3 Compiler Construction (tgg22)

We will extend our language SLANG and the JARGON virtual machine with data type definitions such as

```
type int_list = Nil | Cons of int * int_list

type int_tree = Leaf of int | Node of int * int_tree * int_tree
```

(Note that we will not consider polymorphic types.)

We also extend the language with a `match` expression and pattern matching so that we can write functions such as

```
let tail (l : int_list) : int_list =
    match l with
    | Cons(x, l') -> l'
    end
end

let is_nil (l : int_list) : bool =
    match l with
    | Nil -> true
    | l' -> false
    end
end

let sum (x : int_tree) : int =
    match x with
    | Leaf y -> y
    | Node(y, t1, t2) -> y + (sum t1) + (sum t2)
    end
end
```

The semantics of the `match` expression: Match clauses are attempted from first to last. If no match is found then there is a run-time error and the program halts (we don't have exceptions).

(*a*) Ignoring lexing and parsing, what changes to the compiler's front-end would this require? [3 marks]

(*b*) Discuss possible runtime representations of values of types such as `int_list` and `int_tree`. [3 marks]

[continued ...]

(*c*) Assuming your runtime representation uses tags, do you need distinct tags for `Cons(x, l)` and `Node(x, t1, t2)`? Justify your answer. [4 marks]

(*d*) Suppose that our language allows nested patterns such as

```
match t with
| Node(x, Leaf y, t2) -> e1(x, y, t2)
| Node(x, t1, Leaf y) -> e2(x, t1, y)
| Node(x, Node(y, t1 t2), t3) -> e3(x, y, t1, t2, t3)
end
```

but our front-end generates abstract syntax that cannot contain nested patterns. How would you represent the code above in the same language without nested patterns? [4 marks]

(*e*) Carefully describe the code you would generate for the JARGON virtual machine for the body of the function `sum` defined above. If you need to extend the virtual machine with new instructions, then define their semantics. (You do not need to remember the exact syntax of the JARGON instructions as long as you clearly explain what your code is doing.) [6 marks]