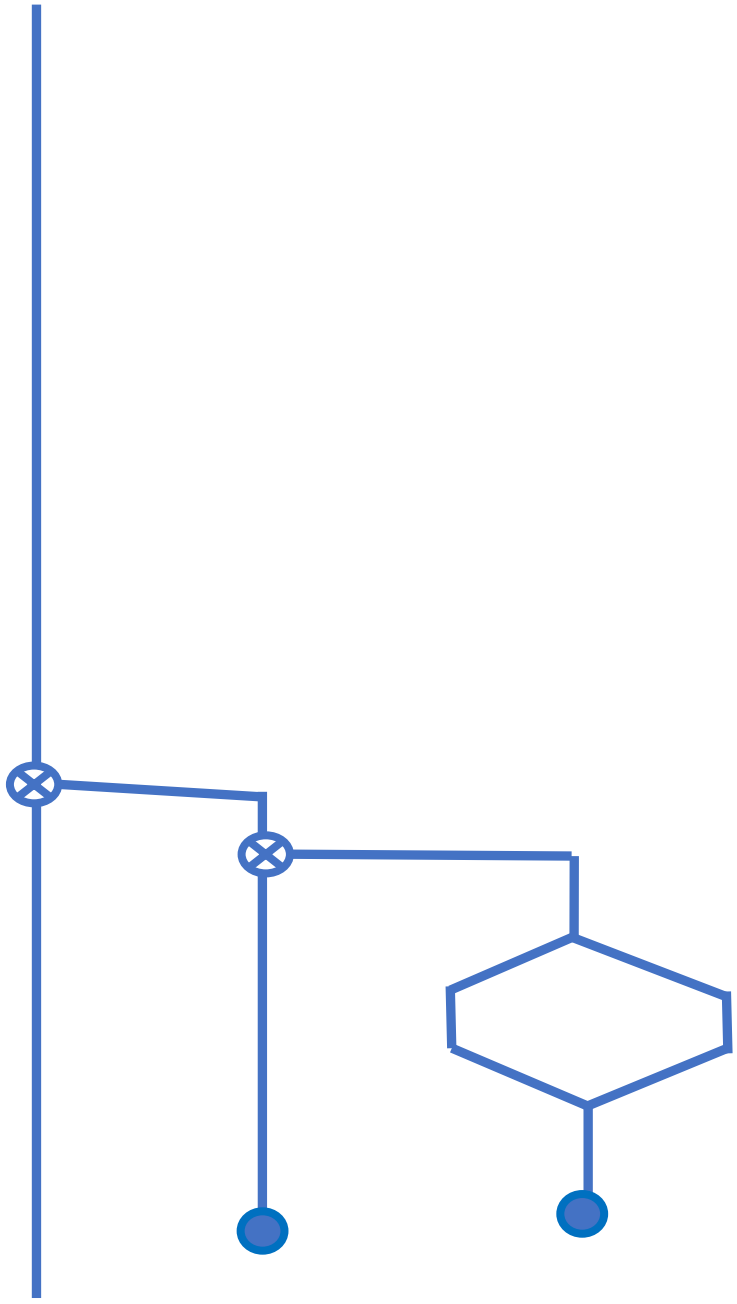


Program, Test, Program, Test

Silicon Valley
Forth Interest Group
Mar. 23, 2024
Bill Ragsdale

The Need

While coding we should test incrementally.

Code - Test; Code – Test

The Need

While coding we should test incrementally.
Code - Test; Code – Test

But once the code is verified we can suppress the testing.

The Need

While coding we should test incrementally.
Code - Test; Code – Test

But once the code is verified we can suppress the testing.

But with major changes we may want to restore the testing.

The Need

While coding we should test incrementally.
Code - Test; Code – Test

But once the code is verified we can suppress the testing.

But with major changes we may want to restore the testing.

When the code is completed the tests may be repeated or only tested from one point onward.

The Need II

Historically the selective execution has been handled by [IF] [ELSE] [THEN]

On [IF], [ELSE] and [THEN]

[IF] [ELSE] [THEN] are interpreted conditionals. The following text may be skipped or executed/compiled.

f [IF] <True- code> [ELSE] <False-code> [THEN]

On [IF], [ELSE] and [THEN]

[IF] [ELSE] [THEN] are interpreted conditionals. The following text may be skipped or executed/compiled.

f [IF] <True- code> [ELSE] <False-code> [THEN]

[IF] Selectively executes/compiles <True-code> or <False-code> based on the Boolean f.

