ISS 0265-5195
March
2003
Issue 120

Forthwrite

news people reviews projects programming

Forth and the Neuron Chip

FIGUK magazine:

Sorting a List

nnCron

Across the Big Teich

F11 UK Hardware Project

March
2003
Issue 120

events

news

reviews

programming

people

projects

Editorial

Welcome to new members Stephen Hayes, Anton Mans and Alan York. There's lots going on in Forth right now and FIG UK continues to play a substantial role. euroFORTH is back in the UK this year and this time FIG UK is getting involved – see details in this issue.

As well as the usual free Forths, Forth News includes a healthy list of new Forth Resources in this issue such as tools for accessing the Internet.

Look out for the first items in our new Forth Inside series – In the first one, we've managed to pull together some inside information on a well-hidden Forth story. There's quite a backlog of potential candidates for this series.

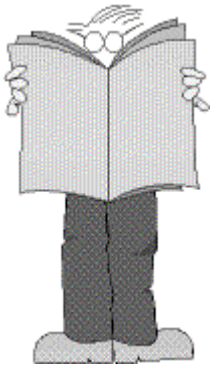
Sorting A List is a tutorial article. The original was written using MS Word as an experiment to combine code, comment and graphics in the same document. Our A5 format is a bit tight but it works well at A4. Expect to see more about this in due course.

Finally, congratulations to the winners of our FIG UK Awards 2002.

PS. Don't forget the monthly IRC session. Our next one is Saturday 5th April on the IRC server called "IRCNet", channel #FIGUK from 9:00pm.

Until next time, keep on Forthing,

Chris Jakeman



Forth News

Forth Events

euroFORTH 2003 will be held on 17th-19th October, in Ross-on-Wye, England.

Forth Resources

"Ugly Home Page" Returns

Neil Bawd has resurrected his famous web page which now includes 57 Forth code samples, some with tutorials. These include recursive and non-recursive versions of Quicksort.

Tools for Blocks

Gary Chanson posted a couple of tools for dealing with DOS-based block files.

One is a full featured block editor which supports multiple files, multi-file search and replace with regular expressions, and a lot more.

The second is a pair of programs which do smart conversion of block files to text files and text files to block files.

See:

http://www.mvps.org/ArcaneIncantations/forth.htm

comp.lang.forth by email

Sam Tardieu has set up a public email gateway at http://ada.eu.org/mailman/listinfo/comp.lang.forth

This service will send comp.lang.forth postings to you as they arrive or collected into a digest and will post your emails messages back to the newsgroup.

Tools for Internet

Marcel Hendrix (author of iForth) has made available a set of Internet-related tools. To any Forth that can access sockets, the tools add examples for posting email, fetching email and news, telnet and using http to get web pages.

See http://home.iae.nl/users/mhx/pipes&socks.html

MacForth Forum

MegaWolf have announced the opening of a public forum for discussion of its own MacForth and any other Mac-related topics such as MOPS Forth. There are a number of advantages in using this forum over Usernet, such as avoiding spam and harvesting of email addresses.

See http://macforth.com/discussion.html

Forth Scientific Library

The FSL needs a new team leader to continue Skip Carter's renowned efforts. Charles Montgomery, who has been a major contributor to the FSL, has offered his services "I do have some time available for helping with such an effort, do favor the concept and practices of the FSL as initiated by Skip, and am willing to try to help out in any way that 'the Community' would find useful."

Machine Characteristics in kForth

W.J.Cody published MACHAR - routines for finding the mathematical characteristics of a computer in a portable way - such as the largest integer.

David Williams has ported these to ANS Forth, see http://www-personal.umich.edu/~williams/archive/computation/dir.html

Krishna Myneni has ported these also to kForth at http://ccreweb.org/software/kforth/kforth4.html

Big Number Packages

Marcel Hendrix has made available Perfectly Scientific's GiantInt library in addition to the older bignum.frt based on Knuth. Both provide routines for large integer arithmetic and number theory.

Big numbers are mainly used in factorization of large (prime) numbers. Encryption and privacy are areas that indirectly depend on efficient factorization techniques.

Non-commercial Systems

New version of 4th

Hans Bezemer has upgraded 4th to v3.3d which now includes versions for 32-bit Windows as well as DOS and Linux. There are new words (eg `DEFER`) and multiple files/pipes can be used at once.

4th may be used as a standalone system or integrated with C, for example to provide scripting. It comes with extensive documentation and excamples, is close to ANS and, uniquely, claims to be crashproof.

Enth v0.4 released

Enth is a near-ANS multi-tasking Forth placed in the public domain by Sean Pringle. Unusually, it is standalone and does *not* require an operating system.

Enth is block-based and shadow blocks are available for comments. Like Chuck Moore's ColorForth, Enth uses colour to specify how words are to be interpreted.

See http://www.ynet.com.au/sean/

MinForth released

Andreas Kochenburger has published a small ANS Forth for DOS, DOS with DPMI, Windows and Linux. It is simple and portable, using a minimal amount of C code to implement the Forth virtual machine. It is robust, containing many crash-proofing features and also has a small interface to the Windows API. See http://home.t-online.de/home/andreas.kochenburger/

GForth will be faster

Bernd Paysan reports that the next version of Gforth will use dynamic superinstructions as well as conventional threaded code for extra performance. It will continue to be entirely ANS-conforming. Two papers on superinstructions were presented at euroFORTH 2002, see http://dec.bournemouth.ac.uk/forth/euro/ef02.html and one at euroFORTH 2001.

kForth updated to v1.0.13

The Windows version is now compatible with the Linux version. Both executable and source packages are available for download from: http://ccreweb.org/software/kforth/kforth.html

Commercial Systems

nnSoft have announced new versions of nnCron, nnCron LITE and nnBackup (see page 27 in this issue). nnCron runs unattended to start applications, display messages, dial and hang up, shutdown/ hibernate and wake up your PC, manage clipboard/files/registry and much more.

It is managed with easy-to-edit text crontab files and has a convenient graphical shell which can be used to remove, add, edit and run tasks, set up reminders and change program settings.

New features include the power-saving management and extensive documentation in English.

Planned Articles

We expect to publish items on the following topics shortly:

- Forth at the Joint European Torus (JET)
- Robust Interfacing
- Anaesthetic Dispenser

Forth is often the vital but invisible core of a product, and its contribution is recognised only by a few. This is the first of a series of "**Forth inside**" articles which reveals the use of Forth technology around the world.

Forth and the Neuron Chip

Over 20 years, Echelon Corporation of California (http://www.echelon.com) have built a world-wide business based on LonWorks technology - a special processor coupled with capable networking software. Our research has identified the Forth roots in this successful product.

Chris Jakeman and Bill Powell

Echelon's aim is to "*be the worldwide standard for networking devices and systems together in buildings, homes, and utilities*". With annual sales around $120m, a worldwide distribution network, and recognition by major standards bodies, Echelon looks close to achieving its objective.

Since its introduction in 1988, Echelon's technology solution—the LonWorks system—has been adopted by thousands of device and system manufacturers. Millions of connected LonWorks devices have been installed into buildings, factories, trains, homes, planes, and hundreds of other applications worldwide.

In a recent deal, industry giant Honeywell committed to "produce Echelon-based products for primary and secondary HVAC[1] plant controllers".

Echelon Corporation was founded and is still run today by CEO Mike Markkula, best known as one of the three founders of Apple. He held a variety of positions there, including Chairman, President/ CEO and Vice President of Marketing.

