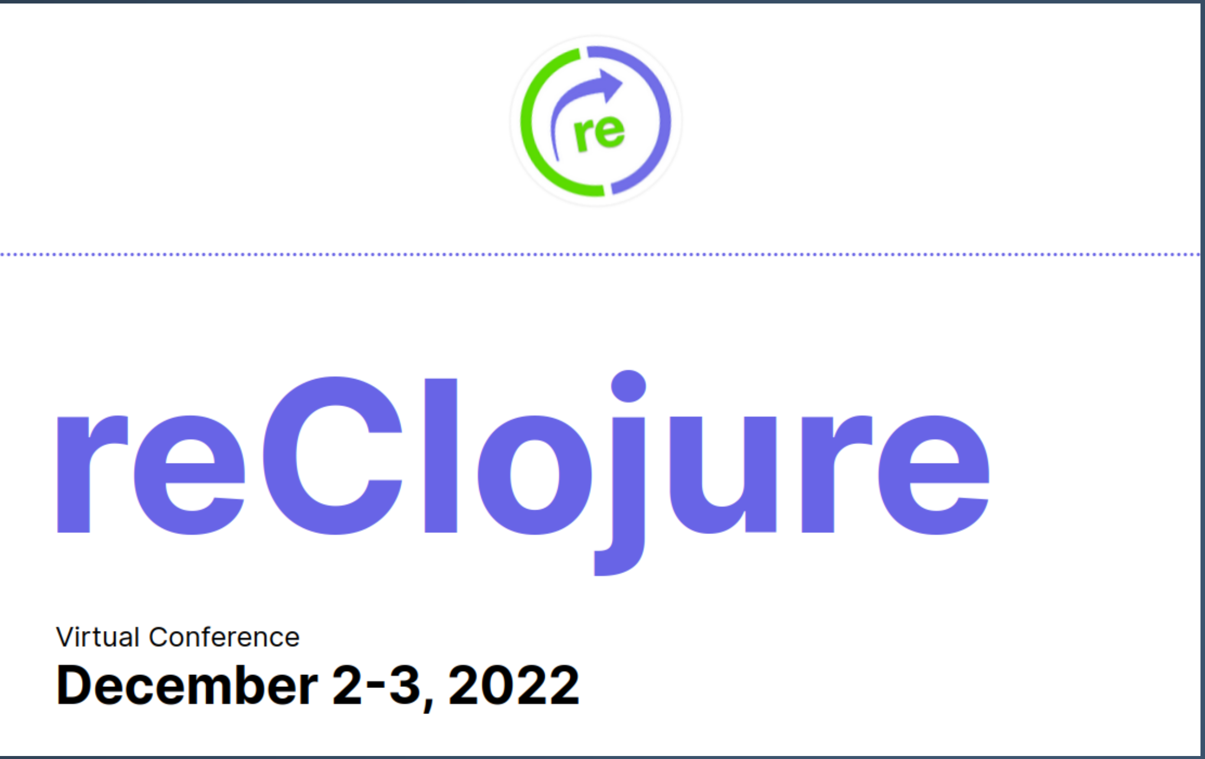
CLORTH

A FORTH IN CLOJURE

LONDON CLOJURIANS MEETUP - OCTOBER 12TH 2022


reClojure

Virtual Conference

December 2-3, 2022

- reclojure.org 2022 is just around the corner!
- The CfP is still open! Hurry up!
- https://sessionize.com/reclojure-2022/

MANY FLAVOURS OF “WHY”?

- Why implementing a language?
- Why implementing a language in Clojure?
- Why implementing Forth in Clojure?

WHY IMPLEMENTING A LANGUAGE?

- Effective way to learn
- Getting familiar with different way of thinking programs
- Trick famously popularized “The Pragmatic Programmer” book:

> “ *Learn at least one new language every year. Different languages solve the same problems in different ways. By learning several different approaches, you can help broaden your thinking and avoid getting struck in a rut.*

BUT ALSO

CLOJURE HAS A CUSTOMISABLE REPL

GREAT FOR BOTH INTERNAL (LISPY) OR EXTERNAL DSLS

CLOJURE HAS GREAT MULTIMETHOD DISPATCH FACILITIES

WHY FORTH?

- What if Clojure didn’t have parenthesis?
- Simple language with almost no grammar
- Yet expressive and powerful
- Heard about it: Michelson, Nasa, etc.
- Great for writing DSLs

FORTH PROGRAMS ARE ACTUALLY DSLS!

Here’s a small program.

What do you think this is going to print?

```
line dot line dot dot cr
```

Let me show you…

FUNCTIONAL VS CONCATENATIVE

- Clojure is **applicative:** it executes by applying `f` to `args`
- A program is written by **composing** functions
- Forth is **concatenative:** it executes by invoking words
- A program is written by **concatenating** words
- The stack is implicit, no `args` required

LET ME SHOW YOU...

