Fall, 2019 Name: ____________________

(Please *don't* write your id number!)

COP 4020 — Programming Languages I

Test on Erlang and Actor Programming

Special Directions for this Test

This test has 9 questions and pages numbered 1 through 10.
This test is open book and notes, but no electronics.
If you need more space, use the back of a page. Note when you do that on the front.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your programs are sloppy and hard to read, you may lose some points. Correct syntax also makes a difference for programming questions. We will take some points off for duplicated code, code with extra unnecessary cases, or code that is excessively hard to follow.
You will lose points if you do not "follow the grammar" when writing programs! You should always assume that the inputs given will follow the grammar for the types specified, and so your code should not have extra cases for inputs that do not follow the grammar.
When you write Erlang code on this test, you may use anything that is built-in to Erlang (module `erlang`) and the modules `lists` and `ets`, unless the problem specifies otherwise.
You are encouraged to define functions not specifically asked for if they are useful to your programming; however, if they are not built-in to Erlang functions or from the `lists` or `ets` modules, then you must write them into your test. That is, you can use built-in functions such as `length/1`, as well as functions from the `lists` and `ets` modules such as: `lists:map/2`, `lists:foreach/2`, `lists:foldr/3`, `lists:filter/2`, `lists:member/2`, `lists:reverse/1`, `lists:sort/1`, `lists:sublist/2`, `lists:sum/1`, `lists:seq/2`, `ets:new/2`, `ets:lookup/2`, `ets:insert/2`, etc., unless the problem specifies otherwise.

For Grading

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	14	5	10	5	6	20	15	5	20	100
Score:										

1. [Concepts] This is a question about free and bound identifiers in Erlang code. As in the homework, both variable names and function names are considered to be identifiers.

   Consider the following Erlang expression.

   ```
   N(fun (N,M) ->
           IsZero(N, one, Mult(N, (Fix(F))(Pred(N))))
     end)
   ```

   (a) (10 points) In set brackets ({ and }), write the complete set of all variable identifiers that occur free in the above expression.

   (b) (4 points) In set brackets ({ and }), write the complete set of all variable identifiers that occur bound in the above expression.

2. (5 points) [Concepts] Consider the following Erlang code.

   ```
   main() ->
       N = 40,
       AddN = cadd(N),
       definesN(AddN, 3).

   cadd(X) ->
       fun(Y) -> X+Y end.

   definesN(F,Z) ->
       N = 2,
       F(Z).
   ```

   What is the result of the expression `main()` with the above program?

3. (10 points) [UseModels] Consider the function concatMap, specified as follows in Erlang,

```
-spec concatMap(F::fun((T) -> list(R)), Ls::list(T)) -> list(R).
```

concatMap takes a function, F (from some type T to a list of some type R) and a list Ls (of elements of type T), and returns a list of elements (of type R), which is the concatenation of the results of applying F to each element of Ls in order. The following are examples.

```
-module(concatMap_tests).
-export([main/0]).
-import(testing,[eqTest/3, dotests/2]).
-import(concatMap,[concatMap/2]).
main() -> compile:file(concatMap),
          dotests("concatMap_tests $Revision: 1.1 $", tests()).
tests() ->
   begin
     Add12 = fun(N) -> [N+1, N+2] end,
     Is7 = fun(N) -> case N==7 of true -> [7]; false -> [] end end,
     [eqTest(concatMap(Add12, []), "==", []),
      eqTest(concatMap(Add12, [30,40,50]), "==", [31,32,41,42,51,52]),
      eqTest(concatMap(Is7, [3,4,5]), "==", []),
      eqTest(concatMap(Is7, [4,0,7,7,2,3]), "==", [7,7]),
      eqTest(concatMap(fun(X) -> [X*X+1] end, [1,20,300,4000]),
             "==", [2,401,90001,16000001])  ]
   end.
```

Circle the choice below that is a correct implementation of concatMap/2 in Erlang (assuming it appears in a module named concatMap).