A striking application from Echelon's files is found in the Emirate Towers nearly 60 stories high, one an office and the other a hotel and shopping mall, where the aim is to "*create the most advanced and sophisticated office accommodation within the Arabian Gulf*". The lighting, HVAC and security systems are all networked with LonWorks. Because the lights are networked they can be re-organised without re-wiring whenever the office partitions are moved and come on automatically in the event of an alarm.

LonWorks also controls the lighting panels which make the building so attractive at night.

[1] HVAC: Heating, ventilation and air conditioning

A key element of the LonWorks solution is the network technology. The LonWorks protocol is specially engineered to suit control systems, follows the 7-layer ISO/OSI (Open Systems Interconnect) model, is a published ANS standard and allows two devices to communicate without needing to know anything about the topology of the network. Its advanced services include the ability to download a new application program across the network.

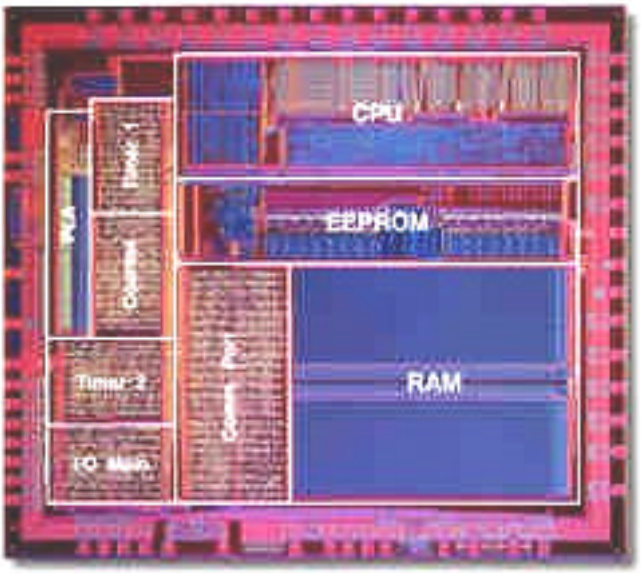
The software which implements this protocol is embedded into LonWorks devices, so anyone building an application will have the protocol available to minimise the size and complexity of their software.

Predictable Performance using CSMA

A key feature of the LonWorks network protocol is the ability to work robustly under overload conditions, such as may be experienced in an emergency. LonWorks achieves this, in contrast to the best known CSMA protocol - Ethernet, by providing "priority" messages, "predictable" messages and message acknowledgement. See http://www.echelon.com/Support/documentation/Bulletin/005-0060-01A.pdf for details.

LonWorks employs peer-to-peer connectivity rather than the traditional hierarchical approach. This has many advantages (eg. simpler design, no single point of failure) but as the same intelligence must be provided in each connected device, it is important to provide that intelligence cheaply and reliably. Which is where the Neuron Chip comes in.

The Neuron chip was originally sourced from Motorola and is now made by both Toshiba and Cypress. In its current incarnation, it uses 0.35 µm Flash technology to provide three identical 8-bit processors which run at 10 or 20MHz and is available as a family of devices of different sizes. Some 24 million Neuron Chips had been installed so far, some as cheaply as $3 each.



Each Neuron Chip is given a permanent unique-in-all-the-world 48-bit code. The three processors run in parallel, sharing the ALU and memory bus, and essentially provide three parallel hardware tasks dedicated to the CSMA protocol (see box), the higher layers of the ISO model and lastly the user application.

Each CPU has a byte-oriented data stack and a return stack which grow towards each other. This architecture is classical Forth and was chosen as it provides high code density. Since the user's application uses on-chip EEPROM memory, and this type of memory is the most

expensive in terms of die area per bit, it's important to have compact programs. As Harold Rabbie of Toshiba puts it, "Some of our chips have as little as ~300 bytes of EEPROM space for the user. Believe it or not, you can write code for a fully-functional analog sensor device in 100 or so bytes. The compiler largely generates compact calls to the hand-tuned firmware in the on-chip ROM."

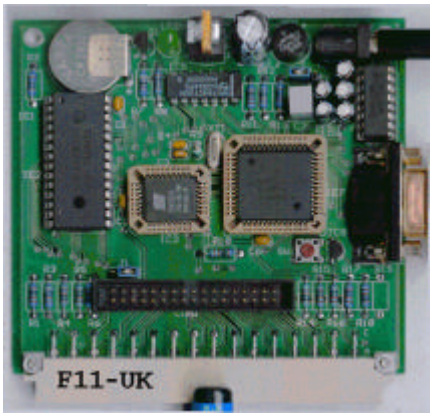
Although the Neuron Chip is a triple Forth processor, no-one has yet built a Forth development system for it. Instead, the rather expensive LonWorks compiler implements C (ANS with 3 extensions). The 256 instruction set includes classic Forth instructions like `DROP` but is not published by Echelon. The company does not want to support machine-level programming and have never released an assembly-level debugger or any of the documentation needed to program at that level.

The reason is that the Neuron Chip isn't a bare piece of silicon, but has ~16KB of embedded firmware in ROM that is tightly coupled with the compiler code generator.
That allows the developer to write C programs to do embedded I/O and networking without having to know any of the details of the implementation.

Implementing a Forth system is made more complicated as this firmware has "about 400 entry points in it, without any published documentation. Programming on the bare silicon is not an option either - you would lose all of the 7-layer networking functionality implemented in the firmware."

In conclusion, 24 million Neuron Chips have been sold so far, using Forth-style technology. However, the Forth-style instruction set is unreleased and, for good marketing reasons, only a C programming interface is provided for the application user. Forth continues to remain a well-hidden secret.

With acknowledgements to Vincent Pawlowski for his valuable information and reviewing the text.



F11-UK
FIG Hardware Project

The F11 UK mailing list continues to promote this board and extensions for it.

After something of a pause during the autumn, considerable activity was kicked off by a query from Garth Wilson (previously a member of USA FIG).

Graeme Dunbar organised an anonymous web-based poll to discover how users of the F11-UK kit have progressed.

Jeremy Fowell proposed the next step should be a range of Extender Boards, for example providing a real-time clock which could be used for logging purposes.

The first board would provide extra digital I/O and a socket to plug in custom prototype boards. Input is also being received from Paul Atkerstam, Boris Fennema, Mike Trueblood and Philip Eaton.

- My F11-UK board is working and I have been programming it.
- My board is running but there are still some bugs in it.
- The hardware is complete or nearly complete, but there are major software problems.
- The board has been assembled, but is faulty as far as I can tell.
- I have an unassembled or partly assembled board.
- I do not have a kit but am thinking of getting one.
- I'm interested, but don't have the tools etc to make one.
- I do not intend to get a kit as I have just a general interest in the subject.

Graeme has shared details of comparable boards designed at the Robert Gordon University.

This mailing list is open to anyone who has an interest in applying Forth to hardware and is not limited to FIG UK members, so feel free to "lurk" or even join in.

F11-UK

provides everything needed in a professional-quality low-cost Forth controller board.

Use it in industrial or hobby projects to control a wide range of devices using the well-known multi-tasking Pygmy Forth.

Designed for hosting from a Windows or DOS PC, you can test your application as it runs on the F11-UK board itself. The board was developed by FIG UK members to provide an easy way to explore the world of controlled devices – a niche where Forth excels.

The kit includes both hardware and software and is supported and sold to members at a nominal profit through a private company.

Software

PC-based PygmyHC11 Forth compiler running under DOS produces code for Motorola HC11 micro-controller.

