Spring, 2013 Name: _____

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

COP 4020 — Programming Languages I

Test on the Message Passing Model in Erlang

Special Directions for this Test

This test has 5 questions and pages numbered 1 through 8.
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:foldr/3`, `lists:filter/2`, `lists:member/2`, `lists:reverse/1`, `lists:sublist/2`, etc.)

For Grading

Question:	1	2	3	4	5	Total
Points:	8	12	25	30	25	100
Score:						

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

```
-module(patternmatch).
-export([patternmatch/1, a/0, b/0, c/0, d/0]).
patternmatch(lst) -> 1;
patternmatch(nil) -> 2;
patternmatch([_Lst]) -> 3;
patternmatch([[_Lst]]) -> 6;
patternmatch([x|xs]) -> 4;
patternmatch([_X|_Xs]) -> 5;
patternmatch(_Number) -> 7.
% Functions for each of the parts of this question:
a() -> patternmatch([atoms, are, here]).
b() -> patternmatch(an_atom).
c() -> patternmatch(lst).
d() -> patternmatch([[some, nested], [lists]]).
```

   (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 `map2/3`, which has the following type specification.

```
-spec map2(fun((A,B) ->C), [A], [B]) -> [C].
```

   A call such as `map2(F,As,Bs)` returns a list of `N` elements, where `N` is the length of the shorter of `As` and `Bs`, and where the ith element of the result is the result of applying `F` to the ith element of `As` and the ith element of `Bs`. The following are tests using the homework's testing module.

```
% $Id: map2_tests.erl,v 1.1 2013/04/21 22:15:13 leavens Exp $
-module(map2_tests).
-import(map2,[map2/3]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0]).
main() -> dotests("map2_tests $Revision: 1.1 $", tests()).
tests() ->
    [eqTest(map2(fun(A,B) -> A+B end, [], []),"==",[]),
     eqTest(map2(fun(_A,_B) -> 3 end, [1,2,3], []),"==",[]),
     eqTest(map2(fun(A,B) -> A*B end, [], [1,2,3]),"==",[]),
     eqTest(map2(fun(A,B) -> A+B end, [4,5,6], [1,2,3]),"==",[4+1,5+2,6+3]),
     eqTest(map2(fun(A,B) -> {A,B} end, [a,b,c,d,e], [5,9,2,4,42]),
            "==", [{a,5}, {b,9}, {c,2}, {d,4}, {e,42}]),
     eqTest(map2(fun(X,Y) -> X == Y end, [x,y,z,z,y,foo],[y,x,a,z,q,foo]),
            "==", [false, false,false,true,false,true])    ].
```

3. (25 points) [UseModels] In Erlang, write a function `start/1`, which creates a semaphore server and returns its process id. A server created by `semaphore:start(N)`, where `N` is a positive integer, keeps track of how many of the `N` resources are available, and also maintains as its state a list of process ids waiting to obtain a resource. The server responds to two types of messages:

   - **`{Pid, lock}`**, where `Pid` is the process id of a process that wants one (1) of the resources. If the semaphore server has a resource available, then it grants it to `Pid` by sending to `Pid` the message **`{MyPid, go_ahead}`**, where `MyPid` is the process id of the semaphore server, and in this case it decrements the number of resources available. Otherwise, if no resource is available, then it adds `Pid` to the list of process ids waiting to obtain a resource.
   - **`{Pid, done}`**, where `Pid` is the process id of a process that already has a resource (by virtue of being previously sent the **`{Pid, go_ahead}`** message by the semaphore server). When the server receivs the message **`{Pid, done}`**, it sends a reply to `Pid` of the form **`{MyPid, thanks}`**, where `MyPid` is the server's own process id. The server then does one of two things, depending on the list of waiting processes: (1) if the list of waiting processes is empty, it increments the number of resources available and continues, (2) otherwise, if the list of waiting processes is not empty, then the server gives the resource to the process that has been waiting for it the longest time, by sending it the message **`{MyPid, go_ahead}`**, and continues by removing that process from the list of waiting processes (without changing the number of available resources). To simplify the problem, it is not necessary to check that `Pid` is actually a process id that is holding a resource.

