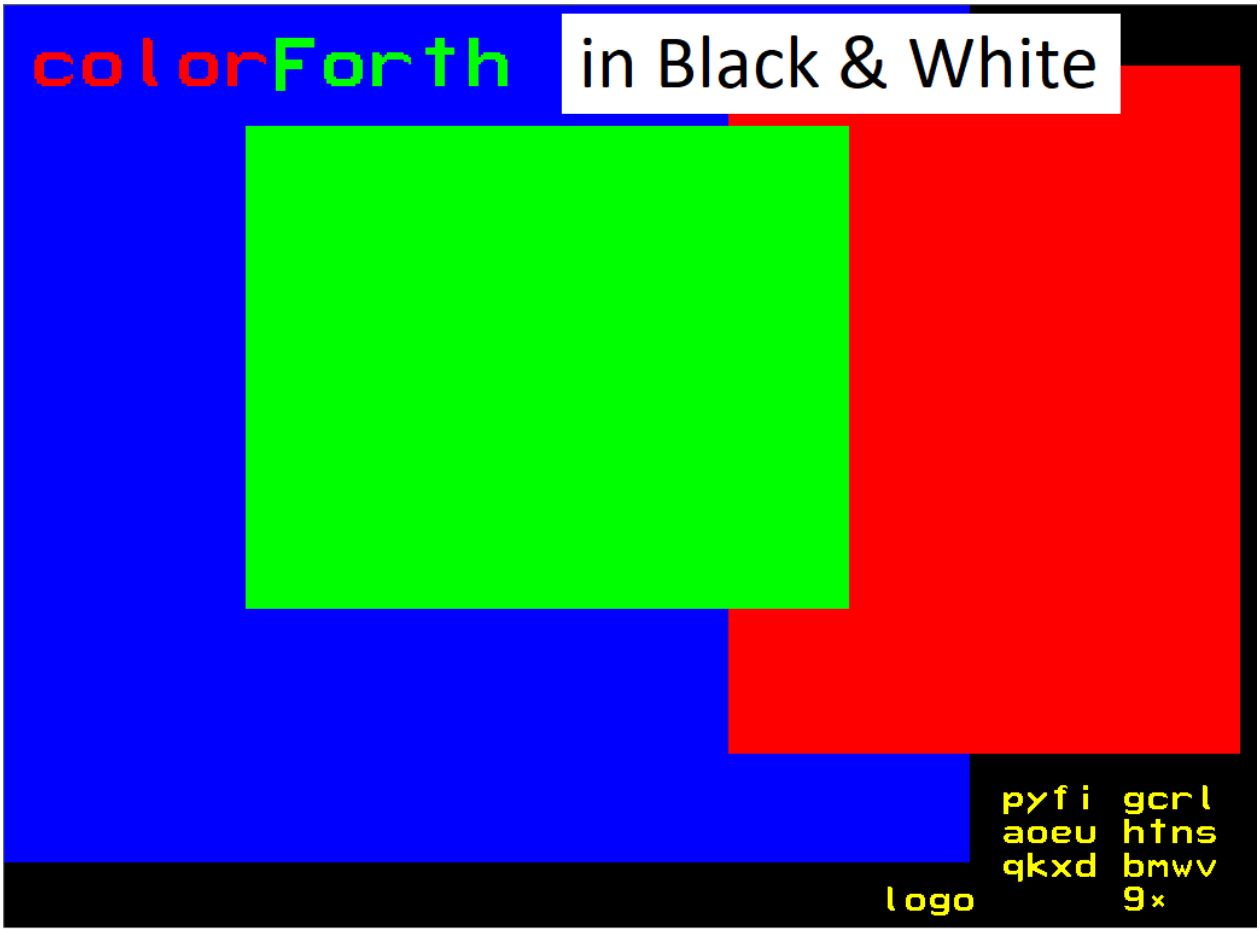

Contents

Summary

colorForth is a dialect of the Forth programming language, both of which languages were invented by Charles H. "Chuck" Moore ; - Forth around 1968, and colorForth in the late 1990's.

In this paper I hope to explain why colorForth is about so much more than just colour.

colorForth uses a 32 bit token as the basic unit of interaction between the computer and human being. Each token has a 28-bit human readable name field and 4 bits of meta-data ("colour"). The token's meta-data field can replace global variables such as STATE, allowing a simpler compiler and a more complex/powerful editor.

Conventional programming environments separate the editor, compiler and interpreter into discrete functional units, whereas colorForth puts them all into a blender and filters the resulting mush into something completely different.

Emergent Properties

One emergent property of the colorForth environment is the Magenta Variable, where setting a new value at run-time actually affects edit-time, by changing the pre-parsed source for the program.