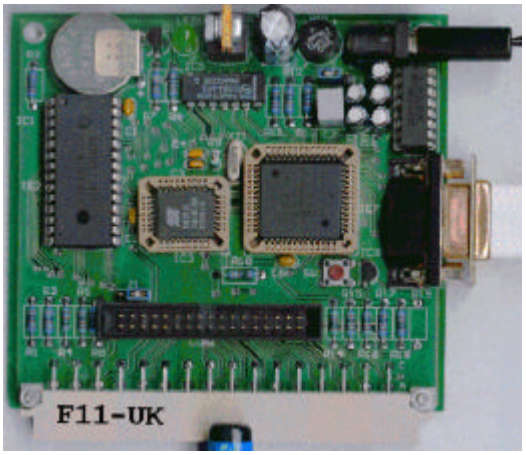
Code is downloaded via standard serial link from the PC to the FLASH memory (or RAM) on the F11-UK single board computer (SBC).

No dongle or programming adaptor of any kind is required.

Forth running on the SBC is interactive which makes debugging and testing much easier.

Multitasking and Assembly included.

The serial link can be disconnected to enable the SBC to function as a stand-alone unit.



All source code provided - 78 pages or so (unlike many commercial systems).

Around 30 pages of additional documentation is supplied including a full glossary of the 300 or so Forth words in the system.

Email mailing list for discussion and limited support.

Hardware:

Processor:
Motorola HC11 version E1 - 8 MHz (2 MHz E-Clock).

Memory:
32k x 8 FLASH
32k x 8 battery backed SRAM
512 x 8 EEPROM onboard HC11.

I/O:
20 lines plus 2 interrupts (IRQ & XIRQ).

Analogue in:
up to 8 lines using onboard 8-bit A/D.

Serial:
1. RS232, UART onboard HC11
2. Motorola SPI bus onboard HC11.

Expansion:
Via HC11 SPI serial bus using
2 or more of 20 available lines.

Timer system:
Inputs: 3 x 16-bit capture channels
Outputs: 4 x 16-bit compare channels.

PCB size: 103 x 100 mm.

Price to FIG UK members: £47.0 plus postage and packing (£2 UK, £4 overseas) plus $25.0 (US Dollars) for registration of 80x86 Pygmy Forth with the author Frank Sergeant.

Delivery: ex-stock.

More information: jeremy.fowell@btinternet.com and 0121 440 1809

Sorting a List

Chris Jakeman

This article began as a short reply to help a member new to Forth. Once started, it grew into an experiment in using a programmable word processor for editing and documenting source code.

The original request was

> “I can see how you build data arrays but how would you operate on a single linked list ?”

which led to the following reply

> “There's quite a lot of Forth material on lists. For example, Forth Dimensions ran a series from Neil Bawd called Stretching Standard Forth which includes Linked Lists (July 97 p20). Dick Pountain's book Object-Oriented Forth (from FIG UK Library) is as much about data structures as about OOF and Chapter 3 is entirely devoted to lists.
>
> Forth provides so much freedom that it becomes seductive. I can point you to several fascinating articles about doing clever things with lists - eg. OOF classes to develop lists and trees or rings used to implement lists, queues and sets. However I cannot find an article devoted to working with straightforward lists using ANS Forth. Neither can I find anything suitable in the on-line tutorials.”

and the final project brief

> “Thanks for the speedy and detailed reply. Currently I have a simple "sorting of list" application in mind - I want something small as a test case for myself. I basically would like to read a list of integer values off the stack and insert them into a list - sort them and output them later. Next stage would be to change the integer for string pointers and sort the strings.”

The aim of this article is to present a sound basis for working with lists of data. We assume that anything we do here with integer data, we can also apply to more complex data elements. We also assume that clarity is more important than performance. For example assertions are included where appropriate. These prevent errors but also document the conditions that the programmer has to satisfy if he is to rely on a word’s behaviour.

The commentary shows how the code grows, and replaces simple words with more complex ones in order to achieve the desired functionality.

Notes:

- The original article uses a dark green colour for the comments.

- The macros used to extract the source code from the document may be published in a later article.

- Thanks to FIG UK members Graeme Dunbar and Leo Wong for their feedback. Any errors are entirely mine.

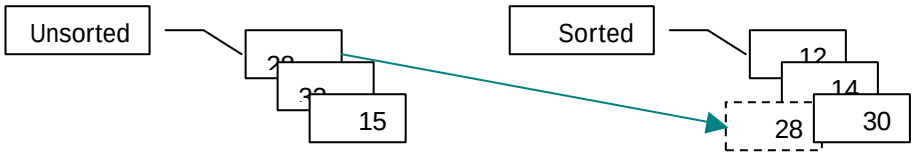
```
\ LIST.FTH
\ 2002-01-19 © Chris Jakeman
```

(Sorting a list of integers

We first move values from the stack into a single linked list called `Unsorted`. We then move the top value from `Unsorted` into the right position on a second list called `Sorted`.

This algorithm is called Insertion Sort and is like sorting a hand of playing cards. While there are faster ways to sort integers than using lists, this technique can also be used for objects of varying size, such as strings.

)

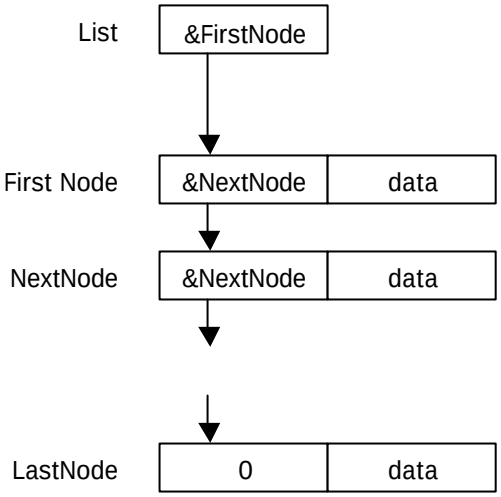


(Structure of a singly-linked list

A singly-linked list is sufficient for the purpose. Working with the list is much simpler if none of the nodes in the list is a special case. The way to avoid this is for the address of the list to be not the first node of the list but rather a pointer to that first node. Also, the field that links to the next node needs to lie at the start of each node. We use 0 to indicate the end of the list.

Also, with this structure, any word that operates on the whole list can also operate on the tail of the list.

)



Forth Style

Names
Forth allows punctuation characters in word names; `Alpha?` is common instead of `IsAlpha` and `>Value` instead of `ToValue`. By default, the scope of all words is global, but simple extensions are available providing scope local to a module or a single word.

I try to use nouns as names for words that add a value to the Data Stack (eg constants and variables) and verbs otherwise.

Variables
Avoid giving names to values - pass them on the stack whenever this is convenient. This example needed no variables at all.

Parameters
Words usually consume their arguments. Accompany each `: ... ;` word with a comment listing its overall stack behaviour, eg: `: + ( n m -- n+m )` With very few exceptions (`?dup` is a useful one), this documented behaviour should not depend on the input data.

Definitions
For many good reasons, words should be short and encapsulate a single idea. In `NodeDetach` below, the word `NodeUnlink` has been identified and factored out just to make the code more readable. You can also improve readability by giving a single phrase a line to itself.

Macros
A macro is merely an abbreviation for a text string and macros are used here to repeat common patterns for the loop that traverses a list.

```
anew [LIST-MANAGEMENT] \ ----------------------------------------
```

`anew <name>` has a single but useful purpose, to restore your Forth dictionary to the state it was in last time `anew <name>` was interpreted. Put this at the beginning of your source. Then, when you edit your source, simply re-load it and anew will remove the old source for you.

