Just a short review!

JForth

Reviewed by Antoine Alary

Product	: JForth Version 1.2
Category	: Forth programming environment
Requirements	: AMIGA 512K, One drive
Format	: 2 non-protected diskettes
Size	: ----
Summary	: A very fast 32-bit implementation of the Forth Programming language. Specially designed for the Amiga.
Price	: $99.95 US
Developer	: Delta Research
Rating	: ***

Introduction

Many of us have heard of Forth as being a strange, almost unreadable language that attracts a quasi-religious following.

But if Forth is so strange why are people attracted to it? What can be the motivation to use an unreadable language? The main feature that attracts people to use Forth is its speed. When you need speed and low level control similar to that of Assembler but also want to retain the structures of high level programming languages, Forth may be your best bet.
And why do those who use it love it so much ? The secret of Forth's adoration by its users, is its extendability. Extendability means that every program you write can become part of the language itself. Forth is the only language that you have to like, because if you don't like it, it's your own fault, because Forth can be completely redefined to meet your needs or simply to please your taste. This can be done because of Forth's very loose syntax. To put it in the words of the authors of JForth's User's Manual, "Forth code is a bunch of words with spaces between them." (A 'word' being any sequence of printable ASCII characters without space between them.) It's hard to find a language that would give you more freedom of structure.
Another characteristic of classical Forth is that it produces very compact code. But this has become less important now that megabytes of memory are available.

Standards

It's easy to understand why Forth could have had trouble with standardisation, with such features as extendability and lack of syntax rules. One has only to think of BASIC and what happened to it after each developer and his dog, started creating his own subset of the language, to imagine the chaos that would have resulted if every Forth user started creating his own dialect. Luckily Forth was placed in the public domain by its creator, and the "Forth Interest Group" (known as FIG in the Forth community) was founded early to promote a standard and spread the good news about Forth. Since then two other standards have emerged: Forth 79 and Forth 83, (named after the date at which they were adopted). Truly standard Forths though, are still quite rare.

How does JForth fit in ?

JForth really departs from being standard in many ways, but also shows great respect for the three standards. (FIG, 79 and 83). In fact when JForth departs from the standards, there usually is a very good reason; either the old standards are outdated or there is simply a better way of doing things.

[illegible]

JForth is its "feel". Forth used to be a complete development environment, with operating system and editor integrated. JForth is closer to being just a compiler, like compilers for other languages. But don't get me wrong, JForth is a complete implementation of Forth, with its interpreter and compiler and all the necessary words, so that you never have to leave JForth if you don't want to. JForth runs under AmigaDOS as a well-behaved task. It allows standard interpreted keyboard input, but also accepts normal ASCII files as its input for compiling-interpreting, leaving you the choice to use your favorite editor instead of the more standard Forth SCREENs which were really a pain. For those who have hard-dying habits, JForth is distributed with a file containing the definitions of all the words needed to do standard BLOCK input-output and SCREEN source file editing. I must admit though that I haven't tried it (and hopefully never will) so I don't know how well it works. But judging by the quality of the other packages that I have used and the overall quality of all of JForth's software, it should work perfectly well.
I think this new "feel" should help newcomers to Forth to feel more at home with this language, having only to learn the language - not a whole environment. I hadn't used Forth for a while before I started using JForth, and coming back to it after using several other languages was greatly eased by this, dare I say, "more standard" user interface.

Threading

The second most important way JForth is not standard, has to do with the way JForth words are compiled into its dictionary. JForth uses what is called JSR-Threading. JSR stands for Jump to SubRoutine, and is a mnemonic in many assembly languages (68000 included). To understand what JSR-Threading means, you first have to know a bit about how Forth usually compiles new definitions. Roughly we can say that new words are defined by a list of previously defined words comprised between a colon ":" and a semi-colon ";". The usual way to compile such a definition is simply to note down in a list the addresses of the old words that make up the new word. Then when the new word is executed by the interpreter, successive jumps are made to the noted addresses until the end of the new word is reached. But these jumps are usually done by giving JSR instruction to the micro-processor. To save one level of indirection and some extra overhead, Delta Research decided to compile the JSRs right into the definition along with the associated addresses. This has the net effect of speeding up execution a lot and increasing the code size by up to 33%. A side effect is that the code thus produced is, in fact, true native 68000 machine code and is directly executable.

Inline Code

But JSR-Threading was just the first step. Some simple Forth words like DUP (duplicate top of stack) for example, have very short codes associated to them and doing the JSR and the associated RTS (ReTurn from Subroutine) is longer than executing the code itself. So the makers of JForth decided that the code for such words should be compiled directly "Inline" instead of compiling a JSR to it. But they had to decide where to stop: when is a word long enough to require a JSR-RTS pair, and when is it short enough to be compiled inline ? They knew that whatever choice they would make, someone somewhere would be unhappy with it. So in a true spirit of freedom they decided to leave the choice to the user. The variable MAX-INLINE can be set to the maximum length in bytes to be compiled Inline, words longer than MAX-INLINE get compiled as JSRs. (In fact, the control is somewhat more subtle, some words can be forced CALLED or INLINE and sometimes BSR (Branch to SubRoutine) are used instead of JSR because they are quicker.) All this makes me like to think of JForth as some kind of super MacroAssembler. And playing with JForth is a good way to get used to assembly language.