RECIPE: IMPLEMENTING A LANGUAGE

You might need some or all of the following:

- A **Reader** (string -> data structure)
- A **Parser** (data structure -> other representation)
- An **Interpreter** (data structure -> evaluation)
- A **Compiler** (data structure -> executable)

THE READER

- If the language is Lispy use the Clojure reader
    - Plenty of examples: `core.test`, `core.async`
- If not Lispy you can:
    - Instaparse it
    - Hybrid approach (Clorth)
    - Roll your own

ARE YOU READY?

```
NON_EMPTY_WORD_LIST = WORD { SPACE WORD }
```

THE CLORTH READER

```
(def reader-specials
  {":" "\\:" ";" "\\;" "~" "\\~"}) ;; A few more

(defn handle-specials [s]
  (reduce (fn [s [s1 s2]]
            (string/replace s s1 s2))
    s reader-specials))

(defn read [s]
  (read-string (str "[" (handle-specials s) "]")))

;; Approach: move the tokens into a vector, parse
the tokens
;; Challenge: some Forth tokens are not valid
Clojure symbols.
```

THE PARSER

- The parser transforms the initial reading
- Normally it enriches tokens with semantic information
- The information is used for evaluation or compilation
- This is not an universal rule!
- Clorth doesn’t really have a parser.

THE INTERPETER

- Interpretation evaluates the language at the interpreter level itself
- Compilation is a level down (e.g. JVM, machine language etc.)
- The lower the level the faster the speed
- Clorth has currently a simple token interpreter

TANGENT: INTERPRETER VS COMPILATION LEVELS

- `1+` invokes `inc` that increments a number on the stack (Program level 0)
- `1+` becomes a function that increments a number on the stack (Program level -1)
- `1+` becomes bytecode `iadd` that increments an item from the JVM stack (Program level -2)
- `1+` becomes `inc` i386 instruction that increments a CPU register (Program level -4)

THE CLORTH INTERPRETER

```
(defmulti word
  (fn [_ args] (cond-> args (coll? args) first)))

(defmethod word 'drop
  [env args]
  (word (pop1 env) (rest args)))
```

THE CLORTH REPL

- Clojure comes with `clojure.main/repl`
- This is a small API to create your own REPL
- As expected it gives you a `:read` and an `:eval` options
- The Print/Loop part is taken care of
- Let me show you…

SOME HIDDED FEATURES

- The stack is immutable and passed as a parameter while interpreting
- There is interop to call into Clojure functions
- To go native, use the “_” prefix in front of a word
- Error messages are as simple as possible, no Java Exceptions
- Multimethods are split by namespaces: stdblib, math and soon others.

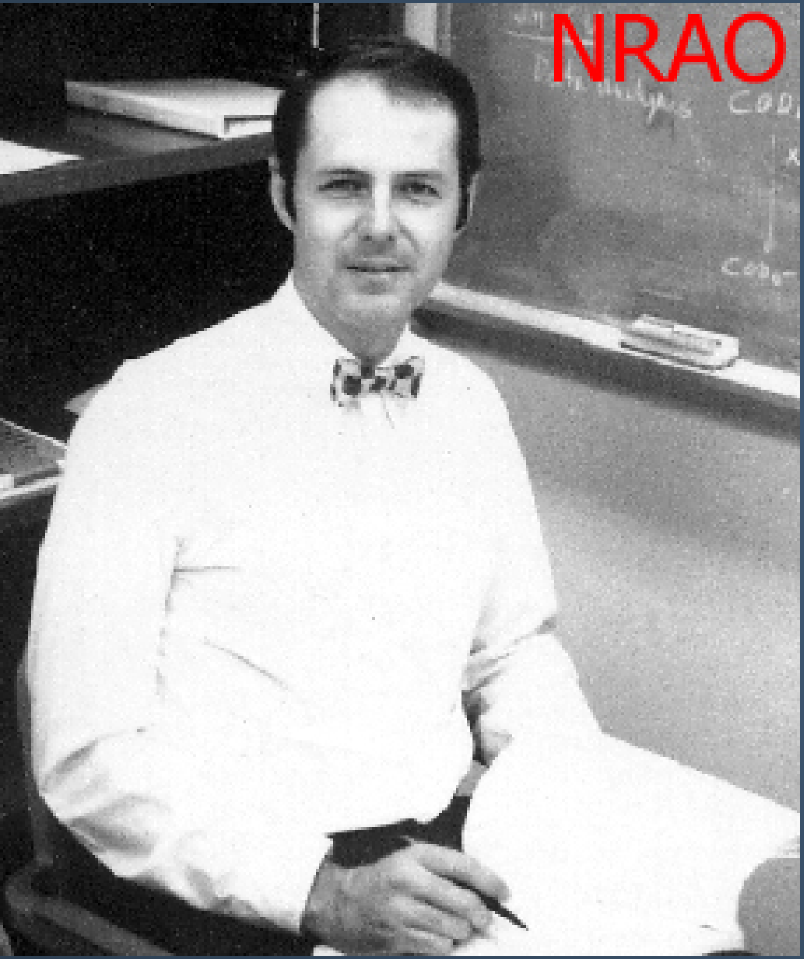
WHAT'S NEXT?

- Variables, arrays
- Meta-programming
- Assembler-like programming
- Running "Worms?" :)

RESOURCES

- Clorth: an ANSI compliant implementation
- These Slides
- GNU Forth: an ANSI compliant implementation
- On the concatenative paradigm: quick essay about concatenative program properties.
- CONCATENATIVE PROGRAMMING An Overlooked Paradigm in Functional Programming: a well written paper about the relationship with functional programming.

~ FIN ~



Charles H. Moore, Inventor of Forth