`anew` takes the place of the older `forget`

`[if] ... [then]` changes the source interpreted at compile-time. `[if]` takes a value of the Data Stack so `0 [if]` skips all the text to the next `[then]`.

`[if] ... [then]` can be nested.

Glossary

It is customary and helpful to provide a glossary of each module containing all the words exported from that module. This glossary is usually extracted from the source automatically.

```
0 [if]                              \ Glossary
2 cells constant >Node<             \ 1 cell for the link and 1 for value
: >Link ( &Node -- &Link )          \ Offset from address of node to link field
: >Value ( &Node -- &Value )        \ Another offset, read the name as ToValue
: NodeMake ( Value -- &Node )       \ Make an anonymous node and store Value
: ListCreate ( -- ++ )              \ Make a named list from the next word in the
                                    \ source.
: NodeAdvance ( &Node -- &NextNode )  \ Advances from one node to the next
: NodeInsert ( &NodeToInsert &PriorNode -- )
                                    \ Insert a single node into a list
ListCreate Unsorted                 \ Initial list of nodes
ListCreate Sorted                   \ Final list of nodes
: NodeUnlink ( &Node -- &NextNode|0 ) \ Stop node pointing at its successor
: NodeDetach ( &PriorNode -- &NodeDetached )
                                    \ Remove a single node from a list
: NodeShow ( &Node -- )             \  Show details of a node
: Macro :                           \ Use as Name " text " . The word Name will
                                    \  be replaced by the text
\ Pair of macros to visit all nodes of a list.
macro ListVisitAll(  " begin  NodeAdvance ?dup while >r " ( &List -- )
macro )ListVisitAll  " r> repeat " ( -- )
: ListShow ( &List -- )             \ Show all nodes in list,
\ Pair of macros to search the nodes of a list and exit with the node before the
\ matching node.
macro ListExitBefore(
            " dup 2>r begin r> r> drop dup NodeAdvance 2>r r@ while "
macro )ListExitBefore " until then 2r> drop "
: NodeFindBefore> ( n &List -- &Node )
                          \  Finds the node before the first node greater than n.
: NodeMove ( &FromPriorNode &ToPriorNode -- )
                                    \ Move a node from one place on a list to
                                    \ another place on the same or another list.
: NodeSort ( &PriorNode &TargetList -- )
                                    \ Insert a node into the TargetList
: ListSort ( &UnsortedList &SortedList -- )
                                    \ Move UnsortedList nodes into
                                    \  sequence in SortedList
: ShowLists ( -- )
: Try ( -- )                        \ TESTING: Finally test the whole sort routine
[then]
```

```
\ Define the list ------------------------------------------------

: ListCreate ( -- ++ )          \ Make a named list from the next word in the
                                \ source.
      create 0 ,                \ List will return address of pointer to first
                                \ node, but initially empty.
      does>  ( -- &List )       \ does> makes action clear but is not strictly
                                \ necessary.
;
```

ListCreate has similar behaviour to variable but documents what's happening and also ensures that the initial value is 0.

```
\ Define the node ------------------------------------------------
```

Each node links to the next one and also contains a single element of data, in this case an integer.

```
2 cells constant >Node<         \ 1 cell for the link and 1 for value
                                \ >Node< is my short-hand for size of node

: >Link ( &Node -- &Link )      \ Offset from address of node to link field
                                \ Read the name as ToLink
                                \ Does nothing, provided for readability
; immediate                     \ This ensures there is no run-time cost.
: >Value ( &Node -- &Value )    \ Another offset, read the name as ToValue
     cell+
;
: NodeMake ( Value -- &Node )   \ Make an anonymous node and store Value
     >r                         \ I use >r and r> instead of swap if it's
                                \ easier to read.
     >Node< allocate abort" Heap exhausted"   \ Make space for node
                                              \ and get address
     0 over >Link !             \ Ensure link points nowhere
     r> over >Value !           \ Store the Value
;
: NodeAdvance ( &Node -- &NextNode )  \ Advances from one node to the next
     >Link @                          \ Does little, provided for readability
;
```

```
\ Tools for working with nodes and lists ----------------------------

\ Design Note:
\ When working with a singly-linked list, it is often necessary to start from the node
\ before the one of interest. See NodeInsert below.

: NodeInsert ( &NodeToInsert &PriorNode -- )
                          \ Insert a single node into a list after the PriorNode.

                          \ The following line is an optional assertion.
    over NodeAdvance abort" NodeInsert: Node to insert is not single"

      2dup >link @        \ Get node pointed at. 2dup is same as over over
      swap >link !        \ Point the inserted node at it
      >link !             \ Attach inserted node
;
```

> Assertions (as above) are usually configured so that they can be included or excluded at compile-time using words like `\Assert ...` or `Assert( ... )Assert`

```
0 [if]  \ TESTING: At this point, we could create some nodes and attach them to
        \ a list, eg:
ListCreate Unsorted                 \ Initial list of nodes
ListCreate Sorted                   \ Final list of nodes

10 NodeMake  Unsorted NodeInsert
12 NodeMake  Unsorted NodeInsert
14 NodeMake  Unsorted NodeInsert
30 NodeMake  Unsorted NodeInsert
28 NodeMake  Unsorted NodeInsert
32 NodeMake  Unsorted NodeInsert
15 NodeMake  Unsorted NodeInsert
13 NodeMake  Unsorted NodeInsert
11 NodeMake  Unsorted NodeInsert
[then]
```

```
: NodeUnlink ( &Node -- &NextNode|0 ) \ Stop node pointing at its successor
      >link dup @                     \ Get &NextNode
      0 rot !                         \ Unlink it from Node
;
\ Note that the parameter is a prior node. The node prior to the first node is the list
\ itself.
: NodeDetach ( &PriorNode -- &NodeDetached )
                                      \ Remove a single node from a list

                                      \ The following line is an optional assertion.
                                      \ Skipping assertions at compile-time is simple.
      dup NodeAdvance 0= abort" NodeDetach: Node is missing"

      >Link >r                        \ Stash &Link pointer
      r@ @                            \ Get &Node to detach
      dup NodeUnlink                  \ and any node it points at.
      r> !                            \ Join up list again.
;
: NodeShow ( &Node -- )               \ Show details of a node
      cr
      dup .                           \ Show address
      >value @ .                      \ and value.
;


0 [if]   \ This definition is revised later.
: ListShow ( &List -- )               \ Show all nodes in list
     begin                            \ Eg. Use Unsorted ListShow to print list
        NodeAdvance
     ?dup while
        dup NodeShow
     repeat
;

Unsorted ListShow                     \ Example of use
```

```
\ Same as ListShow, but moves the current node onto the Return Stack to keep
\ the Data Stack clear. We are about to use this word as the basis for a pair of
\ macros and access to the Data Stack may be needed.
: ListShow2 ( &List -- )
     begin
        NodeAdvance
     ?dup while
        >r
        r@ NodeShow
        r>
     repeat
;
[then]
```

This loop is a common construct and we can factor it out into a pair of macros. These macros will package up the loop so that `ListShow` becomes simply:

```
ListVisitAll(  r@ NodeShow  )ListVisitAll
```

Similarly, you could count the number of entries in a list using:

```
0 swap ListVisitAll(  1+  )ListVisitAll
```

You can find the address of the last node of a list using:

```
dup ListVisitAll(  drop r@  )ListVisitAll
```

The advantages of using macros are more evident later, when we add some to search through the list.

```
: Macro :                       \ Use as Name " text " . The word Name
                                \ will be replaced by the text
    char parse                  \ Use ' text ' instead if text contains "
    postpone sliteral
    postpone evaluate
    postpone ; immediate        \ immediate so macros can be used inside
                                \ and outside definitions.
;

\ Pair of macros to visit all nodes of a list (based on ListShow2 above.)
macro ListVisitAll(  " begin  NodeAdvance ?dup while >r " ( &List -- )
macro )ListVisitAll  " r> repeat " ( -- )
\ Between macros, use r@ to access current node

: ListShow ( &List -- )         \ Show all nodes in list,
                                \ eg SortedList ListShow
    ListVisitAll(  r@ NodeShow  )ListVisitAll
;
```

Looking ahead, we will need to search the list and exit the loop as soon as a match has been found. Initially, we return the node containing the value that matches as in `NodeMatch` below. Once again, we use the Return Stack in order to keep the Data Stack clear.

