Albert Nijhof 26.06.2016

What's in a name?

(About 'data managers' in forth)

In texts that try to explain forth to non-forthers you find sometimes triumphantly presented examples like this:

```
: 1  ." Hello! " ;
```

So flexible! It shows in fact a fundamental weakness in the forth interpreter.

The simple interpreter

The forthinterpreter is very simple: it reads the following name in the input stream ...

Wait! To be precise: in forth a "name" is a contiguous series of characters (ASCII 21-7E), bounded on both sides by spaces, line start or line end. There are, apart from the space, (almost) no characters that receive a special treatment.

Again:

▶ The interpreter reads the next name from the input stream and tries to find it in the dictionary.

- Found? - execute or compile it, depending on `STATE` and `IMMEDIATE`.
- Not found? - treat it as a number. If that does not work out: stop the proces and report an error.

I understand that as follows:

▶ The interpreter reads the next name from the input stream, finds it in the dictionary and executes or compiles it, depending on `STATE` and `IMMEDIATE`. When the name is not found the interpreter has the generosity to treat it as a number. When that does not work out an error is reported.

Of course that comes down to the same thing. The only difference is that my version suggests that it should not be a task of the interpreter to process numbers.

Actually very strange: the programmer knows that he types a number and yet the interpreter will check if that number is a name – in vain, we may hope.

What's in a name?

This is the flaw: data is first treated as a name because the interpreter can not read minds.

Ideally data in forth code should be unmistakably identifiable as data!

The task of the interpreter to recognize data in the input stream is expanding and seems to become a main task. The interpreter has to check names on the presence of special characters as in `#10 &10 %10 "a "b" &a 3.E2 'a C:` etc. in order to classify them as a number, double number, floating point number, string, or whatever. The knowledge of those special characters and their possible positions must be available in the interpreter. This is what we call a Swiss army knife! Even special structures (recognizers) are proposed to achieve this. If those structures are productive, i.e. expandable by the user, a syntax is smuggled into names (see Visions Below).

It is unfortunate, but especially for small forths this is not attractive.

Names and data can be confused due to their appearance. This means that choosing names for new definitions is restricted in an unpredictable way. You better have to avoid names like `A. 2, #3 V: BAD` and that is at least a remarkable limitation.

Intermezzo (Visions)

Modern forth	Old-fashioned forth
`:AMSTERDAM`	`: amsterdam`
`\This is comment`	`\ This is comment`
`CON:LONDON`	`constant london`
`VAR:PARIS`	`variable paris`

Also very useful:

`!PARIS`	`paris !`
`@PARIS`	`paris @`
`VAL:BERLIN`	`value berlin`
`TOBERLIN`	`to berlin`

Extremely convenient:

`,IF`	`postpone if`
`,THEN`	`postpone then`

You do not have to limit yourself to two-part composites:

`ON@STATE`	`state @ if`
`@STATE=0`	`state @ 0=`
`ON@STATE=0`	`state @ 0= if`
`ON@PARIS=200`	`paris @ 200 = if`

And to reduce that annoying postfix thing a little further:

`+LONDON`	`london +`
`*LONDON`	`london *`
`-LONDON`	`london -`
`+@PARIS`	`paris @ +`
`-@PARIS`	`paris @ -`
`+245`	`245 +`
`-245`	`...`

Hey, that's a pity. Maybe ...

`-245`	`-245`
`--245`	`-245 -`

...?

A simpler interpreter

But seriously now: let's have a look at the very forth-fundamentalistic idea of having an interpreter that does not accept any numbers:

```
: INTERPRET
    begin bl word find   dup
    while 0< state @ and
          if compile, else execute then
    repeat
    0= abort" Name not found " ;
```

That's what I call a simple interpreter!
But what to do with data?
There is a simple and effective solution:

Datamanagers

"Data managers" are forth words that announce, read and interpret data in the input stream. They are specialists. Each data type has its own own data manager. A data type no longer needs to be detected.

Let's first create a word `N` that processes single numbers. Note that there is always at least one space between data manager and data. Example:

```
HEX
: VISIBLE? ( x -- flag )
  n 21 n 7F within ;
```

Before you drop out:
Of course I don't wish to argue that this `N` must be introduced in forth. I use it here only as the simplest example to show you how all those other data types could be handled.

Examples of data managers

```
NN 100 10 \ 2 numbers
DN 100    \ double number
FL 100    \ floating point number
HX 100    \ hexadecimal number
DM 100    \ decimal number
BN 100    \ binairy number
CH C      \ ASCII code of C
CTRL C    \ control-C as a number
XT C@     \ token of C@
S "ccc"   \ address/length of the string
CS "ccc"  \ address of counted string
```

About the strings (slightly irrelevant in this context): remove the delimiter (the quote) from `S"` and put it just in front of the string, the result is better readable than with the classic `S"` where you have to think away that space. Moreover, you can now choose your own delimiter (the first visible character after `S` is the delimiter) for example for strings with quotation marks in it:

```
S "red wine" S -"green" wine- CS    ' Ciao! '
S "red wine" S -"green" wine- CS    ' Ciao! '
```

Effects

▶ Instead of forcing the interpreter to look up data in the dictionary (in vain) and then to analyze it in order to decide what to do with it, the programmer puts the correct data manager in front of the data.

▶ The data manager knows how to handle the data. The interpreter no longer needs to detect data and its type. This increases readability, both for the interpreter and for human beings. (At least in the long run, because `DN 100` or `FL 100` will be a bit of a shock for some people. Changing habits can be hard.)

▶ The data manager forms a tandem with the data. That tandem behaves as a whole and is state-smart, just as numbers are in an ordinary forth: depending on `STATE` the data is compiled or put on the stack. See also "state-smart phobia" below. The data manager itself is immediate and is never compiled. Postponing a data manager is at your own risk.

▶ You can define data managers for any data type on top of each and every forth. That's especially useful for small forths, they do not have to be prepared for all kinds of data.

▶ A name in forth is a series of visible characters, none of which gets a special treatment.

> The problem with `: 1 "Hello!" ;` is removed, because now there is a difference between the number `1` and the name `1`.

State-smart phobia, CH and XT

`[CHAR]` and `[']` have been included in the standard at the time. Data managers are an alternative:

```
CH *
XT DROP
```

Use `CH` and `XT` when the data follows immediately,
use `CHAR` and `'` when the data does not follow immediately.

These rules are easier and less esoteric than those for `[CHAR]` and `[']`. And we did not yet talk about the graphic unattractiveness of these names, with those square brackets.

I would like to put the state-smart phobia into perspective. Indeed, you can invent state-smart constructions - with FOO, BAR and sufficient POSTPONEs or EVALUATEs in it - to show that sometimes strange things can happen, but nobody forces you to write that kind of programs.

P.S.

> A forth interpreter should handle only single numbers. All other data require a data manager.