CO545 Spring Term 2020-21, Assessment 3 — Test Mock

This class consists of an in—class test, of duration 40 minutes (scaled to reflect any "extra time" allowances in your ILP). The test is based on the first four Erlang lectures, as well as your general—purpose Functional Programming skills from the Haskell part.

Assessment conditions

You may consult any of your own notes, or my lecture materials (including example files). Please do not use the Internet to look up solutions (unlikely to help anyway).

Submission

- Complete your work in a single file named by your login id, e.g., `smk.erl`.
- Your file must be valid Erlang, i.e., it should load into erl with no errors. Any incomplete code should be commented out with a note. A submission which fails to compile will lose marks.
- Before the end of the allocated time, upload your assessment to the Moodle submission box. Check your upload is present and correct once you have done this. Do not submit again after the end of your allocated time period (remember Moodle logs the time stamp), although you may submit multiple times during the allocated time period.
- You must not use any library functions other than arithmetic and boolean operators and the functions `spawn/1` and `spawn/3`. Any attempt to submit after the end of the session will result in a mark of zero.

Questions

These are all worth 5 marks each.

1. We can represent ternary logic in Erlang using the atoms `true,false,maybe`. Using pattern matching write a function `and3/2` such that both arguments and result should be a value of ternary logic. On "normal" booleans it should do the same thing as boolean `and`, but `maybe` overrides `true` (e.g., `and3(maybe,true)=true`) but not `false` (e.g., `and3(false,maybe)=false`).

2. Using `and3/2` from the previous question, write a function `and3list/1` whose argument is a list of ternary values and which should combine all of them using `and3/2`.

   For example, `and3list([true, maybe, true]) = maybe` and `and3list([true, false]) = false`.

3. We can model Haskell's `Maybe` type using the atom `nothing` (instead of `Nothing`) and the tuple patterns `{just,X}` (instead of `Just x`). Write a function `onemaybe/1` that turns all layers of maybeness that the argument may have into 1 layer. For example, `onemaybe({just,nothing})` should be `nothing`, `onemaybe({just,{just,6}}` should be `{just,6}`, and `onemaybe(56)` should be `{just,56}`.

4. Write a function `bumpyall/1` whose argument is a list of process ids and which sends the message `bump` to all of them.

5. Write a function `holdvalue/1` whose argument is an arbitrary Erlang value, and which (when spawned) can do the following: whenever it receives the message `{req,P}` (where P is meant to be a process id) it sends its value parameter to the process `P`.