If no node matches, then `NodeMatch` returns a value of 0.

The loop construction `begin .. while .. until .. then` used below may be unfamiliar. The loop has two exit points; `until` branches back to `begin` until the match is found. Once the list is exhausted, `while` will branch forward to `then`. Although the more familar pairings are `while .. repeat` and `if .. then`, there is nothing in ANS Forth which prevents the forward branch started by `while` from being resolved by a later `then` – see section A.3.2.2.2 of the draft ANS Forth document at ftp://ftp.forth.org/pub/Forth/Literature/ansforth.pdf

```
0 [if] \ Sample code as basis for macro ListExitAt
: NodeFind ( n &List -- &Node|0 )   \ Find the Node in List with value = n
      >r
      begin
         r> NodeAdvance >r
      r@ while                      \ While list not exhausted
         dup  r@ >Value @  =        \ Compare with n
      until                         \ Match found
      then                          \ List exhausted, node not found
      r> nip
;
```

As before, we can factor the loop out into a pair of macros. These macros will package up the loop so that `NodeFind` becomes simply:

```
ListExitAt(  dup  r@ >Value @  =  )ListExitAt  nip
```

The final `nip` discards the integer value we are trying to match. We do not include it in the macro so that we are not tied to matching integers, but can match more complex objects such as strings.

```
\ Pair of macros to search the nodes of a list and exit with the matching node.
                                        \ Based on NodeMatch above.
macro ListExitAt( " >r begin  r> NodeAdvance >r  r@ while "
macro )ListExitAt " until then r> "
                                        \ Use r@ to access current node
                                        \ Example of use
: NodeFind ( n &List -- &Node|0 )       \ Find Node in List with value n
      ListExitAt(  dup  r@ >Value @  =  )ListExitAt  nip
;
[then]
```

Although `ListExitAt` provides an important facility for matching, when sorting a singly-linked list we actually need the node **before** the matching node. This is because, when searching for a position to insert a node, we don't know we've found the right place in the list until we've moved beyond it. `NodeFindBefore` works differently, retaining some history. If no match can be found, the last node of the list is returned which is the right node to attach a new node to.

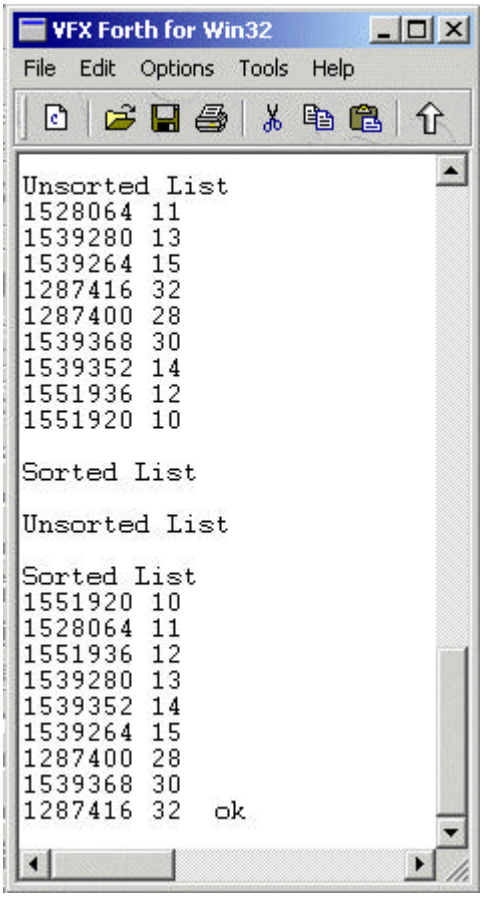
```
0 [if] \ Sample code as basis for macro ListExitBefore
: NodeFindBefore ( n &List -- &PriorNode )
                          \ Find Node in List with value n
      dup 2>r             \ Stash PriorNode and ThisNode on Return Stack.
                          \ Value of PriorNode is dummy but irrelevant here,
                          \  as it will be replaced at once
      begin
         r> r> drop dup       \ Replace PriorNode with current node
         NodeAdvance 2>r      \ Advance current node and stash both
      r@ while                \ While list not exhausted
         dup  r@ >Value @  = \ Compare with n
      until                   \ Match found
      then                    \ List exhausted, node not found
      2r> drop                \ Keep just PriorNode. This is never 0.
      nip
;
[then]
```

Rather than juggling the Return Stack, we could save the PriorNode in a variable. This often simplifies a complex word but the benefit here is not significant.

```
\ Pair of macros to search the nodes of a list and exit with the node before the
\ matching node. Based on NodeFindBefore above.
macro ListExitBefore(
            " dup 2>r begin r> r> drop dup NodeAdvance 2>r r@ while "

macro )ListExitBefore
             " until then 2r> drop "

: NodeFindBefore> ( n &List -- &Node )
                          \  Finds the node before the first node greater than n.
      ListExitBefore(  r@ >Value @  over  >  )ListExitBefore  nip
;


0 [if]    \ TESTING: At this point, we could test by finding the node, eg with a
          \ value before 25:
   Unsorted ListShow
   25 Unsorted NodeFindBefore> NodeShow
[then]
```

```
: NodeMove ( &FromPriorNode &ToPriorNode -- )
                                \ Move a node from one place on a list to
                                \ another place on the same or another list.
      >r                        \ Stash &ToPriorNode
      NodeDetach
      r> NodeInsert
;


0 [if] \ TESTING: At this point, we could test by moving the head node
       \  from one list to another:
   Unsorted ListShow
   Sorted ListShow
   Unsorted Sorted MoveNode     \ Move the head node between lists.
   Unsorted ListShow
   Sorted ListShow
[then]




\ Tools for sorting ------------------------------------------------

: NodeSort ( &PriorNode &TargetList -- ) \ Insert a node into the TargetList
      >r
      dup NodeAdvance >Value @     \ Find value of node to insert
      r>                           \ -- &PriorNode Value &TargetList
      NodeFindBefore>              \ Find place to insert it
      NodeMove                     \ Move node to TargetList
;
: ListSort ( &UnsortedList &SortedList -- ) \ Move UnsortedList nodes
                                            \ into sequence in SortedList
      \ The following line is an optional assertion.
      dup @ abort" ListSort: SortedList is not empty"

      over @ 0= if 2drop exit then  \ Nothing to sort so exit early

      begin
         over >Link @               \ Repeat until UnsortedList is empty
      while
         2dup NodeSort              \ Move first node of UnsortedList
      repeat                        \ into place on SortedList
      2drop
;
```

```
\ Final Testing -------------------------------
: ShowLists ( -- )
      cr cr ." Unsorted List" Unsorted ListShow
      cr cr ." Sorted List"     Sorted ListShow
;
: Try ( -- )
  \ TESTING: Finally we can test the whole sort routine
      ShowLists
      Unsorted Sorted ListSort
      ShowLists
;
try           \  TESTING: Do it
```



Sorting a list of integers can be achieved in far less code – see Leo's contribution below which compiles to just 30% of the version above. This is a dramatic difference and worth examination.

