⇐ STABLE-INDEX ⇒ tutorial ⇒ glossar essay ⇒ cook book

STABLE - ESSAY

an extreme small an fast FORTH-VM

Stable is a **very** different language. It meight be closer to FORTH but has very unique attributes not found there. STABLE looks a bit alien to more traditional developer but one hour should be enought to get used to it (see the ⇒ tutorial).

Did you ever had the wish to change the compiler or syntax of your system. Had you ever had the wish to be independent of your compiler distributor. Had you ever had the wish to have only one file where you can do everything from editing to compile and distribute. Even at your customers computer. Without downloading gigs of Software.

Now think of this. If you could have a System, capable of say, 50 primary functions, where you can build almos anything you want. Isn't it easy to write those 50 functions by hand. For only one person ? You can play any rule you want, you simply can introduce profiling, code coverage, debugging (watchpoints, breakpoints,..). But at a first step you can use our predefined free available tool, written in plain C. So, not the tool is valuable, it's the Idea behind of STABLE.

But, and this seems to be very valuable: one can **proof** the **correctness** of a system so small, easily. There are no hidden rules like the priority order of executed statements, sequence points, undefined behavior and so on. There are no such things like compiler errors or code generation errors (because we don't have a compiler).

The other important feature is, that a system so small, easily fits into first level cache. So it is able to run in full speed of the CPU. Because STABLE is that small the STABLE runtime system can be written in assembly.

And now think of the area of embedded devices. Where CPU power and memory is very limited. STABLE could be its own operating system, running on bare metal. In which programming language can you write a fibonacci programm within 13 Bytes) ?

```
usage: stable fib.txt 24
1       1       2       3       5       8       13      21
34      55      89      144     233     377     610     987
1597    2584    4181    6765    10946   17711   28657   46368

code: 0 1#[.9,$@+a-;]

explain: (see fib.andi.txt for the trace output)
Code  stack-effekt  comment
0     ( 0)         push start literal 0 on data stack
1     ( 0 1)       push next literal on data stack
[     ( 0 1)       begin a while loop
#     ( 0 1 1)     duplicate TOS (dup top of data stack)
.     ( 0 1)       print out TOS (1) as decimal number
9     ( 0 1 9)     push literal 9 on data stack
,     ( 0 1)       send TOS (9) to terminal, 9 is tabulator
$     ( 1 0)       swap exchange TOS with next of stack (NOS)
@     ( 1 0 1)     over, copy NOS (next of stack) to TOS
+     ( 1 1)       add
a     ( 1 1)       select register a (contains counter)
-     ( 1 1)       decrement register a
;     ( 1 1 a)     get the content of register a
]     ( 1 1)       loop to [ until a is 0 (false)

for clearity the next cycle will be explained as well
[     ( 1 1)       begin a while loop
#     ( 1 1 1)     duplicate TOS (dup top of data stack)
.     ( 1 1)       print out TOS (1) as decimal number
9     ( 1 1 9)     push literal 9 on data stack
,     ( 1 1)       send TOS (9) to terminal, 9 is tabulator
$     ( 1 1)       swap exchange TOS with next of stack (NOS)
@     ( 1 1 1)     over, copy NOS (next of stack) to TOS
+     ( 1 2)       add
a     ( 1 2)       select register a (contains counter)
-     ( 1 2)       decrement register a
```

```
;     ( 1 2 a)    get the content of register a
]     ( 1 2)      loop to [ until a is 0 (false)

for clearity the 3rd cycle will be explained as well
[     ( 1 2)      begin a while loop
#     ( 1 2 2)    duplicate TOS (dup top of data stack)
.     ( 1 2)      print out TOS (1) as decimal number
9     ( 1 2 9)    push literal 9 on data stack
,     ( 1 2)      send TOS (9) to terminal, 9 is tabulator
$     ( 2 1)      swap exchange TOS with next of stack (NOS)
@     ( 2 1 2)    over, copy NOS (next of stack) to TOS
+     ( 2 3)      add
a     ( 2 3)      select register a (contains counter)
-     ( 2 3)      decrement register a
;     ( 2 3 a)    get the content of register a
]     ( 2 3)      loop to [ until a is 0 (false)
```

And STABLE could be interactive, so one can develop, test and maintain applications on the same embedded device. No need for a large computer which compiles down the application and than you have to download the code into the board - no, you are directly connected to the board and get the feedback immediately. The PC plays the role of a dumb terminal.

STABLE is the **IDEA** and the basic set of well thought out one character operations within the ASCII character set. You can implement STABLE in any language you want, even Assembly or COBOL and you always will be portable and proofed. We provide a free reference system written in C for linux, Mac and Windows (see ⇒ downloads).

ENJOY STABLE!