Compiler Construction
Supervision 1

Supervisor: Joe Isaacs (josi2).
All work should be submitted in PDF form 36 hours before the supervision to the email `josi2@cam.ac.uk`. If you have any questions on the course please include these at the top of the supervision work and we can talk about them in the supervision.

1. What sequences of translations from 'higher-level' to 'lower-level' languages do modern compilers perform? Give a broad overview.

2. For each stage give an overview of the work performed? What is the input and output from each stage.

3. What are VMs and why are they useful?

4. `http://www.cl.cam.ac.uk/teaching/1617/CompConstr/Exercises_Set_2.ml`

   (**Include commented type signatures for all functions**). Remember a continuation function of

   ```
   (SOMETIHING -> 'b) -> 'b
   ```

5. `http://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2016p3q3.pdf`.

6. `http://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2017p23q4.pdf`

7. Practical exercise: `https://www.cl.cam.ac.uk/teaching/1920/CompConstr/practical.txt`

   (a) *Add a modulo*[1] *operation to the arithmetic operations. How will you handle modulo with zero?*

   Notice this is slightly different to the linked exercise. I don't expect you to change the parser or lexer, just extend the AST and add support for evaluation into Interp_0 and Interp_2. Then you can create you own ast.expr to test evaluation, even though you cannot write a `.slang` program with the modulo operation (yet). You will add support of this in SV4 by extending the parser.

   (b) *Implement simple "tuple pattern matching" in the compiler. For example, instead of writing*
   *let rev (p : int * int) : int * int = (snd p, fst p) in rev (21, 17) end*
   allow users to write
   *let rev (x : int, y : int) : int * int = (y, x) in rev (21, 17) end*
   Hint : treat this as syntax sugar – eliminate in front end by translating the second rev into the first. Or better yet, intro let bindings for "snd p" and "fst p".

   Again here you are not expected to change the lexer or parser, but extend the AST and create a syntactic mapping from tuple pattern matching to let bindings or instances of `fst` and `snd`.

[1] `https://en.wikipedia.org/wiki/Modulo_operation`