On [IF], [ELSE] and [THEN]

[IF] [ELSE] [THEN] are interpreted conditionals. The following text may be skipped or executed/compiled.

f [IF] <True- code> [ELSE] <False-code> [THEN]

[IF] Selectively executes/compiles <True-code> or <False-code> based on the Boolean f.

[IF] is an immediate word. It contains a mini-interpreter that looks for [ELSE] and [THEN].

On [IF], [ELSE] and [THEN]

[IF] [ELSE] [THEN] are interpreted conditionals. The following text may be skipped or executed/compiled.

f [IF] <True- code> [ELSE] <False-code> [THEN]

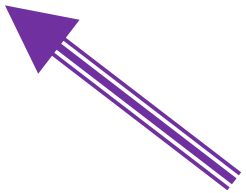
[IF] Selectively executes/compiles <True-code> or <False-code> based on the Boolean f.

[IF] is an immediate word. It contains a mini-interpreter that looks for [ELSE] and [THEN].

But how to selectively activate test sequences?

My Variation On [IF]

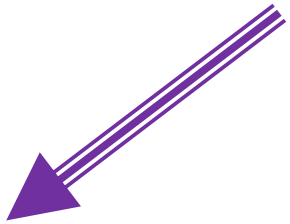
```
7 VALUE TestLimit
\ Execute tests numbered n and greater.


: *IF  ( n --- )
    TestLimit >= TestLimit and  [compile] [IF]  ;
```



Here is the magic.

My Variation On [IF]

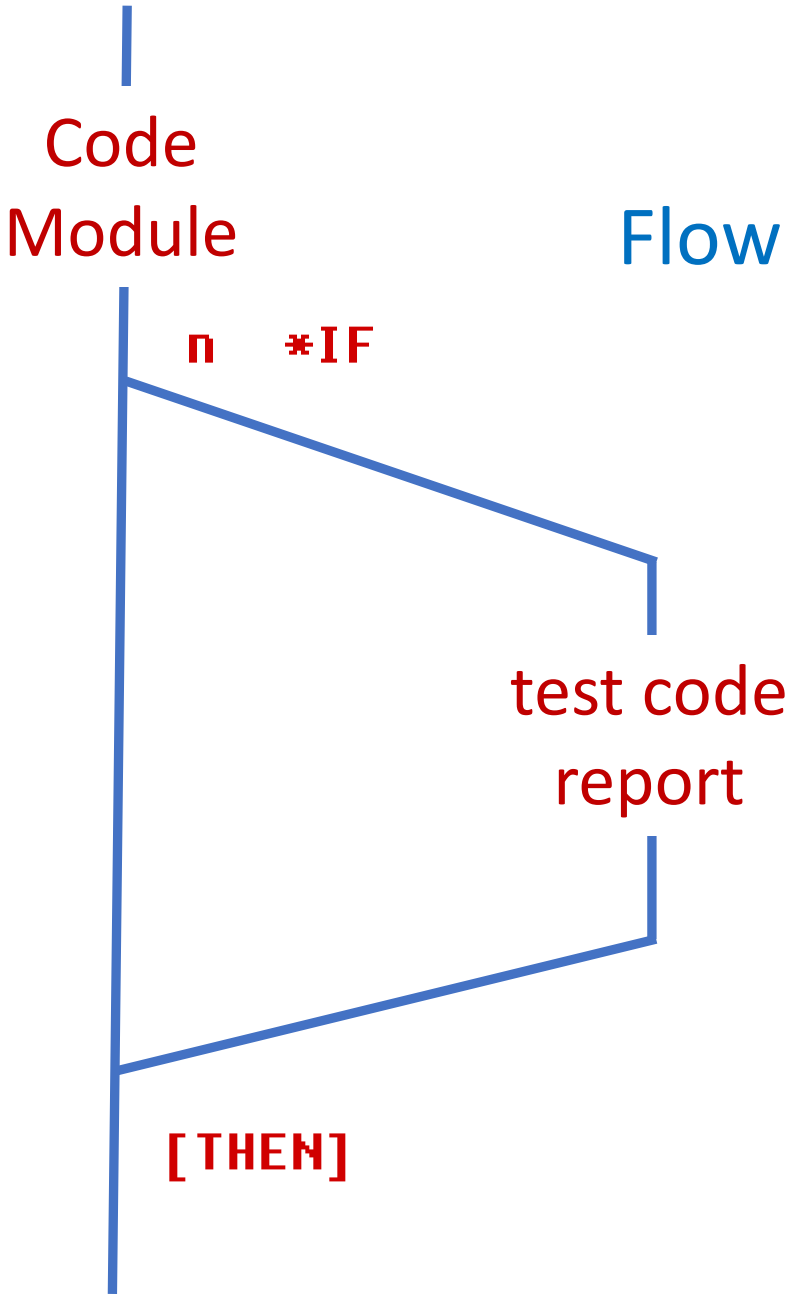
```
7 VALUE TestLimit
\ Execute tests numbered N and greater.


: *IF  ( n --- )
    TestLimit >= TestLimit and  [compile] [IF]  ;
```