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

```
% $Id: semaphore_tests.erl,v 1.1 2013/04/21 22:15:13 leavens Exp $
-module(semaphore_tests).
-import(semaphore,[start/1]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0,clientstart/2,statusloop/3]).
main() -> dotests("semaphore_tests $Revision: 1.1 $", tests()).
tests() ->
    S1 = semaphore:start(1),
    S2 = semaphore:start(2),
    Dijkstra = startclient(S1),
    Hoare = startclient(S2),
    BrinchHansen = startclient(S2),
    Liskov = startclient(S1),
    [eqTest(status(Dijkstra), "==", locked),
     eqTest(status(Liskov), "==", waiting),
     eqTest(finish(Dijkstra), "==", sent_done),
     eqTest(status(Dijkstra), "==", done),
     eqTest(status(Liskov), "==", locked),
     eqTest(status(Hoare), "==", locked),
     eqTest(status(BrinchHansen), "==", locked),
     eqTest(finish(BrinchHansen), "==", sent_done),
     eqTest(status(BrinchHansen), "==", done),
     eqTest(finish(Hoare), "==", sent_done),
     eqTest(status(Hoare), "==", done)
    ].
% helpers for testing (client functions), NOT for you to implement
startclient(Semaphore) ->
    Pid = clientaction(Semaphore),
    receive {Pid, sent_lock} -> Pid end.
clientaction(Semaphore) -> spawn(?MODULE,clientstart,[Semaphore,self()]).
clientstart(Semaphore,TestPid) ->
    Semaphore ! {self(), lock},
    TestPid ! {self(), sent_lock},
    statusloop(Semaphore,TestPid,waiting).
statusloop(Semaphore,TestPid,Status) ->
    receive
        {Semaphore, go_ahead} -> statusloop(Semaphore, TestPid, locked);
        {Semaphore, thanks} -> statusloop(Semaphore, TestPid, done);
        {TestPid, status} -> TestPid ! {self(), status_is, Status},
                             statusloop(Semaphore, TestPid, Status);
        {TestPid, finish} when Status == locked ->
            Semaphore ! {self(), done},
            TestPid ! {self(), sent_done},
            statusloop(Semaphore, TestPid, Status)
    end.
status(Client) ->
    Client ! {self(), status},
    receive {Client, status_is, Status} -> Status end.
finish(Client) ->
    Client ! {self(), finish},
    receive {Client, sent_done} -> sent_done end.
```

4. (30 points) [Concepts] [UseModels] In Erlang, write a function `start/0`, which creates a compositor server and returns its process id. A compositor server maintains as its state a one-argument function, which is initially the identity function. The server responds to messages of the following forms:

   - **{**`Pid, apply_to, Value`**}**, where `Pid` is the process id of the sender, and `Value` is some value. The server responds by applying the function in its state to `Value` and sending to `Pid` the message **{**`MyPid, value_is, Result`**}**, where `MyPid` is the compositor server's own process id, and `Result` is the value returned by the application of the server's remembered function to `Value`.
   - **{**`Pid, compose_with, NewFun`**}**, where `Pid` is the process id of the sender, and `NewFun` is a one-argument function. The server responds by sending the message **{**`MyPid, composed`**}** to `Pid`, where `MyPid` is the server's own process id. The server continues with the composition of `NewFun` and the function it remembered (so that `NewFun` is applied after the remembered function) the next time the compositor server is sent an `apply_to` message.
   - **{**`Pid, start_over_with, NewFun`**}** where `Pid` is the process id of the sender, and `NewFun` is a one-argument function. The server responds by sending the message **{**`MyPid, started_over`**}** to `Pid`, where `MyPid` is the server's own process id. The server continues with `NewFun` replacing the previously remembered composition of functions it was remembering.

