Victoria University of Wellington
School of Engineering and Computer Science

# SWEN221: Software Development

# Mid-term Test (worth 10% of overall mark)

In this test you will be working on a small, text-based spreadsheet. The application reads a spreadsheet source representation and returns a textual view of the evaluated spreadsheet. The source consists of one row per line, with commas seperating cells on each row. Cells may be empty, contain integers, or formulae.

Example input:

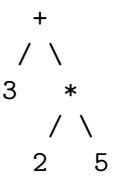
```
1, =(+ 1 (* 3 2)), ,4
1, 2, 3, 4
-2,3, ,
```

Example output:

```
    |       1|       2|       3|       4|
----+--------+--------+--------+--------+
1   |       1|       7|        |       4|
2   |       1|       2|       3|       4|
3   |      -2|       3|        |        |
```

Formulae (introduced with `=`) must be in *polish (prefix) notation*, that is the operator comes first, e.g., $3 + 2$ is represented as `(+ 3 2)` and $3 + 2 * 5$ as `(+ 3 (* 2 5))`. You must always use brackets and there is no precedence.

Internally, a formula is represented as a tree, e.g., `(+ 3 (* 2 5))` is stored as:




The numbers at the leaves of the tree are stored as `IntLiteral` objects, the addition and multiplication nodes are both stored as `BinaryOp` objects.

The supported operators are `+`, `-`, `*`, `/` (which have there standard interpretations), and `$`. `$` is a reference to a cell in the spreadsheet, e.g., `($ 1 2)` in the above example would evaluate to 7 (note that the first coordinate is the row). If there is an error parsing or evaluating a cell, then "`err`" is printed in the cell.

**Downloading the code** You can download the source code from:

`http://ecs.victoria.ac.nz/Courses/SWEN221_2010T1/Midterm/spreadsheet.jar`

The source code is split into four packages: the `parsing` package deals with parsing — parsing strings of input data into a spreadsheet; the `formulae` package is used to represent a formula as an abstract syntax tree; the `main` package does the rest — the `Spreadsheet` class represents a spreadsheet, and `Cell` represents a cell in a spreadsheet; `EvalException` represents an error occurring whilst evaluating the contents of a cell. Finally, the `tests` package contains the `SpreadsheetTests` class, which contains some sample tests and tests for questions 1 and 4. You should add your tests for part 2 to this class.

# 1 Warm-up (10 marks)

The test `SpreadsheetTests.warmUpTest()` currently fails. It fails because a cell reference with a negative index prints `0`, where it should print `err`. Fix this bug. (Hint: `Cell.printableString()` returns "`err`" if evaluating the cell throws an exception.)

# 2 Testing (25 marks)

The program has more bugs! For each of the following bugs, write a JUnit test which fails for the provided code, but passes if the bug is fixed. You should add your tests to the `SpreadsheetTests` class and call the test methods `test1` to `test4`. You may use the `check` method if you wish.

1. An empty cell should display nothing, it currently displays 0 (hints: you should test Spreadsheet.print(); you do NOT need to look at the parsing code).
2. Dividing by zero causes the user to see a stack trace, it should just produce a cell containing "err" (hints: you only need to test the formulae package; make sure your test works for compound formulae, e.g. `(/ 1 (- 3 3))`).
3. Subtraction doesn't seem to work properly (hint: you might want to look at how formulae are parsed for this one).
4. A formula which contains non-integer, non-formulae data (e.g., "`=1.0`") should produce a cell containing "err", but it doesn't.

# 3 Debugging (30 marks)

Fix each of the bugs from part 2.

# 4 Extending the spreadsheet (35 marks)

Extend the code in the following ways. You will get marks for style; please submit any tests you write with your code, as these will contribute to the style marks given. You are encouraged to refactor the provided code to improve the design. Make sure you do not break any of the supplied tests, or the tests you wrote in part 2.

1. **Add support for factorials.** This should be a formula which calculates the factorial[1] of its single parameter. E.g., `=(! 4)` gives 24, `=(! (- 5 3))` gives 2. Your code should throw an `EvalException` (which will be displayed as `err`), if the user tries to calculate the factorial of a number less than 0 ($!0 = 1$). Your solution should pass `factorialTest`, you may wish to write more tests.

2. **Add support for strings.** The spreadsheet should support strings: a cell which starts with a single quote (e.g., "`'hello`") should be treated as a string literal and be displayed verbatim in the spreadsheet. E.g.,

   ```
   1, =(+ 1 (- 3 2)), ,4
   ,'hello,,0
   ,3,hello,'hello world!
   ```

   should give

   ```
       |       1|       2|       3|       4|
   ----+--------+--------+--------+--------+
   1   |       1|       2|        |       4|
   2   |hello   |        |        |       0|
   3   |        |       3|     err|hello ..|
   ```

   Note that, strings should be left justified and strings longer than eight characters should be truncated with "`..`".

**Hints**

- Because factorial takes a single parameter you will not be able to use the `BinaryOp` class (which is used for addition etc.).
- `String.format` can be used to give fixed width columns, e.g., `String.format("%8d", 4)` produces the string "       4", i.e., `4` padded with spaces to make a string at least eight characters wide. Use a negative number to left justify, e.g., `String.format("%-8d", 4)`.

**Submission** You should submit your solutions through the usual assignment submission system. Please make sure you submit to the correct session. Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances). The URL for submission is:

`http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221`

[1] The factorial of a number ($n$) is the product of all numbers between 1 and $n$, e.g., $4! = 1 * 2 * 3 * 4 = 24$