Fall, 2013 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 6 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 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/OTP, then you must write them into your test. (Note that you can use built-in functions 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`, `math:pow/2`, `ets:new/2`, `ets:lookup/2`, `ets:insert/2`, `ets:update_counter/3`, `ets:tab2list/1`, etc.)

For Grading

Question:	1	2	3	4	5	6	Total
Points:	8	12	10	20	25	25	100
Score:							

1. [Concepts] This is a question about pattern matching in Erlang. Consider the following Erlang code.

```
-module(patternmatch).
-export([patternmatch/2, a/0, b/0, c/0, d/0]).
patternmatch(list1, list2) -> 1;
patternmatch(n, m) -> 2;
patternmatch(X, X) -> 3;
patternmatch([elem|elems], [x|xs]) -> 4;
patternmatch([Elem|Elems], [Elem|Elems]) -> 5;
patternmatch([_A|_As], [_B|_Bs]) -> 6;
patternmatch({_X}, _Y) -> 7;
patternmatch({A,B}, {B,A}) -> 8;
patternmatch({_A,_B}, {_C,_D}) -> 9;
patternmatch(_X, _Y) -> 10.
% Functions for each of the parts of this question:
a() -> patternmatch({a_atom}, another).
b() -> patternmatch([elem,mints], [candy,crush]).
c() -> patternmatch({varied,flowers}, {flowers,varied}).
d() -> patternmatch([cross, bow, service], [cross, bow, service]).
```

   (a) (2 points) Given the above code, what is a result of the call `patternmatch:a()` ?

   (b) (2 points) Given the above code, what is a result of the call `patternmatch:b()` ?

   (c) (2 points) Given the above code, what is a result of the call `patternmatch:c()` ?

   (d) (2 points) Given the above code, what is a result of the call `patternmatch:d()` ?

2. (12 points) [UseModels] In Erlang, without using any library functions or list comprehensions, and without using ++, write a function `lookup/2`, which has the following type specification.

```
-spec lookup([{Key,Value}], Key) -> [Value].
```

A call such as `lookup(AList, Key)` returns a list containing all of the second elements of tuples in `AList` whose first elements are the same as `Key`. The following are tests using the homework's testing module.

```
% $Id: lookup_tests.erl,v 1.2 2013/12/05 13:24:32 leavens Exp leavens $
-module(lookup_tests).
-import(lookup,[lookup/2]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0]).
main() -> dotests("lookup_tests $Revision: 1.2 $", tests()).
tests() ->
    StatePopulations = [{cal,38},{tex,26},{ny,20},{fla,19},{ill,13},{penn, 13},{ohio,12}],
    LetterNums = [{a,2},{a,3},{a,0},{b,5},{b,2},{c,3},{a,6}],
    [eqTest(lookup([], 3),"==",[]),
     eqTest(lookup([{3,2}], 3),"==",[2]),
     eqTest(lookup([{4,5},{3,2}], 3),"==",[2]),
     eqTest(lookup([{3,4},{4,5},{3,2}], 3),"==",[4,2]),
     eqTest(lookup(StatePopulations, fla),"==",[19]),
     eqTest(lookup(StatePopulations, mich),"==",[]),
     eqTest(lookup(StatePopulations, cal),"==",[38]),
     eqTest(lookup(StatePopulations, cal),"==",[38]),
     eqTest(lookup(LetterNums, a),"==",[2,3,0,6]),
     eqTest(lookup(LetterNums, b),"==",[5,2])].
```

3. (10 points) [UseModels] In Erlang, write a stateless server in a module named `power`. This server responds to messages of the form `{Pid, power, N, M}`, where `Pid` is the sender's process id, `N` and `M` are non-negative integers. When the server receives such a message, it responds by sending a message of the form `{answer, Res}` to `Pid`, where `Res` is N^M, that is `N` raised to the `M`th power. In your solution you can use the library function `math:pow`, which is defined so that `math:pow(N,M)` returns N^M. The following are tests.

```
% $Id$
-module(power_tests).
-export([main/0]).
-import(power,[start/0]).
-import(testing,[dotests/2,eqTest/3]).
main() -> dotests("power_tests $Revision$", tests()).
tests() ->
    PS = start(),
    [eqTest(compute_power(PS, 0,0),"==",1),
     eqTest(compute_power(PS, 22,0),"==",1),
     eqTest(compute_power(PS, 1,1),"==",1),
     eqTest(compute_power(PS, 6,1),"==",6),
     eqTest(compute_power(PS, 2,3),"==",8),
     eqTest(compute_power(PS, 3,3),"==",27),
     eqTest(compute_power(PS, 3,2),"==",9),
     eqTest(compute_power(PS, 5,2),"==",25)].
%% helper for testing, NOT for you to implement.
compute_power(PS, N, M) ->
    PS ! {self(), power, N, M},
    receive {answer, Res} -> Res end.
```