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

```
% $Id: compositor_tests.erl,v 1.2 2013/04/24 21:51:28 leavens Exp leavens $
-module(compositor_tests).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0, applyTo/2, composeWith/2, startOver/2]).
main() -> dotests("compositor_tests $Revision: 1.2 $", tests()).
tests() ->
    Comp = compositor:start(),
    [eqTest(applyTo(Comp, 42), "==", 42),
     eqTest(applyTo(Comp, atom), "==", atom),
     eqTest(composeWith(Comp, fun(X) -> 3*X*X end), "==", ok),
     eqTest(applyTo(Comp, 10), "==", 300),
     eqTest(applyTo(Comp, 11), "==", 363),
     eqTest(composeWith(Comp, fun(Y) -> 5+Y end), "==", ok),
     eqTest(applyTo(Comp, 10), "==", 305),
     eqTest(applyTo(Comp, 11), "==", 368),
     eqTest(startOver(Comp, fun(X) -> X == turkey end), "==", ok),
     eqTest(applyTo(Comp, turkey), "==", true),
     eqTest(applyTo(Comp, cheese), "==", false)
    ].
% helpers for testing (client functions), NOT for you to implement
applyTo(Comp, Value) ->
    Comp ! {self(), apply_to, Value},
    receive {Comp, value_is, Result} -> Result end.
composeWith(Comp, NewFun) ->
    Comp ! {self(), compose_with, NewFun},
    receive {Comp, composed} -> ok end.
startOver(Comp, NewFun) ->
    Comp ! {self(), start_over_with, NewFun},
    receive {Comp, started_over} -> ok end.
```

5. (25 points) [UseModels] In this problem you will write a server and two client functions to implement a simple version of Twitter™'s server. The three functions you are to implement are the following.

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

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

```
-spec tweet(pid(), string()) -> tweeted.
```

The `tweet/2` function takes as arguments the process id of a twitter server, and a string (the tweet's text). By sending messages to the twitter server, this function arranges for the server to remember the string as the latest tweet. After the server has received the tweeted string and responded to this function, then this function returns the atom `tweeted` to the caller.

```
-spec fetch(pid(), non_neg_integer()) -> [string()].
```

The `fetch/2` function takes the twitter server's process id and a non-negative integer `N` as arguments. It fetches and returns the `N` most recent tweeted strings that the server is remembering as a list. However, if the server is currently remembering fewer than `N` tweeted strings, then it returns all of the tweeted strings that the server is remembering. The order of the returned list is from the most recent to the oldest of the `N` tweeted strings. The server's list of remembered tweeted strings is *not* changed.

There are tests on the following page, written using the homework's testing module.

```
% $Id: twitterserver_tests.erl,v 1.2 2013/04/24 21:51:28 leavens Exp leavens $
-module(twitterserver_tests).
-import(twitterserver,[start/0, tweet/2, fetch/2]).
-import(testing,[dotests/2,eqTest/3]).
-export([main/0]).
main() -> dotests("twitterserver_tests $Revision: 1.2 $", tests()).
tests() ->
    TS = twitterserver:start(),
    [eqTest(tweet(TS, "Starting to use twitter"), "==", tweeted),
     eqTest(tweet(TS, "Where's the send button?"), "==", tweeted),
     eqTest(fetch(TS, 1), "==", ["Where's the send button?"]),
     eqTest(fetch(TS, 2), "==", ["Where's the send button?", "Starting to use twitter"]),
     eqTest(tweet(TS, "Chirp!"), "==", tweeted),
     eqTest(tweet(TS, "Caw! Caw!"), "==", tweeted),
     eqTest(tweet(TS, "Look out for that cat!"), "==", tweeted),
     eqTest(fetch(TS, 0), "==", []),
     eqTest(fetch(TS, 1), "==", ["Look out for that cat!"]),
     eqTest(fetch(TS, 5), "==", ["Look out for that cat!", "Caw! Caw!", "Chirp!",
                                 "Where's the send button?", "Starting to use twitter"]),
     eqTest(fetch(TS, 10), "==", ["Look out for that cat!", "Caw! Caw!", "Chirp!",
                                  "Where's the send button?", "Starting to use twitter"]),
     eqTest(tweet(TS, "My poor birdie!"), "==", tweeted),
     eqTest(tweet(TS, "It didn't see dat putythat!"), "==", tweeted),
     eqTest(fetch(TS, 3), "==", ["It didn't see dat putythat!",
                                 "My poor birdie!",
                                 "Look out for that cat!"])
    ].
```