Spring, 2015 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 7 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`.
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.

For Grading

Question:	1	2	3	4	5	6	7	Total
Points:	14	6	10	10	20	20	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.

```
concat(map(fun(Y) -> F(X,Y) end, (fun(Q,R) -> Q end)(LS,empty)))
```

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

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

2. (6 points) [Concepts] In Erlang, will there necessarily be a type error if one puts different types of elements in a single list, like the following?

```
[3, true, foo, $c, [1,2], [ok,sure], 5.2, self()]
```

   Answer "yes" or "no" and give a brief justification.

3. (10 points) [UseModels] In Erlang, write a function `sumsquares/1`, which has the following type specification:

```
-spec sumsquares(LON :: [number()]) -> number().
```

This function takes a list of numbers, `LON`, and returns the sum of the squares of the numbers in `LON`. The following are tests using the homework's testing module.

```
-module(sumsquares_tests).
-export([main/0]).
-import(sumsquares,[sumsquares/1]).
-import(testing,[dotests/2,eqTest/3]).
main() -> compile:file(sumsquares),
          dotests("sumsquares_tests $Revision: 1.1 $", tests()).
tests() ->
    [eqTest(sumsquares([]), "==", 0),
     eqTest(sumsquares([3]), "==", 9),
     eqTest(sumsquares([4,3]), "==", 25),
     eqTest(sumsquares([1,2,4,3]), "==", 30),
     eqTest(sumsquares([1,10,500]), "==", 250101),
     % note that lists:seq(1,100) == [1,2,3,4,...,98,99,100]
     eqTest(sumsquares(lists:seq(1,100)), "==", 338350),
     eqTest(sumsquares([30|lists:seq(1,100)]), "==", 339250),
     eqTest(sumsquares(lists:seq(7,74321)), "==", 136842860543870)  ].
```

4. (10 points) [UseModels] In Erlang, write a stateless server in a module named `average`. You should write a function `start/0` that returns the process id of a server. This server responds to messages of the form `{Pid, {average, LON}}`, where `Pid` is the sender's process id, and `LON` is a non-empty list of numbers. When the server receives such a message, it responds by sending a message of the form `{Self, {average_is, Res}}` to `Pid`, where `Self` is the server's process id, and `Res` is the average (arithmetic mean) of the numbers in `LON`. In your solution you can use the arithmetic operator `/` for division and the library function `lists:sum/1`, which returns the sum of a list of numbers. The following are tests.

```
-module(average_tests).
-export([main/0]).
-import(average,[start/0]).
-import(testing,[dotests/2,eqTest/3]).
-import(floattesting,[withinTest/3]).  % withinTest checks for approximate equality
main() -> dotests("average_tests $Revision: 1.1 $", tests()).
tests() ->
    AS = start(),
    [withinTest(compute_average(AS, [1]),"~=~",1.0),
     withinTest(compute_average(AS, [5,6,7]),"~=~",6.0),
     withinTest(compute_average(AS, [100,5,6,7]),"~=~",29.5),
     withinTest(compute_average(AS, [1.0,10.0,20.0,30.0,40.0]),"~=~",20.2),
     withinTest(compute_average(AS, [3.14,9.86,21.5,32.7,88.514]),"~=~",31.1428),
     withinTest(compute_average(AS, [100.0,100.1,100.2]),"~=~",100.1) ].
%% helper for testing, NOT for you to implement.
-spec compute_average(AS::pid(), LON::[number()]) -> number().
compute_average(AS, LON) ->
    AS ! {self(), {average, LON}},
    receive {AS, {average_is, Res}} -> Res end.
```

5. (20 points) [UseModels] In an Erlang module named `catalogserver`, write a function `start/0`, which creates a catalog server and returns its process id. A server created by `catalogserver:start()` keeps track of a mapping from keys to values. The keys and values are both Erlang values (of any type). The server responds to two types of messages:

   - `{Pid, {associate, Key, Value}}`, where `Pid` is the sender's process id, and `Key` and `Value` are both Erlang values. This message causes the server to remember that `Key` maps to `Value` (and forget any previous such association for `Key`). The server responds by sending to `Pid` a message of the form `{SPid, ok}`, where `SPid` is the server's process id.
   - `{Pid, {lookup, Key}}`, where `Pid` is the sender's process id. The server responds by sending a message to `Pid` of the form `{SPid, {value_is, Value}}`, where `Value` is the value that the server is remembering as the value that `Key` maps to, and `undefined` if the server is not currently remembering a value that `Key` maps to. Again, `SPid` is the sever's own process id.

