Mar 12, 14 9:29 **main.erl** Page 1/1

```
-module(main).
-export([start/0]).

start() ->
        % start all processes
        display:start(),
        tempConv:start(),
        clock:start(),
        sensor:start(celsiusSensor, fahrenheit2celsius),
        sensor:start(fahrenheitSensor, celsius2fahrenheit),

        % add the functions and sensors
        tempConv:add_new_fun(fahrenheit2celsius, fun(X) -> (X-32)*5/9 end),
        tempConv:add_new_fun(celsius2fahrenheit, fun(X) -> (X*9/5)+32 end),
        clock:add_new_sensor(celsiusSensor),
        clock:add_new_sensor(fahrenheitSensor).
```

Mar 12, 14 1:41 **clock.erl** Page 1/1

```
-module(clock).
-export([start/0, restarter/0, loop/1, add_new_sensor/1]).
-define (CLOCK_PERIOD_MS, 1000).

% clock spawner
start() ->
        spawn(?MODULE, restarter, []).

% the restarter makes sure that except when we explicitly
% killed the process, it will restart itself
restarter() ->
        process_flag(trap_exit, true),
        Pid = spawn_link(?MODULE, loop, [[]]),
        register(clock, Pid),
        receive
                {'EXIT', _Pid, normal} ->
                        io:format("[clock] terminated normally~n");
                {'EXIT', _Pid, shutdown} ->
                        io:format("[clock] manually terminated~n");
                {'EXIT', _Pid, _} ->
                        io:format("[clock] something went wrong,
                                so I'll just restart~n"),
                        restarter()
        end.

% main loop:
% responsable for calling all sensors to send their
% readings every CLOCK_PERIOD_MS ms,
% also receives new sensors and add them to the
% sensor list 'L'
loop(L) ->
        receive
                {new_sensor, A} ->
                        loop([A|L])
        after ?CLOCK_PERIOD_MS ->
                io:format("[clock] tick~n"),
                NL = tick_all(L),
                loop(NL)
        end.

% send a 'tick' message to all processes in the inpu list,
% if an atom in the list represents a dead process the said
% atom will be removed from the list
tick_all([H|L]) ->
        Pid = whereis(H),
        if
                Pid =:= undefined -> tick_all(L);
                true -> Pid ! tick, [H|tick_all(L)]
        end;
tick_all([]) -> [].

% encapsulated message to add a new sensor 'S'
add_new_sensor(S) ->
        clock ! {new_sensor, S}.
```

Mar 12, 14 1:29 **display.erl** Page 1/1

```
-module(display).
-export([start/0, restarter/0, loop/0]).

% display spawner
start() -> spawn(?MODULE, restarter, []).

% the restarter makes sure that except when we explicitly
% killed the process, it will restart itself
restarter() ->
        process_flag(trap_exit, true),
        Pid = spawn_link(?MODULE, loop, []),
        register(display, Pid),
        receive
                {'EXIT', _Pid, normal} ->
                        io:format("[display] terminated normally~n");
                {'EXIT', _Pid, shutdown} ->
                        io:format("[display] manually terminated~n");
                {'EXIT', _Pid, _} ->
                        io:format("[display] something went wrong,
                                so I'll just restart~n"),
                        restarter()
        end.

% main loop:
% responsable for receiving the temperature messages and
% displaying them.
loop() ->
        receive
                {S, T} ->
                        io:format("[display] temp from sensor
                                ~s: ~w~n", [atom_to_list(S), T])
        end,
        loop().
```

Mar 12, 14 1:44 **sensor.erl** Page 1/1

```
-module(sensor).
-export([start/2, restarter/2, loop/2]).

% sensor spawner
start(S, F) ->
        spawn(?MODULE, restarter, [S, F]).

% the restarter makes sure that except when we explicitly
% killed the process, it will restart itself
restarter(S, F) ->
        process_flag(trap_exit, true),
        Pid = spawn_link(?MODULE, loop, [S, F]),
        register(S, Pid),
        receive
                {'EXIT', _Pid, normal} ->
                        io:format("[sensor ~s] terminated normally~n",
                                [atom_to_list(S)]);
                {'EXIT', _Pid, shutdown} ->
                        io:format("[sensor ~s] manually terminated~n",
                                [atom_to_list(S)]);
                {'EXIT', _Pid, _} ->
                        io:format("[sensor ~s] something went wrong,
                                so I'll just restart~n", [atom_to_list(S)]),
                        restarter(S, F)
        end.

% main loop:
% responsable for requesting a temperature conversion every
% 'tick' message and sending the output to 'display'
loop(S, F) ->
        receive
                tick ->
                        io:format("[sensor ~s] requesting temp
                                conversion~n", [atom_to_list(S)]),
                        % get a random reading between 1 and 100
                        T = random:uniform(100),
                        tempConv:convert_temp(self(), F, T);
                {converted, T} ->
                        io:format("[sensor ~s] temp converted.
                                sending to be displayed~n", [atom_to_list(S)]),
                        display ! {S, T}
        end,
        loop(S, F).
```

```
-module(tempConv).
-export([start/0, restarter/0, loop/1,
        convert_temp/3, add_new_fun/2]).

% converter spawner
start() -> spawn(?MODULE, restarter, []).

% the restarter makes sure that except when we explicitly
% killed the process, it will restart itself
restarter() ->
        process_flag(trap_exit, true),
        Pid = spawn_link(?MODULE, loop, [[]]),
        register(tempConv, Pid),
        receive
                {'EXIT', _Pid, normal} ->
                        io:format("[converter] terminated normally~n");
                {'EXIT', _Pid, shutdown} ->
                        io:format("[converter] manually terminated~n");
                {'EXIT', _Pid, _} ->
                        io:format("[converter] something went wrong,
                                so I'll just restart~n"),
                        restarter()
        end.

% main loop:
% responsable for receiving convert requests and responding
% with the converted temperature given the requested function,
% (that is inside its function list 'Fs'),
% can also receive new functions and add them to 'Fs'
loop(Fs) ->
        receive
                {PidSender, convert, F, T} ->
                        NewTFun = get_fun(Fs, F),
                        NewT = NewTFun(T),
                        io:format("[converter] converting:
                                ~s~n", [atom_to_list(F)]),
                        PidSender ! {converted, NewT},
                        loop(Fs);
                {loadNewConvFun, F} ->
                        loop([F|Fs])
        end.

% return the function mached by its atom name 'F'
get_fun([{_F, Fun}|_], _F) -> Fun;
get_fun([{_, _}|Fs], F) -> get_fun(Fs, F);
get_fun([], _) -> [].

% encapsulated message to convert the temperature 'T'
% using function 'F' and return the result to process 'Pid'
convert_temp(Pid, F, T) ->
        tempConv ! {Pid, convert, F, T}.

% encapsulated message to add a new function 'F' with
% the atom name 'N'
add_new_fun(N, F) ->
        tempConv ! {loadNewConvFun, {N, F}}.
```