Starred exercises (⋆) are more challenging. Please try all unstarred exercises before the tutorial meeting.

1. **Pattern matching.** For this problem, you should use the Scala definition of \( \text{L}_{\text{Arith}} \) abstract syntax trees presented in the lectures:

   ```scala
class Expr
object Num extends Expr
object Plus extends Expr
object Times extends Expr
```

(a) Write a Scala function `evens[A] : List[A] => List[A]` that traverses a list and returns all of the elements in even-numbered positions. For example:

\[
\text{evens(List('a','b','c','d','e','f')) = List('a','c','e')}
\]

(b) Write a Scala function `allplus : Expr => Boolean` that traverses a \( \text{L}_{\text{Arith}} \) term and returns `true` if all of the operations in it are additions, `false` otherwise. (For this problem, you may want to use the Scala Boolean AND operation `&&`.)

(c) Write Scala function `consts : Expr => List[Int]` that traverses a \( \text{L}_{\text{Arith}} \) expression and constructs a list containing all of the numerical constants in the expression. (For this problem, you may want to use the Scala list-append operation `::`.)

(d) Write Scala function `revtimes : Expr => Expr` that traverses a \( \text{L}_{\text{Arith}} \) expression and reverses the order of all multiplication operations (i.e. \( e_1 \times e_2 \) becomes \( e_2 \times e_1 \)).

(e) (⋆) Write a Scala function `printExpr : Expr => String` that traverses an expression and converts it into a (fully parenthesised) string. For example:

```scala
scala> printExpr(Times(Plus(Num(1), Num(2)),
                          Times(Num(3), Num(4))))
res0: String = ((1 + 2) * (3 * 4))
```

2. **Evaluation derivations.** Recall the evaluation rules covered in lectures:

\[
e \Downarrow v
\]
Write out derivation trees for the following expressions:

(a) $6 \times 9$
(b) $3 \times 3 + 4 \times 4 == 5 \times 5$
(c) $(\star)(1 + 1 == 2\;\text{then}\;2 + 3\;\text{else}\;2 \times 3)$
(d) $(\star)(1 + 1 == 2\;\text{then}\;3\;\text{else}\;4) + 5$

3. Typechecking derivations. Recall the typechecking rules covered in lectures:

\[
\frac{}{\mathcal{I} : \tau}
\]

\[
\frac{n \in \mathbb{N}}{\vdash n : \text{int}}\quad \frac{\vdash e_1 : \text{int}}{\vdash e_1 + e_2 : \text{int}}\quad \frac{\vdash e_1 : \text{int}}{\vdash e_1 \times e_2 : \text{int}}
\]

\[
\frac{b \in \mathbb{B}}{\vdash b : \text{bool}}\quad \frac{\vdash e_1 : \text{int}}{\vdash e_1 == e_2 : \text{bool}}\quad \frac{\vdash e : \text{bool}}{\vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \text{int}}
\]

Write out typing derivations for the following judgments:

(a) $\vdash 6 \times 9 : \text{int}$
(b) $(\star)\vdash (\text{if } 1 + 1 == 2 \text{ then } 3 \text{ else } 4) + 5 : \text{int}$

4. $(\star)$ Nondeterminism. Suppose we add the following construct $e_1 \boxtimes e_2$ to $L_{\text{Arith}}$:

\[
e := e_1 + e_2 \mid e_1 \times e_2 \mid n \in \mathbb{N}
\mid \text{true} \mid \text{false} \mid e_1 == e_2 \mid \text{if } e \text{ then } e_1 \text{ else } e_2
\mid e_1 \boxtimes e_2
\]

Informally, the semantics of $e_1 \boxtimes e_2$ is that we evaluate either $e_1$ or $e_2$ nondeterministically. To model this we extend the evaluation rules as follows:

\[
e \Downarrow v
\]

\[
\frac{e_1 \Downarrow v}{e_1 \boxtimes e_2 \Downarrow v} \quad \frac{e_2 \Downarrow v}{e_1 \boxtimes e_2 \Downarrow v}
\]

(a) What property of $L_{\text{Arith}}$ (among those discussed in Lecture 2) is violated after we add $\boxtimes$?
(b) Write a sensible rule for typechecking $e_1 \boxtimes e_2$.
(c) For each of the following expressions $e$, list all of the possible values $v$ such that $e \Downarrow v$ is derivable:
   i. $(1 \boxtimes 2) \times (3 \boxtimes 4)$
   ii. if (true $\boxtimes$ false) then 1 else 2
(d) Define an expression $e$ and a value $v$ such that there are two different derivations of the judgment $e \Downarrow v$. (What does it mean for the derivations to be different?)