MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England

tel: +44 (0)23 8063 1441
fax: +44 (0)23 8033 9691
net: sfp@mpeltd.demon.co.uk

Specification - last revision 20 August 2003

Proposal for FLINT - Forth LINT
This is a package that MPE and the Forth community will find useful. MPE will be happy to discuss commercial terms with anyone who can develop this package. Please comment to this proposal as soon as possible. If you need more information please contact us as soon as possible.

Objectives
We need a tool to perform stack and type checking in Forth source code. This tool is provisionally called FLINT by analogy with LINT. FLINT provides error checking above that normally provided by the MPE Forth compilers. The tool is a standalone utility that processes ASCII text files that may have tabs as separators. Screen file support is not required. The typical use will be validation of megabytes of existing code.

The tool can be written in Forth or any other language. If it is written in Forth, it should be written using MPE's VFX Forth for Windows. A copy can be supplied for the job. Full source code for FLINT must be provided, and this source code must be well commented. Full design notes must be provided so that an experienced programmer can extend and maintain the software.

Requirements
FLINT provides a consistency check of Forth source code. It examines a given file/selection of files, checking only high level definitions for stack depth and stack types. If an error is detected, the offending procedure, line, and file should be output to an error log, and checking should proceed after the current procedure.

This tool may be applied to code written for:
  Any Forth
  MPE 16 bit cross compiler
  MPE 32 bit cross compiler

Consequently, it must be possible to define the Forth kernel to the checker by providing a list of known good words and their stack effects in a "glossary file".

Stack Depth Checking
FLINT must use the formal stack comment at the start of the word to check that the required stack action is actually achieved. If FLINT can cope with **?DUP IF ... ELSE ... ENDIF** this would be good, but otherwise a warning should be

given. Variable stack effects at the inputs and outputs are to be treated as illegal.

Stack Type checking
FLINT should keep track of the data types on the stack. In order to do this a formal method of defining the required action must be used, the proposed method is described below.

Thus if double numbers are being processed the phrase **D+ !** should raise a warning.

Formal Stack Comments
House style at MPE requires the following layout at the start of a word:

```
: NAME            \ entry -- exit ; description
            \ a b c d -- x y z ; performs a transform
  ...........
;

: D@        \ addr  --  d ;  fetches  a  32bit  double  from
address
  ......
;
```

The descriptors a b c d x y z are presently only descriptive, but we can parse a definition so that after a **:**, the text between \ and **;** forms a formal stack comment. Other tools are used to scan these lines for documentation purposes, so we would prefer this format to remain. We also permit the following format for lists of **CONSTANT**s and **VARIABLE**s.

```
\ -- addr
VARIABLE V1
VARIABLE V2
...
```

In order to add formal information that FLINT can access, we would propose to add a prefix to the informal notation to provide type information. This information is provided in a similar way as is used by Microsoft programmers in C using `Hungarian notation'.

For example, using **D@** again.

```
: D@        \ addr -- d ; traditional notation

: D@        \ ad.addr -- d.dnum ; formal notation
```

where the formal notation **ad.** specifies that the input (called **addr**) is the address of a double number, and **d.** specifies that the output (called **dnum**) is a double number. We need the formal part as well as the informal part of a stack item description. If there is no '**.**' in the middle the description is assumed to be the formal type information.

To support local variables, MPE uses the notation:

```
: FOO { ip1 ip2 ... | lv1 lv2 ... -- ops ... }
```

```
  ip1 ip2 + -> lv1                       \ like `TO' variables
  ip1 ip2 * -> lv2
  ...
;
```

Between the { and the | are the inputs. Between | and the -- are the local variables, and after -- are the outputs, which are back on the stack. Inputs and local variables can be accessed by name (scope and visibility for that word only). Outputs are dummies.

Assertions
On the assumption that the FLINT will not always be able to cope, the user must be able to insert assertions in the middle of a procedure. An assertion might be of the form:

```
: FOO        \ an.buffer -- d.average
  ........
  A{ ad.addr -- }A      \ force stack effect to be as stated
  ........
;
```

where **A{** opens the assertion and **}A** closes the assertion that the stack effect calculated so far is to be transformed into the state given by the assertion.

Structures
Simple structures are often defined in the forms:

```
CREATE BAR1 \ -- as.buff        ; structure is reserved
  X ,  Y ,  Z ,

or:

CREATE BAR2 \ -- as.buff        ; structure is reserved
  N ALLOT
```

It would be desirable to be able to formally specify the layout of a structure so as to avoid the use of assertions later in the code. The proposed notation is of the form:

```
CREATE BAR3 \ -- as{ ad d n an }as ; pointer to double,
double ....
```

Predefined kernel words
We should be able to supply a list of predefined words (function prototypes) in order to support the Forth kernel. This requires the notation to be defined. The implementor may define this notation. Additionally it may also be necessary that the user defines the types being used. This type notation may be defined by the implementor.

Miscellaneous
These topics are not mandatory.

Can we support vocabularies and **ONLY ALSO** ... search order control?

Can we support defining words?

Related work

Most of the related work in this field has been performed in Europe by Bill Stoddart at Teesside University (UK), and by Janos Poial at Tartu University (Estonia). The references may be found in the EuroForth proceedings.