Another is the Blue Token, which controls the behaviour of the editor at edit-time (over and above seeing what you type). This is similar to putting CRs and TABs into a text file, but Blue Tokens are extensible – you can run Forth code at edit-time.

Because the boundaries between editor, compiler and interpreter are blurred, you can choose what you do, and when you do it, much more easily than in a conventional environment. For example, version control could be added using Blue Tokens to retrieve earlier versions of code at edit-time and compile-time.

Shortening the Conceptual Gap

Edsger Dijkstra in his 1968 paper Go To Statement Considered Harmful states that:

"**we should […] do our utmost to shorten the conceptual gap between the static program and the dynamic process […]**", which I interpret as "**shorten the conceptual gap between *source text* and *program execution*"**. That is, make it as easy as possible for someone reading the source to create a conceptual model of what the program will do when it runs.

To make a program clear and easy to understand, each word should have a name with some mnemonic value, and should do something simple that is hinted at by that name. In this context, a "goto" means that something else happens while this word is executing which is most likely *not* hinted at by the word's name – this is therefore a bad thing.

Giving a Forth word the correct name is of course important, but by adding meta-data, the word (now a token) stores more information – not just what it does, but when it should do it : edit-time, compile-time or interpret-time.

Removing Punctuation

"Shortening the conceptual gap" applies to any computer language – Forth takes it to an extreme by defining a "word" as a sequence of characters surrounded by spaces, and leaving it to the author to decide about almost everything else.

Other languages add a more complicated syntax, restricted keywords and program style guides in order to lock people in to that language. From a Forth perspective these additions are just noise – they do nothing to shorten the conceptual gap between source and program. From a financial perspective these additions increase profit.

When Chuck Moore created colorForth one of his intentions was to use colour to replace punctuation:

```
( Editor Display ) [ mvar cblind 0 ]                    228
: cb cblind @ 0 + drop ; [ mvar state 16 state+ 16
]
: yellow $ffff00 color ;
: +txt white $6d emit space ;
: -txt white $6e emit space ;
: +imm yellow $58 emit space ;
: -imm yellow $59 emit space ;
: +mvar yellow $9 emit $11 emit $5 emit $1 emit spa
ce ;
```

becomes:

```
Editor Display cblind 0                                 228
cb cblind @ 0 + drop ; state 16 state+ 16
yellow $ffff00 color ;
+txt white $6d emit space ;
-txt white $6e emit space ;
+imm yellow $58 emit space ;
-imm yellow $59 emit space ;
+mvar yellow $9 emit $11 emit $5 emit $1 emit space
;
```

While the use of colour to replace punctuation is an interesting idea, it ultimately fails as a general-purpose programming language because a surprisingly high percentage of people are colour-blind. According to Wikipedia, red-green color blindness affects up to 8% of males and 0.5% of females of Northern European descent. It also makes it difficult to exchange "pure" colorForth source code in a monochrome text file.

I should point out here that when I work with Forth source in text files (*.f) I use my favourite editor (EditPlus) with a Forth colouring option, so the text appears in colour – but this has absolutely nothing to do with the use of colour in colorForth.

In colorForth, colours are just a *representation* of the "color" of the token, the bottom four bits of the token value. It is very easy to modify the colorForth editor to add conventional Forth punctuation. That is, the meta-data can be used to control what the user sees in the editor, what the compiler compiles or what the interpreter does.

Metadata and Colour

While the name “colorForth”, the coloured representation colorForth and the colourful appearance of the display all emphasise colour (spelled “color” in the USA), in fact the fundamental principles in colorForth go way beyond colour. Colour in this context is just one way of conveying meta-information about a computer program.

For example, conventional Forth uses ‘:’ to indicate the definition of a new Forth word, colorForth uses the colour red together with starting the definition on a new line.

While conventional Forth can have coding style standards that usually specify that colon definitions start on a new line, this not required. In colorForth, red tokens (that start a new word definition) are displayed on a new line automatically. There are some special blue tokens that modify this default behaviour, and this can in any case be changed, if desired, in the NASM source code.

In the cf2019 distribution of colorForth, pressing the F4 function key toggles between colorForth mode and a more conventional Forth display. This is easy to do because the information and meta-information (information about the information) are stored as 32 bit tokens, and can be displayed in any desired way. The F4 function also makes it easier for people who are colour-blind to read the code.

