CS 511: Homework Assignment 3
Due: 6 April, 11:59 PM

1 Assignment Policies

Under absolutely no circumstances code can be exchanged between students.
Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be reused. Violations will be penalized appropriately.

Late Policy. Late submissions are allowed with a penalty of 2 points per hour past the deadline.

2 Assignment

The aim of this assignment is to get familiar with the functional fragment of Erlang[1] and to learn how to work with some of the data structures provided by the language; notably lists, maps, and records.
You are asked to implement various operations that provide support to a *shipping company*. The shipping company administers (cargo) *ships*. Ships transport *containers* and navigate from one *port* to another. Each port has a number of *docks* where the ships may load and unload. Containers can either be in transit on a ship or in a port (in the latter case, no distinction is made regarding the specific dock).
You are supplied with two files:

- `shipping.hrl`. The Erlang header file for the assignment. It contains all the record declarations you should need to complete this assignment. These are described in further detail below.

- `shipping.erl`. The module file that includes all the functions that need to be completed. This is the file you will be editing.

---

[1]Disregarding process spawning and message passing. The latter will be addressed in an upcoming assignment.

2.0.1 Shipping Header File (shipping.hrl)

- `Ship` This record contains the fields:

  - `name` a string value
  - `id` a unique integer value
  - `container_cap` the maximum number of containers that a ship can hold

- `Container` This record contains the fields:

  - `id` a unique integer value
  - `weight` any positive integer value

- `Port` This record contains the fields:

  - `name` a string value
  - `id` a unique integer value
  - `docks` a list of docks for the port. A dock can be represented by an unique integer or a character and is only required to be unique at a given port (i.e. **Port "New York", Dock 'A'** and **Port "Los Angeles", Dock 'A'** are both valid)
  - `container_cap` the maximum number of containers that a port can hold

- `Shipping State` This record contains the fields:

  - `ships` a list of the ship records currently in the system
  - `containers` a list of the container records currently in the system
  - `ports` a list of the port records currently in the system
  - `ship_locations` a tuple containing a `port_id`, `dock_id`, and `ship_id` (i.e. (1,'A',3)) if **port 1**, **dock 'A'** contains **ship 3**
  - `ship_inventory` a map that takes a **ship id** and maps it to the **list of containers ids** on that ship.
  - `port_inventory` a map that takes a **port id** and maps it to the **list of containers ids** at that port.

3 Shipping Module

We next describe each of the functions you are asked to implement. All of them have to be defined in `shipping.erl`.

To illustrate some of the functions described below we will use a sample shipping company **ShipCo**. The shipping state of **ShipCo** may be obtained by calling `shipping:shipco()`. It is provided for you in the stub. Before you try out the examples below, remember to compile and then load the record definitions into the interpreter, as follows:

```
1> c(shipping).
shipping.erl:2: Warning: export_all flag enabled - all functions will be exported
{ok,shipping}
2> rr(shipping).
[container,port,ship,shipping_state]
```

1. get_ship(Shipping_State, Ship_ID)
   This method returns a ship record for the given id. For example,
   shipping:get_ship(shipping:shipco(),1) will return the ship whose id is 1. If the
   ship does not exist, it should throw the exception **error**[2].

   ```
   3> shipping:get_ship(shipping:shipco(),1).
   #ship{id = 1,name = "Santa Maria",container_cap = 20}
   4> shipping:get_ship(shipping:shipco(),7).
   ** exception throw: error
      in function shipping:get_ship/2 ...
   ```

2. get_container(Shipping_State, Container_ID)
   This method returns a container record for the given id. It should throw the exception
   error, if the container id does not exist. For example:

   ```
   5> shipping:get_container(shipping:shipco(),4).
   #container{id = 4,weight = 62}
   6> shipping:get_container(shipping:shipco(),47).
   ** exception throw: error
      in function shipping:get_container/2 ...
   ```

3. get_port(Shipping_State, Port_ID)
   This method returns a port records for the given id. It should throw the exception
   error, if the port id does not exist. For example:

   ```
   7> shipping:get_port(shipping:shipco(),3).
   #port{id = 3,name = "Miami",
         docks = ['A','B','C','D'],
         container_cap = 200}
   8> shipping:get_port(shipping:shipco(),12).
   ** exception throw: error
      in function shipping:get_port/2 ...
   ```

4. get_occupied_docks(Shipping_State, Port_ID)
   This method returns a list of all the occupied docks for a given port. It returns the
   empty list if the port id does not exist. For example:

   ```
   9> shipping:get_occupied_docks(shipping:shipco(),3).
   ['C']
   10> shipping:get_occupied_docks(shipping:shipco(),23).
   []
   ```

---

[2]Use **throw**(my_exception) to throw an exception called my_exception.

