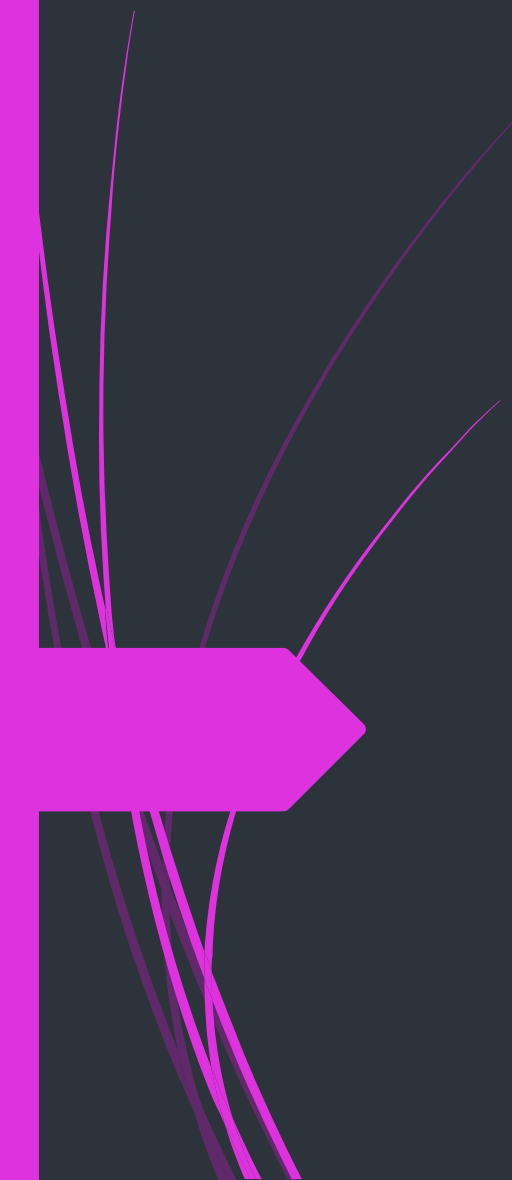

A TALE OF TWO FORTHS

Joseph M. O'Connor

SVFIG Forth Day Presentation

November 16, 2019

Forth background

- First encounter with Forth was with Tom Zimmer's F-PC.
- I found it useful in my assembly language class to first write code in F-PC and then translate it.
- Later on, I found Norman Smith's UNTIL (written in C++) and tinkered around with it.
- Smith's UNTIL was a form of Forth that was intended to serve as a DSL (Domain Specific Language) that sat on top of an existing application rather than as a standalone Forth.
- This inspired me to write my own version(s) of Forth, which I call Creole Forth.

Some Creole Forth history

- As with UNTIL, the focus has been on a domain-specific language that could sit on top of a host application, not on a standalone system.
- As of 2019, there are four different versions which have been developed in four different host environments or languages:

  1. Delphi (1999-2003). It also works for the Lazarus environment.
  2. Excel. (2016).
  3. JavaScript (2018).
  4. Python (2019).

- This presentation will be focusing on the JavaScript and Python versions.

Primary moving parts:

- Stacks. Arrays in JavaScript, lists in Python.
- Dictionary. An associative array with key-value properties. In JavaScript this is naturally built-in – all properties are attached as part of the object and can be accessed with square brackets [].In Python a dictionary object is used.
- Reverse dictionary – this is an indexable array or list which contains the same values as the dictionary but indexed by integer
- GlobalSimpleProps – it's passed as a parameter to all Creole Forth primitives, which are just methods attached to empty objects. The objects are labeled as CorePrims, Interpreter, Compiler, LogicOps, and AppSpec to organize similar primitives together.
- CreoleForthWords – have name fields, code fields, parameter fields, link fields, etc.
- CreoleForthBundle – an assemblage of the previous entitites.

The stacks

- Data Stack.
- Return Stack
- Vocabulary stack
- Others *

The dictionary

- Two parts
- Each entry is accessible as a named property in a Creole Forth bundle, which has a CreoleForthWord as its value.
- The identical CreoleForthWord is stored as an array or list member accessible by an integer index.
- This setup allows the colon compiler to store integer tokens in the parameter field which are looked up by the doColon method.

Control structures

- IF-ELSE-THEN
- BEGIN-UNTIL
- DO-LOOP/+LOOP
- For DO-LOOP, I, J, and K are available as built-in indexes.

Other features

- Single-line comments (\ for JavaScript version and // for Python).
- Multi-line comments (--).
- Help. This is used by VLIST.

What they don’t have

- FORGET
- Many return stack primitives, like RDROP.
- COMPILE, [COMPILE], or POSTPONE.
- Recursion
- As many words as most Forths (< 100 right now).