Token Colours

The following colours and their meaning is described below, from file cf2019.nasm :

```
actionColourTable:          ;    * = number
    dd colour_orange        ; 0    extension token, remove space from previous word, do not change colour
    dd colour_yellow        ; 1    yellow "immediate" word
    dd colour_yellow        ; 2  *  yellow "immediate" 32 bit number in the following pre-parsed cell
    dd colour_red           ; 3    red forth wordlist "colon" word
    dd colour_green         ; 4    green compiled word
    dd colour_green         ; 5  *  green compiled 32 bit number in the following pre-parsed cell
    dd colour_green         ; 6  *  green compiled 27 bit number in the high bits of the token
    dd colour_cyan          ; 7    cyan macro wordlist "colon" word
    dd colour_yellow        ; 8  *  yellow "immediate" 27 bit number in the high bits of the token
    dd colour_white         ; 9    white lower-case comment
    dd colour_white         ; A    first letter capital comment
    dd colour_white         ; B    white upper-case comment
    dd colour_magenta       ; C    magenta variable
    dd colour_silver        ; D
    dd colour_blue          ; E    editor formatting commands
    dd colour_black         ; F
```

Four Times

There are four logical periods of time in computer programming, starting from the human being and ending up at the computer :

1. Design-time - the human being thinks about how to solve the problem at hand
2. Edit-time - the human being types a program, using the computer running a previously written program (the Editor)
3. Compile-time - and compiles the program, using the computer running a previously written program (the Compiler)
4. Run-time - then asks the computer to run the compiled program, and tests the results

In Forth, there is an outer interpreter that collects words typed by the human being and interprets them.

When developing a program, the four times follow each other in a logical progression, repeating by cycling back to design- or edit-time as required, as in a REPL read–eval–print loop.

In general it is best to concentrate on the earliest possible logical time : a problem solved at compile-time consumes less resources than solving the same problem at run time, likewise a better design-time algorithm or way of approaching a problem can save both compile- and run-time effort.

An example is modular multiplication, calculating A B N **MOD , A taken to the power B modulo N.

Montgomery Multipication (https://en.wikipedia.org/wiki/Montgomery_modular_multiplication) is a design-time improvement that maps A to Montgomery form so that taking the result modulo N can be done by dividing by a power of 2 – this is equivalent to shifting and is very much faster than division. The result is then mapped back from Montgomery form to give a usable result, with better run-time performance.

By being very simple and attaching meta-data to data, colorForth can allow more control over the entire development environment – this could allow a major design-time improvement.

Philosophy : Keep It Simple

It is not easy to define simplicity – it is more of a direction than a goal. Sometimes *adding* complexity in one area can *decrease* the complexity overall. An example is the simple Text Input Buffer in conventional Forth being replaced by a pre-parsing Shannon-Fano encoder in colorForth – but this simplifies the compiler.

From Chuck Moore's book Programming a Problem-oriented Language :

"The Basic Principle

• Keep it Simple

As the number of capabilities you add to a program increases, the complexity of the program increases exponentially. The problem of maintaining compatibility among these capabililties, to say nothing of some sort of internal consistency in the program, can easily get out of hand.

You can avoid this if you apply the Basic Principle.

You may be acquainted with an operating system that ignored the Basic Principle. It is very hard to apply. All the pressures, internal and external, conspire to add features to your program.

After all, it only takes a half-dozen instructions; so why not? The only opposing pressure is the Basic Principle, and if you ignore it, there is no opposing pressure."

I am looking forward to discovering new ways of simplifying the total colorForth system, by adding carefully controlled complexity into certain key areas.

colorForth History

Around 2001 I downloaded Chuck Moores' public domain colorForth from his website and copied on to a 3.5 inch floppy disk. It was not easy to get working – I had to add a new, compatible floppy disk ISA board to make it work.

I was impressed, and wrote the article : colorForth and the Art of the Impossible and presented it at EuroForth 2001. I also had the great good fortune to spend about 45 minutes with Chuck, looking at his colorForth CAD system, OKAD II.

I love working in colorForth – I think it must be something genetic, certainly it appears not to be curable.

I presented another paper at EuroForth 2003 "The colorForth Magenta Variable", and handed out floppy disks with the first distribution of my version of colorForth.

Time marches on, and one of my two PCs still with a floppy disk drive, died. I still have the other one, in the cellar, "just in case". But it became obvious that colorForth needed to be updated to run from a USB stick.

A decade or so later, I presented a paper "Crypto colorForth" at EuroForth 2017 (the video is here), and demonstrated colorForth running from a USB stick. I believe that security and complexity are incompatible in computer software, and that colorForth can be the basis of a very secure operating system (without using files).

Actions, not Words

I strongly recommend that you, dear reader, run cf2019 as a program on a suitable computer. There are two ways of doing this :

1. Copy the binary image file cf2019.img directly onto a USB drive, and boot the computer using this drive.
2. Run cf2019 in a bochs environment under Windows. Double click on the file **go.bat** in the cf2019 distribution to do this.

This is because "the map is not the territory" – both Forth and colorForth provide an interactive environment that is best experienced, rather than discussed.

Downloads

colorForth can be downloaded here, and can be copied to a USB drive to run native on most PCs, or under Bochs for Windows.

Documentation is available here, and is included in the distribution.

Enjoy!

Howerd Oakford 2019 Aug 31