4. (20 points) [UseModels] In an Erlang module named `logger`, write a function `start/0`, which creates a log server and returns its process id. A server created by `logger:start()` keeps track of a list of log entries. The entries are simply Erlang values (of any type). The server responds to two types of messages:

   - `{Pid, log, Entry}`, where `Pid` is the sender's process id, and `Entry` is a value. This message causes the server to remember `Entry` in its list. The server responds by sending to `Pid` a message of the form `{SPid, logged}`, where `SPid` is the server's process id.
   - `{Pid, fetch}`, where `Pid` is the sender's process id. The server responds by sending a message to `Pid` of the form `{SPid, log_is, Entries}`, where `SPid` is the server's process id, and `Entries` is a list of all the entries that have been previously received by the log server (`SPid`), in the order in which they were received (oldest first).

   The next page contains tests, written using the homework's testing module.

```
% $Id: logger_tests.erl,v 1.1 2013/12/02 22:13:53 leavens Exp leavens $
-module(logger_tests).
-import(logger,[start/0]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0,logThenFetch/2,log/2,fetch/1]).
main() -> dotests("logger_tests $Revision: 1.1 $", tests()).
tests() ->
    L1 = logger:start(),
    L2 = logger:start(),
    [eqTest(fetch(L1),"==",[]),
     eqTest(fetch(L2),"==",[]),
     eqTest(logThenFetch(L1,[starting,middle,ending]),"==",[starting,middle,ending]),
     eqTest(logThenFetch(L2,[start,between,last]),"==",[start,between,last]),
     eqTest(logThenFetch(L1,[final]),"==",[starting,middle,ending,final]),
     eqTest(logThenFetch(L1,[really]),"==",[starting,middle,ending,final,really]),
     eqTest(logThenFetch(L2,[ultimate]),"==",[start,between,last,ultimate]),
     eqTest(fetch(L1),"==",[starting,middle,ending,final,really])
    ].
% helpers for testing (client functions), NOT for you to implement
logThenFetch(Logger, []) ->
    fetch(Logger);
logThenFetch(Logger, [Entry|Entries]) ->
    log(Logger, Entry),
    logThenFetch(Logger, Entries).
log(Logger, Entry) ->
    Logger ! {self(), log, Entry},
    receive {Logger, logged} -> logged end.
fetch(Logger) ->
    Logger ! {self(), fetch},
    receive {Logger, log_is, Entries} -> Entries end.
```

5. (25 points) [Concepts] [UseModels] In Erlang, write a barrier synchronization server. In barrier synchronization, a group of processes wait until all of them have are done executing up to a certain point (the barrier). You will write a function `start/1`, which takes a positive integer, which is the size of the group of processes, and creates a barrier synchronization server, returning its process id. The barrier synchronization server tracks in its state the number of processes that are still running (are not yet done), and the process ids of all processes that have reached the barrier. The server responds to messages of the following forms:

   - `{Pid, done}`, where `Pid` is the process id of the sender. The server responds sending a message to `Pid` of the form `{SPid, ok}`, where `SPid` is the server's own process id. What it does next depends on whether `Pid` was the last process in the group to finish. If `Pid` is the last process in the group to be done (i.e., if there are no other running processes), then `Pid` and all the processes that have previously sent such a `done` message are sent a message of the form `{SPid, continue}`, which lets them continue past the barrier. If there are other running processes, then the server just remembers `Pid` in the list of processes that have reached the barrier (and are thus waiting).
   - `{Pid, how_many_running}`, where `Pid` is the process id of the sender. The server responds by sending a message to `Pid` of the form `{SPid, number_running_is, Running}`, where `SPid` is the server's own process id and `Running` is the number of processes in the group that have not yet reached the barrier. The server continues with an unchanged state.

