Forth ‘old-style’ code

John Rible <svfig@sandpipers.com>

Forth ‘old-style’ code

John Rible <svfig@sandpipers.com>

Paul Bennett wrote his “Sow’s Ear to a Silk Purse” paper to “highlight some problems” with code he’d seen. Below are the first four words. What’s the first error he saw?

Forth ‘old-style’ code

John Rible <svfig@sandpipers.com>

Paul Bennett wrote his “Sow’s Ear to a Silk Purse” paper to “highlight some problems” with code he’d seen. Below are the first four words. What’s the first error he saw?

```
: penny ( n -- ) dup 1 > if . ." pennies" else dup 0 > if . ." penny" then then ;

: nickel ( n -- n ) 5 /mod ?dup 0= if penny else . ." nickel" then
   ?dup 0 > if ." , " penny then ;

: dime ( n -- n ) 10 /mod ?dup 0= if nickel
   else dup 1 > if . ." dimes" else . ." dime" then
   then ?dup 0 > if ." , " nickel then ;

: quarter ( n -- n ) 25 /mod ?dup 0= if dime
   else dup 1 > if . ." quarters" else . ." quarter" then
   then ?dup 0 > if ." , " dime then ;
```

Paul included four comments on the problems he observed, the last of which was the most important:

4. *Considering where I found the earliest occurrence of an error, I think that he did not test as he coded. Considering that Forth allows you to do the testing as soon as you hit return after typing the semicolon, this is rather remiss of anyone. Create a word and then test it immediately. You then know your foundations are stable enough to support the rest of the structure you are building.*

He did not include other points that I've learned, the most basic of which is that each word ought to do only one thing.

So I sat down with a basic understanding of the program's goals and started to code, typing into SwiftForth's console and testing as I went. About 45 minutes later I had working code for the coins part that Paul had written for his paper.

Unlike the original code, I started with the largest coin since each stood alone. Each word took a value (in pennies), displayed its denomination if present, and left the remaining value on the stack. I'd noticed that the original author had included commas after all terms except the last one listed, so I pulled it out into a separate word. I included Paul's MCC (McCabe Cyclomatic Complexity) measure of complexity:

A word's MCC = 1 + 2 * (number of control structures in it).

```
\ display coin denominations needed to total the value given

: .comma? ( n' - n' (3) dup if ." , " then ;

: .half ( n - n' (3) 50 /mod if ." 1 half dollar" .comma? then ;

: .quarter ( n - n' (3) 25 /mod if ." 1 quarter" .comma? then ;

: .dimes ( n - n' (5) 10 /mod ?dup if dup . ." dime" 1- if ." s" then .comma? then ;

: .nickel ( n - n' (3) 5 /mod if ." 1 nickel" .comma? then ;

: .pennies ( n - (5) ?dup if dup . 1- if ." pennies" exit then ." penny" then ;

: .coins ( n - (3) dup 1 max 99 min over - if . ." is not 1-99 " exit then
   .half .quarter .dimes .nickel .pennies ;

\ The (#) is each word's MCC; avg MCC for this code is (5*3 +2*5)/(5+2) = 3.6


: .0-99 ( - ) \ displays all of the possible values
   100 0 do cr ." \ " i 2 .r ."  = " i .coins loop ;

.0-99
```

I’d decided to do the bills as well, but before going on, the punctuation upset my editor side: I wanted a period and an ‘and.’ Of course it had to be the ‘Oxford and,’ which keeps the comma when three or more items are in the list.

