ISSN 0265-5195

Forthwrite FIGUK

Issue 107 June 2000

FIG UK Committee

Chair	**Chris Hainsworth,**	Microplex Ltd., 5a Riverfield Road, STAINES TW18 2EE	
		01784 457565	chris.hainsworth@dial.pipex.com
Secretary	**Doug Neale,**	58 Woodland Way, MORDEN SM4 4DS	
		020 8542 2747	dneale@w58wmorden.demon.co.uk
Editor	**Chris Jakeman,**	50 Grimshaw Road, PETERBOROUGH PE1 4ET	
		01733 753489	cjakeman@bigfoot.com
Treasurer	**Keith Matthews,**	20 Spindlebury, CULLOMPTON EX15 1SY	
		01884 34818	
Webmaster	**Jenny Brien,**	Windy Hill, Drumkeen, BALLINAMALLARD,	
		Co Fermanagh BT94 2HJ	
		02866 388 253	jennybrien@bmallard.swinternet.co.uk
Librarian	**Sylvia Hainsworth,**	Microplex Ltd., 5a Riverfield Road, STAINES	
		01784 457565	sylvia.hainsworth@dial.pipex.com

Membership enquiries, renewals and changes of address to Doug.
Technical enquiries and anything for publication to Chris.
Borrowing requests for books, magazines and proceedings to Sylvia.

FIG UK Web Site

For indexes to Forthwrite, the FIG UK Library and much more, see `http://forth.org.uk`

FIG UK Membership

Payment entitles you to 6 issues of Forthwrite magazine and our membership services for that period (about a year). Fees are:

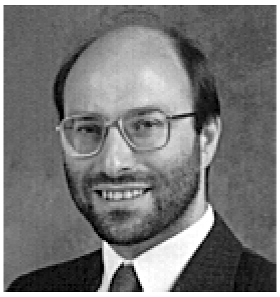
National and international	£12
International served by airmail	£22
Corporate	£36 (3 copies of each issue)

Forthwrite Deliveries

Your membership number appears on your envelope label. Please quote it in correspondence to us. Look out for the message "SUBS NOW DUE" on your sixth and last issue and please complete the renewal form enclosed.

Overseas members can opt to pay the higher price for airmail delivery.

Copyright



Editorial

Once again we've managed to combine articles on new developments with a celebration of past Forth triumphs - see John Tasgal's Introduction to Color Forth and Neal Crook on the Canon Cat.

Since announcing John Tasgal's series on the newsgroup, it has been in such demand that it is now packaged separately as a special issue of Forthwrite, which will only be available electronically.

Do you receive messages including monthly reminders about the FIG UK IRC sessions? If not, then it's because I don't have your current e-mail address - please bring me up to date, 4 bounced last month.

IRC continues to flourish - in May Michael Gassanenko stayed up to the early hours to join in from Russia. Do give it a try on channel #FIGUK on Saturday 1st July from 9:00pm.

Welcome to some more new members - Bruce Cunningham in Gloucester, Rob Probin in Glasgow (see Rob's letter in this issue) and Paul Redmond. Also a warm welcome to old member John Hayhow who has re-joined - did you notice his letter in the last issue?

Paul is a teacher in a small private school who is preparing to use eForth in a Programming Option for the 6th Form in September. He would appreciate advice from anyone with teaching experience. Please contact him on paul74@netlineuk.net

I strongly recommend that new members raid our Library to borrow books and back-numbers of Forthwrite and the US Forth Dimensions. It's the biggest and best Forth library anywhere.

Until next time, keep on Forthing,

[illegible]

PS. **International FIG**
Those who are also members of International FIG will be aware that things are not proceeding smoothly. There has been only one (double) issue of their Forth Dimensions magazine in the past year and there seem to be other problems too.

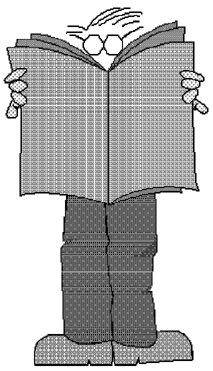
I have written to the Board of Directors recently and will publish their reply in the next issue. In the meantime, Taygeta Scientific have announced that their 3-year administration of the FIG Office has come to an end. Their commitment to web-based services continues:

> Taygeta Scientific Inc. is and will remain deeply committed to providing host service, in addition to administering and managing the forth.org website and the Forth archive/ information pages.
>
> It is our pleasure to provide this service for the Forth Interest Group and to the Forth community at large in the past and we look forward to doing so on into the future.

PS. **Forthwrite and Higher Education**
We are trying to develop a programme for encouraging interest in Forth among programmers and potential programmers who have not yet heard of it.

As part of this programme, we are recruiting members who are able to place copies of Forthwrite in Universities and Colleges, where they may appeal to people willing to look beyond the mainstream of computing. We've had positive responses from 3 members already and are hoping for a few more.

PS. **Jef Raskin**
Shortly after receiving Neal Crook's article on the Canon Cat and corresponding with its designer Jef Raskin, Jef featured in a Sunday Times article (18th June) complete with picture.



Forth News

Systems

Tom Zimmer reports that downloads from his site have approached his ISP's limits.

There have been no less than 145 downloads of his **Win32Forth** executable in the past month so there must be a lot of busy Forth people out there.

Doug Beattie has provided a functional version of **Camel FORTH** for the TRS-80. A complete Z80 Forth system with all source code, it includes a Intel .HEX file ready to serial-boot (upload using SBOOT4T.)

The system can be loaded using sBoot4T.exe v1.1, his latest enhancement to the original 19200-baud Serial Boot Loader (with terminal mode). For details, see

http://www2.whidbey.net/~beattidp/

Ralph Hempel offers a new enhancement to **pbForth** for the Lego Mindstorms RCX. The new GUI is a substitute for Hyperterminal offering:

- Fast and standard speed firmware upload
- Script upload with comment stripping
- XMODEM image download
- Console

Details at
http://www.hempeldesigngroup.com
/lego/pbFORTH

John Sadler reports that **Ficl 2.04** is now available for web download from the usual place:

http://www.taygeta.com/ficl.html

Ficl is a lightweight, efficient language designed to be incorporated into other programs, including (especially) firmware based systems. Ficl includes a simple but capable object model that can wrap existing data structures. Applications include scripting, prototyping, automation, hardware test and debug, command language for embedded targets.

European News

MPE have been looking for Beta Testers for their C to Forth compiler. Contact Stephen Pelc at sfp@mpeltd.demon.co.uk

Comsol have announces Micro-Search, their free web-based microprocessor selection service.

Initially available for the 8051 family, Comsol have collected all the data necessary for you to select a CPU from over 300 different chips from 8 manufacturers. New ones are being added as manufacturers release details.

See www.computer-solutions.co.uk/ info-zone.htm

At their recent AGM, **German FIG** Forth-Gesellschaft elected Fred Behringer as Director.

USA News

Phil Koopman, author of the book ***Stack Computers*** has published it on-line as a PDF

To get your own copy, please visit:

http://www.ices.cmu.edu/koopman/stack_computers and select the ".pdf" option in the download table.

Please do NOT redistribute the file to others (although you're welcome to pass along or link to the URL). He needs to track the number of copies in distribution, and the only way to do this is to count the hits.

Dave Pochin
01905 723037
davep@sunterr.demon.co.uk

Floating Decimal Fudge

Formatting Floating Point using Win32Forth

Dave Pochin

Another useful contribution from Dave, which this time also applies outside Win32Forth. Check the Forthwrite index for earlier items in this valuable series.

I have been exploring the Floating Point extensions in Win32Forth, I'm glad to say the maths worked fine and the results were very satisfactory. However when I started tidying up the results, the screen formatting was a mess, decimal points and signs all over the place, like this:-

```
156767830.
-178923.40

-156.76783E6
178.92340E3
```

What I wanted to see was something like this :-

```
156767832.1000
  -178923.4000
```

Oh dear, maybe using Win32Forth isn't such a good idea after all !
If there is a way of getting the format right, there should be a suitable defining word in the Win32Forth file float.f which lists all the definitions using floating point.

Looking in float.f there are five defining words for printing floating point numbers `F.`, `FE.`, `E.`, `G.` and `FS.` (Don't bother with `FS.`, it's just a rewrite of `E.`). None of these seem to have much to do with my sort of formatting, as running `.Test1` from the listing will show.

