Victoria University of Wellington
School of Engineering and Computer Science

SWEN431: Advanced Programming Languages
Assignment 2: Haskell Stack Deluxe

Due: 8 May 2025, before midnight

1 Outline

You are to implement a "Stack Deluxe" interpreter using Haskell. In terms of input/output requirements, this assignment is identical to assignment 1; the same input files and expected results still apply. The details of the code completion stages are the same as well. The report follows the same structure but now requires a comparison of Haskell to both Ruby and Java (see the "Report" section for more details).

2 Clarifications

Based on previously asked questions, here are some clarifications regarding the interpreter requirements:

Q: Should the division of integers result in a float or an integer?
A: Dividing integers should result in an integer result. Hence, in the following example, the stack should contain "2" not "2.5".

```
INTEGER DIVISION
INPUT    5 2 /
OUTPUT   2
```

Q: Is it a problem to produce float results, if only integers are needed?
A: Any integer operation, such as

```
INTEGER DIVISION
INPUT    5 2 +
OUTPUT   7
```

should produce an integer result, such as "7" as opposed to "7.0".

Remember, if in doubt, assume the Ruby semantics, even for this Haskell assignment. The goal is to produce the expected output, not let the semantics of the implementation language bleed through.

Q: Will invalid input be used for marking?
A: No, you do not have to do handle any cases of invalid input. Our input will not provoke any errors based on incompatible types, division by zero, etc.

3 Haskell Side Effects and Submission Requirements

Input/Output operations are regarded as side effects in Haskell. This means that they cannot be dealt with in a purely functional style. In order to facilitate input/output, and other side effects, Haskell supports so-called monads. The monadic programming style mimics the imperative programming style and *should not be used anywhere in the solution*, with the exception of the input/output file management.

In order to save you the trouble of figuring out the respective details of file management and accessing command line arguments, we are providing a solution template for you. The template only adds `START` and `END` markers to the input to demonstrate a transformation and hence obviously requires changes. You will have to adapt it, so that it

1. produces output files with the correct names,
2. produces the correct content, and
3. is named `ws.hs` instead of `ws-template.hs`.

Your submitted code must produce output file `output-001.txt`, for instance, when invoked as follows:

```
runghc ws.hs input-001.txt
```

You may use standard libraries, but they must not require installation, i.e., your solution needs to run on an ECS machine (or standard Windows installation) without requiring any prior setup/download actions.

Your entire code must be contained in `ws.hs`, i.e. you may use multiple files during development but must ensure that, upon submission, `ws.hs` contains all your code.

For any further requirements, please see section 2 of the assignment 1 description.

4 Report

The code mark contributes 40% of the overall mark for this assignment, while the report contributes the remaining 60%.

Your report must be formatted using 12pt font, standard line spacing, margins no smaller than 2.54cm, *without exceeding* two A4 pages.

Marks will be awarded for the following:

- **Design**. Explain the main design of your solution, e.g., important data structures, distribution of responsibilities, etc. If you considered design alternatives, please mention those here and explain why you have discarded them.
- **Haskell Features**. List which *noteworthy* Haskell features and libraries you used and why. For instance, mention the use of higher-order functions, but not regular function application. Make sure to explain which features supported or hindered development. This section and the one above will highly benefit from attempts on your behalf to use Haskell idiomatically and/or making the best use of its features. If you explicitly did not make use of a Haskell feature because you did not feel it would have been appropriate or it would not have helped, justify such decisions here.

  Remember to refrain from using monads to create a solution that adopts an imperative programming style. The purpose of the assignment is to make you use the pure functional programming style.
- **Tooling**. Briefly describe what tooling you used and how it supported or hindered the development of your solution.

- **Comparison**. Discuss how Haskell made it easier or more difficult to create the solution than Ruby and/or Java would have. Ensure to make comparisons to both languages, but if they do not materially differ with respect to some aspect, you can of course talk about both using one phrase only.

Use the above four sections in this order to structure your report. Note that your discussions need to specifically relate to your implementation. Generic statements, e.g., enumerating Haskell features without referencing your implementation, will not be awarded any marks.

5 Submission

Submit your

- code – as `ws.hs`, and
- report – as `report.pdf`

electronically by 8 May 2025, before midnight.