Red/C3

Smart contracts programming made decent.
C3 languages stack

- C3/System DSL
  - low-level (roughly on par with Solidity)
  - turing-complete

- C3 DSL
  - very high-level
  - limited semantics, turing-incomplete
  - high-level components written in C3/System
C3/System goals

- Human-friendliness
  - rich set of datatypes
  - type inference
  - closer to human syntax

- Safety
  - type-safe
  - security features as first-class citizens
  - array bounds checking at runtime

- Short compiler code (< 5000 LOC)
- Dead-simple toolchain (<1MB, single file, no setup)
- EVM simulator (eventually assisting the compilation)
- Generative tests framework
C3/System vs Solidity

- Safe math built-in (all over/underflows throw exceptions)
- No recursion/reentrance allowed (enforced at compile and runtime)
- No inline assembly
- No inheritance
- DbC pre/post conditions, instead of modifiers
- Custom ABI with function names as signatures
- One contract per source file
- Access control lists for external accesses
- Function names limited to 32 bytes UTF-8
- Self-destruction as an opt-in property in C3 header
- Unicode allowed for identifiers (though whitespaces are not)
C3/System: types 1/2

- integer! (64-bit signed, sub-types: #8, #16, #24, #32, #40)
- logic! (1-bit)
- byte! (8-bit)
- time! (32-bit)
- date! (32-bit)
- money! (128-bit unsigned, fixed-point decimal, synonym: ether!)
- wei! (128-bit unsigned integer)
- address! (160-bit unsigned)
- hash! (256-bit unsigned)
C3/System types 2/2

- **string!** (UTF-8 encoded), subtypes:
  - **tag!** (<html>)
  - **issue!** (#1-201-1234-789)
  - **email!** (nr@red-lang.org)
  - **url!** (http://red-lang.org)
- **record!** (basically a struct)
- **array!** (fixed or dynamically sized)
- **map!** (associative array)

- **char!** (32-bit unsigned, full Unicode character)
- **unit!** (number with unit)
- **bitset!** (array of bits)
# C3/System keywords

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
<th>Option</th>
<th>Conditional</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>//</td>
</tr>
<tr>
<td>=</td>
<td>&lt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td>or</td>
<td>and</td>
</tr>
<tr>
<td>xor</td>
<td>not</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>any</td>
<td>all</td>
<td>if</td>
<td>either</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>loop</td>
<td>make</td>
<td>append</td>
<td>abort</td>
</tr>
<tr>
<td>hash</td>
<td>return</td>
<td>require</td>
<td>ensure</td>
<td>assert</td>
</tr>
<tr>
<td>env</td>
<td>self</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>function</td>
<td>states</td>
<td>storage</td>
<td>access</td>
<td>logs</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>none</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C3/System special accessors

env/block/coinbase
env/block/difficulty
env/block/gas-limit
env/block/number
env/block/timestamp
env/block/gas-left

env/caller/data
env/caller/gas
env/caller/address
env/caller/amount

env/tx/gas-price
env/tx/origin
C3/System: source layout

C3/System [
  ... header, (word: value) pairs...
]

;-- (record templates, global memory variables)
<global definitions>

logs      [...logs prototypes...]
storage   [...state variables...]
access    [...functions access lists...]

;-- constructor (optional)
on-create: function [...spec...][...body...]

;-- functions / transactions
foo: function [...spec...][...body...]

;-- optional state machine
states    [...state cases...]
C3/System [
  Date: 10-Oct-2018
  Version: 1.0.0
  Options: [self-destruct: yes]
]

logs [Hello: string!]

hi: function [name: string!][
  log [Hello name]
]
person!: template [
  name:  string!
  surname:  string!
  age:  integer!
  children: some person!
  parents: 2 person!
]

storage [
  whitelist:  map address! to money!
  token-address: address!
  list: some address!
  children:  map string! to person!
  john:  person!
]

foo: function [][
  a: make person!
  append john/children a
  a/age: 25
  john/parents/1: none
]
C3/System functions

<name>: function [  
  <attributes>  
  <arg1>: <type1>  
  ...  
  <argN>: <typeN>  
  return: <typeR>  
][...body...]

<attributes> : (optional) list of attributes (block! of word!)
<arg1-N>    : (optional) argument names (word!)
<type1-N>   : (optional) argument types (type expression)
<typeR>     : (optional) returned value type (type expression)
C3/System access lists

access [
    some [some <account> some <function>]
]

<account>: address! value or variable | 'anyone
<function>: function name

-----------------------------------------------

manager: 0x123...
access [
    manager [burn mint]
    anyone [transfer getBalance totalSupply decimals name]
]
C3/System toolchain

- One single binary: c3
  - $ c3 hello.c3
  - $ c3 deploy localhost:8501 hello.c3
  - $ c3 test hello.c3
  - $ c3 run hello.c3

- Cross-platform
  - WinXP/10, macOS, Linux, Android, web
  - Intel, ARM

- C3/System compiles directly to EVM bytecode
- C3 compiles to C3/System
Future features

◊ Scheduled transactions that can be expressed from C3 and C3/System.

◊ Importing and storing arbitrary files on IPFS/Swarm from C3 and C3/System, while storing their hash on Ethereum.

◊ Buffering storage write accesses in memory, optimizing the writes across a whole transaction instead of just a function.
C3: very high level DSL

- Challenging design, as domain space is blurry
- IFTTT model considered, but unlikely to map adequately
- High-level objects, exposing properties and event hooks