There are to two problems to tackle, dealing with the sign and right-aligning the column. The prospect of trying to write a new routine from scratch is not attractive, so a hard look at a hard copy of `F.` from the float.f file may prompt a few ideas worth exploring.

The Sign Problem
Early in the definitions of both `F.` and `FE.`, there are lines

```
      IF fabs ." –"
      THEN
```

This looks as if it follows a test for a negative number. When true, the number is converted into absolute form and a minus sign is printed.

So adding an 'ELSE' term seems an obvious step.

```
      IF fabs ." –"
      ELSE space
      THEN
```

This should print a space before each positive number, but will require modified versions of `F.` and `FE.`
Try

```
      : fpos?  fdup 0.e0 f>= if space then ;          (  F:  r - -  )
```

In the listing, the word `fpos?` is tested by running `.Test2` and successfully prints both

```
        156768000.          and            156.768E6
       -156768000.                        -156.768E6
```

The Alignment Problem
Before starting on this step, just a note for newcomers to Win32Forth and to floating point. Win32Forth uses a separate stack to hold floating point numbers, so there are often two stack diagrams used with the defining words in the file float.f . Unfortunately there are several variations used for the floating point stack diagram, such as `( F:  r  -- r )`, `( FS: r  -- r )` and even `( f1 f2 -- f3 )` which is a little confusing at first. The word `F.S` shows the contents of the floating point stack.

The defining word `F.` is forty lines long, but in the second half, there are a number of `IF … ELSE … THEN` statements ending with variations of '`$ftemp precision type`' which hints that printing a floating point number is just a variation of printing a string, where '`$ftemp`' is an address and '`precision`' is a number as in the common '`addr count type`' statements.

If these words, or derivations of them, can be used to convert the floating point number into a suitable string, the problem is solved.

Before looking at the words in detail, it is necessary to decide on the form of the string. As an example, assume;

- The range of numbers required, say `+- 999999999`, so
  - 9 places are needed to the left of the decimal point for number characters, plus
  - 1 place for the minus sign or space, plus say
  - 3 places at the left to separate the number from any text, but which could be used for digits if necessary.

  That's 13 places to the left of the decimal point.

- The number of decimal places required, say 4, but try and make this variable in the final defining word; so that makes a string length of about 18 characters, not forgetting the decimal point itself; something like '`___-nnnnnnnnn.dddd`' should suffice.

The integer part '`___-nnnnnnnnn`' is 13 characters in length.
The 'sign' uses one character (space or -), and comes from either `F.` or `fpos?`
The remaining twelve characters may be either digits or spaces, the number of digits can found from the defined word `Represent` in `float.f`; so by defining a constant `int$len = 12` and subtracting the result of `Represent` gives the number of blanking spaces needed.
This step needs watching, because `Represent` can return a negative value, so the number of spaces must be held between 0 and 12.

The decimal part '`.dddd`' is variable in length. The number of decimal places is required to be on the data stack at execution.

The length of the absolute numerical part '`nnnnnnnnn.dddd`' is variable, but if the output is to be correctly right-aligned, all the characters is this part must be filled with digits. This can be done by setting the precision of the floating point number to this length.

There is a trap for the unwary here, (guess who fell for it!). It is most likely that the new word will be used during long sessions of floating point work, the new word will continually change the precision of the floating point, so it is necessary to store the current value of the precision before printing and to restore it afterwards.

Most of the words we need from the definition of `F.` such as '`Precision`', '`Set-precision`' and the 128 bytes reserved by '`create $ftemp`' are straightforward enough, but '`Represent`' is a key word and is worth investigation.

Represent

`Represent` expects to find an address and an unsigned number on the data stack, and a floating number on the floating number stack and returns two flags and a number.

```
REPRESENT   ( addr u -- n flag1 flag2 ) ( f: r -- )
```

As examples;

```
      1.45678e2 $ftemp precision represent  returns  3 0 -1
      14567.8e4 $ftemp precision represent  returns 9 0 –1
      14567.8e-6 $ftemp precision represent  returns -1 0 –1
```

In each case the sum of the power and the number of places to the left of the decimal point is equal to number returned, using the three examples above gives; `1 + 2 = 3, 5 + 4 = 9` and `5 + (-6) = -1`, which is the number of places to the left of the decimal point if the power were 0. I'm sure there is a correct mathematical term for this, but my school days are long gone. However, the sum of the length of the number string to the left of the decimal point and the number of decimal places gives the precision required.

Now, put all the parts together. Initiate a variable old-precision and a constant `int$len`.
Define a word `fpos?`.
Words used from the file float.f .

`f., $ftemp, precision, set-precision, represent` and `fdup`.

Define a word `F.P`
`F.P` prints a floating point number between `+- int$len` with a variable number of decimal places.

```
: F.P (  n -- ) ( f: r -- )    \ requires n, the number of decimal places
                               \ and r,a floating point number.
  precision old-precision !    \ store current value of precision.
  >r fdup                      \ put the number of decimal places
                               \ required ( d ) on the return stack,
                               \ and duplicate the floating point number.
  Int$len $ftemp precision represent drop drop
                               \ int$len and the number ( n ) returned by
                               \ represent on the data stack.
                               \ floating point number on the float stack.
  dup r> dup                   \ int$len, n, n, d, d on the data stack.
  rot +                        \ int$len, n, d, and ( n + d ) on the
                               \ data stack.
  max set-precision            \ set the precision to maximum
                               \ of ( d and ( n + d ))
                               \ int$len, and n on the data stack.
  0 max -                      \ int$len – ( max ( n and 0 )) on
                               \ the data stack,
                               \ this is the number of leading spaces to print.
  spaces                       \ print the leading spaces. Data stack empty.
  fpos? f.                     \ print either a space or -, then print the
```

```
                                       \ fp number.
                                       \ floating point stack empty.
  old-precision @ precision !          \ restore the original precision.
;
```

This new word `F.P` can be tested by running `.Test3`.

There is nothing specifically Win32Forth-ish about the above. It should run on other Forths with little or no revision, all the floating point words are in DPANS94 ANSI standard A.12 The optional Floating-Point word set.

`F.P` now goes into my folder of utilities, and an watchful eye will be cast over the newsgroups for the better ways of doing the same job; there must be plenty of these out there, somewhere.

nac@forth.org

The Canon Cat

Neal Crook

Neal Crook has investigated the success and untimely demise of this radical post-Macintosh computer powered by Forth. Its innovative user interface reminds us that computers can do much more for their users than the mainstream GUIs currently achieve.

Let me introduce you to the Cat. Here are some of its features:

- **document-centric operation**
- **built-in modem and communications software**
- **OS and graphics toolkit in ROM**
- **open architecture**
- **designed to allow easy integration of 3rd-party software**
- **on-line documentation**
- **automatic resume at power-on**
- **screen saver**
- **instant-on with any keystroke**
- **instant-boot**

That sounds like a feature-list for a rather nice modern PDA, but in fact it is the description of a desktop machine that came to market more than 13 years ago. It was the brainchild of Jef Raskin, a user-interface expert and father of the first Apple Macintosh. To whet your appetite further, the 68000-based machine had bit-mapped graphics and an integral Forth interpreter.

In 1978, Apple recruited Jef Raskin, employee number 31, as Manager of Publications. Raskin had been a writer on Dr Dobb's Journal, but his user-interface credibility dates back to his 1967 Computer Science thesis,

> *"A Hardware-Independent Computer Drawing System Using List-Structure Modelling: The Quick-Draw Graphics System"*

which advocated a graphical what-you-see-is-what-you-get user interface.

In 1970, Xerox had opened its Palo Alto Research Center (PARC) and by 1974 had developed a revolutionary machine with a mouse-like pointer device and a graphical user interface, called the Alto. Raskin was aware of this work and was instrumental in arranging a demonstration for Steve Jobs. Jobs became convinced that the GUI was the way forward for computers, and redirected the development of the Apple Lisa to use some of the technologies that he had seen at PARC.

Months before this, Raskin had persuaded Apple to start a project to develop a machine that he named Macintosh. This was envisaged as a smaller, simpler, cheaper machine than Lisa, with an emphasis on ease of use. However, when Jobs was pulled off the Lisa project, he responded by taking control of the Macintosh development, and putting Raskin in charge of its documentation.

By the time the Macintosh shipped in 1984, Raskin seemed to have been written out of Macintosh history. The two in-depth Macintosh articles in the February 1984 issue of Byte magazine make no mention of him (though this was later remedied in the August 1984 issue). Xerox brought their own Alto-derived machine, the Star, to market in the same year, but with a price tag of $16,595 it sold poorly (for comparison, the Macintosh was priced at between $1,995 and $2,495).

