTEGER_LITERAL> (`86') value = 86

3) MYID2 Token. College ids are made of two parts, an alphabetic and a numeric part. For example, **sbbc287** is made of alphabetic part **sbbc** and numeric part **287**. Add the new token **MYID2** that recognises strings that:

- Start with your id's alphabetic part (e.g. **sbbc**).
- Finish with your id's numeric part. (e.g. **287**)
- Contains zero or more repetitions of your id's alphanumeric and numeric parts.

For example, if your college id were **sbbc287**, the strings:

sbbc287287 sbbcsbbc287sbbc287287 sbbc287287287287

should be recognised as token **MIYD2**. Note that **sbbc287** satisfies the rules for **MYID** and **MYD2**. The parser should report **MYID**. Thus, given input "sbbc287287 sbbc287 sbbcsbbc287", my parser should produce the following output:

<MYID2> (`sbbc287287')
"sbbc287" (`sbbc287')
<MYID2> (`sbbcsbbc287')

IMPORTANT: Note that I'm using as example sbbc287. Your parser should be defined to accept YOUR id number, and its variations.

4) MYID3 Token. Add the new token MYID3 that accepts variations of the numeric part of your id, according to the examples shown below. Given id **sbbc287**, the following strings are valid MYID3 token elements:

sbbc2287 sbbc222287 sbbc22287 sbbc2222222287
sbbc2887 sbbc288887 sbbc28887 sbbc2888888887
sbbc2877 sbbc287777 sbbc28777 sbbc2877777777

That is, a digit of your id can be repeated several times, but only a single digit at a time. This means that **sbbc2228887**, for example, is not a valid **MYID3**.

5) RATIONAL Token. A rational number is made of an unsigned integer (numerator), followed by the "/" character and a second unsigned integer (denominator). No spaces should be allowed after or before the "/" character. Add the new token RATIONAL that recognises rational numbers and update the output part of your program to print the numeric value of the rational number. For example, given input "**23 sbbc287 8/2 2/5 2 / 5 4/0** ", your parser should produce the following output:

INTEGER_LITERAL> (`23') value = 23
"sbbc287" (`sbbc287')
<RATIONAL> (`8/2') value = 4.0
<RATIONAL> (`2/5') value = 0.4
<INTEGER_LITERAL> (`2') value = 2
<HORROR> (`/')
<INTEGER_LITERAL> (`5') value = 5
<RATIONAL> (`4/0') value = infinity

Note that rational numbers with 0 denominator are undefined. Instead of generating an error, print **infinity**, as shown above.

Advice: You can use the indexOf and substring methods from the Java String class in order to extract the numerator and denominator parts of your rational number. It's a good idea to define a second variable (e.g. val2) in order store the value of your denominator. Also, cast the result of the division to double i.e. (double) val / val2.

Points:

- INTEGER_LITERAL: 5 (regexp 3, value 2)
- MYID: 4
- MYID2: 5
- MYID3: 5
- RATIONAL: 6 (regexp 3, value 3)

TOTAL : 25