That required knowing how many terms had been listed BEFORE the current one, as well as knowing if any were to follow. After more head-scratching time than it took to code the previous slide, I decided to use a variable for that state info rather than another stack item. And made words for both plural cases. AND decided to use an immediate compiling word to reduce the MCC complexity. (This was done in a couple of passes, compressed in time here.)

```
\ Display coin and bill values, 1 of 4 - support code

: 0exit ( n - (1) \ at compile time emplace code to exit the run-time word when n=0
   postpone 0= postpone if postpone exit postpone then ; immediate

variable items \ count of bill & coin denominations already displayed

: .and? ( n' - (7) \ possibly insert 'and' and Oxford ',' BEFORE current term
   items @ 0= if drop exit then if ." , " exit then
   items @ 1- if ." ," then  ."  and " ;

: .done? ( n' - n' (3) dup 0= if ." . " then 1 items +! ;

: .ss? ( n - (3) 1- if ." s" then ;

: .ies? ( n - (3) 1- if ." ies" exit then ." y" ;
```

```
\ Display coin and bill values, 2 of 4 - display coins

: .half ( n - n' (1) 50 /mod 0exit ( n' ) dup .and? ." 1 half dollar" .done? ;

: .quarter ( n - n' (1) 25 /mod 0exit ( n' ) dup .and? ." 1 quarter" .done? ;

: .dimes ( n - n' (1) 10 /mod ?dup 0exit
   ( n' n ) over .and? dup . ." dime" .ss? .done? ;

: .nickel ( n - n' (1) 5 /mod 0exit ( n' ) dup .and?  ." 1 nickel" .done? ;

: .pennies ( n - n' (1) 1 /mod ?dup 0exit
   ( n' n ) over .and? dup . ." penn" .ies? .done? ;

: .coins ( n - n' (1) .half .quarter .dimes .nickel .pennies ;


: .value ( n - n′ ) cr ." \ " dup 2 .r ."  = " .coins ;

: .1-99 ( - ) 100 1 do 0 items ! i .value . items ? loop ;

.1-99
```

```
\ Display coin and bill values, 3 of 4 - display bills

: .hundreds ( n - n' (1) 10000 /mod ?dup 0exit
   ( n' n ) dup . ." hundred" .ss? .done? ;

: .fifty ( n - n' (1) 5000 /mod 0exit ( n' ) dup .and? ." 1 fifty" .done? ;

: .twenties ( n - n' (1) 2000 /mod ?dup 0exit
   ( n n’ ) over .and? dup . ." twent" .ies? .done? ;

: .ten ( n - n' (1) 1000 /mod 0exit ( n' ) dup .and? ." 1 ten" .done? ;

: .five ( n - n' (1) 500 /mod 0exit ( n' ) dup .and? ." 1 five" .done? ;

: .twos ( n - n' (1) 200 /mod ?dup 0exit ( n' n) over .and? dup . ." two" .ss? .done? ;

: .ones ( n - n' (1) 100 /mod ?dup 0exit ( n' n) over .and? dup . ." one" .ss? .done? ;

: .bills ( n - n' (1) .hundreds .fifty .twenties .ten .five .twos .ones ;


: .value ( n - n’ ) cr ." \ " dup 5 .r ."  = " .bills ;

\ The variable-sized increment allows more of the small values to be included.
: increment ( i - i’ ) \ 7 & 101 are arbitrary, 98800 includes all denominations
   dup 7 / 101 + 98800 rot - min ?dup 0= if 1200 then ;

: .1-99900 \ display a small selection of the 1000 possible values
   100000 100 do 0 items ! i .value space . items ? i increment +loop ;

.1-99900
```

```
\ Display coin and bill values, 4 of 4 - Top level

: .bills&coins ( n - n' (3) \ display from largest to smallest denomination
   dup 1 max 99999 min over - if dup . ." is not 1-99999 " exit then
   .bills .coins ;

\ avg MCC = (1*7 + 4*3 + 15*1)/(1+4+15) = 1.7


: .value ( n - n' ) cr ." \ " dup 5 .r ."  = " .bills&coins ;

: increment ( i - i' ) \ 5 & 7 are arbitrary, 98894 includes all denominations
   dup 5 / 7 + 98894 rot - min ?dup 0= if 1106 then ;

: .1-99999 \ display a small selection of the 100000 possible values
   100000 0 do 0 items ! i .value . items ? i increment +loop ;

.1-99999
```