Amongst the Raskin legacy to the Macintosh are the click-and-drag user interface and the one-button mouse (the Xerox machines used a three-button pointing device, but Raskin wrote a memo arguing that a user often pressed the wrong button and showing how a single button could be implemented needing fewer or the same number of user actions).

In 1982, Raskin left Apple to form Information Appliance Inc, where he was chairman and CEO. His new company developed Swyft, a document-centric machine that focussed on ease of use. In 1985, Apple brought the SwyftCard to market as an add-in card for the Apple][. Information Appliance produced a more highly integrated Swyft derivative for Canon, called the Cat. They even developed a prototype FlatCat, though this machine never came to market.

Canon's typewriter division launched the Canon Cat Workprocessor in 1987, at a price of $1,495. It was bundled with a daisywheel printer and was programmed entirely in Forth[1] to be a dedicated document processor.

Like the Macintosh, it integrated the system unit and display into a single plastic case but added an integral keyboard.

Internally the Cat was based on the 68000 (like the Macintosh) with 256 Kbytes of RAM and a 720 Kbyte floppy. It had bit-mapped graphics and support for a pointing device such as a

Canon Cat Workprocessor

[1] As designer Jef Raskin confirmed to Forthwrite, " Everything in the Cat was written in Forth".

mouse, but the typewriter-centric marketing meant that Canon never exploited these capabilities. Indeed Raskin claimed that the product marketed by Canon was only a "dim echo" of what his company had designed.

Instant On
The Cat didn't use files, instead it saved its entire machine state (including the state of the screen and the cursor position) to floppy disk. At power-up, it gave the illusion of booting almost instantly; within one second of power-up, the screen display (including the cursor position) was restored from the floppy disk. After a further 6 seconds, the cursor would start to flash and the machine was fully ready for use. If the user typed anything in those 6 seconds, the keystrokes were buffered and displayed as soon as the machine was fully operational. Tests showed that most people spent more than 6 seconds looking at the screen to restore their own context before starting to type.

The Keyboard
The Cat keyboard was small, and had a shift key called USE FRONT to access special functions that were marked on the front of some of the keys.

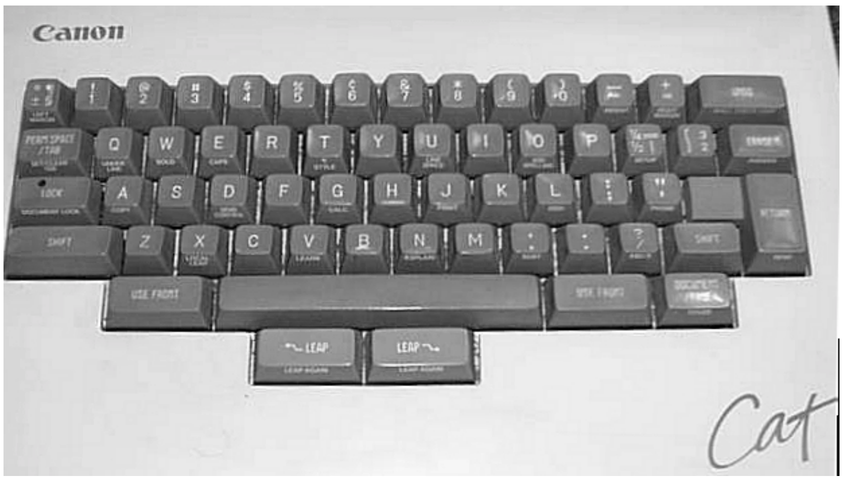
For example:

- **Page** – to start new pages; with a USE FRONT shift, this would start a new document.
- **Learn** – to record a keystroke sequence and assign it to the front of a number key.
- **Explain** – to get on-line help (and "Explain" followed by another key to get help on that key's function).
- **Answer** – to replace a highlighted formula with its calculated result. The formula beneath was retained, and could be named and cross-referenced. This allowed spreadsheets to be created and embedded within documents.

Internationalisation
A set-up screen allowed the keyboard layout to be changed to support different languages.

Navigation
There were no cursor keys. Navigation and selection were achieved using two coloured **LEAP** keys that were positioned for use by the thumbs; in the dead centre front of the keyboard (see below), just in front of the spacebar and rather like the mouse buttons on some modern laptops.

When the **LEAP** keys were simply pressed and released, they acted as cursor-left and cursor-right. Used as shift keys for alpha-numeric characters they enabled incremental search: holding **LEAP>>** and pressing 'a', moved the cursor forward to the next instance of that character. Keeping **LEAP>>** held and pressing 'b' moved the cursor forward to the next instance of the string 'ab'.



Canon Cat keyboard

Anyone who has used the Emacs editor will be familiar with this search mechanism. Using "shift front" with a **LEAP** key repeated the last leap. Using a **LEAP** key as a shift key with, for example, the **Page** key, would page forward or backwards within the document. After **LEAP**ing from one end of a text region to another, pressing both **LEAP** keys simultaneously would highlight the text within the region.

In other writings, Raskin explains how this navigation method (for which he was granted US Patent 5,019,806) can significantly improve ease of use in a number ways:

> Firstly, and most obviously, it requires the user to type the minimum number of letters needed to locate the desired text.
>
> Secondly, it avoids the situation where you mistype the word you're searching for and then have to wait while the computer fails to find it.
>
> Thirdly, it can improve the response time of the computer; as soon as the computer has updated the screen to show (for example) the location of the string 'ab', it can proceed to build an internal data-structure showing the first occurrence of 'aba', 'abb', 'abc' and so forth (it would order the search based on the frequency of letter usage in the English language). By the time the user types the next character, the computer is likely to have already located the next occurrence of it, and simply needs to update the screen. An optimisation like this is particularly useful when your processor is slow or your document large.

Forth In ROM

By highlighting the text "Enable FORTH Language" and pressing "Answer", the function of the "Answer" key was changed so that it would evaluate highlighted text as Forth code. In addition, by entering the deliberately cryptic sequence "use front" + "shift" + "space", the machine entered an interactive Forth editor/interpreter.

Once enabled, you could enter Forth text, highlight it and press "Answer" to evaluate it. By pressing another sequence of keys, you could also get to the interpreter itself.

Once a Forth word had been defined and tested, the user could execute it merely by selecting the name on the screen and pressing the Answer key. (Some years later Niklaus Wirth included similar functionality in his Oberon system.)

Outcome

The Canon Cat was well-received, and won a number of awards, including the 1989 Industrial Designer's Society of America award. Surprisingly after 6 months, with 20,000 units sold, Canon withdrew the Cat from sale. I found two suggestions on the Internet for Canon's motivation.

- Political in-fighting between the typewriter and computer divisions within Canon.
- Pressure from Steve Jobs's new company NeXT, in which Canon became a 16.67% investor in June 1989.

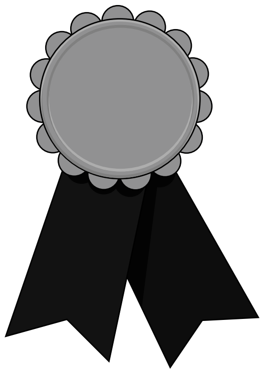
Recently, a group of the original Macintosh designers were in the news with the creation of their new company, Eazel. Eazel is developing user-interface software that will form part of the Gnome free software desktop environment.

Today, Raskin is an interface and system design consultant - see his new book "The Humane Interface". Commenting (in 1996) on some articles about Macintosh history, Raskin wrote: *"What I want to create is software that is as easy to use as the Cat was but with the power of today's applications. I know how to do it, I just haven't found a company where I can build it"*. So there's hope for more new and exciting innovations from this man.

Acknowledgements

My thanks to Al Kossow for providing pictures of the Swyft from his web site, and to George Currie who photographed his Cat for especially for Forthwrite.

Neal Crook (nac@forth.org) is a hardware engineer by trade, with a background in ASIC specification, design and verification. He first became seriously interested in Forth for use as a hardware debug tool whilst developing a validation board for the SA-110 StrongARM microprocessor.



Nominations for the FIG UK Awards of 1999

The FIG UK Awards of 1998 were won by Philip Preston and Paul Bennett. These awards are given to encourage effort and recognise achievement.