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

```
-module(catalogserver_tests).
-import(catalogserver,[start/0]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0, associate/3, lookup/2]).
main() -> dotests("catalogserver_tests $Revision: 1.3 $", tests()).
tests() -> compile:file(catalogserver),
    CS = start(), C2 = start(),  % make 2 catalog servers
    [eqTest(lookup(CS, take),"==",undefined),
     eqTest(lookup(C2, quail),"==",undefined),
     begin associate(CS, take, took),
           associate(CS, keep, kept),
           eqTest(lookup(CS,take),"==",took)
     end,
     eqTest(lookup(CS,keep),"==",kept),
     begin associate(C2, quail, quails),
           associate(C2, duck, ducks),
           associate(C2, owl, owls),
           associate(C2, crow, crows),
           associate(C2, hawk, hawks),
           eqTest(lookup(C2,quail),"==",quails)
     end,
     eqTest(lookup(C2,owl),"==",owls),
     eqTest(lookup(C2,duck),"==",ducks),
     eqTest(lookup(CS,duck),"==",undefined),
     begin associate(CS, take,taking),
           associate(CS, keep,keeping),
           associate(C2, crow, murder),
           associate(C2, quail, covey),
           associate(C2, owl, parliment),
           eqTest(lookup(CS, take), "==", taking)
     end,
     eqTest(lookup(C2,owl), "==", parliment)      ].
% functions to help with testing, NOT for you to implement
-spec associate(CS::pid(), Key::any(), Value::any()) -> ok.
associate(CS, Key, Value) ->
    CS ! {self(), {associate, Key, Value}},
    receive {CS, ok} -> ok after 1000 -> wrong end.
-spec lookup(CS::pid(), Key::any()) -> any().
lookup(CS, Key) ->
    CS ! {self(), {lookup, Key}},
    receive {CS, {value_is, Value}} -> Value after 1000 -> wrong end.
```

Please write your answer below, completing this module.

```
-module(catalogserver).
```

6. (20 points) [UseModels] In Erlang, write a "shared variable" server; this server acts like a single variable shared between all the processes in an Erlang program. The server's state is a value, which can be manipulated by sending the server messages containing functions. These functions transform the server's state from one value to another. You will write a function `start/1`, which takes an initial value for the state, and creates the server, returning its process id. The server responds to messages of the following forms:

   - `{Pid, {run, F}}`, where `Pid` is the process id of the sender, and `F` is a function. The server runs the function by passing it the current state's value, and obtaining a new state value. The response to this message is of the form `{SPid, {result, NV}}`, where `SPid` is the server's own process id, and `NV` is the new state value obtained from running `F`. The server continues with `NV` as its new state.
   - `{Pid, see}`, where `Pid` is the process id of the sender. The server responds by sending a message of the form `{SPid, Value}` to `Pid` where `SPid` is the server's own process id and `Value` is the value that is the server's current state. The server continues with an unchanged state.

   The following are tests, written using the homework's testing module.

```
-module(sharedvarserver_tests).
-import(sharedvarserver,[start/1]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0, run/2, see/1]).
main() -> dotests("sharedvarserver_tests $Revision: 1.2 $", tests()).
tests() -> compile:file(sharedvarserver),
    S1 = start(0), S2 = start([one]),  % make 2 sharedvar servers
    [eqTest(run(S1, fun(X) -> X+1 end),"==",1),
     eqTest(see(S1),"==",1),
     eqTest(run(S1, fun(X) -> X+4 end),"==",5),
     eqTest(run(S1, fun(X) -> X*X end),"==",25),
     eqTest(run(S1, fun(X) -> X*X end),"==",625),
     eqTest(see(S1),"==",625),
     eqTest(see(S2),"==",[one]),
     eqTest(run(S2, fun(Hist) -> [two|Hist] end),"==",[two,one]),
     eqTest(run(S2, fun(Hist) -> [three|Hist] end),"==",[three,two,one]),
     eqTest(see(S2),"==",[three,two,one]),
     eqTest(run(S2, fun(Hist) -> [4|Hist] end),"==",[4,three,two,one]),
     eqTest(run(S2, fun([N|Hist]) -> [N+1|[N|Hist]] end),"==",[5,4,three,two,one])   ].
% functions to help with testing, NOT for you to implement
-spec run(S::pid(), F::fun((State::any()) -> any())) -> any().
run(S, F) ->
    S ! {self(), {run, F}},
    receive {S, {result, Val}} -> Val after 1000 -> wrong end.
-spec see(S::pid()) -> any().
see(S) ->
    S ! {self(), see},
    receive {S, Value} -> Value after 1000 -> wrong end.
```