*IF accepts a number and compares it to TestLimit.

If equal to or greater than TestLimit, interpretation continues until [THEN]. If TestLimit is zero, no tests are performed.

Flow of *IF

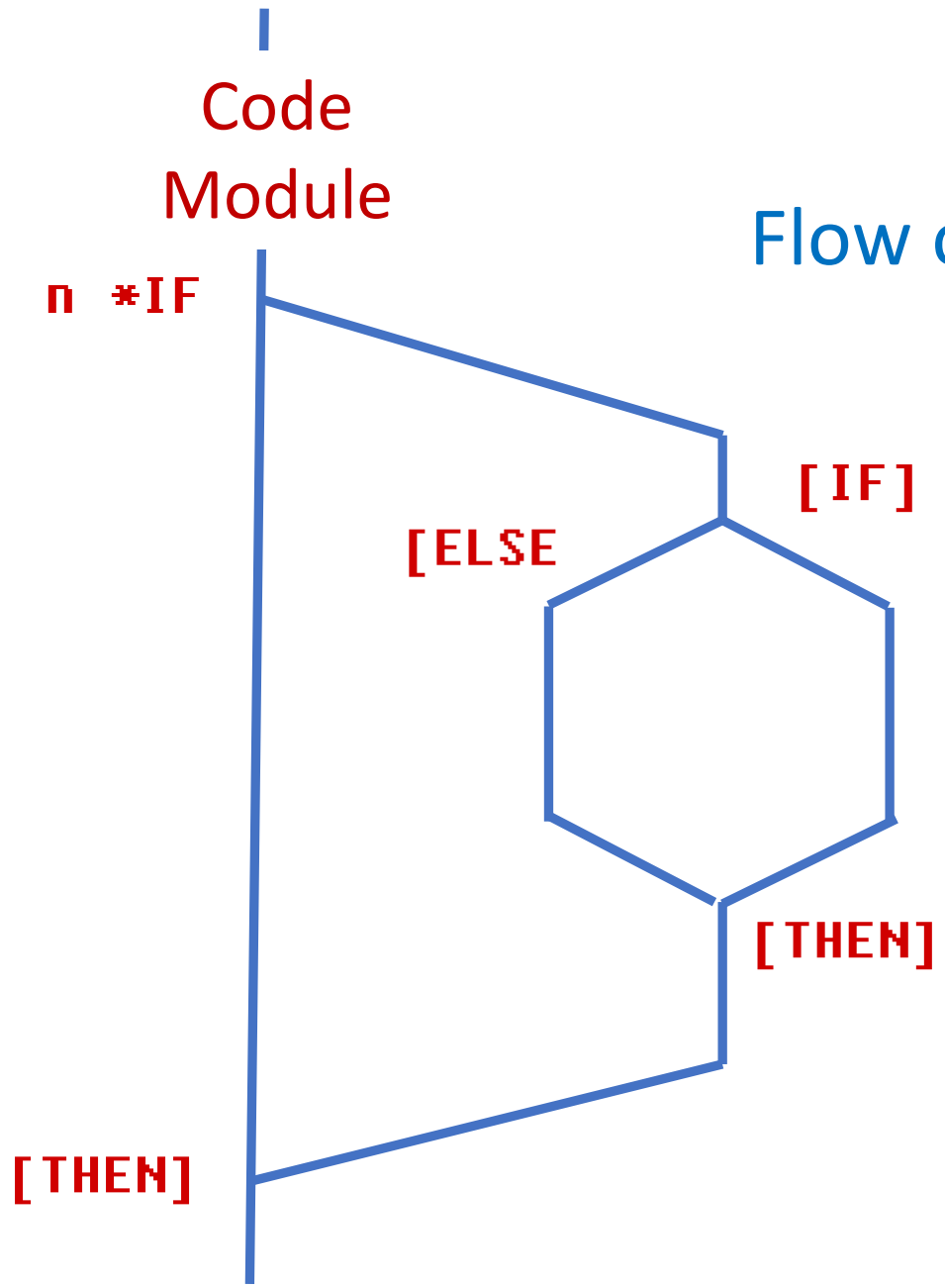


Using *IF

```
0 *IF   .( 0  Test of }sub-random )
        A{ 1 2 1 2 1000e }SubRandom
        A{ }list
  [then]
```

Using *IF

```
8 *IF   .( 8  Test of }sub-random )
        A{ 1 2 1 2 1000e }SubRandom
        A{ }list
  [then]

  And see:

  8  Test of }sub-random
  .00000000  .00000000  .00000000
  .00000000  657.67050  741.66470
  .00000000  619.29350  399.92490
```


Code
Module
Flow of *IF
n *IF
[IF]
[ELSE
[THEN]
[THEN]

Math Operators Test

```
4 *IF   .( 4  Test +  -   /    )
    100 500 +  200 - 4 /   100  =