Free membership

To everyone who sent in their nominations - "thank you". The closing date for these has now passed and it's over to the judges to announce the winner in the next issue. The recipient of each award will receive a place in FIG UK web-site's Hall Of Fame, a mention in Forthwrite and ***a year's free membership***.

Achievement

Jeremy Fowell: for his efforts in leading the FIG UK Hardware Project.

Forthwrite

Alan Wenham: for his work in making German FIG accessible to all.
Dave Pochin: for his sharing his explorations with Win32Forth.

The awards are judged by the officers of FIG UK. All who are members on 31st Dec. 1999 are eligible (except the judges).

John Tasgal
0161 7739365
john@tcl.prestel.co.uk

An Introduction to Color Forth

John Tasgal

Following this Color Forth article is a commentary on some of Chuck's published code, showing how complex code can be written with a simpler Forth. John's introduction to Color Forth is necessarily incomplete as a definitive and comprehensive description will require Chuck's assistance.

Color Forth (CF) is an extremely original and interesting attempt to simplify both the structure and appearance of Forth. It inherits several features of Machine Forth, including the use of address registers. But Charles Moore has reverted in this, his latest Forth, to the destructive conditionals of Classical Forth.

Its two principal innovations are the use of colour to signify syntactic or semantic categories; and the simplification and reduction in the number of control structures.

The original source code was first shown on a monitor using a black background with coloured text. For obvious reasons of legibility I have changed the colour of the execution-mode tokens from white to black. There is also a special space character, a green space, which is shown here as a green underscore after the token
i.e. `'token_'`

The effect is to compile a literal: pop the top of stack; compile it's value; at run-time that value is pushed.

Note that in Color Forth there are no lines of source text: the code is interpreted token by token.

This article and its successor (the BMP example) are intended to be read alongside Charles Moore's description of Color Forth as given in the three references at the end.

Notation and Glossary

Ordinary text	Explanatory text - not source code
`Source text`	Source code (in a variety of colours)

Source Code Colour Key:

`( ) Text`	*Comments in blue*
`WORD`	Interpret mode in red (viz define this token as a new name in the dictionary)
`SWAP`	Compilation mode in green
`BUF`	Execution mode in black (white in the original video source)
`999`	Decimal numbers in grey
`FFFF`	*Hexadecimals in cyan*
`_`	Compile the number on the TOS (A green underscore which has the same meaning as a green space in the (video) source code)

Notation

`flag?`	A word or words which push a boolean value for use by IF.
`w0 w1 ..`	In the examples below these are assumed to be pre-defined application words.

Basic Constructs

Here is a list of elementary program structures. Each program or fragment is first shown on a single line, then explained in detail one token or one expression to the line.

```
1. WORD1 w0 w1 w2 ;  WORD1
Create a word and execute it.
```

The simplest CF program. Build a subroutine called `WORD1` then execute it.

Explanation:

`1.`	A comment
`WORD1`	Create `WORD1`
`w0 w1 w2`	Compile `w0`, `w1` and `w2`
`;`	Not compiled
`WORD1`	Executing `WORD1` causes `w0`, `w1` and `w2` to be executed.

Note that because of tail-recursion optimisation (see previous article) the `;` *is not compiled.*

2. **WORD1** `w0 w1 w2 word1 ;` WORD1
An infinite loop

Explanation:

WORD1	Create WORD1
w0 w1 w2	Compile w0, w1 and w2
word1	Compile a jump to w0
;	Not compiled
WORD1	Executing WORD1 causes w0, w1 and w2 to be repeatedly executed.

3. **WORD1** `flag? IF w0 w1 w2 ; THEN w3 w4 w5 ;` WORD1
A Two-Branched Conditional

Explanation:

WORD1	Create word WORD1
flag?	Compile flag? (which modifies the flag for use by IF)
IF	Compile the run time behaviour for IF
w0 w1 w2	Compile these words. Will be executed when flag? is true
;	Not compiled
THEN	
w3 w4 w5	Compile these words. Will be executed when flag? is false
;	
WORD1	If flag? is true execute w0, w1 and w2, then return. Else execute w3, w4 and w5 and return.

Note: For a single-branched test, just remove w3 w4 w5.

Note: Whereas with Machine Forth, the flag? was preserved, in Color Forth, Chuck has reverted to the classical IF which consumes the flag?.

4. **WORD1** `flag? IF w0 w1 w2 WORD1 ; THEN w3 w4 w5 ;` WORD1
A While-True loop

Explanation:

WORD1	Create word WORD1
flag?	Compile flag?
	Compile the run time behaviour for IF
w0 w1 w2	Compile these words. Will be executed when flag? is true
WORD1	Compile a jump to flag?
;	Not compiled
THEN	

`w3 w4 w5`	Compile these words. Will be executed when `flag?` is false
`;`	Not compiled
`WORD1`	While `flag?` is true, execute `w0`, `w1` and `w2` then repeat. Else execute `w3`, `w4` and `w5` and return.

`5. WORD1 flag? IF w0 w1 w2 ; THEN w3 w4 w5 WORD1 ; WORD1`
`A While-False loop`

Explanation:

`WORD1`	Create word `WORD1`
`flag?`	Compile `flag?`
`IF`	Compile the run time behaviour for `IF`
`w0 w1 w2`	Compile these words. Will be executed when `flag?` is true
`;`	Not compiled
`THEN`	
`w3 w4 w5`	Compile these words. Will be executed when `flag?` is false
`WORD1`	Compile a jump to `flag?`
`;`	Not compiled
`WORD1`	While `flag?` is false, execute `w3`, `w4` and `w5` and repeat. Else execute `w0`, `w1` and `w2` and return.

`6. WORD1 w1 WORD2 w2 WORD3 w3 ; WORD1 WORD2 WORD3`
`Multiple entry points`

Explanation:

This is a feature not supported by ANS Forth.

`WORD1`	Create `WORD1`
`w1`	Compile `w1`
`WORD2`	Create `WORD2`
`w2`	Compile `w2`
`WORD3`	Create `WORD3`
`w3`	Compile `w3`
`;`	Not compiled
`WORD1`	Execute `w1`, `w2` and `w3` then return
`WORD2`	Execute `w2` and `w3` then return
`WORD3`	Execute `w3` then return

`7. 255 FE +`
`Evaluate an expression containing literals`

Explanation:

`255 ( -- 255 )`	Push decimal `255`
`FE  ( -- 255 254 )`	Push hex `FF`

```
+    ( -- 509 )
```
Add them

```
8. WORD1 255_FE_+ ; WORD1
Compile an expression containing literals
```

Explanation:

[Note: The next stack pictures show the stack during *compilation*]

`WORD1`		Compile `Word1`
`255`	`( -- 255 )`	Push decimal `255`
`_`	`( -- )`	Compile it
`FE`	`( -- 254 )`	Push hex `FE`
`_`	`(  --  )`	Compile it
`+`		Compile add
`;`		
`WORD1`		Execute `Word1` with the following run-time behaviour:
`( -- 255 )`		`255` pushed to stack
`( -- 255 254 )`		Hex `FE` pushed
`( -- 509 )`		Add

```
9. 10_BEGIN w0 w1 w2 NEXT
Set up the index for a loop
```

Explanation:

This example pushes decimal `10` onto the stack at run-time, where it will be used as the index for the loop. [This is not as inconvenient as it seems because the A register is available for holding an intermediate value - Ed.]

`w0`, `w1` and `w2` are executed 10 times.

```
10. VARIABLE w0 w1 w2
Declare 3 variables called w1, w2 and w3
```

Some Idioms

Here are some frequently-used instruction sequences:

```
1. BUF_A!
An efficient way to access a variable's address
```

Note: The variable's name is in black, followed by the green underscore.

Explanation:

```
BUF ( -- ^buf )     Push the address
_   ( -- )          Compile the literal
A!  ( -- )          Compile A!
```

At run-time, the address is pushed, then `A!` stores it into the accumulator `A`.

```
2.  A 20_+ A!   ( A = A + 20 )
    A -20_+ A!  ( A = A - 20 )
    Read, modify and write A
```

Explanation:

This is the sequence to modify `A` for pointer arithmetic where an increment-by-one isn't suitable. E.g. adding or subtracting the 'stride' of an array, or stepping through large fields in a record.
Note that there is no subtraction primitive. `'-'` must be defined as a high-level word.

```
3.  WORD1 @+ flag? IF w0 w1 w2 word1 ; THEN w3 w4 w5 ;  ...
    Process a stream using pointer arithmetic
```

Explanation:

While `flag?` is true execute `w0`, `w1` and `w2` `then repeat`; else execute `w3`, `w4` and `w5` then return.
In each loop, initially fetch the contents pointed to by `A`, and increment `A`.