Leo's code is smaller through taking the integers to be sorted from the stack instead of creating an Unsorted list and words to move a node between lists. It also has no assertions or other checks. Most importantly, it does not create words which could be re-used but rather does just one thing with admirable economy.

```
\ slink.f Leo Wong 23 Jan 02002 +
\ Add integers from the stack, sorted, into a linked list
\ Usage: n1 ... nn n <list> sadds
\ Define a linked list
: list CREATE 0 , ;

\ Get address of a new node
: node ( -- addr ) ALIGN HERE ;

\ Add node N to its sorted position  ( Each node consists of: link value )
: sadd (  N list -- )
    node >R 0 , SWAP DUP >R ,      ( list ) ( R: Node n )
    BEGIN DUP @ DUP WHILE DUP CELL+ @ R@ < WHILE NIP REPEAT THEN
    R> DROP R@ ! R> SWAP ! ;

\ Add n integers to their sorted positions
: sadds ( N1 ... Nn n list -- ) SWAP 0 ?DO TUCK sadd LOOP DROP ;

\ Display a list's values
: .list ( list -- ) BEGIN @ ?DUP WHILE DUP CELL+ ? REPEAT ;

\ Test
list Sorted
10 12 14 30 28 32 15 13 11 9 Sorted sadds

Sorted .list
```

euroFORTH 2003

The 18th annual euroFORTH conference is being held on Fri Oct 17th to Sun 19th at the Royal Hotel, Ross-on-Wye, England

The annual conference (held in the UK every third year) returns this year to the UK. For details, see http://www.micross.co.uk/euroforth2003/Call_for_papers.html . (For Bill Stoddart's report on the previous year's conference, see Forthwrite Jan 2003.)

As well as presentation sessions there will be discussion workshops and demonstrations. A limited area of exhibition space can be made available, please contact the Conference organisers for further information.

It is hoped that a visit to an industrial installation controlled by a Micross Electronics system programmed in Forth can be arranged to take place on the Friday morning before the conference or during the Saturday in place of a workshop session.

The conference hotel is two minutes walk from the town centre and boasts dramatic views across the River Wye.

Ross-on-Wye has been attracting tourists since Victorian times, and a full visitor programme is planned for delegates' guests. Visits to Hereford, with its Cathedral housing the famous 12th century Mappa Mundi, the book town of Hay-on-Wye, and shopping in Cardiff are all possible.

Cost

Including two nights accommodation at the Royal Hotel with full board from 2pm Friday 17th till after traditional English lunch on Sunday 19th - £300.

Editor's Comment

euroFORTH is the best opportunity in the calendar for Forth users to get together. This year, FIG UK is joining in to make the event as successful as possible. Early phone calls to FIG UK members indicate that a substantial proportion are hoping to attend.

Micross Electronics are to be congratulated in negotiating reasonable prices for the event – significantly cheaper than the last event in the UK in 2000.

All the officers of FIG UK are planning to be there (possibly presenting a paper or two) and we would like to meet as many of you as possible.

So don't wait until the next UK euroFORTH in 2006; add October 17th to your calendar now.

Extenible Firmware Interface

Has anyone heard about the Intel's Extensible Firmware Interface (EFI)? This is a replacement for the PC BIOS and involves a byte code virtual machine, see http://www.intel.com/technology/efi. The EFI specification is primarily intended for the next generation of IA-32 and Itanium® Architecture-based computers, and is an outgrowth of the "Intel Boot Initiative" (IBI) program that began in 1998. Apparently the specification for EFI runs to a staggering 1,000 pages.

So far, I have been unable to find out how Intel's proprietary EFI compares with the Open Firmware standard - IEEE Std 1275-1994. Among the standard's many features, it provides a machine-independent device interface that can be used to boot plug-in cards without providing OS-specific or machine-dependent binary programs on the plug-in card. So plug-in-card manufacturers can easily support several independent computer architectures without needing to supply different firmware for each one.

Based on Sun Microsystem's OpenBoot 2.x implementations, Open Firmware complies with ANSI Forth. OpenBoot is Sun Microsystems' trademark for the firmware product that ships on SparcStations and SPARCServers.

Forth is often the vital but invisible core of a product, and its contribution is recognised only by a few. This is the second in a series of "**Forth inside**" articles which reveals the use of Forth technology around the world.

nnCron

nnCron is a scheduler, scripting tool and automation manager for Windows PCs developed by Nicholas Nemtzev of nnSoft (http://www.nncron.ru). We feature it here in Forthwrite for its Forth scripting. In fact, nnCron is written in SP-Forth.

At its simplest, nnCron takes commands in the same format at the Unix utility cron and executes them. For example:

```
# application 'chime.exe' is started at  12:15 every
weekday
15 12 * * 1-5 * c:\xxx\chime.exe
```

```
# the  pdf file named in the task is opened daily at 12:00 and at  17:00
0 12,17 * * * * cmd /c "e:\home\re.pdf"
```

```
# the command file named in the task is executed every 5 minutes
*/5 * * * * * d:\fido\bat\blstbbs.cmd
```

But nnCron can also be programmed in Forth words where `#( ... )#` takes the place of the standard `: ... ;` as:

```
#( test_memload
Action:
    MemLoad 90 >
    IF
        BALLOON: "MemLoad Warning"
        "More then  90%PERCENT% of available memory is used"
    THEN
)#
```

In fact nnCron is extended by a number of useful "plug-ins" all programmed in Forth. For example, the plug-in http.spf provides words to get the file at a URL or to find its Last Modified date. Using this nnCron can monitor pages on a web-site for changes.

A simple GUI is provided so that you can start, stop and monitor the nnCron tasks using the mouse.

There is simple and clear English documentation at the site and a user group at http://groups.yahoo.com/group/nncron

The full version is nnCron costs $25 (with a free 30-day evaluation period). If you do not require the programmability, try the free nnCron LITE.

Feedback on Forth Code index

I'm delighted to report that the new Forth Source Code Index on our web site (http://www.figuk.plus.com/codeindex/index.html) is proving successful. The Index is still growing but, of the 265 entries, 40% come from Forthwrite and ¾ of these are only available on paper.

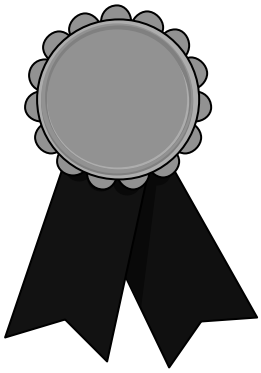
Visitors download the entries available electronically immediately but must request the paper ones (using a "click and send" email). 8 people have gone to the trouble of making such requests in the past 6 weeks, which extrapolates to about 60/year. Not bad for magazine issues which are several years old.

The items requested were:

- Alias alias alias
- Best string search
- Finite state machines
- Heapsort re-visited
- Object-oriented Forth – a minimal approach
- Radix, an extravagant sort
- Stack checking
- String pattern matcher

It was especially encouraging to get some feedback:

```
> Sent: 09 March 2003 17:20
>
> Hi Chris,
>
>     Thanks a lot for taking your time to do this.
>
>     I really appreciated it.
>
> > http://www.figuk.plus.com/articles/issue115.pdf
>
>     I will download this one as well.
>
>     FIG-UK and yourself are doing wonderful things for
>     Forth community.
```



Presenting The FIG UK Awards of 2002

These awards are given to encourage effort and recognise achievement.
The FIG UK Awards of 2001 were won by Chris Hainsworth and Dave Pochin.