A.
```
concatMap(_f,[]) = []
concatMap(f,(x:xs)) = f(x) ++ concatMap(f,xs)
```

B.
```
concatMap(_f,[]) -> [];
concatMap(f,[x|xs]) -> f(x) ++ concatMap(f,xs).
```

C.
```
concatMap(F,Ls) -> lists:foldr(fun(X,Res) -> [F(X)|Res] end, [], Ls).
```

D.
```
concatMap(_F,[]) -> [];
concatMap(F,Ls) -> lists:map(F, Ls).
```

E.
```
concatMap(_f,[]) -> [];
concatMap(f,[x]) -> f(x);
concatMap(f,ls) -> lists:concat(lists:map(f, ls)).
```

F.
```
concatMap(_F,[]) -> [];
concatMap(F,[H|T]) -> F(H) ++ concatMap(F,T).
```

G. None of the above is correct.

4. (5 points) [Concepts] Consider the following Erlang module.

```
-module(semantics).
-export([starttest/0, testIt/1, checktest/0]).

starttest() -> spawn(?MODULE, testIt, [self()]).

testIt(Pid) -> receive
                   {TPid, check} -> TPid ! {result, self() == Pid}
               end.

checktest() -> Spawned = starttest(),
               Spawned ! {self(), check},
               receive
                   {result, B} -> B
               end.
```

Assuming that this module is compiled, circle the correct choice.

A. The call `semantics:checktest()` returns **true**.

B. The call `semantics:checktest()` returns **false**.

C. The call `semantics:checktest()` returns 1.

D. The call `semantics:checktest()` returns 0.

E. The call `semantics:checktest()` returns a process id.

F. The call `semantics:checktest()` encounters an error and does not return normally.

G. None of the above correctly describes what happens when the call `semantics:checktest()` is executed.

5. (6 points) [Concepts] Briefly justify (explain) your answer in question 4.

6. (20 points) [UseModels] In Erlang, consider a table server that can be created by passing to the start function specified by

```
-spec start(Tab::list({_Key,_Value})) -> pid().
```

a list of pairs of the form `{Key,Value}`, where each pair is a 2-tuple whose first element is a key (of some type `Key`) and whose second element is a value (of some type `Value`). The server remembers this list of pairs. It follows a protocol with just one type of message:

- `{Pid, lookup, K}`, where `Pid` is the sender's process id and `K` is a key (of type `Key`). The server responds by sending to `Pid` either a message of the form:
  - `{TPid, {found, V}}`, if `V` is the first value in the server's list that is in a pair of the form `{K,V}`, or
  - `{TPid, not_found}`, if there is no such pair with key `K` in the list.

  In both cases, `TPid` is the table server's process id.

The following are examples using the homework's testing module.

```
-module(table_tests).
-export([main/0]).
-import(table, [start/1,lookup/2]).
-import(testing, [eqTest/3, dotests/2]).
main() -> compile:file(table),
          dotests("table_tests $Revision: 1.1 $", tests()).
tests() ->
    Empty = start([]),
    States = start([{fl, florida},
                    {ga, georgia},
                    {ms, mississippi},
                    {al, alabama},
                    {sc, south_carolina},
                    {mi, michigan}]),
    Nums = start([{one, 1},
                  {two, 2},
                  {three, 3},
                  {four, 4},
                  {five, 5},
                  {six, 6}]),
    [eqTest(lookup(Empty, sosp), "==", not_found),
     eqTest(lookup(Empty, 7), "==", not_found),
     eqTest(lookup(States, fl), "==", {found, florida}),
     eqTest(lookup(Nums,four), "==", {found, 4}),
     eqTest(lookup(Nums,two), "==", {found, 2}),
     eqTest(lookup(Nums,six), "==", {found, 6}),
     eqTest(lookup(Nums,one), "==", {found, 1}),
     eqTest(lookup(Nums,seven), "==", not_found),
     eqTest(lookup(Nums,2), "==", not_found),
     eqTest(lookup(States, al), "==", {found, alabama}),
     eqTest(lookup(States, ut), "==", not_found),
     eqTest(lookup(States, georgia), "==", not_found),
     eqTest(lookup(States, ga), "==", {found, georgia}),
     eqTest(lookup(States, ia), "==", not_found),
     eqTest(lookup(States, sc), "==", {found, south_carolina})
    ].
```