No `flag?` word is needed if it is intended to exit on `A`=zero (as false causes a jump to the exit sequence). So as a special case we can write:

```
WORD1 @+ IF w0 w1 w2 word1 ; THEN w3 w4 w5 ;  ...
```
or
```
WORD2 @+ DUP IF w0 w1 w2 word1 ; THEN DROP w3 w4 w5 ;  ...
```

The `WORD2` version has a `DUP` to allow the data value fetched to be used by the true block.
This is a fast and elegant way of scanning an array with an exit-on-zero.

```
4. WORD1 20_BEGIN @+ w0 w1 w2 NEXT ...
Process a stream using an index
```

Explanation:

Set the index to 20. In each loop, fetch the contents of `A`; increment `A`; process `w0`, `w1` and `w2` then decrement the index and repeat if not zero. `WHILE` loops are the method of choice as they don't require an index.

The next article is my annotation to Charles Moores's Color Forth program **BMP**, which converts a VGA screen to a BMP file.

References

1. Color Forth
   http://www.UltraTechnology.com/color4th.html
2. 1X Forth
   http://www.UltraTechnology.com/1xforth.htm
3. Dispelling the User Illusion
   http://www.UltraTechnology.com/cm52299.htm
   This includes the source code for the BMP example.

Postscript

In a recent message to the newsgroup (13th May) Jeff Fox reported that Chuck plans to publish Color Forth for PCs with Pentium processors. Some progress has been made in this but no release date could be given.

Jobs Roundup

News of job and project opportunities are sent by e-mail to all members that have expressed an interest. If you didn't receive the message sent on 23rd May, then please contact Chris Jakeman to add your name to the list.

From Paul Bennett:

Not just yet but most probably some time soon, one of my clients will have a need for a permanent software engineer with a knowledge of Forth for a variety of Forth dialects (Fig to ANS). The position will be in the West Midlands region of the UK and will involve some site visits.

Duties will be mainly the maintenance of existing code (assembler and Forth, (maybe a bit of C as well) and assistance in developing new products. Experience with real-time machine control would be most useful as would any electronics, electrical and mechanical background.

In the first instance, e-mails to me with a brief (one page) resume which I will pass on to my client for his consideration in due course.

Thank you

```
Paul E. Bennett ..................<email://peb@amleth.demon.co.uk>
Forth based HIDECS Consultancy ....<http://www.amleth.demon.co.uk/>
Tel: +44 (0)7971-620145    /      NOW AVAILABLE:- HIDECS COURSE
Going Forth Safely        / see http://www.feabhas.com for details
```

From Steve Smith:
I should have replied to this a long time ago. After just missing out on a recent Forth-based contract and thinking about how much I would like to get back into any Forth projects, can you put me on your contact list for any Forth-based employment/contract/project work that comes your way.

I do my best to keep my Forth skills up to date (I've got a copy of SwiftForth), but have to admit it's been 9 years since I was actively involved in any commercial Forth work.

Up to 1991 I was Software Manager for a scientific instrumentation company and we used polyForth and chipForth extensively for both the embedded systems for instrument control, and the PC-based data processing systems. My CVs in various formats are currently available on my web site at:

www.diamondb.demon.co.uk/cv.html

John Tasgal
0161 7739365
john@tcl.prestel.co.uk

The BMP Example

John Tasgal

Charles Moore has provided an example of Color Forth[2] in use, which describes the conversion of a video screen to a BMP file. Before looking at the code, here is some background information.

The Program

The aim is to format a video buffer and write it to another area of memory, the BMP buffer.

When this has been done, Color Forth is exited and, having recorded the start and length of that buffer, the memory is saved to disk using DOS. The screen is in VGA mode with a resolution of 640 columns and 480 rows. Each pixel is represented as a single byte, giving 256 colours.

The original implementation of Color Forth uses a 20-bit cell on the i21 processor. This code is for a PC implementation using a 32-bit cell.

BMP Format

A BMP (Window's Bit Map) file has three parts - a header, a palette, and the video data itself, as shown here.

Offset	**Contents**
0000h	Bitmap type ("BM" for Windows)
0002h	File size in bytes.
0006h	Reserved
000Ah	Bitmap Data Offset from beginning of file to the beginning of the bitmap data.
000Eh	Length of the Bitmap Info Header used to describe the bitmap colours etc (= 28h for Windows)
0012h	Horizontal width of bitmap in pixels.
0016h	Vertical height of bitmap in pixels.
001Ah	Number of planes in this bitmap.

[2] Dispelling the User Illusion
http://www.UltraTechnology.com/cm52299.htm

`001Ch`	Bits Per Pixel
`001Eh`	Compression Type. 0 = none; 1 = RLE8; 2 RLE4; 3 = Bitfields
`0022h`	Size of bitmap data in bytes, rounded up to 4 byte boundary.
`0026h`	Horizontal resolution in pixels/m.
`002Ah`	Vertical resolution in pixels/m.
`002Eh`	Number of colors used by this bitmap. For a 8bit/pixel bitmap this will be 256.
`0032h`	Number of important colors
`0036h`	The Palette of size = (#colours* 4) bytes. Each entry has 4 bytes: blue, green, red, filler. The filler is set to zero.
`0436h`	Bitmap Data

The Algorithm

To minimise the amount of data to be stored, the extent of the image must be established.

First, the frame surrounding the image is filled with zeroes. Then, the rectangle defining the outer limits of the image is found by scanning the whole picture in four directions:

- top down to find the top edge;
- bottom up for the lower edge;
- left to right for the left edge;
- and right to left for the right edge.

This yields:

BUF	a pointer variable to hold an address
H	the height, and
W	the width

That buffer is now formatted and written to the BMP buffer. Please refer to the 'User Illusion' text to see Charles Moore's description of this program.

Glossary

Variables:

BUF	the address of the top left corner (and therefore the start of the image array in the video buffer)
H	the height, and
W	the width

Procedures:

```
ROW  ( stride ^row -- true | stride false )
```

	Scan a row beginning at `^row`. Returns `0` for a blank line, non-zero otherwise.
`ROWS` `( stride ^row -- )`	Scan rows to find first non-zero row. Store value in `H`.
`COL` `( stride ^col -- true | stride false )`	Is this a blank column ?
`COLS` `( stride ^col -- stride )`	Scan columns looking for first non-blank. Return width in `W`, and set `BUF` to first column.
`N,` `( u advance -- )`	Write a value to the location pointed to by `BUF` (the BMP buffer pointer).
`2,` `( u -- )`	Write a value to the BMP buffer, incrementing the `BUF` pointer by 2 bytes.
`PACK` `( pel2 -- packpel2 )` `( xbxa -- xxab )`	Pack and invert pixels where `a` and `b` are nibble-sized pixels.
`ROW` `( )`	Write a packed row to the BMP buffer
`ROWS` `( ^buf -- )`	Write the video buffer to the data part of the BMP buffer

The Source Code[3]

Chuck uses a format of blocks organised in 12 lines of 20 characters. The code which follows uses a more conventional format.

`FRAME`	A comment
`EMPTY ( -- )`	Minimise the dictionary
`Declare variables`	
`VARIABLE` **`BUF W H`**	Create buffer, width and height variables

Next define `ROW`, `ROWS`, `COL` and `COLS`. First is `ROW` :

1. set up the loop count
2. enter the begin ... next loop
3. fetch 4 pixels and increment A
4. if they're all blank then continue round the loop
5. else exit

[3]This is a fairly complete annotation but some parts of the code (marked with "??") defied analysis. This may be partly due to errors in the published HTML page which contains a few copying errors. Both John and I have spent some time trying (and failing) to decode the exact stack behaviour. If any reader can solve the puzzle, please write in. - Ed.

```
ROW     ( stride ^row -- true OR stride false )
                                        Scan a row beginning at ^row
                                        Return 0 for a blank line, non-zero otherwise
  A!      ( -- stride )                 Read row address into A
  159_    ( -- stride i )               Push limit for loop to do 160 fetches of 4 bytes to
                                        scan all 640 pixels across image.
                                        [Original reads 169, not 159 - Ed.]
  BEGIN   ( -- stride i )
    @+    ( -- stride i pel4 )          Fetch pel4, increment A.  pel4 is shorthand for 4
                                        byte-sized pixels packed into one 32-bit word.
    IF    ( -- stride i )               Test for a zero value indicating 4 blank pixels.
      +   ( -- stride+i )               If not found, leave a non-zero value on the stack
      ;   ( -- non-zero )               and return.
    THEN  ( -- stride i )               If all pixels are blank, continue.
    DROP                                [?? Surely a mistake - Ed.]
  NEXT    ( -- stride i-1 )             Decrement loop count
  0_      ( -- stride 0 )               If it gets to here, the row is all zeroes,
;                                       so push a zero and return
```

ROWS is a while-false loop .

