Erlang

•••

By Blair, Mark, and Sindhu

Overview

- History
- Purpose & Design Goals
- Language Features
- Code Samples
- Real-World Applications

History of Erlang

- Developed at the Ericsson Computer Science Laboratory by Joe Armstrong
  - According to their website,
    "Ericsson is one of the leading providers of Information and Communication Technology (ICT) to service providers, with about 40% of the world's mobile traffic carried through our networks"
- Erlang
  - First released in 1986
  - First open source released in 1998

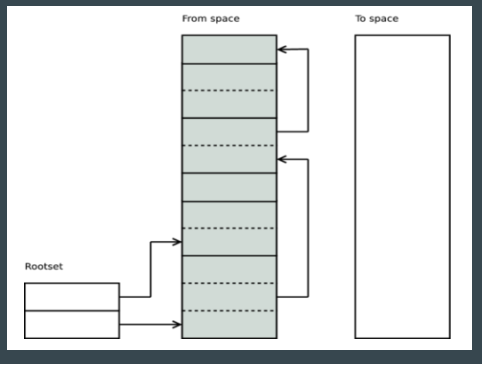
ERICSSON

Purpose

- Before Erlang:
  - PLEX (language similar to basic)
    - Low level language
    - Time consuming
- With Erlang:
    - Concurrency(Process)
    - Scalability(Switch between processes quickly)
    - Reliability
    - Easier to understand and write

Language Features

- Simple functional language
- Uses both stack and heap for memory management
    - Process Control Block: information about the process, current status, initial and current call, pointers to incoming messages
    - Stack: incoming and outgoing parameters, return addresses, local variables (tentative)
    - Heap: physical messages of process mailbox, compound data types such as lists and tuples
    - Provides Garbage Collection
        - Young Heap
            - Copying semi-space garbage collector
            - Terms are copied from one distinct area, called the from space, to a new clean area, called the to space
                - Partition & Copy
        - Old Heap
            - Reference Counts



Language Features

- Pattern Matching
  - Variables bound to values through pattern matching
  - Left-hand side pattern matched against a right-hand side term
    - Successful?
      - Unbound variables become bound
    - Unsuccessful?
      - Run time error
- Dynamically typed

Pattern Matching Example:

```
1> X.
** 1: variable 'X' is unbound **
2> X = 2.
2
3> X + 1.
3
4> {X, Y} = {1, 2}.
** exception error: no match of right hand
side value {1,2}
5> {X, Y} = {2, 3}.
{2,3}
6> Y.
3
```

Control Statements- IF

```
-module(exampleIf).
-export([gradeCalc/1]).
gradeCalc(X) ->

if
        X >= 90  ->
        io:fwrite("Your grade is A");
        X >= 80  ->
        io:fwrite("Your grade is B");
        X >= 70  ->
        io:fwrite("Your grade is C");
        X >= 60  ->
        io:fwrite("Your grade is D");
        true       ->
        io:fwrite("Your grade is F")
end.
```

```
1> c(exampleIf).
{ok,exampleIf}
2> exampleIf:gradeCalc(90).
Your grade is A
3> exampleIf:gradeCalc(80).
Your grade is B
4> exampleIf:gradeCalc(70).
Your grade is C
5> exampleIf:gradeCalc(60).
Your grade is D
6> exampleIf:gradeCalc(50).
Your grade is F
```

Fahrenheit and Celsius

```
-module(helloworld).
-export([fahrToCel/1, celToFahr/1]).
fahrToCel(X) ->
    C = (X-32) * 5/9.
printFahrToCel(X) ->
    io:fwrite("The temperature in Celsius is ~w~n",[fahrToCel(X)]).
celToFahr(N) ->
    F = (N * 9/5) + 32.
printCelToFahr(X) ->
    io:fwrite("The temperature in Fahrenheit is ~w~n",[celToFahr(X)]).
```

```
>> fahrToCel(212)
100.0

>> celToFahr(0)
32.0

>> printFahrToCel(212)
The temperature in Celsius is 100.0

>> printCelToFahr(0)
The temperature in Fahrenheit is 32.0
```

List

ERLANG

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

sum([]) ->
0;
sum([H|T]) ->
H + sum(T).
```

SCHEME

```
(define (my-sum lst)
(cond
([empty? lst] 0)
(else (+ (car lst)
(my-sum (cdr lst)))))
```

Tail-Recursion Optimization

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

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

sum([], Tot) ->
   Tot;

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

```
sum([1,2,3])
sum([1,2,3], 0)
sum([2,3], 1)
sum([3], 3)
sum([], 6)
6
```

Concurrency Example

```
-module(tut14).

-export([start/0, say_something/2]).

say_something(What, 0) ->
        done;
say_something(What, Times) ->
        io:format("~p~n",
[What]),
        say_something(What,
Times - 1).

start() ->
        spawn(tut14,
say_something, [hello, 3]),
        spawn(tut14,
say_something, [goodbye, 3]).
```

```
1> c(tut14).
{ok,tut14}
2>
tut14:say_something(hello,
3).
hello
hello
hello
done

3> tut14:start().
hello
goodbye
<0.63.0>
hello
goodbye
hello
goodbye
```

Real-World Applications

- Amazon
  - for implementing SimpleDB, and for providing database services for Amazon Elastic Compute Cloud
- Facebook
  - for powering the backend of their chat service
- Whatsapp
  - for running messaging servers
  - Reliability
- Ericsson
  - used in its support nodes

Why Erlang?

- Concurrency
  - Several threads of execution
- Scalability
- Upgrade while editing
- Fault-tolerant
- Little Overhead

Sources

http://erlang.org/doc/reference_manual/data_types.html

https://www.tutorialspoint.com/erlang/erlang_loops.htm

https://www.tutorialspoint.com/erlang/erlang_if_statement.htm

http://erlang.org

https://www.safaribooksonline.com/library/view/erlang-programming/9780596803940/ch01.html

https://hamidreza-s.github.io/erlang%20garbage%20collection%20memory%20layout%20soft%20realtime/2015/08/24/erlang-garbage-collection-details-and-why-it-matters.html

http://erlang.org/faq/introduction.html
https://www.erlang-factory.com/upload/presentations/416/MikeWilliams.pdf
http://webcem01.cem.itesm.mx:8005/erlang/cd/downloads/hopl_erlang.pdf
http://erlang.org/pipermail/erlang-questions/2015-March/083825.html
http://erlang.org/doc/getting_started/conc_prog.html