

Forth / Open Firmware

By:

Joseph Ronald Cañedo

Introducing Forth

- Forth was originally written by Charles (Chuck) Moore in 1970 to control the 30 ft telescope at the Kitt Peak Observatory.
- Forth is predominantly written in Forth and is interpreted by itself at runtime.
- Forth is at once a compiler, an interpreter, and, in a fashion, an operating system too.
- In Forth, the words that a programmer writes directly manipulate the stack, taking parameters off and putting new values (return parameters) back.

Introducing Forth

- Forth uses a form of syntax known as Reverse Polish Notation (RPN).
- So the conventional expression $a + b$ becomes $a b +$ in RPN.
- To display this result, use dot (.).
- So:

4 5 + .

5 25 98 + + .

String Words

- How to print out a string of characters?
- The word to output a string is `."` (pronounced dot quote) and is used thus:
 - `." hello world"`
- Note the space between `."` and the string, and the absence of a space before the final quote. That's because `."` is a Forth word, and all words must be separated by a space. The final quote character is not a Forth word, merely a string terminator.
- A carriage-return/line-feed pair is output using the Forth word `cr`, and a space is output using `space`. For more than one space, use the `spaces` word that takes a parameter from the stack to specify the number of spaces to be printed.

Stack Manipulation

- Since all words operate using parameters on the stack, Forth provides tools for the programmer to change the contents of the stack, or to alter the order of values on the stack.
- The Forth word `.s` shows the current stack without removing any entries. This is a useful tool for the programmer, enabling you to nondestructively monitor the stack.
- Other useful stack words are `dup` (duplicate the entry on top of the stack), `drop` (discard the entry on top of the stack), `swap` (exchange the two top entries on the stack), and `over` (copy the second-to-the-top entry to the top of the stack).

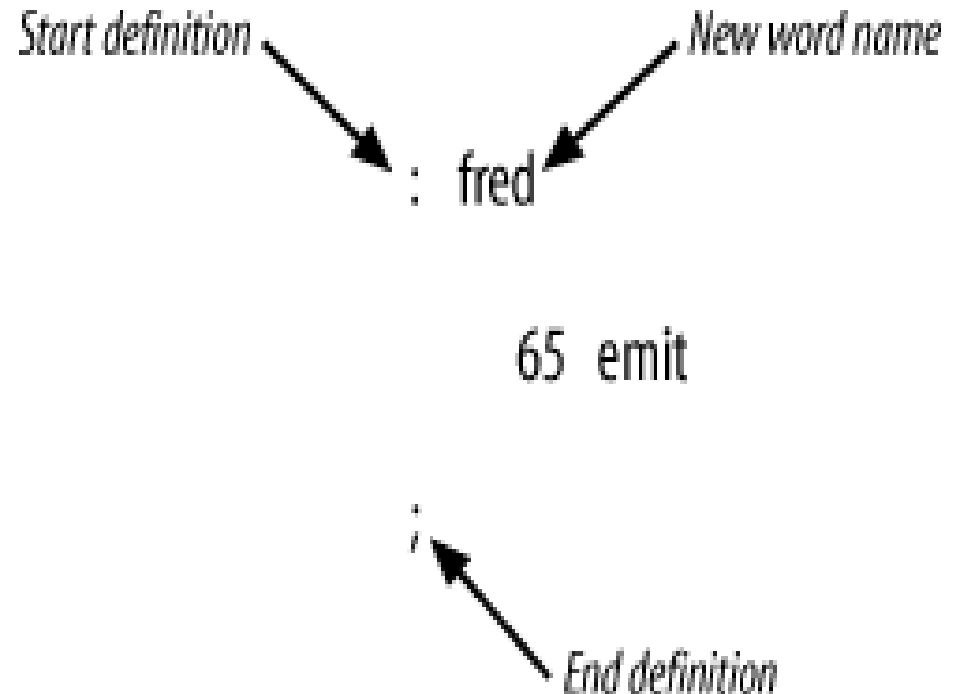
Stack Manipulation

- For example, entering 5 dup results in 5 5 on the stack. 3 4 swap results in 4 3 on the stack. 1 2 over results in 1 2 1 on the stack. dup is most often used to preserve stack entries when they are used with words that remove values from the stack.
- For example, here is Forth code that prints out the ASCII character A with its corresponding numeric code:

```
65 dup emit space . cr
```

Creating New Words

- A new word is created by using a colon definition.



Comments

- Comments in Forth are enclosed in parentheses:
(Here is a comment)

```
: helloworld (-- , says hi)
```

```
  ." hello world"
```

```
;
```

```
: square (A -- A*A , squares top of stack)
```

```
  dup *
```

```
;
```

if ... else

- Forth has if ... else ... like other languages, but the structure is quite different from that which you are used to. if treats the topmost stack entry as a boolean. Any nonzero value is considered as TRue. Here is some sample code that shows you how to use if:

```
: is_it_true
    if
        ." That is true"
    cr
    else
        ." Not true"
    cr
then
```



Loops

- Loops may be created using the Forth words `do` and `loop`. The following simple example prints out five new lines:

```
5 0 do
  cr
loop
```

Same as for loop in C programming

```
for (i = 0; i < 5; i++)
{
    printf("\n");
}
```

Data Structures

- The fact that Forth uses a stack for parameter passing between words makes it very powerful. It is because of the stack that the language is re-entrant and supports recursion. However, the dynamic nature of the stack makes it unsuitable for global data. Fortunately, Forth is not limited to using the stack, as the language provides for both named constants and variables.
- A constant is declared by placing the value for the constant onto the stack, then the word constant followed by the constant name. For example, the following code declares a constant called kilobyte (representing the number of bytes in a K) to have a value of 1024:

```
1024 constant kilobyte
```

Interacting with Hardware and Memory

- Forth is great for debugging embedded hardware. You can use Forth to examine the contents of memory or peripheral registers:

```
2000 @
```

- This places the content of address 2000 onto the stack. You can also use Forth to set memory or peripheral registers:

```
41 2000 !
```

- This sets the content of address 2000 to 41. Remember that you can do this from within a word or interactively from the command prompt. This means that from the prompt, you can interactively probe and change peripheral registers—a very powerful debugging tool.

Forth Programming Guidelines

- As with any other programming language, it's possible to write good Forth, and just as easy to write bad Forth. Here are some Forth tips:
 1. Limit your stack usage to three or four words.
 2. Use several short word definitions rather than one long word definition.
 3. Keep it simple and focus on what you're trying to achieve.
 4. Check for data that is out of range or erroneous and generate useful error messages or diagnostics

