

Erlang

Joel Smith Luke Martin

Background Information

- Erlang was released in 1986 (36 years old)
- General-purpose, functional programming language
- It was influenced by Lisp, Prolog and Smalltalk
- Erlang was made to improve the development of telephone applications

```
fn ->
    receive do
        {:hello, what} -> what
        end
        [> IO.puts
end
[> spawn
[> send {:hello, [119,111,114,108,100]}
```

Program Overview

Euler Phi Totient Function

- **RSR** (Reduced Set of Residues)
- ${n \in Z^+ \mid 1 \le x \le n \text{ and } gcd(x, n) = 1}$
- Euler Φ Function

 $\Phi(n) = | \{n \in Z^+ | 1 \le x \le n \text{ and } gcd(x, n) = 1\} |$

 $\Phi(24) = 8$ RSR = {1, 5, 7, 11, 13, 17, 19, 23}

Syntax Rules

• A piece of any type of data is defined as a **term**

• An **Atom** is a literal, starts with a lowercase character



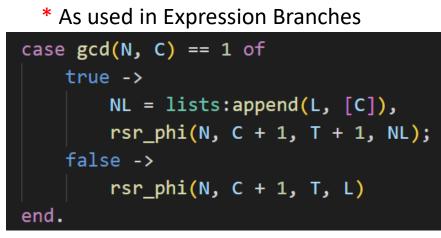
lit(N, literal) * Second argument is considered an Atom

Syntax Rules Continued

Ending Lines in Erlang

- Several ways to end a line
- •, Comma Separates Expressions (Figure 2)
- . Period Used at the end of Functions (Figure 1)
- ; Semicolon is a clause separator
 - * As used in Functional Clauses

gcd(0, B) -> B; gcd(A, B) -> gcd(B rem A, A). Fig. 1



Syntax Rules Continued

Other syntax rules

- The use of the ___ (underline), the compiler just ignores that value
- The -> calls the body of a function
- Comments start with %

% Euclidean Algorithm for GCD. gcd(0, B) -> B; gcd(A, B) -> gcd(B rem A, A).

Binding and Scoping Rules

- Variables **Bindings** are available until the end of the scope.
- Variables introduced in a clause are only available within the body.
- Every function is by default local, unless exported.
- Global scoping is not available

```
-export([phi/1, rsr/1]). * Exports two functions to the client

rsr_phi(N, C, T, L) ->

% Weird thing to prevent side effects, only can use local functions within a case.
% Local functions within guards will error out.
case gcd(N, C) == 1 of

true ->

NL = lists:append(L, [C]),
rsr_phi(N, C + 1, T + 1, NL);
false ->

rsr_phi(N, C + 1, T, L)
end.
```

Exports & Modules

- Modules are functions grouped in a file
- Without **Exports**, the Client can't call any functions
 - Formatted as: -exported([function/# of arguments], ...)

```
% Identify a Module
-module(euler).
% Export the function with a givin number of arguments, n.
-export([phi/1, rsr/1]).
```

- Calling a **Function** is done as so:
 - Module:Function_Name(args).

```
3> euler:phi(30). * Calling a Modules Function
8
```

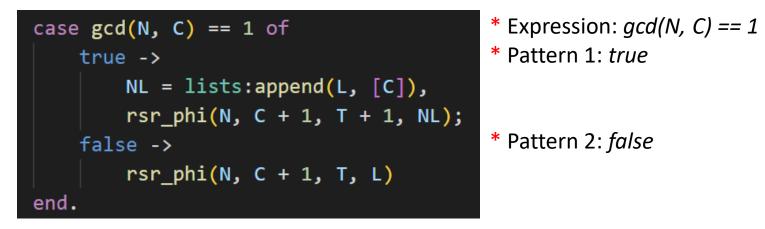
* Compiling a Module

2> c(euler). {ok,euler}

* Returns a Tuple containing
 {status, module_name}

Control Flow

- Erlang provides If and Case Statements
 - May be included within functions



• Return value of the **Body** is the return of the **case** expression

Control Flow

Associativity

• Erlang supports the use of Left and Right associative

Left Associative	Right Associative
div	++
+ -	= !

The use of **parentheses** help with precedence

Recursion

- In Erlang, iteration isn't achievable, so recursion is used.
- As a **result**, there are no **for** or **while** loops.
- Tail recursion is used for recursive calls.

```
rsr_phi(N, C, T, L) ->
% Weird thing to prevent side effects, only can use local functions within a case.
% Local functions within guards will error out.
case gcd(N, C) == 1 of
true ->
NL = lists:append(L, [C]),
rsr_phi(N, C + 1, T + 1, NL);
false ->
rsr_phi(N, C + 1, T, L)
end.
```

* Notice how rsr_phi is called at the tail of the function

Data Types

- There are many data types. Some of the important ones are
- Terms, Number, Atom, Bit Strings and Binaries, Fun, Reference, List and many more.
- The index of the first element is one.
- There are also integers, floats and chars.

Data Types

- For the RSR, PHI program, we used Lists and Tuples as Non-Primitive Data types.
 - A list contains terms
 - A **tuple** has a fixed number of terms

```
rsr_phi(N, N, T, L) -> {L, T}; * Creating a Tuple
 NL = lists:append(L, [C]),
 rsr_phi(N, C + 1, T + 1, NL);
```

* Adding Elements to a List

% Returns the 1st Element of the Tuple which is the RSR. rsr(N) -> element(1, rsr_phi(N)). % Returns The 2nd Element of the Tuple which is the Phi Value. phi(N) -> element(2, rsr_phi(N)).

* Accessing elements requires use of the **element** function



- https://www.tutorialspoint.com/erlang/erlang_tuples.htm
- <u>http://geekhmer.github.io/blog/2015/01/20/erlang-control-flow-statement/</u>
- <u>https://elixir-lang.readthedocs.io/en/latest/technical/scoping.html</u>
- <u>https://www.erlang.org/doc/index.html</u>