Implementing a Forth

- Forth Background.
- Why a Forth?
- Stack Machines.
- Implementation Concepts.
- Execution & Threading.
- Stacks Operations & Postfix.
- *epop* Overview & Examples.
- Resources.
Jack.Pope@ieee.org

- Investment data science infrastructure on UNIX, since ancient times.
- Chairman, Twin Cities IEEE Computer Society.
- Computer Science / Data Science, Minnesota State Colleges & Universities.
- Developer of epop, a Forth inspired programming environment.
Disclaimer

Programming Languages may have standards
(IEEE POSIX, ANSI Forth, ANSI C, ... )

Compiler implementations have principles.
(No rigid rules.)
Forth History

Developed by Charles Moore in the late 1960’s.

- A student of John McCarthy at MIT in the 1950s.
- Possibly influenced by McCarthy’s LISP programming ideas.
- Forth was “functional” long before there was Functional Programming.
Why Forth?

• The efficiencies of a stack machine.
• Can be self-hosted and be its own OS.
• A compiler-implementation paradigm.
• More than a Programming Language.
Why Forth, cont’

• A problem solving language:
  • Compact / concise expressions.
  • Self-documenting syntax.

• Factoring words:
  • Identify general problem / solution.
  • Identify most basic component words.
  • Inductively compose solution of component words.
Host vs Guest System

- Hosted: Host language defines Forth dictionary.

- Self-hosted: Guest system language defines Forth dictionary (minimal machine level / assembly functions).
Minimal Host Components

- Push function: for data stack
- Pop function: for data stack
- Data Stack
- Program Stack
Stack Machines

- Stack: Dedicated registers or dedicated area of memory.
- Stack data is Last-In-First-Out (LIFO).
- Program Stack: instruction sequence.
- Stack Counter: element count; size-of.
- Stack Pointer (top of the Program Stack):
  - Memory address of next instruction.
  - May be indexed by Program Counter.
- Push data (to top of Data Stack).
- Pop data (from top of Data Stack).
- Return Stack:
  - Addresses of functions that call other functions (return address) for continuing program sequence.
  - And/or auxiliary data stack for the current operation.
Forth Execution

• Compile-time generation of host language functions.

• Compile-time generation of guest language functions. (Like Forth’s CREATE DOES> sequence)

• Run-time Virtual Machine: loop -> word parse / tokenize -> stack(s) -> exec
The Virtual Machine Loop

- Read text input -- via user interface or file i/o.
- Interpret / parse -- one or two passes with look-ahead tokenizer.
- Generate high level program (abstract word tree).
- Recursively flatten tree to low level program stack.
- Evaluate program stack.
- Repeat
Indirect Threaded Code

- Portable: No predefined function addresses (not direct).
- More low-level jumps than direct threaded code.
- Replace words (abstract functions) with:
  - Primitive addresses
  - Intermediate opcode
  - Intermediate abstract object (token or subroutine threading)
- Dispatch the replacements to program stack.
Word Dispatch: Vectored Execution

- Replace input vector of abstract words with executable objects.
- Flatten abstract word tree into executable program stack (indirect threading).
  - Use recursive descent operations with
  - Switch statement (switch threading).
- Identify next word (opcode, address,...).
  - IF condition is 1 “immediate” then exec.
  - ELSE push word to program stack.
- Advance stack pointer/counter.
- Execute the program stack.
Stack Operations: Postfix

• Efficient for memory and CPU.
• No rules of precedence.
  • No need of ( ) parentheses, unlike infix notation.
• Linear processing from left-to-right; top-to-bottom.
• Think of assembly’s prefixed notation, in reverse.
• Ex: 2 1 + --> 3
## Separate Stacks

<table>
<thead>
<tr>
<th>Operand</th>
<th>Operator</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Three Separate Stacks (before operations).

Copyright (C) Jack Pope 2023
Three Separate Stacks (after one operation).

<table>
<thead>
<tr>
<th>Operand</th>
<th>Operator</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>+</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright (C) Jack Pope 2023
Separate Stacks

Three Separate Stacks (after two operations).

Operand Operator Return
4
8
A Combined Stack (before & after one operation).
epop overview

- Most operations are postfix and stack oriented.
- Program stack: Linear linked-list.
- Three data stacks: Circular linked-lists (for memory management).
- Data types: numeric, string, table and XT.
  - Use tables for “big data.”
  - XT (execution tokens) can be treated as data.
- Run-time user defined words (in REPL).
- Compile-time “Forth” word definitions and programs.
- Compile-time primitives (D & C host code).
  - D APIs for SQLite (RDBMS) and Curl (networking).
Example: Put data on stack

```
epop> 1 3 5 7 9
epop> .S
Dat stack:
  Value  index  address
  9      4     82A163630
  7      3     82A163580
  5      2     82A165E70
  3      1     82A165C60
  1      0     82A1659A0
epop>  
```
Example: Data stack as “program”
Example: Sum data on stack
Example: Define word to sum data

epop> : SUMDAT { + } GSC i- LREPEAT ;
epop> ODDS
epop> SUMDAT . CR
25
epop>
Example: Define factorial as map & reduce operations

```plaintext
epop> : MAP DUP IF DUP >R i- RECUR ELSE THEN ;
epop> : REDUCE GRC IF R> * RECUR ELSE THEN ;
epop> : FACT MAP REDUCE ;
epop> 5 { FACT . } LE 120
```
Example Program: ASCII printout

```plaintext
( ascii.epop )
( print out ascii table )
: tab 9 EMIT ;
: hdr chr . tab dec . tab bin . tab tab oct . tab tab hex . CR ;
: ascii_row DUP 0 BASE . tab DUP 10 BASE . tab DUP 2 BASE . tab DUP 8 BASE . tab DUP 16 BASE tab . CR ;
hdr
64 { i+ ascii_row DUP } 26 LREPEAT
```