Local Variables

One of the greatest enhancements that have recently come to Forth
is the use of local variables. This feature [illegible]

name just as if they were ordinary VARIABLES. This greatly enhances readability by reducing the amount of juggling with the stack, with words such as SWAP ROT OVER >R and R>. Local Variables are also of great help to write re-entrant or recursive words. JForth's implementation is very complete and has a nice touch of elegance in the way Local Variables are declared. The names of the variables are enclosed between curly brackets and the standard symbol for the Top Of Stack can be used so that the declaration of Local Variables looks a lot like the usual stack usage describing comment that Forth programmers are used to seeing at the beginning of a definition. For example;

```
: MY-SWAP { a b --- b a }
   b a ;
```

clearly expresses its intent (and it works ! So does,

```
: YOUR-SWAP { a b --> b a } ;
```

but this last one uses another feature of local variables that I won't explain.)

Assemblers

It's now so common for a Forth system to include an Assembler package that it can almost been said that it is a Forth standard to include one. JForth departs from this standard by including not one, but two Assembler Packages. One of them uses the more Forthly way of doing things and is a reverse Polish notation assembler. The other is a Motorola type 68000 assembler. So here JForth gives us the best of both worlds. But I must confess I haven't had the time nor felt the need to use either of these assemblers. And I think that with such a good compiler and the provision for INLINE code, the need for assembly language should be greatly reduced.

Amiga Idiosyncrasies

Delta Research advertises JForth as having been "developed for the Amiga" and I have no trouble believing them. Much effort was put into JForth to ensure it would be a good development tool on the Amiga.

Libraries & Include Files

JForth includes facilities that provide full support of the Amiga libraries allowing us to call by name any function in any library. The facilities are both well implemented and easy to use. Words are defined to open, close and access all the existing libraries and clear instruction are given on how to interface to future libraries.

It is now a must for any serious programming package on the Amiga to give some sort of equivalent to the infamous "C .h include files". This is done in JForth with the help of the words :STRUCT , STRUCT; ..@ , ..! , UNION{ , }UNION{ , }UNION and a few others like BYTE , LONG etc. These words gives us the way to translate C structures into similar data structures in JForth. This in turn makes it possible to translate the .h files into JForth understandable .j files. The word H2J is even provided to automate this procedure. The main .h files have already been translated and are included on the JForth disks.

Even More

To make life even easier, JForth supplies the programmer with prefabricated words to access some of the most commonly used libraries. Simple interfaces to the graphic, math (Fast Floating Point only), memory management, Intuition's Menus and IDCMP Messages routines are included. File access is done via AmigaDOS. Full AmigaDOS file and path names are supported and any DOS command can be called from within JForth.

The Final Bonus

An Object Oriented programming development environment is included as a free bonus. Unfortunately I haven't used it yet so all I can say is that it is there. Good or bad, who knows ? I'd put my money on the "good" side.

The Darker Side

Up to here, my comments have been rather positive about JForth. but surely there must be some bad points to this software package. well, the "problem" has not so much to do with the software, as with the documentation. It can easily be seen that the [illegible]

Manual and Reference Guide [illegible]
software. Let me review each principal division of the 350+ page JForth Manual.

Table of Contents

You may think comments on a Table of Contents are superfluous, but in this case the T of C is a real life-saver. Read on to the INDEX section and you'll know what I mean. Just like real life-savers, this one has a hole. Non-letter words (words starting with non-letter characters like #, * or 2), are said to be between page 46 and 102 of the glossary. But words starting with [, \ or] are in fact found at pages 245 and following, because the glossary is sorted in the ASCIIbetical order. This is not too good, because very few of us know our ASCII order, as well as we know our our alphabet.

The Tutorial

This sub-section should in fact be on the other side of "The Darker Side" (The Brighter Side ?). The tutorial is by far the best section of the manual. It is very efficient, doesn't take hours teaching trivialities about the user stack, and it goes quickly to more interesting points without delving too deeply into matters of too high subtlety. You get hands-on experience fairly fast; after just two pages of truly basic stuff, you're up and playing with the stack. Two more pages and you're defining new words. In its very short 30 pages, this section shows you thru all the basic functions of Forth and even, some not so basic, such as CREATE and DOES. My only complaint about this section is that the section on local variables was left behind to be discussed in one of the later sections. Of course this tutorial is in no way a complete Forth course, but then again, it was never intended to be. It gives you just enough to get you started without becoming confused. For a more complete tutorial you are referred by the bibliographic section of the manual to some further reading. [Ed's Note: You are also referred to the AMnews Forth Tutorial]

The Glossary

This is the list of (supposedly) all the words in JForth. Unfortunately many are missing. For example, the non-self-evident word ALSO, that is used in some of the source code supplied with JForth, is missing. The same goes for PREVIOUS and many others. Furthermore, none of the words defined in the utilities files are described in that glossary. Instead they are explained in their respective pseudo-appendices, which makes for a very difficult time searching for them.

