2019-20 **UoB, COMS22201**

Exercises week 17

Exercise 17.1 Use the compiler `scomp` from `Intcomp1.hs` to compile the following expressions to produce a sequence of instructions. You shall try to work it out on paper before checking your answer with GHCi. Note that you will need to translate the programs into abstract syntax before calling `scomp`.

```
let z = 17 in z + z

20 + let z = 17 in z + 2 end + 30
```

Now execute your instruction sequences according to `seval` by specifying the stack content at each step. Again, you shall work it out on paper before using GHCi. Note that if your execution results in an error, it may mean that your instruction sequence from above is wrong.

You are strongly encouraged to repeat this exercise with other expressions you may come out with and check the answer with GHCi.

Exercise 17.2 In the last lab, we have extend the expression language `Expr` from `Intcomp1.hs` with multiple *sequential* let-bindings, such as this (in concrete syntax):

```
let x1 = 5+7   x2 = x1*2 in x1+x2 end
```

Revise the `Expr`-to-`TExpr` compiler `tcomp :: Expr -> TExpr` from `Intcomp1.fs` to work for the extended `Expr` language. There is no need to modify the `TExpr` language or the `teval` interpreter to accommodate multiple sequential let-bindings.

Exercise 17.3 Write a bytecode assembler (in Haskell) that translates a list of bytecode instructions for the simple stack machine in `Intcomp1.fs` into a list of integers. The integers should be the corresponding bytecodes as seen below.

```
SCST = 0, SVAR = 1, SADD = 2, SSUB = 3, SMUL = 4, SPOP = 5, SSWAP = 6;
```

Thus you should write a function `assemble :: [SInstr] -> [Int]`.

Use this function together with `scomp` from `Intcomp1.hs` to make a compiler from the original expressions language `Expr` to a list of bytecodes `[Int]`.

Exercise 17.4 Modify the compiler from Exercise 17.3 to write the lists of integers to a file. A list `inss` of integers may be output to the file called `fname` using this function (found in `Intcomp1.hs`):

```
intsToFile :: [Int] -> String -> IO ()
intsToFile inss fname
 = do let text = intercalate " " (map show inss)
      writeFile fname text
```