About ASCII:
https://www.w3schools.com/charsets/ref_html_ascii.asp
Example: Run the ASCII program

<table>
<thead>
<tr>
<th>chr</th>
<th>dec</th>
<th>bin</th>
<th>oct</th>
<th>hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>b1000001</td>
<td>101</td>
<td>h41</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>b1000010</td>
<td>102</td>
<td>h42</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>b1000011</td>
<td>103</td>
<td>h43</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>b1000100</td>
<td>104</td>
<td>h44</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
<td>b1000101</td>
<td>105</td>
<td>h45</td>
</tr>
<tr>
<td>F</td>
<td>70</td>
<td>b1000110</td>
<td>106</td>
<td>h46</td>
</tr>
<tr>
<td>G</td>
<td>71</td>
<td>b1000111</td>
<td>107</td>
<td>h47</td>
</tr>
<tr>
<td>H</td>
<td>72</td>
<td>b1001000</td>
<td>110</td>
<td>h48</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>b1001001</td>
<td>111</td>
<td>h49</td>
</tr>
<tr>
<td>J</td>
<td>74</td>
<td>b1001010</td>
<td>112</td>
<td>h4A</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>b1001011</td>
<td>113</td>
<td>h4B</td>
</tr>
<tr>
<td>L</td>
<td>76</td>
<td>b1001100</td>
<td>114</td>
<td>h4C</td>
</tr>
<tr>
<td>M</td>
<td>77</td>
<td>b1001101</td>
<td>115</td>
<td>h4D</td>
</tr>
<tr>
<td>N</td>
<td>78</td>
<td>b1001110</td>
<td>116</td>
<td>h4E</td>
</tr>
<tr>
<td>O</td>
<td>79</td>
<td>b1001111</td>
<td>117</td>
<td>h4F</td>
</tr>
<tr>
<td>P</td>
<td>80</td>
<td>b1010000</td>
<td>120</td>
<td>h50</td>
</tr>
<tr>
<td>Q</td>
<td>81</td>
<td>b1010001</td>
<td>121</td>
<td>h51</td>
</tr>
<tr>
<td>R</td>
<td>82</td>
<td>b1010010</td>
<td>122</td>
<td>h52</td>
</tr>
<tr>
<td>S</td>
<td>83</td>
<td>b1010011</td>
<td>123</td>
<td>h53</td>
</tr>
<tr>
<td>T</td>
<td>84</td>
<td>b1010100</td>
<td>124</td>
<td>h54</td>
</tr>
<tr>
<td>U</td>
<td>85</td>
<td>b1010101</td>
<td>125</td>
<td>h55</td>
</tr>
<tr>
<td>V</td>
<td>86</td>
<td>b1010110</td>
<td>126</td>
<td>h56</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>b1010111</td>
<td>127</td>
<td>h57</td>
</tr>
<tr>
<td>X</td>
<td>88</td>
<td>b1011000</td>
<td>130</td>
<td>h58</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>b1011001</td>
<td>131</td>
<td>h59</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
<td>b1011010</td>
<td>132</td>
<td>h5A</td>
</tr>
</tbody>
</table>
Example Program: Fetch HTTP data

```plaintext
( dog-is-dog.epop )
CrAzYpAsSwOrD pssw !
SomeUser unam !
datamart.systemgoats.com/a-dog-is-a-dog.txt url !

url @
unam @
pssw @

HTTPGET
```
Example: Run HTTP Fetch

epop>
epop> dog-is-dog 1 RUN CR
A dog is A Dog
by T. S. Eliot

Now dogs pretend they like to fight;
They often bark, more seldom bite;
But yet a Dog is, on the whole,
What you would call a simple soul.
Of course I'm not including Pekes,
And such fantastic canine freaks.
The usual Dog about the Town
Is much inclined to play the clown
And far from showing too much pride
Is frequently undignified.
He's very easily taken in-
Just chuck him underneath the chin
Or slap his back or shake his paw,
And he will gambol and guffaw.
He's such an easy-going lout,
He'll answer any hail or shout.

Again I must remind you that
A Dog's a Dog - A CAT'S A CAT.
Example: See user defined words
Example Program: Run from CLI

```
fbsdev ~/epop:
fbsdev ~/epop: epop RUN $EPOP_HOME/APPS/hello.epop
Hello_World!
fbsdev ~/epop:
```
Related Resources on the Internet

- https://www.forth.com/starting-forth
- http://forth.org/compilers.html
- https://users.ece.cmu.edu/~koopman/stack_computers
- http://www.complang.tuwien.ac.at/forth/threaded-code.html
- https://compilers.iecc.com/crenshaw
- https://openfirmware.info/Bindings
Forth Systems

• Forth Systems: https://forth-standard.org/systems

• Compilers written in Forth:
  • https://bellard.org/tcc/
  • https://arduino-forth.com/article/FORTH_metacompilation_intro
  • https://git.sr.ht/~vdupras/duskos/tree/master/item/fs/comp/c
  • https://www.mpeforth.com/arena/C2ForthKit.120.zip
  • https://github.com/pzembrod/cc64
https://systemgoats.com/epop.html