Creole Forth Execution

- Values in the input area are split based on the space delimiter and placed into the ParsedInput.
- Each value is looked up in the dictionary by the outer interpreter. The outer interpreter appends each word with a value on the vocabulary stack and looks it up in the dictionary. The vocabulary stack is searched from top to bottom.
- If a search succeeds the search process is halted and the word is executed.
- If it fails, the next vocabulary is searched.
- If no match is found, the value is pushed onto the data stack.

Types of words

- Primitives – These are written in the host language, then introduced with the BuildPrimitive method. All primitives take a GlobalSimpleProps as a parameter.
- High-level definitions. Defined by the colon compiler.
- Compiling words. Have separate words for compile-time and run-time execution and are used for branching and looping.
- Defining words – use CREATE and/or CREATE/DOES>.

Colon compiler

- No state variable
- Compilation starts when the IMMEDIATE vocabulary is pushed onto the vocabulary stack.
- It ends when a semicolon is encountered. The IMMEDIATE vocabulary is popped off the vocabulary stack.
- All IMMEDIATE words are in the IMMEDIATE vocabulary, which is always searched first during compilation.

Colon compiler, Part 2

- From the point of view of the colon compiler, words are in three classes:

1. COMPINPF. Words that are looked up and whose tokens or addresses are compiled into the parameter field.

2. EXECUTE . Words that are looked up and executed such as compiling words.

3. EXEC0. Words that directly manipulate the pointer of the outer interpreter such as comments.

Colon compiler, Part 3 – tokens and actions set up in the PADarea for
: TEST1 IF HELLO ELSE TULIP THEN ;

Word5	Token	Acion
IF	52	EXECUTE
HELLO	5	COMPINPF
ELSE	53	EXECUTE
TULIP	6	COMPINPF
THEN	54	EXECUTE

Colon compiler, Part 4

- For words with a COMPINPF or EXECUTE action, the compiler puts its address in the PADarea next to its associated action.
- Words with an EXEC0 action simply move the outer interpreter pointer past their closing delimiter.
- Once all the words are looked up and in the PAD area, their tokens are simply placed on the stack and executed by their associated action by the outer interpreter.

JavaScript vs Python

- JavaScript OOP is prototyped-based, and classless.
- Python is more conventionally class-based.
- Despite this, the structure of the JavaScript and Python implementations of Creole Forth are very similar.
- JavaScript's flexibility allows adding "syntactic sugar" that can resemble more conventional OOP in other languages.

JavaScript challenges

- Detecting stack underflow. A pop on an “empty” array will blithely return an undefined item.
- Default array behavior can be overridden by working with Array. Prototype, but is not recommended.
- Strategy adopted: Have primitives that affect the data stack do their own stack checking.

Python challenges

- Python is much stricter about type conversions than JavaScript is.
- In the Python implementation only integers and floats can be compiled as literals, while in the JavaScript version strings can be treated as literals too.
- The above is subject to change.

How to use Creole Forth for JavaScript

- Just reference it in a web page
- <script src="CreoleForth.js"></script> or
- <script src=" https://github.com/tiluser/cfjs/blob/master/CreoleForth.js"></script>
- It does not require a web server
- You can use it with other libraries such as Angular, but it's not necessary.

How to use Creole Forth for Python

- 1. Embedded in Python code:

  from CreoleForth import *

  gsp.InputArea = “1 2 + .”

  cfb1.Modules.Interpreter.doParseInput(gsp)

  cfb1.Modules.Interpreter.doOuter(gsp)
- 2. Run from a script

  Put the Forth commands in a file such as script.f and run the following:

  python runcfpyscr.py script.f

Where to get them

- Creole Forth for JavaScript: https://github.com/tiluser/cfjs/
- Creole Forth for Python: https://github.com/tiluser/cfpy/
- If you want to try out Creole Forth for JavaScript online, it's available at http://jmoshowcase.com/cfpage.html .

Demo for JavaScript

- VLIST
- HELLO
- Arithmetic
- TEST – high level definition
- EVAL

Demo for Python

- python runcfpyscr.py script1.f – executes HELLO primitive
- python runcfpyscr.py scrip21.f – executes VLIST
- python runcfpyscr.py script3.f – conditional execution example

Model-View-Controller Example

- Blocitoff – a student project built with AngularJS.
- It's a simple to-do organizer,
- Creole Forth is used in two places: the Task service and the Pasttasks controller.
- Task service uses DTASKSTAT to set incomplete tasks to inactive if a grace period is exceeded.
- Pasttasks controller has the following definitions
- MRT (-- lctask) Pushes last completed task onto the stack
- SHOW (-- task( ) Pops up alert box showing task description and time
- MRTS – high level definition combining MRT and show.

Questions?