       [if]   .( Got the expected 100. )
       [else] .( Error in math operators ) 4 bells [then]
  [then]
```

Math Operators Test

```
4 *IF   .( 4  Test +  -   /    )
    100 500 +  200 - 4 /   100  =
       [if]   .( Got the expected 100. )
       [else] .( Error in math operators ) 4 bells [then]
  [then]

And see:

4 Test +  -   /    Got the expected 100.
```

Math Operators Test

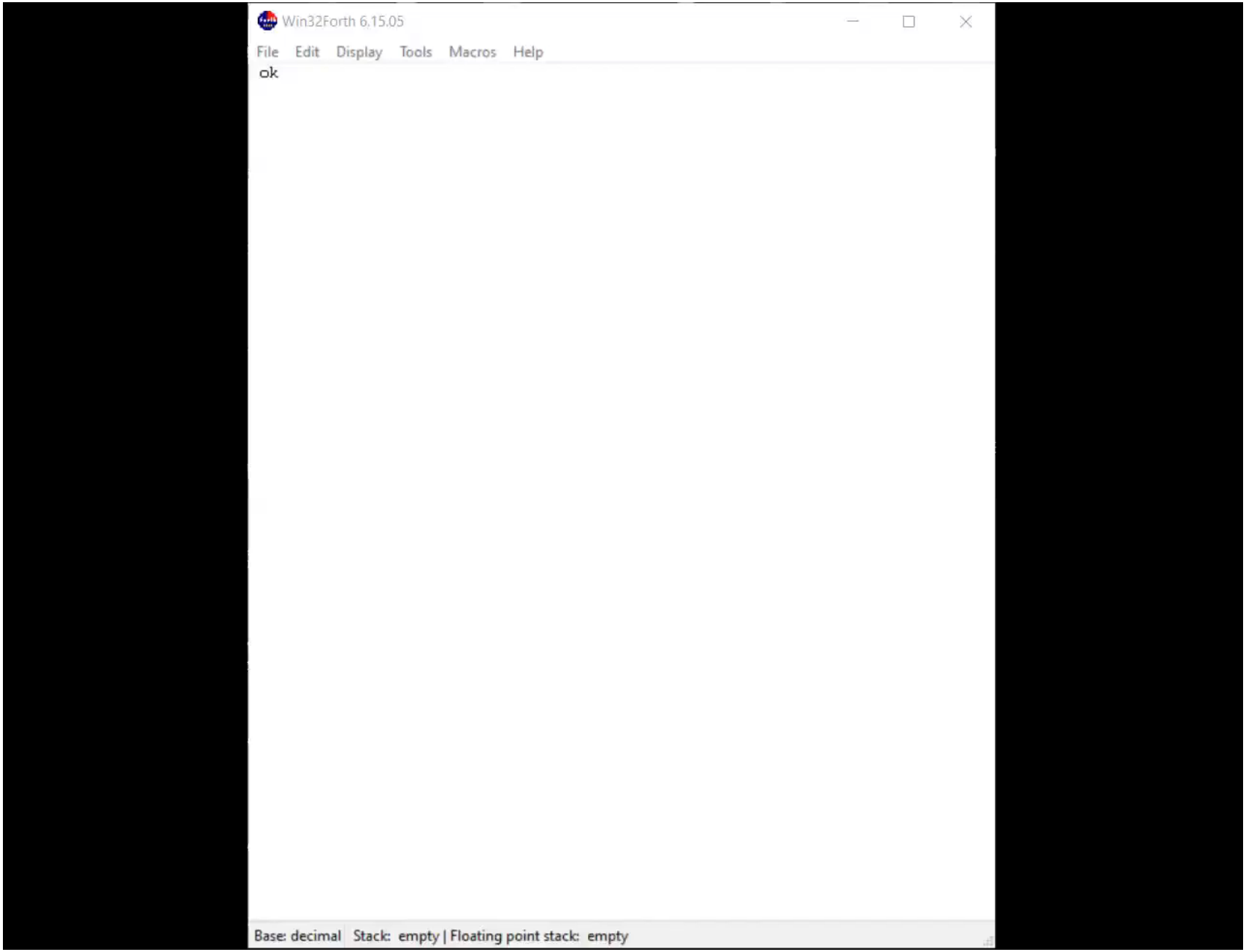
```
4 *IF   .( 4  Test +  -   /    )
    100 500 +  200 - 4 /   100  =
       [if]   .( Got the expected 100. )
       [else] .( Error in math operators ) 4 bells [then]
  [then]

And see:

4 Test +  -   /    Got the expected 100.

Or else

4 Test +  -   /  Error in math operators
   ding ding ding ding
```