1. while ROW returns false (a blank line)
2. decrement H

```
ROWS ( stride ^row -- )
Scans across to find first non-zero row.
Stores value in H.
  DUP BUF_ !    ( -- stride ^row )        Store current row in BUF
  ROW           ( -- true OR stride false ) Is this row non-blank ?
  IF            ( -- )                    If a non-blank line
    DROP DROP                             [?? Surely a mistake - Ed.]
    ;           ( -- )                    return (with BUF set to current row)
  THEN          ( -- stride )             If all pixels are blank, continue.
  DROP                                    [?? Surely a mistake - Ed.]
  -1_H_+!       ( -- stride )             Decrement H
  DUP A +       ( -- stride ^row )        Point to next row[4]
  ROWS                                    Jump back to first DUP (i.e. scan next row)
;
```

Identify the upper and lower edges by calling ROWS twice; first top down, then bottom up.

[4] This would work if A always held the row address at this point.

```
448 H !      Set H to max number of rows. As 480-32 =448, presumably these 32
             rows are for the command line.

VGA 0        ( -- ^vga  0 )
OVER ROWS    ( -- ^vga ) Find first edge by scanning up.
-1280 SWAP   ( -- -1280 ^vga )
640 447 *          This product is the max number of pixels in the image
+            ( -- -1280 ^vga+[640*447]  )
ROWS         ( -- )        Find second edge by scanning down.[5]

COL     ( ^col ^newcol -- true OR ^col false )
Do all rows in this column contain blank pixels?
  A!         ( -- ^col )             Make A point to the start of a column.
  H @ -1 +_  ( -- ^col #rows-1 )     Loop limit is number of rows-1 to scan.
  BEGIN      ( -- ^col i )
    @+       ( -- ^col i pel4 )      Fetch pel4 and increment A by 4
    FF_AND   ( -- ^col i pel )       Extract single pixel by clearing all but the
                                     lower 8 bits.
    IF       ( -- ^col i )           If pixel is not blank,
      +      ( -- non-zero )         leave a non-zero result
      ;      ( -- true )             and return.
    THEN     ( -- ^col i )           Else advance to the next row and continue
    DROP                             [?? Surely a mistake - Ed.]
    A -644_ + A! ( -- ^col i )       Point A to next row -640 - 4. -4 is
                                     needed as @+ incremented A by 4
                                     (Original reads 544, not 644 - Ed.)
    NEXT     ( -- ^col i-1 )         Next row
  0_         ( -- ^col 0 )           If all rows are blank, push 0
;

COLS ( ^col -- )
Scan columns looking for first non-blank one.
Return width in W, and set BUF to first column
  DUP BUF_!     ( -- ^col )          Point BUF to start of column
  COL           ( -- true OR ^col false ) Is this column blank?
  IF                                 If non-blank column,
    DROP
```

[5] We've now found H, the number of rows in the image, but we seem to have discarded the address where the first row of the image starts.

```
    ;          ( -- )              then return
  THEN         ( -- ^col )         Else continue
  DROP                             [?? Surely a mistake - Ed.]

  -1 W +!   ( -- stride )       Decrement W
  DUP A +   ( -- ^nextcol )     Point to start of next column
  COLS                          Jump back to first DUP i.e. scan next column
 ;
```

Identify the left and right edges by calling `COLS` twice: first left-to-right, then right-to-left

```
640 W !      ( -- )          Set W to max # of cols
BUF @ H @    ( -- BUF H )
640 * -1 +   ( -- BUF #pix )    #pix = (H*640)-1, i.e. no. of pixels
                                containing the image after trimming blank rows
                                from top and bottom.
OVER 639 +   ( -- BUF #pix buf' )    buf' = buf + 639
COLS         ( -- BUF #pix )    Scan left-to-right
2 + SWAP     ( -- #pix+2 BUF )
COLS         ( -- #pix+2 )      Scan right-to-left
DROP         ( -- )
W @ 1 + -2 AND ( -- W' )        Rounds W up to nearest even number so that
                                2-byte reads and writes can be used.
W !          ( -- )             Store result in W
```

`BUF`, `H` and `W` now have their final values and we prepare to write to the buffer `BMP`.

```
BUF @              ( -- bufold )    Save old value of BUF
B71000 BUF !       ( -- bufold )    Change BUF
.                  ( -- )
4                  ( -- 4 )

N, ( n Advance -- )      Store a word of data at the location
                         pointed to by BUF. Advance the pointer BUF by
                         Advance bytes.
  SWAP      ( -- n2 Offset n1 )
  BUF @ !                Store the first word
  BUF +!                 Advance the pointer
                         [Original reads +1, not +! - Ed.]
;
```

```
2,    ( n -- )         Write a 16-bit word to the BUF buffer
  2_  ( -- n 2 )       2 = bytes to advance
  N,  ( -- )
;
```

Now writing to the `BMP` buffer can begin. First, the header:

```
BM                                [Is this a misprint? - Ed.]
4D42 2,                           Store ASCII "BM" at offset 0000h
W @ H @      ( -- W H )
2 */         ( -- (W*H)/2 )       The bitmap size in bytes ..
16 4 * +     ( -- size1 )         Add 64 for the palette ..
54 +         ( -- size2 )         Add 54 for the header ..
,                                 and store at offset 0002h.
0 ,          0006h Reserved
118 ,        000Ah Bitmap Data Offset
40 ,         000Eh Length of the Bitmap Info Header
W @ ,        0012h Width
H @ ,        0016h Height
1 2,         001Ah Number of planes in this bitmap
4 2,         001Ch Bits Per Pixel
0 ,          001Eh Compression Type. 0 = none
W @ H @
2 */ ,       0022h  Size of bitmap data in bytes, rounded up to 4 byte boundary
0 ,          0026h  Horizontal resolution
0 ,          002Ah Vertical resolution
0 ,          002Eh  Number of colors used by this bitmap. For a 4bit/pixel bitmap
                    this should be 16 (?)
0 ,          0032h  Number of important colors
(?)          0036h  The Palette of size = (#colours* 4) bytes
ORGB         Probably moves pointer to the palette origin.
```

Next, write the Palette data. The fourth byte is the filler and is always zero. So only up to 3 bytes are ever specified.

```
  FFFFFF ,    FF00 ,      FF ,   FFFFF ,
  E00000 ,  E0C000 ,  FFFF00 ,  808000 ,
  408080 ,  40F040 ,    40FC ,  E000C0 ,
  E00040 ,  C0FFFF ,  404040 ,  FCFCFC ,
```

Finally, we have the video data section. Although the code has been written for 256 colours in memory, only 16 are used at present so the video data is packed before writing to the file. `PACK` takes two pixels of one byte each and packs them in inverted order into one byte. For the algorithm below to work it assumes that the data is in two bytes with the pixel data in each of the lower nibbles; the data in the upper nibbles may be discarded.

i.e. `PACK ( xbxa -- xxab )`

Where `x` means don't care. The aim is to shift and swap, then do this `OR` as in:

```
xxa0
xx0b
xxab   after OR'ing
```

The data in the upper byte is ignored as, although the data is sent out a word at a time, it is overwritten by the next low byte to be sent. '*/' is shown black in the text - changed here to green.

```
PACK ( pel2 -- packpel2 ) Pack two pixels
    ( 0b0a -- xxab )      a and b are nibble-sized pixels
  DUP         ( -- 0b0a  0b0a )
  1_256_*/    ( -- 0b0a  000b )  shift n0 right 8 bits
  SWAP        ( -- 000b  0b0a )
  16_*        ( -- 000b  b0a0 )  shift n1 left 4 bits
  OR          ( -- xxab )        packed and inverted
;
```

