CS 683 Emerging Technologies
Fall Semester, 2008
Doc 2 Erlang Sequential Programming
Sept 4 2008

References

Programming Erlang: Software for a Concurrent World, Armstrong, Chapter 3.

Erlang Documentation

Reading

Programming Erlang: Software for a Concurrent World, Armstrong,
Chapter 4 & 8 for Tuesday Sept. 4
You will find Chapter 6 very useful

Modules

```
-module(stuff).
-export([factorial/1, area/1]).

factorial(1) -> 1;
factorial(N) ->
      N * factorial(N - 1).

area({rectangle, Width, Height}) -> Width * Height;
area({circle, Radius}) -> 3.14159 * Radius * Radius;
area({square, Side}) -> area({rectangle,Side, Side}).
```

```
1> factorial(3).
** exception error: undefined function shell_default:factorial/1
2> stuff:factorial(3).
6
3> stuff:area({square,5}).
25
```


A module must be in a file with the same name as the module and have the extension .erl. So in the example it must live in the file stuff.erl. The file must be in the erlang search path, which includes the current directory. Read chapter 6 of the text to see how to set your erlang search path.

Changing Order

```
-module(stuff).
-export([factorial/1]).

factorial(N) ->
      N * factorial(N - 1);
factorial(1) -> 1.
```

```
4> c(stuff).
./stuff.erl:7: Warning: this clause cannot match because
a previous clause at line 5 always matches
{ok,stuff}

5> stuff:factorial(2).
beam.smp(1151,0xb01aa000) malloc: ***
mmap(size=1426063360) failed (error code=12)
 *** error: can't allocate region
                          *** set a breakpoint in
malloc_error_break to debug


Crash dump was written to: erl_crash.dump
eheap_alloc: Cannot allocate 1425410620 bytes of
memory (of type "heap").
Abort
```

Changing Order

```
-module(stuff).
-export([area/1]).

area({square, Side}) -> area({rectangle,Side, Side});
area({rectangle, Width, Height}) -> Width * Height;
area({circle, Radius}) -> 3.14159 * Radius * Radius.
```

```
1> stuff:area({square,5}).
25
```

Arity creates different Functions

```
-module(list).
-export([sum/1]).

sum(List) -> sum(List, 0).

sum([], N) -> N;
sum([H|T], N) -> sum(T, N + H).
```


There are two different sum functions. One with one argument and one with two arguments.

funs - Anonymous functions

```
4> Increment = fun(N) -> N + 1 end.
#Fun<erl_eval.6.13229925>

5> Increment(3).
4
6> X = Increment.
#Fun<erl_eval.6.13229925>
7> X(10).
11
```

```
8> IsOdd = fun(N) -> (N rem 2) =:= 1 end.
#Fun<erl_eval.6.13229925>

9> IsOdd(5).
true
```


funs are like blocks in Smalltalk or Ruby or lambda's in Lisp. Java will have closures in the near future.

Functions using Funs - map

8> **lists:map(IsOdd, List).**
[true,false,true,false]

9> lists:map(Increment, List).
[2,3,4,5]

```
IsOdd = fun(N) -> (N rem 2) =:= 1 end.
Increment = fun(N) -> N + 1 end.
```

All functions in the lists module are like this

map is like the collect: iterator in Smalltalk and Ruby. We would write list collect: isOdd where isOdd = [:x | x odd].

Funs of Funs

```
6> IncrementBy = fun(Increment) -> (fun(N) -> Increment + N end) end.
#Fun<erl_eval.6.13229925>

7> Increase = IncrementBy(3).
#Fun<erl_eval.6.13229925>

8> Increase(10).
13
```

Erlang Docs

http://www.erlang.org/doc/

API Docs

http://www.erlang.org/doc/man_index.html

List Comprehensions

```
1> List = [1, 2, 3, 4, 5, 6].
[1,2,3,4,5,6]
2> [X + 1 || X <- List].
[2,3,4,5,6,7]
3> [X || X <- List, X < 5].
[1,2,3,4]
4> IsOdd = fun(N) -> (N rem 2) =:= 1 end.
#Fun<erl_eval.6.13229925>
5> [X || X <- List, X < 5, IsOdd(X)].
[1,3]
6> [(Y+2)*3 || Y <- List, Y < 5, IsOdd(Y)].
[9,15]
```

List Comprehensions & Matching

```
14> [ Z || {a, Z} <- [{a, 1}, {b, 2}, "cat", {a, dog}]].
[1,dog]
```

Example

```
-module (split).
-export ([middle/2]).

middle (List,N) ->
      Small = [X || X <- List, X < N],
      Large = [X || X <- List, X > N],
      {small, Small, large, Large}.
```

```
21> split:middle([1,2,3,4,5], 4).
{small,[1,2,3],large,[5]}
```


The list generator (x <- List in the example) must be occur before any filters (X< N) that use the generator.

Multiple Generators

```
1> List = [1,2,3].
[1,2,3]
2> [{X, Y} || X <- List, Y <- List].
[{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3}]
```

Guards

-module (guard).
-export ([max/2]).

max (X,Y) **when X > Y -> X**;
max (X, Y) -> Y.

Or
F(X,Y....) when G1; G2; ... ; GN

And
F(X,Y....) when G1, G2, ... , GN

Guards & Side Effects

Guards can not have side effects

Erlang Side effects

- Message sends
- Database changes
- get, put (per process)

```
1> put(dog,5).
undefined
2> get(dog).
5
3> put(dog,"rat").
5
4> get(dog).
"rat"
```


See page 56 of the text for a list of items that are allowed in guards.
Beware of using get and put

BIF - Built in Functions

BIF docs

http://www.erlang.org/doc/man/erlang.html

case

No side effects allowed in guards
A pattern must match

```
case Expression of
    Pattern1 [when Guard1] -> Expression_sequence1;
    Pattern2 [when Guard2] -> Expression_sequence2;
    ...
    PatternN [when GuardN] -> Expression_sequenceN
end
```

```
-module (guard).
-export ([max/2]).

max (X,Y) ->
      case X > Y of
          true -> X;
          false -> Y
end.
```

if

```
-module (guard).
-export ([max/2]).

max (X,Y) ->
      if
            X > Y -> X;
            X < Y -> Y;
            X =:= Y -> Y
end.
```

Accumulators

```
-module (split).
-export ([middle/2]).

middle (List,N) -> middle(List, N, [], []).

middle ([H|T],N,Small, Large) ->
      case H < N of
            true -> middle(T,N,[H | Small], Large);
            false -> middle(T,N, Small, [H | Large])
      end;
middle ([], _, Small,Large) ->
      {small, lists:reverse(Small), large, lists:reverse(Large)}.
```


Note the use of the "I don't care" variable "_" in the last middle function.