Free membership

To everyone who sent in their nominations - "thank you". Looking back, a lot of good work was done during 2002 and our judges, the officers of FIG UK, have now chosen two winners. They each receive:

- a place in our web site's Hall of Fame
- this mention in Forthwrite
- ***a year's free membership***.

Achievement

Ed Hersom: a member for 17 years, a mathematician and frequent Forthwrite contributor.
(Ed's death was announced in the Jan issue.)

Forthwrite

Howerd Oakford: for his enthusiastic and insightful euroForth Conference reports

We congratulate Howerd on winning
- enjoy your year of free membership!

Volvovid@aol.com

Across the Big Teich

Henry Vinerts

This material was prepared for Vierte Dimension by Henry Vinerts, and
printed by kind permission of Forth Gesellschaft (German FIG)

FIG Silicon Valley Chapter Meeting - Dec 2002

Greetings!

The December SVFIG meeting surprised me with its being very early - on the second Saturday of the month, hence I managed to witness only the first half of it, in which a smaller than usual audience listened to Dr. Ting's report on his present projects in Taiwan. It turns out that the Taiwan Forth Interest Group has about 20 members, and that there is an area in Taiwan which is like Silicon Valley was twenty years ago, offering able programmers and hardware specialists with lots of ideas and enthusiasm.

Ting is working with a company which calls itself eForth Technology, Inc., and one of its latest developments is on the web, both in Chinese and in English - it is the Virtual Campus of the Forth Academy. We were able to visit the web site from
the computers next-door to our meeting room, and I must say that it is interesting. The full set of Chinese characters is available only on WindowsXP, yet other Windows platforms will still show enough to illustrate how programming in Forth can be done in Chinese, without having to learn English first. The web page is: http://www.eforth.com.tw . If you add /academy and click on "kid's classroom" you will see what I mean. For the grownups there is a wealth of information from Dr. Ting's writing's, Forth lessons, manuals, a lot of Forth in one location.

The one Forth you won't find there yet is Win32Forth, which is "too complicated" by Ting's description, and I agree even in my unqualified opinion. That is why I did not stay for the afternoon session, in which a number of devotees were going to work on polishing Win32Forth. "Chacun a son gout", as they say in French.

Mit besten Wuenschen,

Henry

FIG Silicon Valley Chapter Meeting - Jan 2003

Greetings, everybody!

SVFIG started the new year with a meeting announcement that did not list any scheduled talks, it called only for Dr. Ting and for John Peters to lead group discussions in planning future activities, concerning topics on classical aspects of Forth, as well as on Windows Forth.

Perhaps, as George Perry aptly noted, seeing fewer than a dozen participants in the morning session, most of us would rather be "end-users" than producers. I have made my observation a long time ago that SVFIG would have a shaky time hanging together, were it not for Ting and George. Now it
seems that John Peters has come with new enthusiasm, sharing his Win32Forth projects world-wide with some 40 people. The only problem with being able to communicate more and more easily world-wide over the Web is, as John himself said, the diminishing need and willingness to come to meetings and to
socialize with old friends face-to-face.

Clifford Stoll, in his book "The High-Tech Heretic," has devoted a chapter entitled "Isolated by the Internet." He writes that the electronic virtual community is not a positive social development and that, on the contrary, research by a pair of Carnegie Mellon University psychologists has shown that
there are serious negative long-term social effects, ranging from depression to loneliness. Let me express some hope that the Forth spirit will support some high-tech heretics who might well be the backbone of Forth interest groups world-wide.

Having exhausted in the morning session our efforts to plan for the future, we were delighted to and spent the rest of the day listening to impromptu talks by the teaching staff of Cogswell College and, of course, the ever-resourceful Dr. Ting.

The college project to introduce Forth as a music tool is still alive, and basic Forth instruction materials are welcome. On a "higher" plane, Susan Alexjander, who composes music based on frequencies found in nature, requested assistance in

pattern matching among the spectra of vibrations from pulsars, DNA, etc.

Ting briefed us on his current work with F# and showed us a hand-held GameBoy Advance unit into which he had loaded the full King James' Bible, both in English (in 4 MB) and in Chinese characters (which needed only 2 MB !). Clever and amazing.

To finish off the day, Ting led us into Chinese history, the story of 20 silk books preserved in a more than 2000-year old grave, which was discovered in Southern China in 1973. One of the books, the I Ching, was based on Taoist philosophy and was used similarly to an oracle, to answer any and all well-formulated questions relating to decision-making in life, by interpreting the hexagrams that are commented upon in the book. The "easy" way of arriving at the appropriate hexagrams by dividing batches of straws was described in the I Ching. After Ting explained and added his own theories of the probabilities of outputs, he told us the moral of his talk: "eForth and F#" are like the easy I Ching book; if there is a difficult book, it belongs with Win32Forth and SwiftForth.

Any questions?

As always,

Henry

FIG Silicon Valley Chapter Meeting - Feb 2003

Another Hello from California!

Friederich, your Vierte Dimension 1/2003 just arrived, and I want to tell you that even in my position as the world's oldest Forth novice, I find that it contains a lot that I can read with interest and understanding, but I must confess that I skip over the Forth words in preference to learning more German. (I almost wanted to refer to myself as a Forth "Gruenschnabel", but when I found that my Oxford Duden translates it to "whippersnapper," I decided to stick to my "novice" title instead.)

It appears to me that Gerard Baecker, who writes to Vierte Dimension in protest over Forthers' ("Fortherianer!") tendency to look down upon all other programming language users, is no whippersnapper himself. I have seen a number of such, who call themselves software engineers in this country, but the multitude of them come from schools of much lesser standing than HU-Berlin. From my personal polls, it seems that for every twenty C++ and Java students that I meet there might be only one who knows the meaning of RPN and stack architecture. Anyway, I think that Herr Baecker's letter makes
some valid points. Perhaps it should be translated into English for Forthwrite readers.

On February 22nd, 2003, we had another "Ting-less" meeting, barely exceeding a dozen members in attendance. The morning passed with us listening to Tim Duncan's lecture and CAC (computer-aided composition) samples in his music laboratory at Cogswell College. (I wrote about Tim, our host at the college, in my notes to the September, 2002, SVFIG meeting.) Tim still hopes to introduce a Forth language course in the Cogswell curriculum and to extend the use of Forth in the application of music technology. Discussions about the make-up of such a course continued after lunch.

Since the Win32Forth proponents were not present, other Forths could be suggested to Tim Duncan. If he did not know the common saying: "If you have seen one Forth, you have seen one Forth

...," he heard it now, as ANSForth was discussed. John Rible said that since the ANSForth committee had not been
able to agree on everything, a lot of inventions in it happened as compromises. The standard is followed by Forth, Inc., by Sun Microsystems and Apple in the Open Firmware, and has made Forth more accepted in certain circles. Suggested books for a course that follow the standard are the "Forth Programmer's Handbook" and "Forth Application Techniques," both available
from Forth, Inc.

George Perry did a good job kindling informal discussions, but they did not last to fill the day. Without Dr. Ting to pitch in, we heard one more "light-bulb" joke and left early. "How many Forth programmers does it take to replace a lightbulb? - Just one, but it has to be the same one who screwed it in in the first place."