5. `get_ship_location(Shipping_State, Ship_ID)`
   This method returns the location, {`Port_ID, Dock_ID`}, of a given ship. It should throw the exception **error**, if the ship id does not exist. For example:

   ```
   11> shipping:get_ship_location(shipping:shipco(),3).
   {1,'A'}
   12> shipping:get_ship_location(shipping:shipco(),23).
   ** exception throw: error
      in function shipping:get_ship_location/2 ...
   ```

6. `get_container_weight(Shipping_State, Container_IDs)`
   This method returns the total weight of all of the container ids in the list `Container_IDs`. It should throw the exception **error**, if any of the container Ids does not exist.

   ```
   13> shipping:get_container_weight(shipping:shipco(),[3,5]).
   243
   14> shipping:get_container_weight(shipping:shipco(),[89,3,5]).
   ** exception throw: error
       in function  shipping:get_container/2  ...
       ...
   ```

7. `get_ship_weight(Shipping_State, Ship_ID)`
   This method returns the total weight of a ship, measured by the total weight of the ship's containers. It returns the atom **error** if ship Id does not exist.

   ```
   15> shipping:get_ship_weight(shipping:shipco(),2).
   676
   16> shipping:get_ship_weight(shipping:shipco(),22).
   ** exception throw: error
       in function  shipping:get_ship_weight/2
   ```

8. `load_ship(Shipping_State, Ship_ID, Container_IDs)`
   This method returns a shipping state in which the containers in the list `Container_IDs` are moved from a port to the ship with the given `Ship_ID`. Assume `Container_IDs` has no duplicates. You must, however, make sure that all the containers are at the same port as the ship they are loading onto. In the case that `Ship_ID` does not exist or that the loading the ship would put the ship over capacity, throw an **error** exception and do not add any containers to the ship. Also throw an **error** exception if there are container IDs that don't exist in `Shipping_State` or that that aren't in the same port as the ship or if the containers are already on the ship. Here is an example. Notice how containers 16, 18 and 20 have been removed from dock 1 and loaded onto ship 1.

   ```
   10> shipping:load_ship(shipping:shipco(), 1, [16,18,20]).
   {ok,#shipping_state{
      ships = [#ship{id = 1,name = "Santa Maria",container_cap = 20},
               #ship{id = 2,name = "Nina",container_cap = 20},
   ```

4

```
            #ship{id = 3,name = "Pinta",container_cap = 20},
            #ship{id = 4,name = "SS Minnow", container_cap = 20},
            #ship{id = 5,name = "Sir Leaks-A-Lot",container_cap = 20}],
   containers = [...], %% not shown
   ports = [#port{id = 1,name = "New York",
                  docks = ['A','B','C','D'],
                  container_cap = 200},
            #port{id = 2,name = "San Francisco",
                  docks = ['A','B','C','D'],
                  container_cap = 200},
            #port{id = 3,name = "Miami",
                  docks = ['A','B','C','D'],
                  container_cap = 200}],
   ship_locations = [{1,'B',1},
                     {1,'A',3},
                     {3,'C',2},
                     {2,'D',4},
                     {2,'B',5}],
   ship_inventory = #{1 => [14,15,9,2,6,16,18,20],  %% loaded here
                      2 => [1,3,4,13],
                      3 => [],
                      4 => [2,8,11,7],
                      5 => [5,10,12]},
   port_inventory = #{1 => [17,19],  %% removed from port 1
                      2 => [21,22,23,24,25],
                      3 => [26,27,28,29,30]}}}}
> shipping:load_ship(shipping:shipco(), 45, [16,18,20]).
** exception throw: error
   in function  shipping:get_ship/2 ...
```

9. `unload_ship_all(Shipping_State, Ship_ID)`
   This method returns a shipping state in which **all** of the containers that are on a given
   ship are offloaded to the port in which the ship is docked. In the case that offloading
   to a port would put the port over capacity, throw the exception **error** atom and do
   not unload any containers to the port.

```
11> shipping:unload_ship_all(shipping:shipco(), 2).
{ok,#shipping_state{
   ships = [#ship{id = 1,name = "Santa Maria",container_cap = 20},
            #ship{id = 2,name = "Nina",container_cap = 20},
            #ship{id = 3,name = "Pinta",container_cap = 20},
            #ship{id = 4,name = "SS Minnow",container_cap = 20},
            #ship{id = 5,name = "Sir Leaks-A-Lot",container_cap = 20}],
   containers = [...], %% not shown
   ports = [#port{id = 1,name = "New York",
                  docks = ['A','B','C','D'],
                  container_cap = 200},
            #port{id = 2,name = "San Francisco",
                  docks = ['A','B','C','D'],
```