`BMP` is written from bottom to top
Note: This is the second definition of `ROW` and `ROWS`.

```
ROW  ( -- )    Write a packed row to the BMP buffer
  W @ 2/       ( -- W/2 )         No. of bytes to write
  -1 +_        ( -- W/2-1 )
  BEGIN        ( -- i )
    @+         ( -- i pel4 )      Fetch 4 pixels
    PACK       ( -- i packpel2 )  Pack 2 pixels into the lowest byte
    1_N,       ( -- i )           Write the packed byte to BMP buffer
    A -2_+ A!  ( -- i )           Decrement A for next 2 pixels
  NEXT         ( -- i-1 )
;
```

[It is interesting to see that the code for `ROW`, `PACK` and `N`, work together to read data 2 bytes at a time and write it out byte by byte independently of the cell size of Color Forth (which seems to be 4 bytes) - Ed.]

```
ROWS ( ^buf -- )  Write the video buffer to the data part of the BMP
                  buffer.
  A!         ( -- )       Set A to point to the start of the data
  H @ -1 +_  ( -- i=H-1 )       Loop limit, once for each row
  BEGIN
    ROW      ( -- i )           Write a row
    A        ( -- i A )
    W @ -639 +_                 W-639 is computed at compile time
                                (Original reads @ - 639, not @ -639 - Ed.)
    +        ( -- i A+(W-639) ) Address of next row
    A!
  NEXT       ( --i-1 )
;

ROWS ( -- )   Write the video data
```

Acknowledgements
I would like to thank Chris Jakeman for suggesting I write these articles.
Many thanks also to Jeff Fox for kindly reviewing an earlier draft and providing an excellent web site.

Editor's Note: We regret the problems found in decoding this example, which would surely be overcome with help from Chuck himself. We hope that they do not obscure the many techniques Chuck has introduced to simplify the compiler and make Forth more appropriate for his current work.

These include:

- Use of the special A register
- Optimisation using ";"
- Looping back to the start of a word by repeating its name
- Looping using `BEGIN NEXT`
- Colour syntax for brevity

Chris Jakeman
cjakeman@bigfoot.com

Did you Know? - Forth OS

While other parts of Forthwrite bring you all the news and the latest ideas and developments, the **Did You Know?** section highlights achievements in Forth, both recent and historical (taking care always to distinguish hearsay from attested fact).

In the days before micro-computers came already equipped with disk drives and an operating system, Forth provided a well-integrated and powerful environment combining the operating system, the applications and the programming language.

This arrangement survives commercially to this day.

"To be more specific, Greg Bailey at Athena Programming (ftp://ftp.minerva.com/pub/www/athena.htm) supports several OEMs with an aggregate of several hundred installations with native Forth OSes running on low-end PCs providing various application services and also providing extremely high-bandwidth, high security TCP/IP services.

These boxes run for years without rebooting, and performance exceeds any high-end servers on the market. "

Source - Elizabeth Rather, Forth Inc.

Thanks to this concept, Forth occupies a unique place in the history of the 8086 processor.

"One of the very first 8086s was delivered to FORTH Inc, and polyFORTH was running less than two weeks later." This was before DOS and CPM/86 existed.

Source - Andrew Haley

Alan J M Wenham
01932 786440
101745.3615@compuserve.com

Vierte Dimension 2/00

Alan Wenham

Alan provides a look at the latest issue of the German FIG magazine. To borrow a copy or to arrange for a translation of an individual article, please call Alan.

Editorial

Friederich Prinz

`Friederich.Prinz@t-online.de`

Includes a plea, which is familiar to all newsletter editors, for more letters and articles.

Polyalphabetic Encryption Cracker, Part 2

Hugh Aguilar

`haguilar@forth.org`

This is a translation by Fred Behringer (behringe@mathematik.tu-muenchen.de) of a paper by Hugh in Forth Dimensions,Volume 2,Nos 5 and 6, January/April 1999.

Other Groups

Fred Behringer

`behringe@mathematik.tu-muenchen.de`

Reviews by Fred of Forth Dimensions, January/April 1999; of Forthwrite 104 and 105; and of Figleaf 16, 17, and 18.

First experience of Windows 2000

Friederich Prinz

`Friederich.Prinz@t-online.de`

Fritz had difficulties with faulty drivers for the ZIP engine and with the "intelligent" set-up where the installation process did not warn him about totally unsuitable drivers. The result was that Fritz spent many extra hours in installing an "absolutely fool-proof" Microsoft product which left him in the lurch.

The Ciphered Cipher - or `X S 2XOR` - A Riddle

Fred Behringer

behringe@
mathematik.tu-
muenchen.de

A secret message is sent backwards and forwards without reciprocal knowledge of the key, and without exchanging keys between recipient and sender, until the recipient can read the content of the message in plain text. What principle does this involve?

Colour in Life

Martin Bitter

mbitter@
bigfoot.de

It is not at all easy to print to paper in colour under Win32Forth. Martin describes how to do it.

`DO .. LOOP` Code for Extension up to 32k Steps

Fred Behringer

behringe@
mathematik.tu-
muenchen.de

In Forth systems that originated with F83 one can construct DO loops in assembler. These 16bit systems use opcode 7x for this purpose but this permits jumps (backwards and forwards) of only 128 bytes. For 80386 processors and higher Fred shows how the jumps may be extended to 32k by using the `0F 8x` opcodes.

Use of 4 - A Riddle

Fred Behringer

behringe@
mathematik.tu-
muenchen.de

Write down the integers 1 to 50 by only using exactly 4 times the figure "4" in connection with the usual arithmetic operations.

Deutsche Forth-Gesellschaft

Would you like to brush up on your German and at the same time get first-hand information about the activities of fellow Forth-ers in Germany?

Become a member of the German Forth Society for 80 DM (£28) per year (32 DM (£11) for students and retirees). Read about programs, projects, vendors and our annual conventions in the quarterly issues of *Vierte Dimension*.

For more information, please contact the German Forth Society at the e-mail address SECRETARY@ADMIN.FORTH-EV.DE

or visit http://www.forth-ev.de

or write to
Forth-Gesellschaft e.V.
Postfach 161204
18025 Rostock
Germany
Tel.: 0381-4007872

Chris Jakeman
01733 753489
cjakeman@bigfoot.com

From the 'Net - Cube Roots

Chris Jakeman

In the last issue, I reported a series of messages on comp.lang.forth about calculating cube roots, including contributions from member Philip Preston.

The subject surfaced again as a discussion on functional programming and Philip posted a method for using binary search to calculate the inverse of any function that always increases. He illustrates it with a square root and a cube root. He also posed a question, which I don't think was ever answered so is repeated here.

```
From: Philip Preston   philip@preston20.freeserve.co.uk
Date: 30-Jan-2000
Subject: Re: Forth a functional programming language?
```

```
Anton Ertl wrote
> Jonah Thomas <jethomas@ix.netcom.com> writes:
>> anton@mips.complang.tuwien.ac.at (Anton Ertl) wrote:
[snip]
>> This stuff is kind of fun.  I wonder whether it's good for
>> something?
>
> Ithink it's quite good at factoring control flow.  I think we
> recently had a try at factoring the binary-searching square-root
> into a higher-order binary-search word and a few other words.

That was the first thing I thought of. I came up with this:

: (BINSEARCH) ( upperlimit key xt -- result )
    2>R 0 SWAP BEGIN          ( -- low high )
        2DUP 1- < WHILE       \ end if high-low < 2 )
            2DUP + 1 RSHIFT   \ trial value is average of limits
            2R@               ( -- low high trial key xt )
            EXECUTE IF        ( -- low high trial )
                ROT ROT       \ Prepare to drop low limit
            THEN
            NIP               ( -- new_low high | low new_high )
    REPEAT
    2R> 2DROP DROP ;
```

```
: UPPER-LIMIT ( xt -- u )
   >R 0 0 0 BEGIN
       NIP SWAP 1+ TUCK R@ EXECUTE
       2DUP U>
   UNTIL
   R> DROP 2DROP ;

: RUNTIME-XT ( xt1 -- xt2 )
   >R :NONAME POSTPONE OVER R> COMPILE,
   POSTPONE U< POSTPONE 0= POSTPONE ; ;

: INVERSE-FUNCTION ( xt "<spaces>name" --  )
   DUP RUNTIME-XT SWAP UPPER-LIMIT
   CREATE , ,
   DOES> ( u1 -- u2 )
   2@ ROT ROT (BINSEARCH) ;

: SQUARE ( u1 -- u2 ) DUP * ;
' SQUARE INVERSE-FUNCTION SQUARE-ROOT

: CUBE ( u1 -- u2 ) DUP DUP * * ;
' CUBE INVERSE-FUNCTION CUBE-ROOT

etc.

I'm not very happy with UPPER-LIMIT. Is there a better way of
doing it?
```

I think this posting is worth a little commentary including, as it does, indirection and the use of `DOES>`.

