

A Bit About Forth

Dave Eckhardt
de0u@andrew.cmu.edu

1

Disclaimer

- I don't know Forth
- Typical Forth tutorial
 - Stack, postfix stack operators
 - Writing a simple Forth word
 - Loops
 - Stack manipulation, simple built-ins
 - Gee, tutorials take forever to write, so close with:
 - 100%-inscrutable example using Forth's full power
- I am ~40% through the inscrutable stage

2

Outline

- Forth is a language with
 - No syntax¹
 - No operator precedence
 - No operators
 - No functions
 - No variables
 - No constants
 - No loops²

3

No Syntax

- Well, hardly any
 - Whitespace-delimited sequence of digits (in the current input radix) is recognized as a number.
 - In many dialects, a dot in a number is allowed for readability or to signal double precision
 - Whitespace-delimited sequence of characters is a word.

4

Syntax Examples

- 123
- FFEB.09CA
- >entry
- 2dup
- \$entry
- *, +, -, /, etc.

5

No Operator Precedence

- Easy: no operators!
- In C, + and && and || are part of the language
 - So the language arranges for them to be evaluated according to “natural” precedence (more or less)
- In Forth, all executable things are of the same class (“words”)
- Precedence is manual (postfix stack ops)

6

Stack Operations

- 3 4 +
- Push 3 onto the stack.
 - Then push 4 onto the stack.
 - Run +
 - Which traditionally pops two things from the stack, adds them, and pushes the result on the stack. But it could be redefined to do anything else instead.
- “3 + 4 * 2” - meaning is up to you, not Forth
- 3 4 2 * +
- 3 4 + 2 *

7

No Functions

- Words aren't functions
 - They have no types
 - No parameter types
 - Pull whatever they want off the stack
 - First parameter may determine how many parameters
 - No return types
 - Push whatever they want onto the stack
 - Common idiom:
 - success $\Rightarrow$ push answers, then push “true” (-1)
 - failure $\Rightarrow$ push “false” (0)
 - Actually, nothing has any types

8

No Types

- What is the type of items on the stack?
 - “Cell” - approximately “machine word”
 - Same type as BLISS (great-grandfather of C, used to write DEC's VMS, CMU's Hydra)
- Some words operate on multiple cells (extended precision)

9

No Variables

- Most code operates on stack values
- Once you have “too many” values on your stack your code gets confusing
- There is a word called VARIABLE
 - It doesn't “declare” a “variable”, though.
 - It allocates a cell and compiles a word which pushes the address of that cell on the stack.

```
VARIABLE FOO
FOO @ 3 + \ Add 3 to contents of FOO
```

10

VALUE

- If a “variable” will be read more than written, you can use VALUE instead.
 - It places a value into a freshly-allocated cell and compiles a word which fetches the contents of the cell and pushes it on the stack

```
0 value BAR
BAR 3 + \ Add 3 to contents of BAR
4 TO BAR \ sets BAR to 4 - advanced
```

11

No Constants

- There is a word called CONSTANT, though.
 - Can you guess what it does?

12

No Loops

- The language *does* provide words which implement loops
`10 1 DO I . CR LOOP`
- But these words aren't privileged – you can write your own which work just as well.
 - UNLESS, UNTIL, WHEREAS... - go wild!

13

Is There Anything There?

- No...
 - No syntax¹
 - No operator precedence
 - No operators
 - No functions (no types)
 - No variables
 - No constants
 - No loops²
- So what *is* there?

14

Parts of Forth

- “The Stack”
 - Really: the operand stack
- The other stacks
 - Call/return stack – (ab)used by loop words
 - Exception stack – if exceptions are available
- The Dictionary
 - Maps word names to execution tokens
- The “Compiler”
- The “Interpreter” (read loop)

15

“Compiler”

- “Compiler” stitches together code bodies of existing words
`: addone 1 + ;`
- Not a “function definition” which begins with the “:?” token and ends with the “;” token
- : grabs a word from the input stream, saves it “somewhere”, and turns on the compiler
- The compiler creates code sequences for pushing numbers and pushing calls to words

16

“Compiler”

- ; stops the compiler and adds a dictionary entry mapping the name token to the execution-token sequence
- Where's the code?
 - Here comes a vague analogy...
 - ...C code which when compiled would have similar effect to Forth...

17

The Code

```
/* “threaded code” style */

typedef void (*notfun)(void);
notfun push1, plus;
notfun addone[] = { push1, plus, 0 };

void execute(notfun a[])
{
    while (a[0])
        *(a[0]) ();
}
```

18

Threaded Code

- Easy to generate machine code which just calls other machine code
- Also easy to generate machine code for “push integer onto stack”
- Handful of built-in words must be written in assembly language
 - Peek, poke (@, !)
 - +, -, *, /
 - Compiler itself

19

Isn't Threaded Code Slow?

- Other organizations possible
 - Can peephole-optimize threaded code pretty well
 - Can “cache” top N words of stack in registers
 - Can do real optimizing compiler if you want

20

Are We Having Fun Yet?

- Why would people do this?
 - Great for memory-constrained environments
 - Forth runtime, including compiler, editor, “file system”, “virtual memory” can be implemented in a few kilobytes of memory
 - Stacks very small for real applications (small number of kilobytes)
 - **Very** extensible
 - Want software VM? Just redefine @, !
 - “Hard” things may be trivial
 - De-compiling Forth is often very easy...

21

Are We Having Fun Yet?

- Why would people do this?
 - A trained person can bring up a Forth runtime on just about any system in around a week given assembly-language drivers for keyboard and screen
 - GCC+glibc ports to new processors typically take a **little** longer than that...

22

Is Forth Usable?

- It's missing:
 - types, type-checking, pointer-checking
- How can code written this way work?
 - Oddly enough, very well.
 - Forth advocates claim it promotes careful thought. Also, most words are short enough to be solidly tested.
 - Another slant: No way to avoid paying attention.
 - Another slant: anybody who can wrap their mind around it is a **very** good programmer...

23

Curiosity or Language?

- Who uses this?
 - OpenFirmware (every Macintosh since...1996?)
 - PostScript allegedly inspired by Forth
 - Embedded firmware
 - Astronomers...since the 1960's
 - **Lots** of things in space run Forth
 - <http://forth.gsfc.nasa.gov> – very partial list

24

Who Should Learn Forth?

- Long-hair hacker types might find it fun
- Embedded-systems programmers might find it useful
- CS majors might find it challenging

25

Further Reading

- Forth - The Early Years
 - <http://www.colorforth.com/HOPL.html>
- The Evolution of Forth
 - <http://www.forth.com/resources/evolution/>

26