Take care,

Henry

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

Letters

The Magazine Team are always pleased to get feedback and encouragement. The first letter comes from a new member who has already published several items in the magazine.

Boris Fennema

Hi Chris,

Happy new year !

I came across this flavour of bit-reversing routine in a book called "Hacker's Delight', Henry S. Warren Jr, Addison-Wesley, 2002. It has all sorts of bit shuffling and numerical tricks and tips - (most of them way over my poor head) but the bit-reversal routine was elegant.

I remembered reading about bit-reversal in assembler by Julian Noble (Sep 2001) using bit rotation and thought the attached may be of interest. (Bit-reversal is important in signal processing applications, such as the Fast Fourier Transform, and also in network routing - Ed)

I like the first routine best - it is the clearest I think - the second is what is suggested by the author and implement in 'C' but I find it does 'disturb' the symmetry of the first solution.

All the best,

Boris

Boris's code follows:

```
\ Hacker's Delight - bit reversal routines.

: (1bit)
    dup
    $55555555 and $1 lshift swap
    $aaaaaaaa and $1 rshift
    or
;
: (2bits)
       dup
       $33333333 and $2 lshift swap
       $cccccccc and $2 rshift
       or
;
: (4bits)
       dup
       $0f0f0f0f and $4 lshift swap
       $f0f0f0f0 and $4 rshift
       or
;
: (8bits)
       dup
       $00ff00ff and $8 lshift swap
       $ff00ff00 and $8 rshift
       or
;
: (16bits)
       dup
       $0000ffff and $10 lshift swap
       $ffff0000 and $10 rshift
       or
;

\ () can be executed in any order

: stib ( x -- x' ) (1bit) (2bits) (4bits) (8bits) (16bits) ;

: stib2 ( x -- x' )
    (1bit) (2bits) (4bits)
    >r
    r@ [ decimal ] 24 lshift
    r@ $ff00 and   $8 lshift
    r@ $8 rshift $ff00 and
    r> [ decimal ] 24 rshift
    or or or
;

binary

100110011100100 dup stib  swap .( input = ) . .( became stib ) . cr
100110011100100 dup stib2 swap .( input = ) . .( became stib2 ) . cr

decimal
```

Phil Burk is the author of pForth, the portable public domain ANS Forth with a kernel written in C, that has been ported to at least 13 platforms. Here is an exchange of emails that might be of wider interest.

Chris Jakeman

Hi Phil,

Didn't know anything about pForth today, but was inspired to look up http://www.softsynth.com/pforth/ when I saw that it was in use at VUB (Free University of Belgium).

Your FAQ asks about implementing KEY portably and losing the buffering which is all that the C library provides. Here is a copy of the Gforth IO.C file which tackles this problem for Unix, DOS and Windows. I see that it cribs from the "readline library for bash".

This file is an impressive achievement and also shows just how unnecessarily difficult software can be. Simple things should never be this hard!

Hope this is useful, though I suspect this complexity is what you were wanting to avoid...

Bye for now

Chris Jakeman

Phil Burk

Hello Chris,

Thanks for the file.

Wow! That code is incredible. I have figured out how to do KEY on Windows and Linux so I think I will just do that in the next release. Supporting the POSIX/Xenix/Aix/Unix variants is too scary.

Ironically, the easiest platforms on which to implement I/O are minimal embedded systems. I usually just check a hardware bit for KEY? and can just read chars from the UART directly. When Forth is concerned, operating systems are mostly an inconvenience.

Phil Burk

Finally, we re-publish a letter that appeared in Vierte Dimension recently, with thanks to Friederich Prinz for permission and Henry Vinerts for translation.

G. Baecker

With great interest I have read the issues of Vierte Dimension on the Web. I am not a "Forther" myself and actually I am writing this letter only because I am displeased with a certain tendency. From some of your articles one gets the impression that perhaps the younger generation that works with computers has no idea how to program effectively and instead prefers to use programming languages which are wasteful in performance.

I am a computer-science student and I would like to say a few words in defence of my chosen profession.

Naturally, one still learns how a microprocessor functions. (As part of the basic studies one must get to understand the workings of busses, memories, logic elements and circuits, etc.) One also learns how to implement algorithms using minimal resources of processing and memory (Turing machines, assemblers, etc.)

Every student of computer science knows what RPN is and how stack-programming works. If, despite that, little or hardly anything is being programmed in Forth, it is less due to ignorance than to what one needs to know after completion of computer-science studies. A computer-scientist is no engineer, i.e., the studies do not lead to expertise in one or more special programming languages. On the contrary, one learns paradigms, such as procedural, object-oriented, and logical ways of programming. It is even so that in computer science studies not very much emphasis is placed on practical programming (one should not have to study just programming). The objective is to learn to be able to distinguish with which method to solve a problem, if it can even be solved with the help of computers.

I do not find that Forth is the *best* programming language. It is more suitable than other languages for certain kinds of problems, that's all. Understanding of Forth can definitely contribute to understanding how computers function (and even that holds true only for the actually expanded forms of computers), but I think that the increasing complexity of the problems keeps demanding an ever increasing level of abstraction.

My first experiences in programming were with BASIC and assembler on a small 8-bit computer. Later I learned procedural languages like Pascal and C. Then came the functional language like LISP, the object-oriented ones, like Java, Smalltalk, etc., and the logical, as in Prolog or in production control systems. In between I used a Forth-like language for my HP calculator (which at that time was referred to as Reverse Polish Lisp, but in my opinion should have been called something like HP-Forth), yet I never came to the idea of installing such a low-level language into my PC.

Yes, I squander the resources of my computer (and massively so because of my affinity to languages like LISP, Python, and Prolog), but I save my most precious resource - my time. When I wish quickly to test an algorithm, I hardly give a thought to how I could accomplish that on a microcontroller with a 128-byte memory. (Which does not mean that I am not able to do that.)

The efficiency of a program is not necessarily demonstrated by by the fact that it is especially fast or small. The clarity and the readability of a program by others also can be regarded as criteria of efficiency (hence, for example, Python programs are fundamentally better than Perl programs). Having said this, I would be pleased if the Forth society would have a lesser tendency to misinterpret the lack of interest in Forth among the young "whippersnappers" as due to their incapability to do effective programming. Nobody declares that Forth is a dead language (there are other languages that really deserve such description and yet have spread themselves frighteningly far, such as MS Visual Basic, for example).

Forth is a tiny (unfortunately too unfamiliar), yet extremely appealing and charming facet of the computer world.

G. Baecker

Chairman	**Jeremy Fowell**,	11 Hitches Lane, EDGEBASTON B15 2LS 0121 440 1809 jeremy.fowell@btinternet.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 352373 cjakeman@bigfoot.com
Treasurer	**Neville Joseph,**	Marlowe House, Hale Road, WENDOVER HP22 6NE 01296 62 3167 naj@najoseph.demon.co.uk
Webmaster	**Jenny Brien,**	Windy Hill, Drumkeen, BALLINAMALLARD, Co. Fermanagh BT94 2HJ 02866 388 253 webmaster@figuk.plus.com
Librarian	**Graeme Dunbar**	Electrical Engineering, The Robert Gordon University, Schoolhill, ABERDEEN AB10 1FR 01651 882207 g.r.a.dunbar@rgu.ac.uk

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 Graeme.

FIG UK Web Site

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

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

FIG UK Services to Members

Magazine Forthwrite is our regular magazine, which has been in publication for over 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 Jenny Brien maintains our web site at http://www.fig-uk.org. She 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.