Even what is there isn't very usable. The typsetting of the manual leaves a lot to be desired. The text is all in a typewriter-like font, and the entries of the Glossary really don't stand out, to say the least. In fact nothing has even been attempted to make them stand out; no space before or after, no boldface no nothing except for an underline before the entry name that only helps to clutter things up even more. To make things worse, words like >R are found in the non-letter words section but those like .R are found in the alphabetical section near their alphabetically similar cousins (R in this case).

If you finally find a word in this mess (we will later see that the index is of no help at all in doing so), you will soon realize that the useful information is kept to a minimum. Stack diagrams are ok, description are short, examples are few. Then you have a list of standards to which this word obeys and finally a list of related words. This last idea could have been great but the words they show as being related must be related in a straaaange way! Like say he's the brother of the wife of the cousin of the doctor of my little sister's best friend... I thing longer descriptions would have been more appropriate, so would have been information about whether or not this word is IMMEDIATE etc. (But why bother, nobody will even find the word, and if someone is persistent enough to find it he doesn't need any help!)

THE PSEUDO-APPENDICES

[illegible] or
are 'ok', but could have either been incorporated in the tutorial or
the glossary in a more efficient way. A large majority are (as is the
rest of the manual) also sprinkled with spelling errors and typos, and
some are simply useless since they describe non-existent pieces of
software.

The Index

This has to be the (ahem) masterpiece of the manual; a computer
generated Index. Wow! But it's virtually useless. Each word in the
index has far too many entries and they point to just about anywhere
in the manual. I suspect that every occurrence of each word is noted.
Let's take an example. Say we want info on local variables.
Let's search in variables in hope of finding a sub-entry "local". No
such sub-entry exists, but variables has 14 entries, some of which are
page ranges. And this doesn't include entries for "variable" (another
17). Which one to look up? Let's try looking under local. Aha! only
nine entries! Now let's look up the last one since we usually find
more technical information near the end of most manuals. Page 300...
let's see now... Nope! That's not it either. We're in the Assembler
pseudo-appendix and the reference is to local branches. After a few
other tries (three) we find what we wanted. Of course we should have
guessed it, the right entry was a page range. Anyway it could have
been worse. For example, searching for "utilities" in the index,
points you to page 346 which is ...the index !

What's Missing?

To complete the scene, some advertised parts of the package are
missing. There is no DEBUG package (not implemented yet). The Floating
point package is minimal and incomplete. It only supports FFP's basic
operations. There is no "Optimizing target compiler", instead only a
TURNKEY application generator that I suspect to be very crude and
which provides no way of reading the command-line arguments.
Fixes to all these problems of course, are said to be "in the
works."

User Support

Advertisements say there is a free JForth newsletter and that
updates are available for postage, handling and media. But since I
received my review copy from AMnews, I have not yet received any
notices. A company would have to be very foolish or have suicidal
tendencies to advertise such services and not offer them, especially
for a young and promising but not quite finished product like JForth.
So I believe you can count on them in the future.
About direct user support, I can only say that I think that it is
unrealistic to believe that an ordinary user would call up Delta
Research in California to get answers to their problems. I know I
certainly wouldn't. (Do you know the cost of a call from Montreal,
Canada to Palo Alto, California ?)

Last Words (famous or otherwise)

All in all I think JForth is a very good product with an even
better future. It is already one of the fastest programming languages
available on the Amiga. See the benchmark section of AMnews for
confirmation. But the benchmarks don't tell the whole story.
Development time also can really be cut with Forth: compiling is
almost instantaneous, no linking is necessary and debugging is vastly
simplified by Forth's interactive environment.
I hope the "Darker Side" of this review won't leave you with a
bad impression of JForth because such is really not my intent. I like
JForth a lot and plan to use it even more, and I hope you will too.
I would like to leave you on this quote from JForth's Manual:
" This same potential problem exists in any powerful
language, not just Forth. This might be less likely in
BASIC, however, because BASIC protects you from a lot
of things, including the danger of writing powerful
programs."

JForth BenchMarks

(Ed's Note: These benchmarks are the same that were [illegible]
performed in several languages last [illegible]

```
READ                    1.14    ± Ø.Ø2 sec.
PRINT           5.ØØ    ± Ø.Ø2 sec.
                        4.46 ± Ø.Ø2 sec. ( with BlitzFonts )
SIEVE                   1.44    ± Ø.Ø2 sec. ( 1651 primes found )
FLOAT           2.98 ± Ø.Ø2 sec. ( Fast Floating Point Library )
............
.     /\    .
.    //\\   .
.   //==\\  .
.\\// Mnews.
. \/        .
............
```