5

```
                     container_cap = 200},
              #port{id = 3,name = "Miami",
                    docks = ['A','B','C','D'],
                    container_cap = 200}],
    ship_locations = [{1,'B',1},
                      {1,'A',3},
                      {3,'C',2},
                      {2,'D',4},
                      {2,'B',5}],
    ship_inventory = #{1 => [14,15,9,2,6],
                       2 => [],   %% no more containers
                       3 => [],
                       4 => [2,8,11,7],
                       5 => [5,10,12]},
    port_inventory = #{1 => [16,17,18,19,20],
                       2 => [21,22,23,24,25],
                       3 => [26,27,28,29,30,1,3,4,13]} %% loaded here
}}
```

10. `unload_ship(Shipping_State, Ship_ID, Container_IDs)`
    This method returns a shipping state in which **the given containers** on a ship are offloaded to the port in which the ship is docked. Make sure that all the containers are located on the ship. In the case that offloading to a port would put the port over capacity, throw the exception **error** and do not offload any containers to the port. Also throw the exception **error** if the container ID is not on the ship.

```
12> shipping:unload_ship(shipping:shipco(), 1, [2,16,18]).
** exception throw: error
     in function  shipping:unload_ship/3  ...
     ...

13> shipping:unload_ship(shipping:shipco(), 1, [14,2]).
{ok,#shipping_state{
    ships = [#ship{id = 1,name = "Santa Maria",container_cap = 20},
             #ship{id = 2,name = "Nina",container_cap = 20},
             #ship{id = 3,name = "Pinta",container_cap = 20},
             #ship{id = 4,name = "SS Minnow",container_cap = 20},
             #ship{id = 5,name = "Sir Leaks-A-Lot",container_cap = 20}],
    containers = [...], %% not shown
    ports = [#port{id = 1,name = "New York",
                   docks = ['A','B','C','D'],
                   container_cap = 200},
             #port{id = 2,name = "San Francisco",
                   docks = ['A','B','C','D'],
                   container_cap = 200},
             #port{id = 3,name = "Miami",
                   docks = ['A','B','C','D'],
                   container_cap = 200}],
    ship_locations = [{1,'B',1},
```

```
                            {1,'A',3},
                            {3,'C',2},
                            {2,'D',4},
                            {2,'B',5}],
    ship_inventory = #{1 => [15,9,6], %% removed from here
                       2 => [1,3,4,13],
                       3 => [],
                       4 => [2,8,11,7],
                       5 => [5,10,12]},
    port_inventory = #{1 => [16,17,18,19,20,14,2],   %% placed here
                       2 => [21,22,23,24,25],
                       3 => [26,27,28,29,30]}}}
```

11. set_sail(Shipping_State, Ship_ID, {Port_ID, Dock})
    This method changes the given ship's port and dock location to the new port and
    dock location. Be sure to check whether or not the new port and dock is occupied.
    If it is, then throw the exception **error**. Also throw the exception **error** if the ship is
    already located at the destination port.

```
14> shipping:set_sail(shipping:shipco(), 4, {2,'B'}).
** exception throw: error
     in function  shipping:set_sail/3  ...
     ...

15> shipping:set_sail(shipping:shipco(), 4, {3,'A'}).
{ok,#shipping_state{
    ships = [#ship{id = 1,name = "Santa Maria",container_cap = 20},
             #ship{id = 2,name = "Nina",container_cap = 20},
             #ship{id = 3,name = "Pinta",container_cap = 20},
             #ship{id = 4,name = "SS Minnow",container_cap = 20},
             #ship{id = 5,name = "Sir Leaks-A-Lot",container_cap = 20}],
    containers = [...], %% not shown
    ports = [#port{id = 1,name = "New York",
                   docks = ['A','B','C','D'],
                   container_cap = 200},
             #port{id = 2,name = "San Francisco",
                   docks = ['A','B','C','D'],
                   container_cap = 200},
             #port{id = 3,name = "Miami",
                   docks = ['A','B','C','D'],
                   container_cap = 200}],
    ship_locations = [{1,'B',1},
                      {1,'A',3},
                      {3,'C',2},
                      {3,'A',4}, %% new port and dock
                      {2,'B',5}],
    ship_inventory = #{1 => [14,15,9,2,6],
                       2 => [1,3,4,13],
                       3 => [],
```

```
                              4 => [2,8,11,7],
                              5 => [5,10,12]},
    port_inventory = #{1 => [16,17,18,19,20],
                       2 => [21,22,23,24,25],
                       3 => [26,27,28,29,30]}}}
```

4   Your Task

Your task is to complete all the functions in `shipping.erl` as mentioned above. You **DO NOT** have to touch `shipping.hrl`. Some example modules to help you complete some of these functions would be the `lists` and `maps` modules.

5   Submission Instructions

Submit a file **hw3.zip** through Canvas containing all the files included in the stub but where all required operations have been implemented.