Write the table server, without using the lists, ets, dets, or mnesia modules, by filling in the blanks in the code on the following page.

```
-module(table).
-export([start/1,init/1,lookup/2]).
%% Client function
-spec lookup(Pid::pid(), K::any()) -> {found, any()} | not_found.
lookup(Pid, K) ->
    Pid ! {self(), lookup, K},
    receive
       {Pid, Response} -> Response
    end.

-spec start(Tab::list({_Key,_Value})) -> pid().

start(Tab) -> spawn(?MODULE, init, __________________________).

-spec init(Tab::list({_Key,_Value})) -> no_return().
init(Tab) -> loop(Tab).

loop(Tab) ->
    receive

        _________________________ ->

            Pid ! {self(), assoc(Tab, Key)}
    end,

    _______________________________.


% A helping function
-spec assoc(Tab::list({Key,Value}), K::Key) -> {found, Value} | not_found.

assoc(_______________, _______________) -> not_found;

assoc(_______________, _______________) -> {found,V};

assoc(_______________, _______________) -> assoc(Rest, K).
```

7. (15 points) [UseModels] In an Erlang, write a module named `ledger`, which remembers a list of entries (of any type). To implement this, in the module `ledger` write a server with a function `start/0`, which creates a ledger server and returns its process id. The ledger server can be sent two types of messages:

   - `{Pid, enter, Entry}`, where `Pid` is the sender's process id, and `Entry` is a value that should be remembered at the end of the ledger server's list. The ledger server remembers `Entry` at the end of its list and responds by sending the sender (`Pid`) a message of the form `{SPid, entered}`, where `SPid` is the server's process id.
   - `{Pid, list}`, where `Pid` is the sender's process id. The ledger server responds by sending a message to `Pid` of the form `{SPid, Entries}`, where `SPid` is the server's own process id and `Entries` is the list that the ledger server is remembering.

   The following are tests. There is space for your answer on the next page.

```
-module(ledger_tests).
-export([main/0,ledger_enter/2,ledger_list/1]).
-import(testing,[eqTest/3,dotests/2]).
-import(ledger,[start/0]).
main() ->
    compile:file(ledger),
    dotests("ledger_tests $Revision: 1.2 $", tests()).
tests() ->
    L1 = start(), L2 = start(),
    [eqTest(ledger_list(L1), "==", []),
     eqTest(ledger_list(L2), "==", []),
     eqTest(ledger_enter(L1,a), "==", ok),
     eqTest(ledger_list(L1), "==", [a]),
     eqTest(ledger_list(L2), "==", []),
     eqTest(ledger_enter(L1,c), "==", ok),
     eqTest(ledger_enter(L2,z), "==", ok),
     eqTest(ledger_list(L1), "==", [a,c]),
     eqTest(ledger_list(L2), "==", [z]),
     eqTest(ledger_enter(L2,y), "==", ok),
     eqTest(ledger_list(L2), "==", [z,y]),
     eqTest(ledger_enter(L1,a), "==", ok),
     eqTest(ledger_list(L1), "==", [a,c,a]),
     eqTest(ledger_enter(L1,4020), "==", ok),
     eqTest(ledger_list(L1), "==", [a,c,a,4020])
    ].

%% Client functions: NOT FOR YOU TO WRITE
-spec ledger_enter(LPid::pid(), Entry::any()) -> ok.
ledger_enter(LPid, Entry) ->
    LPid ! {self(), enter, Entry},
    receive
        {LPid, entered} -> ok
    end.

-spec ledger_list(LPid::pid()) -> list(any()).
ledger_list(LPid) ->
    LPid ! {self(), list},
    receive
        {LPid, List} -> List
    end.
```

Please write your answer below, completing this module.

```
-module(ledger).
```

8. (5 points) [Concepts] In the test cases for problem 7, the list `[a,c,a,4020]` is formed. Briefly explain why the formation of that list is *not* a type error in Erlang.