Philip has provided the defining word `INVERSE-FUNCTION` that takes a function that is simple to calculate (e.g. `SQUARE`) and defines a word (e.g. `SQUARE-ROOT`) that calculates the more difficult inverse. He does this by cheating - repeatedly calculating the simple function with different values until the correct value is hit upon. Thanks to the magic of binary search, the number of different values needed is never more than a small number.

First of all, `(BINSEARCH)` performs a binary search between `0` and `UPPERLIMIT` for *f*`(Key)`, where *f* is the function (e.g. `CUBE` or `SQUARE`) which we are trying to invert. It uses `EXECUTE` to perform any function it is given - this is the indirection.

Philip's `(BINSEARCH)` is a more general derivative of the version published in Forthwrite Oct 91 by Gordon Charlton.

As an example, if we print out the trial values found by (BINSEARCH), we get:

```
1000 CUBE-ROOT .
trial values = 813 406 203 101 50 25 12 6 9 10 11
10 ok
```

The introduction of UPPER-LIMIT is (in my opinion) an improvement on Gordon's implementation as it is more general. It aims to calculate the largest value x where *f*(x) no longer fits into a single cell. It does this by incrementing up from x=0 until it finds an *f*(x) which appears[6] to be less than *f*(x-1).

By trying all the values (for a 32-bit cell, we have x=65536 for SQUARE and x=1626 for CUBE), UPPER-LIMIT is slow but will only be executed once - at compile time.

RUNTIME-XT creates an anonymous word suitable for passing to (BINSEARCH) that not only executes *f*(x) but also compares the result with the target, returning true if it is less than the target.

INVERSE-FUNCTION is the cunning word that defines the solving words like CUBE-ROOT. Look at the DOES> portion, the part that gets executed whenever CUBE-ROOT runs.

This fetches two values, the execution token xt and the upper limit from the data field of CUBE-ROOT and calls (BINSEARCH) to solve the problem.

The CREATE , , part of the word creates CUBE-ROOT and stores the two values found by executing RUNTIME-XT and UPPER-LIMIT. The anonymous word created by ' CUBE INVERSE-FUNCTION CUBE-ROOT is equivalent to:

```
: ANONYMOUS ( trial key -- trial less_than_key?)
  OVER CUBE U< 0=
;
```

This is an excellent illustration of the power of DOES>, a concept so foreign to other languages that it becomes very difficult to explain. INVERSE-FUNCTION provides the simplest possible way to create a solving word like CUBE-ROOT. Without the use of DOES>, you would have to define CUBE-ROOT as:

```
: CUBE-ROOT ( u1 -- u2 )
  [ ' CUBE UPPER-LIMIT ] LITERAL SWAP
  [ ' CUBE ] LITERAL (BINSEARCH)
;
```

[6] Only appears to be less because the most-significant bits of the number have been discarded to fit the result within a single cell.

(and you would also add `OVER ... U< 0=` into `(BINSEARCH)`). How much easier and simpler it is to write:

```
    ' CUBE INVERSE-FUNCTION CUBE-ROOT
```

Diary Date

euroFORTH 2000 15-18 Sept. in Cambridge

This is the most Northerly venue so far, so there should be some new faces (including mine). Look for details closer to the date on

http://dec.bournemouth.ac.uk/forth/euro/ef00.html

Letters

Graham Telfer responds positively to the last Editorial.

Fred Behringer sends us some unique pictures from Germany - good to put faces to familiar names at last.

New member Rob Probin is trying to build an interactive Forth on a PIC. This is unusual as most Forths for the little PIC are compiled on a host and downloaded.

Graham Telfer

From: gtelfer@po.synapse.ne.jp
Sent: 11 April 2000
To: Chris Jakeman
Subject: Forthwrite on the web & an esoteric language

Hi Chris,
I got the April edition of Forthwrite yesterday. I enjoyed reading it, as always. You asked for comments about putting the magazine on the web site. I think it's a bad idea for FIG UK from the foreign members' point of view. I could quite happily wait the extra couple of months to read the magazine and then save myself 20 pounds.

I think FIG UK is better and bigger than the magazine though, so I'll be renewing my membership when I get the next issue.

Perhaps a quarterly digest including some of the best articles from back issues would be a better idea.

I came across this site on the esoteric language ring:

SockZ

It's described as being forth like!

Yours
Graham Telfer

Fred Behringer

From: behringe@mathematik.tu-muenchen.de

Hi Chris,
The SWAP Dragon was handed over to Egmont Woitzel, this year's Dragon Award winner. Dr. Woitzel has gained merits in favour of Forth-Gesellschaft for over ten years. His dedication to Forth reaches even farther back, some years prior to his joining Forth-Gesellschaft.

He is the initiator and organizer of our new Web site.

Fritz was jumping around with his newly acquired digital camera and has taken some photos of the handing-over ceremony (Egmont and yours truly - and the dragon).

I now know the electorial procedure (Drachenrat) and I was tasting quite a bit of that alcoholic liquid necessary to seal the treaty with the dragon, but I'm afraid I must keep the secret from the other members and from the rest of the world.

Award-winner Egmont Woitzel took this picture of members of Forth Gesellschaft after the recent ceremony. From left to right, we have Friederich Prinz, Klaus Schleisiek, Ulrich Hoffmann, Bernd Paysan and Fred Behringer.

Rob Probin

From: rob@zedworld.demon.co.uk

Hi Chris,

I'm currently trying to read everything I can about Forth. And this is my primary reason to join Fig-UK - to get Forthwrite (delivered on paper is strangely satisfying as well?!) For the subscription amount, I am also willing to support FIG-UK, as I know anything costs money.

There are a few reasons why I'm interested in Forth but the primary reason?... I know quite a number of other languages and want something that gives me a more interactive environment. I think Forth is the only thing close, so rather than re-invent the wheel I feel it's certainly worth a go. I do believe in planned design, UML, OO design, etc, etc, but as evolutionary prototyping schools of thought went, it's not the only tool required.

Forthwrite, Yes, very good. Especially liked the FORML notes STATE, IMMEDIATE, POSTPONE stuff, Machine Forth and Letters.

Currently I'm developing two Forth kernels - a PIC interactive Forth (rather than John Hayhow's described project, or F2P, it's more like a Stamp BASIC type thing. I personally believe that it would be very useful for a lot of people. Xfree style license, with a support CD for sale.), and secondly a "game forth", for use in game development that we do (see http://www.lightsoft.co.uk), that integrates with C/C++. Our development machines are Macintoshes, but we target PC's as well.

Both are in the initial stages. The PIC forth has the inner interpreter running, with a few trial words, and the next the thing to do is decide on the minimal set of words, and of what are good extra's for bigger memory footprints.

And when I get 5, I'll have to get some back issues of Forthwrite ...

FIG UK Services to Members

Magazine	Forthwrite is our regular magazine, which has been in publication for more than 100 issues. Most of the contributions come from our own members and Chris Jakeman, the Editor, is always ready to assist new authors wishing to share their experiences of the Forth world.
Library	Our library provides a service unmatched by any other FIG chapter. Not only are all the major books available, but also conference proceedings, back-issues of Forthwrite and also of the magazine of International FIG, Forth Dimensions. The price of a loan is simply the cost of postage out and back.
Web Site	Jack Brien maintains our web site at http://forth.org.uk. He publishes details of FIG UK projects, a regularly-updated Forth News report, indexes to the Forthwrite magazine and the library as well as specialist contributions such as “Build Your Own Forth” and links to other sites. Don’t forget to check out the “FIG UK Hall of Fame”.
IRC	Software for accessing Internet Relay Chat is free and easy to use. FIG UK members (and a few others too) get together on the #FIG UK channel every month. Check Forthwrite for details.
Members	The members are our greatest asset. If you have a problem, don’t struggle in silence - someone will always be able to help. Do consider joining one of our joint projects. Undertaken by informal groups of members, these are very successful and an excellent way to gain both experience and good friends.
Beyond the UK	FIG UK has links with International FIG, the German Forth-Gesellschaft and the Dutch Forth Users Group. Some of our members have multiple memberships and we report progress and special events. FIG UK has attracted a core of overseas members; please ask if you want an accelerated postal delivery for your Forthwrite.