Ok, I thought I was done. But Paul's table version looked like it would be pretty simple also, so a day later I gave it a shot. It did require some effort to debug - getting the table access done correctly was a pain. I 'rediscovered' that SwiftForth adds a null character after strings. It's up next if there's time.

```
\ Display bills and coins using a table, 1 of 2 - table and access words

create table \ 'sing'       'plural'
 10000 , ," hundred"      ," hundreds"     align
  5000 , ," fifty"        ," fifties"      align
  2000 , ," twenty"       ," twenties"     align
  1000 , ," ten"          ," tens"         align
   500 , ," five"         ," fives"        align
   200 , ," two"          ," twos"         align
   100 , ," one"          ," ones"         align
    50 , ," half dollar" ," half dollars" align
    25 , ," quarter"      ," quarters"     align
    10 , ," dime"         ," dimes"        align
     5 , ," nickel"       ," nickels"      align
     1 , ," penny"        ," pennies"      align
     0 , \ end-of-table marker

: -SF$ ( a - a' (1) count 1+ chars + ; \ move past a SwiftForth counted string

: -entry ( a n' n - n' a' (1) drop swap cell+ -SF$ -SF$ aligned ;

: .SF$ ( a - a' (1) dup count type -SF$ ;

: .plural ( a n' - n' a' (1) swap cell+ -SF$ .SF$ ;

: .singular ( a n' - n' a' (1) swap cell+ .SF$ -SF$ ;

: .entry ( a n' n - n' a' (3)  dup . 1- if .plural else .singular then aligned ;
```

```
\ Display bills and coins using a table, 2 of 2 - all the rest

variable items

: .and? ( n' - (7) \ insert 'and' with 'Oxford comma'
   items @ 0= if drop exit then  if ." , " exit then
   items @ 1- if ." ," then ."  and " ;

: .done? ( n' - (3) 0= if ." . " then 1 items +! ;

: .denomination ( a n - n' a' (3) \ display a table entry if it is present in n
   over @ /mod dup if over .and? .entry over .done? else -entry then ;

: .bills&coins ( n - n' (5) \ display from largest to smallest denomination
   dup 1 max 99999 min over - if dup . ." is not 1-99999 " exit then
   table begin tuck @ while .denomination repeat nip ;

\ Average MCC = (1*7 + 1*5 + 3*3 + 5*1)/(1+1+3+5) = 2.6


: .value ( n - n' )  cr ." \" dup 6 .r ."  = " .bills&coins ;

: increment ( i - i' ) \ 5 & 7 are arbitrary, 98894 includes all denominations
   dup 5 / 7 + 98894 rot - min ?dup 0= if 1106 then ;

: .1-99999 \ display a small selection of the 100000 possible values
   100000 0 do 0 items ! i .value . items ? i increment +loop ;

.1-99999
```

Links to the related files (on my google drive):

SVFIG 2020-07-25.pdf 100 KB
[https://drive.google.com/file/d/1WPx3G_5BmR92PasALrGkMKzYPane2lUP/]

dot-coins on fb.pdf 69KB
[https://drive.google.com/file/d/1uPnChrpkM4h8Jg5i0e5LGQH9n5vDeMxM/]

dot-coins2 on fb.pdf 64KB
[https://drive.google.com/file/d/1mu5biShPCNWq_6Fftb5DCLbd6YZs0KgS/]

DotBills&Coins on fb.pdf 76 KB
[https://drive.google.com/file/d/1xPIqrrCRVExtwhT1C_RJpaWIJUXc8Rph/]

SowsEarSilkPurse on fb.pdf 75 KB
[https://drive.google.com/file/d/1dtW5vTzLLHO-rN0PLZIwxKWNPCQQ2jYW/]