9. (20 points) [UseModels] In Erlang, write a stack server in a module named `stack`. You should write a function `start/0` that returns the process id of the server. A stack server remembers the state of a stack, which is a last-in first-out queue (so the top of the stack is the last element pushed on the stack and the rest of the elements are ordered from most to least recently pushed). This server can be sent messages of the following forms:

   - `{Pid, size}`, to which the server responds by sending a message to the process whose process id is `Pid` of the form `{SPid, N}`, where `SPid` is the process id of the stack server and `N` is the number of elements in the stack that the server is remembering. (The elements remembered are unchanged.)
   - `{Pid, {push, Elem}}`, which causes the server to remember `Elem` as the top of the stack, without forgetting the remaining elements in the remembered stack. The server sends to `Pid` a message of the form `{SPid, ok}` as an acknowledgment.
   - `{Pid, pop}`, which has an effect that differs based on the state of the remembered stack:
     - If the remembered stack is empty, then the server sends a message of the form `{SPid, stack_is_empty}` to `Pid` and continues the server continues to remember the same (empty) stack.
     - If the remembered stack is not empty, then the server sends a message of the form `{SPid, Elem}`, where `Elem` is the top element on the stack, and the server remembers only the rest of the elements (without the top element).

     In both cases, `SPid` is the server's own process id.

   There are tests below.

```
-module(stack_tests).
-export([main/0,stack2list/1]).
-import(stack,[start/0]).
-import(testing, [eqTest/3,dotests/2]).
main() -> compile:file(stack),
          dotests("stack_tests $Revision: 1.1 $", tests()).
tests() ->
    S1 = start(), S2 = start(),
    [eqTest(stack_size(S1), "==", 0),      eqTest(stack_size(S2), "==", 0),
     eqTest(stack_push(S1,e), "==", ok),   eqTest(stack_size(S1), "==", 1),
     eqTest(stack2list(S1), "==", [e]),    eqTest(stack_size(S2), "==", 0),
     eqTest(stack_push(S1,gee), "==", ok), eqTest(stack_size(S1), "==", 2),
     eqTest(stack_pop(S1), "==", gee),     eqTest(stack_pop(S1), "==", e),
     eqTest(stack_push(S1,e), "==", ok),   eqTest(stack_push(S1,gee), "==", ok),
     eqTest(stack_push(S1,e), "==", ok),   eqTest(stack2list(S1), "==", [e,gee,e]),
     eqTest(stack_push(S2,bee), "==", ok), eqTest(stack_size(S2), "==", 1),
     eqTest(stack2list(S2), "==", [bee]),  eqTest(stack_pop(S2), "==", bee),
     eqTest(stack2list(S1), "==", [e,gee,e]),
     eqTest(stack_push(S1,she), "==", ok), eqTest(stack_size(S1), "==", 4),
     eqTest(stack2list(S1), "==", [she,e,gee,e])
    ].
%% Client functions for testing: NOT FOR YOU TO CODE
rpc(Pid, Msg) -> Pid ! {self(), Msg},
                 receive {Pid, Response} -> Response end.
stack_size(Pid) -> rpc(Pid, size).
stack_push(Pid, Elem) -> rpc(Pid, {push, Elem}).
stack_pop(Pid) -> rpc(Pid, pop).
stack2list(Pid) -> stack2list(Pid, stack_size(Pid), []).
stack2list(Pid, 0, Elems) ->
    lists:foreach(fun (E) -> stack_push(Pid, E) end, Elems), % restore the stack
    lists:reverse(Elems);
stack2list(Pid, N, Elems) -> stack2list(Pid, N-1, [stack_pop(Pid)|Elems]).
```

Please write your answer below, completing the `stack` module.

```
-module(stack).
-export(
```