   There is space for your answer on the next page

Please write your answer below, completing the following module.

```
-module(sharedvarserver).
```

7. (20 points) [UseModels] In Erlang, write a "notification" server; this server tracks a list of processes to be notified in case of an event occurring, and supports announcement of such events. The server's state consists of the name of the event and a list of process ids for the event's observers. You will write a function `start/1`, which takes the event name (an atom), creates a server for notifying observers about that event, and returns the server's process id. The server responds to messages of the following forms:

   - **`{Pid, register}`**, which records the process id, `Pid`, as an observer in the server's state, and responds by sending the message **`{SPid, registered}`** back to `Pid`, where `SPid` is the server's own process id.
   - **`{Pid, observers}`**, which sends a message of the form **`{SPid, Observers}`** to `Pid`, where `SPid` is the server's own process id, and `Observers` is a list of the process ids of all the registered observers. The state of the server is unchanged.
   - **`{Pid, announce}`**, which notifies each observer in the server's list of observers by sending it a message of the form **`{SPid, {event, EName}}`**, where `SPid` is the server's own process id and `EName` is the name of the event (as given to `start/1`). It then responds by sending the message **`{SPid, announced}`** back to `Pid`. The server's state is unchanged.

   There are tests below and space for your answer on the next page.

```
-module(notifier_tests).
-import(notifier,[start/1]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0]).
main() -> compile:file(notifier), dotests("notifier_tests $Revision: 1.3 $", tests()).
tests() -> N1 = start(e1), N2 = start(e2),
    W1 = makeWorker(N1), W2 = makeWorker(N1), W3 = makeWorker(N2),
    lists:foreach(fun(W) -> receive {W, registered} -> ok end end, [W1,W2,W3]),
    [eqTest(observers(N1),"==",[W2,W1]), eqTest(observers(N2),"==",[W3]),
     eqTest(getCount(W1),"==",0), eqTest(getCount(W2),"==",0),
     eqTest(getCount(W3),"==",0),
     eqTest(begin announce(N1), getCount(W1) end,"==",1), eqTest(getCount(W2),"==",1),
     eqTest(begin announce(N1), getCount(W1) end,"==",2),
     eqTest(getCount(W2),"==",2), eqTest(getCount(W3),"==",0),
     eqTest(begin announce(N2), getCount(W3) end,"==",1), eqTest(getCount(W2),"==",2),
     eqTest(begin announce(N1), getCount(W1) end,"==",3),
     eqTest(getCount(W2),"==",3)                          ].
% functions to help with testing, NOT for you to implement
-spec observers(NS::pid()) -> [pid()].
observers(NS) -> NS ! {self(), observers},
                 receive {NS, LOP} -> LOP end.
-spec announce(NS::pid()) -> ok.
announce(NS) -> NS ! {self(), announce},
                receive {NS, announced} -> ok end.
% A worker helps with testing
-spec makeWorker(NSPid::pid()) -> pid().
makeWorker(NSPid) -> TPid = self(),
                     spawn(fun() -> workerInit(TPid, NSPid) end).
workerInit(TPid,NSPid) -> NSPid ! {self(), register},
                          receive {NSPid, registered} -> TPid ! {self(), registered} end,
                          worker(TPid,NSPid,0).
worker(TPid,NSPid,Count) ->
    receive {TPid, count} -> TPid ! {self(), Count},
                             worker(TPid,NSPid,Count);
            {NSPid, {event, _EN}} -> worker(TPid, NSPid, Count+1)
    end.
getCount(WPid) -> WPid ! {self(), count},
                  receive {WPid, Count} -> Count end.
```

Please write your answer below, to complete this module.

```
-module(notifier).
```