More Involved Testing

```
7 *IF  .( 7  Test of {[ | ]} executing and compiling )

4 4 create{ A{
A{     {[ 1  2  3  4 |  5  6  7  8  |
          9 10 11 12 | 13 14 15 16  ]}
A{ }list

: }fill   {[ 1  2  3  4 |  5  6  7  8 |
             9 10 11 12 | 13 14 15 16 ]} ;

A{ }zeros  A{ }fill   A{ }list  forget A{

[then]
```

More Involved Testing III

```
8 Test of {[ | ]} executing and compiling

 1.0000  2.0000  3.0000  4.0000
 5.0000  6.0000  7.0000  8.0000
 9.0000 10.000  11.000  12.000
13.000  14.000  15.000  16.000

1.0000  2.0000  3.0000  4.0000
5.0000  6.0000  7.0000  8.0000
9.0000 10.000  11.000  12.000
13.000 14.000  15.000  16.000
```


Win32Forth 6.15.05
File Edit Display Tools Macros Help
ok
Base: decimal Stack: empty | Floating point stack: empty

Time Comparisons

Load with full testing:

FLOAD 'C:\Data\Forth\MatrixTwo\MatrixTwo-L.F'

Elapsed time: 00:00:05.466 (5.5 seconds)

Load with no testing:

FLOAD 'C:\Data\Forth\MatrixTwo\MatrixTwo-L.F'

Elapsed time: 00:00:00.026 (26 miliseconds)

Benefits

You are thinking most clearly about code just after you have written it.

Benefits

You are thinking most clearly about code just after you have written it.

Use that moment to capture test methods.

Benefits

You are thinking most clearly about code just after you have written it.

Use that moment to capture test methods.

And preserve those methods.

Benefits

You are thinking most clearly about code just after you have written it.

Use that moment to capture test methods.

And preserve those methods.

But only reuse them when needed.

Prior Art on Forth Testing

http://www.euroforth.org/ef19/papers/hoffmanna.pdf

Annex F, Test Suite of Forth200x at:
http://www.forth200x.org/documents/

https://github.com/Anding/simple-tester

Credits

- Andrew McKewan and Tom Zimmer for Win32Forth.
- The European team who updated it in the early 2000s.

References

- https://github.com/BillRagsdale/Forth_Projects/Program-And Test.pdf
- https://github.com/BillRagsdale/WIN32Forth-Guide