   You can assume that each process in the group only sends a `done` message to the server once. (But despite this, the server does not "reset" or start over, but keeps running once all the processes in the group are done.) The next page contains tests, written using the homework's testing module.

```
% $Id: barrier_tests.erl,v 1.1 2013/12/04 02:35:23 leavens Exp leavens $
-module(barrier_tests).
-export([main/0]).
-import(barrier,[start/1]).
-import(testing,[dotests/2,eqTest/3]).
-import(lists,[map/2,foreach/2]).
main() -> dotests("barrier_tests $Revision: 1.1 $", tests()).
tests() ->
    Br = barrier:start(4),
    Workers = map(workerCreator(Br),[1,2,3,4]),  %% start the workers
    [eqTest(num_running(Br), "==", 4),
     begin
         send_finish(hd(Workers)),
         eqTest(num_running(Br), "==", 3)
     end,
     eqTest(num_released(), "==", 0),
     begin
         send_finish(hd(tl(Workers))),
         eqTest(num_running(Br), "==", 2)
     end,
     eqTest(num_released(), "==", 0),
     begin
         foreach(fun(W) -> send_finish(W) end, tl(tl(Workers))),
         eqTest(num_running(Br), "==", 0)
     end,
     eqTest(num_released(), "==", 4)
    ].
%% helpers for testing (client functions), NOT for you to implement
workerCreator(Barrier) -> fun(_Num) ->
                              TPid = self(),
                              spawn(fun() -> worker_fun(Barrier, TPid) end)
                          end.
%% worker_fun acts under control of the testing code's finish message, telling
%% it when the barrier acknowledges its done message and when it's released.
worker_fun(Barrier, TestingPid) ->
    receive {TestingPid, finish} -> ok end,
    Barrier ! {self(), done},
    receive {Barrier, ok} -> TestingPid ! {self(), ok} end,
    receive {Barrier, continue} -> ok end,
    TestingPid ! {self(), released}.
%% send finish to the worker process and get an ack (for testing purposes).
send_finish(Pid) ->
    Pid ! {self(), finish},
    receive {Pid, ok} -> ok end.
%% How many processes are not finished (still running)?
num_running(Barrier) ->
    Barrier ! {self(), how_many_running},
    receive {Barrier, number_running_is, Num} -> Num end.
%% How many released messages have been received by the testing process?
num_released() -> num_released(0).
num_released(N) -> receive {_Pid, released} -> num_released(N+1)
                   after 0 -> N
                   end.
```

6. (25 points) [UseModels] In this problem you will write an election server and two client functions. The three functions you are to implement are the following.

```
-spec start() -> pid().
```

The `start/0` function which creates a new election server and returns its process id.

```
-spec vote(ES::pid(), Candidate::atom()) -> ok.
```

The `vote/2` function takes as arguments the process id of an election server, and a candidate's name (an atom). By sending messages to the election server, this function casts a single vote the candidate. After the server has received the vote, this function returns the atom ok to the caller.

```
-spec results(ES::pid()) -> [{atom(), non_neg_integer()}].
```

The `results/1` function takes the election server's process id as an argument. It returns a list of pairs of the form `{Candidate, Vote}`, where `Candidate` is a candidate's name, and `Vote` is the total number of votes cast for the candidate. The returned list sorted in the order given by `lists:sort/1`, i.e., in non-decreasing order by the candidate's name. This function does not change the state of the server.

There are tests on the following page.

```
% $Id: electionserver_tests.erl,v 1.1 2013/12/04 06:25:43 leavens Exp leavens $
-module(electionserver_tests).
-import(electionserver,[start/0, vote/2, results/1]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0]).
main() -> dotests("electionserver_tests $Revision: 1.1 $", tests()).
tests() ->
    ES = start(), E2 = start(),
    [eqTest(results(ES),"==",[]),
     begin vote(ES, clinton), vote(ES, clinton), vote(ES, christy),
           eqTest(results(ES),"==",[{christy,1},{clinton,2}])
     end,
     begin vote(ES, christy), vote(ES, christy), vote(ES, christy),
           vote(ES, abel), vote(ES, baker), vote(ES,clinton),
           eqTest(results(ES),"==",[{abel,1},{baker,1},{christy,4},{clinton,3}])
     end,
     eqTest(results(E2),"==",[]),
     begin vote(E2, ucf), vote(E2, usf), vote(E2, fiu), vote(E2, uf),
           vote(E2, ucf), vote(E2, ucf), vote(E2, fsu),
           eqTest(results(E2),"==",[{fiu,1},{fsu,1},{ucf,3},{uf,1},{usf,1}])
     end
    ].
```